UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

UIT-T X.692

(03/2002)

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

SÉRIE X: RÉSEAUX DE DONNÉES ET COMMUNICATION ENTRE SYSTÈMES OUVERTS

Réseautage OSI et aspects systèmes – Notation de syntaxe abstraite numéro un (ASN.1)

Technologies de l'information – Règles de codage ASN.1: spécification de la notation de contrôle de codage (ECN)

Recommandation UIT-T X.692

RECOMMANDATIONS UIT-T DE LA SÉRIE X

RÉSEAUX DE DONNÉES ET COMMUNICATION ENTRE SYSTÈMES OUVERTS

RÉSEAUX PUBLICS DE DONNÉES	
Services et fonctionnalités	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalisation et commutation	X.50–X.89
Aspects réseau	X.90–X.149
Maintenance	X.150–X.179
Dispositions administratives	X.180–X.199
INTERCONNEXION DES SYSTÈMES OUVERTS	
Modèle et notation	X.200–X.209
Définitions des services	X.210–X.219
Spécifications des protocoles en mode connexion	X.220–X.229
Spécifications des protocoles en mode sans connexion	X.230–X.239
Formulaires PICS	X.240–X.259
Identification des protocoles	X.260–X.269
Protocoles de sécurité	X.270–X.279
Objets gérés des couches	X.280–X.289
Tests de conformité	X.290–X.299
INTERFONCTIONNEMENT DES RÉSEAUX	
Généralités	X.300–X.349
Systèmes de transmission de données par satellite	X.350–X.369
Réseaux à protocole Internet	X.370–X.399
SYSTÈMES DE MESSAGERIE	X.400–X.499
ANNUAIRE	X.500–X.599
RÉSEAUTAGE OSI ET ASPECTS SYSTÈMES	
Réseautage	X.600–X.629
Efficacité	X.630–X.639
Qualité de service	X.640–X.649
Dénomination, adressage et enregistrement	X.650–X.679
Notation de syntaxe abstraite numéro un (ASN.1)	**X.680–X.699**
GESTION OSI	
Cadre général et architecture de la gestion-systèmes	X.700–X.709
Service et protocole de communication de gestion	X.710–X.719
Structure de l'information de gestion	X.720–X.729
Fonctions de gestion et fonctions ODMA	X.730–X.799
SÉCURITÉ	X.800–X.849
APPLICATIONS OSI	
Engagement, concomitance et rétablissement	X.850–X.859
Traitement transactionnel	X.860–X.879
Opérations distantes	X.880–X.899
TRAITEMENT RÉPARTI OUVERT	X.900–X.999

Pour plus de détails, voir la Liste des Recommandations de l'UIT-T.

NORME INTERNATIONALE ISO/CEI 8825-3
RECOMMANDATION UIT-T X.692

Technologies de l'information – Règles de codage ASN.1: spécification de la notation de contrôle de codage (ECN)

Résumé

La présente Recommandation | Norme internationale définit la notation de contrôle de codage (ECN) utilisée afin de spécifier les codages (de types ASN.1) qui diffèrent de ceux qui sont fournis par les règles de codage normalisées telles que les règles de codage de base (BER) et les règles de codage compact (PER).

Source

La Recommandation X.692 de l'UIT-T, élaborée par la Commission d'études 17 (2001-2004) de l'UIT-T, a été approuvée le 8 mars 2002. Un texte identique est publié comme Norme Internationale ISO/CEI 8825-3.

AVANT-PROPOS

L'UIT (Union internationale des télécommunications) est une institution spécialisée des Nations Unies dans le domaine des télécommunications. L'UIT-T (Secteur de la normalisation des télécommunications) est un organe permanent de l'UIT. Il est chargé de l'étude des questions techniques, d'exploitation et de tarification, et émet à ce sujet des Recommandations en vue de la normalisation des télécommunications à l'échelle mondiale.

L'Assemblée mondiale de normalisation des télécommunications (AMNT), qui se réunit tous les quatre ans, détermine les thèmes d'étude à traiter par les Commissions d'études de l'UIT-T, lesquelles élaborent en retour des Recommandations sur ces thèmes.

L'approbation des Recommandations par les Membres de l'UIT-T s'effectue selon la procédure définie dans la Résolution 1 de l'AMNT.

Dans certains secteurs des technologies de l'information qui correspondent à la sphère de compétence de l'UIT-T, les normes nécessaires se préparent en collaboration avec l'ISO et la CEI.

NOTE

Dans la présente Recommandation, l'expression "Administration" est utilisée pour désigner de façon abrégée aussi bien une administration de télécommunications qu'une exploitation reconnue.

DROITS DE PROPRIÉTÉ INTELLECTUELLE

L'UIT attire l'attention sur la possibilité que l'application ou la mise en œuvre de la présente Recommandation puisse donner lieu à l'utilisation d'un droit de propriété intellectuelle. L'UIT ne prend pas position en ce qui concerne l'existence, la validité ou l'applicabilité des droits de propriété intellectuelle, qu'ils soient revendiqués par un Membre de l'UIT ou par une tierce partie étrangère à la procédure d'élaboration des Recommandations.

A la date d'approbation de la présente Recommandation, l'UIT n'avait pas été avisée de l'existence d'une propriété intellectuelle protégée par des brevets à acquérir pour mettre en œuvre la présente Recommandation. Toutefois, comme il ne s'agit peut-être pas de renseignements les plus récents, il est vivement recommandé aux responsables de la mise en œuvre de consulter la base de données des brevets du TSB.

TABLE DES MATIÈRES

Introduction

La notation de contrôle de codage (ECN) est une notation visant à spécifier les codages de types ASN.1 qui diffèrent de ceux qui sont fournis par les règles de codage normalisées. La notation ECN peut être utilisée pour coder tous les types d'une spécification ASN.1, mais peut également être utilisée avec les règles de codage normalisées telles que BER ou PER (Rec. UIT-T X.690 | ISO/CEI 8825-1 et Rec. UIT-T X.691 | ISO/CEI 8825-2) afin de spécifier seulement le codage de types qui ont des exigences spéciales.

Un type ASN.1 spécifie un ensemble de valeurs abstraites. Les règles de codage spécifient la représentation de ces valeurs abstraites sous la forme d'une série de bits. La notation ECN est conçue pour répondre aux besoins de codage suivants:

a) la nécessité d'écrire des types ASN.1 (et d'obtenir l'appui d'outils ASN.1 dans les implémentations) pour des protocoles établis ("existants") où le codage est déjà déterminé et diffère de toutes les règles de codage normalisées;

b) la nécessité de produire des codages qui soient des variantes mineures par rapport aux règles normalisées.

Le lien assuré dans une spécification ECN avec une spécification ASN.1 est bien défini et traitable en machine, de sorte que les codeurs et les décodeurs peuvent être produits automatiquement à partir des spécifications combinées. C'est un facteur important afin de réduire aussi bien la quantité de travail que la possibilité d'erreurs lors de la réalisation de systèmes interopérables. Un autre avantage notable est la capacité d'offrir la prise en charge d'outils automatiques d'essais.

Ces avantages sont disponibles avec la seule notation ASN.1 lorsque les règles de codage normalisées suffisent, mais les travaux en notation ECN offrent ces avantages lorsque les règles de codage normalisées ne sont pas suffisantes.

NOTE 1 – Actuellement la notation ECN prend en charge seulement les codages en mode binaire, mais elle pourrait être étendue ultérieurement afin de couvrir les codages en mode caractère.

L'Annexe A fait partie intégrante de la présente Recommandation | Norme internationale. Elle détaille les modifications à apporter à la Rec. UIT-T X.680 | ISO/CEI 8824-1 afin de prendre en charge la notation utilisée dans la présente Recommandation | Norme internationale.

L'Annexe B fait partie intégrante de la présente Recommandation | Norme internationale. Elle détaille les modifications à apporter à la Rec. UIT-T X.681 | ISO/CEI 8824-2 afin de prendre en charge la notation utilisée dans la présente Recommandation | Norme internationale.

L'Annexe C fait partie intégrante de la présente Recommandation | Norme internationale. Elle détaille les modifications à apporter à la Rec. UIT-T X.683 | ISO/CEI 8824-4 afin de prendre en charge la notation utilisée dans la présente Recommandation | Norme internationale.

NOTE 2 – Il n'est pas prévu que les Annexes A, B et C soient étendues sous forme d'amendements aux Recommandations | Normes internationales citées en référence. Les modifications ne sont destinées qu'à la définition de la notation ECN (voir article 5 et § 9.26).

L'Annexe D ne fait pas partie intégrante de la présente Recommandation | Norme internationale. Elle contient des exemples de l'utilisation de la notation ECN.

L'Annexe E ne fait pas partie intégrante de la présente Recommandation | Norme internationale. Elle offre de plus amples détails sur la prise en charge des codages de Huffman en notation ECN.

L'Annexe F ne fait pas partie intégrante de la présente Recommandation | Norme internationale. Elle identifie un site électronique donnant accès à d'autres informations et à d'autres liens relatifs à la notation ECN.

L'Annexe G ne fait pas partie intégrante de la présente Recommandation | Norme internationale. Elle offre un résumé de la notation ECN au moyen de la notation de l'article 5.

NORME INTERNATIONALE
RECOMMANDATION UIT-T

Technologies de l'information – Règles de codage ASN.1: spécification de la notation de contrôle de codage (ECN)

1 Domaine d'application

La présente Recommandation | Norme internationale définit une notation visant à spécifier les codages de types ASN.1 ou de parties de ces types.

Elle offre plusieurs mécanismes pour une telle spécification, dont

- spécification directe du codage au moyen d'une notation normalisée;
- spécification du codage par référence aux règles de codage normalisées;
- spécification du codage d'un type ASN.1 par référence à une structure de codage;
- spécification du codage au moyen d'une notation non ECN.

Elle offre également le moyen de relier la spécification de codages aux définitions des types auxquels elles doivent être appliquées.

2 Références normatives

Les Recommandations et Normes internationales suivantes contiennent des dispositions qui, par suite de la référence qui y est faite, constituent des dispositions valables pour la présente Recommandation | Norme internationale. Au moment de la publication, les éditions indiquées étaient en vigueur. Toutes Recommandations et Normes internationales sont sujettes à révision et les parties prenantes aux accords fondés sur la présente Recommandation | Norme internationale sont invitées à rechercher la possibilité d'appliquer les éditions les plus récentes des Recommandations et Normes indiquées ci-après. Les membres de la CEI et de l'ISO possèdent le registre des Normes internationales en vigueur. Le Bureau de la normalisation des télécommunications de l'UIT tient à jour une liste des Recommandations UIT-T en vigueur.

2.1 Recommandations | Normes internationales identiques

- Recommandation UIT-T X.660 (1992) | ISO/CEI 9834-1:1993*, Technologies de l'information – Interconnexion des systèmes ouverts – Procédures pour le fonctionnement des autorités d'enregistrement OSI: procédures générales (plus amendements)*.
- Recommandation UIT-T X.680 (2002) | ISO/CEI 8824-1:2002, *Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification de la notation de base.*
- Recommandation UIT-T X.681 (2002) | ISO/CEI 8824-2:2002, *Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification des objets informationnels*.
- Recommandation UIT-T X.682 (2002) | ISO/CEI 8824-3:2002, *Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification des contraintes.*
- Recommandation UIT-T X.683 (2002) | ISO/CEI 8824-4:2002, *Technologies de l'information – Notation de syntaxe abstraite numéro un: paramétrage des spécifications de la notation de syntaxe abstraite numéro un*.
- Recommandation UIT-T X.690 (2002) | ISO/CEI 8825-1:2002, *Technologies de l'information – Règles de codage ASN.1: spécification des règles de codage de base, des règles de codage canoniques et des règles de codage distinctives*.
- Recommandation UIT-T X.691 (2002) | ISO/CEI 8825-2:2002, *Technologies de l'information – Règles de codage ASN.1: spécification des règles de codage compact*.

NOTE 1 – Par dérogation à la date de publication ISO, les spécifications ci-dessus sont normalement désignées comme "ASN.1:2002".

NOTE 2 – Les références ci-dessus doivent être interprétées comme des références aux Recommandations | Normes internationales indiquées, ainsi qu'à tous leurs amendements et corrigenda techniques publiés.

2.2 Autres références

– ISO/CEI 10646-1:1993, *Technologies de l'information – Jeu universel de caractères codés à plusieurs octets – Partie 1: Architecture et table multilingue*.

NOTE – La référence ci-dessus doit être interprétée comme une référence à l'ISO/CEI 10646-1 avec tous ses amendements et corrigenda techniques publiés.

3 Définitions

Pour les besoins de la présente Recommandation | Norme internationale, les définitions suivantes s'appliquent.

3.1 Définitions ASN.1

La présente Recommandation | Norme internationale utilise les termes définis dans l'article 3 des Recommandations UIT-T X.680 | ISO/CEI 8824-1, UIT-T X.681 | ISO/CEI 8824-2, UIT-T X.682 | ISO/CEI 8824-3, UIT-T X.683 | ISO/CEI 8824-4, UIT-T X.690 | ISO/CEI 8825-1 et UIT-T X.691 | ISO/CEI 8825-2.

3.2 Définitions spécifiquement ECN

3.2.1 point d'alignement: point dans un codage (habituellement son début) qui sert de point de référence lorsqu'une spécification de codage nécessite un alignement sur une frontière quelconque.

3.2.2 champ auxiliaire: champ d'une structure de remplacement (qui est ajouté dans la spécification ECN) dont la valeur est activée directement par le codeur sans utilisation d'une quelconque valeur abstraite fournie par l'application.

NOTE – Un exemple de champ auxiliaire est un déterminant de longueur pour un codage d'entier ou pour une répétition.

3.2.3 champ binaire: bits ou octets contigus dans un codage qui sont décodés comme un tout, et qui soit représentent une valeur abstraite, ou fournissent des informations (telles qu'un déterminant de longueur pour un autre champ – voir 3.2.30) nécessaire pour un décodage correct, ou les deux.

NOTE – C'est dans les protocoles existants que l'option "ou les deux" apparaît parfois.

3.2.4 classe de champ binaire: classe de codage dont les objets spécifient le codage de valeurs abstraites (d'un type ASN.1 quelconque) pour les transformer en bits.

NOTE – D'autres classes de codage sont concernées par des procédures de codage plus générales, telles que celles qui sont requises afin de déterminer la fin de répétitions de codages de classe de champ binaire, ou afin de déterminer quel codage, d'un ensemble de codages de champ binaire en variante, est présent.

3.2.5 condition aux limites: condition relative à l'existence de limites d'un champ d'entier (qu'elles permettent des valeurs négatives ou non) qui, si elle est satisfaite, implique que des règles de codage spécifiées doivent être appliquées.

3.2.6 déterminant de choix: champ binaire qui détermine lequel de plusieurs codages possibles (chacun représentant différentes valeurs abstraites) est présent dans un autre champ binaire.

3.2.7 ensemble d'objets de codage combinés: ensemble temporaire d'objets de codage produit par la combinaison de deux ensembles d'objets de codage afin d'appliquer des codages.

3.2.8 codage conditionnel: codage qui doit être appliqué seulement si une condition spécifiée aux limites ou une condition de catégorie de longueur est satisfaite.

3.2.9 type conteneur: type ASN.1 (ou champ de structure de codage) dans lequel une contrainte de contenu a été appliquée aux valeurs de ce type (ou aux valeurs associées à ce champ de structure de codage).

NOTE – Les types ASN.1 auxquels une contrainte de contenu (utilisant **`CONTAINING/ENCODED BY`**) peut être appliquée sont les types "chaîne de bits" et "chaîne d'octets".

3.2.10 point d'application courant: point dans une structure de codage auquel un ensemble d'objets de codage combinés est en cours d'application.

3.2.11 codage-décodage différentiel: spécification de règles pour un décodeur qui nécessitent l'acceptation de codages qui ne peuvent pas être produits par un codeur conforme à la spécification courante.

NOTE – Le codage-décodage différentiel prend en charge la spécification du décodage par un décodeur (conforme à une version initiale d'une norme) prévu avec la capacité de décoder correctement les codages produits par une version ultérieure de cette norme. C'est ce qui est parfois appelé la *prise en charge d'extensibilité*.

3.2.12 classe de codage: ensemble de tous les codages possibles dans une partie spécifique des procédures nécessaires afin d'effectuer le codage ou le décodage d'un type ASN.1.

NOTE – Les classes de codage sont définies pour le codage de types primitifs ASN.1, mais sont également définies pour les procédures associées à la notation par étiquettes ASN.1, pour l'utilisation du mot **`OPTIONAL`** et pour les constructeurs de codage.

3.2.13 catégorie de classe de codage: classes de codage avec certaines caractéristiques communes.

NOTE – Exemples: la catégorie des entiers, la catégorie des booléens, et la catégorie des concaténations.

3.2.14 constructeur de codage: classe de codage dont les objets de codage définissent des procédures permettant de combiner, de sélectionner, ou de répéter des parties d'un codage. (Exemples: les classes **`#ALTERNATIVES`**, **`#CHOICE`**, **`#CONCATENATION`**, **`#SEQUENCE`**, etc.).

3.2.15 modules de définition de codage (EDM, *encoding definition module*): modules qui définissent des codages pour application dans le module de lien de codage.

3.2.16 module de lien de codage (ELM, *encoding link module*): module (unique, pour toute application donnée) qui attribue des codages à des types ASN.1.

3.2.17 objet de codage: spécification de certaines parties des procédures nécessaires pour effectuer le codage ou le décodage d'un type ASN.1.

NOTE – Les objets de codage peuvent spécifier le codage de types primitifs ASN.1, mais peuvent également spécifier les procédures associées à la notation par étiquettes ASN.1, à l'utilisation du mot **`OPTIONAL`** et aux constructeurs de codage.

3.2.18 ensemble d'objets de codage: ensemble d'objets de codage.

NOTE – Un ensemble d'objets de codage est normalement utilisé dans le module de lien de codage afin de déterminer le codage de tous les types de niveau supérieur utilisés dans une application.

3.2.19 propriété de codage: élément informationnel utilisé afin de définir un codage au moyen de la notation spécifiée dans les articles 23, 24 et 25.

3.2.20 espace de codage: nombre de bits (ou d'octets, de mots ou d'autres éléments) utilisés pour coder une valeur abstraite dans un champ binaire (voir § 9.21.5).

3.2.21 structure de codage: structure d'un codage, définie soit à partir de la structure d'une définition de type ASN.1, ou dans un module EDM utilisant des classes de champ binaire et des constructeurs de codage.

NOTE 1 – L'utilisation d'une structure de codage n'est qu'un mécanisme parmi plusieurs (mais un mécanisme important) que la notation de contrôle de codage offre pour la définition de codages pour types ASN.1.

NOTE 2 – La définition d'une structure de codage est également la définition d'une classe de codage correspondante.

3.2.22 structure de codage produite explicitement: structure de codage construite sur la base d'une structure de codage produite implicitement par l'utilisation de la clause de renommage dans un module EDM.

3.2.23 extensibilité: dispositions d'une version antérieure d'une norme qui sont conçues pour maximiser l'interfonctionnement d'implémentations de cette version antérieure avec les implémentations prévues d'une version ultérieure de cette norme.

3.2.24 nom entièrement qualifié: référence à une classe de codage, à un objet, ou à un ensemble d'objets qui contient soit le nom du module EDM dans lequel cette classe de codage, cet objet, ou cet ensemble d'objets a été défini, ou (dans le cas d'une classe de codage produite implicitement) le nom du module ASN.1 dans lequel elle a été produite (voir également 3.2.42).

NOTE – Un nom entièrement qualifié (voir production "ExternalEncodingClassReference au § 10.6) doit être utilisé dans le corps d'un module si la classe de codage est une structure de codage produite implicitement, dont le nom est un nom de classe réservé, ou si l'utilisation du nom seul produirait toute ambiguïté en raison de multiples importations de classes avec ce nom (voir A.1/12.15).

3.2.25 structure de codage produite: structure de codage produite explicitement ou implicitement dont la finalité est de définir les codages du type ASN.1 correspondant au moyen de l'application des codages dans le module ELM.

3.2.26 gouverneur: partie d'une spécification ECN qui détermine la forme syntaxique (et la sémantique) d'une autre partie de la spécification ECN.

NOTE – Un gouverneur est une référence de classe de codage qui détermine la syntaxe à utiliser pour la définition d'un objet de codage (de cette classe). Le concept est le même que celui d'une référence de type en notation ASN.1 jouant le rôle de gouverneur pour la notation de valeur ASN.1.

3.2.27 pointeur d'identification: partie de codage qui sert à distinguer les codages d'une classe de codage de ceux d'autres classes de codage.

NOTE – Les règles ASN.1 de codage de base utilisent des étiquettes afin d'offrir des pointeurs d'identification dans les codages BER.

3.2.28 structure de codage produite implicitement: structure de codage qui est produite implicitement et exportée chaque fois qu'un type est défini dans un module ASN.1.

3.2.29 point d'application initiale: point d'une structure de codage auquel tout ensemble donné d'objets de codage combinés est d'abord appliqué (dans le module ELM et dans les modules EDM).

3.2.30 déterminant de longueur: champ binaire qui détermine la longueur d'un autre champ binaire.

3.2.31 valeur négative d'entier: valeur inférieure à zéro.

3.2.32 valeur non négative d'entier: valeur supérieure ou égale à zéro.

3.2.33 valeur non positive d'entier: valeur inférieure ou égale à zéro.

3.2.34 champ binaire facultatif: champ binaire qui est parfois inclus (pour coder une valeur abstraite) et est parfois omis.

3.2.35 valeur positive d'entier: valeur supérieure à zéro.

3.2.36 déterminant de présence: champ binaire qui détermine si un champ binaire facultatif est présent ou non.

3.2.37 classe primitive: classe de codage qui n'est pas une structure de codage, et qui ne peut pas être déréférencée vers une autre classe (voir §16.1.14).

3.2.38 définition récursive (d'un nom de référence): nom de référence pour lequel la résolution du nom de référence, ou du gouverneur d'une définition du nom de référence, nécessite la résolution du nom de référence original.

NOTE – La définition récursive d'une classe de codage (dont une structure de codage) est permise. La définition récursive d'un objet de codage ou d'un ensemble d'objets de codage est interdite par les § 17.1.4 et 18.1.3 respectivement.

3.2.39 instanciation récursive (d'un nom de référence paramétré): instanciation d'un nom de référence, dans laquelle la résolution des paramètres réels nécessite la résolution du nom de référence original.

NOTE – L'instanciation récursive d'une classe de codage (dont une structure de codage) est permise. L'instanciation récursive d'un objet de codage ou d'un ensemble d'objets de codage est interdite par § 17.1.4 et 18.1.3 respectivement.

3.2.40 structure de remplacement: structure paramétrée utilisée afin de remplacer certaines ou toutes parties d'une construction avant codage de la construction.

3.2.41 codage autodélimiteur: codage d'un ensemble de valeurs abstraites tel qu'il n'y a pas de valeur abstraite dont le codage est une sous-chaîne initiale du codage d'une quelconque autre valeur abstraite dans l'ensemble.

NOTE – Cela comprend non seulement les codages de longueur fixe d'un entier borné, mais également les codages généralement décrits comme "codages de Huffman" (voir Annexe E).

3.2.42 nom de référence simple: référence à une classe de codage, à un objet, ou à un ensemble d'objets qui ne contient ni le nom du module EDM dans lequel cette classe de codage, cet objet, ou cet ensemble d'objets a été défini, ni (dans le cas d'une classe de codage produite implicitement) le nom du module ASN.1 dans lequel cette classe a été produite.

NOTE – Un nom de référence simple ne peut être utilisé que lorsque la référence à la classe de codage est univoque; sinon, un nom entièrement qualifié (voir 3.2.24) doit être utilisé dans le corps d'un module.

3.2.43 condition de catégorie de longueur: condition relative à l'existence de contraintes effectives de longueur applicables à un champ de chaîne ou de répétition (que cette contrainte contienne zéro longueur et/ou autorise des longueurs multiples) qui, si elle est satisfaite, implique que des règles de codage spécifiées doivent être appliquées.

3.2.44 gouverneur source (ou classe source): gouverneur qui détermine la notation visant à spécifier les valeurs abstraites associées à une classe source lors de leur mappage sur une classe cible.

3.2.45 pointeur de début: champ auxiliaire indiquant la présence ou l'absence d'un champ binaire facultatif, et, dans le cas d'une présence, contenant le décalage entre la position actuelle et le champ binaire.

3.2.46 gouverneur cible (ou classe cible): gouverneur qui détermine la notation visant à spécifier des valeurs abstraites associées à une classe cible lors d'un mappage sur ces valeurs à partir d'une classe source.

3.2.47 type(s) de niveau supérieur: types ASN.1 d'une application qui sont utilisés par celle-ci à des fins autres en tant que définir les composants d'autres types ASN.1.

NOTE 1 – Des types de niveau supérieur peuvent également être utilisés (mais habituellement ne le sont pas) comme composants d'autres types ASN.1.

NOTE 2 – Des types de niveau supérieur sont parfois cités comme étant des "messages d'application", ou des unités "PDU". De tels types sont normalement traités spécialement par des utilitaires, car ils forment le niveau supérieur de structures de données d'un langage de programmation présentées à l'application.

3.2.48 transformées: objets de codage de la classe **`#TRANSFORM`** qui spécifient que le codage des valeurs abstraites associées à certaines classes (ou de composites de transformation – voir 3.2.49) doit être le codage de différentes valeurs abstraites associées à la même classe ou à une classe différente (ou de composites de transformation).

NOTE – Les transformées peuvent être utilisées, par exemple, afin de spécifier des opérations arithmétiques simples sur des valeurs entières, ou pour mapper des valeurs entières dans des chaînes de caractères ou des chaînes de bits.

3.2.49 composites de transformation: liste ordonnée d'éléments qui peut elle-même être la source ou le résultat des transformées.

NOTE – Tous les éléments d'un composite doivent avoir la même classification (voir § 9.18.2).

3.2.50 codage de valeur: façon dont un espace de codage est utilisé afin de représenter une valeur abstraite (voir § 9.21.5).

4 Abréviations

Pour les besoins de la présente Recommandation | Norme internationale les abréviations suivantes s'appliquent.

ASN.1	Notation de syntaxe abstraite numéro un (*abstract syntax notation one*)
BCD	Décimal codé binaire (*binary coded decimal*)
BER	Règles de codage de base de la notation ASN.1 (*basic encoding rules of ASN.1*)
CER	Règles de codage canoniques de la notation ASN.1 (*canonical encoding rules of ASN.1*)
DER	Règles de codage distinctives de la notation ASN.1 (*distinguished encoding rules of ASN.1*)
ECN	Notation de contrôle de codage pour ASN.1 (*encoding control notation for ASN.1*)
EDM	Module de définition de codage (*encoding definition module*)
ELM	Module de lien de codage (*encoding link module*)
PDU	Unité de données protocolaire (*protocol data unit*)
PER	Règles de codage compact de la notation ASN.1 (*packed encoding rules of ASN.1*)

5 Définition de la syntaxe ECN

5.1 La présente Recommandation | Norme internationale emploie la convention de notation définie dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, article 5.

5.2 La présente Recommandation | Norme internationale emploie la notation pour classes d'objets informationnels définie dans la Rec. UIT-T X.681 | ISO/CEI 8824-2 telle que modifiée par l'Annexe B.

5.3 La présente Recommandation | Norme internationale fait référence à des productions définies dans la Rec. UIT-T X.680 | ISO/CEI 8824-1 telle que modifiée par l'Annexe A, dans la Rec. UIT-T X.681 | ISO/CEI 8824-2 telle que modifiée par l'Annexe B et dans la Rec. UIT-T X.683 | ISO/CEI 8824-4 telle que modifiée par l'Annexe C.

6 Conventions et notation de codage

6.1 La présente Recommandation | Norme internationale définit la valeur de chaque octet dans un codage par l'utilisation des termes "bit de plus fort poids" et "bit de plus faible poids".

NOTE – Les spécifications de couche inférieure utilisent la même notation afin de définir l'ordre de transmission des bits sur une ligne en série, ou l'attribution de bits à des voies parallèles.

6.2 Aux fins de la présente Recommandation | Norme internationale, les bits d'un octet sont numérotés de 8 à 1, où le bit 8 est le "bit de plus fort poids" et où le bit 1 est le "bit de plus faible poids".

6.3 Aux fins de la présente Recommandation | Norme internationale, les codages sont définis comme une chaîne de bits partant d'un "bit initial" jusqu'à un "bit de fin". Lors de la transmission, les huit premiers bits de cette chaîne de bits à partir du "bit initial" doivent être placés dans le premier octet transmis avec le bit initial comme le bit de plus fort poids de cet octet. Les huit bits suivants doivent être placés dans l'octet suivant, et ainsi de suite. Si le codage n'est pas un multiple de huit bits, dans ce cas les bits restants doivent être transmis comme s'ils étaient des bits 8 à la fin d'un octet suivant.

NOTE – Un codage ECN complet ne doit pas toujours être un multiple de huit bits, mais une spécification ECN peut déterminer l'insertion d'un bourrage afin de garantir cette propriété.

6.4 Lorsque des figures sont représentées dans la présente Recommandation | Norme internationale, le "bit initial" est toujours indiqué à gauche de la figure.

7 Le jeu de caractères ECN

7.1 L'utilisation du terme "caractère" dans toute la présente Recommandation | Norme internationale renvoie aux caractères spécifiés dans l'ISO/CEI 10646-1. Une prise en charge complète de toutes les spécifications ECN possibles peut nécessiter la représentation de tous ces caractères.

7.2 A l'exception des commentaires (tels que définis dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 11.6), d'une définition non ECN d'objets de codage (voir § 17.8) et des valeurs de chaîne de caractères, les spécifications ECN utilisent seulement les caractères énumérés dans le Tableau 1.

7.3 Les items lexicaux définis dans l'article 8 se composent d'une séquence des caractères énumérés dans le Tableau 1.

NOTE – Des restrictions supplémentaires sur les caractères permis pour chaque item lexical sont spécifiées dans l'article 8.

Tableau 1 – Caractères de notation ECN

0 à 9	(CHIFFRE ZÉRO À CHIFFRE 9)
A à Z	(LETTRE LATINE MAJUSCULE A à LETTRE LATINE MAJUSCULE Z)
a à z	(LETTRE LATINE MINUSCULE A à LETTRE LATINE MINUSCULE Z)
"	(GUILLEMETS)
#	(DIÈSE)
&	(PERLUÈTE)
'	(APOSTROPHE)
(	(PARENTHÈSE GAUCHE)
)	(PARENTHÈSE DROITE)
,	(VIRGULE)
-	(TIRET-MOINS)
.	(POINT)
:	(DEUX-POINTS)
;	(POINT-VIRGULE)
<	(SIGNE PLUS PETIT QUE)
=	(SIGNE ÉGAL)
>	(SIGNE PLUS GRAND QUE)
{	(ACCOLADE GAUCHE)
|	(TRAIT VERTICAL)
}	(ACCOLADE DROITE)

7.4 Aucune importance particulière ne doit être donnée au style typographique, à la dimension, à la couleur, à l'intensité, ni à d'autres caractéristiques d'affichage.

7.5 Les lettres majuscules et minuscules doivent être considérées comme distinctes.

8 Items lexicaux ECN

En plus des items lexicaux ASN.1 spécifiés dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, article 11, la présente Recommandation | Norme internationale utilise les items lexicaux spécifiés dans les paragraphes suivants. Les règles générales spécifiées dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 11.1 s'appliquent dans cet article.

NOTE – L'Annexe G énumère tous les items lexicaux et toutes les productions utilisés dans la présente Recommandation | Norme internationale, en précisant ceux qui sont définis dans les Recommandations UIT-T X.680 | ISO/CEI 8824-1, UIT-T X.681 | ISO/CEI 8824-2 et UIT-T X.683 | ISO/CEI 8824-4.

8.1 Références d'objet de codage

Nom de l'item – encodingobjectreference

Un item lexical "encodingobjectreference" doit se composer de la séquence de caractères spécifiée pour un item "valuereference" dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 11.4. Lors de l'analyse d'une instance de l'utilisation de cette notation, un item "encodingobjectreference" est distingué d'un "identificateur" par le contexte dans lequel il apparaît.

8.2 Références d'ensemble d'objets de codage

Nom de l'item – encodingobjectsetreference

Un item lexical "encodingobjectsetreference" doit se composer de la séquence de caractères spécifiée pour un item "typereference" dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 11.2. Il ne doit pas être une des séquences de caractères énumérées dans le § 8.4.

8.3 Références de classe de codage

Nom de l'item – encodingclassreference

Une item lexical "encodingclassreference" doit se composer du caractère "#" suivi par la séquence de caractères spécifiée pour un item "typereference" dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 11.2. Il ne doit pas être une des séquences de caractères énumérées dans le § 8.5 sauf dans un liste d'importations de module EDM (voir Rec. UIT-T X.680 | ISO/CEI 8824-1, § 12.19, telle que modifiée par le § A.1) ou dans un item "ExternalEncodingClassReference (voir la Note du § 14.11).

8.4 Items de mots réservés

Noms d'items de mots réservés:

ALL	**FIELDS**	**PER-BASIC-UNALIGNED**
AS	**FROM**	**PER-CANONICAL-ALIGNED**
BEGIN	**GENERATES**	**PER-CANONICAL-UNALIGNED**
BER	**IF**	**PLUS-INFINITY**
BITS	**IMPORTS**	**REFERENCE**
BY	**IN**	**REMAINDER**
CER	**LINK-DEFINITIONS**	**RENAMES**
COMPLETED	**MAPPING**	**SIZE**
DECODE	**MAX**	**STRUCTURE**
DER	**MIN**	**STRUCTURED**
DISTRIBUTION	**MINUS-INFINITY**	**TO**
ENCODE	**NON-ECN-BEGIN**	**TRANSFORMS**
ENCODING-CLASS	**NON-ECN-END**	**TRUE**
ENCODE-DECODE	**NULL**	**UNION**
ENCODING-DEFINITIONS	**OPTIONAL-ENCODING**	**USE**
END	**OPTIONS**	**USE-SET**
EXCEPT	**ORDERED**	**VALUES**
EXPORTS	**OUTER**	**WITH**
FALSE	**PER-BASIC-ALIGNED**	

Les items désignés par les noms ci-dessus doivent se composer de la séquence de caractères contenue dans ces noms.

NOTE – Les mots (voir Rec. UIT-T X.681 | ISO/CEI 8824-2, § 7.9) utilisés dans la définition de classes de codage (à l'intérieur d'une instruction "WITH SYNTAX") dans l'article 23 ne sont pas des mots réservés (voir également § B.14).

8.5 Items de noms de classe de codage réservés

Noms d'items de noms de classe de codage réservés:

#ALTERNATIVES	**#GeneralString**	**#REAL**
#BITS	**#GraphicString**	**#RELATIVE-OID**
#BIT-STRING	**#IA5String**	**#REPETITION**
#BMPString	**#INT**	**#SEQUENCE**
#BOOL	**#INTEGER**	**#SEQUENCE-OF**
#BOOLEAN	**#NUL**	**#SET**
#CHARACTER-STRING	**#NULL**	**#SET-OF**
#CHARS	**#NumericString**	**#TAG**
#CHOICE	**#OBJECT-IDENTIFIER**	**#TeletexString**
#CONCATENATION	**#ObjectDescriptor**	**#TRANSFORM**
#CONDITIONAL-INT	**#OCTETS**	**#UniversalString**
#CONDITIONAL-REPETITION	**#OCTET-STRING**	**#UTCTime**
#EMBEDDED-PDV	**#OPEN-TYPE**	**#UTF8String**
#ENCODINGS	**#OPTIONAL**	**#VideotexString**
#ENUMERATED	**#OUTER**	**#VisibleString**
#EXTERNAL	**#PAD**	
#GeneralizedTime	**#PrintableString**	

Les items désignés par les noms ci-dessus doivent se composer de la séquence de caractères contenue dans ces noms.

8.6 Item non ECN

Nom de l'item – anystringexceptnonecnend

Un item "anystringexceptnonecnend" doit se composer d'un ou de plusieurs caractères extraits du jeu de caractères ISO/CEI 10646-1, sauf qu'il ne doit pas être la séquence de caractères **NON-ECN-END** et que cette séquence de caractères ne doit pas apparaître dans cet item.

9 Concepts ECN

Cet article décrit les principaux concepts sous-tendant cette Recommandation UIT-T | Norme internationale.

9.1 Spécifications de notation de contrôle de codage (ECN)

9.1.1 Les spécifications ECN se composent d'un ou de plusieurs modules de définition de codage (EDM, *encoding definition module*) qui définissent des règles de codage pour types ASN.1 et d'un seul module de lien de codage (ELM, *encoding link module*) qui applique ces règles de codage à des types ASN.1.

9.1.2 La plus importante partie de la notation ECN est le concept de **définition de la structure de codage**. La notation ASN.1 est utilisée afin de définir des valeurs abstraites complexes, utilisant des types et des constructeurs primitifs. De la même façon, des codages complexes peuvent être définis au moyen d'une notation similaire où des mécanismes de construction sont utilisés afin de combiner des champs binaires simples en codages plus complexes, et finalement en messages complets. C'est ce qui est appelé la définition de la structure de codage. En utilisant la notation ECN avec la notation ASN.1, il est nécessaire en principe de:

a) définir la syntaxe abstraite (l'ensemble de valeurs abstraites à communiquer, et leur sémantique);

b) définir la structure de codage (la structure des champs) utilisée pour transporter ces valeurs abstraites;

c) associer les composants de la valeur abstraite aux champs de structure de codage;

d) de définir le codage de chaque champ de structure de codage et le codage des mécanismes permettant d'identifier les répétitions de champ, les options, etc.

9.1.3 Le processus ci-dessus normalement se déroule en plusieurs étapes. Tout d'abord, une définition ASN.1 est produite afin de détailler la syntaxe abstraite. A partir de là, une structure de codage brute est produite automatiquement (théoriquement dans le module ASN.1). Cette structure produite implicitement contient seulement les champs qui acheminent la sémantique applicative, sans champs pour des éléments tels que la détermination de longueur, le choix d'options, et ainsi de suite.

9.1.4 Cette structure peut être transformée par une série de mécanismes contenue dans la structure de champ qui est réellement requise, dont tous les champs nécessaires afin de prendre en charge l'activité de décodage (déterminants). Ces mécanismes impliquent tous une certaine forme de remplacement d'un simple champ acheminant la sémantique applicative par une structure plus complexe. De tels remplacements constituent une partie importante de la spécification de notation ECN.

9.1.5 L'on peut également définir des **objets de codage** pour chacun des champs contenus dans la structure finale. Ces champs déterminent non seulement le codage des champs, mais également la façon dont un seul champ détermine la longueur (par exemple) d'un autre, ou effectue la résolution de son offres d'options.

9.1.6 Les définitions ci-dessus apparaissent dans les modules de définition de codage (EDM). La dernière étape consiste à appliquer un ensemble d'objets de codage définis à la structure finale de codage afin de déterminer complètement un codage. Cette opération est effectuée dans le module de lien de codage (ELM).

9.2 Classes de codage

9.2.1 Une classe de codage est une propriété implicite de tous les types ASN.1 qui représente l'ensemble de toutes les spécifications de codage possibles pour ce type. Elle offre une référence qui permet aux modules de définition de codage de définir des règles de codage pour les champs de structure de codage correspondant au type. Les noms de classe de codage commencent par le caractère "**#**".

Exemple: les règles de codage pour le type ASN.1 intégré **INTEGER** sont définies par référence à la classe de codage **#INTEGER**. Les règles de codage pour un type défini par l'usager "**My-Type**" sont définies par référence à la classe de codage **#My-Type**.

9.2.2 Il y a plusieurs sortes de classes de codage:

9.2.2.1 Classes de codage intégrées – Ce sont des classes de codage intégrées avec des noms tels que **#INTEGER** et **#BOOLEAN**. Ces classes permettent la définition de codages spéciaux pour types primitifs ASN.1. Il y a également des classes de codage intégrées pour les constructeurs de codage telles que **#SEQUENCE**, **#SEQUENCE-OF** et **#CHOICE** (voir également § 9.3.2) et pour la définition de règles de codage destinées à gérer les offres d'options au moyen du mot **#OPTIONAL**. Le codage d'étiquettes est pris en charge par la classe **#TAG**. Finalement, il y a certaines classes intégrées (**#OUTER**, **#TRANSFORM** et autres) qui permettent la définition de procédures de codage faisant partie du processus de codage/décodage, mais qui n'ont pas de lien direct avec un quelconque champ binaire réel ou une quelconque création ASN.1 réelle.

9.2.2.2 Classes de codage pour les structures de codage produites implicitement – Ces classes ont des noms composés du caractère "**#**" suivi par l'item "typereference" apparaissant dans un élément "TypeAssignment" d'un module ASN.1. De telles classes de codage sont produites implicitement chaque fois qu'un item (non paramétré) "typereference" est attribué dans un module ASN.1. Elles peuvent être importées dans un module de définition de codage afin de permettre la définition de codages spéciaux pour le type ASN.1 correspondant. Ces classes de codage représentent la structure d'un codage ASN.1 et sont formées à partir de classes de codage intégrées reflétant la structure d'une définition de type ASN.1.

9.2.2.3 Classes de codage pour les structures de codage définies par l'usager – Ces classes de codage sont définies par l'usager de la notation ECN au moyen de la spécification d'une structure de codage (voir § 9.3) comme une structure constituée de champ binaires et de constructeurs de codage. Ces structures de codage sont similaires aux structures de codage produites implicitement, mais l'usager de la notation ECN dispose d'une notation de contrôle complète de leur structure. Ces classes permettent de définir des règles de codage complexes et sont importantes pour l'utilisation de la notation ASN.1 avec la notation ECN afin de spécifier des protocoles existants, dans lesquels des champs binaires supplémentaires sont nécessaires dans le codage afin de tenir compte des déterminants.

9.2.2.4 Classes de codage pour les structures de codage produites explicitement – Ces classes de codage sont produites à partir d'une structure de codage produite implicitement par modification sélective des noms de certaines classes afin d'indiquer les emplacements où des codages spécialisés sont nécessaires pour des offres d'options, la terminaison d'une séquence d'items, etc.

9.3 Structures de codage

9.3.1 Les définitions de structure de codage ont une certaine similarité avec les définitions de type ASN.1, et ont un nom commençant par le caractère "**#**", dans ce cas une lettre en haut de casse. Chaque définition de la structure de codage définit une nouvelle classe de codage (l'ensemble de tous les codages possibles de cette structure de codage). Les structures de codage sont formées à partir de champs qui sont soit des classes de codage intégrées soit les noms d'autres structures de codage, combinées au moyen des constructeurs de codage (qui représentent l'ensemble de toutes les règles possibles de codage qui prennent en charge leur type de mécanisme de construction, et qui sont donc appelées classes de codage). (Voir au § D.2.8.4 un exemple de définition de structure de codage.)

9.3.2 Les constructeurs de codage les plus élémentaires sont: **#CONCATENATION**, **#REPETITION** et **#ALTERNATIVES**, correspondant approximativement aux types ASN.1 "sequence" (et "set"), "sequence-of" (et "set-of"), et "choice". Il y a également une classe de codage **#OPTIONAL** qui représentente la présence facultative de codages, correspondant approximativement aux marqueurs ASN.1 **DEFAULT** et **OPTIONAL**.

9.3.3 Une définition de structure de codage définit une classe de codage fondée sur une structure. De telles classes ne peuvent pas avoir les mêmes noms que les classes de codage qui sont importées dans le module. (Voir Rec. UIT-T X.680 | ISO/CEI 8824-1, § 12.12, telle que modifiée par le § A.1 de la présente Recommandation | Norme internationale.)

9.3.4 Les noms de structure de codage peuvent être exportés et importés entre modules de définition de codage et peuvent être utilisés chaque fois qu'un nom de classe de codage est requis dans le groupe catégoriel des champs binaires (voir § 9.6).

9.3.5 Les valeurs de types ASN.1 (primitifs ou définis par l'usager) peuvent être mappées sur les champs d'une structure de codage et les règles de codage pour ces structures fournissent alors des codages de type ASN.1. (Les valeurs mappées sur des structures de codage peuvent être encore mappées sur des champs de structures de codage plus complexes.) Elles offrent un mécanisme très puissant afin de définir des règles de codage complexes.

9.4 Objets de codage

9.4.1 Les objets de codage représentent la définition spécifique de règles de codage pour une classe de codage donnée. Habituellement, les règles se rapportent aux bits réels à produire, mais peuvent également spécifier des procédures associées au codage et au décodage, par exemple la façon dont la présence ou l'absence de composants facultatifs est déterminée.

9.4.2 Afin de définir complètement le codage de types ASN.1 (normalement le ou les types de niveau supérieur d'une application), il est nécessaire de définir (ou d'obtenir à partir des règles de codage normalisées) des objets de codage pour toutes les classes qui correspondent à des composants de ces types ASN.1 et pour les constructeurs de codage qui sont utilisés.

9.4.3 Pour les protocoles existants, cela peut être effectué par définition d'un objet de codage séparé pour chaque composant d'un type ASN.1, mais il est plus généralement possible d'utiliser des objets de codage définis par des règles de codage normalisées (telles que PER).

9.4.4 Bien que les spécifications de codage BER et PER soient antérieures à la notation ECN, elles définissent simplement dans le modèle de notation ECN des objets de codage pour toutes les classes correspondant aux types primitifs et aux constructeurs ASN.1 (c'est-à-dire, pour toutes les classes de codage intégrées). Les règles BER et PER sont également prises en considération afin d'offrir des objets de codage pour les classes de codage utilisées dans la définition de structures de codage (voir § 18.2).

9.5 Ensembles d'objets de codage

9.5.1 Les objets de codage peuvent être regroupés en ensembles de la même façon que les objets informationnels en notation ASN.1, et ce sont ces ensembles d'objets de codage qui sont (dans un module ELM) appliqués à un type ASN.1 afin de déterminer son codage. Le gouverneur utilisé lors de la formation de ces ensembles d'objets de codage est le mot réservé **`#ENCODINGS`**. (Voir § D.1.14 à titre d'exemple.)

9.5.2 Une règle fondamentale de construction d'un ensemble d'objets de codage est que tout ensemble ne peut contenir qu'un seul objet de codage d'une classe de codage donnée (voir également § 9.6.2). Donc il n'y a pas d'ambiguïté lorsqu'un ensemble d'objets de codage est appliqué à un type afin de définir son codage.

9.5.3 Il existe des ensembles d'objets de codage intégrés pour toutes les variantes des règles BER et PER et ces ensembles peuvent être utilisés afin de réaliser des ensembles d'objets de codage définis par l'usager.

9.6 Définition de nouvelles classes de codage

9.6.1 Ceux pour lesquels la notation ASN.1 est familière seront conscients du fait qu'une attribution de type peut être utilisée afin de créer de nouveaux noms (de nouveaux types) à partir, par exemple, des types **`INTEGER`** ou **`BOOLEAN`**. Les nouveaux noms désignent des types qui sont les mêmes que **`INTEGER`** ou **`BOOLEAN`**, mais contiennent une sémantique différente. Ce concept est étendu en notation ECN afin de permettre la création (dans une attribution de classe – voir § 16.1.1) de nouveaux noms (nouvelles classes) pour des constructeurs tels que **`#SEQUENCE`**. Ces nouveaux noms désignent des classes qui remplissent une fonction similaire dans les codages structurants (par exemple, une concaténation), mais qui doivent avoir différents objets de codage qui leur sont appliqués. Un nouveau nom de classe attribué à une ancienne classe conserve certaines caractéristiques de cette ancienne classe, de sorte qu'une attribution telle que "**`#My-Sequence ::= #SEQUENCE`**" crée le nouveau nom de classe **`#My-Sequence`** qui est encore une classe de codage concernant la concaténation de composants. L'on déclare que de telles classes de codage sont dans la même catégorie.

9.6.2 Si une nouvelle classe de codage est créée à partir d'une classe de codage existante, des objets de codage appartenant aussi bien à l'ancienne classe de codage qu'à la nouvelle classe de codage peuvent apparaître dans un ensemble d'objets de codage.

9.6.3 Toutes les classes de codage intégrées sont issues d'un petit nombre de classe primitives de codage. Donc les classes **#SEQUENCE** et **#SET** sont toutes les deux issues de la classe **#CONCATENATION**, les classes **#INTEGER** et **#ENUMERATED** sont toutes les deux issues de la classe **#INT** et les classes pour les différents types ASN.1 de chaîne de caractères sont toutes issues de la classe **#CHARS**. Une structure de codage (par exemple, produite implicitement à partir d'un type ASN.1) peut contenir un mélange de différentes classes toutes issues de la même classe primitive, permettant l'application de différents codages à **#SEQUENCE** et **#SET** (par exemple).

9.6.4 Il est souvent pratique de ranger les classes de codage en catégories, sur la base de la classe primitive dont elles sont issues. Donc l'on déclare que **#INTEGER**, **#ENUMERATED** et **#INT** (et toute classe issue des précédentes dans une instruction d'attribution de classe telle que "**#My-int ::= #INT**") sont dans la catégorie des entiers. Il existe également des groupes catégoriels qui contiennent des classes très différentes qui ont en commun certaines caractéristiques. Donc toute classe qui peut avoir des valeurs abstraites qui lui sont directement associées, et donc qui produit des bits dans un codage, est dite appartenir au groupe catégoriel des champs binaires. Donc toutes les classes qui sont dans la catégorie des entiers ou des booléens ou des chaînes de caractères sont dans le groupe catégoriel des champs binaires. Les classes qui sont chargées de grouper ou de répéter des codages (par exemple les classes appartenant à la catégorie des options ou des répétitions) sont dans le groupe catégoriel des constructeurs de codage. Il existe également deux classes dont les objets de codage définissent des procédures non directement associées à la construction d'un codage (**#TRANSFORM** et **#OUTER**): ces classes sont décrites comme étant dans le groupe catégoriel des procédures de codage. Les structures de codage sont définies au moyen de classes dans le groupe catégoriel des champs binaires qui sont combinées au moyen de classes dans le groupe catégoriel des constructeurs de codage, ainsi qu'avec des classes des catégories d'offres d'options (représentant les procédures de codage pour résoudre les offres d'options) et d'étiquettes (représentant le codage d'étiquettes). Toutes ces classes sont dans la catégorie des structures de codage (et également dans le groupe catégoriel des champs binaires).

9.6.5 Pour les classes primitives, la catégorie est directement attribuée. Pour les classes créées dans une instruction d'attribution de classe de codage, la catégorie est déterminée par la notation située à droite du symbole "**::=**". Si cette notation est une définition de la structure de codage, dans ce cas la classe est aussi bien dans la catégorie des structures de codage que dans le groupe catégoriel des champs binaires. Si la notation est un simple nom de référence de classe, dans ce cas la catégorie de la nouvelle classe est la même que la catégorie de la classe en cours d'attribution.

9.6.6 Les catégories de classe de codage (voir § 16.1.3) sont les suivantes:

- la catégorie des options (classes qui sont obtenues par attribution de classe à partir de **#ALTERNATIVES**);
- la catégorie des concaténations (classes qui sont obtenues par attribution de classe à partir de **#CONCATENATION**);
- la catégorie des répétitions (classes qui sont obtenues par attribution de classe à partir de **#REPETITION**);
- la catégorie des offres d'options (classes qui sont obtenues par attribution de classe à partir de **#OPTIONAL**);
- la catégorie des étiquettes (classes qui sont obtenues par attribution de classe à partir de **#TAG**);
- les catégories des booléens, des chaînes de bits, des chaînes de caractères, des entiers, des néants, des identificateurs d'objet, des chaînes d'octets, des types ouverts, des bourrages et des réels (catégories pour classes issues des classes primitives correspondantes);
- la catégorie des structures de codage (classes produites à partir des définitions de type ASN.1, ou par définition explicite d'une structure de codage).

9.6.7 Les groupes catégoriels suivants sont définis:

- le groupe catégoriel des champs binaires (classes qui correspondent à des champs réels dans un codage, comme celles des catégories des entiers ou des booléens, ainsi que toute classe de la catégorie des structures de codage). Les classes contenues dans ce groupe catégoriel sont également désignées comme des classes de champ binaire;
- le groupe catégoriel des constructeurs de codage (classes qui sont dans les catégories des options, des concaténations ou des répétitions). Les classes contenues dans ce groupe catégoriel sont également désignées comme les classes de constructeur de codage;
- le groupe catégoriel des procédures de codage (classes non directement associées à des créations ASN.1 et qui ne peuvent pas recevoir de nouveaux noms – **#OUTER**, **#TRANSFORM**, **#CONDITIONAL-INT**, **#CONDITIONAL-REPETITION**). Les classes contenues dans ce groupe catégoriel sont également désignées comme les classes de procédure de codage.

9.7 Définition des objets de codage

Il y a huit mécanismes disponibles afin de définir un objet de codage d'une classe de codage donnée. Ils ne sont pas tous disponibles pour toutes les classes de codage.

9.7.1 Le premier mécanisme vise à spécifier cet objet comme étant le même que certains autres objets de codage définis de la classe requise, ce qui revient à fournir un synonyme pour des objets de codage.

9.7.2 Le deuxième mécanisme, disponible pour un ensemble de classes de codage restreint, vise à utiliser une syntaxe définie (voir § 17.2) afin de spécifier les informations nécessaires pour définir un objet de codage de cette classe. Une grande partie des informations nécessaires est commune à toutes les classes de codage, mais certaines de ces informations dépend toujours de la classe spécifique de codage. (Voir § D.1.1.2 à titre d'exemple de définition d'un objet de codage de classe **#BOOLEAN** qui contient des codages pour le booléen de type ASN.1.)

9.7.3 Le troisième mécanisme, disponible pour toutes les classes de codage, vise à définir un objet de codage comme étant le codage de la classe requise qui est contenu dans certains ensembles d'objets de codage existants. Il est surtout utilisé pour nommer un objet de codage d'une classe particulière qui effectuera des codages BER ou PER pour cette classe.

NOTE – Cela peut souvent être utile, mais nécessite la connaissance des codages produits par les règles de codage normalisées.

9.7.4 Le quatrième mécanisme vise à mapper les valeurs abstraites associées à une classe de codage ("**#A**", par exemple) à des valeurs abstraites associées à une autre (normalement plus complexe) classe de codage ("**#B**", par exemple), et à définir un objet de codage pour "**#B**" (au moyen de n'importe lequel des mécanismes disponibles). Un objet de codage pour les valeurs abstraites associées à "**#A**" peut donc être défini comme l'application aux valeurs abstraites correspondantes, associées à "**#B**", de l'objet de codage pour "**#B**". (Voir § D.2.8.3 à titre d'exemple.) Il y a de nombreuses variantes de cette correspondance (voir § 9.17).

NOTE – C'est le modèle qui sous-tend la définition d'un objet afin de coder un type d'entier dans les règles BER. L'entier est mappé sur une structure de codage qui contient un champ de classe d'étiquettes (**UNIVERSAL**, **APPLICATION**, **PRIVATE**, ou propre au contexte), un booléen primitif ou constructeur, un champ de numéro d'étiquette et une partie valeur qui code les valeurs abstraites de l'entier original.

9.7.5 Le cinquième mécanisme vise à définir un objet de codage pour une classe (correspondant par exemple à un type défini par l'usager ASN.1) au moyen d'une définition distincte des objets de codage pour les composants et pour le constructeur de codage utilisé afin de définir la classe de codage.

9.7.6 Le sixième mécanisme vise à définir un objet de codage pour un codage-décodage différentiel (voir § 9.8), au moyen de deux objets de codage distincts, dont l'un définit le comportement du codeur, et l'autre indique à un décodeur quel codage devrait être pris comme hypothèse.

NOTE – Un exemple consisterait à coder par zéro partout un champ "réservé pour usage futur", mais à accepter toute valeur lors du décodage.

9.7.7 Le septième mécanisme vise à définir un objet de codage d'options de codage contenant une liste ordonnée d'objets de codage de la même classe. C'est au codeur qu'il appartient de choisir quel objet de codage doit être appliqué à partir de la liste, sous réserve de la restriction que cette option ne doit être utilisée que si une seule option de codage peut coder une valeur abstraite donnée et sous réserve de la recommandation que le premier codage disponible dans la liste sera utilisé.

NOTE – Un objet de codage d'options de codage pourrait, par exemple, être utilisé dans la spécification de codages de longueur abrégés qui pourraient coder une longueur de chaîne particulière, avec utilisation de codages de longueur de forme longue lorsque la forme courte ne peut pas être utilisée. Il n'y a pas actuellement de mécanisme permettant au spécificateur en notation ECN d'exiger l'utilisation du premier objet de codage disponible (si plus d'un seul peut coder la valeur abstraite), sinon par commentaire.

9.7.8 Finalement, un objet de codage peut être défini au moyen d'une notation non ECN. C'est un moyen permettant d'utiliser une quelconque notation désirée (y compris le langage naturel) afin de définir l'objet de codage (voir § D.2.7.3).

NOTE – Une notation non ECN devrait être utilisée avec précaution, car une prise en charge par utilitaire de l'implémentation est généralement impossible dans ce cas.

9.8 Codage-décodage différentiel

9.8.1 Le codage-décodage différentiel est le terme appliqué à une spécification qui nécessite l'acceptation par une implémentation (lors du décodage) de séquences binaires qui s'ajoutent à celles qu'il est permis de produire lors de l'exécution du codage.

9.8.2 Le codage-décodage différentiel sous-tend toutes les prises en charge relatives à une "extensibilité" (capacité d'une implémentation d'une version antérieure d'une norme à offrir une bonne interopérabilité avec une implémentation d'une version ultérieure de la norme).

9.8.3 La nature précise du codage-décodage différentiel peut être tout à fait complexe. Ce codage implique normalement qu'un décodeur accepte (et néglige sans notification) le bourrage des champs (habituellement de longueur variable) que les versions ultérieures d'une norme utiliseront pour le transfert d'informations supplémentaires à celles qui ont été transférées lors de la communication selon la version antérieure.

9.8.4 La prise en charge du codage-décodage différentiel en notation ECN est fournie par une syntaxe qui permet la définition d'un objet de codage (pour toute classe) qui encapsule deux objets de codage. Chaque objet de codage définit des règles afin de coder. Le premier objet de codage définit les règles qu'un codeur utilise. Le décodeur utilise le second objet de codage en tant que spécification de la façon dont le codage a été effectué.

NOTE – En notation ECN, les règles qu'un décodeur utilise (dans une version antérieure d'une norme) sont toujours exprimées par l'indication des règles de codage que le correspondant de communication est censé être en train d'utiliser. Les règles de décodage ne sont pas données en tant que règles de décodage explicites. Le spécificateur ECN s'assurera que de telles règles de décodage fournissent toute "extensibilité" nécessaire.

9.9 Options de codeur dans les codages

9.9.1 Les options de codeur dans les protocoles sont généralement considérées aujourd'hui comme quelque chose à éviter, mais la notation ECN doit offrir la prise en charge de telles options si un concepteur de protocole décide (ou a déjà décidé) de les inclure.

9.9.2 Lorsque des valeurs sont codées dans un espace de codage, il est possible de spécifier que la taille de l'espace de codage (voir § 9.21.5) est une option de codeur, à condition qu'il y ait une certaine forme de déterminant de longueur associé au codage. (La portée des options de codeur peut être limitée par la valeur maximale qui peut être codée dans le déterminant de longueur.) Elles offrent un niveau détaillé de prise en charge d'options de codeur.

9.9.3 Un mécanisme plus global est semblable à la prise en charge du codage-décodage différentiel (voir § 9.8), mais dans ce cas un objet de codage pour une classe peut être défini comme un choix par un codeur d'un quelconque objet de codage à partir d'une liste ordonnée d'objets de codage définis pour cette classe. Par ailleurs, afin de spécifier la liste de codages possibles, il est également nécessaire d'offrir la spécification d'un objet de codage pour une classe de la catégorie des options (voir § 9.6). Cet objet de codage spécifie les codages et procédures nécessaires afin de permettre à un décodeur de déterminer quel objet de codage a été utilisé par le codeur.

9.10 Propriétés des objets de codage

9.10.1 Les objets de codage ont certaines propriétés générales. Le plus souvent, ils définissent complètement un codage, mais dans certains cas ils sont des **constructeurs de codage**, c'est-à-dire qu'ils définissent seulement les aspects structuraux du codage, ce qui nécessite des objets de codage pour les composants de la structure de codage afin de réaliser la définition d'un codage.

9.10.2 Une autre caractéristique clé d'un objet de codage est qu'il peut recueillir des informations à partir de l'environnement où ses règles sont finalement appliquées. Un aspect de l'environnement qui est complètement pris en charge est la présence de limites dans la définition de type ASN.1, à condition qu'ils soient "visibles par les règles PER" (voir Rec. UIT-T X.691 | ISO/CEI 8825-2, § 9.3).

NOTE – Une dépendance externe un peu différente (et non normalisée) serait la définition d'un objet de codage non ECN pour une classe de codage **`#ALTERNATIVES`** qui détermine l'option choisie sur la base de données externes telles que la voie sur laquelle le message est envoyé.

9.10.3 Une troisième caractéristique clé est qu'un objet de codage peut faire apparaître un **pointeur d'identification** dans ses codages. C'est une partie de tous les codages qu'il produit, qui distingue ses codages de ceux d'autres objets de codage (d'une classe quelconque) qui font apparaître le même pointeur d'identification. Les pointeurs d'identification doivent être visibles par les décodeurs sans la connaissance de la classe de codage ou de la valeur abstraite qui a été codée (mais avec la connaissance du nom du pointeur d'identification qui est utilisé). Ce concept modélise (et généralise) l'utilisation d'étiquettes dans les codages BER: la valeur d'étiquette dans les règles BER peut être déterminée sans la connaissance de la classe de codage, pour tous les codages BER et sert à désigner le codage pour la résolution d'offres d'options, le séquencement d'ensembles et le choix d'options.

9.11 Paramétrage

9.11.1 Comme avec les types et valeurs ASN.1, les objets de codage, les ensembles d'objets de codage et les classes de codage peuvent être paramétrés. Il ne s'agit que d'une extension du mécanisme normal ASN.1.

9.11.2 Une utilisation primaire du paramétrage est la définition d'un objet de codage qui a besoin de l'identification d'un déterminant afin de réaliser la définition du codage (voir § 9.13.2). (Voir § D.1.11.3 à titre d'exemple de définition ECN paramétrée.)

9.11.3 Une autre importante utilisation de paramétrage est la définition d'une structure de codage que sera utilisée afin de remplacer de nombreuses classes différentes dans un codage (voir également § 9.16.5). Par exemple, le mécanisme utilisé pour gérer les offres d'options est souvent un "bit de présence" (obligatoire) immédiatement antérieur, pour chaque composant facultatif. Une structure paramétrée peut être définie comme étant composée d'une concaténation d'une classe **#BOOLEAN** (utilisée comme déterminant de présence) suivie d'un composant facultatif défini comme un paramètre fictif (qui sera instancié avec le composant que la structure remplacera) et dont la présence est déterminée par la classe **#BOOLEAN**. La procédure de codage **#OPTIONAL** originale est donc définie comme le remplacement du composant original par cette structure obligatoire, au moyen du composant facultatif original en tant que paramètre réel. (Le § D.3.2 est un exemple plus complet de ce processus.)

9.11.4 Les paramètres fictifs peuvent être des objets de codage, des ensembles d'objets de codage, des classes de codage, des références à des champs de structure de codage et des valeurs d'un quelconque des types ASN.1 utilisés dans les classes de codage intégrées définies dans l'article 23, comme spécifié dans la Rec. UIT-T X.683 | ISO/CEI 8824-4 telle que modifiée par le § B.10 de la présente Recommandation | Norme internationale.

9.11.5 La modification de la syntaxe de paramétrage qui est spécifiée dans l'Annexe C nécessite l'utilisation du symbole "**{<**" (sans espaces) au lieu de "**{**" pour commencer une liste de paramètres fictifs ou réels, et de "**>}**" pour en terminer une.

NOTE – Cette opération a été effectuée afin de faciliter l'analyse de la syntaxe ECN par les ordinateurs et d'éviter toute ambiguïté lorsque des classes définies par l'usager sont utilisées dans des définitions de structure à la place des classes **#SEQUENCE**, **#CHOICE**, **#REPETITION**, **#SEQUENCE-OF** ou **#SET-OF**.

9.12 Gouverneurs

9.12.1 Le concept de gouverneur et de notation gouvernée sera familier d'après la notation de valeur ASN.1, où il y a toujours une définition de type qui "gouverne" la notation de valeur et qui détermine sa syntaxe et sa signification.

9.12.2 Le même concept s'étend à la définition d'objets de codage d'une classe de codage donnée. La syntaxe définissant un objet de codage de classe **#BOOLEAN** (par exemple) est très différente de la syntaxe définissant un objet de codage de classe **#INTEGER** (par exemple). Dans tous les cas où une définition d'objet de codage est requise, il y a une notation associée qui définit la classe de cet objet de codage et qui "gouverne" la syntaxe à utiliser dans sa spécification.

9.12.3 La syntaxe ECN nécessite des gouverneurs qui soient des noms de référence de classe à créer ou des noms de référence de classe paramétrés.

9.12.4 Si la notation gouvernée est un nom de référence pour un objet de codage, dans ce cas cet objet de codage doit être de la même classe que le gouverneur (voir § 17.1.7).

9.13 Aspects généraux des codages

9.13.1 La notation ECN offre la prise en charge d'un certain nombre de techniques normalement utilisées afin de définir des règles de codage (et non pas seulement les techniques utilisées dans les règles BER ou PER). Par exemple, elle reconnaît que les offres d'options peuvent être résolues dans n'importe laquelle des trois façons suivantes: par l'utilisation d'un déterminant de présence, par l'utilisation d'un pointeur d'identification (voir § 9.13.3), ou par atteinte de l'extrémité d'un conteneur délimité en longueur (ou de l'extrémité de l'unité PDU) avant que le composant facultatif apparaisse.

9.13.2 De la même façon, elle reconnaît que la délimitation de répétitions peut être effectuée (par exemple) comme suit:

- par comptage de longueur selon une méthode quelconque;
- par détection de l'extrémité d'un conteneur (ou d'une unité PDU) dans lequel cet item est le dernier;
- par utilisation d'un pointeur d'identification sur chacune des répétitions et sur les codages suivants (voir § 9.13.3);
- par une séquence de terminaison qui ne peut jamais apparaître dans le codage d'une série récurrente. (Un simple exemple en est une chaîne de caractères à terminaison par néant.);
- par l'utilisation d'un "bit d'extension" avec chaque élément, mis à un afin d'indiquer qu'une autre répétition suit, et mis à zéro afin d'indiquer la fin de la répétition.

La notation ECN prend en charge tous ces mécanismes pour la délimitation de répétitions et des mécanismes analogues pour identification des options et pour la résolution des offres d'options.

9.13.3 En plus de l'indication de la fin de répétitions, la technique du pointeur d'identification peut également être utilisée afin de déterminer la présence de composants facultatifs ou d'options. Le mécanisme est semblable dans tous

ces cas. Le codage de toutes les valeurs d'un codage de "prochaine classe possible" aura la même séquence binaire (leur identification) à certains endroits de ce codage (le pointeur), mais l'identification de différents codages de "prochaine classe possible" sera différente pour chacun d'eux. Tous les codages de ce type peuvent être interprétés par un décodeur comme un codage d'une quelconque "prochaine classe possible" et l'identification du pointeur déterminera quel codage de "prochaine classe possible" est présent. Ce concept est semblable à celui de l'utilisation d'étiquettes à de telles fins dans les règles BER. Des pointeurs d'identification ont des noms qui doivent être uniques à l'intérieur d'une spécification ECN.

9.13.4 Il importe ici de remarquer que la notation ECN autorise la définition de codages d'une façon très flexible, mais ne peut pas garantir qu'une spécification de codage est correcte – c'est-à-dire qu'un décodeur peut rétablir correctement les valeurs abstraites originales à partir d'un codage. Par exemple, un spécificateur ECN pourrait attribuer la même séquence binaire pour les valeurs booléennes "vrai" et "faux". Ce serait une erreur, et dans ce cas un outil pourrait assez facilement détecter l'erreur. Une autre erreur serait de déclarer qu'un codage était autodélimitateur (et ne nécessitait aucun déterminant de longueur), alors qu'en fait il ne l'était pas. Cette erreur pourrait également être détectée par un outil. Dans des cas plus subtils et plus complexes, cependant, un outil peut trouver très difficile de diagnostiquer une spécification erronée (c'est-à-dire qui ne peut pas toujours être décodée correctement).

9.14 Identification des éléments d'information

9.14.1 De nombreux protocoles ont un codage (habituellement d'un nombre fixe de bits) permettant de désigner les éléments qui sont souvent appelés "éléments d'information" ou "éléments de données" dans un protocole. Ces identifications correspondent approximativement aux étiquettes ASN.1, mais elles sont habituellement moins complexes. Elles sont souvent utilisées en tant que pointeurs d'identification, mais ce n'est pas toujours le cas.

9.14.2 La notation ECN contient une classe **`#TAG`** afin de prendre en charge la définition du codage d'identificateurs d'élément d'information au moyen de la notation par étiquettes ASN.1. (Elle prend également en charge l'inclusion de tels éléments à l'intérieur d'une structure de codage sans référence aux étiquettes ASN.1.)

9.14.3 Lorsqu'une structure de codage est produite implicitement à partir d'une définition de type ASN.1 (voir l'article 11), la première notation par étiquettes ASN.1 **présente textuellement** dans cette définition produit une instance de la classe **`#TAG`**, avec le numéro de l'étiquette ASN.1 associée à cette instance de la classe **`#TAG`**. Les instances textuellement présentes de notation par étiquettes ASN.1 suivantes ne sont pas mappées dans des classes **`#TAG`** contenues dans la structure produite implicitement, mais ces étiquettes et leur valeur deviennent des propriétés de l'élément. Un codage pour cette classe de codage peut être défini de façon analogue à un codage pour la classe **`#INTEGER`** et codera le numéro dans la notation par étiquettes.

9.14.4 La liste complète d'étiquettes ASN.1 (étiquettes multiples, chacune avec une classe et un numéro) est théoriquement associée à toutes les valeurs abstraites d'un type étiqueté, conformément au modèle ASN.1. Une telle information n'est, cependant, accessible que dans la version actuelle de la notation ECN, au moyen d'une définition non ECN d'un objet de codage (voir § 9.7.8). La production d'une classe **`#TAG`** est un mécanisme distinct, plus simple et plus concis, pleinement pris en charge dans notation ECN.

9.14.5 Il est cependant important de noter qu'aux fins de la production d'une classe **`#TAG`**, c'est seulement la notation par étiquettes présente textuellement qui est visible. Les étiquettes de classe universelle et les étiquettes produites par balisage automatique ne sont pas visibles. De la même façon, la classe d'une quelconque notation par étiquettes présente textuellement est ignorée. Seul le numéro d'étiquette est disponible aux objets de codage de la classe **`#TAG`**.

9.15 Champs et déterminants de référence

9.15.1 Une façon très courante (mais non la seule) de déterminer la présence d'un champ facultatif, la longueur d'une répétition, ou le choix d'une option, consiste à inclure (quelque part dans le message) un champ de déterminant. Les champs de déterminant doivent être identifiés si ce mécanisme est utilisé pour la détermination, et cela nécessite fréquemment un paramètre fictif d'une définition d'objet de codage, le paramètre réel fournissant le nom du champ de structure de codage du déterminant étant fourni lorsque l'objet de codage est appliqué à une structure de codage.

9.15.2 Un nouveau concept – le **champ de référence** – est introduit afin de répondre au besoin d'un paramètre fictif qui fait référence à un champ de structure de codage. Le gouverneur est le mot réservé **`REFERENCE`**, et la notation permise pour un paramètre réel avec ce gouverneur est tout nom de champ de structure de codage situé à l'intérieur de la structure de codage à laquelle un objet de codage ou un ensemble d'objets de codage est en cours d'application avec un tel paramètre (voir § 17.5.15). (Voir § D.1.11.3 à titre d'exemple de références à des noms de champ de structure de codage.)

9.16 Classes et structures de remplacement

9.16.1 Lors de la rédaction de spécifications ASN.1 pour protocoles existants (ou afin de produire des codages spécialisés pour de nouveaux protocoles), il est normal d'ignorer les questions de codage et, en particulier, les champs de déterminant qui ne sont présents qu'afin de prendre en charge le décodage. Seuls les champs relatifs à un code application (acheminant la sémantique applicative) sont inclus dans la spécification ASN.1.

9.16.2 Lorsque de tels protocoles utilisent plusieurs mécanismes de codage afin de prendre en charge (par exemple) des constructions **`SEQENCE OF`** à différents endroits du protocole, il n'est pas possible (ni approprié) de spécifier formellement cela dans la notation ASN.1 elle-même.

9.16.3 Cela implique que la structure de codage produite implicitement ne fera pas de distinction entre de telles constructions, ni ne contiendra de champs relatifs au codage afin de tenir compte des déterminants. Il sera donc nécessaire de la modifier afin de "corriger" ces deux problèmes avant qu'une structure répondant aux exigences de codage soit disponible.

9.16.4 La première et la plus simple modification vise à remplacer certaines instances d'une classe (à l'intérieur de la structure produite implicitement) par les noms de nouvelle classe auxquels a été attribuée l'ancienne classe dans une instruction d'attribution de classe. Cette opération est effectuée par création d'une **structure produite explicitement** utilisant une clause de renommage dans un module EDM. Cette clause importe une structure produite implicitement à partir d'un module ASN.1 et effectue des remplacements spécifiés d'instances (textuelles) de classes nommées. Le remplacement peut porter sur toutes les instances se trouvant textuellement dans une liste de classes produites implicitement (correspondant aux définitions de type ASN.1 dans un module), ou à l'intérieur de composants de l'une de ces classes, ou sur "toutes les instances sauf" celles qui se trouvent dans une définition de données ou dans un composant de données (voir § 15.3). Il importe ici de noter que ces remplacements sont restreints à l'utilisation de classes qui ont été définies avec une instruction d'attribution de classe de codage qui attribue le nom d'une classe de remplacement à une ancienne classe (par exemple: "**`#Replacement-class ::= #Old-class`**"), de sorte que ce mécanisme est parfois familièrement désigné par le terme de "coloration". La "coloration" identifie les parties de la spécification qui nécessitent différents codages à partir d'autres parties. (Un exemple de "coloration" est donné au § D.3.7.)

9.16.5 Même avec "coloration", la structure de codage produite explicitement, comme la structure de codage produite implicitement, contient seulement les champs correspondant aux champs contenus dans la spécification ASN.1, et il est habituellement nécessaire de modifier les structures produites en ajoutant des champs afin de tenir compte des déterminants, etc. Une nouvelle **structure de remplacement** est nécessaire (pour tout ou partie de la structure originale), avec des champs ajoutés. Il est également important de désigner (pour chaque champ dans la structure originale) quels champs de la structure de remplacement (et quelles valeurs abstraites de ce champ) sont utilisés afin de contenir la sémantique des valeurs abstraites originales. Il s'agit ici du mappage des valeurs abstraites à partir de la structure originale vers la structure de remplacement.

9.16.6 Il y a de nombreux mécanismes permettant de définir un objet de codage pour une structure existante, comme un objet de codage pour une structure de remplacement totalement différente, avec des **mappages de valeur** définis entre l'ancienne structure et la structure de remplacement. Ces mécanismes sont décrits au § 9.17.

9.16.7 Une situation plus simple apparaît fréquemment, cependant, dans laquelle le concepteur oblige l'ancienne structure à former (dans son intégralité) un composant isolé de la structure de remplacement, toutes les valeurs abstraites étant mappées à partir de l'ancienne structure vers la valeur correspondante de ce composant de la structure de remplacement. Afin que ce mécanisme soit d'usage général, la structure de remplacement doit offrir un paramètre fictif représentant ce composant isolé, afin qu'il soit instancié avec l'ensemble de paramètres réels auprès de l'ancienne structure. Cela a été décrit au § 9.11.3.

9.16.8 Lors d'une définition des objets de codage pour une classe (quelconque), il est toujours possible de spécifier que la première action de cet objet de codage vise à remplacer la classe qu'il est en train de coder par une structure paramétrée de remplacement instanciée comme décrit au § 9.16.7 et par des valeurs abstraites mappées à partir de l'ancienne classe vers le composant.

9.16.9 Il est également possible de définir des objets de codage pour la classe **`#OPTIONAL`** (ou pour toute classe de la catégorie des offres d'options) qui remplacent le composant facultatif par une structure paramétrée de remplacement (contenant souvent un champ #BOOLEAN en tant que déterminant de présence). (Un exemple en est donné au § D.3.2.3.)

9.16.10 Pour les classes de constructeur telles que **`#CONCATENATION`**, **`#REPETITION`**, et ainsi de suite, il est également possible de définir des objets de codage qui remplacent non la structure entière, mais chaque composant séparément (ou seulement les composants obligatoires, ou seulement les composants facultatifs).

9.16.11 Un mécanisme plus élaboré, mais puissant, consiste à exiger de l'action de remplacement qu'elle comprenne également l'insertion d'un champ spécifié dans l'en-tête d'une structure **#CONCATENATION** (ou structure semblable). Un exemple en est donné au § D.3.1.5.

9.17 Mappage de valeurs abstraites sur des champs de structures de codage

Six mécanismes sont offerts à cette fin.

9.17.1 Le premier mécanisme vise à mapper des valeurs abstraites spécifiées associées à une simple classe de codage à des valeurs abstraites spécifiées associées à une autre simple classe de codage. Ce mécanisme peut être utilisé de plusieurs façons. Par exemple, les valeurs d'une chaîne de caractères (ou de chiffres) peuvent être mappées sur des valeurs entières (et donc codées en tant que valeurs entières). Des valeurs de type énuméré peuvent être mappées sur des valeurs entières, et ainsi de suite (voir § 19.2). (Voir § D.1.10.2 à titre d'exemple.)

9.17.2 Le second mécanisme vise à mapper un champ complet d'une structure de codage dans un champ d'une structure de codage compatible, qui peut contenir des champs supplémentaires – normalement pour utilisation en tant que déterminants de longueur ou de choix (voir § 19.3). (Voir § D.2.8.3 à titre d'exemple.)

9.17.3 Le troisième mécanisme vise à mapper par transformation toutes les valeurs abstraites associées à une classe de codage sur des valeurs abstraites associées à une classe de codage (normalement, mais non nécessairement) différente, au moyen d'un objet de codage de transformée (voir § 9.18). Ce mécanisme permet, par exemple, de mapper une classe **#INTEGER** sur une classe **#CHARS** afin d'obtenir des caractères qui peuvent alors être codés de n'importe quelle façon, au choix (par exemple, Binary-Coded Decimal ou ASCII). (Voir § D.1.6.3 à titre d'exemple.)

9.17.4 Le quatrième mécanisme de mappage consiste à utiliser une séquence définie des valeurs abstraites de certains types et constructions et à effectuer le mappage conformément à cette séquence. C'est un très puissant moyen de codeur des valeurs abstraites associées à une classe de codage comme si elles étaient des valeurs abstraites associées à une classe de codage totalement indépendante (voir § 19.5). (Voir § D.1.4.2 à titre d'exemple.)

9.17.5 Le cinquième mécanisme consiste à distribuer les valeurs abstraites (au moyen de la notation d'étendue de valeurs) associées à une classe de codage (normalement **#INTEGER**) entre les champs d'un autre classe de codage. (Voir § 19.6 et D.2.1.3 par exemple.)

9.17.6 Le mécanisme final autorise le spécificateur ECN à offrir un mappage explicite à partir de valeurs entières (qui peuvent avoir été produites par des mappages antérieurs à partir, par exemple, d'une classe **#ENUMERATED**) vers les bits qui sont à utiliser pour coder ces valeurs. Cela est prévu afin de prendre en charge les codages de Huffman, lorsque la fréquence d'instance de chaque valeur est (au moins approximativement) connue et où le codage optimal est requis. L'Annexe E décrit les codages de Huffman en de plus amples détails et donne des exemples de ce mécanisme de correspondance, ainsi qu'une référence à un logiciel qui produira la syntaxe ECN pour ces mappages sur la seule base de la fréquence relative à laquelle chaque valeur de l'entier est censée être utilisée (voir § 19.7).

9.18 Transformées et composites de transformée

9.18.1 Les transformées sont des objets de codage de la classe **#TRANSFORM**. Elles peuvent être utilisées afin de transformer des valeurs abstraites entre différentes classes de codage. Elles peuvent également être utilisées afin de définir de simples fonctions arithmétiques telles que la multiplication par une valeur fixe, la soustraction d'une valeur fixe, et ainsi de suite. Lorsqu'elles sont appliquées en succession, les transformées permettent de spécifier une arithmétique générale (voir § 19.4). (Voir § D.2.4.2 à titre d'exemple.)

9.18.2 Une transformée peut prendre une seule valeur en tant que source puis produire une seule valeur en tant que résultat. La liste suivante est une classification des valeurs qui peuvent être des sources et des résultats des transformées:

- un entier;
- un booléen;
- une chaîne de caractères;
- une chaîne de bits;
- un caractère isolé;
- un bit isolé (source seulement, prenant en charge le codage d'une chaîne de bits – voir § 23.2).

9.18.3 Les composites de transformation sont une liste ordonnée d'éléments, dont chacun est une seule valeur et a la même classification (comme énuméré dans le § 9.18.2). (Par exemple, une liste ordonnée de caractères isolés, ou d'octets isolés, ou d'entiers.) Ils ne sont produits que comme résultat de transformées et ne peuvent être utilisés que comme source d'une transformée suivante.

9.18.4 Si la classification est "chaîne de bits" la taille de chaque valeur de chaîne de bits dans le composite est la même et est déterminée statiquement par la transformée qui produit ce composite. (Par exemple, une liste ordonnée de bits isolés, ou d'unités de six bits.)

9.18.5 Il y a des transformées vers des composites à partir des valeurs abstraites suivantes:

- chaîne de caractères à composite de caractère isolé;
- chaîne de bits à composite de chaîne de bits (toutes les valeurs de chaîne de bits du composite sont de la même taille);
- chaîne d'octets à composite de chaîne de bits (toutes les valeurs de chaîne de bits du composite sont de taille 8 bits).

9.18.6 Il y a des transformées vers des valeurs abstraites à partir des composites suivants:

- composites de caractère isolé à valeurs de chaîne de caractères;
- composites de chaîne de bits à valeurs de chaîne de bits;
- composites de chaîne de bits (avec des valeurs de chaîne de bits de taille 8 bits) à valeurs de chaîne d'octets.

9.18.7 Toutes les autres transformées peuvent prendre une valeur comme source et produire une nouvelle valeur (de classification identique ou différente). Elles peuvent également prendre un composite de transformée comme source et produire un composite en tant que résultat, par transformation de chaque élément du composite-source en élément du composite-résultat.

9.19 Contenu des modules de définition de codage

9.19.1 Les modules de définition de codage (ou EDM, *encoding definition module*) contiennent des instructions d'exportation et d'importation exactement comme la notation ASN.1 (mais ils ne peuvent importer que des objets de codage, des ensembles d'objets de codage, et des classes de codage à partir d'autres modules EDM, ou à partir de modules ASN.1 dans le cas de structures de codage produites implicitement).

9.19.2 Un module EDM peut également contenir une clause de renommage (voir l'article 15) qui fait référence à des structures de codage produites implicitement à partir d'un ou de plusieurs modules ASN.1 et produit pour chacune par "coloration" de celle-ci (voir § 9.16.4), une structure de codage produite explicitement. Ces structures de codage produites explicitement sont disponibles pour utilisation dans le module EDM, mais sont également automatiquement exportées en vue d'une éventuelle importation dans le module de lien de codage.

9.19.3 Le corps d'un module EDM contient:

- des instructions "EncodingObjectAssignment" qui définissent et nomment un objet de codage pour certaines classe de codage (il y a huit formes de cette instruction, examinées au § 9.7 et définies dans l'article 17);
- des instructions "EncodingObjectSetAssignment" qui définissent des ensembles d'objets de codage (voir l'article 17);
- des instructions "EncodingClassAssignment" qui définissent et nomment de nouvelles classes de codage (voir l'article 15).

9.19.4 Le module EDM peut également contenir des versions paramétrées de ces instructions, comme spécifié dans l'article 14 et au § C.1.

9.19.5 Des objets de codage peuvent être définis pour des classes de codage intégrées dans tout module EDM. Des objets de codage ne peuvent être définis pour une structure de codage produite que dans des modules EDM qui importent la structure de codage produite implicitement à partir du module ASN.1 qui définit le type correspondant (au moyen d'une clause d'importation ou d'une clause de renommage) ou qui importent la structure de codage produite à partir d'un module EDM qui l'a exportée.

NOTE – Si une structure de codage produite implicitement se trouve avoir un nom qui est le même qu'un nom réservé de classe de codage (voir § 8.5), cette structure peut encore être importée dans un module EDM, mais doit être citée en référence dans le corps du module EDM au moyen d'un nom entièrement qualifié (voir "ExternalEncodingClassReference au § 10.6).

9.20 Contenu du module de lien de codage

9.20.1 Toutes les applications de la notation de contrôle de codage nécessitent l'identification d'un seul module de lien de codage (ou ELM).

9.20.2 Le module ELM applique des ensembles d'objets de codage à des types ASN.1 (formellement, à une structure de codage produite qui correspond au type ASN.1). Ces ensembles d'objets de codage (ou leurs objets de codage constituants) sont importés dans le module ELM à partir d'un ou de plusieurs modules EDM.

9.20.3 Il y a des restrictions sur l'application d'ensembles d'objets de codage afin de garantir qu'il n'y a pas d'ambiguïté au sujet des règles de codage qui sont réellement appliquées (voir § 12.2.5). Par exemple, il n'est pas permis qu'un module ELM applique plusieurs ensembles d'objets de codage à une certaine structure produite implicitement.

9.20.4 Il est possible dans les cas simples qu'un module ELM ne contienne qu'une instruction isolée (faisant suite à une clause d'importation) qui applique un ensemble d'objets de codage à la structure de codage produite implicitement qui correspond au type isolé de niveau supérieur d'une application. (Voir § D.1.17 à titre d'exemple.)

9.21 Définition des codages pour classes primitives de codage

9.21.1 Les règles de codage pour certaines classes primitives de codage peuvent être définies au moyen d'une syntaxe conviviale, qui est spécifiée dans les définitions des instuctions de classe de codage **`WITH SYNTAX`** (voir les articles 23 et 25). Cette syntaxe peut également être utilisée afin de définir des règles de codage pour classes de codage issues de ces classes primitives de codage (par instructions d'attribution de classe de codage).

9.21.2 La notation utilisée pour les définitions de classe de codage dans les articles 23 et 25 est fondée sur la notation utilisée pour la définition de la classe des objets informationnels. Cette syntaxe (et sa sémantique associée) est définie par référence à la Rec. UIT-T X.681 | ISO/CEI 8824-2 telle que modifiée par l'Annexe B de la présente Recommandation | Norme internationale.

9.21.3 La définition de classe de codage spécifie les informations qui doivent être fournies afin de définir des règles de codage pour des classes de codage particulières. L'ensemble des règles de codage qui peuvent être définies de cette façon ne contient évidemment pas toutes les règles possibles, mais est estimé couvrir les spécifications de codage que les usagers de la notation ECN sont susceptibles d'exiger.

9.21.4 Ces définitions de classe de codage spécifient une série de champs (avec les types et la sémantique ASN.1 correspondants). Les règles de codage sont spécifiées par fourniture de valeurs pour ces champs. Les valeurs de ces champs vont effectivement fournir les valeurs d'une série de propriétés de codage qui collectivement définissent un codage.

9.21.5 La signification des propriétés de codage est spécifiée au moyen d'un modèle de codage (voir Figure 1) lorsque la valeur de chaque classe de champ binaire produit un **codage de valeur** qui est placé (avec justification à gauche ou à droite) dans un **espace de codage**.

9.21.6 L'espace de codage peut avoir son bord antérieur aligné sur une frontière quelconque (telle qu'une limite d'octet) par prébourrage de l'espace de codage et sa taille peut être fixe ou variable. Le codage de valeur s'y intègre, éventuellement avec justification à gauche ou à droite et avec le bourrage autour de ce codage. Si la taille de l'espace de codage est variable, dans ce cas soit le codage de valeur doit être autodélimitateur, ou il doit y avoir certains mécanismes externes afin de permettre à un décodeur de déterminer la taille de l'espace de codage. Plusieurs mécanismes sont disponibles pour cette détermination.

9.21.7 Finalement, l'espace de codage complet est mappé, avec le codage de valeur et tout prébourrage ou postbourrage de valeur éventuel, sur les bits en ligne selon une spécification facultative d'inversion **de l'ordre des bits**. Ce mécanisme traite les codages qui nécessitent un "octet de poids fort en premier" ou un "octet de poids fort en dernier" pour les entiers, ou qui nécessitent que les bits contenus dans un octet soient dans l'inverse de l'ordre normal.

9.21.8 Il y a donc trois grandes catégories d'informations nécessaires:

- la première se rapporte à l'espace dans lequel le codage est placé;
- la deuxième se rapporte à la façon dont une valeur abstraite est mappée sur des bits (codage de valeur) et le positionnement de ces bits à l'intérieur de l'espace de codage;
- le troisième se rapporte à toute inversion requise de l'ordre des bits.

9.21.9 La Figure 1 montre l'espace de codage (avec prébourrage) et le codage de valeur (avec prébourrage et postbourrage de valeur). La Figure 1 décrit également la spécification d'une unité d'espace de codage. L'espace de codage est toujours un multiple entier de ce nombre de bits spécifié.

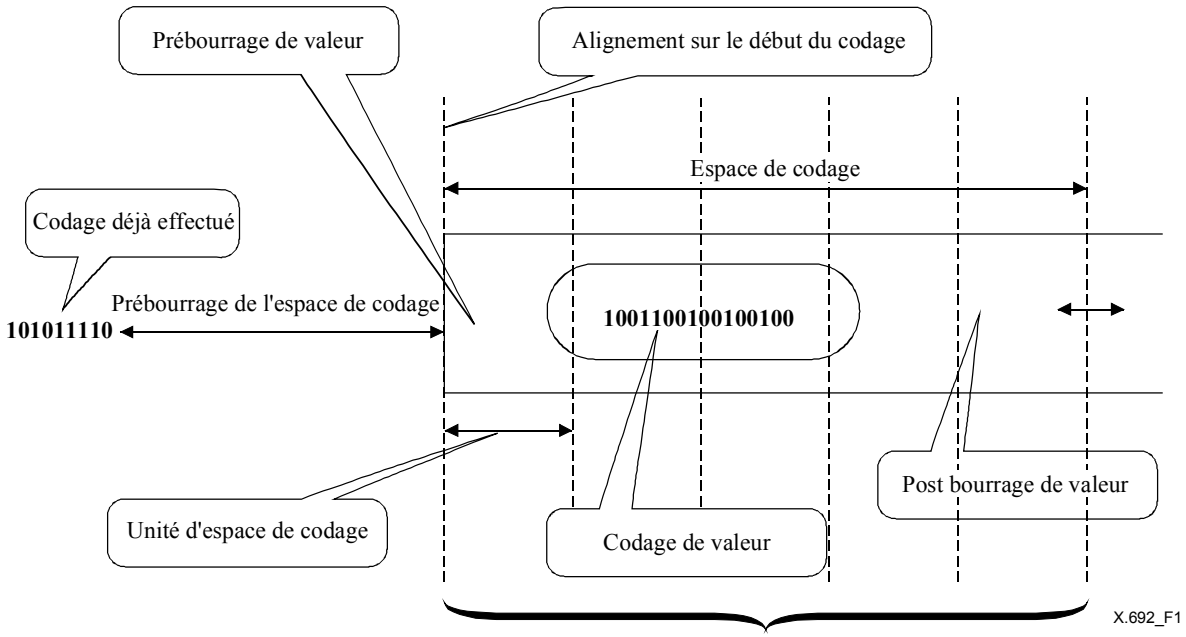

Figure 1 – Concepts d'espace de codage, de codage de valeur et de bourrage de valeur

9.21.10 Si l'espace de codage n'est pas de la même taille pour toutes les valeurs codées par un objet de codage, dans ce cas un certain mécanisme additionnel est nécessaire afin de déterminer l'espace de codage réellement utilisé dans une instance de codage.

9.21.11 Il est également possible de spécifier une quantité arbitraire de prébourrage par codeur (au-delà de celui qui est nécessaire pour l'alignement) qui se termine lorsque la valeur d'un **pointeur de début** antérieur identifie le début d'un champ.

9.21.12 Les étapes d'une définition de codage pour une classe de codage primitive de champ binaire sont les suivantes:

- elles spécifient l'alignement (si présent) requis pour le bord antérieur de l'espace de codage (par rapport au **point d'alignement** – normalement le début du codage du type de niveau supérieur, c'est-à-dire le type auquel un ensemble d'objets de codage est appliqué dans le module ELM) (voir § 22.2);
- elles spécifient la forme de tout bourrage nécessaire à ce point (prébourrage de l'espace de codage) (voir § 22.2);
- elles spécifient (si nécessaire) un champ qui offre un pointeur sur le point de départ de l'espace de codage (voir § 22.3);
- elles spécifient le codage de valeurs abstraites pour les transformer en bits (codage de valeur);
- elles spécifient les unités de l'espace de codage (qui sera toujours un multiple entier de ces unités) (voir § 22.4);
- elles spécifient la taille de l'espace de codage en ces unités. Cette taille peut être fixe (au moyen de la connaissance d'entiers ou de limites de taille associées aux valeurs abstraites à coder), ou variable (selon chaque valeur abstraite). La spécification peut également (dans tous les cas) spécifier l'utilisation d'un déterminant de longueur qui doit être codé avec la longueur du champ. Elle permet également le décodage ou offre des informations redondantes (dans le cas d'un espace de codage de taille fixe) qu'un décodeur peut vérifier (voir § 22.4);
- elles spécifient l'alignement du codage de valeur à l'intérieur de l'espace de codage (voir § 22.8);
- elles spécifient la forme de tout bourrage éventuellement nécessaire à partir du début de l'espace de codage jusqu'au début du codage de valeur (prébourrage de valeur) (voir § 22.8);

- elles spécifient la forme de tout bourrage éventuellement nécessaire entre l'extrémité du codage de valeur et l'extrémité de l'espace de codage (postbourrage de valeur) (voir § 22.8);
- elles spécifient toute inversion nécessaire de l'ordre des bits de contenu de l'espace de codage avant l'ajout des bits au codage déjà effectué (voir § 22.12).

9.21.13 Les propriétés de codage sont disponibles afin de prendre en charge la spécification du codage des règles pour toutes ces étapes.

9.21.14 Dans les cas réels, seules certaines (ou aucune!) de ces propriétés de codage auront (n'aura) des valeurs inhabituelles et les valeurs par défaut opèrent si elles ne sont pas spécifiées. (Voir § D.1.3 à titre d'exemple d'une définition du codage pour un entier qui est aligné à droite dans un champ fixe de deux octets, commençant à une limite d'octet.)

9.22 Application des codages

9.22.1 L'application des codages (règles de codage) à des structures de codage est une partie essentielle des travaux en notation ECN, mais elle est très distincte d'une définition des règles de codage. L'application finale des codages (à une structure de codage produite à partir d'une définition de type ASN.1) n'intervient qu'à l'intérieur d'un module de lien de codage, mais l'application des codages à des champs d'une structure de codage peut être utilisée dans la définition de codages pour une plus grande structure de codage.

9.22.2 Les codages sont appliqués par référence à un ensemble d'objets de codage (ou à un seul objet de codage). Une telle application peut apparaître dans un module EDM en vue d'une définition d'objets de codage pour une classe quelconque (y compris les objets de codage pour une structure de codage produite et pour une structure de codage définie par l'usager). Une telle application dans un module EDM est simplement la définition d'un plus grand nombre d'objets de codage pour cette classe de codage: l'application définitive à un type réel intervient seulement dans le module ELM.

9.22.3 Lorsqu'un ensemble d'objets de codage est en cours d'application, il en résulte toujours une spécification de codage complète pour les classes de codage auxquelles les objets sont appliqués. Une erreur est propagée si, dans une application donnée, des codages sont nécessaires pour des classes de codage (présentes à l'intérieur d'une structure de codage en cours de codage) auxquelles aucun objet de codage de l'ensemble n'est appliqué. (Voir § 13.2.11).

NOTE – Bien que la spécification du codage des règles soit complète, la forme précise du codage réel (par exemple la présence ou l'absence du prébourrage de l'espace de codage, ou l'effet des valeurs de limites citées en référence dans les règles de codage) ne peut être déterminé que lorsque la définition du codage est appliquée à un type ASN.1 de niveau supérieur.

9.22.4 Il y a deux exceptions au § 9.22.3. La première exception se produit lorsque le mécanisme de paramétrage (de type ASN.1) est utilisé afin de définir un objet de codage paramétré. Dans de tels cas, le codage complet n'est défini qu'après instanciation avec des paramètres réels. La deuxième exception se produit lorsqu'un objet de codage est défini pour un constructeur de codage (**`#CONCATENATION`**, **`#ALTERNATIVES`**, **`#REPETITION`**, **`#SEQUENCE`**, etc.). Dans ce dernier cas, les règles de codage associées à la classe de codage définissent simplement les règles associées aux aspects de structuration. Une spécification de codage complète pour une structure de codage utilisant ces classes de codage nécessitera également des règles afin de coder les composants de cette structure de codage.

NOTE – L'on distingue ici les objets de codage de classe **`#SEQUENCE`** (constructeur de codage) et les objets de codage relatifs à une structure de codage produite implicitement "**`#My-Type`**" (qui se trouve définie au moyen du type ASN.1 "**`SEQUENCE`**"). Ce dernier n'est pas un constructeur de codage et les objets de codage de cette classe fourniront des règles de codage complètes pour le codage de valeurs de type "**`My-Type`**".

9.23 Ensemble d'objets de codage combinés

9.23.1 Afin d'offrir un codage complet, l'usager de la notation ECN peut fournir un ensemble d'objets de codage primaire, et un second ensemble d'objets de codage introduit par les mots réservés "**`COMPLETED BY`**".

9.23.2 L'ensemble d'objets de codage qui est appliqué est défini comme étant l'ensemble **d'objets de codage combinés** formé par l'ajout au premier ensemble d'objets de codage pour toute classe de codage pour laquelle ce premier ensemble est dépourvu d'un objet de codage alors que le second ensemble le contient (voir § 13.2). Un ensemble fréquemment utilisé avec "**`COMPLETED BY`**" est l'ensemble intégré "**`PER-BASIC-UNALIGNED`**". (Voir § D.1.17 à titre d'exemple de l'application d'un ensemble d'objets de codage combinés.)

9.23.3 Alors qu'un ensemble d'objets de codage ne peut contenir qu'un seul objet de codage pour une classe **#SEQUENCE-OF** (par exemple), cet ensemble peut également contenir un objet de codage pour une classe **#Special-sequence-of** (par exemple) qui est définie comme "**#Special-sequence-of ::= #SEQUENCE-OF**". Une structure de codage produite explicitement peut avoir aussi bien la classe **#SEQUENCE-OF** que la classe **#Special-sequence-of** dans sa définition. De cette façon, un seul ensemble d'objets de codage combinés peut être appliqué afin de produire des codages normalisés pour certaines des constructions "**SEQUENCE OF**" originales et des codages spécialisés pour d'autres.

9.24 Point d'application

9.24.1 Dans toute application donnée des codages, il y a un point de départ défini (pour le module ELM, il s'agit de ou des structures de codage de niveau supérieur produites auxquelles des codages sont appliqués). C'est ce qui est appelé le "point d'application initial" pour la structure qui est codée par le module ELM.

9.24.2 L'ensemble d'objets de codage combinés est appliqué à une structure de codage produite et ce sont les codages définis pour les valeurs abstraites de cette structure de codage qui codent les valeurs abstraites du type ASN.1.

9.24.3 S'il y a un objet de codage dans l'ensemble d'objets de codage combinés qui correspond à une classe de codage de champ binaire (initialement une structure de codage produite) au point d'application, cet objet est appliqué et le processus terminé. Sinon la classe au point d'application est "étendue" par déréférencement. Cette expansion par déréférencement continuera jusqu'à ce qu'un objet de codage soit trouvé ou qu'une classe primitive soit atteinte. Si la classe au point d'application est un constructeur de codage et s'il y a un objet de codage pour ce constructeur de codage (**#CHOICE**, **#SEQUENCE**, **#SEQUENCE-OF**, etc.) dans ce cas cet objet est appliqué et le point d'application passe alors à chaque composant (en tant qu'activité parallèle).

9.24.4 Dans un cas plus complexe, il peut y avoir une classe **#OPTIONAL** suivant une classe de composant (et une classe **#TAG** la précédant). Le point d'application passe d'abord à la classe **#OPTIONAL** et l'objet de codage pour cette classe peut remplacer le composant (voir § 9.16.9). Puis le point d'application passe à l'étiquette et finalement au composant lui-même.

9.25 Codages conditionnels

9.25.1 Mention a déjà été faite de la classe de codage **#TRANSFORM** en tant que moyen d'effectuer des opérations arithmétiques simples sur des valeurs entières (voir § 9.17.3). Cette classe de codage joue, cependant, un rôle plus fondamental dans la spécification de codages pour certaines classe primitives. En général, la spécification de codages pour n'importe lequel des types ASN.1 intégrés est un processus à deux ou trois étapes, utilisant des objets de codage de classe **#TRANSFORM** et (par exemple) de classe **#CONDITIONAL-INT** ou **#CONDITIONAL-REPETITION**.

9.25.2 Les classes de codage **#TRANSFORM**, **#CONDITIONAL-INT**, et **#CONDITIONAL-REPETITION** sont restreintes dans leur usage. Des objets de codage ne peuvent être définis pour ces classes qu'au moyen de la syntaxe de l'article 24 et des § 23.7 et 23.13 selon la classe ou par définition non ECN d'un objet de codage. Ils ne peuvent être utilisés que dans la définition d'autres objets de codage. Ils ne peuvent pas apparaître dans des ensembles d'objets de codage ou être appliqués directement pour coder des champs de structures de codage (voir § 18.1.7).

9.25.3 La spécification de codage pour les classes de codage de la catégorie des entiers s'effectue comme suit: les codages (de la classe de codage **#CONDITIONAL-INT**) sont définis pour une **condition aux limites** particulière, spécifiant la taille du conteneur (et la façon dont il est délimité), la transformée de l'entier en bits (au moyen d'un codage de complément à deux ou d'entiers positifs) et la façon dont ces bits s'adaptent dans le conteneur. (Un exemple de condition aux limites est l'existence d'une limite supérieure et d'une limite inférieure non négative.) C'est ce qui est appelé un **codage conditionnel**. Le codage de la classe de la catégorie des entiers est défini comme une liste de ces codages conditionnels, le codage réel appliqué dans toute circonstance donnée étant celui qui est le premier dans la liste dont la condition aux limites est satisfaite. (Voir § D.1.5.4 à titre d'exemple.)

9.25.4 La spécification de codage pour les classes de codage de la catégorie des répétitions utilise la classe de codage **#CONDITIONAL-REPETITION** qui définit la façon dont l'espace de codage pour les items répétés est délimité et la façon dont les codages répétés doivent y être placés, pour une **condition d'étendue** donnée, afin de produire de nouveau un codage conditionnel. Comme dans le cas du codage de classes de la catégorie des entiers, le codage final est défini comme une liste ordonnée de codages conditionnels.

9.25.5 La spécification de codage pour les classes de codage de la catégorie des chaînes d'octets s'effectue comme suit. Tout d'abord, des objets de codage de classe **#TRANSFORM** sont définis pour mapper un seul octet sur une chaîne de bits autodélimitatrice. Ensuite, un ou plusieurs objets de codage de classe **#CONDITIONAL-REPETITION** (pour des conditions spécifiques de taille étendue) sont définis de façon à prendre chacune des chaînes de bits (transformées à partir d'un octet dans la chaîne d'octets) et à les concaténer dans un conteneur délimité (la définition de tels objets de codage n'est pas spécifique au codage de la classe **#OCTETS**). Le codage final de la classe de la catégorie des chaînes

d'octets est défini comme une liste ordonnée d'objets de codage de classe **#CONDITIONAL-REPETITION**. (Voir § D.1.8.2 à titre d'exemple.)

9.25.6 Les spécifications de codage pour les classes de codage de la catégorie des chaînes de bits s'effectuent comme suit. Tout d'abord, des objets de codage de classe **#TRANSFORM** sont définis afin de mapper un bit isolé sur une chaîne de bits, comme dans le codage d'un entier pour le transformer en bits, mais dans ce cas le mappage du bit doit s'effectuer vers une chaîne autodélimitatrice. En deuxième lieu, un ou plusieurs objets de codage de classe **#CONDITIONAL-REPETITION** sont définis pour la répétition des bits (il pourrait s'agir des mêmes objets de codage que ceux qui ont été définis pour utilisation avec une classe de codage dans les catégories de répétition ou de chaîne d'octets). Finalement, le codage de la classe de la catégorie des chaînes de bits est défini comme une liste ordonnée d'objets de codage de classe **#CONDITIONAL-REPETITION**. (Voir § D.1.7.3 à titre d'exemple.)

9.25.7 Les spécifications de codage pour les classes de codage de la catégorie des chaînes de caractères s'effectue comme suit. Tout d'abord, des objets de codage de classe **#TRANSFORM** sont définis afin de mapper un caractère isolé sur une chaîne de bits autodélimitatrice au moyen de plusieurs mécanismes possibles afin de définir le codage du caractère et au moyen de l'effective contrainte d'alphabet permis si elle est disponible. En deuxième lieu, un ou plusieurs objets de codage de classe **#CONDITIONAL-REPETITION** sont définis et finalement le codage de la classe de la catégorie des chaînes de caractères est défini comme une liste ordonnée de ces chaînes. (Voir § D.1.9.2 à titre d'exemple.)

9.26 Modifications apportées aux Recommandations | Normes internationales ASN.1

9.26.1 La présente Recommandation | Norme internationale fait référence à d'autres Recommandations | Normes internationales ASN.1 afin de définir sa notation sans répétition. Afin que de telles références soient correctes, la sémantique de la notation (par exemple la clause d'importation, le paramétrage et la définition d'objet informationnel) doit être étendue à la reconnaissance des noms de référence des classes de codage, des objets de codage et des autres éléments qui font partie de la notation ECN.

9.26.2 Il est également nécessaire d'étendre la notation de classe d'objets informationnels afin de permettre des champs qui sont des listes ordonnées de valeurs ou d'objets, et non seulement des ensembles d'objets non ordonnés. Cela permettra d'utiliser cette notation dans la définition de la syntaxe ECN pour la définition d'objets de codage de certaines classes.

9.26.3 Finalement, les règles de paramétrage sont allégées afin de permettre d'utiliser un paramètre fictif d'une référence à un objet de codage (attribué par une instruction d'attribution) en tant que paramètre réel de la référence de classe de codage qui gouverne la notation définissant le nom de référence de l'objet de codage. En particulier, une classe de codage paramétrée peut être utilisée comme gouverneur dans une instruction d'attribution à un objet de codage (voir § C.2/8.4), le paramètre réel étant un paramètre fictif de l'objet de codage qui est défini.

9.26.4 Ces modifications à d'autres Recommandations | Normes internationales ASN.1 sont spécifiées dans les Annexes A à C, et ne sont reproduites qu'aux fins de la présente Recommandation | Norme internationale.

10 Identification des classes de codage, objets de codage et ensembles d'objets de codage

10.1 Un grand nombre des productions figurant dans la présente Recommandation | Norme internationale nécessitent l'identification d'une classe de codage, d'un objet de codage, ou d'un ensemble d'objets de codage.

10.2 Pour chacune de ces productions, il y a cinq façons d'effectuer l'identification:

a) au moyen d'un nom de référence simple;

b) au moyen d'un nom de référence intégré (non applicable aux objets de codage, car il n'y a pas d'objets de codage intégrés);

c) au moyen d'une référence externe (également appelée *nom entièrement qualifié*);

d) au moyen d'une référence paramétrée;

e) par définition en ligne.

NOTE – La forme de référence paramétrée peut être utilisée avec un nom de référence simple ou avec une référence externe (voir § C.3).

10.3 Il y a des productions (ou des items lexicaux) pour tous ces moyens d'identification. Il y a également des productions qui permettent plusieurs options. Ces noms d'items lexicaux ou de production sont utilisés où il convient dans d'autres productions et sont définis dans la suite du présent paragraphe.

10.4 Les items lexicaux pour utilisation d'un nom de référence simple sont:

encoding class	**"encodingclassreference"** (voir § 8.3)
encoding object	**"encodingobjectreference"** (voir § 8.1)
encoding object set	**"encodingobjectsetreference"** (voir § 8.2)

10.4.1 Une référence "encodingclassreference" est un nom qui est soit:

a) attribué à une classe de codage par une attribution "EncodingClassAssignment" (voir article 16); ou qui est

b) importé dans un module EDM à partir d'un autre module EDM duquel il a été exporté; ou qui est

c) importé comme nom d'une structure de codage produite implicitement à partir d'un module ASN.1 (voir § 14.11); ou qui est

d) produit par une clause de renommage dans le module EDM (Voir article 15).

NOTE – Seules les classes qui sont des structures de codage produites peuvent être importées dans un module ELM (voir § 12.1.8).

10.4.2 Une référence "encodingclassreference" ne doit pas être importée à partir d'un module EDM (comme spécifié dans le § 10.4.1) à moins:

a) qu'elle ne soit définie ou importée dans le module cité en référence, et que ce module n'ait pas de clause d'exportation; ou

NOTE 1 – Si le module cité en référence n'a pas de clause d'exportation, cela équivaut à exporter tout.

b) qu'elle ne soit définie ou importée dans le module cité en référence, et apparaisse en tant que symbole dans la clause d'exportation de ce module; ou

c) qu'elle ne soit un des noms de référence explicitement produits par une clause de renommage dans le module à partir duquel elle est importée.

NOTE 2 – Les structures de codage produites implicitement ne peuvent être importées qu'à partir du module ASN.1 qui les produit.

10.4.3 Une référence de structure de codage produite implicitement n'apparaît jamais dans la clause d'exportation d'un quelconque module ASN.1, mais peut toujours être importée à partir de tout module ASN.1 dans lequel le type correspondant est défini et exporté.

10.4.4 Une référence de structure de codage produite explicitement (qui est automatiquement exportée par la clause de renommage qui la produit) ne doit pas apparaître dans la clause d'exportation du module EDM dans lequel elle est produite, mais toute utilisation de cette référence dans un autre module EDM ou dans le module ELM nécessite son importation à partir de ce module EDM.

10.4.5 Une référence "encodingobjectreference" est un nom qui est soit:

a) attribué à un objet de codage par une attribution "EncodingObjectAssignment" (voir article 17) dans un module EDM; ou qui est

b) importé dans un module EDM ou ELM à partir d'un autre module EDM dans lequel il est soit attribué à un objet de codage soit importé.

10.4.6 Une référence "encodingobjectreference" ne doit pas être importée à partir d'un module EDM si le module cité en référence a une clause d'exportation et si la référence "encodingobjectreference" n'apparaît pas en tant que symbole dans cette clause d'exportation.

NOTE – Si le module cité en référence n'a pas de clause d'exportation, cela équivaut à exporter tout.

10.4.7 Une référence "encodingobjectsetreference" est un nom qui est soit:

a) attribué à un ensemble d'objets de codage dans une attribution "EncodingObjectSetAssignment" (voir article 18) dans un module EDM; ou qui est

b) importé dans un module EDM ou ELM à partir d'un autre module EDM dans lequel il est soit attribué à un ensemble d'objets de codage soit importé.

10.4.8 Une référence "encodingobjectsetreference" ne doit pas être importée à partir d'un module EDM si le module cité en référence a une clause d'exportation et si une référence "encodingobjectsetreference" n'apparaît pas en tant que symbole dans cette clause d'exportation.

NOTE – Si le module cité en référence n'a pas de clause d'exportation, cela équivaut à exporter tout.

10.5 Les productions pour l'utilisation d'un nom de référence intégré sont les suivantes:

encoding class	**"BuiltinEncodingClassReference"** (voir § 16.1.6)
encoding object set	**"BuiltinEncodingObjectSetReference"** (voir § 18.2.1)

10.6 Les productions pour l'utilisation d'un nom de référence externe sont les suivantes:

```
ExternalEncodingClassReference ::=
        modulereference "." encodingclassreference      |
        modulereference "." BuiltinEncodingClassReference

ExternalEncodingObjectReference ::=
        modulereference "." encodingobjectreference

ExternalEncodingObjectSetReference ::=
        modulereference "." encodingobjectsetreference
```

10.6.1 La référence "modulereference" est définie dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 11.5. Elle identifie un module qui est cité en référence dans la liste d'importations du module EDM ou ELM.

10.6.2 L'option "ExternalEncodingClassReference qui contient une référence "BuiltinEncodingClassReference" doit être utilisée si et seulement s'il y a une structure de codage produite (dont le nom est le même que celui d'une référence "BuiltinEncodingClassReference") dans le corps d'un module EDM qui est soit:

a) défini implicitement dans le module ASN.1 cité en référence par la référence "modulereference" (voir § 11.4.1); ou

b) importé dans un autre module EDM cité en référence par la référence "modulereference" et exporté à partir de ce module; ou

c) produit dans une clause de renommage d'un autre module EDM cité en référence par la référence "modulereference"; ou

d) produit dans ce module EDM dans une clause de renommage, auquel cas la référence "modulereference" doit faire référence à ce module EDM.

NOTE – Le nom "BuiltinEncodingClassReference" peut apparaître en tant que "Symbol" dans la clause d'importation (voir § A.1).

10.6.3 Les productions définies au § 10.6 (sauf comme spécifié dans le § 10.6.2) doivent être utilisées si et seulement si le nom de référence simple correspondant a été importé à partir du module identifié par la référence "modulereference", et si soit:

a) des noms de référence identiques ont été importés à partir de modules différents, ou ont été produits dans une clause de renommage de ce module EDM, ou ont été aussi bien importés que produits; ou

b) le nom de référence simple est une référence "BuiltinEncodingClassReference" (voir § 10.5); ou

c) ces deux conditions sont vérifiées.

10.7 Une référence paramétrée est un nom de référence défini dans une attribution "ParameterizedAssignment" (voir § C.1) et fourni avec un paramètre réel conformément à la syntaxe du § C.3. Les productions en cause sont les suivantes:

encoding classes	**"ParameterizedEncodingClassAssignment"** (voir § C.1) **"ParameterizedEncodingClass"** (voir § C.3)
encoding objects	**"ParameterizedEncodingObjectAssignment"** (voir § C.1) **"ParameterizedEncodingObject"** (voir § C.3)
encoding object sets	**"ParameterizedEncodingObjectSetAssignment"** (voir § C.1) **"ParameterizedEncodingObjectSet"** (voir § C.3)

10.8 Les productions qui permettent toutes les formes d'identification sont les suivantes:

encoding classes	**"EncodingClass"** (voir § 16.1.5)
encoding objects	**"EncodingObject"** (voir § 17.1.5)
encoding object sets	**"EncodingObjectSet"** (voir § 18.1)

10.9 Les productions qui permettent toutes les formes sauf la définition en ligne sont les suivantes:

encoding classes	**"DefinedEncodingClass" and "DefinedOrBuiltinEncodingClass"**
encoding objects	**"DefinedEncodingObject"**
encoding object sets	**"DefinedEncodingObjectSet" and "DefinedOrBuiltinEncodingObjectSet"**

sauf que les classes de codage intégrées et les ensembles d'objets de codage intégrés ne sont pas permis par "DefinedEncodingClass" et "DefinedEncodingObjectSet".

NOTE – Une autre production "SimpleDefinedEncodingClass" est également utilisée. Elle est définie au § C.3 et autorise seulement "encodingclassreference" et "ExternalEncodingClassReference".

10.9.1 Les classes "DefinedEncodingClass" et "DefinedOrBuiltinEncodingClass" sont les suivantes:

```
DefinedEncodingClass ::=
        encodingclassreference
    |   ExternalEncodingClassReference
    |   ParameterizedEncodingClass

DefinedOrBuiltinEncodingClass ::=
        DefinedEncodingClass
    |   BuiltinEncodingClassReference
```

10.9.2 L'objet "DefinedEncodingObject" est le suivant:

```
DefinedEncodingObject ::=
        encodingobjectreference
    |   ExternalEncodingObjectReference
    |   ParameterizedEncodingObject
```

10.9.3 Les ensembles "DefinedEncodingObjectSet" et "DefinedOrBuiltinEncodingObjectSet" sont:

```
DefinedEncodingObjectSet ::=
        encodingobjectsetreference
    |   ExternalEncodingObjectSetReference
    |   ParameterizedEncodingObjectSet

DefinedOrBuiltinEncodingObjectSet ::=
        DefinedEncodingObjectSet
    |   BuiltinEncodingObjectSetReference
```

11 Codage des types ASN.1

11.1 Généralités

11.1.1 Pour tous les types ASN.1, il y a une structure de codage produite implicitement correspondante. Cette structure de codage est produite implicitement pour chaque attribution de type ASN.1 et est automatiquement exportée à partir du module ASN.1 qui contient cette attribution de type. (Elle doit, cependant, être importée dans un module EDM pour pouvoir être utilisée.) Le nom de la structure de codage correspondante est celui du type précédé par un caractère "**#**". Cette structure de codage définit une classe de codage et est appelée **structure de codage produite implicitement**.

11.1.2 Il peut y avoir également une ou plusieurs **structures de codage produites explicitement**, qui sont produites dans un module EDM utilisant une clause de renommage.

11.1.3 Le codage d'un type ASN.1 est formellement défini comme le résultat de codages appliqués à précisément une des structures de codage (implicitement ou explicitement) produites à partir du type ASN.1. Les codages sont appliqués par instructions dans le module ELM (voir article 12), au moyen d'objets de codage choisis dans un ensemble d'objets de codage combinés. Un module ELM doit appliquer ces codages à au plus une des structures de codage produites correspondant à tout type ASN.1 donné.

11.1.4 La structure de codage produite implicitement est définie d'abord par simplification et expansion de la notation ASN.1 (comme spécifié dans le § 11.3), puis par du mappage des types ASN.1, des constructeurs de type et des noms de composant dans les classes de codage intégrées, les constructeurs de codage et les noms de champ de structure de codage correspondants.

11.1.5 Une structure de codage produite explicitement est définie par l'application de modifications spécifiées à la structure de codage produite implicitement, au moyen d'une clause de renommage.

11.1.6 Chaque champ d'une structure de codage produite contient son association aux valeurs abstraites du type correspondant et aux informations relatives aux contraintes issues d'une définition de type ASN.1 (voir § 11.4.2). Les codages des valeurs abstraites de la structure de codage produite sont définis comme étant les codages pour les valeurs abstraites correspondantes du type ASN.1 original.

11.1.7 Cet article 11 spécifie:

a) les classes de codage intégrées qui sont utilisées afin de définir les structures de codage produites implicitement correspondant à des types ASN.1 (voir § 11.2).

NOTE – Le § 16.1.14 spécifie des classes additionnelles qui sont utilisées dans la définition de structures de codage définies par l'usager.

b) les transformations de la syntaxe ASN.1 (simplification et expansion) avant que la structure produite implicitement soit créée (voir § 11.3);

c) la structure de codage produite implicitement pour tout type ASN.1 (voir § 11.4).

11.2 Classes de codage intégrées utilisées pour les structures de codage produites implicitement

11.2.1 Les classes de codage utilisées pour les structures de codage produites implicitement, et les types ASN.1 ou constructeurs auxquels elles correspondent sont énumérés dans le Tableau 2 ci-dessous.

11.2.2 La colonne 1 indique la notation ASN.1 qui est remplacée par une classe de codage dans la structure de codage produite implicitement. La colonne 2 indique la classe de codage qui remplace la notation de la colonne 1. La colonne 3 indique la classe primitive dont la classe de colonne 2 est issue.

Tableau 2 – Classes de codage pour notation ASN.1

Notation ASN.1	Classe de codage	Classe primitive
BIT STRING	**#BIT-STRING**	**#BITS**
BOOLEAN	**#BOOLEAN**	**#BOOL**
CHARACTER STRING	**#CHARACTER-STRING**	*Définis au moyen de* ***#SEQUENCE***
CHOICE	**#CHOICE**	**#ALTERNATIVES**
EMBEDDED PDV	**#EMBEDDED-PDV**	*Définis au moyen de* ***#SEQUENCE***
ENUMERATED	**#ENUMERATED**	**#INT**
EXTERNAL	**#EXTERNAL**	*Définis au moyen de* ***#SEQUENCE***
INTEGER	**#INTEGER**	**#INT**
NULL	**#NULL**	**#NUL**
OBJECT IDENTIFIER	**#OBJECT-IDENTIFIER**	**#OBJECT-IDENTIFIER**
OCTET STRING	**#OCTET-STRING**	**#OCTETS**
notation de type ouvert	**#OPEN-TYPE**	**#OPEN-TYPE**
OPTIONAL	**#OPTIONAL**	**#OPTIONAL**
REAL	**#REAL**	**#REAL**
RELATIVE-OID	**#RELATIVE-OID**	**#OBJECT-IDENTIFIER**
SEQUENCE	**#SEQUENCE**	**#CONCATENATION**
SEQUENCE OF	**#SEQUENCE-OF**	**#REPETITION**
SET	**#SET**	**#CONCATENATION**
SET OF	**#SET-OF**	**#REPETITION**
GeneralizedTime	**#GeneralizedTime**	**#CHARS**
UTCTime	**#UTCTime**	**#CHARS**
ObjectDescriptor	**#ObjectDescriptor**	**#CHARS**
BMPString	**#BMPString**	**#CHARS**
GeneralString	**#GeneralString**	**#CHARS**
GraphicString	**#GraphicString**	**#CHARS**
IA5String	**#IA5String**	**#CHARS**
NumericString	**#NumericString**	**#CHARS**
PrintableString	**#PrintableString**	**#CHARS**
TeletexString	**#TeletexString**	**#CHARS**
UniversalString	**#UniversalString**	**#CHARS**
UTF8String	**#UTF8String**	**#CHARS**
VideotexString	**#VideotexString**	**#CHARS**
VisibleString	**#VisibleString**	**#CHARS**
Notation par étiquettes présentes textuellement	**#TAG**	**#TAG**

11.3 Simplification et expansion de la notation ASN.1 aux fins du codage

11.3.1 La notation ECN part du principe que certaines constructions syntaxiques ASN.1 ont été étendues (ou réduites) dans des constructions équivalentes ou plus simples.

NOTE – Les types définis par les constructions plus simples sont capables d'acheminer le même ensemble de valeurs abstraites que les structures syntaxiques ASN.1 originales et ces valeurs abstraites sont mappées sur les constructions plus simples.

11.3.2 L'expansion ou la simplification des productions syntaxiques ASN.1 est soit:

a) entièrement définie dans le § 11.3.4 ci-dessous; ou

b) citée en référence dans les paragraphes tels que "voir § 11.3.2 b" et entièrement définie dans la Rec. UIT-T X.680 | ISO/CEI 8824-1 (y compris l'Annexe F) avec tous les amendements et corrigenda techniques publiés; ou

c) citée en référence dans les paragraphes tels que "voir § 11.3.2 c" et entièrement définie dans la Rec. UIT-T X.681 | ISO/CEI 8824-2 avec tous les amendements et corrigenda techniques publiés;

d) citée en référence dans les paragraphes tels que "voir § 11.3.2 d" et entièrement définie dans la Rec. UIT-T X.683 | ISO/CEI 8824-4 avec tous les amendements et corrigenda techniques publiés.

11.3.3 Les constructions syntaxiques ASN.1 supprimées par les expansions et simplifications ci-dessous ne seront plus citées en référence dans la présente Recommandation | Norme internationale.

11.3.4 Les expansions et simplifications doivent être appliquées à tous les modules ASN.1:

11.3.4.1 Les transformations suivantes ne sont pas récursives et ne sont donc appliquées qu'une fois:

a) toutes les constructions "ValueSetTypeAssignment" doivent être remplacées par leur construction équivalente "TypeAssignment" avec des contraintes de sous-type (voir § 11.3.2 b);

b) la construction ASN.1 **INSTANCE OF** doit être étendue dans son type de séquence équivalent (voir § 11.3.2 c);

c) la construction "TypeFromObject" doit être remplacée par le type qui est cité en référence (voir § 11.3.2 c);

d) la construction "ValueSetFromObjects" doit être remplacée par le type qui est cité en référence (voir § 11.3.2 c);

e) lorsqu'une instance de notation par étiquettes ASN.1 est textuellement suivie par une ou plusieurs autres instances de notation par étiquettes ASN.1, la deuxième instance et les instances suivantes de notation par étiquettes sont rejetées.

NOTE – Cela est semblable aux règles pour l'étiquetage implicite en notation ASN.1, mais cela s'applique à tous les environnements d'étiquetage. L'étiquetage multiple du même type est encore possible au moyen de noms de référence de type.

11.3.4.2 Les transformations doivent être appliquées de façon récursive dans l'ordre spécifié, jusqu'à ce qu'un point fixe soit atteint:

a) tout paramétrage ASN.1 doit être entièrement résolu par le remplacement des paramètres réels par des paramètres fictifs (voir § 11.3.2 d);

NOTE – Cela implique que si la notation de type ASN.1 contient une instanciation d'un type ASN.1 paramétré, cette instanciation devient une définition en ligne.

b) toutes les structures "ComponentsOf" doivent être étendues à leur forme complète (voir § 11.3.2 b);

c) toutes les utilisations de "SelectionType" doivent être résolues (voir § 11.3.2 b).

11.3.4.3 Les transformations doivent ensuite être appliquées:

a) les listes de numéros d'éléments nommés dans une définition de type entier doivent être supprimées. Les numéros d'éléments nommés ne sont pas visibles par la notation ECN. La notation ECN voit une seule classe **#INTEGER** (éventuellement avec des limites comme spécifié dans le § 11.3.4.3 c);

b) les listes de bits nommés dans des définitions de chaîne de bits doivent être supprimées. Les bits nommés ne sont pas visibles par la notation ECN;

c) toutes les notations de contrainte non visibles par les règles PER, sauf la contrainte de contenu, doivent être rejetées. Les contraintes visibles par les règles PER doivent être résolues afin d'offrir les valeurs suivantes, qui peuvent être citées en référence dans la définition de règles de codage:

i) une limite supérieure des entiers et énumérations;

ii) une limite inférieure des entiers et énumérations;

iii) les contraintes de longueur effective et contraintes d'alphabet effectivement permis par les règles PER (voir Rec. UIT-T X.691 | ISO/CEI 8825-2, § 9.3);

d) s'il y a une contrainte de contenu avec une construction **CONTAINING**, l'existence de la contrainte de contenu, son type de contenu et la présence ou l'absence d'une clause **ENCODED BY** deviennent des propriétés associées aux valeurs abstraites d'un tel type de chaîne contrainte d'octets ou de bits, et la contrainte doit ensuite être rejetée. S'il y a une contrainte de contenu sans construction **CONTAINING**, cette contrainte n'est pas visible par la notation ECN et doit être rejetée;

NOTE – Lors de la spécification de codages pour valeurs avec une contrainte de contenu associée, un ensemble d'objets de codage combinés distinct peut être fourni afin de coder le type de contenu. Il est possible de spécifier que cet ensemble doit avoir ou ne pas avoir priorité sur toute construction "**ENCODED BY**" éventuellement présente, en tant qu'option du concepteur (voir § 11.3 et 13.2).

e) tout étiquetage qui n'est pas textuellement présent dans la notation ASN.1 doit être ignoré lors du mappage sur des structures de codage, mais (afin de modéliser les codages BER et les procédures PER), la liste complète d'étiquettes d'un type devient une propriété du champ de la structure de codage sur laquelle la valeur correspondante est mappée;

f) la notation par étiquettes présente textuellement possède la classe de l'étiquette supprimée (voir également § 11.3.4.1 e);

g) la construction "**DEFAULT** Value" doit être remplacée par "**OPTIONAL-ENCODING #OPTIONAL**" et la valeur par défaut est associée au champ de la structure sur laquelle le composant ASN.1 est mappé;

h) le mot "**OPTIONAL**" doit être remplacé par "**OPTIONAL-ENCODING #OPTIONAL**";

i) le mot "**T61String**" doit être remplacé par **#TeletexString**;

j) le mot "**ISO646String**" doit être remplacé par **#VisibleString**.

11.3.4.4 Finalement, les transformations suivantes doivent donc être appliquées:

a) l'attribution automatique de valeurs aux énumérations (si applicable) doit être effectuée. La forme syntaxique **ENUMERATED** doit être remplacée par la classe de codage **#ENUMERATED** avec fixation d'une limite supérieure et d'une limite inférieure (voir § 11.3.4.3 c);

NOTE 1 – La classe **#ENUMERATED** déréférence à la classe **#INT** (voir § 11.2.2) et les énumérations sont mappées sur des valeurs entières bornées de la classe. Les noms réels des énumérations ne sont pas visibles en notation ECN.

b) toutes les instances du type "ObjectClassFieldType" (voir Rec. UIT-T X.681 | ISO/CEI 8824-2, article 14) qui se rapportent à un champ de type, à un champ de valeur variable de type, ou à un champ d'ensemble de valeurs variables de type doivent être remplacées par la classe de codage **#OPEN-TYPE** (voir § 11.3.2 c);

c) les séquences de marqueurs d'extensibilité et de numéros de version entre crochets, ainsi que les constructions d'ensemble et de choix sont supprimées, mais (afin de modéliser les codages BER et les procédures PER) l'identification d'un composant en tant que partie de la racine ou de la version 1, version 2, etc., devient une propriété du composant et l'existence du marqueur d'extensibilité devient une propriété de la classe sur laquelle la construction est mappée;

d) le marqueur d'extensibilité contenu dans les contraintes est supprimé, mais l'existence du marqueur d'extensibilité devient une propriété de la classe et le fait qu'une valeur abstraite soit dans la racine ou dans une extension devient une propriété de la valeur abstraite.

NOTE 2 – Les propriétés citées en référence aux points c) et d) ci-dessus ne peuvent être interrogées que par définition non ECN d'objets de codage dans cette version de la présente Recommandation | Norme internationale. Une prise en charge complète d'extensibilité sera sans doute assurée dans une version ultérieure de la présente Recommandation | Norme internationale.

11.3.5 Avec ces transformations, toutes les constructions associées aux types ASN.1 ont des classes de codage correspondantes, énumérées dans le Tableau 2. La structure de codage produite implicitement doit être construite par mappage des constructions associées aux types ASN.1 de la colonne 1 sur les classes contenues dans la colonne 2 du Tableau 2 (comme spécifié dans le § 11.4).

11.4 La structure de codage produite implicitement

11.4.1 Il y a une structure produite implicitement pour chaque définition de type ASN.1 avec un nom construit à partir du nom de référence de type ASN.1 par préfixation d'un caractère "**#**". Lorsqu'un nom entièrement qualifié est requis pour une structure de codage produite implicitement, ce nom entièrement qualifié doit contenir l'identificateur "ModuleIdentifier" du module ASN.1 contenant la définition de type. (Un exemple de structure produite implicitement est donné au § D.1.9.2.)

NOTE – Une structure produite implicitement est produite et exportée pour chaque type ASN.1 dans un module ASN.1 que ce type soit ou non énuméré dans la clause **EXPORTS**.

11.4.2 La structure de codage produite implicitement a la même structure que la définition de type ASN.1, avec les conditions suivantes:

a) les identificateurs de composant ASN.1 sont mappés sur des noms de champ de structure de codage;

b) la notation ASN.1 de la colonne 1 du Tableau 2 est mappée sur les classes de codage intégrées dans la colonne 2 du Tableau 2;

NOTE 1 – La première étiquette textuellement présente est mappée sur une construction "**[#TAG]**" contenue dans la structure produite implicitement. La structure produite implicitement ne contient aucune construction "**[#TAG]**" pour les étiquettes textuellement présentes qui suivent.

c) les types ASN.1 "DefinedType" sont mappés sur un nom de classe de codage issu de la référence de type par l'adjonction d'un caractère "**#**". Si un type est importé dans le module ASN.1, toute référence "ExternalEncodingClassReference" à la classe correspondante dans une structure produite implicitement doit faire référence au module ASN.1 qui contient la définition du type cité en référence;

NOTE 2 – Si la classe résultante est le nom d'une classe de codage intégrée, toutes références à cette classe dans la clause de renommage ou dans le module ELM utiliseront la notation "ExternalEncodingClassReference".

d) les valeurs abstraites sont mappées à partir d'un champ de définition de type sur le champ correspondant de la structure de codage;

e) les limites supérieures et inférieures des types entiers et énumérés, ainsi que toutes les contraintes effectives de longueur et d'alphabet permis (voir Rec. UIT-T X.691 | ISO/CEI 8825-2, § 9.3) sont mappées à partir d'une définition de type sur le champ correspondant de la structure de codage;

f) le numéro d'étiquette de la première étiquette textuellement présente est mappé sur la classe **#TAG**.

11.4.3 Trois autres structures produites implicitement sont créées et exportées à partir de tous les modules ASN.1. Ces structures sont nommées **#CHARACTER-STRING**, **#EMBEDDED-PDV** et **#EXTERNAL** et les structures qu'elles déréférencent sont les structures produites implicitement qui correspondent aux types associés pour **CHARACTER STRING**, **EMBEDDED-PDV** et **#EXTERNAL**, spécifiées dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 40.5, 33.5 et 34.5 respectivement.

11.4.4 Toutes les structures de codage produites implicitement peuvent être codées par les ensembles d'objets de codage intégrés (voir §18.2). Elles produiront les codages spécifiés par la Recommandation | Norme internationale correspondante pour leur application à des types ASN.1.

12 Le module de lien de codage (ELM)

NOTE – Il y a deux productions de niveau supérieur en notation ECN: la production "ELMDefinition" spécifiée dans ce paragraphe et la production "EDMDefinition" spécifiée dans l'article 14. Ces paragraphes spécifient la syntaxe définissant respectivement les modules ELM et EDM.

12.1 Structure du module ELM

12.1.1 La notation "ELMDefinition" est la suivante:

```
ELMDefinition ::=
        ModuleIdentifier
        LINK-DEFINITIONS
        "::="
        BEGIN
        ELMModuleBody
        END
```

12.1.2 Dans toute application donnée de la notation ECN, il doit y avoir précisément un seul module ELM qui détermine le codage de tous les messages utilisés dans cette application.

NOTE – Les types ASN.1 définissant des "messages" sont souvent désignés comme étant des "types de niveau supérieur".

12.1.3 La notation "ModuleIdentifier" (et sa sémantique) est définie dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 12.1.

12.1.4 La notation "ModuleIdentifier" offre une identification univoque d'un quelconque module dans l'ensemble de tous les modules ASN.1, ELM et EDM.

12.1.5 La notation "ELMModuleBody" est la suivante:

```
ELMModuleBody ::=
    Imports ?
    EncodingApplicationList

EncodingApplicationList ::=
        EncodingApplication
        EncodingApplicationList ?
```

12.1.6 La production "Imports" (et sa sémantique) est définie dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 12.1, 12.15, et 12.16, telle que modifiée par le § A.1 de la présente Recommandation | Norme internationale.

12.1.7 Tous les noms de référence utilisés dans la notation "ELMModuleBody" doivent être importés dans le module ELM.

NOTE – Il s'agit là d'une exigence plus sévère que celle qui est imposée aux modules ASN.1. Dans les modules ASN.1, des références externes peuvent être utilisées pour des types et des valeurs qui n'ont pas été importés. Dans un module ELM (et dans un module EDM), des références externes ne peuvent être utilisées que pour des classes de codage qui ont été citées en référence dans une clause d'importation. L'objectif des références externes ne consiste qu'à résoudre des ambiguïtés entre noms importés et noms intégrés, ou entre deux noms identiques importés à partir de modules différents.

12.1.8 La clause "Imports" met à disposition à l'intérieur du module ELM:

a) des structures de codage produites implicitement à partir d'un module ASN.1;

b) des structures de codage produites explicitement à partir d'un module EDM;

NOTE – Lorsqu'un module ELM importe une structure de codage produite explicitement à partir d'un module EDM, la clause de renommage contenue dans d'autres modules EDM n'a aucun effet sur le codage de cette structure (voir § 15.2.4).

c) les objets et ensembles d'objets de codage à partir d'un module EDM.

12.1.9 La production "EncodingApplicationList" est requise à contenir au moins un "EncodingApplication", que la seule fonction d'un module ELM consiste à appliquer des codages.

12.2 Types de codage

12.2.1 Une notation "EncodingApplication" est la suivante:

```
EncodingApplication ::=
    ENCODE
    SimpleDefinedEncodingClass "," +
    CombinedEncodings
```

12.2.2 Une notation "EncodingApplication" définit le codage des types ASN.1 correspondant aux classes "SimpleDefinedEncodingClass" qui doivent être des structures de codage produites. Le codage des types est spécifié par les codages "CombinedEncodings" appliqués à des structures de codage produites comme spécifié au § 13.2.

NOTE – Il sera courant qu'un module ELM code un type isolé d'un seul module, mais lorsque de multiples types sont codés, les vendeurs d'utilitaires ECN peuvent (mais sans obligation) partir du principe que ce module identifie implicitement des types de niveau supérieur nécessitant une prise en charge dans des structures produites de données.

12.2.3 Les codages appliqués à une structure de codage produite correspondant à un type ASN.1 défini dans un certain module ASN.1 sont associés seulement à l'utilisation de ce type en tant que messages applicatifs. Ils n'ont aucune incidence sur le codage de ce type lorsque celui-ci est cité en référence par d'autres types ou lorsqu'il est exporté à partir de ce module ASN.1 et importé dans un autre module ASN.1.

12.2.4 Le codage du type dans une contrainte de contenu est spécifié par l'objet de codage appliqué à la classe contenante dans la catégorie des chaînes d'octets ou de bits, et peut être tout ensemble d'objets de codage combinés, ou peut être l'ensemble d'objets de codage combinés qui a été appliqué à la classe contenante dans la catégorie des chaînes d'octets ou de bits.

12.2.5 Un module ELM ne doit pas appliquer des codages plus d'une fois au même type ASN.1.

NOTE – Les règles d'application des codages (spécifiées dans l'article 13) impliquent qu'une application "EncodingApplication" définisse complètement le codage d'un type à moins qu'il ne contienne une instance d'une contrainte de contenu.

13 Application des codages

13.1 Généralités

13.1.1 Les codages sont appliqués par le module ELM à une structure produite (ou indépendamment à de multiples structures produites) au moyen d'une définition "CombinedEncodings" comme spécifié au § 13.1.3. Le présent paragraphe, ainsi que le § 13.2, spécifie l'application d'une définition "CombinedEncodings" à une structure de codage produite.

13.1.2 Dans le module ELM, l'application vise des structures de codage produites qui ont été identifiées dans la définition "EncodingApplication". Des paragraphes ultérieurs spécifient également l'application des codages à tout ou partie d'une définition arbitraire de la structure de codage. Ce paragraphe est applicable aux deux cas.

13.1.3 La définition "CombinedEncodings" est la suivante:

```
CombinedEncodings ::=
    WITH
    PrimaryEncodings
    CompletionClause ?

CompletionClause ::=
    COMPLETED BY
    SecondaryEncodings

PrimaryEncodings ::= EncodingObjectSet

SecondaryEncodings ::= EncodingObjectSet
```

13.1.4 La notation "EncodingObjectSet" est définie au § 18.1.1.

13.1.5 L'utilisation de la notation "CombinedEncodings" est spécifiée au § 13.2.

13.2 L'ensemble d'objets de codage combinés et son application

13.2.1 Un **ensemble d'objets de codage combinés** est formé à partir de la production "CombinedEncodings" (voir § 13.1.3) comme suit:

13.2.2 S'il n'y a pas de clause "CompletionClause", les codages "PrimaryEncodings" forment l'ensemble d'objets de codage combinés.

13.2.3 Sinon,

a) tous les objets de codage contenus dans les codages "PrimaryEncodings" sont placés dans l'ensemble d'objets de codage combinés, puis

b) chaque objet de codage contenu dans les codages "SecondaryEncodings" est ajouté à l'ensemble d'objets de codage combinés si (et seulement si) il n'y a pas déjà d'objet de codage dans l'ensemble d'objets de codage combinés qui a la même classe de codage (voir § 17.1.7 et 9.23.2).

13.2.4 Après cette construction théorique de l'ensemble d'objets de codage combinés, le codage commence par le nom "encodingclassreference" des structures de codage identifiées dans l'application de codage (voir § 13.1.2 et 17.5).

13.2.5 S'il y a plusieurs applications de codage dans le module ELM, les règles du § 12.2 garantissent que les applications ne se superposent pas. Elles se déroulent indépendamment. De la même façon, l'application des codages à des structures de codage dans les modules EDM (spécifiée au § 13.2.10) se fait toujours sans superposition. Les paragraphes suivants fournissent les règles pour application à une unique structure de codage.

13.2.6 Des objets de codage issus de l'ensemble d'objets de codage combinés sont appliqués à un **point d'application**. Le point d'application est initialement la référence "encodingclassreference" pour une structure de codage produite (lorsque l'application est dans le module ELM, comme spécifié dans le § 13.1.2) ou est un composant d'une structure de codage (lorsque l'application est dans un module EDM, comme spécifié au § 17.5).

13.2.7 Toute classe de codage dans les catégories des options, des concaténations et des répétitions (voir § 16.1.8, 16.1.9 et 16.1.10) est un constructeur de codage.

13.2.8 Le terme "composant" dans le texte suivant se rapporte à n'importe lequel des points suivants:

a) les options d'un constructeur qui est dans la catégorie des options;

b) le champ suivant un constructeur qui est dans la catégorie des répétitions;

c) les composants d'un constructeur qui est dans la catégorie des concaténations;

d) un type de contenu (un type spécifié dans une contrainte de contenu);

e) le type choisi (dans une instance de communication) pour utilisation avec une classe de la catégorie des types ouverts.

13.2.9 A des étapes ultérieures de ces procédures, le point d'application peut être un des suivants:

a) un nom de classe de codage. Ce nom est complètement codable au moyen de la spécification contenue dans un objet de codage de la même classe (voir § 17.1.7);

b) un constructeur de codage (voir § 16.2.12). Les procédures de construction peuvent être déterminées par la spécification contenue dans un objet de codage de la classe des constructeurs de codage, mais cet objet de codage ne détermine pas le codage des composants. La spécification de l'objet de codage qui est

appliquée peut nécessiter qu'un ou plusieurs des composants du constructeur soient remplacés par d'autres structures (paramétrées) avant que le point d'application passe aux composants;

c) une classe de la catégorie des chaînes de bits ou d'octets qui possède un type contenu en tant que propriété associée aux valeurs (voir § 11.3.4.3 d). Le codage du type contenu dépend de savoir si une construction **ENCODED BY** est présente et de l'application de la spécification de l'objet de codage (voir § 22.11);

d) un composant qui est une classe de codage (éventuellement précédée par une ou plusieurs classes de la catégorie des étiquettes), suivie par une classe de codage dans la catégorie des offres d'options. Les procédures et codages permettant de déterminer la présence ou l'absence sont déterminés par la spécification contenue dans un objet de codage de la classe de la catégorie des offres d'options. Cet objet de codage peut également nécessiter le remplacement de la classe de codage (ainsi que toutes ses classes précédentes dans la catégorie des étiquettes) par une structure (paramétrée) de remplacement avant que cette classe soit codée. Le point d'application passe alors à la première classe de la catégorie des étiquettes (éventuelle), ou au composant, ou à son remplacement;

e) une classe de codage précédée par une classe de codage de la catégorie des étiquettes. Les numéros d'étiquette associés à la classe de la catégorie des étiquettes sont codés au moyen de la spécification contenue dans un objet de codage de la classe de la catégorie des étiquettes et le point d'application passe ensuite à la classe étiquetée;

f) toute autre classe de codage intégrée. Cet élément est complètement codable au moyen de la spécification contenue dans un objet de codage de cette classe.

13.2.10 Le codage s'effectue comme suit:

13.2.10.1 Si l'ensemble d'objets de codage combinés contient un objet de codage de la même classe (voir § 17.1.7) que le point d'application courant, cet objet de codage est appliqué. Cette application peut provoquer le remplacement d'un ou de plusieurs composants de la classe à laquelle le codage est en cours d'application. Si l'ensemble d'objets de codage combinés ne contient pas un tel objet de codage, soit:

a) la classe de codage au point d'application courant est une référence à une autre classe de codage; dans ce cas elle est déréférencée et les procédures du § 13.2.10 sont appliquées de façon récursive; ou

b) la classe de codage au point d'application courant n'est pas une référence à une autre classe de codage; dans ce cas la spécification ECN est en erreur.

13.2.10.2 Si un codage a été appliqué au point d'application à la classe de codage, et qu'il ne soit pas dans la catégorie des offres d'options ou des étiquettes et ne possède pas de composants (voir § 13.2.7), dans ce cas cette application détermine complètement le codage de la classe et met fin à ces procédures.

13.2.10.3 Si un codage a été appliqué au point d'application à une classe de codage qui est dans la catégorie des offres d'options, alors le point d'application passe au composant facultatif (éventuellement étiqueté).

13.2.10.4 Si un codage a été appliqué au point d'application à une classe de codage qui est dans la catégorie des étiquettes, alors le point d'application passe à l'élément étiqueté et les procédures du § 13.2.10 sont appliquées de façon récursive.

13.2.10.5 Si un codage a été appliqué au point d'application à une classe de codage qui possède des composants qui ne sont pas un type contenu, dans ce cas les procédures du § 13.2.10 sont appliquées de façon récursive à chaque composant.

NOTE – Cela implique que l'ensemble courant d'objets de codage combinés est appliqué au type choisi (dans une instance de communication) pour utilisation avec une classe de la catégorie des types ouverts (voir § 13.2.8 e)).

13.2.10.6 Si un codage a été appliqué à une classe de codage au point d'application qui a un composant qui est une classe de la catégorie des chaînes de bits ou d'octets avec un type de contenu associé à la valeur, dans ce cas il y a quatre cas qui peuvent apparaître:

a) la contrainte de contenu contient un **ENCODED BY** et l'objet de codage pour cette classe soit ne contient pas une spécification du codage du type contenu, ou spécifie qu'il ne devrait pas prendre la priorité sur un **ENCODED BY** (voir § 22.11). Dans ce cas la spécification **ENCODED BY** doit être utilisée pour le type contenu et le point d'application passe au type contenu au moyen de cette spécification de codage;

b) la contrainte de contenu contient un **ENCODED BY**, mais l'objet de codage pour cette classe contient une spécification du codage du type contenu et spécifie qu'il doit prendre la priorité sur un **ENCODED BY**. Dans ce cas, la spécification contenue dans l'objet de codage doit être appliquée au type contenu et le point d'application passe au type contenu au moyen de cette spécification de codage;

c) la contrainte de contenu ne contient pas un **ENCODED BY** et l'objet de codage pour cette classe contient une spécification de codage du type contenu. Dans ce cas, la spécification contenue dans l'objet de

codage est appliquée au type contenu et le point d'application passe au type contenu au moyen de cette spécification de codage;

d) la contrainte de contenu ne contient pas un **ENCODED BY** et l'objet de codage pour cette classe ne contient pas de spécification de codage du type contenu. Dans ce cas l'ensemble d'objets de codage combinés appliqué à la classe doit également être appliqué au type de contenu et le point d'application passe au type contenu au moyen de cette spécification de codage.

13.2.10.7 S'il n'y a pas d'objet de codage dans l'ensemble d'objets de codage combinés de la même classe (voir § 17.1.7) que le point d'application courant et que celui-ci soit un nom de référence, dans ce cas l'ensemble est déréférencé et ces procédures sont appliquées de façon récursive à la nouvelle structure de codage.

13.2.10.8 Sinon la spécification ECN est en erreur.

13.2.11 L'algorithme ci-dessus peut être résumé comme suit. L'ensemble d'objets de codage combinés est appliqué de haut en bas. Si au cours de ce processus un nom de référence de structure de codage est rencontré et qu'il y ait un objet dans l'ensemble d'objets de codage combinés qui peut le coder, cet objet détermine son codage. Sinon, le nom de référence est étendu par déréférencement. Si à une étape quelconque un codage est requis (et n'existe pas) pour une classe de codage qui ne peut pas être déréférencée, dans ce cas la spécification ECN est incorrecte et la classe de codage combinée est dite incomplète. Lorsqu'une classe primitive de champ binaire est atteinte, le codage se termine avec le codage de cette classe, sauf que s'il possède un type contenu, le codage s'effectue sur la structure de codage produite correspondant au type contenu. Lorsqu'un type avec composants est atteint, le processus continue par l'application de l'ensemble d'objets de codage combinés à chaque composant indépendamment. Lorsque des étiquettes et des offres d'options sont impliquées, la classe des offres d'options est codée en premier, puis la classe de codage dans la catégorie des étiquettes et finalement l'élément. Lorsque des codages sont appliqués à des classes de constructeur, ces codages peuvent provoquer le remplacement d'un ou de plusieurs composants. Lorsqu'ils sont appliqués à une classe d'offres d'options, ils peuvent provoquer le remplacement de l'élément entier (sauf la classe des offres d'options, mais y compris toute classe de codage dans la catégorie des étiquettes).

13.2.12 Dans le processus de codage, des objets de codage appliqués à des constructeurs de codage (et à des classes de la catégorie des offres d'options) peuvent nécessiter que les objets de codage appliqués à leurs composants offrent des pointeurs d'identification (d'un nom donné) afin de résoudre les options, ou les offres d'options, ou l'ordre dans une concaténation de type ensemble. Si dans ce cas les codages des composants n'offrent pas les pointeurs d'identification requis, dans ce cas la spécification ECN est en erreur.

NOTE – Ce problème est très susceptible d'apparaître si des objets de codage BER sont appliqués à des constructeurs de codage et non à leurs composants, car les règles BER dépendent beaucoup des pointeurs d'identification. Les objets de codage PER n'utilisent pas de pointeurs d'identification.

14 Le module de définition de codage (EDM)

NOTE – Il y a deux productions de niveau supérieur en notation ECN: la production "EDMDefinition" spécifiée dans ce paragraphe et la production "ELMDefinition" spécifiée dans l'article 12. Ces paragraphes spécifient la syntaxe définissant les modules EDM et le module ELM respectivement.

14.1 La production "EDMDefinition" est la suivante:

```
EDMDefinition ::=
    ModuleIdentifier
    ENCODING-DEFINITIONS
    "::="
    BEGIN
    EDMModuleBody
    END
```

14.2 Dans toute application donnée de la notation ECN, il y a zéro, un ou plusieurs modules EDM qui définissent des objets de codage pour application dans le module ELM.

NOTE – S'il y a zéro module EDM, seuls des ensembles d'objets de codage intégrés peuvent être utilisés dans le module ELM.

14.3 La production "ModuleIdentifier" (et sa sémantique) est définie dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 12.1.

14.4 La production "ModuleIdentifier" offre une identification univoque d'un quelconque module dans l'ensemble de tous les modules ASN.1, ELM et EDM.

14.5 La production "EDMModuleBody" est la suivante:

```
EDMModuleBody ::=
        Exports ?
        RenamesAndExports ?
        Imports ?
        EDMAssignmentList ?

EDMAssignmentList ::=
        EDMAssignment
        EDMAssignmentList ?


EDMAssignment ::=
            EncodingClassAssignment
        |   EncodingObjectAssignment
        |   EncodingObjectSetAssignment
        |   ParameterizedAssignment
```

14.6 Les productions "Exports" et "Imports" (et leur sémantique) sont définies dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 12.1, telle que modifiée par le § A.1 de la présente Recommandation | Norme internationale.

14.7 La clause "Exports" met à disposition pour importation dans d'autres modules EDM (et le module ELM) tout nom de référence défini ou importé dans le module EDM courant sauf celui d'une structure produite implicitement. Le "Symbol" contenu dans la clause "Exports" peut faire référence à toute classe de codage (sauf une classe de codage intégrée ou une structure produite implicitement), à tout objet de codage, ou à tout ensemble d'objets de codage. Le "Symbol" doit avoir été défini dans ce module EDM, ou y avoir été importé.

NOTE – Lorsque le nom d'une structure de codage produite implicitement et importée est une référence de classe de codage intégrée, ce nom peut être utilisé dans le module EDM avec un nom entièrement qualifié. Une structure de codage produite implicitement ne peut jamais être exportée à partir d'un module EDM (cependant, des structures de codage définies au moyen de ce module peuvent, évidemment, être exportées).

14.8 La production "RenamesAndExports" est définie dans l'article 15.

14.9 La production "RenamesAndExports" (appelée clause de renommage) met à disposition (dans le module EDM) des structures de codage produites explicitement issues des structures de codage produites implicitement contenues dans des modules ASN.1 spécifiés. Elle rend également ces structures de codage produites explicitement disponibles pour importation dans d'autres modules EDM (et le module ELM). (Voir l'article 15.)

14.10 La clause "Imports" met à disposition (dans le module EDM) des classes de codage, des objets de codage et des ensembles d'objets de codage exportés à partir d'autres modules EDM ou automatiquement exportés à partir de modules ASN.1.

14.11 Tous les modules ASN.1 qui définissent des noms de référence de type non paramétrés produisent et exportent automatiquement une structure de codage produite implicitement du même nom précédé par le caractère "**#**". De telles classes de codage peuvent être importées dans un module EDM à partir de ce module ASN.1.

NOTE – Lorsque de tels noms sont les mêmes que les noms de classes de codage intégrées, la forme externe de la référence, comme spécifié dans le § A.1, doit être utilisée dans le corps du module importateur et dans toute clause de renommage.

14.12 Chaque production "EDMAssignment" définit un nom de référence et peut faire usage d'autres noms de référence. Chaque nom de référence utilisé dans un module doit soit être importé dans ce module ou doit être défini précisément une seule fois à l'intérieur de ce module.

NOTE – C'est une exigence plus sévère que celle qui est imposée aux modules ASN.1. Dans les modules ASN.1, des références externes peuvent être utilisées pour des types et des valeurs qui n'ont pas été importés. Dans un module EDM (et dans un module ELM) des références externes ne peuvent être utilisées que pour des classes de codage qui ont été citées en référence dans une clause d'importation. L'objectif des références externes ne consiste qu'à résoudre des ambiguïtés entre noms importés et noms intégrés, ou entre deux noms identiques importés à partir de modules différents.

14.13 Il n'est pas obligatoire qu'un nom de référence utilisé dans une même attribution soit défini (dans une autre instruction d'attribution) textuellement avant son utilisation.

14.14 Les productions de type "EDMAssignment" sont définies dans les paragraphes suivants comme suit:

EncodingClassAssignment	article 16
EncodingObjectAssignment	article 17
EncodingObjectSetAssignment	article 18
ParameterizedAssignment	§ C.1

NOTE – La production "ParameterizedAssignment" autorise le paramétrage d'une production "EncodingClassAssignment", "EncodingObjectAssignment" et "EncodingObjectSetAssignment", comme spécifié dans le § C.1.

15 La clause de renommage

15.1 Structures produites explicitement et exportées

La production "RenamesAndExports" est la suivante:

```
RenamesAndExports ::=
        RENAMES
        ExplicitGenerationList ";"

ExplicitGenerationList ::=
        ExplicitGeneration
        ExplicitGenerationList ?

ExplicitGeneration ::=
        OptionalNameChanges
        FROM GlobalModuleReference

OptionalNameChanges ::=
        NameChanges |  GENERATES
```

NOTE – Un exemple de l'utilisation de la clause de renommage afin de créer des structures de codage produites explicitement est donné au § D.3.7.

15.1.2 La production "GlobalModuleReference" est définie dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 12.1 et doit désigner un module ASN.1.

15.1.3 La production "RenamesAndExports" est appelée clause de renommage.

15.1.4 Chaque production "ExplicitGeneration" produit et exporte, à partir de ce module, une structure de codage produite explicitement pour chacune des structures de codage produites implicitement du module ASN.1 cité en référence par "GlobalModuleReference". Chaque champ de la structure de codage produite explicitement est associé aux mêmes valeurs abstraites que le champ correspondant de la structure de codage produite implicitement (qui sont les valeurs associées au champ correspondant du type ASN.1 à partir duquel elle a été créée).

15.1.5 Si une clause de renommage fait référence à plusieurs modules ASN.1 et qu'il en résulte deux structures produites explicitement ayant le même nom simple, dans ce cas aucune de ces structures n'est disponible pour importation explicite dans un module ELM ou dans un module EDM.

NOTE – Ces structures produites explicitement existent néanmoins, et sont susceptibles d'être implicitement citées en référence par d'autres structures produites explicitement qui sont exportées sans restriction.

15.1.6 L'objectif principal de la clause de renommage consiste à rendre disponibles les structures produites explicitement pour importation dans d'autres modules, particulièrement le module ELM. Cependant, cette clause rend également ces structures disponibles pour référence dans le module EDM contenant la clause de renommage sauf comme spécifié dans le § 15.1.7. Si le nom simple est ambigu, dans ce cas un nom entièrement qualifié doit être utilisé dans le module EDM contenant la clause de renommage, comme spécifié dans le § 15.1.9.

NOTE – Une ambiguïté peut apparaître soit en raison de conflits avec les noms de classes intégrées, ou en raison de conflits de noms simples entre structures produites à partir de plusieurs modules ASN.1, ou les deux.

15.1.7 Lorsqu'une clause de renommage crée une structure produite explicitement à partir d'une structure produite implicitement, cette structure produite implicitement ne peut pas être importée dans ce module EDM au moyen d'une clause d'importation et la structure produite implicitement n'est jamais disponible dans ce module EDM.

15.1.8 Ces structures de codage produites explicitement ont le même nom de référence simple que la structure de codage produite implicitement à partir de laquelle elles ont été formées (mais sont des classes distinctes). Lorsqu'un nom entièrement qualifié est requis pour une structure de codage produite explicitement, ce nom entièrement qualifié doit contenir le "ModuleIdentifier" du module EDM contenant la clause de renommage, comme spécifié dans le § 15.1.9.

NOTE – Les structures de codage produites implicitement utilisées lors de leur production ont le même nom de référence simple, mais leur nom entièrement qualifié contient le "ModuleIdentifier" du module ASN.1 dans lequel le type correspondant a été défini.

15.1.9 Si un module EDM crée des structures de codage produites explicitement à partir de plusieurs modules ASN.1, il est possible que certaines de ces structures aient les mêmes noms simples de classe de codage. Si n'importe laquelle de ces structures est citée en référence dans le corps de ce module EDM, dans ce cas la référence doit être une notation "ExternalEncodingClassReference" contenant la référence "modulereference" utilisée comme référence de module ASN.1 dans la clause de remplacement de ce module EDM.

15.1.10 La notation "ExternalEncodingClassReference" ne doit pas être utilisée dans une clause d'importation sauf où elle est requise par le § 15.1.9.

15.1.11 Si un nom qui a été importé au moyen d'une notation "ExternalEncodingClassRefererence" est utilisé dans le corps d'un module, dans ce cas la référence simple "encodingclassreference" peut être utilisée à moins qu'une référence "ExternalEncodingClassReference" ne soit requise comme spécifié dans le § 15.1.9.

15.1.12 Si la production "OptionalNameChanges" est **GENERATES**, dans ce cas toutes les structures de codage produites explicitement sont identiques aux structures de codage produites implicitement et utilisées lors de leur production, sauf comme spécifié dans le § 15.1.14.

NOTE – (Didactique) Si, dans un module EDM, il y a de multiples structures ayant le même nom de référence simple (que ces noms apparaissent à partir d'une clause d'importation ou à partir d'une clause de renommage, ou à partir de conflits avec des classes intégrées, ou à partir de toute combinaison de ces éléments), alors un nom entièrement qualifié est utilisé sauf pour les références à une classe intégrée. Pour les structures produites implicitement, le nom entièrement qualifié utilise toujours le nom du module ASN.1. Pour les structures produites par la clause de renommage dans un module EDM, le nom entièrement qualifié est utilisé. Ce nom entièrement qualifié dans le corps de ce module EDM utilise toujours le nom du module ASN.1 cité en référence par la clause de renommage. Pour les structures importées à partir d'un autre module EDM, le nom entièrement qualifié utilise le nom de ce module EDM. Cela est toujours univoque, car l'importation n'est pas permise si un module EDM crée de multiples structures produites explicitement ayant le même nom de référence simple.

15.1.13 Si "OptionalNameChanges" est "NameChanges", dans ce cas le § 15.1.14 s'applique encore, mais les structures de codage produites explicitement sont encore modifiées comme spécifié dans le § 15.2.

15.1.14 Soit une structure de codage produite implicitement (A, par exemple) qui contient une référence de classe de codage à une autre structure de codage produite implicitement (B, par exemple). Dans ce cas:

a) si cette clause de renommage (contenue dans n'importe laquelle de ses productions "ExplicitGeneration") crée une structure de codage produite explicitement correspondant à B (B1, par exemple), dans ce cas la référence correspondante contenue dans la structure de codage produite explicitement correspondant à A est une référence à B1;

b) s'il n'y a pas de structure de codage produite explicitement correspondant à B, dans ce cas la référence contenue dans la structure de codage produite correspondant à A est une référence à B.

15.2 Renommages

15.2.1 La production "NameChanges" est la suivante:

```
NameChanges ::=
        NameChange
        NameChanges ?

NameChange ::=
        OriginalClassName
        AS
        NewClassName
        IN
        NameChangeDomain

OriginalClassName ::= SimpleDefinedEncodingClass | BuiltinEncodingClassReference

NewClassName ::= encodingclassreference
```

15.2.2 Chaque "NameChanges" spécifie que, dans la production de structures de codage produites explicitement, toutes les instances de "OriginalClassName" à l'intérieur de "NameChangeDomain" dans les structures de codage produites implicitement doivent être renommées dans la classe "NewClassName". "NameChangeDomain" est spécifié dans le § 15.3 et identifie une ou plusieurs structures de codage produites implicitement (ou composants de ces structures) à partir du module ASN.1 cité en référence par la référence "GlobalModuleReference" dans la production "ExplicitGeneration".

NOTE 1 – Cela permet que différents codages soient appliqués à certaines instances d'une classe à partir du codage appliqué à d'autres instances.

NOTE 2 – Cela implique que le nom "OriginalClassName" ne peut être un nom produit implicitement à partir d'un type ASN.1, c'est-à-dire le nom d'un type défini par l'usager ASN.1 (précédé par "**#**") ou un des noms de classe énumérés dans la colonne 2 du Tableau 2.

15.2.3 Les références par nom "OriginalClassName" à des champs de la structure de codage produite implicitement qui correspond à l'utilisation de la référence "ExternalTypeReference" dans la définition de type ASN.1 doivent utiliser la notation "SimpleDefinedEncodingClass" avec la même référence "modulereference" que "ExternalTypeReference". Sinon, si le type "DefinedType" (précédé par un caractère "**#**") n'est pas une référence "BuiltinEncodingClassReference", une simple référence "encodingclassreference" doit être utilisée. Si une référence "typereference" (précédée par un caractère "**#**") est une référence "BuiltinEncodingClassReference", alors la notation "SimpleDefinedEncodingClass" doit être utilisée avec la même référence "modulereference" que le module ASN.1 qui a créé la structure de codage produite implicitement.

15.2.4 Lorsqu'un module ELM importe une structure de codage produite explicitement à partir d'un module EDM, les clauses de renommage dans d'autres modules EDM n'ont aucun effet sur le codage de cette structure.

NOTE – Cela signifie en pratique que toute la "coloration" (voir § 9.16.4) nécessaire pour un message particulier, doit être effectuée dans un même module EDM.

15.2.5 Le nom "NewClassName" doit être défini dans une instruction d'attribution de classe de codage (voir article 16) de la forme:

<NewClassName> ::= <OriginalClassName>

où "***<NewClassName>***" et "***<OriginalClassName>***" sont les noms des classes nouvelles et originales apparaissant dans la production "NameChanges". L'attribution doit être dans le module EDM avec la clause de renommage.

NOTE – Le nom "***<OriginalClassName>***" est tenu de faire référence à une classe de codage intégrée ou à une structure de codage produite extérieurement par la clause de renommage contenue dans ce module. En cas d'ambiguïté, cela nécessitera l'utilisation d'une référence externe dans le nom "***<OriginalClassName>***".

15.3 Spécification de la région pour renommages

La production "NameChangeDomain" est la suivante:

NameChangeDomain ::=
IncludedRegions
Exception ?

Exception ::=
EXCEPT
ExcludedRegions

IncludedRegions ::=
ALL | RegionList

ExcludedRegions ::= RegionList

RegionList ::=
Region "," +

Region ::=
SimpleDefinedEncodingClass |
ComponentReference

ComponentReference ::=
SimpleDefinedEncodingClass
"."
ComponentIdList

ComponentIdList ::=
identifier "." +

15.3.2 Chaque "SimpleDefinedEncodingClass" doit être le nom d'une structure de codage produite implicitement à partir du module ASN.1 cité en référence par la référence "GlobalModuleReference" dans la production "ExplicitGeneration". Lorsqu'il est utilisé dans la production "Region", ce nom identifie l'ensemble de cette définition de la structure de codage.

NOTE – La forme "ExternalEncodingClassReference" du nom "SimpleDefinedEncodingClass" est utilisée si la classe citée en référence est issue d'un nom "typereference" qui (lorsque précédé par "**#**") est une référence "BuiltinEncodingClassReference" (voir § 15.2.3).

15.3.3 Chaque "identifier" doit être l'identificateur "identifier" dans un champ "NamedField" de la structure de codage produite implicitement et identifiée par la référence "encodingclassreference" dans la production "ComponentReference", qui identifie l'entière définition du composant identifié de cette structure de codage.

15.3.4 Le premier "identifier" de la liste "ComponentIdList" doit être un "identifier" dans un champ "NamedField" de la structure de codage produite implicitement et identifiée par la référence "encodingclassreference" dans la production "ComponentReference". Il identifie l'entière définition de ce composant de la structure de codage. Chaque "identifier" subséquent de la liste "ComponentIdList" doit être un "identifier" dans un "NamedField" de la structure de codage produite implicitement et identifiée par la partie précédente de la liste "ComponentIdList" et identifie l'entière définition de ce composant.

15.3.5 Les définitions identifiées par différentes productions "Region" dans une liste "RegionList" doivent être disjointes. Une définition est identifiée par une liste "RegionList" si et seulement si elle est identifiée par une "Region" dans une liste "RegionList".

15.3.6 Si la production "IncludedRegions" a la valeur **ALL**, elle identifie toutes les parties de toutes les structures de codage produites implicitement à partir du module ASN.1 cité en référence par la référence "GlobalModuleReference" dans la production "ExplicitGeneration".

15.3.7 Les définitions identifiées par la production "ExcludedRegions" doivent être un sous-ensemble correct des définitions identifiées par la production "IncludedRegions".

15.3.8 La spécification "NameChangeDomain" specification identifie les définitions dans lesquelles les renommages doivent être effectués. Les définitions contenues dans la spécification "NameChangeDomain" sont celles qui sont identifiées par les productions "IncludedRegions" qui ne sont pas également identifiées par "ExcludedRegions".

16 Attribution des classes de codage

16.1 Généralités

16.1.1 La production "EncodingClassAssignment" est la suivante:

```
EncodingClassAssignment ::=
        encodingclassreference
        "::="
        EncodingClass
```

16.1.2 La production "EncodingClassAssignment" attribue la classe "EncodingClass" à la production "encodingclassreference".

NOTE – Toute notation "EncodingObject" qui a été validée avec "EncodingClass" en tant que gouverneur est valide avec "encodingclassreference" en tant que gouverneur.

16.1.3 Une classe de codage est dans une des catégories suivantes:

a) une catégorie dans le groupe catégoriel des champs binaires (voir § 16.1.7);

b) la catégorie des options (voir § 16.1.8);

c) la catégorie des concaténations (voir § 16.1.9);

d) la catégorie des répétitions (voir § 16.1.10);

e) la catégorie des offres d'options (voir § 16.1.11);

f) la catégorie des étiquettes (voir § 16.1.12);

g) une catégorie dans le groupe catégoriel des procédures de codage (voir § 16.1.13).

NOTE – Le terme *constructeur de codage* est utilisé pour toute classe dans les catégories des options, des concaténations et des répétitions. Ces catégories sont également rangées dans le groupe catégoriel des constructeurs de codage.

16.1.4 La catégorie de chaque classe de codage intégrée est spécifiée dans le § 16.1.14.

NOTE – Si une classe de codage est étiquetée (voir § 16.2.1) ou a des limites (voir § 16.2.6), la catégorie de cette classe est celle de la classe avec suppression des étiquettes et des limites.

16.1.5 La production "EncodingClass" est la suivante:

```
EncodingClass ::=
        BuiltinEncodingClassReference
    |   EncodingStructure
```

16.1.6 La production "BuiltinEncodingClassReference" est la suivante:

```
BuiltinEncodingClassReference ::=
        BitfieldClassReference
    |   AlternativesClassReference
    |   ConcatenationClassReference
    |   RepetitionClassReference
    |   OptionalityClassReference
    |   TagClassReference
    |   EncodingProcedureClassReference
```

16.1.7 La production "BitfieldClassReference" est la suivante:

```
BitfieldClassReference ::=
        #NUL
    |   #BOOL
    |   #INT
    |   #BITS
```

```
        |  #OCTETS
        |  #CHARS
        |  #PAD
        |  #BIT-STRING
        |  #BOOLEAN
        |  #CHARACTER-STRING
        |  #EMBEDDED-PDV
        |  #ENUMERATED
        |  #EXTERNAL
        |  #INTEGER
        |  #NULL
        |  #OBJECT-IDENTIFIER
        |  #OCTET-STRING
        |  #OPEN-TYPE
        |  #REAL
        |  #RELATIVE-OID
        |  #GeneralizedTime
        |  #UTCTime
        |  #ObjectDescriptor
        |  #BMPString
        |  #GeneralString
        |  #GraphicString
        |  #IA5String
        |  #NumericString
        |  #PrintableString
        |  #TeletexString
        |  #UniversalString
        |  #UTF8String
        |  #VideotexString
        |  #VisibleString
```

Les catégories des classes auxquelles ces noms intégrés font référence (voir § 16.1.14) sont toutes définies comme étant dans le groupe catégoriel des champs binaires.

16.1.8 La production "AlternativesClassReference" est la suivante:

```
AlternativesClassReference ::=
        #ALTERNATIVES
      | #CHOICE
```

16.1.9 La production "ConcatenationClassReference" est la suivante:

```
ConcatenationClassReference ::=
        #CONCATENATION
      | #SEQUENCE
      | #SET
```

16.1.10 La production "RepetitionClassReference" est la suivante:

```
RepetitionClassReference ::=
        #REPETITION
      | #SEQUENCE-OF
      | #SET-OF
```

16.1.11 La production "OptionalityClassReference" est la suivante:

```
OptionalityClassReference ::=
        #OPTIONAL
```

16.1.12 La production "TagClassReference" est la suivante:

```
TagClassReference ::=
        #TAG
```

16.1.13 La production "EncodingProcedureClassReference" est la suivante:

```
EncodingProcedureClassReference ::=
        #TRANSFORM
      | #CONDITIONAL-INT
      | #CONDITIONAL-REPETITION
      | #OUTER
```

16.1.14 Certaines de ces classes sont définies comme étant primitives, et ne peuvent être codées par des objets de codage de leur propre classe. D'autres sont issues d'une classe primitive au moyen d'instructions d'attribution de classe

et peuvent être déréférencées à ces classes. Leur catégorie est celle de la classe de laquelle ils sont issus. Les classes primitives dont chaque classe intégrée est issue au moyen d'instructions d'attribution de classe sont énumérées ci-après. Lors d'une définition des objets de codage de classes dérivées, toute syntaxe permise pour la classe primitive correspondante peut être utilisée pour la classe dérivée. La troisième colonne du tableau indique la catégorie pour chacune des classes intégrées qui ne sont pas issues d'autres classes.

Classe intégrée	Dérivée de	Catégorie
#ALTERNATIVES	(primitive)	**alternatives (options)**
#BITS	(primitive)	**bitstring (chaîne de bits)**
#BIT-STRING	**#BITS**	
#BOOL	(primitive)	**boolean (booléen)**
#BOOLEAN	**#BOOL**	
#CHARACTER-STRING	(défini utilisant **#SEQUENCE**)	
#CHARS	(primitive)	**characterstring (chaîne de caractères)**
#CHOICE	**#ALTERNATIVES**	
#CONCATENATION	(primitive)	**concatenation (concaténation)**
#CONDITIONAL-INT	(primitive)	**encoding procedure (procédure de codage)**
#CONDITIONAL-REPETITION	(primitive)	**encoding procedure (procédure de codage)**
#EMBEDDED-PDV	(défini utilisant **#SEQUENCE**)	
#ENUMERATED	**#INT**	
#EXTERNAL	(défini utilisant **#SEQUENCE**)	
#INT	(primitive)	**integer (entier)**
#INTEGER	**#INT**	
#NUL	(primitive)	**null (néant)**
#NULL	**#NUL**	
#OBJECT-IDENTIFIER	(primitive)	**objectidentifier (identificateur d'objet)**
#OCTETS	(primitive)	**octetstring (chaîne d'octets)**
#OCTET-STRING	**#OCTETS**	
#OPEN-TYPE	(primitive)	**opentype (type ouvert)**
#OPTIONAL	(primitive)	**optionality (offre d'option)**
#OUTER	(primitive)	**encoding procedure (procédure de codage)**
#PAD	(primitive)	**pad (bourrage)**
#REAL	(primitive)	**real (réel)**
#RELATIVE-OID	**#OBJECT-IDENTIFIER**	
#REPETITION	(primitive)	**repetition (répétition)**
#SEQUENCE	**#CONCATENATION**	
#SEQUENCE-OF	**#REPETITION**	
#SET	**#CONCATENATION**	
#SET-OF	**#REPETITION**	
#TAG	(primitive)	**tag (étiquette)**
#TRANSFORM	(primitive)	**encoding procedure (procédure de codage)**
#GeneralizedTime	**#CHARS**	
#UTCTime	**#CHARS**	
#ObjectDescriptor	**#CHARS**	
#BMPString	**#CHARS**	
#GeneralString	**#CHARS**	
#GraphicString	**#CHARS**	
#IA5String	**#CHARS**	
#NumericString	**#CHARS**	
#PrintableString	**#CHARS**	
#TeletexString	**#CHARS**	
#UniversalString	**#CHARS**	
#UTF8String	**#CHARS**	
#VideotexString	**#CHARS**	
#VisibleString	**#CHARS**	

16.2 Définition de la structure de codage

16.2.1 La production "EncodingStructure" est la suivante:

```
EncodingStructure ::=
        TaggedStructure
    |   UntaggedStructure

TaggedStructure ::=
        "["
        TagClass
        TagValue ?
        "]"
        UntaggedStructure
```

```
UntaggedStructure ::=
        DefinedEncodingClass
    |   EncodingStructureField
    |   EncodingStructureDefn

TagClass ::=
        DefinedEncodingClass |
        TagClassReference

TagValue ::=
        "(" number ")"
```

16.2.2 Une production "EncodingStructure" définit une classe de codage fondée sur une structure au moyen de la notation spécifiée ci-dessous. Cette notation permet la définition de classes de codage arbitraires au moyen de classes de codage intégrées et de classes de codage définies (qui peuvent être des structures de codage produites) pour les classes des champs binaires, des constructeurs de codage et des procédures de codage dans la catégorie des offres d'options. Toutes les classes définies par la production "EncodingStructure" sont dans la catégorie des structures de codage. (Des exemples d'attribution de structure de codage illustrant n'importe laquelle des structures syntaxiques sont donnés au § D.2.8.4; le § D.2.2.3 est un exemple de l'utilisation de la classe **`#TAG`**.)

NOTE – La syntaxe interdit la spécification d'une classe d'étiquettes suivant immédiatement une autre classe d'étiquettes dans la définition d'une structure de codage; et de telles structures ne peuvent être produites par de multiples étiquettes textuelles dans une définition de type ASN.1 (voir § 11.3.4.1 e)).

16.2.3 La classe "DefinedEncodingClass" est spécifiée dans le § 10.9.1 et doit appartenir au groupe catégoriel des champs binaires.

16.2.4 La classe "DefinedEncodingClass" contenue dans la classe "TagClass" doit être une classe de la catégorie des étiquettes (voir § 16.1.3).

16.2.5 La notation "number" dans "TagValue" spécifie un numéro d'étiquette qui est associé à la classe de la catégorie des étiquettes.

16.2.6 La production "EncodingStructureField" est la suivante:

```
EncodingStructureField ::=
        #NUL
    |   #BOOL
    |   #INT                        Bounds?
    |   #BITS                       Size?
    |   #OCTETS                     Size?
    |   #CHARS                      Size?
    |   #PAD
    |   #BIT-STRINGSize?
    |   #BOOLEAN
    |   #CHARACTER-STRING
    |   #EMBEDDED-PDV
    |   #ENUMERATED                 Bounds?
    |   #EXTERNAL
    |   #INTEGER                    Bounds?
    |   #NULL
    |   #OBJECT-IDENTIFIER
    |   #OCTET-STRING               Size?
    |   #OPEN-TYPE
    |   #REAL
    |   #RELATIVE-OID
    |   #GeneralizedTime
    |   #UTCTime
    |   #ObjectDescriptor           Size?
    |   #BMPString                  Size?
    |   #GeneralString              Size?
    |   #GraphicString              Size?
    |   #IA5String                  Size?
    |   #NumericString              Size?
    |   #PrintableString            Size?
    |   #TeletexString              Size?
    |   #UniversalString            Size?
    |   #UTF8String                 Size?
    |   #VideotexString             Size?
    |   #VisibleString              Size?
```

16.2.7 Les productions "EncodingStructureField" représentent tous les codages de chaîne de bits possibles pour les types ASN.1 correspondants. Elles peuvent être attribuées à des valeurs de ces types par mappage de valeurs (voir article 19).

16.2.8 Les valeurs ASN.1 qui peuvent être associées à chaque champ primitif sont les suivantes:

#NUL	The null value (valeur néant)
#BOOL	The boolean values (valeurs booléennes)
#INT	The integer values (valeurs entières)
#BITS	Bitstring values (valeurs chaîne de bits)
#OCTETS	Octetstring values (valeurs chaîne d'octets)
#CHARS	Character string values (valeurs chaîne de caractères)
#PAD	None (aucune)
#OBJECT-IDENTIFIER	Object identifier values (valeurs d'identificateur d'objet)
#OPEN-TYPE	Open type values (valeurs type ouvert)
#REAL	Real values (valeurs réelles)
#TAG	Tag numbers (numéro d'étiquette)

NOTE – Le champ de classe **#PAD** ne peut pas avoir de valeurs ASN.1 associées et n'est jamais visible en dehors des procédures de codage et de décodage.

16.2.9 Les notations "Bounds" et "Size" spécifient respectivement les limites ou la contrainte de longueur effective applicables aux valeurs abstraites qui peuvent être mappées sur le champ (voir article 19).

NOTE – Les contraintes d'alphabet effectivement permis ne peuvent pas être attribuées dans une définition de la structure de codage. Elles ne peuvent être attribuées qu'au moyen des mappages de valeur de l'article 19.

16.2.10 Les notations "Bounds" et "Size" sont les suivantes:

```
Bounds ::= "(" EffectiveRange ")"

EffectiveRange ::=
        MinMax
    |   Fixed

Size      ::= "(" SIZE SizeEffectiveRange ")"

SizeEffectiveRange ::=
          "(" EffectiveRange ")"

MinMax ::=
          ValueOrMin
          ".."
          ValueOrMax

ValueOrMin  ::=
          SignedNumber |
          MIN

ValueOrMax  ::=
          SignedNumber |
          MAX

Fixed ::=  SignedNumber
```

16.2.11 Les notations **MIN** et **MAX** spécifient respectivement qu'il n'y a pas de limite inférieure ou supérieure. La notation **MIN** ne doit pas être utilisée dans "Size". La notation "Fixed" signifie une seule valeur ou une seule longueur. La notation "SignedNumber" est spécifiée dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 18.1. Elle doit être non négative lorsqu'elle est utilisée dans "Size". "ValueOrMin" et "ValueOrMax" spécifient des limites inférieures et supérieures respectivement.

16.2.12 La production "EncodingStructureDefn" est la suivante:

```
EncodingStructureDefn ::=
          AlternativesStructure
      |   RepetitionStructure
      |   ConcatenationStructure
```

16.2.13 Ces structures de codage sont définies dans les paragraphes suivants:

AlternativesStructure	§ 16.3
RepetitionStructure	§ 16.4
ConcatenationStructure	§ 16.5

16.3 Structure de codage à option

16.3.1 La production "AlternativesStructure" est la suivante:

```
AlternativesStructure ::=
        AlternativesClass
        "{"
        NamedFields
        "}"

AlternativesClass ::=
        DefinedEncodingClass        |
        AlternativesClassReference

NamedFields ::= NamedField "," +

NamedField ::=
        identifier
        EncodingStructure
```

16.3.2 La production "AlternativesStructure" identifie la présence dans un codage de précisément une des structures "EncodingStructure" contenues dans ses champs "NamedFields". La classe "DefinedEncodingClass" doit être dans la catégorie des options (voir § 16.1.8). Les mécanismes utilisés afin de désigner laquelle des structures "EncodingStructure" est présente dans un codage sont spécifiés par un objet de codage de la classe "AlternativesClass".

16.3.3 La production "AlternativesStructure" est un constructeur de codage: lorsqu'un ensemble d'objets de codage est appliqué à cette structure comme spécifié dans le § 13.2, le codage de la classe "AlternativesClass" détermine le choix d'options et le point d'application passe alors à chacune des structures "EncodingStructure" dans ses "NamedFields".

16.4 Structure de codage de répétition

16.4.1 La production "RepetitionStructure" est la suivante:

```
RepetitionStructure ::=
        RepetitionClass
        "{"
        identifier ?
        EncodingStructure
        "}"
        Size?

RepetitionClass ::=
        DefinedEncodingClass        |
        RepetitionClassReference
```

16.4.2 La production "RepetitionStructure" identifie la présence dans un codage d'instances répétées de la structure "EncodingStructure" dans la production. La construction facultative "Size" (voir § 16.2.9) spécifie des limites applicables au nombre de répétitions. Les mécanismes utilisés afin de désigner la façon dont de nombreuses répétitions de la structure "EncodingStructure" sont présentes dans un codage sont spécifiés par un objet de codage de la classe "RepetitionClass". La classe "DefinedEncodingClass" doit être dans la catégorie des répétitions (voir § 16.1.10).

16.4.3 La production "RepetitionStructure" est un constructeur de codage: lorsqu'un objet de codage est appliqué à cette structure comme spécifié dans le § 13.2, le codage de la classe "RepetitionClass" détermine les mécanismes permettant de déterminer le nombre de répétitions, et le point d'application passe alors à la structure "EncodingStructure" dans la production.

NOTE – Les caractères "**{**" et "**}**" sont utilisés dans cette construction, mais ne sont pas présents dans la construction ASN.1 **SEQUENCE OF** associée. Cette opération a été effectuée afin de contribuer à éviter des ambiguïtés syntaxiques dans une définition de structure.

16.5 Structure de codage à concaténation

16.5.1 La production "ConcatenationStructure" est la suivante:

```
ConcatenationStructure ::=
        ConcatenationClass
        "{"
        ConcatComponents
        "}"
```

```
ConcatenationClass ::=
        DefinedEncodingClass        |
        ConcatenationClassReference

ConcatComponents ::=
        ConcatComponent "," *

ConcatComponent ::=
        NamedField
        ConcatComponentPresence ?

ConcatComponentPresence ::=
        OPTIONAL-ENCODING
        OptionalClass

OptionalClass ::=
        DefinedEncodingClass        |
        OptionalityClassReference
```

16.5.2 La production "ConcatenationStructure" identifie la présence dans un codage de zéro ou un des codages pour chacune des structures "EncodingStructure" dans ses champs "NamedField". La classe "DefinedEncodingClass" dans la classe "ConcatenationClass" doit être une classe de la catégorie des concaténations (voir § 16.1.9) et la classe "DefinedEncodingClass" dans la classe "OptionalClass" doit être une classe de la catégorie des offres d'options (voir § 16.1.3).

16.5.3 Si la production "ConcatComponentPresence" est absente d'un "composant", dans ce cas la structure "EncodingStructure" contenue dans ce champ nommé doit apparaître précisément une fois dans le codage.

16.5.4 Si la production "ConcatComponentPresence" est présente, le mécanisme utilisé afin de déterminer s'il y a un codage de la structure "EncodingStructure" correspondante est spécifié par l'objet de codage qui code la classe "OptionalClass".

16.5.5 L'ordre dans lequel les codages de chaque champ "NamedField" apparaissent dans un codage de la concaténation (et le moyen d'identification du champ "NamedField" qui est représenté par un codage) est déterminé par un objet de codage de la classe "ConcatenationClass".

16.5.6 La production "ConcatenationStructure" est un constructeur de codage: lorsqu'un objet de codage est appliqué à cette structure comme spécifié dans le § 13.2, le codage de la classe "ConcatenationClass" détermine les procédures de concaténation et le point d'application passe alors à chacune des structures "EncodingStructure" dans ses champs nommés.

17 Attribution des objets de codage

17.1 Généralités

17.1.1 La production "EncodingObjectAssignment" est la suivante:

```
EncodingObjectAssignment ::=
    encodingobjectreference
    DefinedOrBuiltinEncodingClass
    "::="
    EncodingObject
```

17.1.2 La production "EncodingObjectAssignment" définit la référence "encodingobjectreference" en tant que référence à un objet de codage "EncodingObject" qui doit être une production engendrant un objet de la classe de codage "DefinedOrBuiltinEncodingClass". (Les § D.1.2.2, D.1.7.3 et D.1.8.2 fournissent des exemples d'attribution d'objet de codage pour les différentes constructions syntaxiques relatives à l'objet "EncodingObject" spécifié ci-dessous.)

17.1.3 La production "DefinedOrBuiltinEncodingClass" est appelée *gouverneur* de la notation "EncodingObject" dans cette production.

NOTE 1 – Chaque fois que la production "EncodingObject" apparaît en notation ECN, il y a un gouverneur et la syntaxe de la notation gouvernée dépend de la classe de codage du gouverneur.

NOTE 2 – La syntaxe de la notation gouvernée a été conçue de façon qu'un analyseur de syntaxe puisse trouver son extrémité sans avoir connaissance du gouverneur.

17.1.4 Il ne doit y avoir aucune définition récursive (voir 3.2.38) d'une référence "encodingobjectreference" et il ne doit y avoir aucune instanciation récursive (voir 3.2.39) d'une référence "encodingobjectreference".

17.1.5 La production "EncodingObject" est la suivante:

```
EncodingObject ::=
        DefinedEncodingObject
    |   DefinedSyntax
    |   EncodeWith
    |   EncodeByValueMapping
    |   EncodeStructure
    |   DifferentialEncodeDecodeObject
    |   EncodingOptionsEncodingObject
    |   NonECNEncodingObject
```

17.1.6 La production "DefinedEncodingObject" identifie un objet de codage et est spécifiée dans le § 10.9.2. La production "DefinedEncodingObject" doit être de la même classe de codage que le gouverneur, ou d'une classe qui peut être obtenue à partir du gouverneur par déréférencement. La référence "encodingobjectreference" qui est définie fait apparaître un pointeur d'identification si et seulement si la production "DefinedEncodingObject" fait apparaître ce pointeur d'identification.

17.1.7 Dans la présente Recommandation | Norme internationale, les expressions "la même classe de codage" et "la même classe" doivent être interprétées comme signifiant que la notation utilisée afin de définir les deux classes doit avoir le même nom de référence de classe de codage, ou doit avoir un nom de référence qui déréférence au même nom de classe de codage.

17.1.8 Les productions restantes du type "EncodingObject" sont définies dans les paragraphes suivants et fournissent d'autres moyens de définition des objets de codage de la classe de gouverneur:

DefinedSyntax	§ 17.2 avec les articles 20 à 25
EncodeWith	§ 17.3
EncodeByValueMapping	§ 17.4
EncodeStructure	§ 17.5
DifferentialEncodeDecodeObject	§ 17.6
EncodingOptionsEncodingObject	§ 17.7
NonECNEncodingObject	§ 17.8

17.2 Codage avec une syntaxe définie

17.2.1 La production "DefinedSyntax" est spécifiée dans la Rec. UIT-T X.681 | ISO/CEI 8824-2, § 11.5 et 11.6, telle que modifiée par le § B.16 de la présente Recommandation | Norme internationale. Elle est utilisée afin de définir des objets de codage relatifs à une classe de codage gouvernante. La syntaxe détaillée à cette fin est spécifiée dans les § 23 à 25, et la sémantique des constructions est spécifiée dans l'article 22.

17.2.2 Cette notation de définition des objets de codage n'est disponible que pour les classes de codage gouvernantes des catégories (ou classes) énumérées dans le Tableau 3 ci-dessous. La syntaxe à utiliser pour chaque objet de codage est "DefinedSyntax" pour la catégorie ou classe de codage correspondante (spécifiée dans les articles 23 à 25).

NOTE 1 – L'utilisation de cette syntaxe néccessite fréquemment l'inclusion d'un paramètre pour un déterminant. Les objets de codage paramétrés avec de tels paramètres (éventuellement inclus en tant que partie d'un ensemble paramétré d'objets de codage) ne sont utiles que pour l'application à une structure de codage contenue dans un module EDM, ou pour l'inclusion comme objets de codage à appliquer dans le cadre d'une action de remplacement. Ils ne peuvent pas être appliqués dans le module ELM.

NOTE 2 – Cette notation permet aux usagers de spécifier des objets de codage qui codent la classe **`#SET`** de la même façon que les règles PER codent normalement la classe **`#SEQUENCE`**, et vice versa. Les usagers sont censés être responsables de l'utilisation de cette notation.

Tableau 3 – Catégories et classes prises en charge par une syntaxe définie

catégorie néant
catégorie des booléens
catégorie des entiers
catégorie des chaînes de bits
catégorie des chaînes d'octets
catégorie des chaînes de caractères
catégorie des bourrages
catégorie des options
catégorie des répétitions
catégorie des concaténations
catégorie des offres d'options
classe **`#CONDITIONAL-INT`**
classe **`#CONDITIONAL-REPETITION`**
catégorie des étiquettes
classe **`#TRANSFORM`**
classe **`#OUTER`**

17.2.3 Les informations requises (et la syntaxe à utiliser) afin de spécifier un objet de codage de l'une de ces catégories ou classes au moyen de la syntaxe "DefinedSyntax" sont spécifiées par les définitions contenues dans les articles 23 à 25.

17.2.4 Si un gouverneur de valeur d'un des champs apparaissant dans la construction "DefinedSyntax" est nécessaire pour utilisation dans une liste de paramètres fictifs, la notation "EncodingClassFieldType" (spécifiée dans le § B.17) doit être utilisée. Aucun autre usage ne doit être fait de la notation"EncodingClassFieldType".

17.2.5 Lorsque la syntaxe définie dans l'article 23 nécessite la présence d'une **`REFERENCE`**, celle-ci ne peut être fournie que dans la construction "DefinedSyntax" au moyen d'un paramètre fictif de l'objet de codage qui est défini ou, dans le cas d'un "**`flag-to-be-used`**" ou "**`flag-to-be-set`**", au moyen d'un nom de référence qui est textuellement présent dans la définition d'une structure de remplacement.

17.2.6 La notation "DefinedSyntax" spécifie si la référence "encodingobjectreference" définie fait apparaître un pointeur d'identification.

17.3 Codage avec des ensembles d'objets de codage

17.3.1 La production "EncodeWith" est la suivante:

```
EncodeWith ::=
    "{" ENCODE CombinedEncodings "}"
```

17.3.2 La production "CombinedEncodings" et son application à une classe de codage est spécifiée dans l'article 13.

17.3.3 L'objet de codage défini par la notation "EncodeWith" est l'application de la production "CombinedEncodings" à la classe de codage qui est le gouverneur (voir § 17.1.3) de la notation "EncodeWith".

17.3.4 Il y a erreur de spécification si cela ne produit pas une spécification de codage complète pour la classe de gouverneur.

17.3.5 Si un ensemble d'objets de codage contenu dans la production "CombinedEncodings" est paramétré avec un paramètre qui est une **`REFERENCE`**, le paramètre réel fourni dans cette construction ne peut être un paramètre fictif de l'objet de codage qui est défini.

17.3.6 Dans l'application des codages spécifiés dans l'article 13, il y a un objet de codage (A par exemple) qui produit le premier champ binaire dans le codage résultant. La référence "encodingobjectreference" qui est définie fait apparaître un pointeur d'identification si et seulement si l'objet de codage A fait apparaître ce pointeur d'identification.

17.4 Codage avec des mappages de valeur

17.4.1 La production "EncodeByValueMapping" est la suivante:

```
EncodeByValueMapping ::=
    "{"
    USE
    DefinedOrBuiltinEncodingClass
    MAPPING
    ValueMapping
```

```
                WITH
                ValueMappingEncodingObjects
                "}"

        ValueMappingEncodingObjects ::=
            EncodingObject                          |
            DefinedOrBuiltinEncodingObjectSet
```

17.4.2 La production "DefinedOrBuiltinEncodingClass" et sa sémantique sont définies au § 10.9.1. Ce doit être une structure de codage définie par l'usager ou une classe intégrée dans le groupe catégoriel des champs binaires (voir § 16.1.7).

17.4.3 La production "ValueMapping" est spécifiée dans le § 19.1.7, et doit être un mappage de valeurs associées à la classe de codage gouvernante sur la classe identifiée par la classe "DefinedOrBuiltinEncodingClass". La classe de codage gouvernante doit appartenir au groupe catégoriel des champs binaires.

17.4.4 La production "ValueMappingEncodingObjects" spécifie le codage de la classe "DefinedOrBuiltinEncodingClass". La production "EncodingObject" doit définir un objet de codage au moyen de la notation gouvernée par cette classe, ou par une classe à laquelle cet objet peut être déréférencé (voir § 17.1.3). La production "DefinedOrBuiltinEncodingObjectSet" peut en variante être utilisée afin de spécifier le codage de la classe "DefinedOrBuiltinEncodingClass" et doit contenir suffisamment d'objets de codage pour spécifier entièrement le codage de cette classe au moyen de l'application des codages spécifiés dans l'article 13.

17.4.5 La syntaxe pour "EncodingObject" autorise aussi bien la définition en ligne d'objets de codage (application récursive de cette clause de correspondance) et l'utilisation de noms de référence. (Le § D.2.9.3 indique un exemple de définition en ligne pour effectuer deux mappages de valeur dans une même attribution.)

17.4.6 Lorsque la production "EncodingObject" nécessite la présence d'une **REFERENCE**, celle-ci ne peut être fournie dans cette construction qu'au moyen d'un paramètre fictif de l'objet de codage qui est défini.

17.4.7 S'il y a des limites ou contraintes effectives de longueur applicables à des champs de la classe "DefinedOrBuiltinEncodingClass" et que les spécifications contenues dans l'article 19 nécessitent que les valeurs soient mappées sur ceux des champs qui violent les limites ou contraintes effectives de longueur spécifiées, dans ce cas de telles valeurs ne sont pas mappées, et le codage de telles valeurs n'est pas possible. Il y a erreur de notation ECN ou d'application si de telles valeurs sont soumises pour codage.

17.4.8 Si la variante "EncodingObject" de la production "ValueMappingEncodingObjects" est utilisée, dans ce cas la référence "encodingobjectreference" définie fait apparaître un pointeur d'identification si et seulement si la variante "EncodingObject" fait apparaître ce pointeur d'identification. Si la variante "DefinedOrBuiltinEncodingObjectSet" de la production "ValueMappingEncodingObjects" est utilisée afin de définir le codage de la classe "DefinedOrBuiltinEncodingClass", dans ce cas la détermination du fait que la référence "encodingobjectreference" fait apparaître un pointeur d'identification est conforme au § 17.3.6.

17.5 Codage d'une structure de codage

17.5.1 La production "EncodeStructure" est la suivante:

```
EncodeStructure ::=
                "{"
                ENCODE STRUCTURE
                "{"
                ComponentEncodingList
                StructureEncoding ?
                "}"
                CombinedEncodings ?
                 "}"

StructureEncoding ::=
                STRUCTURED WITH
               TagEncoding ?
                    EncodingOrUseSet

TagEncoding ::= "[" EncodingOrUseSet "]"

EncodingOrUseSet ::=
                    EncodingObject    |
                    USE-SET
```

17.5.2 La production "EncodeStructure" peut être utilisée afin de définir un codage seulement si la classe de codage gouvernante déréférence à une construction définie au moyen d'un constructeur de codage dans les catégories des

options, des concaténations ou des répétitions, ou à une construction définie au moyen d'une de ces catégories précédée par une classe appartenant à la catégorie des étiquettes. Ce constructeur de codage est appelé le constructeur de codage gouvernant.

17.5.3 La production "StructureEncoding", si elle est présente, doit définir un codage pour le constructeur de codage gouvernant et pour toute classe précédente dans la catégorie des étiquettes qui précède le constructeur du codage de gouverneur. Si la production est absente, la production "CombinedEncodings" doit être présente et doit contenir des objets de codage qui peuvent coder le constructeur de codage gouvernant et toute classe précédente dans la catégorie des étiquettes; sinon la spécification ECN est en erreur.

NOTE – La production "CombinedEncodings" doit être présente si la production "StructureEncoding" est absente, parce qu'un codage complet doit être produit. Si l'on souhaite différer la spécification d'une partie d'un codage, dans ce cas un paramètre fictif devrait être utilisé.

17.5.4 L'objet de codage appliqué au constructeur de codage gouvernant (que ce soit à partir de **`STRUCTURED WITH`** ou à partir de "CombinedEncodings") ne doit spécifier aucune action de remplacement.

17.5.5 Si l'ensemble "EncodingOrUseSet" dans la production "StructureEncoding" est un objet "EncodingObject", il doit être gouverné par le constructeur de codage gouvernant.

17.5.6 Si la production **`USE-SET`** est spécifiée dans un quelconque ensemble "EncodingOrUseSet", dans ce cas le codage de la classe correspondante est obtenu par l'application de la production "CombinedEncodings", qui doit être présente, et doit être suffisant pour coder la classe correspondante; sinon la spécification ECN est en erreur.

17.5.7 La production "ComponentEncodingList" est la suivante:

```
ComponentEncodingList ::=
    ComponentEncoding "," *

ComponentEncoding ::=
    NonOptionalComponentEncodingSpec    |
    OptionalComponentEncodingSpec
```

17.5.8 Il doit y avoir au plus une production "ComponentEncoding" pour chaque composant du constructeur de codage gouvernant. Les codages "ComponentEncodings" doivent être dans le même ordre textuel.

NOTE – L'absence de codages "ComponentEncoding" peut être détectée par les champs nommés suivants, ou par l'extrémité de la liste "ComponentEncodingList".

17.5.9 La production "OptionalComponentEncodingSpec" ne doit être utilisée que si et seulement si le composant est facultatif (c'est-à-dire contient une classe de codage dans la catégorie des offres d'options).

17.5.10 Si la production "ComponentEncoding" pour tout composant n'est pas présente dans la liste "ComponentEncodingList", dans ce cas la production "CombinedEncodings" doit être présente (mais voir également § 17.5.6) et doit, lors de son application au composant (voir § 13.2), offrir un codage complet de ce composant (éventuellement avec utilisation de paramètres fictifs); sinon il y a erreur dans la spécification ECN.

```
NonOptionalComponentEncodingSpec ::=
        identifier ?
        TagAndElementEncoding

OptionalComponentEncodingSpec ::=
        identifier
        TagAndElementEncoding
        OPTIONAL-ENCODING
        OptionalEncoding

TagAndElementEncoding ::=
        TagEncoding ?
        EncodingOrUseSet

OptionalEncoding ::= EncodingOrUseSet
```

17.5.11 La production "identifier" doit être l'identificateur "identifier" du composant du constructeur de codage gouvernant. L'identificateur "identifier" contenu dans "NonOptionalComponentEncodingSpec" ne doit être omis que si et seulement si le constructeur de codage gouvernant est une classe de la catégorie des répétitions pour laquelle il n'y a pas d'identificateur concernant l'élément répété.

17.5.12 La production "TagAndElementEncoding" contenue dans la production "ComponentEncoding" doit fournir un codage complet pour le composant (y compris toute classe de la catégorie des étiquettes qui est préfixée à l'élément, mais à l'exclusion de toute classe de la catégorie des offres d'options faisant suite à cet élément).

17.5.13 Les productions "EncodingObject" contenues dans les productions "EncodingOrUseSet" de la production "TagAndElementEncoding" doivent être gouvernées par les classes de codage correspondantes dans le composant. Si une production "EncodingOrUseSet" est **USE-SET**, alors le codage est obtenu par l'application de la production "CombinedEncodings" (qui doit être présente).

17.5.14 La production "EncodingOrUseSet" contenue dans "OptionalEncoding" doit coder complètement la classe de la catégorie des offres d'options du composant. Si une production "EncodingOrUseSet" est **USE-SET**, alors le codage de la classe de la catégorie des offres d'options est obtenu par l'application de la production "CombinedEncodings" (qui doit être présente).

17.5.15 Si une référence **REFERENCE** est nécessaire en tant que paramètre réel d'un quelconque des objets de codage ou ensembles d'objets de codage utilisés dans cette production, dans ce cas elle peut soit être fournie en tant que paramètre fictif de l'objet de codage qui est défini, ou être fournie en tant que n'importe lequel des identificateurs "identifier" qui sont textuellement présents dans la construction obtenue par déréférencement du gouverneur. Si le constructeur de codage gouvernant est une classe de la catégorie des répétitions, le paramètre réel pour la référence **REFERENCE** peut être tout identificateur textuellement présent dans la définition de la structure "EncodingStructure" contenue dans la structure "RepetitionStructure" de la répétition. Si la référence **REFERENCE** est requise afin de désigner un conteneur, elle peut également être fournie comme suit:

a) **STRUCTURE** (à condition que le constructeur de la structure codée ne soit pas une catégorie d'options) lorsque cette référence renvoie à cette structure;

b) **OUTER** lorsque cette référence renvoie au conteneur du codage complet.

NOTE – La production "EncodeStructure" est la seule production dans laquelle des références **REFERENCE** peuvent être fournies, sauf au moyen de paramètres fictifs ou de la construction **OUTER**, ou lorsque les références visent un **flag-to-be-used**" ou un **flag-to-be-set** dans la définition d'un objet de codage pour une classe de la catégorie des répétitions qui utilise le remplacement.

17.5.16 La détermination du fait que la référence "encodingobjectreference" définie fait apparaître un pointeur d'identification est conforme au § 17.3.6.

17.6 Codage-décodage différentiel

17.6.1 La production "DifferentialEncodeDecodeObject" est la suivante:

```
DifferentialEncodeDecodeObject ::=
        "{"
        ENCODE-DECODE
        SpecForEncoding
        DECODE AS IF
        SpecForDecoders
        "}"

SpecForEncoding ::= EncodingObject

SpecForDecoders ::= EncodingObject
```

17.6.2 La production "DifferentialEncodingObject" spécifie des règles afin de coder des valeurs abstraites associées à la classe du gouverneur de cette notation et (séparément) des règles à utiliser par des décodeurs afin de récupérer des valeurs abstraites à partir de codages censés avoir été produits par des objets de codage de la classe du gouverneur.

17.6.3 La production "SpecForEncoding" doit être appliquée par des codeurs. Les décodeurs ne doivent décoder que si le codeur a appliqué la spécification "SpecForDecoders".

NOTE 1 – La production "SpecForDecoders" est encore une spécification de codage. Elle indique aux décodeurs l'hypothèse que des codeurs ont utilisé cette spécification.

NOTE 2 – Le comportement des décodeurs qui décodent dans l'hypothèse qu'un codeur a utilisé la spécification "SpecForDecoders", mais qui détectent des erreurs de codage, n'est pas normalisé.

17.6.4 La production "SpecForEncoding" et les objets de codage "SpecForDecoders" ne doivent pas avoir été définis au moyen de **ENCODE-DECODE**, de même que les éventuels objets de codage utilisés dans leur définition ne doivent pas avoir été définis au moyen de **ENCODE-DECODE**.

NOTE – Cette restriction est présente parce que sinon la spécification de la signification de la construction **ENCODE-DECODE** deviendrait plus complexe sans fonctionnalité ajoutée.

17.6.5 La production "encodingobjectreference" qui est définie ne fait apparaître un pointeur d'identification que si et seulement si le même pointeur d'identification est affiché par les productions "SpecForEncoding" et "SpecForDecoders".

17.7 Options de codage

17.7.1 La production "EncodingOptionsEncodingObject" est la suivante:

```
EncodingOptionsEncodingObject ::=
    "{"
       OPTIONS
       EncodingOptionsList
       WITH AlternativesEncodingObject
    "}"

EncodingOptionsList ::= OrderedEncodingObjectList

AlternativesEncodingObject ::= EncodingObject
```

17.7.2 La production "EncodingOptionsEncodingObject" spécifie que le codeur peut coder (sous réserve du § 17.7.5) au moyen de n'importe lequel des objets "EncodingObject" contenus dans la liste "EncodingOptionsList". Ces objets "EncodingObject" doivent tous être objets de la classe gouvernante.

NOTE – Il est fortement recommandé que les nouvelles implémentations effectuent le codage au moyen du plus proche objet "EncodingObject" de la liste ordonnée qui soit capable de coder la valeur abstraite à coder (voir § 17.7.5). La spécification des options de codage n'est fournie que parce qu'il est nécessaire de refléter les options offertes dans les protocoles existants et de prendre en charge différentes formes de codage de longueur pour chaînes. Toutes les options de codage peuvent, évidemment, apparaître lors du décodage.

17.7.3 La production "AlternativesEncodingObject" doit être un objet de codage d'une classe quelconque dans la catégorie des options. Les codeurs et les décodeurs ne doivent utiliser les codages et procédures spécifiés par cet objet de codage que si les options de codage étaient des codages pour composants d'une instance de cette classe. L'objet "AlternativesEncodingObject" ne doit pas contenir de spécification **`REPLACE`** (voir § 23.1.1). Le paramètre **`DETERMINED BY`** doit être mis à la valeur **`handle`**.

NOTE – Si l'objet "AlternativesEncodingObject" est paramétré avec un champ de référence, dans ce cas la référence "encodingobjectreference" qui est définie doit être paramétrée avec un champ de référence fictif qui est utilisé comme paramètre réel pour l'objet "AlternativesEncodingObject".

17.7.4 Tous les objets "EncodingObject" contenus dans la liste "EncodingOptionsList" doivent offrir ce pointeur d'identification.

17.7.5 Le codeur doit limiter son choix d'objets "EncodingObject" dans la liste "EncodingOptionsList" à ceux qui fournissent des codages pour la valeur abstraite réelle en cours de codage. Il y a erreur de spécification ou d'application ECN s'il n'y a pas au moins un tel objet "EncodingObject" pour toute valeur abstraite qui doit être codée.

NOTE 1 – Il est possible que les ensembles de valeurs abstraites codées par les objets "EncodingObject" de la liste "EncodingOptionsList" soient disjoints. Cela n'est pas une erreur et peut être une façon pratique de spécifier différentes structures afin de coder différentes étendues de valeurs abstraites de la classe gouvernante, par exemple des codages de forme courte et de forme longue où la forme courte est obligatoire pour les petites valeurs.

NOTE 2 – Il est possible d'utiliser un objet de codage d'options de codage en tant que spécification "SpecForDecoders" (voir § 17.6), lorsque la production "SpecForEncoding" est un objet de codage d'options de codage qui contient exactement une des options contenues dans la spécification "SpecForDecoders". C'est une autre approche de l'extensibilité.

17.8 Définition non ECN d'objets de codage

17.8.1 La production "NonECNEncodingObject" est la suivante:

```
NonECNEncodingObject::=
    NON-ECN-BEGIN
    AssignedIdentifier
    anystringexceptnonecnend
    NON-ECN-END
```

17.8.2 La production "NonECNEncodingObject" doit spécifier un objet de codage de la classe des gouverneurs (voir § 17.1.3). La notation utilisée à cette fin est contenue dans "anystringexceptnonecnend" et n'est pas normalisée.

17.8.3 La production "AssignedIdentifier" et sa sémantique sont définies dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 12.1, telle que modifiée par le § A.1 de la présente Recommandation | Norme internationale. Elle identifie la notation utilisée dans la production "anystringexceptuserdefinedend" afin de spécifier le codage.

17.8.4 Si l'option "empty" de l'identificateur "AssignedIdentifier" est utilisée, dans ce cas la notation est déterminée par des moyens qui sont en dehors de la présente Recommandation | Norme internationale.

17.8.5 L'attribution d'identificateurs d'objet à toute notation pour utilisation dans la production "anystringexceptnonecnend" suit les règles normales pour l'attribution d'identificateurs d'objet comme spécifié dans les Recommandations UIT-T de la série X.660 | ISO/CEI 9834.

17.8.6 Un pointeur d'identification est affiché par la référence "encodingobjectreference" qui n'est définie que si et seulement si la production "anystringexceptnonecnend" spécifie qu'elle agit ainsi. Le moyen d'une telle spécification n'est pas défini dans la présente Recommandation | Norme internationale.

18 Attribution d'ensembles d'objets de codage

18.1 Généralités

18.1.1 La production "EncodingObjectSetAssignment" est la suivante:

```
EncodingObjectSetAssignment ::=
    encodingobjectsetreference
    #ENCODINGS
    "::="
    EncodingObjectSet
    CompletionClause ?

EncodingObjectSet ::=
    DefinedOrBuiltinEncodingObjectSet |
    EncodingObjectSetSpec
```

18.1.2 La notation "EncodingObjectSet" est gouvernée par le mot réservé **`#ENCODINGS`**, et doit répondre aux conditions indiquées ci-dessous.

18.1.3 Il ne doit y avoir aucune définition récursive (voir 3.2.38) d'une référence "encodingclassreference" et il ne doit y avoir aucune instanciation récursive (voir 3.2.39) d'une référence "encodingclassreference".

18.1.4 La production "DefinedOrBuiltinEncodingObjectSet" est définie au § 10.9.3.

18.1.5 La production "EncodingObjectSetSpec" est la suivante:

```
EncodingObjectSetSpec ::=
        "{"
    EncodingObjects UnionMark *
        "}"

EncodingObjects ::=
    DefinedEncodingObject  |
    DefinedEncodingObjectSet

UnionMark ::=
    "|" |
UNION
```

18.1.6 La production "EncodingObjectSetSpec" définit un ensemble d'objets de codage au moyen d'un ou de plusieurs objets de codage ou ensembles d'objets de codage.

18.1.7 Les objets de codage formant un ensemble d'objets de codage doivent tous être de classes distinctes de codage et ne doivent pas être des classes contenues dans le groupe catégoriel des procédures de codage à moins qu'ils ne soient de la classe **`#OUTER`** (voir § 16.1.13).

NOTE – Un ensemble d'objets de codage est utilisé afin de définir d'autres ensembles d'objets de codage, afin de définir des objets de codage dans le module EDM, et afin d'effectuer une importation dans le module ELM pour l'application de codages.

18.1.8 Si la clause "CompletionClause" est présente, l'ensemble d'objets de codage défini par la production "EncodingObjectSetSpec" est considéré comme des codages "PrimaryEncodings" (voir § 13.2) et l'ensemble d'objets de codage attribué à la référence "encodingobjectsetreference" est l'ensemble d'objets de codage combinés formé comme spécifié dans le § 13.2.

18.2 Ensembles d'objets de codage intégrés

18.2.1 La production "BuiltinEncodingObjectSetReference" est la suivante:

```
BuiltinEncodingObjectSetReference ::=
        PER-BASIC-ALIGNED
    |   PER-BASIC-UNALIGNED
    |   PER-CANONICAL-ALIGNED
    |   PER-CANONICAL-UNALIGNED
    |   BER
    |   CER
    |   DER
```

18.2.2 Ces noms d'ensemble d'objets de codage font référence aux ensembles d'objets de codage définis par la Rec. UIT-T X.690 | ISO/CEI 8825-1 et par la Rec. UIT-T X.691 | ISO/CEI 8825-2. Les identificateurs d'objet pour le codage des règles fournissant ces ensembles d'objets de codage sont donnés dans le Tableau 4.

NOTE – Ces Recommandations | Normes internationales ont été rédigées avant la présente Recommandation | Norme internationale ECN. Elles n'utilisent pas la terminologie des objets de codage. Elles définissent, par exemple, la façon dont un type ASN.1 **INTEGER** ou **BOOLEAN** doit être codé, ce qui devrait être interprété comme la définition d'un objet de codage de classe **#INTEGER** ou **#BOOLEAN**.

Tableau 4 – Noms d'ensemble d'objet de codage intégré et identificateurs d'objet associés

PER-BASIC-ALIGNED	**{joint-iso-itu-t(2) asn1(1) packed-encoding(3) basic(0) aligned(0)}**
PER-BASIC-UNALIGNED	**{joint-iso-itu-t(2) asn1(1) packed-encoding(3) basic(0) unaligned(1)}**
PER-CANONICAL-ALIGNED	**{joint-iso-itu-t(2) asn1(1) packed-encoding(3) canonical(1) aligned(0)}**
PER-CANONICAL-UNALIGNED	**{joint-iso-itu-t(2) packed-encoding(3) canonical(1) unaligned(1)}**
BER	**{joint-iso-itu-t(2) asn1(1) basic-encoding(1)}**
CER	**{joint-iso-itu-t(2) asn1(1) ber-derived(2) canonical-encoding(0)}**
DER	**{joint-iso-itu-t(2) asn1(1) ber-derived(2) distinguished-encoding(1)}**

18.2.3 Ces ensembles d'objets de codage sont chacun un ensemble complet d'objets de codage qui peuvent être appliqués à toute structure de codage (soit produite implicitement à partir d'un type ASN.1 ou définie par l'usager) afin de spécifier les codages BER ou PER correspondants.

NOTE – Il est possible d'ajouter à un tel ensemble un objet de codage pour une classe de codage définie par l'utilisateur ou générée implicitement; un tel objet aura la précédence sur tout autre codage qui pourrait être obtenu par déréférencement.

18.2.4 Les ensembles ci-dessus contiennent tous les objets de codage pour les classes utilisées dans des structures de codage produites implicitement (voir § 11.2) qui sont différentes pour chaque ensemble de règles de codage. Ils contiennent également chacun des objets de codage identiques pour les classes **#INT**, **#BOOL**, **#NUL**, **#CHARS**, **#OCTETS**, **#BITS**, **#CONCATENATION**. Ils **ne** contiennent **pas** d'objets de codage pour les classes **#ALTERNATIVES**, **#REPETITION** et **#PAD**.

18.2.5 Ces classes de codage constituent des éléments fondamentaux de construction de codages. Elles sont codées simplement par tous les ensembles ci-dessus d'objets de codage intégrés. Les objets de codage pour ces classes spécifient les codages comme suit:

18.2.5.1 La classe **#INT** est codée en tant que classe **#INTEGER** de type **PER-BASIC-UNALIGNED**, à condition qu'elle soit bornée. Il y a erreur de conception de la notation ECN si la classe **#INT** ne possède pas une limite inférieure aussi bien que supérieure lorsque cet objet de codage est appliqué à la classe **#INT**.

18.2.5.2 Les classes **#BOOL** et **#NUL** sont codées respectivement en tant que classes **#BOOLEAN** et **#NULL** de type **PER-BASIC-UNALIGNED**.

18.2.5.3 Les classes **#CHARS**, **#OCTETS** et **#BITS** sont codées respectivement en tant que classes **UTF8String**, **#OCTET-STRING** et **#BIT-STRING** de type **PER-BASIC-UNALIGNED**, à condition qu'elles soient de même longueur. Il y a erreur de conception de la notation ECN si les classes **#CHARS**, **#OCTETS** ou **#BITS** ne sont pas soumises à une contrainte de longueur effective les limitant à une seule longueur.

18.2.5.4 La classe **#CONCATENATION** est codée en tant que classe **#SEQUENCE** de type **PER-BASIC-UNALIGNED** sans composants facultatifs. Si ces objets de codage sont appliqués à une classe **#CONCATENATION** avec composants facultatifs, il y a erreur de spécification ECN.

18.2.6 Les objets de codage de la classe **#OPEN-TYPE** objets de codage dans les ensembles d'objets de codage intégrés soumis aux règles BER, CER, et DER ne produisent aucun codage additionnel pour la classe **#OPEN-TYPE**. Lorsque ces objets de codage sont appliqués à une classe de la catégorie des types ouverts, il y a erreur de spécification

ECN si les codages des valeurs du type choisi (dans une instance de communication) pour utilisation avec la classe **#OPEN-TYPE** ne sont pas autodélimitateurs.

NOTE – L'ensemble d'objets de codage combinés appliqués au type choisi pour utilisation avec la classe **#OPEN-TYPE** est toujours le même que l'ensemble d'objets de codage combinés appliqué à la classe **#OPEN-TYPE** (voir § 13.2.10.5).

19 Mappage de valeurs

19.1 Généralités

19.1.1 Cet article spécifie la syntaxe permettant le mappage de valeurs (et de numéros d'étiquettes) qui doivent être codées par les champs d'une structure de codage (qui peut être une structure de codage produite ou toute autre structure de codage) vers les champs d'une autre structure de codage.

NOTE – La puissance offerte par une utilisation simple de cette notation a été limitée (afin d'éviter toute complexité). Des mappages plus complexes peuvent être réalisés au moyen de multiples instances de la production "EncodeByValueMapping" (voir § 17.4 et l'exemple au § D.1.10.2). Ces mécanismes de mappages peuvent être étendus et généralisés, mais cela ne sera effectué que si d'autres besoins d'usager sont identifiés.

19.1.2 Lors de la spécification de la notation "EncodeByValueMapping" (voir § 17.4.1), la structure à laquelle est déréférencée la classe "DefinedOrBuiltinEncodingClass" contenue dans la production "EncodingObjectAssignment" (voir § 17.1.1), dont elle fait partie, est appelée *gouverneur source* ou *classe de codage source* (selon le contexte). La structure à laquelle est déréférencée la classe "DefinedOrBuiltinEncodingClass" contenue dans la production "EncodeByValueMapping" est elle-même appelée *gouverneur cible* ou *classe de codage cible* (selon le contexte).

19.1.3 Si le gouverneur source a une classe initiale de la catégorie des étiquettes, dans ce cas ce gouverneur cible doit avoir une classe initiale dans la catégorie des étiquettes et le numéro d'étiquette de la classe contenue dans le gouverneur source est mappé sur le numéro d'étiquette de la classe de la catégorie des étiquettes dans le gouverneur cible. Si la classe de la catégorie des étiquettes contenue dans le gouverneur cible a un numéro d'étiquette associé, dans ce cas il y a erreur de spécification ECN si ce numéro diffère du numéro d'étiquette mappé à partir du gouverneur source.

19.1.4 Si le gouverneur source ne possède pas de classe initiale dans la catégorie des étiquettes, dans ce cas le gouverneur cible n'est pas tenu d'offrir une classe initiale dans la catégorie des étiquettes, mais s'il le fait, dans ce cas il doit y avoir un numéro d'étiquette associé à cette étiquette dans la définition du gouverneur cible.

19.1.5 L'effet de la présence d'une classe initiale de la catégorie des étiquettes dans les gouverneurs sources ou cibles est complètement déterminé par les § 19.1.3 et 19.1.4, et le texte suivant ne tient pas compte de la présence possible de telles classes.

19.1.6 Les codages spécifiés pour les valeurs mappées sur la classe de codage cible deviennent les codages de ces valeurs dans la classe de codage source.

NOTE 1 – Si la spécification ECN totale mappe seulement certaines des valeurs à partir d'un type ASN.1 sur des codages, cela n'est pas une erreur. C'est une contrainte imposée par la notation ECN aux valeurs qui peuvent être utilisées par l'application. De telles contraintes devraient normalement être identifiées par des commentaires, soit dans la spécification ASN.1 ou dans la spécification ECN (voir § 17.4.7).

NOTE 2 – Si la spécification ECN totale mappe deux valeurs dans le même codage produit par un seul objet de codage, dans ce cas il s'agit d'une erreur de spécification ECN. De telles erreurs peuvent être détectées par les utilitaires de la notation ECN, mais les règles permettant de les éviter ne sont pas complètes dans la présente Recommandation | Norme internationale. La responsabilité en reste confiée à l'usager de la notation ECN.

19.1.7 La production "ValueMapping" est la suivante:

```
ValueMapping ::=
        MappingByExplicitValues
    |   MappingByMatchingFields
    |   MappingByTransformEncodingObjects
    |   MappingByAbstractValueOrdering
    |   MappingByValueDistribution
    |   MappingIntToBits
```

NOTE – Toutes les instances de cette syntaxe sont précédées par le mot réservé **MAPPING**. (Les § D.1.2.2, D.1.4.2, D.1.10.2, D.2.1.3 et l'Annexe E donnent des exemples d'une définition de codages au moyen de chacun de ces mappages de valeur.)

19.1.8 Les productions "ValueMapping" sont spécifiées comme suit:

MappingByExplicitValues	§ 19.2
MappingByMatchingFields	§ 19.3
MappingByTransformEncodingObjects	§ 19.4
MappingByAbstractValueOrdering	§ 19.5
MappingByValueDistribution	§ 19.6
MappingIntToBits	§ 19.7

NOTE – Il arrive fréquemment que plusieurs des mappages de valeur puissent être utilisés afin de définir le même codage, mais certains produiront une spécification plus évidente ou moins prolixe que d'autres. Il y a lieu que les concepteurs de notation ECN sélectionnent soigneusement la forme de mappage de valeur à utiliser.

19.2 Mappage par valeurs explicites

19.2.1 Cet article offre la notation visant à spécifier le mappage de valeurs entre différentes classes primitives de champ binaire de codage. (Le § D.1.10.2 donne un exemple.)

19.2.2 Cet article utilise la notation de valeurs ASN.1 spécifiée dans la Rec. UIT-T X.680 | ISO/CEI 8824-1 pour le type qui correspond à une classe de codage.

19.2.3 Le Tableau 5 spécifie la notation de valeur en notation ASN.1 à utiliser avec chaque classe de codage gouvernante. Dans chaque cas, la classe peut avoir ou ne pas avoir une contrainte associée de longueur ou d'étendue de valeurs.

19.2.4 La notation ECN prend en charge le mappage par valeurs explicites (à destination ou en provenance de la classe de codage) pour toutes les classes de codage des catégories énumérées dans la colonne 1 du Tableau 5. La colonne 2 de ce tableau spécifie la notation de valeur (soit comme production ASN.1 ou par référence à une clause de la Rec. UIT-T X.680 | ISO/CEI 8824-1 ou les deux) qui doit être utilisée lorsqu'une classe de codage de la catégorie énumérée dans la colonne 1 est spécifiée comme gouverneur de la notation. Elle spécifie également le paragraphe de la Rec. UIT-T X.680 | ISO/CEI 8824-1 qui définit la notation de valeur.

NOTE – Aucune des notations de valeur ASN.1 suivantes ne peut utiliser de valeurs "DefinedValue" (telles que définies dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 13.1) parce que les productions "valuereference" ne peuvent être ni importées ni définies dans un module EDM ou ELM.

Tableau 5 – Catégories de classes de codage et notations de valeur utilisées dans le mappage par valeurs explicites

Catégorie de classe de codage gouvernante	**Notation de valeur ASN.1**
chaînes de bits	**"bstring" ou "hstring"** (voir Rec. UIT-T X.680 | ISO/CEI 8824-1, § 11.10 et 11.12)
booléens	**"BooleanValue"** (voir Rec. UIT-T X.680 | ISO/CEI 8824-1, § 17.3)
chaînes de caractères	**"RestrictedCharacterStringValue"** (voir Rec. UIT-T X.680 | ISO/CEI 8824-1, § 37.8)
entiers	**"SignedNumber" (voir Rec. UIT-T X.680 | ISO/CEI 8824-1,** § 18.1**)**
néant	**"NullValue"** (voir Rec. UIT-T X.680 | ISO/CEI 8824-1, § 23.3)
identificateurs d'objet	**"DefinitiveIdentifier"** (voir § A.1)
chaînes d'octets	**"bstring" ou "hstring"** (voir Rec. UIT-T X.680 | ISO/CEI 8824-1, § 11.10 et 11.12)
réels	**"RealValue"** (voir Rec. UIT-T X.680 | ISO/CEI 8824-1, § 20.6)

19.2.5 La production "MappingByExplicitValues" est la suivante:

```
MappingByExplicitValues ::=
        VALUES
        "{"
        MappedValues "," +
        "}"
```

```
MappedValues ::=
        MappedValue1
        TO
        MappedValue2

MappedValue1 ::= Value

MappedValue2 ::= Value
```

19.2.6 La production "MappedValue1" doit être une notation de valeur gouvernée par le gouverneur source et "MappedValue2" doit être une notation de valeur gouvernée par le gouverneur cible (voir § 19.1.2). La valeur contenue dans la source spécifiée par "MappedValue1" est mappée sur la valeur contenue dans la cible spécifiée par "MappedValue2".

19.2.7 Il y a erreur de spécification ECN si "MappedValue2" est une valeur qui viole une contrainte de limite ou de longueur dans la cible.

19.3 Mappage par champs appariés

19.3.1 Ce mappage est assuré essentiellement afin de permettre le codage d'un type ASN.1 à définir comme le codage d'une structure de codage qui possède non seulement des champs correspondant aux composants du type, mais également des champs ajoutés afin de tenir compte des déterminants.

19.3.2 La production "MappingByMatchingFields" est la suivante:

```
MappingByMatchingFields ::=
        FIELDS
```

19.3.3 Si les classes de codage sources ou cibles sont des structures de codage définies par l'usager (voir § 9.2.2.3) ou des structures de codage produites, dans ce cas ces références sont résolues jusqu'à ce que la source et la cible commencent par un constructeur de codage. Si ce constructeur de codage contenu dans la cible est dans la catégorie des répétitions, dans ce cas le déréférencement du composant de ce constructeur de codage de répétition est effectué jusqu'à ce que le composant commence par un constructeur de codage. Les références situées à l'intérieur des structures résultantes ne sont pas résolues.

19.3.4 L'effet de la possible présence de classes de la catégorie des étiquettes sur le déréférencement initial de noms de classe "DefinedOrBuiltinEncodingClass" dans la source et cible a été entièrement spécifié aux § 19.1.3 à 19.1.5. Il y a erreur de spécification ECN si d'autres classes initiales de la catégorie des étiquettes sont introduites par l'application du § 19.3.3.

19.3.5 Après l'application du § 19.3.3, les classes de codage sources et cibles doivent commencer par le même constructeur de codage. Celui-ci doit être soit un constructeur de codage de la catégorie des concaténations, ou un constructeur de codage de la catégorie des répétitions. Si ce constructeur de codage est dans la catégorie des répétitions, son composant contenu dans la cible doit être une classe de la catégorie des concaténations. Aux fins du présent § 19.3, les structures de codage résultantes sont appelées respectivement *structures de codage sources et cibles*.

19.3.6 Les noms de champ des composants (de niveau supérieur) du constructeur de codage produits par l'application du § 19.3.3 à la source sont appelés *champs sources*.

NOTE – Les champs sources sont restreints aux champs de niveau supérieur d'une concaténation ou du composant d'une répétition. Cette restriction est imposée afin de faciliter l'implémentation de la notation ECN, et pourrait être allégée ultérieurement.

19.3.7 Les noms de champ des composants du constructeur de codage dans la catégorie des concaténations, produits par l'application du § 19.3.3 à la cible sont appelés *champs cibles potentiels*.

NOTE – Les champs cibles potentiels peuvent être soit les composants d'une concaténation de niveau supérieur, ou les composants d'une concaténation qui est le composant d'une répétition.

19.3.8 Pour chaque champ source, il doit y avoir un champ cible potentiel ayant le même nom de champ (le champ cible correspondant).

NOTE – Un composant d'une classe de répétition ne peut être mappé s'il contient un identificateur (correspondant à un homologue dans la cible). L'utilisation du mappage par champs appariés ne serait pas légale si l'identificateur était absent.

19.3.9 Un champ cible correspondant doit être un élément facultatif dans une concaténation si et seulement si son champ source est un élément facultatif dans une concaténation. La présence ou l'absence du champ source dans une valeur abstraite associée à la structure de codage source détermine la présence ou l'absence du champ cible contenu dans la structure de codage cible.

19.3.10 Si le champ source a une classe initiale dans la catégorie des étiquettes, dans ce cas le champ cible correspondant doit avoir une classe initiale dans la catégorie des étiquettes et le numéro d'étiquette de la classe dans le

champ source est mappé sur le numéro d'étiquette de la classe de la catégorie des étiquettes dans le champ cible correspondant. Si la classe de la catégorie des étiquettes dans le champ cible correspondant a un numéro d'étiquette associé, dans ce cas il y a erreur de spécification ECN si ce numéro diffère du numéro d'étiquette qui est mappé à partir du champ source.

19.3.11 Si le champ source ne possède pas de classe initiale dans la catégorie des étiquettes, dans ce cas le champ cible correspondant n'est pas tenu d'offrir une classe initiale dans la catégorie des étiquettes mais, s'il le fait, dans ce cas il doit y avoir un numéro d'étiquette associé à cette étiquette dans la définition du champ cible correspondant.

19.3.12 Sauf présence ou absence de classes dans les catégories des étiquettes et des offres d'options (comme spécifié aux § 19.3.9 à 19.3.11), le champ cible correspondant et le champ source doivent avoir la même classe de codage (voir § 17.1.7) ou doivent être définis au moyen de la même séquence d'items lexicaux, sans tenir compte des commentaires ni des spécifications d'espace libre et de limites.

19.3.13 Toutes les valeurs abstraites sont mappées à partir de chacun des champs sources sur les champs cibles correspondants. Des champs supplémentaires contenus dans la structure de codage cible n'acquièrent pas de valeurs abstraites. Dans une spécification ECN correcte, la valeur de tels champs doit être spécifiée par référence en tant que déterminant.

19.3.14 Si les constructeurs de codage sources et cibles sont des classes de la catégorie des répétitions, dans ce cas le nombre de répétitions dans la valeur abstraite associée à la structure de codage source est mappé sur le nombre de répétitions contenu dans la structure de codage cible.

19.3.15 Si un champ source a une contrainte de contenu associée, celle-ci est mappée en tant que contrainte de contenu associée au champ cible correspondant.

19.3.16 Si, en raison de la présence de limites ou de contraintes de longueur, il y a des valeurs dans le champ source qui ne sont pas présentes dans le champ cible correspondant, dans ce cas le § 17.4.7 s'applique.

19.4 Mappage par objets de codage de la classe `#TRANSFORM`

19.4.1 Ce mappage permet qu'un ou plusieurs objets de codage de la classe **`#TRANSFORM`** soient appliqués afin de produire le mappage.

19.4.2 La classe de codage **`#TRANSFORM`** est définie dans l'article 24. Elle permet de spécifier des objets de codage qui transformeront des valeurs abstraites sources en valeurs abstraites résultantes. Les règles permettant de former une liste ordonnée des transformées (pour créer la liste "OrderedTransformList") sont spécifiées dans l'article 24. La liste complète est définie afin de transformer une source en un résultat.

NOTE – Des exemples de mappages définis avec ces transformées sont donnés aux § D.1.2.2 et D.2.4.2. L'exemple du § D.1.6.3 montre l'utilisation de cette production afin de définir des codages BCD d'un entier ASN.1.

19.4.3 La production "MappingByTransformEncodingObjects" est la suivante:

```
MappingByTransformEncodingObjects ::=
    TRANSFORMS
    "{"
    OrderedTransformList
    "}"

OrderedTransformList ::= Transform "," +

Transform ::= EncodingObject
```

19.4.4 Tous les objets "EncodingObject" contenus dans la liste "OrderedTransformList" doivent être gouvernés par la classe de codage **`#TRANSFORM`**.

19.4.5 Les classes cibles et sources pour ce mappage (voir § 19.1.2) doivent être de la catégorie des chaînes de bits, des booléens, des chaînes de caractères, des entiers, ou des chaînes d'octets. La source de la première transformée dans la liste et le résultat de la dernière transformée dans la liste doivent s'accorder avec la catégorie de la source et de la cible comme spécifié au § 24.2.7.

19.4.6 Il y a erreur de spécification ou d'application ECN si une quelconque transformée "Transform" contenue dans la liste "OrderedTransformList" n'est pas réversible vers la valeur abstraite qui est mappée.

NOTE – L'article 24 spécifie, pour chaque transformée, les valeurs abstraites vers lesquelles cette transformée est réversible.

19.4.7 S'il y a des limites ou des contraintes effectives de longueur sur la classe cible de codage, le § 17.4.7 s'applique.

19.5 Mappage par séquencement de valeurs abstraites

19.5.1 Ce mappage permet de répartir des valeurs abstraites associées à des classes de codage simples dans les champs de structures de codage complexes, et de mapper les valeurs abstraites associées à des structures de codage complexes sur des classes de codage simples telles que **`#INT`**. Il autorise également le compactage de valeurs entières ou d'énumérations dans un ensemble contigu de valeurs entières (voir § D.1.4).

NOTE – Les numéros d'étiquette associés aux classes de la catégorie des étiquettes ne sont pas des valeurs abstraites.

19.5.2 La production "MappingByAbstractValueOrdering" est la suivante:

```
MappingByAbstractValueOrdering ::=
    ORDERED VALUES
```

19.5.3 Pour ce mappage, tous les noms de classe de codage sont déréférencés (de façon récursive), et le résultat doit être une classe de la catégorie néant, booléens, entiers ou réels, ou doit être une construction définie au moyen d'une classe de la catégorie des options, ou doit être une classe de la catégorie des concaténations possédant un seul composant non facultatif.

19.5.4 L'ensemble ordonné de valeurs peut être fini ou infini.

19.5.4.1 Un ensemble fini de valeurs abstraites ordonnées est défini pour les classes de codage appartenant aux catégories suivantes:

a) néant;

b) booléens;

c) entiers bornés;

d) réels contraints à un nombre de valeurs fini;

e) structure de codage définie au moyen de la catégorie des options, à condition que toutes les options aient un ordre fini défini;

f) une structure de codage définie au moyen de la catégorie des concaténations qui possède un seul composant non facultatif, à condition que le composant ait un ordre fini défini.

19.5.4.2 Un ensemble infini de valeurs abstraites ordonnées est défini pour les classes de codage appartenant aux catégories suivantes:

a) entiers, contraints à offrir une limite inférieure finie;

b) une structure de codage définie au moyen de la catégorie des options, à condition que toutes les options sauf la dernière soient définies de façon à offrir un ensemble fini de valeurs ordonnées, et la dernière option est définie de façon à offrir un ensemble infini de valeurs ordonnées;

c) une structure de codage définie au moyen de la catégorie des concaténations et possédant un seul composant non facultatif, à condition que ce composant soit défini de façon à offrir un ensemble infini de valeurs abstraites ordonnées.

19.5.5 Les classes de la catégorie néant ont une seule valeur abstraite. Les classes de la catégorie des booléens sont définies de façon à avoir la valeur **`TRUE`** avant **`FALSE`**. Les classes de la catégorie des entiers sont définies de façon que les valeurs entières supérieures viennent à la suite des valeurs entières inférieures. Les classes de la catégorie des réels sont définies de façon que les valeurs entières supérieures viennent à la suite des valeurs entières inférieures.

NOTE – Le nombre de valeurs abstraites associées à une classe de la catégorie des entiers n'est pas nécessairement fini.

19.5.6 Les éventuelles limites présentes dans la source ou la destination doivent être prises entièrement en compte lors de la détermination de l'ensemble ordonné de valeurs abstraites.

19.5.7 L'ordre des valeurs abstraites associées à une classe de la catégorie des options (dont toutes les options ont une séquence définie de valeurs abstraites) est défini comme étant les valeurs abstraites (ordonnées) à partir de la première option textuelle, suivie de celles de la deuxième option textuelle, et ainsi de suite jusqu'à la dernière option textuelle.

19.5.8 L'ordre des valeurs abstraites associées à une classe de la catégorie des concaténations qui possède un composant isolé non facultatif doit être celui qui est déterminé par l'ordre des valeurs abstraites de ce composant isolé.

19.5.9 Le mappage est défini à partir des valeurs abstraites contenues dans la première classe de codage jusqu'aux valeurs abstraites contenues dans la seconde classe de codage par leur position dans l'ordre ci-dessus.

19.5.10 Noter que les règles ci-dessus garantissent qu'il y a une première valeur définie dans chaque ordre, et une valeur suivante définie. Il n'est pas nécessaire qu'il y ait une dernière valeur définie (un des deux ensembles ou les deux peuvent être infinis).

19.5.11 Si le nombre de valeurs abstraites dans l'ordre de destination est inférieur au nombre de valeurs abstraites contenues dans l'ordre d'origine, cela n'est pas une erreur. Cependant, la spécification ECN sera incapable de coder certaines des valeurs abstraites de la spécification ASN.1 et cela devrait être indiqué par des commentaires soit dans la spécification ASN.1 ou dans la spécification ECN.

19.5.12 Si le nombre de valeurs abstraites dans l'ordre de destination dépasse celui de l'ordre d'origine, il peut y avoir certains codages définis par notation ECN qui n'ont aucune valeur abstraite ASN.1 et qui ne seront jamais produits.

19.5.13 Ce mappage peut également être appliqué dans tous les cas où les seules valeurs abstraites contenues dans la structure cible sont celles qui sont associées à une même instance de la même classe que la structure source.

NOTE – Ce cas se produirait si la structure cible était la même que la structure source précédée par une ou plusieurs instances de classes de la catégorie des étiquettes.

19.5.14 Des classes de la catégorie des étiquettes peuvent être présentes dans la structure cible, mais doivent avoir un numéro d'étiquette associé spécifié dans la définition de structure. Leur présence n'a pas d'incidence sur le mappage de valeurs abstraites.

19.6 Mappage par distribution de valeurs

19.6.1 Ce mappage prend des étendues de valeurs à partir d'une classe de codage de la catégorie des entiers, et met chaque étendue en correspondance avec un champ d'entier différent dans une structure de codage plus complexe. Les champs qui ne reçoivent pas de valeurs abstraites doivent avoir leur valeur définie par l'application de déterminants.

19.6.2 Tous les noms de structure de codage sont déréférencés (de façon récursive) avant l'application de ce mappage.

19.6.3 La classe de codage source doit donc être une classe de la catégorie des entiers, éventuellement avec une classe précédente de la catégorie des étiquettes qui est mappée conformément aux § 19.1.3 à 19.1.5.

19.6.4 La classe de codage cible peut être toute structure de codage, et peut contenir des classes de la catégorie des étiquettes, mais tous les noms de champ contenus dans la structure entière de codage doivent être distincts, et toutes les classes de la catégorie des étiquettes contenues dans la cible (sauf celles qui sont mappées par le § 19.6.3) doivent avoir un numéro d'étiquette dans leur définition et sont sinon ignorées dans le mappage.

19.6.5 Les valeurs doivent être mappées seulement sur des champs contenus dans la structure cible qui sont des classes de la catégorie des entiers, éventuellement précédés par des classes de la catégorie des étiquettes (voir § 19.6.4), et éventuellement avec des limites.

La production "MappingByValueDistribution" est la suivante:

```
MappingByValueDistribution ::=
    DISTRIBUTION
    "{"
    Distribution "," +
    "}"

Distribution ::=
    SelectedValues
    TO
    identifier

SelectedValues ::=
        SelectedValue
    |   DistributionRange
    |   REMAINDER

DistributionRange ::=
     DistributionRangeValue1
    ".."
    DistributionRangeValue2

SelectedValue ::= SignedNumber

DistributionRangeValue1 ::= SignedNumber
DistributionRangeValue2 ::= SignedNumber
```

19.6.7 La production "SignedNumber" est spécifiée dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 18.1.

19.6.8 La production "DistributionRangeValue1" doit être inférieure à la production "DistributionRangeValue2".

19.6.9 La valeur spécifiée par "SelectedValue" dans "SelectedValues", ou l'ensemble de valeurs supérieures ou égales à "DistributionRangeValue1" et inférieures ou égales à "DistributionRangeValue2" sont mappés sur le champ spécifié par "identifier".

19.6.10 Le mot réservé **REMAINDER** ne doit être utilisé qu'une seule fois pour la dernière production "SelectedValues". Il spécifie toutes les valeurs abstraites de la classe de codage source qui n'ont pas été distribuées par des productions "SelectedValues" antérieures.

19.6.11 Une valeur ne doit pas être mappée sur plusieurs champs cibles, mais plusieurs productions "SelectedValues" peuvent avoir la même destination.

19.6.12 S'il y a des limites applicables aux champs cibles, dans ce cas le § 17.4.7 s'applique.

19.6.13 Si une valeur issue de la source est mappée dans un champ contenu dans la cible dont la présence dépend d'offres d'options ou de choix d'options ou les deux, cela n'est pas une erreur, mais les offres d'options et les choix d'options contenus dans la cible (lors du codage de telles valeurs) doivent être tels que le codage de la cible contienne les champs cibles.

19.7 Mappage de valeurs entières sur des bits

19.7.1 Ce mappage prend des valeurs isolées ou des étendues de valeurs dans une classe de codage de la catégorie des entiers (éventuellement précédée par des classes de la catégorie des étiquettes comme spécifié aux § 19.1.3 à 19.1.5) et met chaque valeur entière en correspondance avec une valeur de chaîne de bits (éventuellement précédée par des classes de la catégorie des étiquettes).

NOTE – Ce mappage est prévu afin de prendre en charge des codages d'entiers autodélimitateurs, tels que les codages de Huffman. (Voir dans l'Annexe E une analyse plus approfondie et des exemples de codages de Huffman.)

19.7.2 La classe de codage source doit être une classe de la catégorie des entiers, éventuellement précédée par des classes de la catégorie des étiquettes.

19.7.3 La classe de codage de destination doit être de la catégorie des chaînes de bits, éventuellement précédée par des classes de la catégorie des étiquettes.

19.7.4 Les classes de la catégorie des étiquettes sont mappées comme spécifié aux § 19.1.3 à 19.1.5.

19.7.5 La production "MappingIntToBits" est la suivante:

```
MappingIntToBits ::=
    TO BITS
    "{"
    MappedIntToBits "," +
    "}"

MappedIntToBits ::=
    SingleIntValMap  |
    IntValRangeMap
```

19.7.6 Chaque production "SingleIntValMap" mappe une valeur entière isolée à une valeur de chaîne de bits isolée.

19.7.7 Chaque production "IntValRangeMap" mappe une étendue de valeurs entières contiguës et croissantes sur une étendue de valeurs de chaîne de bits contiguës et croissantes.

19.7.8 Les valeurs de chaîne de bits sont définies comme étant contiguës si:

a) elles ont toutes la même longueur en bits;

b) lorsque interprétées en tant que valeurs positives entières, les valeurs entières correspondantes sont contiguës et croissantes.

19.7.9 Seules les valeurs spécifiées dans le mappage sont codables. Les autres valeurs abstraites de la source ne sont pas mappées et ne peuvent pas être codées par l'objet de codage défini par l'attribution d'objet de codage au moyen de cette construction. Il y a erreur de notation ECN ou d'application si de telles valeurs sont présentées à un codeur.

NOTE – Cette limitation du codage devrait être représentée par des contraintes sur le type ASN.1 auquel elles sont appliquées, ou par des commentaires dans la spécification ASN.1.

19.7.10 La production "SingleIntValMap" est la suivante:

```
SingleIntValMap ::=
    IntValue
    TO
    BitValue
```

```
IntValue ::= SignedNumber
BitValue ::=
    bstring   |
    hstring
```

19.7.11 Les productions "SignedNumber", "bstring", et "hstring" sont spécifiées dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 18.1, 11.10 et 11.12 respectivement.

19.7.12 La production "SingleIntValMap" mappe la valeur entière spécifiée sur la valeur spécifiée de chaîne de bits.

19.7.13 La production "IntValRangeMap" est la suivante:

```
IntValRangeMap ::=
        IntRange
        TO
        BitRange

IntRange ::=
        IntRangeValue1
        ".."
        IntRangeValue2

BitRange ::=
        BitRangeValue1
        ".."
        BitRangeValue2

IntRangeValue1 ::= SignedNumber

IntRangeValue2 ::= SignedNumber

BitRangeValue1 ::=
        bstring |
        hstring

BitRangeValue2 ::=
        bstring |
        hstring
```

19.7.14 Les chaînes de bits "BitRangeValue1" et "BitRangeValue2" doivent avoir le même nombre de bits.

19.7.15 La valeur "IntRangeValue2" doit être supérieure à la valeur "IntRangeValue1".

19.7.16 Interprétée en tant que codage d'entier positif (voir Rec. UIT-T X.690 | ISO/CEI 8825-1, § 8.3.3), la production "BitRangeValue2" doit représenter une valeur entière ("B", par exemple) supérieure à celle qui est représentée par "BitRangeValue1" ("A", par exemple), et la différence entre les valeurs entières correspondant à "BitRangeValue2" et à "BitRangeValue1" ("B" – "A") doit être égale à la différence entre les valeurs de "IntRangeValue2" et "IntRangeValue1".

19.7.17 La production "bitRange" représente l'ensemble ordonné des chaînes de bits correspondant aux valeurs entières entre "A" et "B".

19.7.18 La production "IntValRangeMap" mappe chacun des entiers dans l'étendue spécifiée sur la valeur de chaîne de bits correspondante dans la production "bitRange". (L'Annexe E donne des exemples de production "IntValRangeMap".)

19.7.19 Il y a erreur de spécification ECN si une éventuelle production "BitRange" contient une valeur qui viole une contrainte de longueur dans la cible.

20 Définition des objets de codage au moyen d'une syntaxe définie

20.1 Les articles 21 à 25 spécifient les informations nécessaires afin de définir des objets de codage pour chaque catégorie de classe de codage et la syntaxe à utiliser. Cette syntaxe est appelée *syntaxe définie* et est spécifiée au moyen de la notation de classe d'objets informationnels de la Rec. UIT-T X.681 | ISO/CEI 8824-2 telle que modifiée par l'Annexe B de la présente Recommandation | Norme internationale.

20.2 La syntaxe définie pour chaque catégorie peut également être utilisée afin de définir des objets de codage pour les structures qui sont des classes de cette catégorie, précédées par une ou plusieurs instances d'une classe de la catégorie des étiquettes. Lorsque le texte suivant nécessite qu'une classe soit dans une catégorie spécifiée, cela comprend le cas où la classe est précédée par une classe de la catégorie des étiquettes.

20.3 L'utilisation de la notation de classe d'objets informationnels modifiée n'est à utiliser qu'à l'intérieur de la présente Recommandation | Norme internationale

20.4 L'utilisation de la notation de syntaxe définie afin de définir des objets de codage est spécifiée au § 17.2. La syntaxe définie pour la définition des objets de codage doit être la syntaxe spécifiée par les instructions **WITH SYNTAX** figurant dans les articles 23 à 25.

20.5 Les instructions **WITH SYNTAX** imposent des contraintes applicables aux valeurs de certaines des propriétés de codage, en liaison avec les valeurs d'autres propriétés de codage, afin d'appliquer certaines (mais non toutes) contraintes sémantiques. D'autres contraintes relatives à l'utilisation des instructions **WITH SYNTAX** sont spécifiées dans le texte.

20.6 La syntaxe définie pour chaque classe de codage spécifie un certain nombre des propriétés de codage qui peuvent être fournies avec des valeurs des types ASN.1 définis dans l'article 21 (ou dans certains cas avec d'autres classes de codage et objets de codage) afin d'offrir les informations nécessaires lors de la spécification d'un objet de codage de cette classe. Les informations nécessaires afin de définir un objet de codage sont en général une combinaison de valeurs de propriété de codage, ainsi que l'instance particulière de la syntaxe définie qui a été utilisée afin de spécifier ces valeurs.

NOTE – Cela diffère de l'utilisation d'une instruction **WITH SYNTAX** dans la définition normale d'un objet informationnel, lorsque la sémantique associée à l'objet informationnel dépend seulement des valeurs choisies pour les champs de cette classe d'objets informationnels et non de la forme de l'instruction **WITH SYNTAX** utilisée pour régler ces valeurs (voir § B.15).

20.7 Les propriétés de codage spécifiées dans les articles 23 à 25 opèrent ensemble par groupes de propriétés de codage et utilisent des valeurs de types ASN.1 pour leur définition. L'article 21 spécifie la signification des valeurs de type généralement utilisées dans la spécification de ces propriétés de codage.

20.8 Quelques textes définitifs figurant dans les articles 21 et 22 sont copiés dans les articles 22 à 25. Lorsqu'il apparaît, le texte copié est "atténué par grisé" et une référence est donnée au texte définitif.

20.9 L'article 25 spécifie un certain nombre des transformées qui peuvent être appliquées à des valeurs abstraites. Plusieurs groupes de propriétés de codage nécessitent une liste ordonnée des transformées qui doivent être appliquées par un codeur. Pour que le décodage soit possible, les transformées appliquées par un codeur doivent être réversibles par un décodeur afin de retrouver les valeurs abstraites originales. Les articles 23 et 24 indiquent si les transformées doivent être réversibles, et l'article 25 spécifie les valeurs abstraites pour lesquelles une transformée de données quelconque est réversible.

21 Types utilisés lors de la spécification de syntaxe définie

NOTE – Toutes les définitions de type ASN.1 données ici impliquent la présence d'étiquettes automatiques et l'absence d'extensibilité.

21.1 Le type `Unit`

21.1.1 La production"**`Unit`**" est la suivante:

```
Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
      dword32(32)} (0..256)
```

21.1.2 La valeur par défaut pour ce type est toujours **`bit`**.

21.1.3 Une propriété de codage de ce type spécifie l'unité dans laquelle d'autres propriétés de codage ou d'autres champs de déterminant effectuent le comptage.

21.1.4 La valeur d'une propriété de codage de ce type est restreinte dans tous les cas sauf un aux valeurs différentes de zéro. Dans ces cas, la propriété de codage spécifie un certain nombre de bits, qui détermine l'unité dans laquelle d'autres propriétés de codage ou d'autres champs de déterminant effectuent le comptage.

21.1.5 Lorsqu'elle est utilisée dans la définition d'un objet de codage d'une classe de la catégorie des répétitions, la valeur **`repetitions`** est également autorisée et spécifie que le décompte associé indique le nombre de répétitions dans le codage.

21.2 Le type `EncodingSpaceSize`

21.2.1 La production "**`EncodingSpaceSize`**" est la suivante:

```
EncodingSpaceSize ::= INTEGER
      { encoder-option-with-determinant(-3),
        variable-with-determinant(-2),
        self-delimiting-values(-1),
        fixed-to-max(0)} (-3..MAX)
```

21.2.2 La valeur par défaut pour ce type est toujours **`self-delimiting-values`**.

21.2.3 Une propriété de codage de ce type spécifie la taille de l'espace de codage (voir § 9.21.5).

21.2.4 Les valeurs positives (différentes de zéro) spécifient une taille fixe pour l'espace de codage, en tant que valeur de type "**`Unit`**" multipliée par la valeur de type "**`EncodingSpaceSize`**", en bits. Si la valeur de type "**`Unit`**" est "**`repetitions`**", dans ce cas la taille de l'espace de codage peut être variable (car l'espace de codage nécessaire pour chaque composant peut être différent), mais est toujours ce nombre fixe de répétitions. Il y a erreur de spécification ou d'application ECN si une valeur abstraite qui ne possède pas ce nombre de répétitions doit être codée.

21.2.5 La valeur "**`encoder-option-with-determinant`**" spécifie que la taille de l'espace de codage peut varier conformément à la valeur abstraite en cours de codage, et que le codeur doit choisir la taille de l'espace de codage, en enregistrant la taille choisie dans le déterminant associé. Dans ce cas, une valeur de type "**`EncodingSpaceDetermination`**" (voir § 21.3) ou "**`RepetitionSpaceDetermination`**" (voir § 21.7) est requise.

NOTE – Une valeur de type "**`EncodingSpaceDetermination`**" ou "**`RepetitionSpaceDetermination`**" (afin de déterminer la taille de l'espace de codage) est requise dans ce cas (et dans le cas du § 21.2.6), mais la fourniture d'un déterminant est permise dans tous les autres cas afin de prendre en charge les codages (semblables aux règles BER) qui utilisent des déterminants de longueur même lorsqu'ils sont redondants. Toute différence entre ces deux déterminations est une erreur. Il peut, cependant, ne pas être toujours possible de déterminer s'il y a erreur de spécification ou d'application ECN, mais les codeurs conformes sont tenus de ne pas transmettre de tels codages.

21.2.6 La valeur "**`variable-with-determinant`**" spécifie que la taille de l'espace de codage peut varier conformément à la valeur abstraite en cours de codage. Dans ce cas, la valeur de type "**`EncodingSpaceDetermination`**" (voir § 21.3) ou "**`RepetitionSpaceDetermination`**" (voir § 21.7) est requise (afin d'offrir un moyen précis de déterminer la taille de l'espace de codage).

21.2.7 La valeur "**`self-delimiting-values`**" spécifie que le codage de valeur est autodélimitateur, c'est-à-dire que chaque valeur code dans un multiple de la valeur spécifiée de type "**`Unit`**". Il ne doit pas y avoir de paire de valeurs abstraites pour laquelle le codage d'une seule valeur abstraite est la première partie du codage de l'autre valeur abstraite.

NOTE – Un décodeur peut (après détermination possible des bits inutilisés et de la justification) déterminer l'extrémité de l'espace de codage par mise en correspondance du codage de chaque valeur abstraite possible avec le codage qui est examiné. Précisément un seul décodeur correspondra aux codages produits par un codeur conforme. Les décodeurs peuvent mettre au point des approches plus efficaces mais équivalentes.

21.2.8 La valeur "**`fixed-to-max`**" spécifie que l'espace de codage doit être le même pour le codage de toutes les valeurs abstraites. Elle spécifie que la taille de l'espace de codage doit être le plus petit multiple de "**`Unit`**" qui peut contenir le codage spécifié d'une quelconque des valeurs abstraites (ou de toutes). Cette valeur ne doit pas être utilisée si la valeur abstraite à coder dans l'espace de codage est une valeur abstraite associée à une classe de la catégorie des concaténations (voir § 23.5.2.3) ou des répétitions (voir § 23.13.2.5).

NOTE 1 – Un cas spécial existe lorsqu'il y a une seule valeur abstraite dont le codage de valeur est en bits zéro. Il en résulte un espace de codage vide (bits zéro).

NOTE 2 – Si une telle spécification est appliquée lorsqu'une taille maximale ne peut pas être déterminée (par exemple afin de coder un entier non borné), il y a erreur de spécification ECN, mais les codeurs conformes doivent refuser de produire des codages dans de tels cas.

21.3 Le type `EncodingSpaceDetermination`

21.3.1 La production "**`EncodingSpaceDetermination`**" est la suivante:

```
EncodingSpaceDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, container}
```

21.3.2 La valeur par défaut pour ce type est toujours "**`field-to-be-set`**".

21.3.3 Une propriété de codage de ce type spécifie la façon dont l'espace de codage est déterminé lorsqu'une propriété de codage de type "**`EncodingSpaceSize`**" (voir § 21.2) est fixée à "**`variable-with-determinant`**" ou "**`encoder-option-with-determinant`**".

21.3.4 La valeur "**field-to-be-set**" nécessite la spécification d'une référence **REFERENCE** à un champ qui sera réglé par le codeur de façon à contenir des informations de longueur et utilisé par un décodeur. La spécification de codage détermine la façon dont un codeur doit régler la valeur de ce champ à partir de la taille (en unités d'espace de codage) de l'espace de codage. Si un champ est activé plus d'une fois au moyen de la valeur "**field-to-be-set**" ou "**flag-to-be-set**" (voir § 21.7), dans ce cas il y a erreur de spécification ou d'application ECN si différentes valeurs sont produites par les différentes procédures de codage, et les codeurs ne doivent pas produire de codages dans ce cas.

21.3.5 La valeur "**field-to-be-used**" nécessite la spécification d'une référence **REFERENCE** à un champ dont la valeur peut être fixée à partir de la syntaxe abstraite (c'est-à-dire un champ correspondant apparaît à l'intérieur de la spécification ASN.1) ou peut être fixée par certaines autres actions du codeur invoquées par "**field-to-be-set**" ou "**flag-to-be-set**". La spécification de codage détermine la façon dont un décodeur doit obtenir la taille de l'espace de codage à partir de la valeur de ce champ. Un codeur conforme ne doit pas produire de codages dans lesquels les transformées par le décodeur de ce champ ne désignent pas correctement l'extrémité de l'espace de codage.

21.3.6 La valeur "container" nécessite soit la spécification d'une référence **REFERENCE** à un autre champ dont la classe de codage (le conteneur) a un déterminant de longueur et dont le contenu comprend cet espace de codage, ou la spécification que l'extrémité de l'unité PDU détermine l'extrémité de l'espace de codage (au moyen de **OUTER**). L'espace de codage se termine lorsque le conteneur spécifié se termine ou lorsque l'extrémité de l'unité PDU est rencontrée. Cette spécification ne peut être utilisée que si l'espace de codage de l'élément codé est le dernier codage à placer dans le conteneur.

NOTE – Il y a erreur du codeur ECN (éventuellement résultant d'une erreur de spécification ou d'application ECN) si des codages additionnels sont placés dans le conteneur.

21.4 Le type `UnusedBitsDetermination`

21.4.1 Le type "**UnusedBitsDetermination**" est le suivant:

```
UnusedBitsDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, not-needed}
```

21.4.2 La valeur par défaut pour ce type est toujours "**field-to-be-set**".

21.4.3 Une propriété de codage de ce type spécifie la façon dont un décodeur peut déterminer les bits non utilisés lorsque le codage de valeur est avec justification à gauche ou à droite dans un espace de codage.

21.4.4 La valeur "**field-to-be-set**" nécessite la spécification d'une référence **REFERENCE** à un champ qui sera réglé par le codeur de façon à contenir des informations sur les bits inutilisés, et qui sera utilisé par un décodeur. La spécification de codage détermine la façon dont un codeur vise à déterminer le nombre de bits inutilisés et la façon de régler la valeur de ce champ à partir du nombre de bits inutilisés. Si un champ est activé plus d'une fois au moyen de "**field-to-be-set**" ou "**flag-to-be-set**" (voir § 21.7), il y a erreur de spécification ou d'application ECN si différentes valeurs sont produites par les différentes procédures de codage et les codeurs ne doivent pas produire de codages dans ce cas.

21.4.5 La valeur "**field-to-be-used**" nécessite la spécification d'une référence **REFERENCE** à un champ dont la valeur peut être fixée à partir de la syntaxe abstraite (c'est-à-dire un champ correspondant apparaît à l'intérieur de la spécification ASN.1) ou peut être fixée par certaines autres actions du codeur invoquées par "**field-to-be-set**" ou "**flag-to-be-set**". La spécification de codage détermine la façon dont un décodeur vise à déterminer le nombre de bits inutilisés à partir de la valeur de ce champ. Un codeur conforme ne doit pas produire de codages dans lesquels les transformées par le décodeur de ce champ ne désignent pas correctement le nombre de bits inutilisés.

21.4.6 La valeur "**not-needed**" indique qu'un décodeur ne nécessite pas un déterminant explicite afin de découvrir le nombre de bits inutilisés. Le nombre de bits inutilisés pourra être déduit à partir de la spécification de codage sans la connaissance de la valeur abstraite réelle qui a été codée. Cette détermination est décrite pour chaque codage de valeur.

21.5 Le type `OptionalityDetermination`

21.5.1 Le type "**OptionalityDetermination**" est le suivant:

```
OptionalityDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, container, handle, pointer}
```

21.5.2 La valeur par défaut pour ce type est toujours "**field-to-be-set**".

21.5.3 Une propriété de codage de ce type spécifie la façon dont la présence ou l'absence d'un composant facultatif est déterminé.

21.5.4 La valeur "**field-to-be-set**" nécessite la spécification d'une référence **REFERENCE** à un champ qui sera réglé par le codeur de façon à contenir des informations d'offres d'options et utilisé par un décodeur. La spécification ECN comprendra également une propriété de codage qui spécifie la façon dont un codeur doit régler la valeur de ce champ à partir d'une valeur booléenne théorique qui est vraie si le composant facultatif est présent et fausse si le composant facultatif est absent. Si un champ est activé plus d'une fois au moyen de "**field-to-be-set**" ou "**flag-to-be-set**" (voir § 21.7), il y a erreur de spécification ou d'application ECN si différentes valeurs sont produites par les différentes procédures de codage et les codeurs ne doivent pas produire de codages dans ce cas.

21.5.5 La valeur "**field-to-be-used**" nécessite la spécification d'une référence **REFERENCE** à un champ dont la valeur peut être fixée à partir de la syntaxe abstraite (c'est-à-dire qu'un champ correspondant apparaît à l'intérieur de la spécification ASN.1) ou peut être fixée par certaines autres actions du codeur invoquées par "**field-to-be-set**" ou "**flag-to-be-set**". La spécification comprendra également une propriété de codage qui spécifie la façon dont un décodeur vise à déterminer la présence ou l'absence du composant facultatif à partir de la valeur de ce champ. Un codeur conforme doit garantir que la valeur de ce champ détermine correctement la présence ou l'absence du champ facultatif.

21.5.6 La valeur "**container**" nécessite soit la spécification d'une référence **REFERENCE** à un autre champ dont la classe de codage (le conteneur) a un déterminant de longueur et dont le contenu comprend ce composant facultatif, ou la spécification du fait que le conteneur est l'extrémité de l'unité PDU (au moyen de **OUTER)**. Si l'extrémité du conteneur est présente lorsqu'un décodeur recherche le début de ce composant facultatif, dans ce cas le décodeur doit déterminer que ce composant facultatif est absent.

NOTE – Cette spécification ne peut être utilisée que si les valeurs abstraites codées sont telles qu'aucun autre codage ne doit être placé dans le conteneur. Cela peut nécessiter que des restrictions soient imposées aux valeurs abstraites du type ASN.1, par exemple, afin d'interdire l'inclusion d'un composant facultatif ultérieur à moins que tous les composants facultatifs antérieurs soient présents. C'est soit une erreur de spécification ECN ou une erreur d'application si des codages additionnels doivent être placés dans le conteneur suivant un composant dont l'offre d'options est déterminée de cette façon, mais un codeur conforme ne doit pas produire de tels codages.

21.5.7 La valeur "**handle**" nécessite qu'un pointeur d'identification soit spécifié. Ce pointeur d'identification doit être affiché par l'objet de codage pour le composant facultatif et par tout codage en option pouvant suivre si ce composant facultatif est absent. La valeur du pointeur doit être différente pour le codage du composant facultatif et pour tous les codages suivants qui peuvent être offerts en option. Si l'extrémité d'un quelconque conteneur ouvert (ou l'extrémité de l'unité PDU) est détectée au moment où un décodeur tente de détecter la présence ou l'absence de ce composant facultatif, le composant est absent. Sinon, un décodeur doit déterminer que le composant est présent si et seulement si le décodage des parties restantes du codage produit pour le pointeur d'identification spécifié une valeur qui correspond à celle du composant facultatif. Il y a erreur de spécification ECN s'il n'en résulte pas une identification correcte de la présence ou de l'absence d'un codage du composant facultatif; mais les codeurs conformes ne doivent pas produire de tels codages.

21.5.8 La valeur "**pointer**" nécessite la spécification d'une référence **REFERENCE** de début de codage à un autre champ. Si ce champ est zéro, dans ce cas ce composant est absent. S'il est différent de zéro, les règles pour un pointeur de début de codage s'appliquent (voir § 22.3).

21.6 Le type `AlternativeDetermination`

21.6.1 Le type "**AlternativeDetermination**" est le suivant:

```
AlternativeDetermination ::=
      ENUMERATED {field-to-be-set, field-to-be-used, handle}
```

21.6.2 La valeur par défaut pour ce type est toujours "**field-to-be-set**".

21.6.3 Une propriété de codage de ce type spécifie la façon dont un décodeur détermine quelle option est présente dans un codage d'une classe de la catégorie des options.

21.6.4 La valeur "**field-to-be-set**" nécessite la spécification d'une référence **REFERENCE** à un champ qui sera réglé par le codeur de façon à contenir des informations désignant une option, et qui sera utilisé par un décodeur. La spécification comprendra également une propriété de codage qui spécifie la façon dont un codeur doit régler la valeur de ce champ à partir d'une valeur entière théorique qui identifie chaque option (au moyen d'un ordre spécifié dans d'autres propriétés de codage). Si un champ est activé plus d'une fois au moyen de "**field-to-be-set**" ou "**flag-to-be-set**" (voir § 21.7), il y a erreur de spécification ou d'application ECN si différentes valeurs sont produites par les différentes procédures de codage et les codeurs ne doivent pas produire de codages dans ce cas.

21.6.5 La valeur "**field-to-be-used**" nécessite la spécification d'une référence **REFERENCE** à un champ dont la valeur peut être fixée à partir de la syntaxe abstraite (c'est-à-dire un champ correspondant apparaît à l'intérieur de la spécification ASN.1) ou peut être fixée par certaines autres actions du codeur invoquées par "**field-to-be-set**" ou

"**flag-to-be-set**". La spécification comprendra également une propriété de codage qui spécifie la façon dont un décodeur vise à déterminer (à partir de la valeur du champ cité en référence) une valeur entière théorique qui identifie l'option (au moyen d'un ordre spécifié dans d'autres propriétés de codage).

21.6.6 La valeur "**handle**" nécessite qu'un pointeur d'identification soit spécifié. Ce pointeur d'identification doit être affiché par (les codages de) toutes les options dans la classe et le codage de chaque option doit avoir une valeur différente pour le pointeur d'identification. (La violation de cette règle est une erreur de spécification ECN, mais les codeurs conformes sont tenus de ne pas produire de codages où cette règle est violée.) Cette valeur spécifie qu'un décodeur doit déterminer l'option qui est présente par le décodage des parties restantes du codage afin de produire une valeur pour le pointeur d'identification spécifié. L'option dont le pointeur d'identification a une valeur qui correspond à cette valeur est l'option qui est présente. Si l'extrémité d'un quelconque conteneur ouvert (ou l'extrémité de l'unité PDU) est atteinte avant que le pointeur d'identification puisse être décodé, ou si la valeur du pointeur d'identification ne correspond pas à celle d'une quelconque option, dans ce cas il y a erreur de codage.

21.7 Le type RepetitionSpaceDetermination

21.7.1 Le type "**RepetitionSpaceDetermination**" est le suivant:

```
RepetitionSpaceDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, flag-to-be-set, flag-to-be-used,
       container, pattern, handle, not-needed}
```

21.7.2 La valeur par défaut pour ce type est toujours "**field-to-be-set**".

21.7.3 Une propriété de codage de ce type spécifie la façon dont un décodeur détermine l'extrémité de l'espace de codage dans un codage d'une classe de la catégorie des répétitions. Elle remplace l'utilisation d'une propriété de codage de type "**EncodingSpaceDetermination**" dans le codage de répétitions.

21.7.4 La valeur "**field-to-be-set**" nécessite la spécification d'une référence **REFERENCE** à un champ qui sera réglé par le codeur de façon à contenir des informations qui identifient la taille de l'espace de répétition. La spécification de codage détermine la façon dont un codeur doit régler la valeur de ce champ à partir de la taille (en unités d'espace de répétition) de l'espace de répétition. Si un champ est activé plus d'une fois au moyen de "**field-to-be-set**" ou "**flag-to-be-set**", dans ce cas il y a erreur de spécification ou d'application ECN si différentes valeurs sont produites par les différentes procédures de codage et les codeurs ne doivent pas produire de codages dans ce cas.

21.7.5 La valeur "**field-to-be-used**" nécessite la spécification d'une référence **REFERENCE** à un champ dont la valeur peut être fixée à partir de la syntaxe abstraite (c'est-à-dire un champ correspondant apparaît à l'intérieur de la spécification ASN.1) ou peut être fixée par certaines autres actions du codeur invoquées par "**field-to-be-set**" ou "**flag-to-be-set**". La spécification de codage détermine la façon dont un décodeur doit obtenir la taille (en unités d'espace de répétition) de l'espace de codage à partir de la valeur de ce champ. Un codeur conforme ne doit pas produire de codages dans lesquels les transformées par le décodeur de ce champ ne désignent pas correctement l'extrémité de l'espace de codage.

21.7.6 La valeur "**flag-to-be-set**" nécessite la spécification d'une référence **REFERENCE** à un champ qui fait partie de l'élément répété et qui sera réglé par le codeur afin de désigner le dernier élément de la répétition. La spécification de codage détermine la façon dont un codeur doit régler la valeur de ce champ à partir d'une valeur booléenne qui est fausse si l'élément est le dernier dans la répétition, et est vraie sinon. Si un champ est activé plus d'une fois au moyen de "**flag-to-be-set**" ou "**field-to-be-set**", dans ce cas il y a erreur de spécification ou d'application ECN si différentes valeurs sont produites par les différentes procédures de codage, et les codeurs ne doivent pas produire de codages dans ce cas.

21.7.7 La valeur "**flag-to-be-used**" nécessite la spécification d'une référence **REFERENCE** à un champ qui fait partie de l'élément répété et dont la valeur peut être fixée à partir de la syntaxe abstraite (c'est-à-dire un champ correspondant apparaît à l'intérieur de la spécification ASN.1) ou peut être fixée par certaines autres actions du codeur invoquées par "**flag-to-be-set**" ou "**field-to-be-set**". La spécification de codage détermine la façon dont un décodeur doit obtenir une valeur booléenne à partir de la valeur de ce champ. La valeur booléenne sera fausse si l'élément est le dernier dans la répétition, et vraie sinon. Un codeur conforme ne doit pas produire de codages dans lesquels les transformées par le décodeur de ce champ ne désignent pas correctement le dernier élément de la répétition.

21.7.8 La valeur "**container**" nécessite soit la spécification d'une référence **REFERENCE** à un autre champ dont la classe de codage (le conteneur) a un déterminant de longueur et dont le contenu comprend la classe de codage de la catégorie des répétitions, ou la spécification (au moyen de **OUTER**) en tant qu'extrémité de l'unité PDU détermine l'extrémité des répétitions. Les répétitions se terminent lorsque le conteneur spécifié se termine ou quand, suivant le codage complet d'une seule répétition, l'extrémité de l'unité PDU est rencontrée.

NOTE – Cette spécification ne peut être utilisée que si le codage de la classe (dans la catégorie des répétitions) est le dernier codage à placer dans le conteneur. Il y a erreur de spécification ECN si des codages additionnels sont placés dans le conteneur, mais les codeurs conformes ne doivent pas produire de tels codages.

21.7.9 La valeur "**pattern**" spécifie qu'une certaine séquence de bits spécifiée (voir § 21.10) mettra fin aux répétitions. Dans ce cas les propriétés de codage additionnelles nécessiteront l'insertion par un codeur d'une séquence spécifiée et la détection de cette séquence par un décodeur. Il y a erreur de spécification ECN si le codage de la séquence peut être la partie initiale du codage d'une valeur abstraite de répétition. Un codeur conforme doit détecter de telles erreurs et ne doit pas produire de codages qui violent cette règle.

NOTE – Un exemple est une chaîne de caractères terminée par néant dont le contenu n'est pas autorisé à comprendre un caractère néant.

21.7.10 La valeur "**handle**" nécessite qu'un pointeur d'identification soit spécifié. Ce pointeur d'identification doit être affiché par l'élément répété et par tous les éléments pouvant suivre (compte tenu de l'offre d'options). La valeur du pointeur d'identification pour l'élément répété doit être différente de celle de tous les éléments pouvant suivre.

21.7.11 La valeur "**not-needed**" spécifie que le nombre de répétitions est fixe dans la syntaxe abstraite.

NOTE – Il y a erreur de spécification ECN (qui doit être détectée et bloquée par les codeurs) si ce codage est spécifié et que le nombre de répétitions ne soit pas restreint de cette façon, ou si l'application viole cette restriction.

21.8 Le type `Justification`

21.8.1 Le type "**Justification**" est le suivant:

```
Justification ::= CHOICE
                { left              INTEGER (0..MAX),
                  right             INTEGER (0..MAX)}
```

21.8.2 La valeur par défaut pour ce type est toujours "**right:0**".

21.8.3 Une propriété de codage de ce type spécifie justification à droite ou à gauche du codage d'une valeur à l'intérieur de l'espace de codage, avec un décalage en bits à partir de l'extrémité de l'espace de codage.

21.8.4 L'option "**left**" spécifie que le bit initial du codage de valeur est positionné par rapport au bord antérieur de l'espace de codage. La valeur entière spécifie le nombre de bits entre le bord antérieur de l'espace de codage et le bit initial du codage de valeur.

NOTE – Si le codage de valeur n'est pas de longueur fixe ou autodélimitateur, l'utilisation du bourrage de valeur dans un conteneur de taille fixe peut en certaines circonstances rendre impossible à un décodeur de rétablir les valeurs abstraites originales, ce qui serait une erreur de spécification ECN.

21.8.5 L'option "**right**" spécifie que le bit de fin du codage de valeur est positionné par rapport au bord final de l'espace de codage. La valeur entière spécifie le nombre de bits entre le bit de fin du codage de valeur et le bord final de l'espace de codage.

21.8.6 Le réglage des bits (éventuels) avant ou après le codage de valeur est déterminé par les propriétés de codage de type "**Padding**" et "**Pattern**" (voir § 21.9 et 21.10).

21.9 Le type `Padding`

21.9.1 Le type "**Padding**" est le suivant:

```
Padding ::= ENUMERATED {zero, one, pattern, encoder-option}
```

21.9.2 La valeur par défaut d'une propriété de codage de ce type est toujours "**zero**".

21.9.3 Une propriété de codage de ce type spécifie les détails du bourrage pour le prébourrage, pour les classes de la catégorie des bourrages, et pour le post-bourrage d'une unité PDU spécifiée dans la classe de codage **#OUTER**.

21.9.4 Si la valeur est "**zero**", le bourrage est effectué avec zéro bit.

21.9.5 Si la valeur est "**one**", le bourrage est effectué avec un seul bit.

21.9.6 Si la valeur est "**pattern**", les bits sont réglés conformément à la propriété de codage de type "**Pattern**" (voir § 21.10).

21.9.7 Si la valeur est "**encoder-option**", le codeur choisit librement les valeurs des bits.

21.10 Les types `Pattern` et `Non-Null-Pattern`

21.10.1 Le type "**`Pattern`**" est le suivant:

```
Pattern ::= CHOICE
      {bits                       BIT STRING,
       octets                     OCTET STRING,
       char8                      IA5String,
       char16                     BMPString,
       char32                     UniversalString,
       any-of-length              INTEGER (1..MAX),
      different                   ENUMERATED {any} }
```

21.10.2 Le type "**`Non-Null-Pattern`**" est le suivant:

```
Non-Null-Pattern ::= Pattern
      (ALL EXCEPT (bits:''B | octets:''H | char8:"" | char16:"" |
                   char32:""))
```

21.10.3 La valeur par défaut pour une propriété de codage de ce type est toujours "**`bits:'0'B`**".

21.10.4 L'option "**`bits`**" ou "**`octets`**" spécifie une séquence de longueur et de valeur égales respectivement à la chaîne de bits ou à la chaîne d'octets.

21.10.5 L'option "**`char8`**" alternative spécifie une séquence (multiple de 8 bits) dans laquelle chaque caractère contenu dans la chaîne donnée est converti en sa valeur ISO/CEI 10646-1 en tant que valeur de 8 bits.

21.10.6 L'option "**`char16`**" alternative spécifie une séquence (multiple de 16 bits) dans laquelle chaque caractère contenu dans la chaîne donnée est converti en sa valeur ISO/CEI 10646-1 en tant que valeur de 16 bits.

21.10.7 L'option "**`char32`**" spécifie une séquence (multiple de 32 bits) dans laquelle chaque caractère contenu dans la chaîne donnée est converti en sa valeur ISO/CEI 10646-1 en tant que valeur de 32 bits.

21.10.8 L'option "**`any-of-length`**" spécifie une longueur pour la séquence. La valeur réelle de la séquence est une option de codeur.

21.10.9 La valeur "**`different:any`**" value n'est permise que lorsqu'il y a une autre propriété de codage de type "**`Pattern`**" dans le même groupe de propriétés de codage. Dans ce cas, une des deux (mais non les deux) propriétés de codage de type "**`Pattern`**" peut être mise à la valeur "**`different:any`**", qui spécifie que la longueur de la séquence doit être la même que la longueur de la séquence spécifiée pour l'autre propriété de codage. Elle spécifie également que sa valeur est une option de codeur, à condition que cette valeur soit différente de celle de la séquence spécifiée pour l'autre propriété de codage.

21.10.10 Le type "**`Non-Null-Pattern`**" est utilisé pour le prébourrage et pour la justification (mais non pour d'autres usages) et la séquence est tronquée et/ou reproduite si nécessaire afin d'offrir un nombre de bits suffisant pour le prébourrage, le prébourrage de valeur, ou le post-bourrage de valeur.

21.10.11 La valeur "**`different:any`**" du type "**`Pattern`**" est exclue de la plupart des utilisations de ce type. Lorsqu'un paramètre de type "**`Pattern`**" est utilisé afin de spécifier la séquence pour une valeur booléenne (**`TRUE`**, par exemple), dans ce cas la valeur "**`different:any`**" peut être utilisée afin de spécifier la séquence pour l'autre valeur booléenne (**`FALSE`** dans ce cas). Lorsqu'elle est utilisée de cette façon, la valeur "**`different:any`**" indique une option de codeur pour la séquence. Le codeur peut utiliser toute séquence qu'il choisit, mais cette séquence doit avoir la même longueur que l'autre séquence et doit en différer d'au moins une position binaire.

21.11 Le type `RangeCondition`

21.11.1 Le type "**`RangeCondition`**" est le suivant:

```
RangeCondition ::= ENUMERATED
      { unbounded-or-no-lower-bound,
        semi-bounded-with-negatives,
        bounded-with-negatives,
        semi-bounded-without-negatives,
        bounded-without-negatives}
```

21.11.2 La valeur par défaut d'une propriété de codage de ce type est toujours "**`unbounded-or-no-lower-bound`**".

21.11.3 Une propriété de codage de type "**`RangeCondition`**" est utilisée dans la spécification d'un prédicat qui vérifie l'existence et la nature de limites applicables aux valeurs entières associées à une classe de codage dans la catégorie des entiers.

21.11.4 Le prédicat est satisfait pour chaque valeur d'énumération si et seulement si les conditions suivantes sont satisfaites par les limites relatives à la classe de codage dans la catégorie des entiers:

a) **`unbounded-or-no-lower-bound`**: soit il n'y a pas de limites, ou il y a seulement une limite supérieure mais aucune limite inférieure.

b) **`semi-bounded-with-negatives`**: il y a une limite inférieure qui est inférieure à zéro, mais aucune limite supérieure.

c) **`bounded-with-negatives`**: il y a une limite inférieure qui est inférieure à zéro et une limite supérieure.

d) **`semi-bounded-without-negatives`**: il y a une limite inférieure qui est supérieure ou égale à zéro, mais aucune limite supérieure.

e) **`bounded-without-negatives`**: il y a une limite inférieure qui est supérieure ou égale à zéro, et une limite supérieure.

NOTE – Pour tout ensemble donné de limites, exactement un seul prédicat sera satisfait.

21.12 Le type `SizeRangeCondition`

21.12.1 Le type "**`SizeRangeCondition`**" est le suivant:

```
SizeRangeCondition ::= ENUMERATED
      { no-ub-with-zero-lb,
        ub-with-zero-lb,
        no-ub-with-non-zero-lb,
        ub-with-non-zero-lb,
        fixed-size}
```

21.12.2 La valeur par défaut pour une propriété de codage de ce type est toujours "**`no-ub-with-zero-lb`**".

21.12.3 Une propriété de codage de type "**`SizeRangeCondition`**" est utilisée afin de vérifier les propriétés des limites dans une contrainte de longueur effective associée à une classe de la catégorie des répétitions ou des chaînes de caractères.

21.12.4 Le prédicat est satisfait pour chaque valeur d'énumération si et seulement si la contrainte de longueur effective répond aux conditions suivantes:

a) **`no-ub-with-zero-lb`**: il n'y a pas de limite supérieure sur la taille et la limite inférieure est zéro.

b) **`ub-with-zero-lb`**: il y a une limite supérieure sur la taille et la limite inférieure est zéro.

c) **`no-ub-with-non-zero-lb`**: il n'y a pas de limite supérieure sur la taille et la limite inférieure est différente de zéro.

d) **`ub-with-non-zero-lb`**: il y a une limite supérieure sur la taille et la limite inférieure est différente de zéro.

e) **`fixed-size`**: la limite inférieure et la limite supérieure sur la taille ont la même valeur.

NOTE – Seul le cas de la valeur "**`fixed-size`**" recouvre d'autres prédicats.

21.13 Le type `ReversalSpecification`

21.13.1 Le type "**`ReversalSpecification`**" est le suivant:

```
ReversalSpecification ::= ENUMERATED
      {no-reversal,
      reverse-bits-in-units,
      reverse-half-units,
      reverse-bits-in-half-units}
```

21.13.2 La valeur par défaut pour une propriété de codage de ce type est toujours "**`no-reversal`**".

21.13.3 Une propriété de codage de type "**`ReversalSpecification`**" est utilisée dans la transformation finale de bits à partir d'un espace de codage vers un tampon de sortie pour transmission (la transformation inverse étant effectuée pour le décodage).

NOTE – Les bits insérés en tant que résultat du prébourrage spécifié par un objet de codage ne font pas partie du codage dû à l'inversion de l'ordre des bits spécifiée par cet objet de codage, mais ils peuvent être soumis à l'inversion de l'ordre des bits spécifiée par un objet de codage pour un conteneur dans lequel le codage complet est intégré.

21.13.4 Les valeurs de ce type sont toujours utilisées en liaison avec une propriété de codage de type "**Unit**" qui spécifie une taille unitaire en bits (voir § 21.1).

21.13.5 Il y a erreur de spécification ECN si les valeurs "**reverse-half-units**" et "**reverse-bits-in-half-units**" sont utilisées lorsque la propriété de codage de type "**Unit**" n'a pas le même nombre de bits.

21.13.6 Les énumérations spécifient (dans l'ordre des énumérations reproduites ci-dessus) soit:

a) pas d'inversion de bits, ou

b) inversion de l'ordre de demi-unités (sans changer l'ordre des bits dans chaque demi-unité), ou

c) inversion de l'ordre de bits dans chaque demi-unité mais sans inversion de l'ordre des demi-unités, ou

d) inversion de l'ordre des bits dans chaque unité.

21.13.7 Il y a erreur de spécification ECN si le nombre de bits n'est pas un multiple entier de "**Unit**" dans un codage auquel l'inversion de l'ordre des bits est appliquée.

21.13.8 L'inversion de l'ordre des bits peut être spécifiée pour le codage de toutes les classes qui peuvent apparaître en tant que champs de structure de codage, sauf une classe de codage de la catégorie des options, qui n'utilise pas le concept d'espace de codage.

21.14 Le type `ResultSize`

21.14.1 Le type "**ResultSize**" est le suivant:

```
ResultSize ::= INTEGER {variable(-1), fixed-to-max(0)} (-1..MAX)
```

21.14.2 La valeur par défaut d'une propriété de codage de ce type est toujours "**variable**".

21.14.3 Une propriété de codage de ce type spécifie la taille du résultat dans une classe **#TRANSFORM**.

21.14.4 La valeur "**variable**" spécifie que la taille du résultat de la classe **#TRANSFORM** variera pour différentes valeurs abstraites et sera déterminée par la spécification détaillée de la transformée.

21.14.5 La valeur "**fixed-to-max**" spécifie que la taille du résultat de la classe **#TRANSFORM** doit être la même pour la transformée de toutes les valeurs abstraites. Elle spécifie que la taille cible doit être la plus petite taille qui peut contenir le codage spécifié d'une quelconque (ou de la totalité) des valeurs abstraites. Les détails précis de cette spécification sont définis pour chaque transformée dans laquelle des valeurs de ce type sont utilisées.

21.14.6 Une valeur positive de type "**ResultSize**" spécifie que la taille du résultat de la classe **#TRANSFORM** est fixe. Cette valeur est utilisée dans la spécification de la transformée réelle.

21.15 Le type `HandleValue`

21.15.1 Le type "**HandleValue**" est le suivant:

```
HandleValue ::= CHOICE {
      bits        BIT  STRING,
      octets      OCTET STRING,
      number      INTEGER (0..MAX),
      tag         ENUMERATED {any}}
```

21.15.2 La production "**HandleValue**" est utilisée afin de spécifier la valeur d'un pointeur d'identification qui est affichée par des objets de codage particuliers.

21.15.3 Les valeurs d'un quelconque pointeur d'identification qui est affiché par un objet de codage doivent être les mêmes pour toutes les valeurs abstraites codées par cet objet de codage (voir § 22.9.2.2). La valeur d'un pointeur d'identification peut être utilisée afin de désigner la présence ou l'absence de composants facultatifs, le choix d'options, ou la fin d'une répétition. Il est également requis dans de telles circonstances que les valeurs de pointeur affichées par le codage de différentes options ou de différents composants soient distinctes (voir § 21.5.7, 21.6.6 et 21.7.10).

NOTE – Les valeurs de pointeur d'identification affichées par un objet de codage donné peuvent, théoriquement, être déterminées par codage d'une valeur d'essai. Cependant, afin de faciliter la tâche d'implémentation, le spécificateur ECN est tenu de spécifier la valeur du pointeur dans tous les cas sauf si (pour les codages d'une classe d'étiquettes) la valeur du pointeur d'identification dépend du numéro d'étiquette associé à cette classe d'étiquettes, soit directement par production implicite à partir d'une étiquette ASN.1, ou par mappage à partir d'une structure produite implicitement.

21.15.4 Les options "**bits**", "**octets**" et "**number**" spécifient respectivement la valeur de pointeur en tant que chaîne de bits, chaîne d'octets ou valeur entière respectivement. Il y a erreur de spécification ECN si cette valeur ne peut pas être codée dans le nombre de bits spécifié pour le pointeur d'identification (voir § 22.9).

21.15.5 L'option "**tag:any**" spécifie que la valeur du pointeur est déterminée par le numéro spécifié dans une structure de codage ECN pour une classe de la catégorie des étiquettes, ou par le numéro d'étiquette mappé à partir d'une construction d'étiquette ASN.1. Elle ne doit être utilisée que lors de la spécification de l'identification par pointeur pour le codage d'une classe de la catégorie des étiquettes.

22 Groupes couramment utilisés de propriétés de codage

Ce paragraphe spécifie des groupes de propriétés de codage qui sont généralement utilisés dans la syntaxe définie (voir article 20). L'objet de chaque groupe, les restrictions visant aussi bien les valeurs des propriétés de codage que la syntaxe qui peut être utilisée, ainsi que le codeur et les actions du décodeur pour chaque groupe, sont également spécifiés.

22.1 Spécification de remplacement

Il y a trois variantes de spécification de remplacement:

a) spécification de remplacement complète: cette spécification est utilisée pour les classes de la catégorie des concaténations, où le remplacement peut concerner la structure entière, ou peut remplacer sélectivement des composants facultatifs et non facultatifs;

b) spécification de remplacement de structure ou de composant: cette spécification est utilisée pour les classes de la catégorie des options et pour la classe de codage **#CONDITIONAL-REPETITION**, où le remplacement peut concerner la structure entière ou le composant.

NOTE – Lorsqu'un objet de codage de la classe **#CONDITIONAL-REPETITION** est utilisé afin de définir des codages pour une classe de la catégorie des chaînes de bits, des chaînes de caractères ou des chaînes d'octets, il ne peut effectuer qu'un remplacement limité aux structures.

c) spécification de remplacement limité aux structures: cette spécification est utilisée pour les classes qui ne possèdent pas de composants.

22.1.1 Propriétés, syntaxe et finalité du codage

22.1.1.1 Une spécification de remplacement complète utilise les propriétés de codage suivantes:

```
&#Replacement-structure
      OPTIONAL,
&#Replacement-structure2
      OPTIONAL,
&replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,
&replacement-structure-encoding-object2 &#Replacement-structure2   OPTIONAL,
&#Head-end-structure                                                OPTIONAL,
&#Head-end-structure2                                               OPTIONAL
```

22.1.1.2 La syntaxe à utiliser pour une spécification de remplacement complète doit être la suivante:

```
[REPLACE
      [STRUCTURE]
      [COMPONENT]
      [ALL COMPONENTS]
      [OPTIONALS]
      [NON-OPTIONALS]
      WITH &#Replacement-structure
            [ENCODED BY &replacement-structure-encoding-object
                  [INSERT AT HEAD &#Head-end-structure]]
            [AND OPTIONALS WITH &#Replacement-structure2
                  [ENCODED BY &replacement-structure-encoding-object2
                        [INSERT AT HEAD &#Head-end-structure2]]] ]
```

22.1.1.3 La spécification de remplacement de structure ou de composant utilise les propriétés de codage suivantes:

```
&#Replacement-structure
      OPTIONAL,
&replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,
&#Head-end-structure                                                OPTIONAL
```

22.1.1.4 La syntaxe à utiliser pour la spécification de remplacement de structure ou de composant doit être la suivante:

```
[REPLACE
      [STRUCTURE]
      [COMPONENT]
      [ALL COMPONENTS]
      WITH &Replacement-structure
      [ENCODED BY &replacement-structure-encoding-object
            [INSERT AT HEAD &#Head-end-structure]]]
```

22.1.1.5 Une spécification de remplacement limitée aux structures utilise les propriétés de codage suivantes:

```
&#Replacement-structure
      OPTIONAL,
&replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL
```

22.1.1.6 La syntaxe à utiliser pour la spécification de remplacement limité aux structures doit être la suivante:

```
[REPLACE
      [STRUCTURE]
      WITH &#Replacement-structure
            [ENCODED BY &replacement-structure-encoding-object]]
```

22.1.1.7 L'utilisation de l'instruction "**WITH SYNTAX**" pour ces groupes de propriétés de codage spécifie que:

a) soit la classe de codage à laquelle cet objet de codage est appliqué doit être remplacée complètement ("**REPLACE STRUCTURE**"); dans le cas d'une classe de codage de la catégorie des offres d'options, le composant entier est remplacé; dans le cas d'un objet de codage de la classe **#CONDITIONAL-REPETITION** utilisé afin de définir un objet de codage pour une classe de la catégorie des chaînes de bits, des chaînes de caractères, des chaînes d'octets ou des répétitions, dans ce cas (si la condition d'étendue est satisfaite) la structure entière de chaîne de bits, chaîne de caractères, chaîne d'octets ou répétition est remplacée; ou

b) tous ses composants (sauf pour la spécification limitée aux structures) doivent être remplacés (avec la même action de remplacement pour tous les composants) ("**REPLACE COMPONENT**" ou "**REPLACE ALL COMPONENTS**"); ou

c) tous ses composants facultatifs (seulement pour spécification de remplacement complète) doivent être remplacés ("**REPLACE OPTIONALS**"); ou

d) tous ses composants non facultatifs (seulement pour spécification de remplacement complète) doivent être remplacés ("**REPLACE NON-OPTIONALS**"); ou

e) tous ses composants (seulement pour spécification de remplacement complète) doivent être remplacés avec une action de remplacement différente pour les composants facultatifs et pour les composants non facultatifs ("**REPLACE NON-OPTIONALS AND OPTIONALS**").

22.1.1.8 "**REPLACE COMPONENT**" est un synonyme de "**REPLACE ALL COMPONENTS**". Il serait normal mais non requis de l'utiliser s'il n'y a qu'un composant isolé.

22.1.1.9 Les composants facultatifs "**ENCODED BY**" spécifient un objet de codage pour la structure de remplacement.

22.1.1.10 Les composants facultatifs "**INSERT AT HEAD**" spécifient une structure de codage (l'insertion en tête de séquence) à insérer avant tous les composants de la classe (du constructeur) effectuant le remplacement. Il y a une seule insertion en tête de séquence pour chaque composant qui est remplacé et ces composants sont insérés dans l'ordre des composants originaux.

22.1.2 Restrictions de spécification

22.1.2.1 Exactement une des syntaxes permises entre "**REPLACE**" et "**WITH**" doit être utilisée.

22.1.2.2 Les structures de remplacement "**WITH**" doivent être des structures de codage paramétrées avec un seul paramètre de classe de codage. Lorsque ces structures sont spécifiées dans la syntaxe définie ci-dessus, seul le nom de référence de la classe de la structure doit être indiqué. Il ne doit pas y avoir de liste paramétrique dans cette utilisation des noms.

22.1.2.3 Ces structures paramétrées sont instanciées pendant l'action de remplacement avec un paramètre réel comme spécifié au § 22.1.3. L'utilisation du paramètre fictif dans les structures de remplacement paramétrées doit être compatible avec la classe du paramètre réel qui sera fourni dans l'action de remplacement.

NOTE – En particulier, si la clause "**REPLACE STRUCTURE**" est utilisée pour une classe de codage de la catégorie des étiquettes, le paramètre fictif ne peut apparaître dans la structure de remplacement que si une classe de codage de la catégorie des étiquettes est permise.

22.1.2.4 Les objets de codage "**ENCODED BY**" doivent être paramétrés pour les structures de codage "**WITH**". Ils doivent avoir un paramètre fictif (**#D**, par exemple) qui est une classe de codage, et ils doivent être définis dans une attribution d'objet de codage paramétré dans laquelle le gouverneur est la structure paramétrée de codage "**WITH**", instanciée avec **#D**. Lorsque ces objets sont spécifiés dans la syntaxe définie ci-dessus, seul le nom de référence de l'objet de codage doit être indiqué. Ils ne doivent pas avoir de liste paramétrique dans cette utilisation des noms.

22.1.2.5 Ces objets sont instanciés pendant l'action de remplacement avec un paramètre réel qui est le même que le paramètre réel utilisé afin d'instancier la structure de remplacements de codage "**WITH**" correspondante. Ils peuvent également avoir:

- (facultativement) un autre (mais un seul) paramètre fictif qui est un ensemble d'objets de codage; lorsqu'ils sont instanciés pendant l'action de remplacement, le paramètre réel pour ce paramètre fictif est l'ensemble courant d'objets de codage combinés;
- (conditionnellement) un autre (mais un seul) paramètre fictif qui est un paramètre **REFERENCE**. Ce paramètre doit être présent si et seulement si l'instruction "**INSERT AT HEAD**" est spécifiée. Lorsque les objets de codage sont instanciés pendant l'action de remplacement, le paramètre réel pour ce paramètre fictif est une référence à la structure correspondante "**INSERT AT HEAD**".

22.1.2.6 Tous les champs de la structure de remplacement qui ne font pas partie du paramètre de classe de codage sont des champs auxiliaires qui doivent être fixés par le codage de la structure de remplacement.

22.1.2.7 Les structures de codage "**INSERT AT HEAD**" ne doivent pas avoir de paramètres fictifs. Tous leurs champs sont des champs auxiliaires qui doivent être fixés par l'objet de codage "**ENCODED BY**" au moyen de son paramètre **REFERENCE**.

22.1.2.8 Si un objet de codage contient une clause "**REPLACE STRUCTURE**", il ne doit pas avoir de clause "**INSERT AT HEAD**" et doit avoir une clause "**ENCODED BY**".

22.1.3 Actions du codeur

22.1.3.1 Si un objet de codage d'une classe du groupe catégoriel des champs binaires ou de la catégorie des étiquettes spécifie "**REPLACE STRUCTURE**", dans ce cas un codeur doit remplacer la structure par une instanciation de la structure de remplacement, au moyen du nom de la structure originale en tant que paramètre réel.

22.1.3.2 Si un objet de codage d'une classe de la catégorie des constructeurs de codage spécifie "**REPLACE STRUCTURE**", dans ce cas un codeur doit remplacer la construction entière par une instanciation de la structure de remplacement, au moyen de la construction originale entière en tant que paramètre réel.

22.1.3.3 Si un objet de codage d'une classe de la catégorie des offres d'options spécifie "**REPLACE STRUCTURE**", dans ce cas un codeur doit remplacer le composant facultatif entier par une instanciation non facultative de la structure de remplacement. Le paramètre réel doit être un nom de structure caché (qui ne correspond à aucune autre structure et qui ne peut jamais avoir d'objets de codage). Ce nom de structure caché doit déréférencer au composant facultatif original entier (dont toute classe de la catégorie des étiquettes) sauf pour la classe de la catégorie des offres d'options.

22.1.3.4 Si un objet de codage d'une classe quelconque spécifie "**REPLACE COMPONENT**", "**REPLACE ALL COMPONENTS**", "**REPLACE OPTIONAL COMPONENTS**", ou "**REPLACE NON-OPTIONAL COMPONENTS**", dans ce cas un codeur doit remplacer le ou les composants spécifiés entiers par une instanciation non facultative de la structure de remplacement. Le paramètre réel doit être un nom de structure caché (qui ne correspond à aucune autre structure et qui ne peut jamais avoir d'objets de codage). Ce nom de structure caché doit déréférencer au composant facultatif original entier (dont toute classe de la catégorie des étiquettes) sauf pour une classe de la catégorie des offres d'options.

22.1.3.5 Toutes les valeurs abstraites et tous les numéros d'étiquette de la structure originale ou du composant original doivent être mappés sur les valeurs abstraites correspondantes et sur les numéros d'étiquette correspondants dans le paramètre réel de la structure de remplacement. Les valeurs des autres champs contenus dans la structure de remplacement doivent être fixées conformément à la spécification contenue dans l'objet de codage de la structure de remplacement.

22.1.3.6 Si une insertion en tête de séquence est spécifiée, dans ce cas le codeur doit insérer la structure de tête de séquence avant tous les composants de la structure dont l'objet de codage doit effectuer le remplacement. Les insertions en tête de séquence doivent être effectuées dans le même ordre textuel que les composants remplacés. Les valeurs des champs de cette structure doivent être fixées conformément à la spécification contenue dans l'objet de codage de la structure de remplacement.

NOTE – Ces structures seront normalement un simple champ d'entier fournissant un déterminant d'emplacement pour le champ remplacé.

22.1.3.7 Le codeur doit instancier l'objet ou les objets de codage de la structure de remplacement avec des paramètres réels comme suit:

a) le paramètre fictif qui est une classe de codage doit recevoir un paramètre réel qui est le même que le paramètre réel de l'instanciation de la structure de remplacement;

b) le paramètre fictif (si présent) qui est un paramètre **REFERENCE** doit recevoir un paramètre réel qui est une référence à la structure de tête de séquence insérée;

c) le paramètre fictif (si présent) qui est un ensemble d'objets de codage (dont le gouverneur est **#ENCODINGS**) doit recevoir un paramètre réel qui est l'ensemble courant d'objets de codage combinés.

22.1.3.8 Le codeur doit alors utiliser cet objet de codage instancié pour coder la structure de remplacement correspondante au lieu de l'ensemble d'objets de codage combinés.

NOTE – Le codage de l'insertion en tête de séquences est déterminé par l'application de l'ensemble courant d'objets de codage combinés.

22.1.4 Actions du décodeur

Un décodeur doit produire (pour une application) les valeurs abstraites de la structure originale qui a été en cours de codage, en masquant toute activité de remplacement (même si effectuée par application répétée de remplacements).

22.2 Spécification de préalignement et de bourrage

22.2.1 Propriétés, syntaxe et finalité du codage

22.2.1.1 La spécification de préalignement et de bourrage utilise les propriétés de codage suivantes:

```
&encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
&encoding-space-pre-padding        Padding DEFAULT zero,
&encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                   DEFAULT bits:'0'B
```

22.2.1.2 La syntaxe à utiliser pour la spécification de préalignement et de bourrage doit être la suivante:

```
[ALIGNED TO
      [NEXT]
      [ANY]
            &encoding-space-pre-alignment-unit
            [PADDING &encoding-space-pre-padding
                  [PATTERN &encoding-space-pre-pattern]]]
```

22.2.1.3 La définition des types utilisés en spécification de préalignement et de bourrage est la suivante:

```
Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
       dword32(32)} (0..256) --  (voir § 21.1)

Padding ::= ENUMERATED {zero, one, pattern, encoder-option} -- (voir § 21.9)

Pattern ::= CHOICE
      {bits                       BIT STRING,
       octets                     OCTET STRING,
       char8                      IA5String,
       char16                     BMPString,
       char32                     UniversalString,
       any-of-length              INTEGER (1..MAX),
       different                  ENUMERATED {any} }

Non-Null-Pattern ::= Pattern
      (ALL EXCEPT (bits:''B | octets:''H | char8:"" | char16:"" |
                   char32:"")) -- (voir § 21.10)
```

22.2.1.4 Les propriétés de codage du préalignement utilisent une valeur de type "**Unit**" afin de spécifier qu'un conteneur doit commencer à un multiple de "**Unit**" bits à partir du point d'alignement. Le point d'alignement est le début du codage du type auquel un module ELM a appliqué un codage, sauf lorsqu'il a été réinitialisé pour le codage d'un type de contenu par l'utilisation d'un objet de codage de la classe **#OUTER** (voir article 25). Les propriétés de codage de type "**Padding**" et "**Pattern**" sont utilisées afin de commander les bits qui fournissent le bourrage à l'alignement requis. La spécification de "**ALIGNED TO NEXT**" produit le nombre minimal de bits insérés. La

spécification de "**ALIGNED TO ANY**" laisse le nombre réel de bits insérés (sous réserve de la restriction ci-dessus à un multiple de "**Unit**") en tant qu'option des codeurs, et nécessite la spécification d'un pointeur de début.

22.2.2 Contraintes de spécification

22.2.2.1 Un au plus des mots "**NEXT**" et "**ANY**" doit être spécifié. En l'absence de leur spécification, c'est le mot "**NEXT**" qui est pris par défaut.

22.2.2.2 Si l'instruction "**ALIGNED TO ANY**" est spécifiée, dans ce cas la spécification d'objet de codage doit contenir la clause "**START-POINTER**".

22.2.3 Actions du codeur

22.2.3.1 Si le mot "**NEXT**" est spécifié (ou est pris par défaut), le codeur doit insérer le nombre minimal de bits nécessaire afin de garantir que le nombre total de bits dans le codage (à partir du point d'alignement jusqu'au début du conteneur, voir § 22.2.1.4) est un multiple de la propriété de codage de type "**Unit**".

22.2.3.2 Si le mot "**ANY**" est spécifié, le codeur doit insérer un nombre de bits dépendant du codeur, à condition que le nombre total de bits dans le codage (à partir du point d'alignement) soit un multiple de la propriété de codage de type "**Unit**".

22.2.3.3 Les bits insérés doivent être réglés de sorte que le premier bit inséré soit le bit initial de "**Pattern**" et ainsi de suite. S'il faut plus de bits qu'il n'y en a dans la propriété de codage de type "**Pattern**", dans ce cas la séquence doit être réutilisée, bit de poids fort en premier.

22.2.4 Actions du décodeur

22.2.4.1 Le décodeur doit déterminer le nombre de bits insérés à partir des actions du codeur si le mot "**NEXT**" est spécifié.

22.2.4.2 Le décodeur doit déterminer le nombre de bits insérés à partir de la spécification du pointeur de début si le mot "**ANY**" est spécifié.

22.2.4.3 Dans tous les cas, le décodeur doit rejeter les bits insérés en transparence par rapport à l'application. Il ne doit pas diagnostiquer une erreur de codeur ou de spécification si les bits ne sont pas en accord avec les actions spécifiées des codeurs.

22.3 Spécification du pointeur de début

22.3.1 Propriétés, syntaxe et finalité du codage

22.3.1.1 La spécification du pointeur de début utilise les propriétés de codage suivantes:

```
&start-pointer                    REFERENCE   OPTIONAL,
&start-pointer-unit               Unit (ALL EXCEPT repetitions) DEFAULT bit,
&Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL
```

22.3.1.2 La syntaxe à utiliser pour la spécification du pointeur de début doit être la suivante:

```
[START-POINTER   &start-pointer
      [MULTIPLE OF     &start-pointer-unit]
      [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
```

22.3.1.3 La définition du type utilisé en spécification du pointeur de début est la suivante:

```
Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
       dword32(32)} (0..256) --  (voir § 21.1)
```

22.3.1.4 Cette spécification identifie le début de l'espace de codage pour un élément. Si le début de l'espace de codage pour l'élément est un décalage de "n" "**MULTIPLE OF**" unités, dans ce cas la valeur placée dans le champ cité en référence par la propriété de codage "**START-POINTER**" est la valeur obtenue par l'application de "**ENCODER-TRANSFORMS**" à "n".

NOTE 1 – Si "**MULTIPLE OF**" n'est pas "**bits**", cela implique que le décalage à partir du début du champ cité en référence par la propriété de codage "**START-POINTER**" jusqu'au début de l'espace de codage doit être un multiple entier de "**MULTIPLE OF**" unités.

NOTE 2 – Il y aura en général des codages d'autres éléments, et éventuellement d'autres points de départ entre le champ cité en référence par la propriété de codage "**START-POINTER**" et le début du codage de cet élément.

22.3.2 Contraintes de spécification

22.3.2.1 Si le champ "**ENCODER-TRANSFORMS**" n'est pas présent, dans ce cas "**START-POINTER**" doit être une classe de la catégorie des entiers.

22.3.2.2 Si le champ "**ENCODER-TRANSFORMS**" est présent, dans ce cas "**START-POINTER**" doit être une classe avec une catégorie qui peut coder la valeur du résultat de la transformée finale dans le champ "**ENCODER-TRANSFORMS**".

22.3.2.3 Il y a erreur de spécification ou d'application ECN si une éventuelle transformée contenue dans le champ "**ENCODER-TRANSFORMS**" n'est pas réversible pour la valeur abstraite à laquelle elle est appliquée. La première transformée doit avoir une source qui est un entier.

22.3.3 Actions du codeur

22.3.3.1 Le codeur doit déterminer le nombre "n" de "**MULTIPLE OF**" unités à partir du début du codage du champ "**START-POINTER**" (après éventuel préalignement de ce champ) jusqu'au début du codage de l'élément avec la spécification de pointeur de début (après éventuel préalignement de cet élément). Il y a erreur de spécification ECN si "n" n'est pas un nombre entier. Si l'élément codé est facultatif et est absent, dans ce cas "n" doit être mis à zéro.

22.3.3.2 La valeur "n" doit être transformée au moyen du champ "**ENCODER-TRANSFORMS**" (si présent) afin de produire une valeur théorique "m". Si cette valeur résultante "m" n'est pas une valeur abstraite qui peut être associée à la classe de codage du pointeur "**START-POINTER**", dans ce cas il y a erreur de spécification ECN et le codage ne doit pas continuer. Sinon la valeur "m" doit être la valeur codée dans le champ cité en référence par "**START-POINTER**".

NOTE – L'objet de codage appliqué au champ cité en référence par "**START-POINTER**" déterminera le codage de la valeur "m".

22.3.4 Actions du décodeur

22.3.4.1 Le décodeur doit déterminer la valeur théorique "m" dans le champ cité en référence par "**START-POINTER**" et doit utiliser la connaissance des actions de codeur afin d'inverser les transformées (éventuelles) et de produire la valeur entière "n".

22.3.4.2 Si "n" est zéro, dans ce cas le décodeur doit diagnostiquer une erreur de codeur si l'élément en cours de décodage n'est pas un élément facultatif avec une offre d'options déterminant une spécification d'offre d'options par le pointeur de début. Si "n" est zéro et que l'élément en cours de décodage soit un élément facultatif avec une spécification d'offre d'options déterminant des offres d'options par le pointeur de début, dans ce cas le décodeur doit déterminer que l'élément est absent.

22.3.4.3 La valeur "n" est multipliée par "**MULTIPLE OF**" et le début du codage du champ "**START-POINTER**" est ajouté afin de produire une position "p". Si "p" est une position dans le codage qui est antérieure au point de décodage actuel, dans ce cas le décodeur doit diagnostiquer une erreur de codage.

22.3.4.4 Si "p" est une position dans le codage qui est égale ou supérieure au point de décodage actuel, dans ce cas le décodeur doit négliger sans notification tous bits jusqu'à la position supérieure "p" et doit continuer le décodage de cet élément à partir de cette position "p".

22.4 Spécification de l'espace de codage

22.4.1 Propriétés, syntaxe et finalité du codage

22.4.1.1 La spécification de l'espace de codage utilise les propriétés de codage suivantes:

```
&encoding-space-size              EncodingSpaceSize
                                  DEFAULT self-delimiting-values,
&encoding-space-unit              Unit (ALL EXCEPT repetitions)
                                  DEFAULT bit,
&encoding-space-determination     EncodingSpaceDetermination
                                  DEFAULT field-to-be-set,
&encoding-space-reference         REFERENCE OPTIONAL,
&Encoder-transforms               #TRANSFORM ORDERED OPTIONAL,
&Decoder-transforms               #TRANSFORM ORDERED OPTIONAL
```

22.4.1.2 La syntaxe à utiliser pour la spécification de l'espace de codage doit être la suivante:

```
ENCODING-SPACE
      [SIZE &encoding-space-size
            [MULTIPLE OF &encoding-space-unit]]
      [DETERMINED BY &encoding-space-determination]
      [USING &encoding-space-reference
            [ENCODER-TRANSFORMS &Encoder-transforms]
            [DECODER-TRANSFORMS &Decoder-transforms]]
```

22.4.1.3 La définition des types utilisés dans cette spécification est la suivante:

```
EncodingSpaceSize ::= INTEGER
      { encoder-option-with-determinant(-3),
        variable-with-determinant(-2),
        self-delimiting-values(-1),
        fixed-to-max(0)} (-3..MAX) -- (voir § 21.2)

Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
       dword32(32)} (0..256) --  (voir § 21.1)

EncodingSpaceDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, container}  -- (voir § 21.3)
```

22.4.1.4 L'objet de cette spécification vise à déterminer les actions de codeur et de décodeur afin de garantir qu'un décodeur peut correctement déterminer l'extrémité d'un espace de codage.

NOTE – Un codage de valeur réel ne remplit pas nécessairement tout l'espace de codage, et la reconstitution du codage de valeur par un décodeur nécessitera en général également des actions spécifiées de bourrage et de justification de valeur (voir § 22.8).

22.4.1.5 La signification des propriétés de codage de type "**Unit**", "**EncodingSpaceSize**" et "**EncodingSpaceDetermination**" ont été données aux § 21.1, 21.2 et 21.3. Ensemble, elles spécifient la façon dont l'extrémité de l'espace de codage est déterminée pour cet élément.

NOTE – La propriété "**variable-with-determinant**" peut être spécifiée même si l'espace de codage est de taille fixe, si le spécificateur ECN exige qu'un déterminant de longueur soit inclus, même s'il n'est pas nécessaire.

22.4.1.6 La spécification "**USING**" est une référence qui permet à un décodeur de déterminer l'extrémité de l'espace de codage. C'est une référence à un champ auxiliaire ou à un champ acheminant des valeurs abstraites, ou à un conteneur, selon la valeur de "**DETERMINED BY**".

22.4.2 Restrictions de spécification

22.4.2.1 Si "**SIZE**" est "**variable-with-determinant**" et si "**DETERMINED BY**" n'est pas présent, dans ce cas la valeur par défaut ("**field-to-be-set**") est prise en compte.

22.4.2.2 Le champ "**USING**" doit être spécifié si et seulement si "**SIZE**" est "**variable-with-determinant**" ou "**encoder-option-with-determinant**".

22.4.2.3 Le champ "**ENCODER-TRANSFORMS**" doit être présent seulement si la valeur de "**DETERMINED BY**" est activée à (prend par défaut) "**field-to-be-set**". La référence "**USING**" doit alors être un champ auxiliaire de la catégorie des chaînes de bits, des chaînes de caractères ou des entiers.

22.4.2.4 Il y a erreur de spécification ou d'application ECN si une éventuelle transformée dans le champ "**ENCODER-TRANSFORMS**" n'est pas réversible pour la valeur abstraite à laquelle elle est appliquée. La première transformée doit avoir une source qui est un entier et la dernière transformée doit avoir un résultat qui peut être codé par la classe du champ cité en référence par "**USING**".

22.4.2.5 "**DECODER-TRANSFORMS**" doit être présent seulement si "**DETERMINED BY**" est activé à "**field-to-be-used**". La première transformée doit avoir une source qui est la même que la catégorie du champ cité en référence par "**USING**", champ qui ne doit pas être un champ auxiliaire. La dernière transformée doit avoir un résultat qui est un entier.

22.4.2.6 La propriété de codage "**USING**", si présente, doit être une référence à un champ qui est présent dans le codage antérieur au champ en cours de codage. Il y a erreur d'application ou de spécification ECN si, dans une instance de codage, le champ en cours de codage est présent mais le champ cité en référence par la propriété de codage "**USING**" est absent (grâce à l'application d'offres d'options).

22.4.2.7 Si "**DETERMINED BY**" est "**container**", la référence "**USING**" doit viser une concaténation ou une répétition (ou une chaîne de bits ou une chaîne d'octets avec un type contenu) de laquelle l'élément codé est un composant (ou un composant de composant, jusqu'à une profondeur quelconque). Il y a erreur d'application ou de spécification ECN si,

dans une instance de codage, des éléments ultérieurs à l'intérieur de la même concaténation ou répétition doivent être codés.

22.4.2.8 Cette spécification est considérée comme fixée si le mot clé "**ENCODING-SPACE**" est utilisé, et il est obligatoire qu'il soit activé à tous endroits dans la syntaxe définie où cela est autorisé. La fixation par défaut de toutes les propriétés de codage de ce groupe (par exemple, l'utilisation de "**ENCODING-SPACE**" seul) ne satisferait pas les contraintes ci-dessus.

22.4.3 Actions du codeur

22.4.3.1 Les codeurs ne doivent pas produire de codages si les conditions du § 22.4.2 ne sont pas satisfaites.

22.4.3.2 Si "**SIZE**" est une valeur positive, dans ce cas l'espace de codage est ce multiple de "**MULTIPLE OF**" unités et il n'y a pas d'autre action du codeur.

22.4.3.3 Si "**SIZE**" n'est pas mis à une valeur positive, dans ce cas le codeur doit déterminer la taille ("s", par exemple) de l'espace de codage en "**MULTIPLE OF**" unités à partir de la spécification de codage de valeur. Cette détermination est spécifiée dans les paragraphes relatifs à la spécification de codage de valeur.

22.4.3.4 Si "**SIZE**" est "**encoder-option-with-determinant**", alors le codeur (à titre d'option de codeur) peut augmenter la taille "s" (telle que déterminée au § 22.4.3.3) en "**MULTIPLE OF**" unités à partir de celle qui a été déterminée à partir de la spécification de codage de valeur, jusqu'à toute valeur qui peut être codée dans le déterminant associé.

22.4.3.5 Si "**SIZE**" a la valeur "**fixed-to-max**" ou "**self-delimiting-values**", dans ce cas il n'y a pas d'autre action du codeur.

22.4.3.6 Si "**SIZE**" est "**variable-with-determinant**" et "**DETERMINED BY**" est "**container**", dans ce cas il n'y a pas d'autre action du codeur.

22.4.3.7 Si "**DETERMINED BY**" est "**field-to-be-set**", dans ce cas le codeur doit appliquer les transformées spécifiées par "**ENCODER-TRANSFORMS**" (le cas échéant) jusqu'à la valeur "s" afin de produire une valeur qui doit être codée dans la référence "**USING**".

NOTE – Le codage de la référence "**USING**" (champ binaire "A", par exemple) dans ce cas apparaît plus tôt dans le codage que le codage de ce champ (champ binaire "B", par exemple) et un codeur aura besoin de différer le codage de champ binaire "A" jusqu'à ce que la valeur à coder ait été déterminée par le codage de champ binaire "B".

22.4.3.8 Si "**DETERMINED BY**" est "**field-to-be-used**", alors le codeur doit vérifier que la valeur contenue dans la référence "**USING**", après transformation par le champ "**DECODER-TRANSFORMS**" (si présent), est égale à "s". Il y a erreur d'application si cette condition n'est pas satisfaite et le codage ne doit pas continuer.

22.4.4 Actions du décodeur

22.4.4.1 Si "**SIZE**" est une valeur positive, dans ce cas le décodeur détermine l'espace de codage en tant que multiple de "**MULTIPLE OF**" unités.

22.4.4.2 Si "**SIZE**" est à "**fixed-to-max**" ou à "**self-delimiting-values**", dans ce cas le décodeur doit déterminer l'extrémité de l'espace de codage conformément à la spécification du codage de valeur. Cette détermination est spécifiée dans les paragraphes relatifs à la spécification de codage de valeur.

22.4.4.3 Si "**SIZE**" est "**variable-with-determinant**" et si "**DETERMINED BY**" est activé à "**container**", dans ce cas le décodeur doit utiliser l'extrémité du conteneur spécifiée par "**USING**" en tant qu'extrémité de l'espace de codage.

22.4.4.4 Si "**SIZE**" est "**variable-with-determinant**" et "**DETERMINED BY**" est activé à (ou prend par défaut) la valeur "**field-to-be-set**", dans ce cas le décodeur doit reconstituer la valeur "s" par l'application de l'inversion du champ "**ENCODER-TRANSFORMS**" (si présent) à la valeur de la référence "**USING**".

22.4.4.5 Si "**DETERMINED BY**" est "**field-to-be-used**", alors le décodeur doit reconstituer la valeur "s" par l'application du champ "**DECODER-TRANSFORMS**" (si présent) à la valeur de ce champ.

22.5 Détermination de l'offre d'options

22.5.1 Propriétés, syntaxe et finalité du codage

22.5.1.1 La détermination de l'offre d'options utilise les propriétés de codage suivantes:

```
&optionality-determination        OptionalityDetermination
                                  DEFAULT field-to-be-set,
&optionality-reference            REFERENCE OPTIONAL,
&Encoder-transforms               #TRANSFORM ORDERED OPTIONAL,
&Decoder-transforms               #TRANSFORM ORDERED OPTIONAL,
&handle-id                        PrintableString
                                  DEFAULT "default-handle"
```

22.5.1.2 La syntaxe à utiliser pour la détermination de l'offre d'options doit être la suivante:

```
PRESENCE
      [DETERMINED BY &optionality-determination
            [HANDLE &handle-id]]
      [USING &optionality-reference
            [ENCODER-TRANSFORMS &Encoder-transforms]
            [DECODER-TRANSFORMS &Decoder-transforms]]
```

22.5.1.3 La définition de types utilisée dans la détermination de l'offre d'options est la suivante:

```
OptionalityDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, container, handle, pointer} -- (voir
      § 21.5)
```

22.5.1.4 L'objet de cette spécification vise à spécifier des règles garantissant qu'un décodeur peut correctement déterminer si un codeur a codé la valeur d'un composant facultatif. Lorsqu'un pointeur est utilisé afin de déterminer l'offre d'options, la spécification du préalignement et du pointeur de début est également requise.

22.5.1.5 Un codeur codera la valeur d'un composant facultatif s'il y est requis par l'application, à moins qu'un tel codage ne soit en violation des règles régissant la présence de composants facultatifs.

NOTE – Un exemple de violation d'une telle règle serait que la présence d'un composant facultatif (absent) a été déterminée par l'extrémité d'un conteneur et que l'application a demandé que des composants facultatifs ultérieurs soient codés dans le même conteneur.

22.5.1.6 Cette spécification est considérée comme fixée si le mot clé "**PRESENCE**" est utilisé, et il est obligatoire qu'elle soit fixée à tous endroits de la syntaxe définie où cela est autorisé. La fixation par défaut de toutes les autres parties de cette syntaxe définie (par exemple, l'utilisation du mot "**PRESENCE**" seul) ne satisferait pas les contraintes ci-dessus.

22.5.2 Restrictions de spécification

22.5.2.1 Si "**DETERMINED BY**" n'est pas présent, dans ce cas la valeur par défaut ("**field-to-be-set**") est prise en compte.

22.5.2.2 "**HANDLE**" ne doit pas être spécifié à moins que "**DETERMINED BY**" ne soit "**handle**".

22.5.2.3 "**USING**" ne doit pas être spécifié si "**DETERMINED BY**" est "**handle**" ou "**pointer**".

22.5.2.4 Si "**DETERMINED BY**" est "**pointer**", il doit y avoir une spécification "**START-POINTER**" dans le même objet de codage (voir § 22.3).

NOTE – Une spécification du pointeur de début normalement nécessite également une spécification de préalignement avec "**ALIGNED TO ANY**" (voir § 22.2).

22.5.2.5 Si "**HANDLE**" est spécifié, dans ce cas le composant dont la présence est déterminée ainsi que tous les codages facultatifs suivants et le prochain codage obligatoire (si présent) doivent tous être produits par des objets de codage dont toutes les spécifications offrent un pointeur d'identification ayant le même nom que "**HANDLE**". Le prochain codage obligatoire peut être un composant de la concaténation contenant le composant facultatif, ou peut être un codage suivant la concaténation. La valeur du pointeur d'identification doit être différente pour tous ces composants.

NOTE – Il est prescrit que les bits qui forment un pointeur d'identification aient la même valeur pour toutes les valeurs abstraites codées par un objet de codage contenant ce pointeur d'identification (voir § 22.9.2.2).

22.5.2.6 "**ENCODER-TRANSFORMS**" doit être présent seulement si "**DETERMINED BY**" est activé à (ou prend par défaut) "**field-to-be-set**". La référence "**USING**" dans ce cas doit être un champ auxiliaire de catégorie chaînes de bits, booléens, chaînes de caractères ou entiers.

22.5.2.7 Il y a erreur de spécification ou d'application ECN si une éventuelle transformée dans le champ "**ENCODER-TRANSFORMS**" n'est pas réversible pour la valeur abstraite à laquelle elle est appliquée. La première transformée doit avoir une source qui est booléenne et la dernière transformée doit avoir un résultat qui peut être codé par la classe du champ cité en référence par "**USING**".

22.5.2.8 "**DECODER-TRANSFORMS**" ne doit être présent que si "**DETERMINED BY**" est activé à "**field-to-be-used**". La première transformée doit avoir une source qui est la même que la catégorie du champ cité en référence par "**USING**", lequel ne doit pas être un champ auxiliaire. La dernière transformée doit avoir un résultat qui est booléen.

22.5.2.9 La propriété de codage "**USING**", si présente, doit être une référence à un champ qui est présent dans le codage antérieur au champ dont la présence est déterminée. Il y a erreur d'application ou de spécification ECN si, dans une instance de codage, le champ cité en référence par la propriété de codage "**USING**" est requis par un décodeur mais est absent (par l'application d'offres d'options).

22.5.2.10 Si "**DETERMINED BY**" est "**container**", la référence "**USING**" doit renvoyer à une concaténation ou à une répétition (ou à une chaîne de bits ou à une chaîne d'octets avec un type contenu) dans laquelle l'élément codé est un composant (ou un composant d'un composant, jusqu'à une profondeur quelconque). Il y a erreur d'application ou de spécification ECN si, dans une instance de codage, des éléments ultérieurs à l'intérieur de la même concaténation ou répétition doivent être codés lorsque le composant dont l'offre d'options est déterminée est absent.

22.5.2.11 Si "**DETERMINED BY**" est "**container**", dans ce cas il y a erreur de spécification ECN si l'une quelconque des valeurs abstraites du composant facultatif a un codage qui est zéro bit.

22.5.3 Actions du codeur

22.5.3.1 Les codeurs ne doivent pas produire de codages si les conditions du § 22.5.2 ne sont pas satisfaites.

22.5.3.2 Un codeur doit déterminer si l'application vise que le composant facultatif soit codé et doit créer une valeur booléenne théorique "**element-is-present**" mise à "**TRUE**" si la valeur du composant doit être codée et à "**FALSE**" sinon.

22.5.3.3 Si "**DETERMINED BY**" est "**field-to-be-set**", dans ce cas le codeur doit appliquer les transformées spécifiées par "**ENCODER-TRANSFORMS**" (si présent) à la valeur booléenne théorique "**element-is-present**" afin de produire une valeur qui doit être codée dans la référence "**USING**".

NOTE – Le codage de la référence "**USING**" dans ce cas apparaît plus tôt dans le codage que le codage de ce champ, et un codeur aura besoin de suspendre le codage de ce champ jusqu'à ce que la valeur à coder ait été déterminée par le codage de ce champ.

22.5.3.4 Si "**DETERMINED BY**" est "**field-to-be-used**", alors le codeur doit vérifier que la valeur contenue dans la référence "**USING**" après transformation par le champ "**DECODER-TRANSFORMS**" (si présent) est une valeur booléenne égale à la valeur théorique "**element-is-present**". Il y a erreur d'application si cette condition n'est pas satisfaite, et le codage ne doit pas continuer.

22.5.3.5 Si "**DETERMINED BY**" est "**container**", il n'y a pas d'autre action nécessaire par le codeur, sauf à détecter une erreur et à cesser le codage si l'application demande le codage d'autres composants dans le conteneur "**USING**" lorsque la valeur théorique "**element-is-present**" est fausse pour ce composant facultatif.

22.5.3.6 Si "**DETERMINED BY**" est "**handle**", il n'y a pas d'autre action nécessaire par le codeur.

22.5.3.7 Si "**DETERMINED BY**" est "**pointer**", alors il n'y a pas d'action requise du codeur sauf celles des spécifications connexes de préalignement (si présent) et de pointeur de début.

22.5.4 Actions du décodeur

22.5.4.1 Si "**DETERMINED BY**" est activé à (ou prend par défaut) "**field-to-be-set**", dans ce cas le décodeur doit reconstituer la valeur "**element-is-present**" par l'application de l'inversion du champ "**ENCODER-TRANSFORMS**" (si présent) à la valeur de la référence "**USING**".

22.5.4.2 Si "**DETERMINED BY**" est "**field-to-be-used**", alors le décodeur doit reconstituer la valeur théorique "**element-is-present**" par l'application du champ "**DECODER-TRANSFORMS**" (si présent) à la valeur de ce champ.

22.5.4.3 Si "**DETERMINED BY**" est "**container**", alors le décodeur doit fixer la valeur théorique "**element-is-present**" à **TRUE** si et seulement s'il y a au moins un bit restant dans le conteneur "**USING**".

22.5.4.4 Si "**DETERMINED BY**" est "**handle**", dans ce cas le décodeur doit déterminer la valeur du pointeur d'identification spécifié. Si la valeur correspond à celle du pointeur d'identification du composant facultatif, dans ce cas le décodeur doit fixer la valeur théorique "**element-is-present**" à **TRUE**, sinon le décodeur doit la fixer à **FALSE**.

22.5.4.5 Si "**DETERMINED BY**" est "**pointer**", alors le décodeur doit procéder comme spécifié au § 22.3 afin de déterminer la valeur théorique de "**element-is-present**".

22.5.4.6 Si le décodeur détermine (par n'importe lequel des moyens ci-dessus) que la valeur théorique "**element-is-present**" est **FALSE**, dans ce cas le décodage s'effectue au composant suivant; sinon le décodeur s'attend à un codage d'une valeur du composant facultatif et diagnostiquera une erreur de codage si un tel codage n'est pas présent.

22.6 Détermination des options

22.6.1 Propriétés, syntaxe et finalité du codage

22.6.1.1 La détermination des options utilise les propriétés de codage suivantes:

```
&alternative-determination      AlternativeDetermination
                                DEFAULT field-to-be-set,
&alternative-reference          REFERENCE OPTIONAL,
&Encoder-transforms             #TRANSFORM ORDERED OPTIONAL,
&Decoder-transforms             #TRANSFORM ORDERED OPTIONAL,
&handle-id                      PrintableString
                                DEFAULT "default-handle",
&alternative-ordering           ENUMERATED {textual, tag}
                                DEFAULT textual
```

22.6.1.2 La syntaxe à utiliser pour la détermination des options doit être la suivante:

```
ALTERNATIVE
      [DETERMINED BY &alternative-determination
            [HANDLE &handle-id]]
      [USING &alternative-reference
            [ORDER &alternative-ordering]
            [ENCODER-TRANSFORMS &Encoder-transforms]
            [DECODER-TRANSFORMS &Decoder-transforms]]
```

22.6.1.3 La définition des types utilisés pour la détermination des options est la suivante:

```
AlternativeDetermination ::=
      ENUMERATED {field-to-be-set, field-to-be-used, handle} -- (voir § 21.6)
```

22.6.1.4 L'objet de cette spécification vise à déterminer les règles garantissant qu'un décodeur peut correctement indiquer le composant d'une classe de codage de la catégorie des options qui a été codé.

22.6.2 Restrictions de spécification

22.6.2.1 Si "**DETERMINED BY**" n'est pas présent, dans ce cas la valeur par défaut ("**field-to-be-set**") est prise en compte.

22.6.2.2 "**HANDLE**" ne doit pas être spécifié à moins que "**DETERMINED BY**" ne soit "**handle**".

22.6.2.3 "**USING**" ne doit pas être spécifié si "**DETERMINED BY**" est "**handle**".

22.6.2.4 Si "**HANDLE**" est spécifié, dans ce cas toutes les options de la classe de codage de la catégorie des options doivent être codées par des objets de codage dont la spécification fait apparaître et définit un pointeur d'identification ayant le même nom que "**HANDLE**" et avec la même valeur du pointeur d'identification. La valeur du pointeur d'identification doit être différente pour toutes ces options.

NOTE – Il est prescrit qu'un pointeur d'identification doit avoir la même valeur pour toutes les valeurs abstraites codées par un objet de codage contenant ce pointeur d'identification (voir § 22.9.2.2).

22.6.2.5 Le champ "**ENCODER-TRANSFORMS**" ne doit être présent que si "**DETERMINED BY**" est activé à (ou prend par défaut) "**field-to-be-set**". La première transformée doit avoir une source qui est un entier et la dernière transformée doit avoir un résultat qui peut être codé par la classe du champ cité en référence par "**USING**".

22.6.2.6 Il y a erreur de spécification ou d'application ECN si une éventuelle transformée dans le champ "**ENCODER-TRANSFORMS**" n'est pas réversible pour la valeur abstraite à laquelle elle est appliquée.

22.6.2.7 "**DECODER-TRANSFORMS**" ne doit être présent que si "**DETERMINED BY**" est activé à "**field-to-be-used**". La première transformée doit avoir une source qui est la même que la catégorie du champ cité en référence par "**USING**", qui ne doit pas être un champ auxiliaire. La dernière transformée doit avoir un résultat qui est un entier.

22.6.2.8 La propriété de codage "**USING**", si présente, doit être une référence à un champ présent dans le codage antérieur au codage de l'option. Il y a erreur d'application ou de spécification ECN si, dans une instance de codage, le

champ cité en référence par la propriété de codage "**USING**" est requis par un décodeur mais est absent (par l'application d'offres d'options).

22.6.2.9 Cette spécification est considérée comme fixée si le mot clé "**ALTERNATIVE**" est utilisé et il est obligatoire qu'elle soit fixée à tous les endroits de la syntaxe définie où cela est autorisé. La fixation par défaut de toutes les autres parties de cette syntaxe définie (par exemple, l'utilisation de "**ALTERNATIVE**" seul) ne satisferait pas les contraintes ci-dessus.

22.6.2.10 Si "**ORDER**" est "**tag**", dans ce cas chaque option doit commencer par une classe de codage de la catégorie des étiquettes. Le numéro d'étiquette associé à cette classe est appelé l'étiquette de composant.

22.6.2.11 L'étiquette de composant de chaque option doit être distincte.

22.6.3 Actions du codeur

22.6.3.1 Les codeurs ne doivent pas produire de codages si les conditions du § 22.6.2 ne sont pas satisfaites.

22.6.3.2 Un codeur doit déterminer quelle option l'application vise à coder et doit créer une valeur entière théorique "**alternative-index**" afin de désigner cette option.

22.6.3.3 La valeur "**alternative-index**" doit être zéro pour la première option, un pour la suivante, et ainsi de suite, où l'ordre des options est déterminé par "**ORDER**".

22.6.3.4 Si "**ORDER**" est "textual", l'ordre textuel contenu dans la spécification de type ASN.1 ou dans la définition de structure ECN doit être utilisé. Si "**ORDER**" est "**tag**", dans ce cas l'ordre doit être celui des numéros d'étiquette contenus dans l'étiquette de composant (le plus faible numéro d'étiquette d'abord).

22.6.3.5 Si "**DETERMINED BY**" est "**field-to-be-set**", dans ce cas le codeur doit appliquer les transformées spécifiées par "**ENCODER-TRANSFORMS**" (si présent) à la valeur théorique "**alternative-index**" afin de produire une valeur qui doit être codée dans la référence "**USING**".

NOTE – Le codage de la référence "**USING**" dans ce cas apparaît plus tôt dans le codage que le codage de l'option, et un codeur aura besoin de suspendre le codage de ce champ jusqu'à ce que l'option à coder ait été déterminée.

22.6.3.6 Si "**DETERMINED BY**" est "**field-to-be-used**", alors le codeur doit vérifier que la valeur contenue dans la référence "**USING**" après transformation par le champ "**DECODER-TRANSFORMS**" (si présent) est une valeur entière égale à la valeur théorique "**alternative-index**". Il y a erreur d'application si cette condition n'est pas satisfaite et le codage ne doit pas continuer.

22.6.3.7 Si "**DETERMINED BY**" est "**handle**", il n'y a pas d'autre action nécessaire par le codeur.

22.6.4 Actions du décodeur

22.6.4.1 Le décodeur doit utiliser "**ORDER**" comme spécifié pour les actions du codeur afin de déterminer la valeur d'indice d'option ("**alternative-index**") qui est associée à chaque option, et doit impliquer la présence d'un codage de l'option associée une fois qu'une valeur "**alternative-index**" théorique a été déterminée.

22.6.4.2 Si "**DETERMINED BY**" est activé à (ou prend par défaut) "**field-to-be-set**", dans ce cas le décodeur doit reconstituer la valeur "**alternative-index**" par l'application de l'inversion "**ENCODER-TRANSFORMS**" (si ce champ présent) à la valeur de la référence "**USING**".

22.6.4.3 Si "**DETERMINED BY**" est "**field-to-be-used**", alors le décodeur doit reconstituer la valeur théorique "**alternative-index**" par l'application du champ "**DECODER-TRANSFORMS**" (si présent) à la valeur de ce champ.

22.6.4.4 Si "**DETERMINED BY**" est "**handle**", dans ce cas le décodeur doit déterminer la valeur du pointeur d'identification. Cette valeur doit être comparée à la valeur du pointeur d'identification de chaque option. Si aucune ne correspond, dans ce cas le décodeur doit diagnostiquer une erreur de codeur. Sinon la valeur théorique "**alternative-index**" doit être mise à l'option concordante.

22.7 Spécification de l'espace de répétition

22.7.1 Propriétés, syntaxe et finalité du codage

22.7.1.1 La spécification de l'espace de répétition utilise les propriétés de codage suivantes:

```
&repetition-space-size          EncodingSpaceSize
                                DEFAULT self-delimiting-values,
&repetition-space-unit          Unit
                                DEFAULT bit,
```

```
&repetition-space-determination   RepetitionSpaceDetermination
                                  DEFAULT field-to-be-set,
&main-reference                   REFERENCE OPTIONAL,
&Encoder-transforms               #TRANSFORM ORDERED OPTIONAL,
&Decoder-transforms               #TRANSFORM ORDERED OPTIONAL,
&handle-id                        PrintableString
                                  DEFAULT "default-handle",
&termination-pattern              Non-Null-Pattern (ALL EXCEPT
                                  different:any) DEFAULT bits '0'B
```

22.7.1.2 La syntaxe à utiliser pour la spécification de l'espace de répétition doit être la suivante:

```
REPETITION-SPACE
      [SIZE &repetition-space-size
            [MULTIPLE OF &repetition-space-unit]]
      [DETERMINED BY &repetition-space-determination
            [HANDLE &handle-id]]
      [USING &main-reference
            [ENCODER-TRANSFORMS &Encoder-transforms]
            [DECODER-TRANSFORMS &Decoder-transforms]]
      [PATTERN &termination-pattern]
```

22.7.1.3 La définition des types utilisés dans cette spécification est la suivante:

```
EncodingSpaceSize ::= INTEGER
      { encoder-option-with-determinant(-3),
        variable-with-determinant(-2),
        self-delimiting-values(-1),
        fixed-to-max(0)} (-3..MAX) -- (voir § 21.2)

Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
       dword32(32)} (0..256) --  (voir § 21.1)

RepetitionSpaceDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, flag-to-be-set, flag-to-be-used,
       container, pattern, handle, not-needed} -- (voir § 21.7)

Non-Null-Pattern ::= Pattern
      (ALL EXCEPT (bits:''B | octets:''H | char8:"" | char16:"" |
                   char32:"")) -- (voir § 21.10.2)
```

22.7.1.4 L'objet de cette spécification vise à déterminer des actions de codeur et de décodeur afin de garantir qu'un décodeur peut correctement déterminer l'extrémité de l'espace de codage occupé par une répétition.

NOTE – Un codage réel de répétition ne remplit pas nécessairement tout l'espace de codage et la reconstitution du codage de répétition par un décodeur nécessitera également, en général, des actions spécifiées de bourrage et de justification de valeur (voir § 22.8).

22.7.1.5 La signification des propriétés de codage de type "**Unit**", "**EncodingSpaceSize**", et "**RepetitionSpaceDetermination**" a été donnée aux § 21.1, 21.2, et 21.7. Ensemble, ces propriétés spécifient la façon dont est déterminée l'extrémité de l'espace de codage pour répétitions.

NOTE – Si le spécificateur de notation ECN nécessite qu'un déterminant de longueur soit inclus, la valeur "**variable-with-determinant**" de "**SIZE**" peut être spécifiée même si l'espace de répétition est de taille fixe.

22.7.1.6 La spécification "**USING**" est une référence à un champ auxiliaire ou à un champ acheminant des valeurs abstraites, ou à un conteneur, selon la valeur de "**DETERMINED BY**".

22.7.2 Contraintes de spécification

22.7.2.1 Si "**SIZE**" est "**variable-with-determinant**" et si "**DETERMINED BY**" n'est pas présent, dans ce cas la valeur par défaut ("**field-to-be-set**") est prise en compte.

22.7.2.2 "**USING**" doit être spécifié si et seulement si "**SIZE**" est "**variable-with-determinant**" et si "**DETERMINED BY**" est "**field-to-be-set**" ou "**field-to-be-used**" ou "**flag-to-be-set**" ou "**flag-to-be-used**", ou "**container**".

22.7.2.3 "**ENCODER-TRANSFORMS**" doit être présent seulement si "**DETERMINED BY**" est activé à (ou prend par défaut) "**field-to-be-set**" ou "**flag-to-be-set**". La première transformée doit avoir une source qui est un entier si "**DETERMINED BY**" est "**field-to-be-set**" et qui est booléenne si "**DETERMINED BY**" est "**flag-to-be-set**". La dernière transformée doit avoir un résultat qui peut être codé par la classe du champ cité en référence par "**USING**".

22.7.2.4 Il y a erreur de spécification ou d'application ECN si une éventuelle transformée dans le champ "**ENCODER-TRANSFORMS**" n'est pas réversible pour la valeur abstraite à laquelle elle est appliquée.

22.7.2.5 "**DECODER-TRANSFORMS**" doit être présent seulement si "**DETERMINED BY**" est activé à "**field-to-be-used**" ou "**flag-to-be-used**". La première transformée doit avoir une source qui est la même que la catégorie du champ cité en référence par "**USING**". La dernière transformée doit avoir un résultat qui est un entier si "**DETERMINED BY**" est "**field-to-be-used**" et qui est booléenne si "**DETERMINED BY**" est "**flag-to-be-used**".

22.7.2.6 La propriété de codage "**USING**", si présente pour une valeur "**field-to-be-set**" ou "**field-to-be-used**", doit être une référence à un champ qui est présent dans le codage antérieur au champ en cours de codage. Il y a erreur d'application ou de spécification ECN si, dans une instance de codage, la répétition en cours de codage est présente mais que le champ cité en référence par la propriété de codage "**USING**" soit absent (grâce à l'application d'offres d'options).

22.7.2.7 La propriété de codage "**USING**", si présente pour une valeur "**flag-to-be-set**" ou "**flag-to-be-used**" doit être une référence à un champ qui est présent dans l'élément répété d'une répétition. Il y a erreur d'application ou de spécification ECN si, dans une instance de codage, le champ cité en référence par la propriété de codage "**USING**" est absent (grâce à l'application d'offres d'options) à partir de tout élément répété.

NOTE – L'exigence que le champ cité en référence soit présent dans un élément de la répétition est satisfaite si c'est un identificateur qui est visible conformément au § 17.5 (structure de codage), au § 19.3 (mappage par champs appariés), et au § 19.6 (mappage par distribution de valeurs), ou si ce champ est textuellement présent dans la définition d'une structure de remplacement lorsque "**REPLACE COMPONENT**" est utilisé par un objet de codage d'une classe de la catégorie des répétitions.

22.7.2.8 Si "**DETERMINED BY**" est "**container**", la référence "**USING**" doit renvoyer à une concaténation ou à une répétition (ou à une chaîne de bits ou à une chaîne d'octets avec un type contenu) dans laquelle la répétition en cours de codage est un composant (ou un composant d'un composant, jusqu'à une profondeur quelconque). Il y a erreur d'application ou de spécification ECN si, dans une instance de codage, des éléments ultérieurs à l'intérieur de la même concaténation ou répétition doivent être codés.

22.7.2.9 "**HANDLE**" doit être spécifié seulement si "**SIZE**" est "**variable-with-determinant**" et "**DETERMINED BY**" est "**handle**".

22.7.2.10 Si "**HANDLE**" est spécifié, dans ce cas l'élément répété, ainsi que tout élément qui (par l'utilisation d'offres d'options) peut suivre l'élément répété, doivent tous être codés par des objets de codage dont la spécification fait apparaître un pointeur d'identification ayant le même nom que "**HANDLE**". La valeur du pointeur d'identification dans l'élément de répétition doit être différente de celle d'un éventuel élément suivant.

NOTE – Il est prescrit qu'un pointeur d'identification ait la même valeur pour toutes les valeurs abstraites codées par un objet de codage contenant ce pointeur d'identification (voir § 22.9.2.2).

22.7.2.11 "**PATTERN**" doit être spécifié seulement si "**SIZE**" est "**variable-with-determinant**" et "**DETERMINED BY**" est "**pattern**".

22.7.2.12 "**PATTERN**" ne doit pas être la sous-chaîne initiale du codage d'une quelconque valeur de l'élément répété.

NOTE – Il n'y a pas d'interdiction relative à l'instance de "**PATTERN**" à l'intérieur d'un codage de l'élément répété autre qu'à son début.

22.7.2.13 Cette spécification est considérée comme fixée si le mot clé "**REPETITION-SPACE**" est utilisé et il est obligatoire qu'il soit activé à tous endroits dans la syntaxe définie où cela est autorisé. La fixation par défaut de toutes les autres parties de cette syntaxe définie (par exemple, l'utilisation de "**REPETITION-SPACE**" seul) ne satisferait pas les contraintes ci-dessus.

22.7.3 Actions du codeur

22.7.3.1 Les codeurs ne doivent pas produire de codages si les conditions du § 22.7.2 ne sont pas satisfaites.

22.7.3.2 Si "**SIZE**" est une valeur positive, dans ce cas l'espace de codage est le multiple de "**MULTIPLE OF**" unités. Si "**MULTIPLE OF**" concerne des répétitions, dans ce cas le codeur doit cesser le codage si la valeur abstraite en cours de codage n'est pas "**SIZE**" répétitions, diagnostiquant une erreur de spécification ou d'application.

22.7.3.3 Si "**SIZE**" n'est pas mis à une valeur positive, dans ce cas le codeur doit déterminer la taille "s" de l'espace de répétition en "**MULTIPLE OF**" unités à partir de la spécification de codage de valeur. Cette détermination est spécifiée dans les paragraphes relatifs à la spécification de codage de valeur.

22.7.3.4 Si "**SIZE**" est "**encoder-option-with-determinant**", alors le codeur (à titre d'option de codeur) peut augmenter la taille "s" (telle que déterminée au § 22.7.3.3) en "**MULTIPLE OF**" unités à partir de la valeur déterminée à partir de la spécification de codage de valeur, jusqu'à toute valeur qui peut être codée dans le déterminant associé.

22.7.3.5 Si "**SIZE**" est "**fixed-to-max**" ou à "**self-delimiting-values**", dans ce cas il n'y a pas d'autre action du codeur.

22.7.3.6 Si "**SIZE**" est **variable-with-determinant** et "**DETERMINED BY**" est "**container**", dans ce cas il n'y a pas d'autre action du codeur.

22.7.3.7 Si "**DETERMINED BY**" est "**field-to-be-set**", dans ce cas le codeur doit appliquer les transformées spécifiées par "**ENCODER-TRANSFORMS**" (si présent) à la valeur "s" afin de produire une valeur qui doit être codée dans la référence "**USING**".

NOTE – Le codage de la référence "**USING**" dans ce cas apparaît plus tôt dans le codage que le codage de la répétition, et un codeur aura besoin de suspendre le codage de ce champ jusqu'à ce que la répétition en cours de codage ait été déterminée.

22.7.3.8 Si "**DETERMINED BY**" est "**field-to-be-used**", alors le codeur doit vérifier que la valeur contenue dans la référence "**USING**" après transformation par le champ "**DECODER-TRANSFORMS**" (si présent) est égale à "s". Il y a erreur d'application si cette condition n'est pas satisfaite, et le codage ne doit pas continuer.

22.7.3.9 Si "**DETERMINED BY**" est "**flag-to-be-set**", dans ce cas le codeur doit appliquer (pour chaque élément répété) les transformées spécifiées par "**ENCODER-TRANSFORMS**" (si présent) à une valeur booléenne qui est vraie pour tous les éléments sauf le dernier et qui est fausse pour le dernier élément. Le résultat du champ "**ENCODER-TRANSFORMS**" doit être codé dans la référence "**USING**".

22.7.3.10 Si "**DETERMINED BY**" est "**flag-to-be-used**", alors le codeur doit vérifier (pour chaque élément répété) que la valeur contenue dans la référence "**USING**" après transformation par le champ "**DECODER-TRANSFORMS**" (si présent) est une valeur booléenne qui est vraie pour tous les éléments sauf le dernier et qui est fausse pour le dernier élément. Il y a erreur d'application si cette condition n'est pas satisfaite et le codage ne doit pas continuer.

22.7.3.11 Si "**DETERMINED BY**" est "**handle**", il n'y a pas d'autre action nécessaire par le codeur.

22.7.3.12 Si "**DETERMINED BY**" est "**pattern**", dans ce cas le codeur doit vérifier que la séquence spécifiée n'est pas une sous-chaîne initiale d'un quelconque des codages de l'élément répété et doit cesser le codage si cette vérification échoue, diagnostiquant une erreur de spécification ou d'application. Le codeur doit ajouter la séquence "**PATTERN**" à l'extrémité du codage de la répétition.

22.7.4 Actions du décodeur

22.7.4.1 Si "**SIZE**" est une valeur positive, dans ce cas le décodeur détermine l'espace de codage en tant que multiple de "**MULTIPLE OF**" unités. Si "**MULTIPLE OF**" concerne des répétitions, dans ce cas la fin réelle de l'espace de répétition est déterminée par le décodage et par le comptage des répétitions.

22.7.4.2 Si "**SIZE**" n'est pas mis à une valeur positive, dans ce cas le codeur doit déterminer la taille "s" de l'espace de répétition en "**MULTIPLE OF**" unités à partir de la spécification de codage de valeur. Cette détermination est spécifiée dans les paragraphes relatifs à la spécification de codage de valeur.

22.7.4.3 Si "**SIZE**" est "**variable-with-determinant**" et si "**DETERMINED BY**" est activé à "**container**", dans ce cas le décodeur doit utiliser l'extrémité du conteneur spécifiée par "**USING**" en tant qu'extrémité de l'espace de codage.

22.7.4.4 Si "**SIZE**" est "**variable-with-determinant**" et si "**DETERMINED BY**" est activé à (ou prend par défaut) "**field-to-be-set**", dans ce cas le décodeur doit reconstituer la valeur "s" par l'application de l'inversion du champ "**ENCODER-TRANSFORMS**" (si présent) à la valeur de la référence "**USING**".

22.7.4.5 Si "**DETERMINED BY**" est "**field-to-be-used**", alors le décodeur doit reconstituer la valeur "s" par l'application du champ "**DECODER-TRANSFORMS**" (si présent) à la valeur de la référence "**USING**".

22.7.4.6 Si "**DETERMINED BY**" est "**flag-to-be-set**", dans ce cas le décodeur doit reconstituer une valeur booléenne par l'application de l'inversion du champ "**ENCODER-TRANSFORMS**" (si présent) à la valeur de la référence "**USING**". Cet élément est le dernier de la répétition si et seulement si la valeur booléenne est fausse.

22.7.4.7 Si "**DETERMINED BY**" est "**flag-to-be-used**", alors le décodeur doit reconstituer une valeur booléenne par l'application du champ "**DECODER-TRANSFORMS**" (si présent) à la valeur de la référence "**USING**". Cet élément est le dernier de la répétition si et seulement si la valeur booléenne est fausse.

22.7.4.8 Si "**DETERMINED BY**" est "**handle**", dans ce cas le décodeur doit déterminer la valeur du pointeur d'identification et tenter de décoder l'élément suivant (en parallèle), soit en tant que nouvelle répétition ou en tant qu'élément suivant, au moyen de la valeur du pointeur d'identification afin de distinguer ces options. Si le décodage réussit pour plusieurs de ces éléments ou pour aucun d'entre eux, il y a erreur de codage ou de spécification.

22.7.4.9 Si "**DETERMINED BY**" est "**pattern**", alors le décodeur doit, au début du décodage de chaque répétition, vérifier si "**PATTERN**" est présent. Si "**PATTERN**" est présent, les bits de séquence doivent être rejetés et la répétition doit être terminée.

22.8 Bourrage et justification de valeur

22.8.1 Propriétés, syntaxe et finalité du codage

22.8.1.1 Le bourrage et la justification de valeur utilisent les propriétés de codage suivantes:

```
&value-justification              Justification DEFAULT right:0,
&value-pre-padding                Padding DEFAULT zero,
&value-pre-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
&value-post-padding               Padding DEFAULT zero,
&value-post-pattern               Non-Null-Pattern DEFAULT bits:'0'B,
&unused-bits-determination        UnusedBitsDetermination
                                  DEFAULT field-to-be-set,
&unused-bits-reference            REFERENCE OPTIONAL,
&Unused-bits-encoder-transforms   #TRANSFORM ORDERED OPTIONAL,
&Unused-bits-decoder-transforms   #TRANSFORM ORDERED OPTIONAL
```

22.8.1.2 La syntaxe à utiliser pour le bourrage et la justification de valeur doit être la suivante:

```
[VALUE-PADDING
      [JUSTIFIED &value-justification]
      [PRE-PADDING &value-pre-padding
            [PATTERN &value-pre-pattern]]
      [POST-PADDING &value-post-padding
            [PATTERN &value-post-pattern]]
      [UNUSED BITS
            [DETERMINED BY &unused-bits-determination]
            [USING &unused-bits-reference
                  [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                  [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]
```

22.8.1.3 La définition des types utilisés en justification est la suivante:

```
Justification ::= CHOICE
              { left              INTEGER (0..MAX),
                right             INTEGER (0..MAX)} -- (voir § 21.8)

Padding ::= ENUMERATED {zero, one, pattern, encoder-option} -- (voir § 21.9)

Pattern ::= CHOICE
      {bits                       BIT STRING,
       octets                     OCTET STRING,
       char8                      IA5String,
       char16                     BMPString,
       char32                     UniversalString,
       any-of-length              INTEGER (1..MAX),
       different                  ENUMERATED {any} }

Non-Null-Pattern ::= Pattern
      (ALL EXCEPT (bits:''B | octets:''H | char8:"" | char16:"" |
                   char32:"")) -- (voir § 21.10)

UnusedBitsDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, not-needed} -- (voir § 21.4)
```

22.8.1.4 Cette spécification vise à déterminer la façon dont un codeur place un codage de valeur dans un espace de codage et permet à un décodeur de déterminer la position du codage de valeur.

22.8.1.5 Le nombre précis de bits à ajouter par un codeur dépend aussi bien de la spécification de l'espace de codage que de la spécification de codage de valeur. Il est spécifié pour chaque instance de codage de valeur.

22.8.1.6 "**USING**" est une référence qui permet à un décodeur de déterminer le nombre des bits de bourrage insérés. C'est une référence à un champ auxiliaire ou à un champ acheminant des valeurs abstraites, selon "**DETERMINED BY**".

22.8.2 Restrictions de spécification

22.8.2.1 Le nombre de bits spécifié dans la justification doit être inférieur ou égal au nombre total des bits de bourrage "b" (voir ci-dessous).

22.8.2.2 "**USING**" doit être spécifié si et seulement si "**DETERMINED BY**" n'est pas "**not-needed**".

22.8.2.3 "**ENCODER-TRANSFORMS**" doit être présent seulement si "**DETERMINED BY**" est activé à (ou prend par défaut) "**field-to-be-set**". La première transformée doit avoir une source qui est un entier et la dernière transformée doit avoir un résultat qui peut être codé par la classe du champ cité en référence par "**USING**".

22.8.2.4 Il y a erreur de spécification ou d'application ECN si une éventuelle transformée dans le champ "**ENCODER-TRANSFORMS**" n'est pas réversible pour la valeur abstraite à laquelle elle est appliquée.

22.8.2.5 "**DECODER-TRANSFORMS**" doit être présent seulement si "**DETERMINED BY**" est activé à "**field-to-be-used**". La première transformée doit avoir une source qui est la même que la catégorie du champ cité en référence par "**USING**", lequel ne doit pas être un champ auxiliaire. La dernière transformée doit avoir un résultat qui est un entier.

22.8.2.6 La propriété de codage "**USING**", si présente, doit être une référence à un champ qui est présent dans le codage antérieur au champ en cours de codage. Il y a erreur d'application ou de spécification ECN si, dans une instance de codage, le champ en cours de codage est présent mais que le champ cité en référence par la propriété de codage "**USING**" soit absent (par l'application d'offres d'options).

22.8.2.7 Cette spécification est considérée comme fixée si le mot clé "**VALUE-PADDING**" est utilisé. Si elle n'est pas fixée, les actions sont spécifiées dans tous les emplacements où cette syntaxe est permise.

22.8.3 Actions du codeur

22.8.3.1 Les codeurs ne doivent pas produire de codages si les conditions du § 22.8.2 ne sont pas satisfaites.

22.8.3.2 Cette spécification est appliquée si et seulement si l'espace de codage ou la spécification de codage de l'espace de répétition, ainsi que la spécification de codage de valeur, déterminent que des bits de bourrage peuvent être ajoutés autour de la valeur ou du codage de répétition à l'intérieur de l'espace de codage ou de répétition. Soit "**b**" (où "**b**" est supérieur ou égal à 0) le nombre déterminé de bits de bourrage ajoutés dans une instance de codeur.

22.8.3.3 Si "**JUSTIFIED**" est "**right:n**", dans ce cas "b"-"n" bits doivent être ajoutés en tant que prébourrage avant la valeur ou le codage de répétition et "n" bits doivent être ajoutés ensuite en tant que post-bourrage.

22.8.3.4 Si "**JUSTIFIED**" est "**left:n**", dans ce cas "n" bits doivent être ajoutés en tant que prébourrage avant la valeur ou le codage de répétition et "b"-"n" bits doivent être ajoutés ensuite en tant que post-bourrage.

22.8.3.5 Les bits de bourrage doivent être fixés conformément aux spécifications "**PRE-PADDING**" et "**POST-PADDING**" avec le bit initial de la séquence en tant que premier bit inséré dans chaque cas.

22.8.3.6 Si "**DETERMINED BY**" est "**not-needed**", alors ce champ complète les actions des codeurs.

22.8.3.7 Si "**DETERMINED BY**" est "**field-to-be-set**", dans ce cas le codeur doit appliquer les transformées spécifiées par "**ENCODER-TRANSFORMS**" (si présent) à la valeur "b" afin de produire une valeur qui doit être codée dans la référence "**USING**".

NOTE – Le codage de la référence "**USING**" dans ce cas apparaît plus tôt dans le codage que le codage de ce champ et un codeur aura besoin de suspendre le codage de ce champ jusqu'à ce que la valeur à coder ait été déterminée par le codage de ce champ.

22.8.3.8 Si "**DETERMINED BY**" est "**field-to-be-used**", alors le codeur doit vérifier que la valeur contenue dans la référence "**USING**" après transformation par "**DECODER-TRANSFORMS**" (si présent) est égale à "b". Il y a erreur d'application si cette condition n'est pas satisfaite et le codage ne doit pas continuer.

22.8.4 Actions du décodeur

22.8.4.1 Si "**DETERMINED BY**" est "**not-needed**", dans ce cas le décodeur doit déterminer la valeur de "b" telle qu'indiquée par la spécification de codage de valeur et d'espace de codage ou par la détermination de répétition.

22.8.4.2 Si "**DETERMINED BY**" est activé à (ou prend par défaut) "**field-to-be-set**", dans ce cas le décodeur doit reconstituer la valeur "b" par l'application de l'inversion de "**ENCODER-TRANSFORMS**" (si présent) à la valeur de la référence "**USING**".

22.8.4.3 Si "**DETERMINED BY**" est "**field-to-be-used**", alors le décodeur doit reconstituer la valeur "b" par l'application du champ "**DECODER-TRANSFORMS**" (si présent) à la valeur de ce champ.

22.8.4.4 Le décodeur doit utiliser le composant "**JUSTIFIED**" et la valeur de "b" afin de déterminer la position du codage de valeur à l'intérieur de l'espace de codage et doit ignorer la valeur de tous les bits de bourrage.

22.9 Spécification de pointeur d'identification

22.9.1 Propriétés, syntaxe et finalité du codage

22.9.1.1 La spécification de pointeur d'identification utilise les propriétés de codage suivantes:

```
&exhibited-handle                PrintableString OPTIONAL,
&Handle-positions                INTEGER (0..MAX) OPTIONAL,
&handle-value                    HandleValue DEFAULT tag:any
```

22.9.1.2 La syntaxe à utiliser pour spécification de pointeur d'identification doit être la suivante:

```
[EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
      [AS &handle-value]]
```

22.9.1.3 La définition du type utilisé en spécification de pointeur d'identification est la suivante:

```
HandleValue ::= CHOICE {
      bits        BIT  STRING,
      octets      OCTET STRING,
      number      INTEGER (0..MAX),
      tag         ENUMERATED {any}}  -- (voir § 21.15)
```

22.9.1.4 Cette spécification est utilisée afin d'indiquer qu'un objet de codage fait apparaître un pointeur d'identification à l'intérieur de tous ses codages (c'est-à-dire pour toutes les valeurs abstraites possibles qu'il code). Le nom du pointeur d'identification est spécifié, ainsi que les bits qui sont associés à ce pointeur d'identification. La valeur du pointeur d'identification est spécifiée par "**HandleValue**".

22.9.1.5 La liste des positions contenues dans la production "**AT**" doit viser les positions des bits formant le pointeur d'identification dans le codage final, après qu'un éventuel préalignement a été appliqué et après que d'éventuelles actions d'inversion par le codeur de l'ordre des bits sont intervenues, sauf les inversions de l'ordre des bits qui résultent de la spécification d'un objet de codage dans la classe **#OUTER**.

NOTE – Cela implique qu'un décodeur ait besoin d'effectuer d'éventuelles inversions de l'ordre des bits spécifié dans **#OUTER** pour l'unité PDU entière, mais sinon qu'il examine les positions des bits et leur valeur sans aucune considération des éventuelles inversions de l'ordre des bits qui peuvent être spécifiées pour des objets de codage particuliers.

22.9.1.6 La liste des positions contenues dans la production "**AT**" est un ensemble de valeurs entières (non nécessairement contiguës, et non nécessairement en ordre croissant dans la spécification ECN). Ces positions doivent être rangées en ordre croissant par les codeurs et les décodeurs à partir de la position zéro (le premier bit dans la partie du codage qui contient le pointeur) et les bits contenus dans ces positions forment un champ de pointeur théorique.

22.9.1.7 Pour une valeur "**number**" de "**HandleValue**" ou pour le codage d'un numéro d'étiquette, le bit contenu dans le champ de pointeur théorique le plus proche de la position zéro est le bit de poids fort et la valeur "**number**" ou le numéro d'étiquette qui spécifie la valeur "**HandleValue**" est justifié à droite à l'intérieur de ce champ. Si la valeur "**number**" ou le numéro d'étiquette est trop grand pour le champ, il y a erreur de spécification ECN.

22.9.1.8 Si les options "**bitstring**" ou "**octetstring**" de "**HandleValue**" sont utilisées, dans ce cas leur valeur doit avoir le nombre de bits qui est spécifié pour le pointeur d'identification par "**AT**". Le bit contenu dans le champ de pointeur théorique le plus proche de la position zéro est le bit initial de l'option "**bitstring**" ou "**octetstring**" qui spécifie la valeur "**HandleValue**".

22.9.1.9 La valeur "**HandleValue**" ne doit pas être spécifiée en tant que "**tag:any**" à moins que la spécification ne vise un objet de codage de la classe **#TAG**. Dans ce cas la valeur du pointeur d'identification est déterminée soit par le numéro d'étiquette contenu dans la spécification ECN ou par le numéro d'étiquette mappé à partir d'une étiquette ASN.1 (comme spécifié dans l'article 19) et n'a pas besoin d'être spécifiée au moyen de "**HandleValue**". Si, cependant, la valeur est spécifiée par "**HandleValue**" et diffère de celle qui a été attribuée dans une spécification ECN d'une classe d'étiquettes ou dans une étiquette ASN.1 mappée sur une étiquette ECN, il s'agit d'une erreur de spécification ECN.

22.9.2 Contraintes de spécification

22.9.2.1 Dans une application quelconque de la notation ECN, tous les pointeurs d'identification ayant le même nom doivent spécifier le même ensemble de bits pour l'emplacement du pointeur d'identification.

NOTE – Il n'y a pas d'exigence générale que la valeur du pointeur d'identification (révélée par différents objets de codage) soit distincte, mais des valeurs distinctes sont requises lorsque le pointeur d'identification est utilisé afin de résoudre des offres d'options, un choix d'options, ou une terminaison de répétition (voir § 21.5.7, 21.6.6 et 21.7.10).

22.9.2.2 Le spécificateur ECN doit veiller à ce qu'un objet de codage contenant un pointeur d'identification produise la même valeur du pointeur d'identification pour chaque valeur abstraite qui est codée.

22.9.2.3 Tous les objets de codage qui font apparaître le même pointeur d'identification doivent soit n'avoir aucune spécification de préalignement, ou s'aligner sur la même unité de préalignement.

NOTE – Cette restriction est imposée de façon que les décodeurs puissent passer à la position d'alignement avant de rechercher le pointeur lorsque le décodage dépend de la valeur du pointeur.

22.9.2.4 Si un objet de codage pour une classe de la catégorie des répétitions fait apparaître un pointeur d'identification, dans ce cas ce pointeur d'identification doit également être affiché (avec la même valeur) par le codage de l'élément répété.

22.9.2.5 Si un objet de codage pour une classe de la catégorie des options fait apparaître un pointeur d'identification, dans ce cas ce pointeur d'identification doit également être affiché par (le codage de) toutes les options, et la valeur du pointeur d'identification doit être la même pour toutes les options.

NOTE – Dans ce cas, ce pointeur d'identification ne peut pas être utilisé pour la détermination des options dans cette option. La détermination des options doit donc être effectuée au moyen d'un autre pointeur d'identification ou par d'autres moyens.

22.9.2.6 Si un objet de codage pour une classe de la catégorie des concaténations fait apparaître un pointeur d'identification, dans ce cas le premier composant codé (si présent) (ou, s'il est étiqueté, l'étiquette), compte tenu des offres d'options, doit offrir ce pointeur d'identification avec la même valeur.

22.9.2.7 Cette spécification est considérée comme fixée si le mot clé "**EXHIBITS-HANDLE**" est utilisé. S'il n'est pas activé, alors il n'y a pas de pointeur d'identification en évidence.

22.9.3 Actions des codeurs

22.9.3.1 Si un objet de codage fait apparaître un pointeur d'identification, le codeur doit vérifier que le codage a la valeur du pointeur d'identification et doit diagnostiquer une erreur de spécification ou d'application si ce n'est pas le cas.

22.9.4 Actions des décodeurs

22.9.4.1 Il n'y a pas d'actions des décodeurs résultant directement de la mise en évidence d'un pointeur d'identification. Les actions du décodeur ne résultent que de l'utilisation du pointeur d'identification afin de déterminer une offre d'options, une fin de répétitions, ou un choix d'options.

22.10 Spécification de concaténation

22.10.1 Propriétés, syntaxe et finalité du codage

22.10.1.1 La spécification de concaténation utilise les propriétés de codage suivantes:

```
&concatenation-order            ENUMERATED {textual, tag, random}
                                DEFAULT textual,
&concatenation-alignment        ENUMERATED {none, aligned}
                                DEFAULT aligned,
&concatenation-handle           PrintableString
                                DEFAULT "default-handle"
```

22.10.1.2 La syntaxe à utiliser pour la spécification de concaténation doit être la suivante:

```
[CONCATENATION
      [ORDER &concatenation-order]
      [ALIGNMENT &concatenation-alignment]
      [HANDLE &concatenation-handle]]
```

22.10.1.3 Cette spécification détermine l'ordre dans lequel les composants d'une classe de codage dans la catégorie des concaténations sont codés, les moyens qu'un codeur utilise afin de désigner chaque composant et un éventuel bourrage de préalignement qui doit être assuré entre composants.

22.10.2 Contraintes de spécification

22.10.2.1 Si "**ORDER**" est "**random**", dans ce cas "**HANDLE**" prend la valeur par défaut "**default-handle**" s'il n'est pas activé et le codage de tous les composants doit offrir "**HANDLE**" avec des valeurs distinctes pour le pointeur d'identification.

22.10.2.2 Si "**ALIGNMENT**" est "**aligned**", dans ce cas la spécification de préalignement prend la valeur par défaut à moins qu'elle ne soit fixée.

22.10.2.3 Si un composant possède son propre préalignement explicite, celui-ci est appliqué après éventuel préalignement du composant résultant du réglage de "**ALIGNMENT**" dans la classe de codage de la catégorie des concaténations.

NOTE – La fonction équivalente n'est pas assurée pour les répétitions, car elle peut être réalisée plus simplement par préalignement du composant isolé.

22.10.2.4 Si "**ORDER**" est "**tag**", dans ce cas chaque composant doit commencer par une classe de codage dans la catégorie des étiquettes. Le numéro d'étiquette associé à cette classe est appelé *étiquette de composant*.

22.10.2.5 L'étiquette de composant de chaque option doit être distincte.

22.10.2.6 Cette spécification est considérée comme fixée si le mot clé "**CONCATENATION**" est utilisé. Si elle n'est pas fixée, alors les codeurs et les décodeurs agissent comme si elle était fixée avec chaque propriété de codage prenant sa valeur par défaut.

22.10.2.7 Si (grâce à l'application d'offres d'options) il y a au moins une valeur abstraite d'une concaténation qui ne possède aucun bit dans son codage, dans ce cas la concaténation ne doit avoir aucun préalignement.

NOTE – Ce paragraphe s'appliquera si une concaténation n'a pas de composants obligatoires, ou si tous ses composants obligatoires peuvent n'avoir (grâce à l'application d'offres d'options) aucun bit dans leurs codages.

22.10.3 Actions du codeur

22.10.3.1 Si "**ORDER**" est "**textual**", l'ordre textuel contenu dans la spécification de type ASN.1 ou dans la définition de structure ECN doit être utilisé.

22.10.3.2 Si "**ORDER**" est "**tag**", dans ce cas l'ordre doit être celui des numéros d'étiquette contenus dans les étiquettes de composant (le plus faible numéro d'étiquette d'abord).

22.10.3.3 Si "**ORDER**" est "**random**", dans ce cas le codeur doit déterminer l'ordre de concaténation sans contrainte.

22.10.3.4 Si "**ALIGNMENT**" est "**none**", le codeur doit juxtaposer les codages de composants sans bits insérés.

22.10.3.5 Si "**ALIGNMENT**" est "**aligned**", dans ce cas le codeur doit appliquer la spécification de préalignement contenue dans la classe de la catégorie des concaténations avant codage de chaque composant, sauf qu'une spécification de préalignement de valeur "**ALIGNED TO ANY**" doit être interprétée comme une spécification de "**ALIGNED TO NEXT**" (voir § 22.2).

NOTE 1 – La raison de ce qui précède est qu'il ne peut y avoir un seul pointeur de début pour "**ALIGNED TO ANY**".

NOTE 2 – Tout éventuel préalignement spécifié pour un composant (dont "**ALIGNED TO ANY**") est appliqué après les actions ci-dessus.

22.10.4 Actions du décodeur

22.10.4.1 Lors du décodage d'un composant, un décodeur doit d'abord exécuter les actions du décodeur associées à la spécification de préalignement pour "**ALIGNMENT**" si cette spécification est activée à "**aligned**", le champ "**ALIGNED TO ANY**" étant traité comme étant "**ALIGNED TO NEXT**" (voir § 22.2). Si "**ALIGNMENT**" est activé à "**none**", dans ce cas le décodeur doit procéder directement au décodage du composant.

22.10.4.2 Le décodeur doit déterminer l'ordre des composants à partir de l'ordre défini pour le codeur si "**ORDER**" est "**textual**" ou "**tag**".

22.10.4.3 Si "**ORDER**" est "**random**", le décodeur doit déterminer l'ordre des composants par examen de la valeur des bits associés à "**HANDLE**".

22.10.4.4 Chaque composant a une valeur distincte pour les bits associés à "**HANDLE**", qui permet que le composant soit identifié. Le décodage doit continuer jusqu'à ce qu'une valeur abstraite ait été obtenue pour chaque composant et un décodeur doit diagnostiquer une erreur de codeur si plusieurs codages sont identifiés pour un composant, ou si des valeurs inattendues apparaissent pour des pointeurs d'identification pendant le décodage.

NOTE – Des valeurs inattendues peuvent apparaître dans le cadre d'une disposition d'extensibilité, mais cela n'est pas pris en charge dans cette version de la présente Recommandation | Norme internationale. De telles instances doivent être traitées comme des erreurs de codeur.

22.11 Spécification de codage du type confiné

22.11.1 Propriétés, syntaxe et finalité du codage

22.11.1.1 La spécification de codage du type confiné utilise les propriétés de codage suivantes:

```
&Primary-encoding-object-set      #ENCODINGS OPTIONAL,
&Secondary-encoding-object-set    #ENCODINGS OPTIONAL,
&over-ride-encoded-by             BOOLEAN DEFAULT FALSE
```

22.11.1.2 La syntaxe à utiliser pour une spécification de codage du type confiné doit être la suivante:

```
[CONTENTS-ENCODING &Primary-encoding-object-set
                  [COMPLETED BY &Secondary-encoding-object-set]
                  [OVERRIDE &over-ride-encoded-by]]
```

22.11.1.3 L'objet de cette spécification est de déterminer le codage d'un type contenu, et de déterminer si une contrainte de contenu ASN.1 **ENCODED BY**, associée à ce type contenu, doit être outrepassée.

22.11.1.4 La présente spécification offre un ou deux ensembles d'objets de codage. Si deux ensembles sont offerts, ils sont combinés conformément au § 13.2 afin de produire un ensemble d'objets de codage combinés.

22.11.1.5 Cette spécification est considérée comme fixée si le mot clé "**CONTENTS-ENCODING**" est utilisé.

22.11.2 Actions du codeur

22.11.2.1 Si "**CONTENTS-ENCODING**" n'est pas activé, dans ce cas un type de contenu doit être codé au moyen de l'ensemble d'objets de codage combinés appliqué au conteneur si "**ENCODED BY**" n'est pas présent dans la contrainte de contenu ASN.1; sinon avec les règles de codage spécifiées par l'instruction "**ENCODED BY**".

22.11.2.2 Si "**CONTENTS-ENCODING**" est activé, l'ensemble d'objets de codage combinés formé à partir de "**COMPLETED BY**" doit être appliqué au type contenu si "**ENCODED BY**" n'est pas présent dans la contrainte de contenu ASN.1, ou si "**ENCODED BY**" est présent et "**OVERRIDE**" est **TRUE**. Sinon l'ensemble d'objets de codage combinés appliqué au type conteneur doit être appliqué au type contenu.

22.11.3 Actions du décodeur

22.11.3.1 Un décodeur doit décoder le type contenu conformément au codage appliqué par le codeur, comme spécifié ci-dessus.

22.12 Spécification de l'inversion de l'ordre des bits

22.12.1 Propriétés, syntaxe et finalité du codage

22.12.1.1 La spécification de l'inversion de l'ordre des bits utilise la propriété de codage suivante:

```
&bit-reversal                       ReversalSpecification
                                    DEFAULT no-reversal
```

22.12.1.2 La syntaxe à utiliser pour spécification de l'inversion de l'ordre des bits doit être la suivante:

```
[BIT-REVERSAL &bit-reversal]
```

22.12.1.3 La définition des types utilisés dans ce groupe est la suivante:

```
ReversalSpecification ::= ENUMERATED
      {no-reversal,
      reverse-bits-in-units,
      reverse-half-units,
      reverse-bits-in-half-units} -- (voir § 21.13)
```

22.12.1.4 L'objet de cette spécification vise à permettre que l'ordre de bits dans le codage final soit différent des bits produits dans le cadre d'un espace de codage ou d'un espace de répétition, ou dans le codage complet d'une unité PDU (voir article 25).

NOTE 1 – L'inversion de l'ordre des bits peut être spécifiée pour des codages de champ binaire individuels et également pour les résultats de concaténation ou de répétition. Il convient de veiller à ce qu'une inversion ne contredise pas l'autre inversion.

NOTE 2 – L'inversion de l'ordre des bits s'applique au contenu d'un espace de codage ou de répétition (y compris tout prébourrage ou post-bourrage de valeur), mais ne s'applique pas à un éventuel bourrage de préalignement.

22.12.2 Contraintes de spécification

22.12.2.1 Cette spécification n'est disponible que lorsqu'un codage d'espace de codage ou d'espace de répétition est requis et à l'intérieur de la classe **#OUTER**.

22.12.2.2 "**BIT-REVERSAL**" ne doit pas être "**reverse-half-units**" ou "**reverse-bits-in-half-units**" à moins que "**MULTIPLE OF**" ne soit activé à un même nombre de bits pour l'inversion de l'espace de codage ou de répétition ou dans la classe **#OUTER**. (Cette exigence implique qu'une valeur de "repetition" pour "**MULTIPLE OF**" n'est pas permise dans ce cas.)

22.12.2.3 "**BIT-REVERSAL**" ne doit pas être activé à moins que la valeur de "**MULTIPLE-OF**" ne soit "**repetitions**" ou ne soit supérieure à un bit.

22.12.2.4 Cette spécification est considérée comme fixée si le mot clé "**BIT-REVERSAL**" est utilisé. Si elle n'est pas fixée, alors les codeurs et les décodeurs agissent comme si elle était fixée avec la propriété de codage prenant sa valeur par défaut.

22.12.3 Actions du codeur

22.12.3.1 Sauf lors de l'exécution d'actions de la classe **#OUTER**, un codeur doit subdiviser le contenu de l'espace de codage ou de répétitions en "**MULTIPLE OF**" unités à moins que "**MULTIPLE OF**" n'ait la valeur "**repetitions**", auquel cas tout l'espace de codage doit être traité en tant qu'unité isolée. Lors de l'exécution de l'inversion de l'ordre des bits pour **#OUTER**, le codage entier (après qu'un éventuel bourrage ait été appliqué) doit être subdivisé en "**MULTIPLE OF**" unités. Il y a erreur de spécification ECN si le codage entier n'est pas un multiple entier de "**MULTIPLE OF**" unités.

22.12.3.2 Le codeur ne doit pas effectuer d'inversion (valeur par défaut), ou doit inverser les bits dans chaque unité, ou doit inverser les demi-unités (sans changer l'ordre de bits dans chaque demi-unité) ou doit inverser les bits à l'intérieur de chaque demi-unité, comme spécifié par la valeur de "**BIT-REVERSAL**".

22.12.4 Actions du décodeur

22.12.4.1 Le décodeur doit d'abord déterminer (voir la spécification de l'espace de codage et de l'espace de répétition) l'extrémité de l'espace de codage ou de répétition ou (pour la spécification de l'inversion de l'ordre des bits à l'intérieur de la classe **#OUTER**) l'extrémité du codage entier et doit alors exécuter les actions d'inversion spécifiées pour le codeur avant de poursuivre le décodage.

NOTE – L'exécution des mêmes inversions rétablira l'ordre original des bits.

23 Spécification de syntaxe définie pour classes de champ binaire et de constructeur

Cet article offre la syntaxe complète pour la définition des objets de codage de chaque classe de codage dans les différentes catégories.

NOTE – Les actions du codeur et du décodeur sont spécifiées dans les paragraphes suivants comme étant soumises à la condition qu'un groupe de propriétés de codage soit activé. Un groupe est activé si et seulement si le mot clé initial de ce groupe est présent dans la spécification de l'objet de codage.

23.1 Définition des objets de codage pour les classes de la catégorie des options

23.1.1 La syntaxe définie

La syntaxe de définition des objets de codage pour les classes de la catégorie des options est définie comme suit:

```
#ALTERNATIVES ::= ENCODING-CLASS {

    -- Spécification de remplacement de structure ou de composant (voir § 22.1)
    &#Replacement-structure
         OPTIONAL,
    &replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,
    &#Head-end-structure                                                OPTIONAL,

    -- Spécification de préalignement et de bourrage (voir § 22.2)
    &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding        Padding DEFAULT zero,
    &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                       DEFAULT bits:'0'B,

    -- Spécification du pointeur de début (voir § 22.3)
    &start-pointer                     REFERENCE  OPTIONAL,
    &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

    -- Détermination des options (voir § 22.6)
    &alternative-determination         AlternativeDetermination
                                       DEFAULT field-to-be-set,
    &alternative-reference             REFERENCE OPTIONAL,
    &Encoder-transforms                #TRANSFORM ORDERED OPTIONAL,
    &Decoder-transforms                #TRANSFORM ORDERED OPTIONAL,
```

```
      &handle-id                          PrintableString
                                          DEFAULT "default-handle",
      &alternative-ordering               ENUMERATED {textual, tag}
                                          DEFAULT textual,

      -- Spécification de pointeur d'identification (voir § 22.9)
      &exhibited-handle                   PrintableString OPTIONAL,
      &Handle-positions                   INTEGER (0..MAX) OPTIONAL,
      &handle-value                       HandleValue DEFAULT tag:any

} WITH SYNTAX {
      [REPLACE
            [STRUCTURE]
            [COMPONENT]
            [ALL COMPONENTS]
           WITH &Replacement-structure
            [ENCODED BY &replacement-structure-encoding-object
                  [INSERT AT HEAD &#Head-end-structure]]]
      [ALIGNED TO
            [NEXT]
            [ANY]
                 &encoding-space-pre-alignment-unit
                 [PADDING &encoding-space-pre-padding
                       [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
      ALTERNATIVE
            [DETERMINED BY &alternative-determination
                  [HANDLE &handle-id]]
            [USING &alternative-reference
                  [ORDER &alternative-ordering]
                  [ENCODER-TRANSFORMS &Encoder-transforms]
                  [DECODER-TRANSFORMS &Decoder-transforms]]
      [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]

}
```

23.1.2 Finalité et restrictions

23.1.2.1 Cette syntaxe est utilisée afin de définir le début de l'espace de codage pour une classe de codage de la catégorie des options, la détermination de l'option qui a été codée, et une déclaration facultative que tous les codages offrent un pointeur d'identification spécifié (avec des valeurs de pointeur d'identification distinctes).

23.1.2.2 Si "**REPLACE STRUCTURE**" est activé, dans ce cas aucun autre groupe de propriétés de codage ne doit être activé.

23.1.2.3 Les codages de cette classe n'offrent pas de pointeur d'identification à moins que "**EXHIBITS HANDLE**" ne soit activé (même si tous les composants offrent un pointeur d'identification, qui peut être ou ne pas être le même).

23.1.2.4 Si "**EXHIBITS HANDLE**" est activé, dans ce cas les codages de toutes les options de cette classe doivent offrir le pointeur d'identification défini, et offrir des valeurs distinctes pour ce pointeur d'identification.

NOTE – Cela nécessitera normalement que chaque composant ait un champ "**EXHIBITS HANDLE**" mis à la même valeur, à moins qu'une insertion en tête de séquence n'ait affiché le pointeur d'identification (voir § 9.10.3).

23.1.3 Actions du codeur

23.1.3.1 Pour tout groupe de propriétés de codage qui est activé, le codeur doit exécuter les actions de codage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) remplacement;

b) préalignement et bourrage;

c) pointeur de début;

d) détermination des options;

e) pointeur d'identification.

23.1.4 Actions du décodeur

23.1.4.1 Pour tout groupe de propriétés de codage qui est activé, le décodeur doit exécuter les actions de décodage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) préalignement et bourrage;

b) pointeur de début;

c) détermination des options.

23.2 Définition des objets de codage pour les classes de la catégorie des chaînes de bits

23.2.1 La syntaxe définie

La syntaxe de définition des objets de codage pour les classes de la catégorie des chaînes de bits est définie comme suit:

```
#BITS ::= ENCODING-CLASS {

      -- Spécification de préalignement et de bourrage (voir § 22.2)
      &encoding-space-pre-alignment-unitUnit (ALL EXCEPT repetitions) DEFAULT bit,
      &encoding-space-pre-padding       Padding DEFAULT zero,
      &encoding-space-pre-pattern       Non-Null-Pattern (ALL EXCEPT different:any)
                                        DEFAULT bits:'0'B,

      -- Spécification du pointeur de début (voir § 22.3)
      &start-pointer                    REFERENCE  OPTIONAL,
      &start-pointer-unit               Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL,

      -- codage de valeur des bits
      &value-reversal                   BOOLEAN DEFAULT FALSE,
      &Transforms                       #TRANSFORM ORDERED OPTIONAL,
      &Bits-repetition-encodings        #CONDITIONAL-REPETITION ORDERED OPTIONAL,
      &bits-repetition-encoding         #CONDITIONAL-REPETITION OPTIONAL,

      -- Spécification de pointeur d'identification (voir § 22.9)
      &exhibited-handle                 PrintableString OPTIONAL,
      &Handle-positions                 INTEGER (0..MAX) OPTIONAL,
      &handle-value                     HandleValue DEFAULT tag:any,

      -- Spécification de codage du type confiné (voir § 22.11)
      &Primary-encoding-object-set      #ENCODINGS OPTIONAL,
      &Secondary-encoding-object-set    #ENCODINGS OPTIONAL,
      &over-ride-encoded-by             BOOLEAN DEFAULT FALSE


} WITH SYNTAX {

      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
      [VALUE-REVERSAL             &value-reversal]
      [TRANSFORMS                 &Transforms]
      [REPETITION-ENCODINGS       &Bits-repetition-encodings]
      [REPETITION-ENCODING        &bits-repetition-encoding]
      [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]
      [CONTENTS-ENCODING &Primary-encoding-object-set
                        [COMPLETED BY &Secondary-encoding-object-set]
                        [OVERRIDE &over-ride-encoded-by]]

}
```

23.2.2 Modèle de codage de classes de la catégorie des chaînes de bits

23.2.2.1 Le modèle des codages de bits est le suivant:

a) l'ordre des bits dans la chaîne de bits peut être inversé;

b) les bits sont alors considérés comme une répétition de bits;

c) il y a une transformée facultative (spécifiée par "**TRANSFORMS**") dans laquelle chaque bit est transformé en une chaîne de bits (auto-délimitatrice);

d) soit "**REPETITION-ENCODING**" ou "**REPETITION-ENCODINGS**" spécifie la façon dont la répétition de la séquence de bits (ou des bits d'origine si "**TRANSFORMS**" n'est pas activé) doit être codée.

NOTE – Le seul objectif de permettre "**REPETITION-ENCODING**" ainsi que "**REPETITION-ENCODINGS**" est d'offrir une syntaxe qui ne contient pas de double accolade ("**{{**") dans le cas courant d'un codage conditionnel isolé. L'utilisation de "**REPETITION-ENCODINGS**" lorsqu'il y a un codage conditionnel isolé est déconseillée mais autorisée.

23.2.2.2 Les limites (si présentes) de la classe en cours de codage (une classe de la catégorie des chaînes de bits) sont des limites applicables au nombre de bits dans la chaîne de bits formant chaque valeur abstraite.

23.2.2.3 Lorsqu'elles sont considérées comme une répétition de bit, ces limites doivent être interprétées comme étant applicables au nombre de répétitions, et peuvent être utilisées dans la spécification des objets de codage de classe **#CONDITIONAL-REPETITION** qui sont utilisés dans la spécification de cet objet de codage.

23.2.3 Finalité et restrictions

23.2.3.1 Cette syntaxe est utilisée afin de définir le début de l'espace de codage pour une classe de la catégorie des chaînes de bits, le codage des valeurs abstraites de cette classe, une déclaration facultative que tous les codages de bits offrent un pointeur d'identification spécifié, et une spécification de la façon de coder un type contenu.

23.2.3.2 La classe **#CONDITIONAL-REPETITION** qui est appliquée par cet objet de codage ne doit pas spécifier "**REPLACE**" à moins qu'il ne s'agisse de "**REPLACE STRUCTURE**".

23.2.3.3 Si n'importe lequel des objets de la classe **#CONDITIONAL-REPETITION** contient une clause "**REPLACE STRUCTURE**", dans ce cas tous les objets de la classe **#CONDITIONAL-REPETITION** doivent contenir une clause "**REPLACE STRUCTURE**".

23.2.3.4 S'il y a une clause "**REPLACE STRUCTURE**" dans les objets de codage de la classe **#CONDITIONAL-REPETITION**, dans ce cas aucun autre paramètre ne doit être activé.

23.2.3.5 La première transformée contenue dans "**TRANSFORMS**" (si ce champ est présent) doit avoir une source qui est un bit isolé et la dernière transformée doit avoir un résultat qui est une chaîne de bits. Les chaînes de bits produites pour un bit unité et pour un bit zéro doivent former un ensemble autodélimitateur (voir 3.2.41).

NOTE – Cela implique que la transformée finale soit autodélimitatrice.

23.2.3.6 Il y a erreur de spécification ou d'application ECN si une éventuelle transformée dans le champ "**TRANSFORMS**" n'est pas réversible pour la valeur abstraite à laquelle elle est appliquée.

23.2.3.7 Exactement un seul champ choisi entre "**REPETITION-ENCODING**" et "**REPETITION-ENCODINGS**" doit être activé.

23.2.3.8 Si un objet de codage contenu dans la liste ordonnée "**REPETITION-ENCODINGS**" est définif au moyen de "**IF**", dans ce cas tous les objets de codage précédents de cette liste doivent être définis au moyen de "**IF**".

23.2.3.9 Si "**DETERMINED BY**" est "**not-needed**" dans une ou plusieurs des spécifications "**REPETITION-ENCODING(S)**", dans ce cas les valeurs abstraites de la chaîne de bits originale à laquelle cet objet de codage est appliqué doivent être contraintes à un ensemble autodélimitateur fini qui peut être identifié à partir de la spécification ECN.

NOTE – Ce serait le cas si les valeurs de chaîne de bits provenaient d'un codage de type Huffman (voir Annexe E) spécifié par mappage de valeurs entières sur des bits (voir § 19.7), ou si les valeurs de chaîne de bits avaient une limite visible par la notation ECN les limitant à un nombre fixe de bits.

23.2.3.10 Si "**EXHIBITS HANDLE**" est activé, dans ce cas tous les codages de valeurs associés à cette classe doivent offrir le pointeur d'identification spécifié.

NOTE – Cela nécessitera en général des restrictions applicables aux valeurs abstraites du type associé ou l'adjonction de bits redondants dans la transformation en bits, ou les deux.

23.2.3.11 Si "**EXHIBITS HANDLE**" est activé, dans ce cas "**ALIGNED TO**" ne doit être activé dans aucune des spécifications "**REPETITION-ENCODING(S)**".

23.2.4 Actions du codeur

23.2.4.1 Pour tout groupe de propriétés de codage qui est activé, le codeur doit exécuter les actions de codage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) préalignement et bourrage;

b) pointeur de début;

c) codage de valeur de bits (voir § 23.2.4.2);

d) pointeur d'identification;

e) codage de type contenu.

23.2.4.2 Pour le codage de valeur des bits, le codeur doit:

a) inverser l'ordre des bits dans la valeur abstraite de la chaîne de bits entière si "**VALUE-REVERSAL**" est activée à **TRUE**;

b) traiter la valeur de chaîne de bits comme une répétition de bit;

c) appliquer le champ "**TRANSFORMS**" (éventuellement) spécifié à chaque bit afin de produire une répétition de bits;

d) coder la répétition par l'application du premier champ "**REPETITION-ENCODING(S)**" dont la condition est satisfaite.

23.2.4.3 Il y a erreur de spécification ECN s'il n'y a pas de champ "**REPETITION-ENCODING(S)**" dont la condition est satisfaite.

23.2.5 Actions du décodeur

23.2.5.1 Pour tout groupe de propriétés de codage qui est activé, le décodeur doit exécuter les actions de décodage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) préalignement et bourrage;

b) pointeur de début;

c) décodage de valeur des bits (voir § 23.2.5.2);

d) décodage du type contenu.

23.2.5.2 Pour le décodage de valeur des bits, le décodeur doit utiliser le champ "**REPETITION-ENCODING(S)**" afin de déterminer l'espace de répétition et de rétablir l'ordre original des bits au moyen de la spécification "**BIT-REVERSAL**".

23.2.5.3 Si "**TRANSFORMS**" est activé, dans ce cas le décodeur doit utiliser la propriété autodélimitatrice du codage de chaque bit afin de déterminer l'extrémité de chaque répétition, et doit inverser les transformées afin de rétablir la valeur de la chaîne de bits originale.

23.2.5.4 Si "**VALUE-REVERSAL**" est activé à **TRUE**, dans ce cas l'ordre final des bits dans la valeur abstraite de la chaîne de bits doit être inversé.

23.3 Définition des objets de codage pour les classes de la catégorie des booléens

23.3.1 La syntaxe définie

La syntaxe de définition des objets de codage pour les classes de la catégorie des booléens est définie comme suit:

```
#BOOL ::= ENCODING-CLASS {

    -- Spécification de remplacement limité aux structures (voir § 22.1)
    &#Replacement-structure
         OPTIONAL,
    &replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,

    -- Spécification de préalignement et de bourrage (voir § 22.2)
    &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding        Padding DEFAULT zero,
    &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                       DEFAULT bits:'0'B,
```

```
      -- Spécification du pointeur de début (voir § 22.3)
      &start-pointer                    REFERENCE  OPTIONAL,
      &start-pointer-unit               Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL,

      -- Spécification de l'espace de codage (voir § 22.4)
      &encoding-space-size              EncodingSpaceSize
                                        DEFAULT self-delimiting-values,
      &encoding-space-unit              Unit (ALL EXCEPT repetitions)
                                        DEFAULT bit,
      &encoding-space-determination     EncodingSpaceDetermination
                                        DEFAULT field-to-be-set,
      &encoding-space-reference         REFERENCE OPTIONAL,
      &Encoder-transforms               #TRANSFORM ORDERED OPTIONAL,
      &Decoder-transforms               #TRANSFORM ORDERED OPTIONAL,

      -- codage de valeur booléenne
      &value-true-pattern               Pattern DEFAULT bits:'1'B,
      &value-false-pattern              Pattern DEFAULT bits:'0'B,

      -- Bourrage et justification de valeur (voir § 22.8)
      &value-justification              Justification DEFAULT right:0,
      &value-pre-padding                Padding DEFAULT zero,
      &value-pre-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
      &value-post-padding               Padding DEFAULT zero,
      &value-post-pattern               Non-Null-Pattern DEFAULT bits:'0'B,
      &unused-bits-determination        UnusedBitsDetermination
                                        DEFAULT field-to-be-set,
      &unused-bits-reference            REFERENCE OPTIONAL,
      &Unused-bits-encoder-transforms   #TRANSFORM ORDERED OPTIONAL,
      &Unused-bits-decoder-transforms   #TRANSFORM ORDERED OPTIONAL,

      -- Spécification de pointeur d'identification (voir § 22.9)
      &exhibited-handle                 PrintableString OPTIONAL,
      &Handle-positions                 INTEGER (0..MAX) OPTIONAL,
      &handle-value                     HandleValue DEFAULT tag:any,

      -- Spécification de l'inversion de l'ordre des bits (voir § 22.12)
      &bit-reversal                     ReversalSpecification
                                        DEFAULT no-reversal

} WITH SYNTAX {
      [REPLACE
            [STRUCTURE]
            WITH &#Replacement-structure
                  [ENCODED BY &replacement-structure-encoding-object]]
      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]

      ENCODING-SPACE
            [SIZE &encoding-space-size
                  [MULTIPLE OF &encoding-space-unit]]
            [DETERMINED BY &encoding-space-determination]
            [USING &encoding-space-reference
                  [ENCODER-TRANSFORMS &Encoder-transforms]
                  [DECODER-TRANSFORMS &Decoder-transforms]]

      [TRUE-PATTERN &value-true-pattern]
      [FALSE-PATTERN &value-false-pattern]
      [VALUE-PADDING
            [JUSTIFIED &value-justification]
            [PRE-PADDING &value-pre-padding
                  [PATTERN &value-pre-pattern]]
```

```
                [POST-PADDING &value-post-padding
                      [PATTERN &value-post-pattern]]
                [UNUSED BITS
                      [DETERMINED BY &unused-bits-determination]
                      [USING &unused-bits-reference
                            [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                            [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
          [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
                [AS &handle-value]]
          [BIT-REVERSAL &bit-reversal]

    }
```

23.3.2 Finalité et restrictions

23.3.2.1 Cette syntaxe est utilisée afin de définir le début de l'espace de codage pour une classe de la catégorie des booléens, le codage des valeurs abstraites de cette classe, leur positionnement à l'intérieur de l'espace de codage, une déclaration facultative que tous les codages de bits offrent un pointeur d'identification spécifié, et une éventuelle inversion de l'ordre des bits de l'espace de codage pour les booléens.

23.3.2.2 Si "**REPLACE**" est activé, dans ce cas aucun autre groupe de propriétés de codage ne doit être activé.

23.3.2.3 Au plus une seule des spécifications "**TRUE-PATTERN**" et "**FALSE-PATTERN**" doit être mise à "**different:any**".

23.3.2.4 Si l'option "**any-of-length**" est sélectionnée pour l'une des deux séquences (ou les deux), dans ce cas la longueur en bits des deux séquences doit être différente.

23.3.2.5 Si "**ENCODING-SPACE SIZE**" est "**self-delimiting**", dans ce cas "**TRUE-PATTERN**" et "**FALSE-PATTERN**" doivent former un ensemble autodélimitateur (voir 3.2.41).

23.3.2.6 "**UNUSED BITS DETERMINED BY**" ne doit pas être "**not-needed**" sauf si:

a) les deux séquences sont des multiples entiers de "**ENCODING-SPACE MULTIPLE OF**" unités et "**ENCODING SPACE SIZE**" est "**variable-with-determinant**"; ou

b) les deux séquences sont de même longueur; ou

c) "**JUSTIFIED**" est "**left**" et les séquences forment un ensemble autodélimitateur; ou

d) "**JUSTIFIED**" est "**right**" et l'inverse des séquences forme un ensemble autodélimitateur (voir 3.2.41).

23.3.2.7 S'il y a des bits inutilisés dans l'espace de codage, dans ce cas "**VALUE-PADDING**" doit être activé.

23.3.3 Actions du codeur

23.3.3.1 Pour tout groupe de propriétés de codage qui est activé, le codeur doit exécuter les actions de codage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) remplacement;

b) préalignement et bourrage;

c) pointeur de début;

d) espace de codage (voir § 23.3.3.2);

e) codage de valeur (voir § 23.3.3.3);

f) bourrage et justification de valeur;

g) pointeur d'identification;

h) inversion de l'ordre des bits.

23.3.3.2 Si "**ENCODING-SPACE SIZE**" n'est pas mis à une valeur positive, dans ce cas la taille de l'espace de codage "s" est le plus petit nombre de "**MULTIPLE OF**" unités (sous réserve du § 23.3.3.3) qui peut accepter la séquence de la valeur qui doit être codée.

23.3.3.3 Un codeur (à titre d'option de codeur) peut augmenter la taille de l'espace de codage "s" (telle que déterminée au § 23.3.3.2) de "**MULTIPLE OF**" unités (sous réserve d'éventuelles restrictions imposées par l'étendue de valeurs d'un quelconque champ "**field-to-be-set**" ou "**field-to be-used**") si le champ "**ENCODING-SPACE SIZE**" est activé à la valeur "**encoder-option-with-determinant**".

23.3.3.4 Le nombre de bits inutilisés peut être déterminé à partir de la valeur "s" et à partir de la séquence de la valeur à coder.

23.3.3.5 Si le nombre de bits inutilisés est différent de zéro, dans ce cas la spécification "**VALUE-PADDING**" doit être appliquée.

23.3.4 Actions du décodeur

23.3.4.1 Pour tout groupe de propriétés de codage qui est activé, le décodeur doit exécuter les actions de décodage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) préalignement et bourrage;

b) pointeur de début;

c) espace de codage;

d) inversion de l'ordre des bits;

e) bourrage et justification de valeur;

f) le décodage de valeur (voir § 23.3.4.2).

23.3.4.2 Le décodage de valeur doit être effectué par identification de la spécification "**TRUE-PATTERN**" ou "**FALSE-PATTERN**" comme suit:

a) au moyen d'une éventuelle détermination de la spécification "**UNUSED BITS**" détermination; ou

b) au moyen de la propriété autodélimitatrice des séquences ou de leurs inversions.

23.4 Définition des objets de codage pour les classes de la catégorie des chaînes de caractères

23.4.1 La syntaxe définie

La syntaxe de définition des objets de codage pour les classes de la catégorie des chaînes de caractères est définie comme suit:

```
#CHARS ::= ENCODING-CLASS {

      -- Spécification de préalignement et de bourrage (voir § 22.2)
      &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &encoding-space-pre-padding        Padding DEFAULT zero,
      &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                         DEFAULT bits:'0'B,

      -- Spécification du pointeur de début (voir § 22.3)
      &start-pointer                     REFERENCE  OPTIONAL,
      &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

      -- codage de valeur de caractères
      &value-reversal                    BOOLEAN DEFAULT FALSE,
      &Transforms                        #TRANSFORM ORDERED OPTIONAL,
      &Chars-repetition-encodings        #CONDITIONAL-REPETITION ORDERED OPTIONAL,
      &chars-repetition-encoding         #CONDITIONAL-REPETITION OPTIONAL,
      -- Spécification de pointeur d'identification (voir § 22.9)
      &exhibited-handle                  PrintableString OPTIONAL,
      &Handle-positions                  INTEGER (0..MAX) OPTIONAL,
      &handle-value                      HandleValue DEFAULT tag:any

} WITH SYNTAX {
      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
```

```
        [VALUE-REVERSAL          &value-reversal]
        [TRANSFORMS              &Transforms]
        [REPETITION-ENCODINGS    &Chars-repetition-encodings]
        [REPETITION-ENCODING     &chars-repetition-encoding]
        [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
              [AS &handle-value]]

}
```

23.4.2 Modèle de codage de classes de la catégorie des chaînes de caractères

23.4.2.1 Le modèle des codages de chaîne de caractères est le suivant:

a) l'ordre des caractères dans la chaîne de caractères peut être inversé;

b) les caractères sont considérés comme une répétition d'un caractère;

c) il y a une transformée (spécifiée par "**TRANSFORMS**") dans laquelle chaque caractère est transformé en une chaîne de bits autodélimitatrice;

d) soit "**REPETITION-ENCODING**" ou "**REPETITION-ENCODINGS**" spécifie la façon dont la répétition de chaîne de bits doit être codée.

NOTE – Le seul objectif de permettre "**REPETITION-ENCODING**" ainsi que "**REPETITION-ENCODINGS**" est d'offrir une syntaxe qui ne contient pas de double accolade ("**{{**") dans le cas courant d'un codage conditionnel isolé. L'utilisation de "**REPETITION-ENCODINGS**" lorsqu'il y a un codage conditionnel isolé est déconseillée mais autorisée.

23.4.2.2 Les limites (si présentes) de la classe en cours de codage (une classe de la catégorie des chaînes de caractères) sont des limites applicables au nombre de caractères contenus dans la chaîne de caractères formant chaque valeur abstraite.

23.4.2.3 Considérées comme une répétition de caractères, ces limites doivent être interprétées comme étant applicables au nombre de répétitions, et peuvent être utilisées dans la spécification des objets de codage de classe **#REPETITION-ENCODING** qui sont utilisés dans la spécification de cet objet de codage.

23.4.3 Finalité et restrictions

23.4.3.1 Cette syntaxe est utilisée afin de définir le début de l'espace de codage pour une classe de la catégorie des chaînes de caractères, le codage des valeurs abstraites associées à cette classe, une déclaration facultative que tous les codages de caractères offrent un pointeur d'identification spécifié.

23.4.3.2 La classe **#CONDITIONAL-REPETITION** qui est appliquée par cet objet de codage ne doit pas spécifier "**REPLACE**" à moins qu'il ne s'agisse de "**REPLACE STRUCTURE**".

23.4.3.3 Si n'importe lequel des objets de la classe **#CONDITIONAL-REPETITION** contient une clause "**REPLACE STRUCTURE**", dans ce cas tous les objets de la classe **#CONDITIONAL-REPETITION** doivent contenir une clause "**REPLACE STRUCTURE**".

23.4.3.4 S'il n'y a pas de clause "**REPLACE STRUCTURE**" dans les objets de codage de la classe **#CONDITIONAL-REPETITION**, dans ce cas la spécification "**TRANSFORMS**" doit être activée. S'il y a une clause "**REPLACE STRUCTURE**" dans les objets de codage de la classe **#CONDITIONAL-REPETITION**, dans ce cas aucun autre paramètre ne doit être activé.

23.4.3.5 La première transformée de "**TRANSFORMS**" doit avoir une source qui est un caractère isolé et la dernière transformée doit avoir un résultat qui est une chaîne de bits. Les chaînes de bits produites pour l'ensemble de tous les caractères en cours de codage doivent former un ensemble autodélimitateur (voir 3.2.41).

NOTE – Cela implique que la transformée finale soit autodélimitatrice.

23.4.3.6 Il y a erreur de spécification ou d'application ECN si une éventuelle transformée dans la spécification "**TRANSFORMS**" n'est pas réversible pour la valeur abstraite à laquelle elle est appliquée.

23.4.3.7 Exactement une des spécifications "**REPETITION-ENCODING**" ou "**REPETITION-ENCODINGS**" doit être activée.

23.4.3.8 Si un objet de codage contenu dans la liste ordonnée de "**REPETITION-ENCODINGS**" est défini au moyen de "**IF**", dans ce cas tous les objets de codage précédents de cette liste doivent être définis au moyen de "**IF**".

23.4.3.9 Si "**EXHIBITS HANDLE**" est activé, dans ce cas tous les codages de valeurs associés à cette classe doivent offrir le pointeur d'identification spécifié.

NOTE – Cela nécessitera en général des restrictions applicables aux valeurs abstraites du type associé, ou l'inclusion de bits redondants dans le codage de chaque caractère, ou les deux.

23.4.3.10 Si "**EXHIBITS HANDLE**" est activé, la production "**ALIGNED TO**" ne doit être fixée dans aucune des spécifications "**REPETITION-ENCODING(S)**".

23.4.4 Actions du codeur

23.4.4.1 Pour tout groupe de propriétés de codage qui est activé, le codeur doit exécuter les actions de codage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) préalignement et bourrage;

b) pointeur de début;

c) codage de valeur de caractères (voir § 23.4.4.3);

d) codage de répétition comme spécifié par la première spécification "**REPETITION-ENCODING(S)**" dont la condition est satisfaite;

e) spécification de pointeur d'identification.

23.4.4.2 Il y a erreur de spécification ECN s'il n'y a pas de spécification "**REPETITION-ENCODING(S)**" dont la condition est satisfaite.

23.4.4.3 Pour le codage de valeur de chaîne de caractères, le codeur doit:

a) inverser l'ordre de caractères dans la valeur abstraite de la chaîne de caractères entière si la spécification "**VALUE-REVERSAL**" est activée à **TRUE**;

b) traiter la valeur des caractères d'une chaîne comme une répétition de caractères;

c) appliquer la production "**TRANSFORMS**" spécifiée (si présent) à chaque caractère afin de produire une répétition de bits;

d) coder la répétition par l'application du champ "**REPETITION-ENCODING(S)**".

23.4.5 Actions du décodeur

23.4.5.1 Pour tout groupe de propriétés de codage qui est activé, le décodeur doit exécuter les actions de décodage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) préalignement et bourrage;

b) pointeur de début;

c) décodage de répétition comme spécifié par la première spécification "**REPETITION-ENCODING(S)**" dont la condition est satisfaite;

d) décodage de valeur d'une chaîne de caractères (voir § 23.4.5.2).

23.4.5.2 Pour le décodage de valeur d'une chaîne de caractères, le décodeur doit utiliser la spécification "**REPETITION-ENCODING(S)**" afin de déterminer l'espace de répétition et de rétablir les caractères originaux. Si la spécification "**TRANSFORMS**" est activée, le décodeur doit utiliser la propriété autodélimitatrice du codage de chaque caractère (qui contient une éventuelle longueur fixe) afin de déterminer l'extrémité de chaque répétition, et doit inverser les transformées afin de rétablir une valeur de chaîne de caractères.

23.4.5.3 Si la spécification "**VALUE-REVERSAL**" est activée à **TRUE**, dans ce cas l'ordre final des caractères contenus dans la valeur abstraite de chaîne de caractères doit être inversé.

23.5 Définition des objets de codage pour les classes de la catégorie des concaténations

23.5.1 La syntaxe définie

La syntaxe de définition des objets de codage pour les classes de la catégorie des concaténations est définie comme suit:

```
#CONCATENATION ::= ENCODING-CLASS {

    -- Spécification de remplacement complète (voir § 22.1)
    &#Replacement-structure
          OPTIONAL,
    &#Replacement-structure2
          OPTIONAL,
    &replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,
    &replacement-structure-encoding-object2 &#Replacement-structure2   OPTIONAL,
    &#Head-end-structure                                                OPTIONAL,
    &#Head-end-structure2                                               OPTIONAL,
```

```
        -- Spécification de préalignement et de bourrage (voir § 22.2)
        &encoding-space-pre-alignment-unitUnit (ALL EXCEPT repetitions) DEFAULT bit,
        &encoding-space-pre-padding       Padding DEFAULT zero,
        &encoding-space-pre-pattern       Non-Null-Pattern (ALL EXCEPT different:any)
                                          DEFAULT bits:'0'B,

        -- Spécification du pointeur de début (voir § 22.3)
        &start-pointer                    REFERENCE  OPTIONAL,
        &start-pointer-unit               Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL,

        -- Spécification de l'espace de codage (voir § 22.4)
        &encoding-space-size              EncodingSpaceSize
                                          DEFAULT self-delimiting-values,
        &encoding-space-unit              Unit (ALL EXCEPT repetitions)
                                          DEFAULT bit,
        &encoding-space-determination     EncodingSpaceDetermination
                                          DEFAULT field-to-be-set,
        &encoding-space-reference         REFERENCE OPTIONAL,
        &Encoder-transforms               #TRANSFORM ORDERED OPTIONAL,
        &Decoder-transforms               #TRANSFORM ORDERED OPTIONAL,

        -- Spécification de concaténation (voir § 22.10)
        &concatenation-order              ENUMERATED {textual, tag, random}
                                          DEFAULT textual,
        &concatenation-alignment          ENUMERATED {none, aligned}
                                          DEFAULT aligned,
        &concatenation-handle             PrintableString
                                          DEFAULT "default-handle",

        -- Bourrage et justification de valeur (voir § 22.8)
        &value-justification              Justification DEFAULT right:0,
        --&value-pre-padding              Padding DEFAULT zero,
        &value-pre-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
        &value-post-padding               Padding DEFAULT zero,
        &value-post-pattern               Non-Null-Pattern DEFAULT bits:'0'B,
        &unused-bits-determination        UnusedBitsDetermination
                                          DEFAULT field-to-be-set,
        &unused-bits-reference            REFERENCE OPTIONAL,
        &Unused-bits-encoder-transforms   #TRANSFORM ORDERED OPTIONAL,
        &Unused-bits-decoder-transforms   #TRANSFORM ORDERED OPTIONAL,

        -- Spécification de pointeur d'identification (voir § 22.9)
        &exhibited-handle                 PrintableString OPTIONAL,
        &Handle-positions                 INTEGER (0..MAX) OPTIONAL,
        &handle-value                     HandleValue DEFAULT tag:any,

        -- Spécification de l'inversion de l'ordre des bits (voir § 22.12)
        &bit-reversal                     ReversalSpecification
                                          DEFAULT no-reversal

    } WITH SYNTAX {
          [REPLACE
                [STRUCTURE]
                [COMPONENT]
                [ALL COMPONENTS]
                [OPTIONALS]
                [NON-OPTIONALS]
                WITH &#Replacement-structure
                      [ENCODED BY &replacement-structure-encoding-object
                            [INSERT AT HEAD &#Head-end-structure]]
                      [AND OPTIONALS WITH &#Replacement-structure2
                            [ENCODED BY &replacement-structure-encoding-object2
                                  [INSERT AT HEAD &#Head-end-structure2]]] ]
          [ALIGNED TO
                [NEXT]
                [ANY]
                      &encoding-space-pre-alignment-unit
                      [PADDING &encoding-space-pre-padding
                            [PATTERN &encoding-space-pre-pattern]]]
```

```
        [START-POINTER   &start-pointer
              [MULTIPLE OF     &start-pointer-unit]
              [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]

        ENCODING-SPACE
              [SIZE &encoding-space-size
                    [MULTIPLE OF &encoding-space-unit]]
              [DETERMINED BY &encoding-space-determination]
              [USING &encoding-space-reference
                    [ENCODER-TRANSFORMS &Encoder-transforms]
                    [DECODER-TRANSFORMS &Decoder-transforms]]

        [CONCATENATION
              [ORDER &concatenation-order]
              [ALIGNMENT &concatenation-alignment]
              [HANDLE &concatenation-handle]]
        [VALUE-PADDING
              [JUSTIFIED &value-justification]
              [PRE-PADDING &value-pre-padding
                    [PATTERN &value-pre-pattern]]
              [POST-PADDING &value-post-padding
                    [PATTERN &value-post-pattern]]
              [UNUSED BITS
                    [DETERMINED BY &unused-bits-determination]
                    [USING &unused-bits-reference
                          [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                          [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
        [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
              [AS &handle-value]]
        [BIT-REVERSAL &bit-reversal]
}
```

23.5.2 Finalité et restrictions

23.5.2.1 Cette syntaxe est utilisée afin de définir le début de l'espace de codage pour une classe de la catégorie des concaténations, la façon dont les codages des composants doivent être combinés, leur positionnement à l'intérieur de l'espace de codage, une déclaration facultative que tous les codages offrent un pointeur d'identification spécifié, et éventuelle inversion de l'ordre des bits de l'espace de codage.

23.5.2.2 Si "**REPLACE STRUCTURE**" est activé, dans ce cas aucun autre paramètre de codage des groupes ne doit être activé.

23.5.2.3 "**ENCODING-SPACE SIZE**" doit être soit "**variable-with-determinant**" ou "**self-delimiting-values**".

23.5.2.4 Si "**EXHIBITS HANDLE**" est activé, alors le codage de toutes les valeurs abstraites possibles associées à cette classe doivent offrir le pointeur d'identification défini

NOTE – Cela sera souvent réalisé par vérification du fait que le premier composant de la concaténation, ou une insertion en tête de séquence, a affiché le pointeur d'identification.

23.5.3 Actions du codeur

23.5.3.1 Pour tout groupe de propriétés de codage qui est activé, le codeur doit exécuter les actions de codage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) remplacement;

b) préalignement et bourrage;

c) pointeur de début;

d) espace de codage (voir § 23.5.3.2);

e) concaténation;

f) bourrage et justification de valeur;

g) spécification de pointeur d'identification;

h) inversion de l'ordre des bits.

23.5.3.2 Si "**ENCODING SPACE**" est "**variable-with-determinant**", il doit indiquer le nombre minimal de "**MULTIPLE OF**" unités nécessaire afin de contenir la concaténation.

23.5.4 Actions du décodeur

23.5.4.1 Pour tout groupe de propriétés de codage qui est activé, le décodeur doit exécuter les actions de décodage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) préalignement et bourrage;

b) pointeur de début;

c) espace de codage;

d) inversion de l'ordre des bits;

e) bourrage et justification de valeur;

f) concaténation.

23.6 Définition des objets de codage pour les classes de la catégorie des entiers

23.6.1 La syntaxe définie

La syntaxe de définition des objets de codage pour les classes de la catégorie des entiers est définie comme suit:

```
#INT ::= ENCODING-CLASS {

      -- Codage d'entier
      &Integer-encodings          #CONDITIONAL-INT ORDERED OPTIONAL,
      &integer-encoding           #CONDITIONAL-INT OPTIONAL

} WITH SYNTAX {
      [ENCODINGS &Integer-encodings]
      [ENCODING &integer-encoding]

}
```

23.6.2 Finalité et restrictions

23.6.2.1 Cette syntaxe est utilisée afin de définir le codage d'une classe de la catégorie des entiers au moyen de la spécification d'un ou de plusieurs codages de la classe **#CONDITIONAL-INT**.

23.6.2.2 Exactement une des spécifications "**ENCODING**" et "**ENCODINGS**" doit être activée.

NOTE – Le seul objectif de permettre "**ENCODING**" ainsi que "**ENCODINGS**" est d'offrir une syntaxe qui ne contient pas de double accolade ("**{{**") dans le cas courant d'un seul objet de codage. L'utilisation de "**ENCODINGS**" lorsqu'il y a un seul objet de codage est déconseillée mais autorisée.

23.6.2.3 Si un objet de codage contenu dans la liste ordonnée de la spécification "**ENCODINGS**" est défini au moyen de "**IF**", dans ce cas tous les objets de codage précédents de cette liste doivent être définis au moyen de "**IF**".

23.6.3 Actions du codeur

23.6.3.1 Le codeur doit sélectionner et appliquer le premier objet de codage de la classe **#CONDITIONAL-INT** contenu dans "**ENCODING(S)**" dont les conditions sont satisfaites. Il y a erreur de spécification ECN si aucun des codages conditionnels ont des conditions qui sont satisfaites.

NOTE – Il serait inhabituel mais non interdit s'il y avait des objets de codage de la classe **#CONDITIONAL-INT** qui ne pourraient jamais être utilisés parce que les conditions relatives à l'utilisation d'objets de codage antérieurs seraient toujours satisfaites.

23.6.4 Actions du décodeur

23.6.4.1 Le décodeur doit sélectionner et utiliser le premier objet de codage **#CONDITIONAL-INT** contenu dans la classe "**ENCODING(S)**" dont les conditions sont satisfaites.

23.7 Définition des objets de codage pour la classe #CONDITIONAL-INT

23.7.1 La syntaxe définie

La syntaxe de définition des objets de codage pour la classe **#CONDITIONAL-INT** est définie comme suit:

```
#CONDITIONAL-INT ::= ENCODING-CLASS {

      -- Condition (voir § 21.11)
      &range-condition                        RangeCondition  OPTIONAL,
```

```
      -- Spécification de remplacement limité aux structures (voir § 22.1)
      &#Replacement-structure
            OPTIONAL,
      &replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,

      -- Spécification de préalignement et de bourrage (voir § 22.2)
      &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &encoding-space-pre-padding        Padding DEFAULT zero,
      &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                         DEFAULT bits:'0'B,

      -- Spécification du pointeur de début (voir § 22.3)
      &start-pointer                     REFERENCE  OPTIONAL,
      &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

      -- Spécification de l'espace de codage (voir § 22.4)
      &encoding-space-size               EncodingSpaceSize
                                         DEFAULT self-delimiting-values,
      &encoding-space-unit               Unit (ALL EXCEPT repetitions)
                                         DEFAULT bit,
      &encoding-space-determination      EncodingSpaceDetermination
                                         DEFAULT field-to-be-set,
      &encoding-space-reference          REFERENCE OPTIONAL,
      &Encoder-transforms                #TRANSFORM ORDERED OPTIONAL,
      &Decoder-transforms                #TRANSFORM ORDERED OPTIONAL,

      -- Codage de valeur
      &Transform                    #TRANSFORM ORDERED OPTIONAL,
      &encoding                     ENUMERATED
                                    {positive-int, twos-complement,
                                    reverse-positive-int, reverse-twos-complement}
                                    DEFAULT twos-complement,

      -- Bourrage et justification de valeur (voir § 22.8)
      &value-justification               Justification DEFAULT right:0,
      &value-pre-padding                 Padding DEFAULT zero,
      &value-pre-pattern                 Non-Null-Pattern DEFAULT bits:'0'B,
      &value-post-padding                Padding DEFAULT zero,
      &value-post-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
      &unused-bits-determination         UnusedBitsDetermination
                                         DEFAULT field-to-be-set,
      &unused-bits-reference             REFERENCE OPTIONAL,
      &Unused-bits-encoder-transforms    #TRANSFORM ORDERED OPTIONAL,
      &Unused-bits-decoder-transforms    #TRANSFORM ORDERED OPTIONAL,

      -- Spécification de pointeur d'identification (voir § 22.9)
      &exhibited-handle                  PrintableString OPTIONAL,
      &Handle-positions                  INTEGER (0..MAX) OPTIONAL,
      &handle-value                      HandleValue DEFAULT tag:any,

      -- Spécification de l'inversion de l'ordre des bits (voir § 22.12)
      &bit-reversal                      ReversalSpecification
                                         DEFAULT no-reversal

} WITH SYNTAX {
      [IF &range-condition] [ELSE]
      [REPLACE
            [STRUCTURE]
            WITH &#Replacement-structure
                  [ENCODED BY &replacement-structure-encoding-object]]
      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
```

```
            ENCODING-SPACE
                  [SIZE &encoding-space-size
                        [MULTIPLE OF &encoding-space-unit]]
                  [DETERMINED BY &encoding-space-determination]
                  [USING &encoding-space-reference
                        [ENCODER-TRANSFORMS &Encoder-transforms]
                        [DECODER-TRANSFORMS &Decoder-transforms]]

            [TRANSFORMS     &Transforms]
            [ENCODING       &encoding]
            [VALUE-PADDING
                  [JUSTIFIED &value-justification]
                  [PRE-PADDING &value-pre-padding
                        [PATTERN &value-pre-pattern]]
                  [POST-PADDING &value-post-padding
                        [PATTERN &value-post-pattern]]
                  [UNUSED BITS
                        [DETERMINED BY &unused-bits-determination]
                        [USING &unused-bits-reference
                              [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                              [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
            [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
                  [AS &handle-value]]
            [BIT-REVERSAL &bit-reversal]

      }
```

23.7.2 Finalité et restrictions

23.7.2.1 Cette syntaxe est utilisée afin de définir un objet de codage de la classe **`#CONDITIONAL-INT`**. La seule utilisation d'un tel objet de codage est la spécification d'un objet de codage d'une classe de la catégorie des entiers.

23.7.2.2 La syntaxe autorise la spécification d'une condition unique concernant les limites de l'entier pour que ce codage soient appliqué (utilisation de "**`IF`**"). Elle autorise également la spécification qu'il n'y a pas de condition. L'utilisation de "**`ELSE`**", ou l'omission d'aussi bien "**`IF`**" que "**`ELSE`**" spécifie qu'il n'y a pas de condition.

23.7.2.3 Au moyen de cette syntaxe, le spécificateur ECN peut définir le début de l'espace de codage afin de coder: une classe de la catégorie des entiers, des valeurs abstraites associées à cette classe, leur positionnement à l'intérieur de l'espace de codage, et une éventuelle inversion de l'ordre des bits de l'espace de codage.

23.7.2.4 Au plus un des mots "**`IF`**" ou "**`ELSE`**" doit être présent.

23.7.2.5 Si "**`REPLACE`**" est activé, dans ce cas aucun autre groupe de propriétés de codage ne doit être activé.

23.7.2.6 Il y a erreur de spécification ou d'application ECN si une éventuelle transformée contenue dans la spécification "**`TRANSFORMS`**" n'est pas réversible pour la valeur abstraite à laquelle elle est appliquée. La première transformée de "**`TRANSFORMS`**", si présente, doit avoir une source qui est un entier et la dernière transformée doit avoir un résultat qui est un entier.

NOTE – L'essai de la condition "**`IF`**" a lieu aux limites de la valeur originale, et n'est pas affecté par ces transformées.

23.7.2.7 La transformation "**`INT-TO-INT`**" ayant la valeur "**`substract:lower-bound`**" ne doit être incluse que si la condition "**`IF`**" restreint l'application de ce codage aux classes de la catégorie des entiers avec une limite inférieure, et (si présente) doit être la première transformée dans la liste.

23.7.2.8 La production "**`ENCODING-SPACE SIZE`**" ne doit pas être à la valeur "**`fixed-to-max`**" à moins que la condition "**`IF`**" ne restreigne le codage à une classe ayant aussi bien une limite supérieure qu'une limite inférieure.

23.7.2.9 "**`ENCODING-SPACE SIZE`**" ne doit pas être mis à "**`self-delimiting-values`**".

23.7.2.10 Si "**`EXHIBITS HANDLE`**" est activé, dans ce cas le spécificateur déclare par assertion que le codage de toutes les valeurs fait apparaître le pointeur d'identification.

NOTE – Cela nécessitera normalement l'utilisation de "**`VALUE-PADDING`**" avec justification à partir de la gauche afin de permettre au bourrage d'offrir le pointeur d'identification.

23.7.3 Actions du codeur

23.7.3.1 Le codeur doit détecter une erreur de spécification ou d'application ECN si n'importe laquelle des restrictions indiquée au § 23.7.2 est violée.

23.7.3.2 Pour tout groupe de propriétés de codage qui est activé, le codeur doit exécuter les actions de codage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) remplacement;

b) préalignement et bourrage;

c) pointeur de début;

d) espace de codage;

e) codage de valeur (voir ci-dessous);

f) bourrage et justification de valeur;

g) pointeur d'identification;

h) inversion de l'ordre des bits.

23.7.3.3 Le codeur doit appliquer les éventuelles spécifications "**TRANSFORMS**" à la valeur en cours de codage.

23.7.3.4 Le codeur doit utiliser le tableau suivant qui donne l'étendue des valeurs entières qui peuvent être codées sur "n" bits:

"**ENCODING**"	Valeur minimale	Valeur maximale
"**positive-int**"	0	$2^n - 1$
"**reverse-positive-int**"	0	$2^n - 1$
"**twos-complement**"	-2^{n-1}	$2^{n-1} - 1$
"**reverse-twos-complement**"	-2^{n-1}	$2^{n-1} - 1$

23.7.3.5 Le paramètre "**ENCODING**" sélectionne le codage en tant que codage par complément à deux ou en tant que codage d'entier positif, ou en tant que l'inversion d'un de ces codages. La spécification de codage par complément à deux et de codage d'entier positif est donnée dans la Rec. UIT-T X.690 | ISO/CEI 8825-1, § 8.3.2 et 8.3.3. Une inversion de ces codages est un codage dans lequel, après production des "n" bits, l'ordre de ces "n" bits est inversé.

23.7.3.6 Un codeur doit détecter une spécification ECN ou une erreur d'application si la valeur doit être codée dans un nombre de bits qui est insuffisant, comme spécifié au § 23.7.3.4.

23.7.3.7 Si la production "**ENCODING-SPACE SIZE**" est un entier positif, dans ce cas sa taille en bits est calculée en tant que "**SIZE**" multiplié par "**MULTIPLE OF**" unités. Si "**VALUE-PADDING**" n'est pas activé, dans ce cas sa valeur doit être le nombre de bits "n" que l'entier doit y coder et il n'y a pas de bits inutilisés. Si "**VALUE-PADDING**" est activé, dans ce cas le nombre de bits que l'entier doit y coder est réduit de la valeur entière "m" spécifiée pour "**JUSTIFIED**", et il y aura "m" bits inutilisés.

23.7.3.8 Si "**ENCODING-SPACE SIZE**" est "**fixed-to-max**", dans ce cas le codeur doit déterminer le nombre minimal de "**MULTIPLE OF**" unités qui possèdent un nombre de bits suffisant pour coder n'importe laquelle des valeurs de la classe, et doit procéder (comme spécifié ci-dessus) comme si "**SIZE**" était un entier positif mis à cette valeur.

23.7.3.9 Si la valeur de "**ENCODING-SPACE SIZE**" est "**variable-with-determinant**", dans ce cas le codeur doit déterminer le nombre minimal de "**MULTIPLE OF**" unités ("s", par exemple) qui possèdent un nombre de bits suffisant pour coder la valeur abstraite réelle en cours de codage, et doit procéder (comme spécifié ci-dessus) comme si "**SIZE**" était un entier positif mis à cette valeur.

23.7.3.10 Le codeur (à titre d'option de codeur) peut augmenter "s" (comme déterminé au § 23.7.3.9) de "**MULTIPLE OF**" unités (sous réserve d'éventuelles restrictions imposées par l'étendue de valeurs d'un quelconque champ "**field-to-be-set**" ou "**field-to-be-used**") si "**ENCODING-SPACE SIZE**" est activé à la valeur "**encoder-option-with-determinant**".

23.7.3.11 Le codeur doit alors procéder (comme spécifié ci-dessus) comme si "**SIZE**" était un entier positif mis à "s".

23.7.4 Actions du décodeur

23.7.4.1 Pour tout groupe de propriétés de codage qui est activé, le décodeur doit exécuter les actions de décodage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) préalignement et bourrage;

b) pointeur de début;

c) espace de codage;

d) inversion de l'ordre des bits;

e) bourrage et justification de valeur;

f) décodage de valeur (voir § 23.7.4.2).

23.7.4.2 Le décodeur doit reconstituer la valeur entière à partir des bits utilisés pour la coder, le décodage étant effectué conformément au codage spécifié, et doit ensuite inverser la valeur "**TRANSFORMS**" (si spécifiée) afin de rétablir la valeur originale abstraite.

23.8 Définition des objets de codage pour les classes de la catégorie néant

23.8.1 La syntaxe définie

La syntaxe de définition des objets de codage pour les classes de la catégorie néant est définie comme suit:

```
#NUL ::= ENCODING-CLASS {

      -- Spécification de remplacement limité aux structures (voir § 22.1)
      &#Replacement-structure
            OPTIONAL,
      &replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,

      -- Spécification de préalignement et de bourrage (voir § 22.2)
      &encoding-space-pre-alignment-unitUnit (ALL EXCEPT repetitions) DEFAULT bit,
      &encoding-space-pre-padding       Padding DEFAULT zero,
      &encoding-space-pre-pattern       Non-Null-Pattern (ALL EXCEPT different:any)
                                        DEFAULT bits:'0'B,

      -- Spécification du pointeur de début (voir § 22.3)
      &start-pointer                    REFERENCE  OPTIONAL,
      &start-pointer-unit               Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL,

      -- Spécification de l'espace de codage (voir § 22.4)
      &encoding-space-size              EncodingSpaceSize
                                        DEFAULT self-delimiting-values,
      &encoding-space-unit              Unit (ALL EXCEPT repetitions)
                                        DEFAULT bit,
      &encoding-space-determination     EncodingSpaceDetermination
                                        DEFAULT field-to-be-set,
      &encoding-space-reference         REFERENCE OPTIONAL,
      &Encoder-transforms               #TRANSFORM ORDERED OPTIONAL,
      &Decoder-transforms               #TRANSFORM ORDERED OPTIONAL,

      -- Structure de valeur
      &value-pattern                    Pattern (ALL EXCEPT different:any)
                                        DEFAULT bits:''B,
      -- Bourrage et justification de valeur (voir § 22.8)
      &value-justification              Justification DEFAULT right:0,
      &value-pre-padding                Padding DEFAULT zero,
      &value-pre-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
      &value-post-padding               Padding DEFAULT zero,
      &value-post-pattern               Non-Null-Pattern DEFAULT bits:'0'B,
      &unused-bits-determination        UnusedBitsDetermination
                                        DEFAULT field-to-be-set,
      &unused-bits-reference            REFERENCE OPTIONAL,
      &Unused-bits-encoder-transforms   #TRANSFORM ORDERED OPTIONAL,
      &Unused-bits-decoder-transforms   #TRANSFORM ORDERED OPTIONAL,

      -- Spécification de pointeur d'identification (voir § 22.9)
      &exhibited-handle                 PrintableString OPTIONAL,
      &Handle-positions                 INTEGER (0..MAX) OPTIONAL,
      &handle-value                     HandleValue DEFAULT tag:any,

      -- Spécification de l'inversion de l'ordre des bits (voir § 22.12)
      &bit-reversal                     ReversalSpecification
                                        DEFAULT no-reversal
```

```
} WITH SYNTAX {
      [REPLACE
            [STRUCTURE]
            WITH &#Replacement-structure
                  [ENCODED BY &replacement-structure-encoding-object]]
      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]

      ENCODING-SPACE
            [SIZE &encoding-space-size
                  [MULTIPLE OF &encoding-space-unit]]
            [DETERMINED BY &encoding-space-determination]
            [USING &encoding-space-reference
                  [ENCODER-TRANSFORMS &Encoder-transforms]
                  [DECODER-TRANSFORMS &Decoder-transforms]]

      [NULL-PATTERN &value-pattern]
      [VALUE-PADDING
            [JUSTIFIED &value-justification]
            [PRE-PADDING &value-pre-padding
                  [PATTERN &value-pre-pattern]]
            [POST-PADDING &value-post-padding
                  [PATTERN &value-post-pattern]]
            [UNUSED BITS
                  [DETERMINED BY &unused-bits-determination]
                  [USING &unused-bits-reference
                        [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                        [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
      [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]
      [BIT-REVERSAL &bit-reversal]

}
```

23.8.2 Finalité et restrictions

23.8.2.1 Cette syntaxe est utilisée afin de définir le codage d'une classe de la catégorie néant.

23.8.2.2 Si "**REPLACE STRUCTURE**" est activé, dans ce cas aucun autre groupe de propriétés de codage ne doit être activé.

23.8.2.3 Si la valeur de "**ENCODING-SPACE SIZE**" est positive, elle doit être suffisante pour contenir la taille de la structure "**NULL-PATTERN**" ainsi que tous bits ajoutés en tant que résultat d'une spécification "**VALUE-PADDING**".

23.8.2.4 S'il y a des bits inutilisés dans l'espace de codage, dans ce cas "**VALUE-PADDING**" doit être activé.

23.8.3 Actions du codeur

23.8.3.1 Pour tout groupe de propriétés de codage qui est activé, le codeur doit exécuter les actions de codage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) remplacement;

b) préalignement et bourrage;

c) pointeur de début;

d) espace de codage;

e) codage de valeur (voir § 23.8.3.2);

f) bourrage et justification de valeur;

g) pointeur d'identification;

h) inversion de l'ordre des bits.

23.8.3.2 Le codage de valeur doit être les bits de la structure "**NULL-PATTERN**".

23.8.3.3 Si "**ENCODING-SPACE SIZE**" est "**variable-with-determinant**" ou "**encoder-option-with-determinant**", ce doit être le nombre minimal de "**MULTIPLE OF**" unités nécessaire afin de contenir la séquence ("s", par exemple), sous réserve du § 23.8.3.4.

23.8.3.4 Un codeur (à titre d'option de codeur) peut augmenter "s" (comme déterminé au § 23.8.3.3) de "**MULTIPLE OF**" unités (sous réserve d'éventuelles restrictions imposées par l'étendue de valeurs d'un quelconque champ "**field-to-be-set**" ou "**field-to-be-used**") si "**ENCODING-SPACE SIZE**" est activé à la valeur "**encoder-option-with-determinant**".

23.8.3.5 S'il y a des bits inutilisés dans l'espace de codage, dans ce cas la spécification "**VALUE-PADDING**" doit être appliquée.

23.8.4 Actions du décodeur

23.8.4.1 Pour tout groupe de propriétés de codage qui est activé, le décodeur doit exécuter les actions de décodage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) préalignement et bourrage;

b) pointeur de début;

c) espace de codage;

d) inversion de l'ordre des bits;

e) bourrage et justification de valeur.

23.8.4.2 Le décodeur doit déterminer la taille de la structure néant, et désigner ces bits dans le codage, mais doit accepter sans notification toute valeur pour ces bits.

23.9 Définition des objets de codage pour les classes de la catégorie des chaînes d'octets

23.9.1 La syntaxe définie

La syntaxe de définition des objets de codage pour les classes de la catégorie des chaînes d'octets est définie comme suit:

```
#OCTETS ::= ENCODING-CLASS {

      -- Spécification de préalignement et de bourrage (voir § 22.2)
      &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &encoding-space-pre-padding        Padding DEFAULT zero,
      &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                         DEFAULT bits:'0'B,

      -- Spécification du pointeur de début (voir § 22.3)
      &start-pointer                     REFERENCE  OPTIONAL,
      &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

      -- Octets codage de valeur
      &value-reversal                    BOOLEAN DEFAULT FALSE,
      &Transforms                        #TRANSFORM ORDERED OPTIONAL,
      &Octets-repetition-encodings       #CONDITIONAL-REPETITION ORDERED OPTIONAL,
      &octets-repetition-encoding        #CONDITIONAL-REPETITION OPTIONAL,

      -- Spécification de pointeur d'identification (voir § 22.9)
      &exhibited-handle                  PrintableString OPTIONAL,
      &Handle-positions                  INTEGER (0..MAX) OPTIONAL,
      &handle-value                      HandleValue DEFAULT tag:any,

      -- Spécification de codage du type confiné (voir § 22.11)
      &Primary-encoding-object-set       #ENCODINGS OPTIONAL,
      &Secondary-encoding-object-set     #ENCODINGS OPTIONAL,
      &over-ride-encoded-by              BOOLEAN DEFAULT FALSE
```

```
} WITH SYNTAX {
      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
      [VALUE-REVERSAL          &value-reversal]
      [TRANSFORMS              &Transforms]
      [REPETITION-ENCODINGS    &Octets-repetition-encodings]
      [REPETITION-ENCODING     &octets-repetition-encoding]
      [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]
      [CONTENTS-ENCODING &Primary-encoding-object-set
                  [COMPLETED BY &Secondary-encoding-object-set]
                  [OVERRIDE &over-ride-encoded-by]]

}
```

23.9.2 Modèle de codage de classes de la catégorie des chaînes d'octets

23.9.2.1 Le modèle de codage de chaîne d'octets est le suivant:

a) l'ordre des octets dans une chaîne d'octets peut être inversé;

b) les octets sont alors considérés comme une répétition d'un octet;

c) il y a une transformée facultative (spécifiée par "**TRANSFORMS**") dans laquelle chaque octet est transformé en une chaîne de bits autodélimitatrice;

d) soit "**REPETITION-ENCODING**" ou "**REPETITION-ENCODINGS**" spécifie la façon dont la répétition d'octet doit être codée.

NOTE – Le seul objectif de permettre "**REPETITION-ENCODING**" ainsi que "**REPETITION-ENCODINGS**" est d'offrir une syntaxe qui ne contient pas de double accolade ("**{{**") dans le cas courant d'un codage conditionnel isolé. L'utilisation de "**REPETITION-ENCODINGS**" lorsqu'il y a un codage conditionnel isolé est déconseillée mais autorisée.

23.9.2.2 Les limites (si présentes) de la classe en cours de codage (une classe de la catégorie des chaînes d'octets) sont applicables au nombre d'octets dans une chaîne d'octets formant chaque valeur abstraite.

23.9.2.3 Considérées comme une répétition d'un octet, ces limites doivent être interprétées comme étant applicables au nombre de répétitions, et peuvent être utilisées dans la spécification des objets de codage de classe **#CONDITIONAL-REPETITION** qui sont utilisés dans la spécification de cet objet de codage.

23.9.3 Finalité et restrictions

23.9.3.1 Cette syntaxe est utilisée afin de définir le début de l'espace de codage pour une classe de la catégorie des chaînes d'octets, le codage des valeurs abstraites associées à cette classe, une déclaration facultative que tous les codages de chaîne d'octets offrent un pointeur d'identification spécifié, une spécification de la façon de coder un type contenu.

23.9.3.2 La classe **#CONDITIONAL-REPETITION** qui est appliquée par cet objet de codage ne doit pas spécifier "**REPLACE**" à moins qu'il ne s'agisse de "**REPLACE STRUCTURE**".

23.9.3.3 Si n'importe lequel des objets de la classe **#CONDITIONAL-REPETITION** contient une clause "**REPLACE STRUCTURE**", dans ce cas tous les objets de la classe **#CONDITIONAL-REPETITION** doivent contenir une clause "**REPLACE STRUCTURE**".

23.9.3.4 S'il y a une clause "**REPLACE STRUCTURE**" dans les objets de codage de la classe **#CONDITIONAL-REPETITION**, dans ce cas aucun autre paramètre ne doit être activé.

23.9.3.5 La première transformée de "**TRANSFORMS**" (si présente) doit avoir une source qui est une chaîne de bits et la dernière transformée doit avoir un résultat qui est une chaîne de bits autodélimitatrice (voir 3.2.41).

23.9.3.6 Il y a erreur de spécification ou d'application ECN si une éventuelle transformée dans la spécification "**TRANSFORMS**" n'est pas réversible pour la valeur abstraite à laquelle elle est appliquée.

23.9.3.7 Exactement une des spécifications "**REPETITION-ENCODING**" et "**REPETITION-ENCODINGS**" doit être activée.

23.9.3.8 Si un objet de codage contenu dans la liste ordonnée "**REPETITION-ENCODINGS**" est défini au moyen de "**IF**", dans ce cas tous les objets de codage précédents de cette liste doivent être définis au moyen de "**IF**".

23.9.3.9 Si "**EXHIBITS HANDLE**" est activé, dans ce cas tous les codages de valeurs de cette classe doivent offrir le pointeur d'identification spécifié.

NOTE – Cela nécessitera en général des restrictions applicables aux valeurs abstraites du type associé.

23.9.3.10 Si "**EXHIBITS HANDLE**" est activé, dans ce cas "**ALIGNED TO**" ne doit être activé dans aucune des spécifications "**REPETITION-ENCODING(S)**".

23.9.4 Actions du codeur

23.9.4.1 Pour tout groupe de propriétés de codage qui est activé, le codeur doit exécuter les actions de codage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) préalignement et bourrage;

b) pointeur de début;

c) codage de valeur comme spécifié ci-dessous;

d) codage de répétition comme indiqué par la première spécification "**REPETITION-ENCODING(S)**" dont la condition est satisfaite;

e) pointeur d'identification;

f) codage de type contenu.

23.9.4.2 Pour le codage de valeur, le codeur doit:

a) inverser l'ordre des octets dans la valeur abstraite d'une chaîne d'octets entière si "**VALUE-REVERSAL**" est activé à **TRUE**;

b) traiter la valeur d'une chaîne d'octets comme une répétition d'octet;

c) appliquer la spécification "**TRANSFORMS**" (si présente) à chaque octet afin de produire une répétition de chaîne de bits.

NOTE – S'il n'y a pas de transformées, chaque octet forme une chaîne de bits.

d) coder la répétition par l'application de la première spécification "**REPETITION-ENCODING(S)**" dont la condition est satisfaite.

23.9.4.3 Il y a erreur de spécification ECN s'il n'y a pas de spécification "**REPETITION-ENCODING(S)**" dont la condition est satisfaite.

23.9.5 Actions du décodeur

23.9.5.1 Pour tout groupe de propriétés de codage qui est activé, le décodeur doit exécuter les actions de décodage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) préalignement et bourrage;

b) pointeur de début;

c) le décodage de valeur (voir § 23.9.5.2);

d) décodage du type contenu.

23.9.5.2 Le décodeur doit inverser la valeur de "**TRANSFORMS**" (si présente) afin de rétablir les octets originaux.

23.9.5.3 Si "**VALUE-REVERSAL**" est activé à **TRUE**, dans ce cas l'ordre final des octets dans une valeur abstraite de chaîne d'octets doit être inversé.

23.10 Définition des objets de codage pour les classes de la catégorie des offres d'options

23.10.1 La syntaxe définie

La syntaxe de définition des objets de codage pour les classes de la catégorie des offres d'options est définie comme suit:

```
#OPTIONAL ::= ENCODING-CLASS {

      -- Spécification de remplacement limité aux structures (voir § 22.1)
      &#Replacement-structure
            OPTIONAL,
      &replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,
```

```
        -- Spécification de préalignement et de bourrage (voir § 22.2)
        &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &encoding-space-pre-padding        Padding DEFAULT zero,
        &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                           DEFAULT bits:'0'B,

        -- Spécification du pointeur de début (voir § 22.3)
        &start-pointer                     REFERENCE  OPTIONAL,
        &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

        -- Détermination de l'offre d'options (voir § 22.5)
        &optionality-determination         OptionalityDetermination
                                           DEFAULT field-to-be-set,
        &optionality-reference             REFERENCE OPTIONAL,
        &Encoder-transforms                #TRANSFORM ORDERED OPTIONAL,
        &Decoder-transforms                #TRANSFORM ORDERED OPTIONAL,
        &handle-id                         PrintableString
                                           DEFAULT "default-handle"

} WITH SYNTAX {
      [REPLACE
            [STRUCTURE]
            WITH &#Replacement-structure
                  [ENCODED BY &replacement-structure-encoding-object]]
      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]

      PRESENCE
            [DETERMINED BY &optionality-determination
                  [HANDLE &handle-id]]
            [USING &optionality-reference
                  [ENCODER-TRANSFORMS &Encoder-transforms]
                  [DECODER-TRANSFORMS &Decoder-transforms]]

}
```

23.10.2 Finalité et restrictions

23.10.2.1 Cette syntaxe est utilisée afin de définir le codage d'une classe de la catégorie des offres d'options.

23.10.2.2 Si "**REPLACE STRUCTURE**" est activé, dans ce cas aucun autre groupe de propriétés de codage ne doit être activé.

23.10.3 Actions du codeur

23.10.3.1 Pour tout groupe de propriétés de codage qui est activé, le codeur doit exécuter les actions de codage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) remplacement (voir § 23.10.3.2);

b) préalignement et bourrage;

c) pointeur de début;

d) détermination de l'offre d'options.

23.10.3.2 Si "**REPLACE STRUCTURE**" est activé, alors le composant entier (dont toute classe de la catégorie des étiquettes, mais à l'exclusion des classes de la catégorie des offres d'options) est fourni en tant que paramètre réel pour la structure de remplacement, qui devient un composant obligatoire.

23.10.4 Actions du décodeur

23.10.4.1 Pour tout groupe de propriétés de codage qui est activé, le décodeur doit exécuter les actions de décodage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) préalignement et bourrage;

b) pointeur de début;

c) détermination de l'offre d'options.

23.11 Définition des objets de codage pour les classes de la catégorie des bourrages

23.11.1 La syntaxe définie

La syntaxe de définition des objets de codage pour les classes de la catégorie des bourrages est définie comme suit:

```
#PAD ::= ENCODING-CLASS {

      -- Spécification de remplacement limité aux structures (voir § 22.1)
      &#Replacement-structure
            OPTIONAL,
      &replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,

      -- Spécification de préalignement et de bourrage (voir § 22.2)
      &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &encoding-space-pre-padding        Padding DEFAULT zero,
      &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                         DEFAULT bits:'0'B,

      -- Spécification du pointeur de début (voir § 22.3)
      &start-pointer                     REFERENCE  OPTIONAL,
      &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

      -- Spécification de l'espace de codage (voir § 22.4)
      &encoding-space-size               EncodingSpaceSize
                                         DEFAULT self-delimiting-values,
      &encoding-space-unit               Unit (ALL EXCEPT repetitions)
                                         DEFAULT bit,
      &encoding-space-determination      EncodingSpaceDetermination
                                         DEFAULT field-to-be-set,
      &encoding-space-reference          REFERENCE OPTIONAL,
      &Encoder-transforms                #TRANSFORM ORDERED OPTIONAL,
      &Decoder-transforms                #TRANSFORM ORDERED OPTIONAL,

      -- Codage de valeur
      &pad-pattern                       Pattern (ALL EXCEPT different:any)
                                         DEFAULT bits:''B,

      -- Spécification de pointeur d'identification (voir § 22.9)
      &exhibited-handle                  PrintableString OPTIONAL,
      &Handle-positions                  INTEGER (0..MAX) OPTIONAL,
      &handle-value                      HandleValue DEFAULT tag:any,

      -- Spécification de l'inversion de l'ordre des bits (voir § 22.12)
      &bit-reversal                      ReversalSpecification
                                         DEFAULT no-reversal

} WITH SYNTAX {
      [REPLACE
            [STRUCTURE]
            WITH &#Replacement-structure
                  [ENCODED BY &replacement-structure-encoding-object]]
      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
```

```
        [START-POINTER   &start-pointer
              [MULTIPLE OF     &start-pointer-unit]
              [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]

        ENCODING-SPACE
              [SIZE &encoding-space-size
                    [MULTIPLE OF &encoding-space-unit]]
              [DETERMINED BY &encoding-space-determination]
              [USING &encoding-space-reference
                    [ENCODER-TRANSFORMS &Encoder-transforms]
                    [DECODER-TRANSFORMS &Decoder-transforms]]

        [PAD-PATTERN &pad-pattern]
        [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
              [AS &handle-value]]
        [BIT-REVERSAL &bit-reversal]
}
```

23.11.2 Finalité et restrictions

23.11.2.1 Cette syntaxe est utilisée afin de définir le codage d'une classe de la catégorie des bourrages.

23.11.2.2 Si "**ENCODING-SPACE SIZE**" est positif, "**PAD-PATTERN**" ne doit pas être de longueur nulle, et est reproduite et tronquée afin de remplir l'espace de codage.

23.11.2.3 Si "**REPLACE STRUCTURE**" est activé, dans ce cas aucun autre groupe de propriétés de codage ne doit être activé.

23.11.3 Actions du codeur

23.11.3.1 Pour tout groupe de propriétés de codage qui est activé, le codeur doit exécuter les actions de codage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) remplacement;

b) préalignement et bourrage;

c) pointeur de début;

d) espace de codage;

e) codage de valeur (voir ci-dessous);

f) pointeur d'identification;

g) inversion de l'ordre des bits.

23.11.3.2 Si "**ENCODING-SPACE SIZE**" est positif, la valeur doit être celle de "**PAD-PATTERN**", reproduite et tronquée afin de remplir l'espace de codage.

23.11.3.3 Si "**ENCODING-SPACE SIZE**" est "**fixed-to-max**", ou est "**variable-with-determinant**" ou est "**encoder-option-with-determinant**", dans ce cas l'espace de codage doit être le plus petit nombre de "**MULTIPLE OF**" unités qui est supérieur à la taille de "**PAD-PATTERN**" ("s", par exemple), et la valeur "**PAD-PATTERN**" doit ensuite être reproduite et tronquée afin de remplir cet espace (mais voir § 22.11.3.4).

NOTE – Ce sera un espace de codage vide si la valeur de "**PAD-PATTERN**" est néant.

23.11.3.4 Un codeur (à titre d'option de codeur) peut augmenter "s" (telle que déterminée au § 23.11.3.3) de "**MULTIPLE OF**" unités (sous réserve d'éventuelles restrictions imposées par l'étendue de valeurs d'un quelconque champ "**field-to-be-set**" ou "**field-to-be-used**") si la production "**ENCODING-SPACE SIZE**" est activée à la valeur "**encoder-option-with-determinant**".

23.11.4 Actions du décodeur

23.11.4.1 Pour tout groupe de propriétés de codage qui est activé, le décodeur doit exécuter les actions de décodage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) préalignement et bourrage;

b) pointeur de début;

c) inversion de l'ordre des bits;

d) espace de codage.

23.11.4.2 Le décodeur doit déterminer la taille de la valeur de codage du bourrage, et désigner ces bits dans le codage, mais doit accepter sans notification toute valeur pour ces bits.

23.12 Définition des objets de codage pour les classes de la catégorie des répétitions

23.12.1 La syntaxe définie

La syntaxe de définition des objets de codage pour les classes de la catégorie des répétitions est définie comme suit:

```
#REPETITION ::= ENCODING-CLASS {

      -- Codage de répétition
      &Repetition-encodings      #CONDITIONAL-REPETITION ORDERED OPTIONAL,
      &repetition-encoding       #CONDITIONAL-REPETITION OPTIONAL

} WITH SYNTAX {
      [REPETITION-ENCODINGS &Repetition-encodings]
      [REPETITION-ENCODING &repetition-encoding]

}
```

23.12.2 Finalité et restrictions

23.12.2.1 Cette syntaxe est utilisée afin de définir le codage d'une classe de la catégorie des répétitions au moyen de la spécification d'un ou de plusieurs codages de la classe **#CONDITIONAL-REPETITION**.

23.12.2.2 Exactement une des spécifications "**REPETITION-ENCODING**" et "**REPETITION-ENCODINGS**" doit être activée.

NOTE – Le seul objectif de permettre "**REPETITION-ENCODING**" ainsi que "**REPETITION-ENCODINGS**" est d'offrir une syntaxe qui ne contient pas de double accolade ("**{{**") dans le cas courant d'un seul objet de codage. L'utilisation de "**REPETITION-ENCODINGS**" lorsqu'il y a un seul objet de codage est déconseillée mais autorisée.

23.12.2.3 Si un objet de codage dans la liste ordonnée "**REPETITION-ENCODINGS**" est défini au moyen de "**IF**", dans ce cas tous les objets de codage précédents de cette liste doivent être définis au moyen de "**IF**".

23.12.3 Actions du codeur

23.12.3.1 Le codeur doit sélectionner et appliquer le premier objet de codage de la classe **#CONDITIONAL-REPETITION** objet de codage contenu dans "**ENCODING(S)**" dont les conditions sont satisfaites. Il y a erreur de spécification ECN si aucun des codages conditionnels n'a des conditions qui sont satisfaites.

NOTE – Il serait inhabituel mais non interdit qu'il y ait des objets de codage de la classe **#CONDITIONAL-REPETITION** qui ne pourraient jamais être utilisés parce que les conditions relatives à l'utilisation d'objets de codage antérieurs seraient toujours satisfaites.

23.12.4 Actions du décodeur

23.12.4.1 Le décodeur doit sélectionner et utiliser le premier objet de codage de la classe **#CONDITIONAL-REPETITION** contenu dans "**ENCODING(S)**" dont les conditions sont satisfaites.

23.13 Définition des objets de codage pour la classe #CONDITIONAL-REPETITION

23.13.1 La syntaxe définie

La syntaxe de définition des objets de codage pour la classe **#CONDITIONAL-REPETITION** est définie comme suit:

```
#CONDITIONAL-REPETITION ::= ENCODING-CLASS {

      -- Condition (voir § 21.12)
      &size-range-condition              SizeRangeCondition  OPTIONAL,

      -- Spécification de remplacement de structure ou de composant (voir § 22.1)
      &#Replacement-structure
            OPTIONAL,
      &replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,
      &#Head-end-structure                                               OPTIONAL,

      -- Spécification de préalignement et de bourrage (voir § 22.2)
      &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &encoding-space-pre-padding        Padding DEFAULT zero,
      &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
```

```
                                        DEFAULT bits:'0'B,

      -- Spécification du pointeur de début (voir § 22.3)
      &start-pointer                    REFERENCE  OPTIONAL,
      &start-pointer-unit               Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL,

      -- Spécification de l'espace de répétition (voir § 22.7)
      &repetition-space-size            EncodingSpaceSize
                                        DEFAULT self-delimiting-values,
      &repetition-space-unit            Unit
                                        DEFAULT bit,
      &repetition-space-determination   RepetitionSpaceDetermination
                                        DEFAULT field-to-be-set,
      &main-reference                   REFERENCE OPTIONAL,
      &Encoder-transforms               #TRANSFORM ORDERED OPTIONAL,
      &Decoder-transforms               #TRANSFORM ORDERED OPTIONAL,
      &handle-id                        PrintableString
                                        DEFAULT "default-handle",
      &termination-pattern              Non-Null-Pattern (ALL EXCEPT
                                        different:any) DEFAULT bits '0'B,

      -- Classe &repetition-alignment
      (d'alignement de répétition)      ENUMERATED {none, aligned}
                                        DEFAULT none,

      -- Bourrage et justification de valeur (voir § 22.8)
      &value-justification              Justification DEFAULT right:0,
      &value-pre-padding                Padding DEFAULT zero,
      &value-pre-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
      &value-post-padding               Padding DEFAULT zero,
      &value-post-pattern               Non-Null-Pattern DEFAULT bits:'0'B,
      &unused-bits-determination        UnusedBitsDetermination
                                        DEFAULT field-to-be-set,
      &unused-bits-reference            REFERENCE OPTIONAL,
      &Unused-bits-encoder-transforms   #TRANSFORM ORDERED OPTIONAL,
      &Unused-bits-decoder-transforms   #TRANSFORM ORDERED OPTIONAL,

      -- Spécification de pointeur d'identification (voir § 22.9)
      &exhibited-handle                 PrintableString OPTIONAL,
      &Handle-positions                 INTEGER (0..MAX) OPTIONAL,
      &handle-value                     HandleValue DEFAULT tag:any,

      -- Spécification de l'inversion de l'ordre des bits (voir § 22.12)
      &bit-reversal                     ReversalSpecification
                                        DEFAULT no-reversal

} WITH SYNTAX {
      [IF &size-range-condition] [ELSE]
      [REPLACE
            [STRUCTURE]
            [COMPONENT]
            [ALL COMPONENTS]
           WITH &Replacement-structure
            [ENCODED BY &replacement-structure-encoding-object
                  [INSERT AT HEAD &#Head-end-structure]]]
      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
      REPETITION-SPACE
            [SIZE &repetition-space-size
                  [MULTIPLE OF &repetition-space-unit]]
            [DETERMINED BY &repetition-space-determination
                  [HANDLE &handle-id]]
```

```
                [USING &main-reference
                      [ENCODER-TRANSFORMS &Encoder-transforms]
                      [DECODER-TRANSFORMS &Decoder-transforms]]
                [PATTERN &termination-pattern]
          [ALIGNMENT  &repetition-alignment]
          [VALUE-PADDING
                [JUSTIFIED &value-justification]
                [PRE-PADDING &value-pre-padding
                      [PATTERN &value-pre-pattern]]
                [POST-PADDING &value-post-padding
                      [PATTERN &value-post-pattern]]
                [UNUSED BITS
                      [DETERMINED BY &unused-bits-determination]
                      [USING &unused-bits-reference
                            [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                            [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
          [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
                [AS &handle-value]]
          [BIT-REVERSAL &bit-reversal]

    }
```

23.13.2 Finalité et restrictions

23.13.2.1 Cette syntaxe est utilisée afin de définir le codage d'une classe de la catégorie des répétitions sous réserve de la réalisation d'une condition sur la base des limites de la répétition (utilisation de "**IF**"). Elle autorise également la spécification qu'il n'y a pas de condition. L'utilisation de "**ELSE**", ou l'omission d'aussi bien "**IF**" que "**ELSE**" spécifie qu'il n'y a pas de condition.

23.13.2.2 Au plus un des mots "**IF**" et "**ELSE**" doit être présent.

23.13.2.3 Si "**REPLACE STRUCTURE**" est activé, dans ce cas aucun autre groupe de propriétés de codage ne doit être activé.

23.13.2.4 Si "**EXHIBITS HANDLE**" est activé, cela indique que tous les codages de cette classe offrent le pointeur d'identification spécifié (voir également § 22.9.2.4).

23.13.2.5 "**REPETITION-SPACE SIZE**" ne doit pas être "**fixed-to-max**".

23.13.2.6 Si "**REPETITION-SPACE SIZE**" est à "**self-delimiting-values**", et si "**MULTIPLE OF**" est à "**repetitions**", dans ce cas le nombre de répétitions doit être contraint par des limites à une seule valeur.

23.13.2.7 S'il y a des bits inutilisés dans l'espace de codage, dans ce cas "**VALUE-PADDING**" doit être activé.

23.13.3 Actions du codeur

23.13.3.1 Pour tout groupe de propriétés de codage qui est activé, le codeur doit exécuter les actions de codage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) remplacement;

b) préalignement et bourrage;

c) pointeur de début;

d) espace de répétition;

e) codage de répétition (voir § 23.13.3.4);

f) bourrage et justification de valeur;

g) pointeur d'identification;

h) inversion de l'ordre des bits.

23.13.3.2 Si "**ALIGNMENT**" est activé à "**aligned**", dans ce cas les réglages de préalignement et de bourrage doivent être utilisés afin de préaligner chaque codage du composant.

NOTE – Cela est effectué avant un éventuel préalignement spécifié par le composant.

23.13.3.3 Les codages complets des composants (avec éventuel préalignement cependant spécifié) doivent être concaténés afin de former les bits pour la valeur de la répétition.

23.13.3.4 Si la production "**REPETITION-SPACE SIZE**" est à la valeur "**variable-with-determinant**" ou "**encoder-option-with-determinant**", dans ce cas la taille doit être le plus petit multiple de "**MULTIPLE OF**" unités ("s", par exemple) qui contiendra la valeur de la répétition (mais voir § 23.13.3.5).

23.13.3.5 Un codeur (à titre d'option de codeur) peut augmenter "s" (telle que déterminée au § 23.13.3.4) de "**MULTIPLE OF**" unités (sous réserve d'éventuelles restrictions imposées par l'étendue de valeurs d'un quelconque champ "**field-to-be-set**" ou "**field-to-be-used**") si "**ENCODING-SPACE SIZE**" est activé à "**encoder-option-with-determinant**".

23.13.3.6 La valeur de répétition est alors placée dans l'espace de codage, au moyen de "**VALUE-PADDING**" s'il y a des bits inutilisés.

23.13.4 Actions du décodeur

23.13.4.1 Pour tout groupe de propriétés de codage qui est activé, le décodeur doit exécuter les actions de décodage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) préalignement et bourrage;

b) pointeur de début;

c) espace de répétition;

d) inversion de l'ordre des bits;

e) bourrage et justification de valeur;

f) décodage de répétition (voir § 23.13.4.2).

23.13.4.2 Chaque répétition doit être extraite et décodée conformément à la spécification de codage du composant de la classe de répétition.

23.14 Définition des objets de codage pour les classes de la catégorie des étiquettes

23.14.1 La syntaxe définie

La syntaxe de définition des objets de codage pour les classes de la catégorie des étiquettes est définie comme suit:

```
#TAG ::= ENCODING-CLASS {

    -- Spécification de remplacement limité aux structures (voir § 22.1)
    &#Replacement-structure
         OPTIONAL,
    &replacement-structure-encoding-object  &#Replacement-structure   OPTIONAL,

    -- Spécification de préalignement et de bourrage (voir § 22.2)
    &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding        Padding DEFAULT zero,
    &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                       DEFAULT bits:'0'B,

    -- Spécification du pointeur de début (voir § 22.3)
    &start-pointer                     REFERENCE  OPTIONAL,
    &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

    -- Spécification de l'espace de codage (voir § 22.4)
    &encoding-space-size               EncodingSpaceSize
                                       DEFAULT self-delimiting-values,
    &encoding-space-unit               Unit (ALL EXCEPT repetitions)
                                       DEFAULT bit,
    &encoding-space-determination      EncodingSpaceDetermination
                                       DEFAULT field-to-be-set,
    &encoding-space-reference          REFERENCE OPTIONAL,
    &Encoder-transforms                #TRANSFORM ORDERED OPTIONAL,
    &Decoder-transforms                #TRANSFORM ORDERED OPTIONAL,

    -- Bourrage et justification de valeur (voir § 22.8)
    &value-justification               Justification DEFAULT right:0,
    &value-pre-padding                 Padding DEFAULT zero,
    &value-pre-pattern                 Non-Null-Pattern DEFAULT bits:'0'B,
    &value-post-padding                Padding DEFAULT zero,
    &value-post-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
    &unused-bits-determination         UnusedBitsDetermination
                                       DEFAULT field-to-be-set,
```

```
        &unused-bits-reference              REFERENCE OPTIONAL,
        &Unused-bits-encoder-transforms     #TRANSFORM ORDERED OPTIONAL,
        &Unused-bits-decoder-transforms     #TRANSFORM ORDERED OPTIONAL,

        -- Spécification de pointeur d'identification (voir § 22.9)
        &exhibited-handle                   PrintableString OPTIONAL,
        &Handle-positions                   INTEGER (0..MAX) OPTIONAL,
        &handle-value                       HandleValue DEFAULT tag:any,

        -- Spécification de l'inversion de l'ordre des bits (voir § 22.12)
        &bit-reversal                       ReversalSpecification
                                            DEFAULT no-reversal

    } WITH SYNTAX {
        [REPLACE
            [STRUCTURE]
            WITH &#Replacement-structure
                [ENCODED BY &replacement-structure-encoding-object]]
        [ALIGNED TO
            [NEXT]
            [ANY]
                &encoding-space-pre-alignment-unit
                [PADDING &encoding-space-pre-padding
                    [PATTERN &encoding-space-pre-pattern]]]
        [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]

        ENCODING-SPACE
            [SIZE &encoding-space-size
                [MULTIPLE OF &encoding-space-unit]]
            [DETERMINED BY &encoding-space-determination]
            [USING &encoding-space-reference
                [ENCODER-TRANSFORMS &Encoder-transforms]
                [DECODER-TRANSFORMS &Decoder-transforms]]

        [VALUE-PADDING
            [JUSTIFIED &value-justification]
            [PRE-PADDING &value-pre-padding
                [PATTERN &value-pre-pattern]]
            [POST-PADDING &value-post-padding
                [PATTERN &value-post-pattern]]
            [UNUSED BITS
                [DETERMINED BY &unused-bits-determination]
                [USING &unused-bits-reference
                    [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                    [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
        [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]
        [BIT-REVERSAL &bit-reversal]

    }
```

23.14.2 Finalité et restrictions

23.14.2.1 Cette syntaxe est utilisée afin de définir le codage d'une classe de la catégorie des étiquettes.

23.14.2.2 Si "**`REPLACE STRUCTURE`**" est activé, dans ce cas aucune autre spécification ne doit être activée.

23.14.2.3 La production "**`ENCODING-SPACE SIZE`**" ne doit pas être "**`fixed-to-max`**" ou "**`self-delimiting-values`**".

23.14.3 Actions du codeur

23.14.3.1 Pour tout groupe de propriétés de codage qui est activé, le codeur doit exécuter les actions de codage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) remplacement;

b) préalignement et bourrage;

c) pointeur de début;

d) espace de codage;

e) codage de valeur (voir § 23.14.3.3);

f) bourrage et justification de valeur;

g) pointeur d'identification;

h) inversion de l'ordre des bits.

23.14.3.2 Le codeur doit déterminer le nombre minimal de bits "n" nécessaire pour coder le numéro d'étiquette en tant que plus petite valeur de "n" telle que $2^n - 1$ soit supérieur ou égal au numéro d'étiquette. Si "n" est zéro, sa valeur doit être augmentée jusqu'à 1.

23.14.3.3 Le codage doit être un codage d'entier positif. La spécification d'un codage d'entier positif est donnée dans la Rec. UIT-T X.690 | ISO/CEI 8825-1, § 8.3.2 et 8.3.3.

23.14.3.4 Un codeur doit détecter une erreur de spécification ECN si un numéro d'étiquette doit être codé dans un nombre de bits qui est insuffisant, comme spécifié ci-dessus.

23.14.3.5 Si "**ENCODING-SPACE SIZE**" est un entier positif, dans ce cas sa taille en bits est calculée en tant que "**SIZE**" multipliée par "**MULTIPLE OF**" unités. Si "**VALUE-PADDING**" n'est pas activé, dans ce cas ce doit être le nombre de bits "n" que le numéro d'étiquette doit y coder et il n'y a pas de bits inutilisés. Si "**VALUE-PADDING**" est activé, dans ce cas le nombre de bits que le numéro d'étiquette doit y coder est réduit par la valeur entière "m" spécifiée pour "**JUSTIFIED**", et il y aura "m" bits inutilisés.

23.14.3.6 Si "**ENCODING-SPACE SIZE**" est "**variable-with-determinant**" ou "**encoder-option-with-determinant**", dans ce cas le codeur doit déterminer le nombre minimal de "**MULTIPLE OF**" unités qui possède un nombre de bits suffisant pour coder le numéro d'étiquette ("s", par exemple), et doit procéder (comme spécifié ci-dessus) comme si "**SIZE**" était un entier positif mis à cette valeur (mais voir § 23.14.3.7).

23.14.3.7 Un codeur (à titre d'option de codeur) peut augmenter "s" (telle que déterminée au § 23.14.3.6) de "**MULTIPLE OF**" unités (sous réserve d'éventuelles restrictions imposées par l'étendue de valeurs d'un quelconque champ "**field-to-be-set**" ou "**field-to-be-used**") si "**ENCODING-SPACE SIZE**" est activé à "**encoder-option-with-determinant**".

23.14.4 Actions du décodeur

23.14.4.1 Pour tout groupe de propriétés de codage qui est activé, le décodeur doit exécuter les actions de décodage spécifiées dans l'article 22, dans l'ordre suivant et conformément à la définition d'objet de codage:

a) préalignement et bourrage;

b) pointeur de début;

c) espace de codage;

d) inversion de l'ordre des bits;

e) bourrage et justification de valeur;

f) décodage de valeur.

23.14.4.2 Le décodeur doit reconstituer le numéro d'étiquette à partir des bits utilisés pour le coder, le décodage s'effectuant à partir d'un codage d'entier positif.

23.15 Définition des objets de codage pour classes d'autres catégories

Dans cette version de la présente Recommandation | Norme internationale, il n'y a pas de syntaxe définie pour les classes appartenant aux catégories suivantes:

objectidentifier
opentype
real

24 Spécification de syntaxe définie pour la classe de codage #TRANSFORM

24.1 Résumé des propriétés de codage et syntaxe définie

24.1.1 La syntaxe de définition des objets de codage pour la classe **#TRANSFORM** doit être la suivante:

```
#TRANSFORM ::= ENCODING-CLASS {

      -- entier vers entier (voir § 24.3)
      &int-to-int            CHOICE
                             {increment       INTEGER (1..MAX),
                             decrement        INTEGER (1..MAX),
                             multiply         INTEGER (2..MAX),
                             divide           INTEGER (2..MAX),
                             negate           ENUMERATED{value},
                             modulo           INTEGER (2..MAX),
                             subtract         ENUMERATED{lower-bound}
                             } OPTIONAL,

      -- booléen vers booléen (voir § 24.4)
      &bool-to-bool          CHOICE
                             {logical        ENUMERATED{not}}
                             DEFAULT logical:not,

      -- booléen vers entier (voir § 24.5)
      &bool-to-int           ENUMERATED {true-zero, true-one}
                             DEFAULT true-one,

      -- entier vers booléen (voir § 24.6)
      &int-to-bool                ENUMERATED {zero-true, zero-false}
                                  DEFAULT zero-false,
      &Int-to-bool-true-is        INTEGER OPTIONAL,
      &Int-to-bool-false-is       INTEGER OPTIONAL,

      -- entier vers caractères (voir § 24.7)
      &int-to-chars-size          ResultSize DEFAULT variable,
      &int-to-chars-plus          BOOLEAN DEFAULT FALSE,
      &int-to-chars-pad           ENUMERATED
                                  {spaces, zeros} DEFAULT zeros,

      -- entier vers bits (voir § 24.8)
      &int-to-bits-encoded-as     ENUMERATED
                                  {positive-int, twos-complement}
                                  DEFAULT twos-complement,
      &int-to-bits-unit           Unit (1..MAX) DEFAULT bit,
      &int-to-bits-size           ResultSize DEFAULT variable,

      -- bits vers entier (voir § 24.9)
      &bits-to-int-decoded-assuming     ENUMERATED
                                        {positive-int, twos-complement}
                                        DEFAULT twos-complement,

      -- caractères vers bits (voir § 24.10)
      &char-to-bits-encoded-as    ENUMERATED
                                  {iso10646, compact, mapped}
                                  DEFAULT compact,
      &Char-to-bits-chars         UniversalString (SIZE(1))
                                  ORDERED OPTIONAL,
      &Char-to-bits-values        BIT STRING ORDERED OPTIONAL,
      &char-to-bits-unit          Unit (1..MAX) DEFAULT bit,
      &char-to-bits-size          ResultSize DEFAULT variable,

      -- bits vers caractères (voir § 24.11)
      &bits-to-char-decoded-assuming    ENUMERATED
                                        {iso10646, mapped}
                                        DEFAULT iso10646,
      &Bits-to-char-values              BIT STRING ORDERED OPTIONAL,
      &Bits-to-char-chars               UniversalString (SIZE(1))
                                        ORDERED OPTIONAL,
```

```
      -- bits vers bits (voir § 24.12)
      &bit-to-bits-one            Non-Null-Pattern DEFAULT bits:'1'B,
      &bit-to-bits-zero           Non-Null-Pattern DEFAULT bits:'0'B,

      -- bits vers bits (voir § 24.13)
      &Source-values              BIT STRING ORDERED,
      &Result-values              BIT STRING ORDERED,

      -- caractères vers caractères composites (voir § 24.14)
      -- Il n'y a pas de propriétés de codage pour cette transformation

      -- bits vers bits composites (voir § 24.15)
      &bits-to-composite-bits-unit      Unit (1..MAX) DEFAULT bit

      -- octets vers bits composites (voir § 24.16)
      -- Il n'y a pas de propriétés de codage pour cette transformation

      -- caractères composite vers caractères (voir § 24.17)
      -- Il n'y a pas de propriétés de codage pour cette transformation

      -- bits composites vers bits (voir § 24.18)
      -- Il n'y a pas de propriétés de codage pour cette transformation

      -- bits composites vers octets (voir 24.19)
      -- Il n'y a pas de propriétés de codage pour cette transformation

} WITH SYNTAX {

      -- Only one of the following clauses can be used.

      [INT-TO-INT &int-to-int]

      [BOOL-TO-BOOL [AS &bool-to-bool]]

      [BOOL-TO-INT AS &bool-to-int]

      [INT-TO-BOOL
            [AS &int-to-bool]
            [TRUE-IS &Int-to-bool-true-is]
            [FALSE-IS &Int-to-bool-false-is]]

      [INT-TO-CHARS
            [SIZE &int-to-chars-size]
            [PLUS-SIGN &int-to-chars-plus]
            [PADDING &int-to-chars-pad]]

      [INT-TO-BITS
            [AS &int-to-bits-encoded-as]
            [SIZE &int-to-bits-size]
            [MULTIPLE OF &int-to-bits-unit]]

      [BITS-TO-INT
            [AS &bits-to-int-decoded-assuming]]

      [CHAR-TO-BITS
            [AS &char-to-bits-encoded-as]
            [CHAR-LIST &Char-to-bits-chars]
            [BITS-LIST &Char-to-bits-values]
            [SIZE &char-to-bits-size]
            [MULTIPLE OF &char-to-bits-unit]]

      [BITS-TO-CHAR
            [AS &bits-to-char-decoded-assuming]
            [BITS-LIST &Bits-to-char-values]
            [CHAR-LIST &Bits-to-char-chars]]

      [BIT-TO-BITS
            [ZERO-PATTERN &bit-to-bits-zero]
            [ONE-PATTERN &bit-to-bits-one]]
```

```
        [BITS-TO-BITS
              SOURCE-LIST &Source-values
              RESULT-LIST &Result-values]

        [CHARS-TO-COMPOSITE-CHAR]

        [BITS-TO-COMPOSITE-BITS
              [UNIT &bits-to-composite-bits-unit]]

        [OCTETS-TO-COMPOSITE-BITS]

        [COMPOSITE-CHAR-TO-CHARS]

        [COMPOSITE-BITS-TO-BITS]

        [COMPOSITE-BITS-TO-OCTETS]

}
```

24.2 Source et cible des transformées

24.2.1 La classe de codage **#TRANSFORM** autorise la spécification de procédures qui transforment des valeurs abstraites d'entrée (la source) en valeurs abstraites de sortie de type identique ou différent (le résultat). Elle autorise également la spécification de procédures qui mappent une source en chaîne de caractères, d'octets ou de bits dans un composite de transformée, et qui mappent un composite de transformée (dont les valeurs sont un caractère isolé, un octet isolé, ou des chaînes de bits de longueur unitaire fixe) dans une valeur abstraite (une chaîne de caractères, une chaîne d'octets, ou une chaîne de bits). La source est le résultat d'une transformation précédente, ou est obtenue à partir d'une classe source (voir § 19.4). Le résultat est soit la source pour une transformée suivante, ou une nouvelle association à une classe cible (voir § 19.4).

NOTE – L'article 23 utilise également des transformées dont la source est un bit isolé et un caractère isolé.

24.2.2 Ces transformées sont utilisées dans la définition de mappages de valeur et dans la définition d'objets de codage pour classes de codage dans le groupe catégoriel des champs binaires (voir les articles 20 à 23).

24.2.3 La source et le résultat sont indiqués par les mots ("**INT-TO-INT**", "**BOOL-TO-BOOL**", etc.) dans la spécification d'un objet de codage de la classe **#TRANSFORM**, et sont définis dans le texte associé.

24.2.4 Les § 24.2.4.1 à 24.2.4.3 spécifient des règles permettant d'utiliser des transformées en succession, et concernant les classes sources et cibles d'une liste de transformées.

24.2.4.1 Lorsque des objets de codage de la classe **#TRANSFORM** sont spécifiés dans une liste ordonnée, la source d'un objet de codage suivant de la classe **#TRANSFORM** doit être le résultat de l'objet de codage précédent de la classe **#TRANSFORM**.

24.2.4.2 Pour le premier et le dernier d'une liste ordonnée des transformées utilisé dans la définition d'objets de codage figurant dans les articles 22 et 23, dans ce texte les paragraphes spécifient la source pour la première transformée et le résultat requis pour la dernière transformée.

24.2.4.3 Pour le premier et le dernier d'une liste ordonnée de transformées utilisée dans la spécification de mappages de valeurs par transformées au § 19.4, le texte du présent paragraphe spécifie une classe source et une classe cible, qui seront toutes deux de la catégorie des chaînes de bits, des booléens, des chaînes de caractères, des entiers ou des chaînes d'octets (voir § 19.4.2). La source requise pour la première transformée et le résultat requis de la dernière transformée (pour chacune de ces catégories) sont spécifiés au § 24.2.7.

24.2.5 Le texte du présent paragraphe spécifie la source d'une transformée et le résultat d'une transformée sous la forme d'un entier, d'un booléen, d'une chaîne de caractères, d'une chaîne de bits, d'un caractère isolé, ou d'un bit isolé (source seulement). La source et le résultat d'une transformée peuvent également être un composite de ces valeurs. Les composites de transformation ne peuvent être produits que par des transformées, et doivent être traités par une autre transformée (la suivante) dans une liste de transformées. Il existe deux groupes de transformées: celles qui sont conçues afin de créer des composites à partir de valeurs abstraites ou de produire une valeur abstraite d'un composite; et celles qui sont conçues afin de transformer des valeurs isolées. Le dernier groupe peut également transformer des composites de ces valeurs, afin de produire un composite dont le résultat est la transformée de chaque élément contenu dans le composite-source.

24.2.6 Une source ou une cible qui est un bit isolé ou un caractère isolé n'apparaît que lorsqu'elle se trouve en sortie ou en entrée de transformées successives, ou comme spécifié dans les articles 22 et 23. La première transformée de la liste ordonnée citée en référence au § 19.4 ne doit pas avoir une source qui est un bit isolé ou un caractère isolé. La

dernière transformée de la liste ordonnée citée en référence au § 19.4 ne doit pas avoir une cible qui est un bit isolé ou un caractère isolé.

24.2.7 Lors de l'utilisation au § 19.4, la source pour la première transformée et la cible pour la dernière transformée doivent être (respectivement) les mêmes que la catégorie de la classe de codage source et que la catégorie de la classe de codage cible, avec les exceptions suivantes. Lorsque la catégorie de la classe de codage source est une chaîne d'octets, la source pour la première transformée doit être une chaîne de bits (chaque valeur de chaîne d'octets étant traitée comme une valeur de chaîne de bits). Lorsque la dernière transformée est "**BITS-TO-BITS**" avec "**MULTIPLE OF**" mis à 8, la classe cible peut être une chaîne d'octets.

24.2.8 Les paragraphes suivants spécifient les conditions applicables aux valeurs abstraites de la source qui permettent qu'une transformée soit définie comme réversible. Il y a erreur de notation ECN ou d'application si de telles valeurs sont fournies à une transformée qui doit être réversible, et les codeurs ne doivent pas produire de codages pour de telles valeurs.

24.3 La transformée "int-to-int"

NOTE – Des exemples de cette transformée sont donnés au § D.1.2.2.

24.3.1 La transformée "int-to-int" utilise la propriété de codage suivante:

```
&int-to-int             CHOICE
                        {increment        INTEGER (1..MAX),
                        decrement         INTEGER (1..MAX),
                        multiply          INTEGER (2..MAX),
                        divide            INTEGER (2..MAX),
                        negate            ENUMERATED{value},
                        modulo            INTEGER (2..MAX),
                        subtract          ENUMERATED{lower-bound}
                        } OPTIONAL
```

24.3.2 La syntaxe pour la transformée "int-to-int" doit être la suivante:

```
[INT-TO-INT &int-to-int]
```

24.3.3 Aussi bien la source que le résultat de cette transformée sont un entier ou un entier composite. Il n'existe pas de limites associées au résultat à moins qu'il ne s'agisse de la dernière transformée dans un mappage par transformées (voir § 19.4) (qui implique que ni la source ni la cible ne peuvent être un composite) et la classe cible du mappage par transformées possède des limites. Dans ce cas, il y a erreur de spécification ou d'application ECN si la transformée est appliquée à des valeurs entières de source qui ne s'intègrent pas dans les limites de la classe cible.

24.3.4 Une transformée "int-to-int" est définie par attribution de valeur à "**INT-TO-INT**", ce qui permet à tout objet de codage donné de spécifier précisément une certaine opération arithmétique. Une arithmétique générale peut cependant être définie par l'utilisation d'une liste ordonnée de transformées (ce qui est permis chaque fois que des transformées contenant des entiers sont autorisées).

24.3.5 Les valeurs "**increment:n**", "**decrement:n**", "**multiply:n**", "**negate:n**" ont leur signification mathématique normale.

24.3.6 La valeur "**divide:n**" est définie afin de produire un résultat d'entier qui est la valeur entière la plus proche du résultat mathématique, mais qui ne s'écarte pas plus de zéro que ce résultat. En termes de programmation, "**divide:n**" tronque vers zéro, de sorte que la valeur de -1 avec "**divide:2**" donnera zéro.

24.3.7 La transformée pour la valeur "**modulo:n**" est définie comme suit. Soit "i" la valeur entière originale et soit "**modulo:n**" le type de transformation. Soit "j" le résultat de l'application de "**divide:n**" suivi par "**multiply:n**" à "i". Ensuite, "**modulo:n**" appliqué à "i" est défini comme étant le même que l'application de "**decrement:j**" à "i".

24.3.8 La transformée pour la valeur "**substract:lower-bound**" ne doit être utilisée que la première d'une liste ordonnée de transformées (et donc ne peut jamais être utilisée si la source est un composite). La source doit avoir une limite inférieure.

24.3.9 Chacune de ces transformées est définie comme étant réversible si la source est une seule valeur, non un composite, et si la condition applicable à la valeur abstraite (à laquelle elle est en cours d'application) énumérée dans le Tableau 6 est satisfaite. Elle est également définie comme étant réversible si la source est un composite et si le Tableau 6 spécifie *Toujours réversible* en tant que condition.

Tableau 6 – Inversion de transformées "INT-TO-INT"

Transformée	Condition
`increment:n`	*Toujours réversible*
`decrement:n`	*Toujours réversible*
`multiply:n`	*Toujours réversible*
`divide:n`	La valeur est un multiple de n
`negate:value`	*Toujours réversible*
`modulo:n`	*Jamais réversible*
`subtract:lower-bound`	*Toujours réversible*

24.4 La transformée "bool-to-bool"

24.4.1 La transformée "**BOOL-TO-BOOL**" utilise la propriété de codage suivante:

```
&bool-to-bool          CHOICE
                       {logical        ENUMERATED{not}}
                       DEFAULT logical:not
```

24.4.2 La syntaxe pour la transformée "bool-to-bool" doit être la suivante:

```
[BOOL-TO-BOOL [AS &bool-to-bool]]
```

24.4.3 Aussi bien la source que le résultat de cette transformée sont un booléen ou un composite booléen.

24.4.4 Si la source est un booléen, le résultat est un booléen. Si la source est un composite booléen, le résultat est un composite booléen dans lequel chaque élément de la source a été transformé comme spécifié au § 24.4.5.

24.4.5 Il y a seulement une valeur pour "**BOOL-TO-BOOL**", "**AS logical:not**", qui peut être omise. Cette transformation convertit le booléen **TRUE** en **FALSE**, et vice versa.

24.4.6 Cette transformation est définie comme étant réversible pour toutes les valeurs abstraites.

24.5 La transformée "bool-to-int"

24.5.1 La transformée "bool-to-int" utilise la propriété de codage suivante:

```
&bool-to-int           ENUMERATED {true-zero, true-one}
                       DEFAULT true-one
```

24.5.2 La syntaxe pour la transformée "bool-to-int" doit être la suivante:

```
[BOOL-TO-INT AS &bool-to-int]
```

24.5.3 La source pour cette transformée est un booléen ou un composite booléen et le résultat est un entier ou un entier composite. L'entier résultant (et chaque élément de l'entier composite) a la valeur zéro ou un. Le résultat n'a pas de limites associées.

24.5.4 Si la source est un booléen, le résultat est un entier. Si la source est un composite booléen, le résultat est un entier composite dans lequel chaque élément de la source a été transformé comme spécifié au § 24.5.5.

24.5.5 La valeur "**true-zero**" de "**BOOL-TO-INT**" produit l'entier 0 pour **TRUE** et l'entier 1 pour **FALSE**. La valeur "**true-one**" produit l'entier 1 pour **TRUE** et l'entier 0 pour **FALSE**.

24.5.6 Cette transformation est définie comme étant réversible pour toutes les valeurs abstraites.

24.6 La transformée "int-to-bool"

24.6.1 La transformée "int-to-bool" utilise les propriétés de codage suivantes:

```
&int-to-bool                ENUMERATED {zero-true, zero-false}
                            DEFAULT zero-false,
&Int-to-bool-true-is        INTEGER OPTIONAL,
&Int-to-bool-false-is       INTEGER OPTIONAL
```

24.6.2 La syntaxe pour la transformée "int-to-bool" doit être la suivante:

```
[INT-TO-BOOL
      [AS &int-to-bool]
      [TRUE-IS &Int-to-bool-true-is]
      [FALSE-IS &Int-to-bool-false-is]]
```

24.6.3 La source pour cette transformée est un entier ou un entier composite et le résultat est un booléen ou un composite booléen.

24.6.4 Un seul des mots "**AS**", "**TRUE-IS**" et "**FALSE-IS**" est activé, ou les deux mots "**TRUE-IS**" et "**FALSE-IS**" sont activés (et "**AS**" n'est pas activé), ou aucun n'est activé. Si aucun n'est activé, dans ce cas la valeur par défaut pour "**AS**" est prise en compte.

24.6.5 Si "**AS**" est activé (ou est pris par défaut), dans ce cas la valeur "**zero-true**" produit **TRUE** pour la valeur zéro et **FALSE** pour toutes valeurs différentes de zéro, et la valeur "**zero-false**" produit **FALSE** pour la valeur zéro et **TRUE** pour toutes valeurs différentes de zéro.

24.6.6 Si "**TRUE-IS**" seulement est activé, toutes les valeurs entières pour "**TRUE-IS**" produisent **TRUE** et toutes les autres valeurs entières produisent **FALSE**.

24.6.7 Si "**FALSE-IS**" seulement est activé, toutes les valeurs entières pour "**FALSE-IS**" produisent **FALSE** et toutes les autres valeurs entières produisent **TRUE**.

24.6.8 Si aussi bien "**TRUE-IS**" et "**FALSE-IS**" sont activés, dans ce cas les valeurs entières en "**TRUE-IS**" et "**FALSE-IS**" doivent être disjointes. Dans ce cas, il y a erreur de spécification ou d'application ECN si des valeurs abstraites qui ne sont pas incluses dans "**TRUE-IS**" ou "**FALSE-IS**" sont incluses dans la source, et les codeurs ne doivent pas produire de codages pour de telles valeurs.

24.6.9 Cette transformation est définie comme étant réversible si et seulement si aussi bien "**TRUE-IS**" que "**FALSE-IS**" sont activés, et ces mots spécifient chacun une valeur entière isolée.

24.7 La transformée "int-to-chars"

24.7.1 La transformée "int-to-chars" utilise les propriétés de codage suivantes:

```
&int-to-chars-size          ResultSize DEFAULT variable,
&int-to-chars-plus          BOOLEAN DEFAULT FALSE,
&int-to-chars-pad           ENUMERATED
                            {spaces, zeros} DEFAULT zeros
```

24.7.2 La syntaxe pour la transformée "int-to-chars" doit être la suivante:

```
[INT-TO-CHARS
      [SIZE &int-to-chars-size]
      [PLUS-SIGN &int-to-chars-plus]
      [PADDING &int-to-chars-pad]]
```

24.7.3 La définition du type utilisé dans la transformée "int-to-chars" est la suivante:

```
ResultSize ::= INTEGER {variable(-1), fixed-to-max(0)} (-1..MAX) -- (voir § 21.14)
```

24.7.4 La source pour cette transformée est un entier ou un entier composite, et le résultat est une chaîne de caractères ou une chaîne de caractères composite.

24.7.5 Si la source est un entier, le résultat est une chaîne de caractères. Si la source est un entier composite, le résultat est une chaîne de caractères composite dans laquelle chaque élément de la source a été transformé comme spécifié dans les § 24.7.6 à 24.7.13.

24.7.6 "**SIZE**", "**PLUS-SIGN**", et "**PADDING**" ont tous des valeurs par défaut et peuvent être omis.

24.7.7 "**SIZE**" spécifie soit:

a) une taille fixe en caractères pour la taille résultante (une valeur positive de "**SIZE**"); ou

b) le fait qu'une chaîne de caractères de longueur variable doit être produite (la valeur "**variable**" de "**SIZE**"); ou

c) une taille fixe juste assez grande pour contenir la transformée de toutes les valeurs abstraites contenues dans la classe source (la valeur "**fixed-to-max**" de "**SIZE**").

24.7.8 Le champ "**SIZE**" ne doit pas être mis à "**fixed-to-max**" à moins qu'il ne s'agisse de la première transformée dans un ensemble ordonné, et la classe source a aussi bien des limites inférieures que supérieures. Cela est synonyme avec la spécification d'une valeur positive égale à la plus petite valeur nécessaire pour contenir la transformée de chaque valeur abstraite dans les limites.

24.7.9 La valeur entière est d'abord convertie en représentation décimale sans zéros initiaux et avec un préfixe "-" (TIRET-MOINS) si la valeur est négative. Si, et seulement si, "**PLUS-SIGN**" est activé à "true", les valeurs positives ont un préfixe "+" (SIGNE PLUS) avant les chiffres.

24.7.10 Le chiffre de poids fort doit être à l'extrémité initiale de la chaîne de caractères.

24.7.11 Si "**SIZE**" est "**variable**", dans ce cas c'est la chaîne de caractères résultante. Dans ce cas, ce n'est pas une erreur que de spécifier une valeur pour "**PADDING**", mais cette valeur est ignorée.

24.7.12 Si "**SIZE**" est une valeur positive ou "**fixed-to-max**", et que la chaîne résultante (dans une instance d'application de cette transformée pendant le codage) soit trop grande pour la taille fixe, dans ce cas il y a erreur de spécification ou d'application ECN, et les codeurs ne doivent pas produire de codages pour de telles valeurs abstraites.

24.7.13 Si "**SIZE**" est une valeur positive ou "**fixed-to-max**", et que la chaîne soit plus petite que la taille fixe, dans ce cas cette valeur est bourrée avec soit " " (ESPACE) ou "0" (CHIFFRE ZERO), déterminé par la valeur de "**PADDING**" et mis en préfixe afin de produire la taille spécifiée.

24.7.14 Cette transformation est définie comme étant réversible pour toutes les valeurs abstraites.

24.8 La transformée "int-to-bits"

NOTE – Un exemple de cette transformée est donné au § D.1.5.5.

24.8.1 La transformée "int-to-bits" utilise les propriétés de codage suivantes:

```
&int-to-bits-encoded-as      ENUMERATED
                             {positive-int, twos-complement}
                             DEFAULT twos-complement,
&int-to-bits-unit            Unit (1..MAX) DEFAULT bit,
&int-to-bits-size            ResultSize DEFAULT variable
```

24.8.2 La syntaxe pour la transformée "int-to-chars" doit être la suivante:

```
[INT-TO-BITS
      [AS &int-to-bits-encoded-as]
      [SIZE &int-to-bits-size]
      [MULTIPLE OF &int-to-bits-unit]]
```

24.8.3 La définition des types utilisés dans la transformée "int-to-bits" est:

```
Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
       dword32(32)} (0..256) --  (voir § 21.1)

ResultSize ::= INTEGER {variable(-1), fixed-to-max(0)} (-1..MAX) --  (voir § 21.14)
```

24.8.4 La source pour cette transformation est un entier ou un entier composite et le résultat est une chaîne de bits ou un composite de chaîne de bits. Il n'existe pas de limites associées au résultat. Les paragraphes suivants utilisent le terme *chaîne de bits résultante*.

24.8.5 Si la source est un entier, le résultat est la chaîne de bits résultante. Si la source est un entier composite, le résultat est un composite de chaîne de bits dans lequel chaque élément de la source a été transformé en chaîne de bits résultante comme spécifié au § 24.5.5.

24.8.6 "**AS**" et "**MULTIPLE OF**" ont des valeurs par défaut et n'ont pas besoin d'être activés.

24.8.7 "**SIZE**" possède une valeur par défaut et n'a pas besoin d'être activé si la source n'est pas un composite. Ce champ doit être mis à une valeur positive si la source est un composite.

24.8.8 "**SIZE**" ne doit pas être mis à "**fixed-to-max**" à moins qu'il ne s'agisse de la première transformée dans un ensemble ordonné dans la syntaxe définie au § 19.4, et la classe source a aussi bien des limites inférieures que supérieures. Cela est synonyme avec la spécification d'une valeur positive égale à la plus petite valeur nécessaire afin de contenir la transformée de chaque valeur abstraite dans les limites.

NOTE – "**SIZE**" ne peut pas être mis à "**fixed-to-max**" si la source est un composite de transformée.

24.8.9 "**AS**" sélectionne le codage de l'entier comme étant soit un codage par complément à deux ou un codage d'entier positif. La définition de ces codages est donnée dans la Rec. UIT-T X.690 | ISO/CEI 8825-1, § 8.3.2 et 8.3.3.

24.8.10 Le bit de plus fort poids doit être à l'extrémité initiale de la chaîne de bits.

24.8.11 L'entier doit d'abord être codé afin d'obtenir le nombre minimal de bits nécessaire pour produire une chaîne initiale de bits. Cela implique que le codage d'entier positif n'ait pas zéro comme bit initial (à moins qu'il n'y ait qu'un seul bit zéro dans le codage), et un codage en complément à deux ne doit pas avoir deux bits zéro initiaux successifs ou deux bits unité initiaux successifs.

24.8.12 Si "**AS**" est activé à "**positive-int**", et que la valeur à transformer soit négative, il y a erreur de spécification ou d'application ECN et les codeurs ne doivent pas coder de telles valeurs.

24.8.13 Si "**SIZE**" est "**variable**", dans ce cas la chaîne de bits initiale devient la chaîne de bits résultante. Dans ce cas, ce n'est pas une erreur de spécifier une valeur pour "**MULTIPLE OF**", mais cette valeur est ignorée.

NOTE – Ce paragraphe ne peut pas s'appliquer si la source est composite.

24.8.14 Si "**SIZE**" est une valeur positive, la taille de la chaîne résultante de bits doit être "**MULTIPLE OF**" multiplié par "**SIZE**".

24.8.15 Si "**SIZE**" est "**fixed-to-max**", dans ce cas la taille de la chaîne résultante de bits doit être le plus petit multiple de "**MULTIPLE OF**" qui est assez grand pour recevoir le codage d'une quelconque valeur abstraite de la classe à laquelle la transformée est appliquée.

NOTE – Ce paragraphe ne peut pas s'appliquer si la source est composite.

24.8.16 Si la chaîne initiale de bits (dans une instance d'application de cette transformée pendant le codage) est trop grande pour la taille fixe, dans ce cas il y a erreur de spécification ou d'application ECN et les codeurs ne doivent pas coder de telles valeurs.

24.8.17 Si la chaîne initiale de bits est plus petite que la taille spécifiée, dans ce cas, pour un codage d'entier positif, il doit avoir un préfixe de bits zéro afin de produire la chaîne de bits résultante. Si le codage est en complément à 2, dans ce cas il doit avoir un préfixe binaire de valeur égale au bit initial original, afin de produire la chaîne de bits résultante.

24.8.18 Cette transformation est définie comme étant réversible pour toutes les valeurs abstraites. Cette transformation produit une chaîne de bits autodélimitatrice si et seulement si "**SIZE**" n'est pas "**variable**" et si la source n'est pas composite. Un résultat composite n'est jamais autodélimitateur.

24.9 La transformée "bits-to-int"

24.9.1 La transformée "bits-to-int" utilise la propriété de codage suivante:

```
&bits-to-int-decoded-assuming      ENUMERATED
                                   {positive-int, twos-complement}
                                   DEFAULT twos-complement
```

24.9.2 La syntaxe pour la transformée "bits-to-int" doit être la suivante:

```
[BITS-TO-INT
      [AS &bits-to-int-decoded-assuming]]
```

24.9.3 La source pour cette transformation est une chaîne de bits ou un composite de chaîne de bits et le résultat est un entier ou un entier composite. Il n'existe pas de limites associées au résultat.

24.9.4 Si la source est une chaîne de bits, le résultat est un entier. Si la source est un composite de chaîne de bits, le résultat est un entier composite dans lequel chaque entier est le résultat de la spécification du § 24.9.5.

24.9.5 La valeur entière doit être produite par interprétation des bits comme étant en codage par complément à 2 ou en codage d'entier positif, comme spécifié dans la Rec. UIT-T X.690 | ISO/CEI 8825-1, § 8.3.2 et 8.3.3. La valeur de "**AS**" (ou sa valeur par défaut si ce champ n'est pas activé) détermine le codage à prendre en compte.

24.9.6 Cette transformation ne doit pas être utilisée lorsque des transformées réversibles sont requises.

24.10 La transformée "char-to-bits"

24.10.1 La transformée "char-to-bits" utilise les propriétés de codage suivantes:

```
&char-to-bits-encoded-as      ENUMERATED
                              {iso10646, compact, mapped}
                              DEFAULT compact,
```

```
&Char-to-bits-chars          UniversalString (SIZE(1))
                             ORDERED OPTIONAL,
&Char-to-bits-values         BIT STRING ORDERED OPTIONAL,
&char-to-bits-unit           Unit (1..MAX) DEFAULT bit,
&char-to-bits-size           ResultSize DEFAULT variable
```

24.10.2 La syntaxe pour la transformée "char-to-bits" doit être la suivante:

```
[CHAR-TO-BITS
      [AS &char-to-bits-encoded-as]
      [CHAR-LIST &Char-to-bits-chars]
      [BITS-LIST &Char-to-bits-values]
      [SIZE &char-to-bits-size]
      [MULTIPLE OF &char-to-bits-unit]]
```

24.10.3 La définition des types utilisés dans la transformée "char-to-bits" est:

```
Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
       dword32(32)} (0..256) --  (voir § 21.1)

ResultSize ::= INTEGER {variable(-1), fixed-to-max(0)} (-1..MAX) --  (voir § 21.14)
```

24.10.4 La source pour cette transformation est un caractère isolé à partir de soit:

a) la spécification d'un codage pour la catégorie des chaînes de caractères (voir § 23.4.2.1); ou

b) un composite de caractère isolé.

et le résultat est une chaîne de bits dans le cas a) et un composite de chaîne de bits dans le cas b).

24.10.5 Le composite de chaîne de bits dans le cas b) doit être la séquence ordonnée des chaînes de bits produites par les transformations appliquées à chaque élément du composite de chaîne de bits source. Il y a erreur de spécification ECN si les champs "**ZERO-PATTERN**" et "**ONE-PATTERN**" ont des tailles différentes.

24.10.6 La source pour cette transformation est un caractère isolé ou un composite de caractère isolé. Si la source est un caractère isolé, le résultat est une chaîne de bits. Si la source est un composite de caractère isolé, le résultat est un composite de chaîne de bits.

24.10.7 Si la source est un composite, le composite résultant est déterminé par l'application de la spécification suivante à tous les éléments du composite-source de façon à former le composite-résultat. Il y a erreur de spécification ECN si cette transformation est appliquée à un composite avec "**AS**" mis à "**mapped**" et si la taille des chaînes de bits dans la production "**BITS-LIST**" ne sont pas tous le même.

24.10.8 Si le texte suivant renvoie à une éventuelle "contrainte d'alphabet effectivement permis", une telle contrainte existe si et seulement si la transformée est la première dans une liste ordonnée utilisée au § 23.4 et si la classe à laquelle l'objet de codage est appliqué a une contrainte d'alphabet effectivement permis.

NOTE – Cela ne peut être le cas si la classe à laquelle la transformée est appliquée fait partie d'une structure produite implicitement ou explicitement. Ce paragraphe ne peut jamais s'appliquer à un composite dont les éléments n'ont jamais de contraintes d'alphabet effectivement permis.

24.10.9 Les champs "**AS**", "**SIZE**" et "**MULTIPLE OF**" ont tous des valeurs par défaut et n'ont pas besoin d'être activés. Les champs "**CHAR-LIST**" et "**BITS-LIST**" ne sont utilisés que si "**AS**" est activé à "**mapped**", auquel cas leur présence est obligatoire, et ils doivent alors contenir au moins un élément dans la liste ordonnée.

24.10.10 La notation ECN prend en charge seulement les caractères contenus dans le jeu de caractères ISO/CEI 10646-1. Si des types ASN.1 tels que "GeneralString" sont utilisés, des caractères étrangers à ce jeu de caractères peuvent théoriquement apparaître. De tels caractères ne sont pas pris en charge par cette transformation.

24.10.11 Si "**AS**" est "**mapped**", dans ce cas la transformée est spécifiée par les valeurs de "**CHAR-LIST**" et "**BITS-LIST**", qui doivent tous deux être spécifiés, et les valeurs de "**MULTIPLE OF**" et "**SIZE**" sont ignorées. La transformation est spécifiée dans les § 24.10.11.1 à 24.10.11.5.

24.10.11.1 "**CHAR-LIST**" et "**BITS-LIST**" sont respectivement une liste ordonnée de caractères isolés et une liste ordonnée de valeurs de chaîne de bits. (Ces paramètres sont ignorés si "**AS**" n'est pas mis à "**mapped**".)

24.10.11.2 Il doit y avoir un nombre égal de valeurs dans chaque liste, et toutes les valeurs de caractère contenues dans "**CHAR-LIST**" doivent être distinctes.

24.10.11.3 La transformée d'un caractère en "**CHAR-LIST**" est la chaîne de bits spécifiée dans la position correspondante contenue dans une liste "**BITS-LIST**".

24.10.11.4 Si, dans une instance d'application de cette transformée, un caractère doit être transformé alors qu'il ne figure pas dans la liste "**CHAR-LIST**", il s'agit d'une erreur de spécification ou d'application ECN.

NOTE – En général, il ne sera possible de vérifier cette erreur par utilitaire qu'au moment du codage, car des restrictions relatives à d'éventuelles valeurs abstraites peuvent ne pas être formellement présentes dans la spécification ASN.1.

24.10.11.5 Dans ce cas ("**AS**" mis à "**mapped**"), la transformée est définie comme étant réversible (pour toutes les valeurs abstraites) si et seulement si l'ensemble de toutes les valeurs de chaîne de bits contenues dans une liste "**BITS-LIST**" sont distinctes; sinon elle ne doit pas être utilisée lorsqu'une transformation réversible est requise. Le résultat est autodélimitateur si les valeurs de chaîne de bits contenues dans la liste "**BITS-LIST**" sont autodélimitatrices (voir § 3.2.41). Un résultat composite n'est jamais autodélimitateur.

24.10.12 Si "**AS**" est "**iso10646**", la transformée est spécifiée dans les § 24.10.12.1 à 24.10.12.5.

24.10.12.1 Le caractère est d'abord converti en un entier avec la valeur numérique spécifiée dans ISO/CEI 10646-1.

NOTE – L'ISO/CEI 10646-1 contient les caractères ASCII de commande, qui sont positionnés dans la rangée 1.

24.10.12.2 Si le caractère est issu d'une chaîne de caractères qui possède une contrainte d'alphabet effectivement permis associée (voir § 24.10.8), dans ce cas l'entier a des contraintes effectives de longueur juste suffisantes pour contenir les valeurs numériques de tous les caractères contenus dans l'alphabet effectivement permis.

24.10.12.3 S'il n'y a pas de contraintes d'alphabet effectivement permis, dans ce cas l'entier a une contrainte de longueur effective associée de 0..32767.

24.10.12.4 Cette valeur entière est alors convertie en bits au moyen de la transformée:

```
INT-TO-BITS -- (voir § 24.8)
      AS positive-int
      SIZE <size>
      MULTIPLE OF <multiple-of>
```

où "<size>" est la valeur de "**SIZE**" et "<multiple-of>" est la valeur de "**MULTIPLE OF**" pour la transformée "char-to-bits". ("**SIZE**" et "**MULTIPLE OF**" prennent leurs valeurs par défaut si ces champs ne sont pas activés.)

24.10.12.5 Dans ce cas ("**AS**" mis à "**iso10646**"), la transformée est définie comme étant réversible pour toutes les valeurs abstraites. Elle produit une chaîne autodélimitatrice de bits si et seulement si "**SIZE**" n'est pas "variable". Un résultat composite n'est jamais autodélimitateur.

24.10.13 Si "**AS**" est "**compact**", dans ce cas il y a erreur de spécification ECN s'il n'y a pas de contraintes d'alphabet effectivement permis; sinon la transformée est spécifiée dans les § 24.10.13.1 à 24.10.13.4.

24.10.13.1 Tous les caractères contenus dans l'alphabet effectivement permis sont placés en ordre canonique au moyen de leur valeur ISO/CEI 10646-1, en commençant pas la plus petite valeur. La première valeur dans la liste est alors attribuée à la valeur entière zéro, la suivante à un, et ainsi de suite.

24.10.13.2 Si l'alphabet effectivement permis contient "n" caractères, dans ce cas l'entier a une contrainte de longueur effective de 0..n-1.

24.10.13.3 Cet entier est alors converti en bits au moyen de la transformée:

```
INT-TO-BITS -- (voir § 24.8)
      AS positive-int
      SIZE <size>
      MULTIPLE OF <multiple-of>
```

où "<size>" est la valeur de "**SIZE**" et "<multiple-of>" est la valeur de "**MULTIPLE OF**" pour la transformée "char-to-bits". ("**SIZE**" et "**MULTIPLE OF**" prennent leurs valeurs par défaut si ces champs ne sont pas activés.)

NOTE – Le codage PER des types de chaîne de caractères n'utilise l'équivalent de "**compact**" que si l'application de cet algorithme réduit le nombre de bits requis pour coder des caractères (au moyen de "**fixed-to-max**"). Ce degré de commande n'est pas possible dans cette version de la présente Recommandation | Norme internationale.

24.10.13.4 Dans ce cas ("**AS**" mis à "**compact**"), la transformée est définie comme étant réversible pour toutes les valeurs abstraites. Elle produit une chaîne autodélimitatrice de bits si et seulement si "**SIZE**" n'est pas "**variable**". Un résultat composite n'est jamais autodélimitateur.

24.11 La transformée "bits-to-char"

24.11.1 La transformée "bits-to-char" utilise les propriétés de codage suivantes:

```
&bits-to-char-decoded-assuming     ENUMERATED
                                   {iso10646, mapped}
                                   DEFAULT iso10646,
&Bits-to-char-values               BIT STRING ORDERED OPTIONAL,
&Bits-to-char-chars                UniversalString (SIZE(1))
                                   ORDERED OPTIONAL
```

24.11.2 La syntaxe pour la transformée "bits-to-char" doit être la suivante:

```
[BITS-TO-CHAR
      [AS &bits-to-char-decoded-assuming]
      [BITS-LIST &Bits-to-char-values]
      [CHAR-LIST &Bits-to-char-chars]]
```

24.11.3 La source pour cette transformation est une chaîne de bits ou un composite de chaîne de bits. Si la source est une chaîne de bits, le résultat est un caractère isolé. Si la source est un composite de chaîne de bits, le résultat est un composite de caractère isolé.

24.11.4 Si la source est un composite de chaîne de bits, dans ce cas le composite de caractère isolé qui en résulte est une liste ordonnée de caractères isolés résultant de la transformation de chacun des éléments du composite de chaîne de bits.

24.11.5 Si "**AS**" est "**iso10646**", dans ce cas la chaîne de bits doit être interprétée en tant que codage d'entier positif qui contient la valeur numérique ISO/CEI 10646-1 d'un caractère. Il y a erreur de spécification ECN si la valeur entière dépasse 32767.

24.11.6 Si "**AS**" est "**mapped**", dans ce cas la transformée est spécifiée par les valeurs de "**CHAR-LIST**" et "**BITS-LIST**". La transformée est définie dans les § 24.11.6.1 à 24.11.6.5.

24.11.6.1 "**CHAR-LIST**" et "**BITS-LIST**" sont respectivement une liste ordonnée de caractères isolés et une liste ordonnée de valeurs de chaîne de bits. (Ces paramètres sont ignorés si "**AS**" n'est pas mis à "**mapped**".)

24.11.6.2 Il doit y avoir un nombre égal de valeurs dans chaque liste, et toutes les valeurs de caractère et de chaîne de bits dans la liste doivent être distinctes.

24.11.6.3 La transformée d'une chaîne de bits dans la production "**BITS-LIST**" est le caractère spécifié dans la position correspondante dans la liste "**CHAR-LIST**".

24.11.6.4 Si, dans une instance d'application de cette transformation, une chaîne de bits doit être transformée sans figurer dans la production "**BITS-LIST**", il s'agit d'une erreur de spécification ou d'application ECN.

NOTE – En général, il ne sera possible de vérifier cette erreur par utilitaire qu'au moment du codage, car des restrictions relatives à d'éventuelles valeurs abstraites peuvent ne pas être formellement présentes dans la spécification ASN.1.

24.11.6.5 La transformée est définie comme étant réversible pour toutes les valeurs abstraites.

24.12 La transformée "bit-to-bits"

24.12.1 La transformée "bit-to-bits" utilise les propriétés de codage suivantes:

```
&bit-to-bits-one            Non-Null-Pattern DEFAULT bits:'1'B,
&bit-to-bits-zero           Non-Null-Pattern DEFAULT bits:'0'B
```

24.12.2 La syntaxe pour la transformée "bit-to-bits" doit être la suivante:

```
[BIT-TO-BITS
      [ZERO-PATTERN &bit-to-bits-zero]
      [ONE-PATTERN &bit-to-bits-one]]
```

24.12.3 La définition du type utilisé dans la transformée "bit-to-bits" est la suivante:

```
Non-Null-Pattern ::= Pattern
      (ALL EXCEPT (bits:''B | octets:''H | char8:"" | char16:"" |
                  char32:"")) -- (voir § 21.10.2)
```

24.12.4 La source pour cette transformation est un bit isolé à partir de soit:

a) la spécification d'un codage pour la catégorie des chaînes de bits (voir § 23.2); ou

b) un composite de chaîne de bits avec une unité de 1 bit.

Le résultat est une chaîne de bits dans le cas a) et un composite de chaîne de bits dans le cas b).

24.12.5 Le composite de chaîne de bits dans le cas b) doit être la séquence ordonnée des chaînes de bits produites par les transformations appliquées à chaque élément du composite de chaîne de bits source. Il y a erreur de spécification ECN si les champs "**ZERO-PATTERN**" et "**ONE-PATTERN**" ont des tailles différentes.

24.12.6 Au plus un des champs "**ZERO-PATTERN**" et "**ONE-PATTERN**" doit avoir la valeur "**different:any**".

NOTE – Une valeur "**different:any**" signifie ici une séquence qui n'est pas la même que l'autre séquence mais qui a la même longueur.

24.12.7 L'option "**any-of-length**" ne doit pas être utilisée pour soit "**ZERO-PATTERN**" ou "**ONE-PATTERN**".

24.12.8 Si le bit est mis à zéro, le résultat est le champ "**ZERO-PATTERN**". Si le bit est mis à un, le résultat est le champ "**ONE-PATTERN**".

24.12.9 Il y a erreur de spécification ECN si "**ZERO-PATTERN**" et "**ONE-PATTERN**" sont identiques, ou si l'un est une sous-chaîne initiale de l'autre.

24.12.10 Cette transformation est définie comme étant réversible pour toutes les valeurs abstraites et le résultat est autodélimitateur à moins que la transformée ne soit appliquée à un composite. Un résultat composite n'est jamais autodélimitateur.

24.13 La transformée "bits-to-bits"

24.13.1 La transformée "bits-to-bits" utilise les propriétés de codage suivantes:

```
&Source-values              BIT STRING ORDERED,
&Result-values              BIT STRING ORDERED
```

24.13.2 La syntaxe pour la transformée " bits-to-bits " doit être la suivante:

```
[BITS-TO-BITS
      SOURCE-LIST &Source-values
      RESULT-LIST &Result-values]
```

24.13.3 La source pour cette transformation est soit une chaîne de bits ou un composite de chaîne de bits. Si la source est une chaîne de bits, le résultat est une chaîne de bits. Si la source est un composite de chaîne de bits, le résultat est un composite de chaîne de bits.

24.13.4 Si la source est un composite de chaîne de bits, dans ce cas le composite résultant de la chaîne de bits est la liste ordonnée des chaînes de bits obtenue par l'application de la spécification suivante à chaque chaîne de bits contenue dans la source.

24.13.5 Les deux champs "**SIZE**" et "**MULTIPLE OF**" ont des valeurs par défaut et n'ont pas besoin d'être activés. "**SOURCE-LIST**" et "**RESULT-LIST**" sont requis, et doivent contenir au moins un élément dans la liste ordonnée.

24.13.6 La transformée est spécifiée par les valeurs de "**SOURCE-LIST**" et "**RESULT-LIST**".

24.13.7 Il doit y avoir un nombre égal de valeurs de chaîne de bits dans chaque liste, et toutes les valeurs de chaîne de bits contenues dans la liste "**SOURCE-LIST**" doivent être distinctes.

24.13.8 La transformée d'une chaîne de bits en "**SOURCE-LIST**" est la chaîne de bits spécifiée dans la position correspondante de la liste "**RESULT-LIST**".

24.13.9 Si cette transformation est appliquée à un composite, toutes les chaînes de bits contenues dans la liste "**RESULT-LIST**" doivent avoir la même taille.

24.13.10 Si, dans une instance d'application de cette transformation, une chaîne de bits source n'est pas dans la liste "**SOURCE-LIST**", il s'agit d'une erreur de spécification ou d'application ECN.

NOTE – En général, il ne sera possible de vérifier cette erreur par utilitaire qu'au moment du codage, car des restrictions relatives à d'éventuelles valeurs abstraites peuvent ne pas être formellement présentes dans la spécification ASN.1.

24.13.11 La transformée est définie comme étant réversible (pour toutes les valeurs abstraites) si et seulement si l'ensemble de toutes les valeurs de chaîne de bits en "**RESULT-LIST**" sont distinctes; sinon elle ne doit pas être utilisée lorsqu'une transformation réversible est requise. Le résultat est autodélimitateur si les valeurs de chaîne de bits en

"**RESULT-LIST**" sont distinctes et autodélimitatrices (voir 3.2.41) et la transformée est appliquée à une chaîne de bits. Un résultat composite n'est jamais autodélimitateur.

24.14 La transformée "chars-to-composite-char"

24.14.1 La transformée "chars-to-composite-char" convertit une chaîne de caractères en un composite de caractère isolé.

24.14.2 La syntaxe pour la transformée "chars-to-composite-char" doit être la suivante:

```
[CHARS-TO-COMPOSITE-CHAR]
```

24.14.3 La source de cette transformée est une chaîne de caractères et le résultat est un composite de caractère isolé.

24.14.4 Le composite de caractère isolé est une liste ordonnée des caractères contenus dans la chaîne de caractères source.

24.14.5 Cette transformation est définie comme étant réversible pour toutes les valeurs abstraites.

24.15 La transformée "bits-to-composite-bits"

24.15.1 La transformée "bits-to-composite-bits" convertit une chaîne de bits en un composite de chaîne de bits, où chaque élément de chaîne de bits a la même longueur (connue).

24.15.2 La transformée "bits-to-composite-bits" utilise les propriétés de codage suivantes:

```
&bits-to-composite-bits-unit      Unit (1..MAX) DEFAULT bit
```

24.15.3 La syntaxe pour la transformée "bits-to-composite-bits" doit être la suivante:

```
[BITS-TO-COMPOSITE-BITS
      [UNIT &bits-to-composite-bits-unit]]
```

24.15.4 La définition du type utilisé dans la transformée "bits-to-composite-bits" est la suivante:

```
Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
       dword32(32)} (0..256) --  (voir § 21.1)
```

24.15.5 La source de cette transformée est une chaîne de bits et le résultat est un composite de chaîne de bits de taille "**UNIT**".

24.15.6 Le composite de chaîne de bits de taille "**UNIT**" est une liste ordonnée des chaînes de bits dont chacune est de taille "**UNIT**". La première chaîne de bits contenue dans le composite est constituée des "**UNIT**" premiers bits issus de la chaîne de bits source. La deuxième chaîne est constituée des "**UNIT**" bits suivants, et ainsi de suite. Si la chaîne de bits source n'est pas un multiple de "**UNIT**" bits, il y a erreur de spécification ou d'application ECN.

24.15.7 Cette transformation est définie comme étant réversible pour toutes les valeurs abstraites.

24.16 La transformée "octets-to-composite-bits"

24.16.1 La transformée "octets-to-composite-bits" convertit une chaîne d'octets en un composite de chaîne de bits de taille 8 bits.

24.16.2 La syntaxe pour la transformée "octets-to-composite-bits" doit être la suivante:

```
[OCTETS-TO-COMPOSITE-BITS]
```

24.16.3 La source de cette transformée est une chaîne d'octets et le résultat est un composite de chaîne de bits de taille 8 bits.

24.16.4 Le composite de chaîne de bits de taille 8 est une liste ordonnée des chaînes de bits correspondant aux octets contenus dans la chaîne d'octets source.

24.16.5 Cette transformation est définie comme étant réversible pour toutes les valeurs abstraites.

24.17 La transformée "composite-char-to-chars"

24.17.1 La transformée "composite-char-to-chars" convertit un composite de caractère isolé en une chaîne de caractères.

24.17.2 La syntaxe pour la transformée "composite-char-to-chars" doit être la suivante:

```
[COMPOSITE-CHAR-TO-CHARS]
```

24.17.3 La source de cette transformée est un composite de caractère isolé et le résultat est une chaîne de caractères.

24.17.4 La chaîne de caractères est formée à partir de la liste ordonnée de caractères présente dans le composite de caractère isolé (source).

24.17.5 Cette transformation est définie comme étant réversible pour toutes les valeurs abstraites.

24.18 La transformée "composite-bits-to-bits"

24.18.1 La transformée "composite-bits-to-bits" convertit un composite de chaîne de bits de taille unitaire connue en une chaîne de bits.

24.18.2 La syntaxe pour la transformée "composite-bits-to-bits" doit être la suivante:

```
[COMPOSITE-BITS-TO-BITS]
```

24.18.3 La source de cette transformée est un composite de chaîne de bits et le résultat est une chaîne de bits.

24.18.4 La chaîne de bits est formée à partir de la liste ordonnée des chaînes de bits présente dans le composite (source) de chaîne de bits.

24.18.5 Cette transformation est définie comme étant réversible pour toutes les valeurs abstraites. La chaîne de bits résultante n'est pas autodélimitatrice.

NOTE – Cette transformation est réversible parce que les unités utilisées dans sa production sont spécifiées dans la transformée qui a produit le composite de chaîne de bits, et sont associées à ce composite.

24.19 La transformée "composite-bits-to-octets"

24.19.1 La transformée "composite-bits-to-octets" convertit un composite de chaîne de bits de taille unitaire 8 en une chaîne d'octets. Il y a erreur de spécification ECN si elle est appliquée à un composite de chaîne de bits qui possède une taille unitaire qui n'est pas 8.

24.19.2 La syntaxe pour la transformée "composite-bits-to-octets" doit être la suivante:

```
[COMPOSITE-BITS-TO-OCTETS]
```

24.19.3 La source de cette transformée est un composite de chaîne de bits et le résultat est une chaîne d'octets.

24.19.4 La chaîne d'octets est formée à partir de la liste ordonnée des chaînes de bits présente dans le composite (source) de chaîne de bits.

24.19.5 Cette transformation est définie comme étant réversible pour toutes les valeurs abstraites.

25 Codages complets et la classe #OUTER

S'il n'y a pas d'objet de codage de la classe **#OUTER** dans l'ensemble d'objets de codage combinés appliqué à un type dans le module ELM, dans ce cas le codeur et le décodeur doivent considérer un objet de codage de cette classe dans lequel toutes les propriétés de codage ont leurs valeurs par défaut.

25.1 Propriétés, syntaxe et finalité du codage pour la classe #OUTER

25.1.1 La syntaxe de définition des objets de codage de la classe **#OUTER** est définie comme suit:

```
#OUTER ::= ENCODING-CLASS {

    -- Point d'alignement
    &alignment-point                ENUMERATED
                                    {unchanged, reset } DEFAULT reset,
```

```
        -- Bourrage
        &post-padding-unit                Unit (1..MAX) DEFAULT octet,
        &post-padding                     Padding DEFAULT zero,
        &post-padding-pattern             Non-Null-Pattern (ALL EXCEPT different:any)
                                          DEFAULT bits:'0'B,

        -- Spécification de l'inversion de l'ordre des bits (voir § 22.12)
        &bit-reversal                     ReversalSpecification
                                          DEFAULT no-reversal,

        -- Ajoutée bits action
        &added-bits                       ENUMERATED
                                          {hard-error, signal-application,
                                          silently-ignore, next-value}
                                          DEFAULT hard-error

    } WITH SYNTAX {

        [ALIGNMENT &alignment-point]
        [PADDING
              [MULTIPLE OF &post-padding-unit]
              [POST-PADDING &post-padding
                    [PATTERN &post-padding-pattern]]]
        [BIT-REVERSAL &bit-reversal]
        [ADDED BITS DECODING          &added-bits]

    }
```

25.1.2 La définition des types utilisés dans la spécification de la classe **#OUTER** est la suivante:

```
Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
       dword32(32)} (0..256) -- (voir § 21.1)

Padding ::= ENUMERATED {zero, one, pattern, encoder-option} -- (voir § 21.9)

Non-Null-Pattern ::= Pattern
      (ALL EXCEPT (bits:''B | octets:''H | char8:"" | char16:"" |
                   char32:"")) -- (see 21.10.2)
```

25.1.3 Les objets de codage de la classe **#OUTER** spécifient les actions du codeur et du décodeur dans le cadre du codage entier d'un type qui est codé par soit:

a) application d'un codage dans le module ELM; ou

b) application d'un codage à un type contenu.

25.1.4 Trois spécifications indépendantes peuvent être appliquées (voir § 25.1.5 à 25.1.7).

25.1.5 La spécification "**ALIGNMENT**" n'est applicable qu'à un type contenu, et détermine si le point d'alignement doit être réinitialisé au début du conteneur ou doit être celui qui a été utilisé pour le codage du conteneur.

25.1.6 La spécification "**PADDING**" détermine que le codage entier doit être bourré avec des bits de fin afin de transformer le nombre de bits à partir du point d'alignement en un multiple entier d'une certaine unité.

25.1.7 La spécification "**ADDED BITS DECODING**" n'est applicable qu'à des décodeurs, et détermine l'action à effectuer s'il y a encore des bits dans l'unité PDU après que le décodage a été effectué conformément aux spécifications de codage.

NOTE – Cette disposition vise essentiellement à offrir un mécanisme simple pour l'extensibilité sans l'utilisation du marqueur ASN.1 d'extensibilité. Une version ultérieure de la présente Recommandation | Norme internationale est censée donner une prise en charge améliorée de l'extensibilité.

25.1.8 Les champs "**ALIGNMENT**", "**PADDING**", et "**ADDED BITS DECODING**" prennent tous leurs valeurs par défaut s'ils ne sont pas activés ou s'il n'y a pas d'objet de codage de classe **#OUTER** dans l'ensemble d'objets de codage combinés.

NOTE – Les valeurs par défaut sont celles qui ont été utilisées par l'objet de codage de classe **#OUTER** pour les règles PER de base sans alignement.

25.2 Actions du codeur pour #OUTER

25.2.1 Si "**ALIGNMENT**" est "**unchanged**", dans ce cas le point d'alignement utilisé dans le codage d'un type de contenu doit être le point d'alignement utilisé dans le codage du conteneur.

25.2.2 Si "**ALIGNMENT**" est "**reset**", dans ce cas le point d'alignement utilisé dans le codage d'un type de contenu doit être le début du codage de ce type.

25.2.3 Si "**PADDING**" est activé, dans ce cas le codeur doit ajouter des bits conformément à la valeur de "**PADDING**" et "**PATTERN**" afin de transformer le nombre de bits à partir du point d'alignement en un multiple de "**MULTIPLE OF**" unités. La spécification "**PATTERN**" doivent être reproduite et tronquée si nécessaire.

25.2.4 Le codeur doit diagnostiquer une erreur de spécification ou d'application ECN si le codage vise un type se trouvant dans une contrainte de contenu relative à une chaîne d'octets, et si le codage du type (après toutes les actions spécifiées "**PADDING**") n'est pas un multiple entier de huit bits.

25.2.5 Si l'inversion de l'ordre des bits est activée, les actions du codeur spécifiées au § 22.12 doivent être appliquées au moyen de la valeur de "**MULTIPLE OF**" spécifiée pour (ou prise par défaut dans) "**PADDING**".

25.2.6 Le codeur doit ignorer "**ADDED BITS DECODING**".

25.3 Actions du décodeur pour #OUTER

25.3.1 Si l'inversion de l'ordre des bits est activée, les actions du décodeur spécifiées au § 22.12 doivent être appliquées au moyen de la valeur de "**MULTIPLE OF**" spécifiée pour (ou prise par défaut dans) "**PADDING**".

25.3.2 Si "**ALIGNMENT**" est "**unchanged**", dans ce cas le point d'alignement utilisé dans le codage d'un type de contenu doit être le point d'alignement utilisé dans le codage du conteneur.

25.3.3 Si "**ALIGNMENT**" est "**reset**", dans ce cas le point d'alignement utilisé dans le codage d'un type de contenu doit être le début du codage de ce type.

25.3.4 Le décodeur doit déterminer les bits ajoutés par "**PADDING**" (si présent), et doit négliger sans notification les bits ajoutés, quelle que soit leur valeur.

25.3.5 Si l'unité PDU (ou le conteneur d'un type contenu) contient encore des bits après l'extrémité du codage, dans ce cas le décodeur doit effectuer les actions suivantes:

a) si "**ADDED BITS DECODING**" est "**hard-error**": diagnostiquer une erreur de codeur;

b) Si "**ADDED BITS DECODING**" est "**signal-application**": ignorer tous bits suivants et signaler à l'application qu'il peut y avoir des extensions critiques au protocole;

c) Si "**ADDED BITS DECODING**" est "**silently-ignore**": ignorer tous bits suivants;

d) Si "**ADDED BITS DECODING**" est "**next-value**": cesser le décodage et s'attendre que l'application entreprendra le décodage d'une nouvelle valeur à partir des bits restants.

Annexe A

Addendum à la Rec. UIT-T X.680 | ISO/CEI 8824-1

(Cette annexe fait partie intégrante de la présente Recommandation | Norme internationale)

La présente annexe spécifie les modifications qui doivent être appliquées lorsque des productions et/ou des clauses issues de la Rec. UIT-T X.680 | ISO/CEI 8824-1 sont citées en référence dans la présente Recommandation | Norme internationale.

A.1 Clause d'exportations et importations

Les productions "AssignedIdentifier", "symbol" et "Reference" du § 12.1, ainsi que les § 12.12 et 12.15 de la Rec. UIT-T X.680 | ISO/CEI 8824-1 sont modifiés comme suit:

12.1 **AssignedIdentifier ::= DefinitiveIdentifier |**
empty

Symbol ::=
Reference
| **BuiltinEncodingClassReference**
| **ParameterizedReference**

Reference ::=
encodingclassreference
| **ExternalEncodingClassReference**
| **encodingobjectreference**
| **encodingobjectsetreference**

NOTE 1 – La spécification "AssignedIdentifier" est modifiée parce que les références "valuereference" ne peuvent être ni définies ni importées dans des modules ELM ou EDM.

NOTE 2 – La spécification "BuiltinEncodingClassReference ne peut être utilisée qu'en tant que "Symbol" dans une clause d'importation. L'utilisation de la production "ExternalEncodingClassReference dans "Reference" est expliquée au § 14.11.

12.12 Lorsque l'option "SymbolsExported" de "Exports" est sélectionnée, dans ce cas chaque production "Symbol" en "SymbolsExported" doit répondre à une et seulement une des conditions suivantes:

a) elle est définie dans le module à partir duquel elle est exportée; ou

b) elle apparaît exactement une fois dans l'option "SymbolsImported" de "Imports" dans le module à partir duquel elle est exportée.

12.15 Lorsque l'option "SymbolsImported" de "Imports" est sélectionnée:

a) chaque production "Symbol" en "SymbolsFromModule" doit soit

1) être définie dans le corps du module indiqué par "GlobalModuleReference" en "SymbolsFromModule", ou

2) être présente précisément une fois dans la clause d'importation du module indiqué par la référence "GlobalModuleReference" en "SymbolsFromModule".

NOTE – Cela n'interdit pas d'importer le même nom de symbole, défini en deux modules différents, dans un autre module. Cependant, si le même nom "Symbol" apparaît plus d'une fois dans la clause d'importation du module "A", ce nom "Symbol" ne peut pas être exporté à partir de "A" pour importation dans un autre module "B".

b) Toutes les productions "SymbolsFromModule" contenues dans la liste "SymbolsFromModuleList" doivent contenir des instances de "GlobalModuleReference" telles que:

i) les références "modulereference" qu'elles contiennent soient toutes différentes les unes des autres (qu'elles soient ASN.1, ou des modules EDM) et à partir du champ "modulereference" associé au module effectuant la référence; et

ii) la production "AssignedIdentifier", lorsqu'elle n'est pas vide, indique des valeurs d'identificateur d'objet qui sont toutes différentes les unes des autres et de la valeur d'identificateur d'objet (si présente) associée au module effectuant la référence.

A.2 Addition de `REFERENCE`

NOTE – Cette modification est introduite pour le seul objectif de l'article 23.

La production "Type" dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 16.1, est modifiée comme suit:

```
Type ::=
        BuiltinType
    |   ReferencedType
    |   ConstrainedType
    |   REFERENCE
```

A.3 Notation pour valeurs de chaîne de caractères

La production "CharsDefn" de la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 37.8, est modifiée comme suit:

```
CharsDefn ::=
        cstring
    |   Quadruple
    |   Tuple
    |   AbsoluteCharReference

AbsoluteCharReference ::=
        ModuleIdentifier
        "."
        valuereference
```

La production "AbsoluteCharReference" est un nom entièrement qualifié qui fait référence à une valeur de chaîne de caractères (de type "`IA5String`" ou "`BMPString`") définie dans le module "`ASN1-CHARACTER-MODULE`" (voir Rec. UIT-T X.680 | ISO/CEI 8824-1, § 38.1).

Annexe B

Addendum à la Rec. UIT-T X.681 | ISO/CEI 8824-2

(Cette annexe fait partie intégrante de la présente Recommandation | Norme internationale)

La présente annexe spécifie les modifications qui doivent être appliquées lorsque des productions et/ou des clauses issues de la Rec. UIT-T X.681 | ISO/CEI 8824-2 sont citées en référence dans la présente Recommandation | Norme internationale.

B.1 Définitions

Les définitions suivantes sont ajoutées à la Rec. UIT-T X.681 | ISO/CEI 8824-2, § 3.4:

champ de classe de codage: champ qui contient une classe de codage arbitraire.

type de champ de classe de codage: type spécifié par référence à un certain champ de type d'une classe d'objets de codage.

champ d'objet de codage: champ qui contient un objet de codage d'une classe de codage spécifiée. Un tel champ est soit de classe fixe ou de classe variable. Dans le premier cas, la classe de l'objet de codage est fixée par la spécification du champ. Dans le second cas, la classe de l'objet de codage est contenue dans un champ de classe de codage (spécifique) du même objet de codage.

champ d'ensemble d'objets de codage: champ qui contient un ensemble d'objets de codage d'une certaine classe de codage spécifiée.

champ de liste de valeurs ordonnées de type fixe: champ qui contient une liste ordonnée (éventuellement vide) de valeurs d'un certain type spécifié.

champ de liste ordonnée d'objets de codage: champ qui contient une liste ordonnée non vide d'objets de codage d'une certaine classe de codage spécifiée.

champ de référence: champ qui contient une référence à un champ de structure de codage (voir également § 17.5.15).

B.2 Items lexicaux additionnels

NOTE – Cette modification est introduite pour le seul objectif de l'article 23.

Les définitions suivantes sont ajoutées à la Rec. UIT-T X.681 | ISO/CEI 8824-2, article 7:

B.2.1 Références de champ de liste ordonnée de valeurs

Nom de l'item – orderedvaluelistfieldreference

Une référence "orderedvaluelistfieldreference" doit se composer d'un perluète ("**&**") immédiatement suivi par une séquence de caractères comme spécifié pour une référence "typereference" dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, § 11.2.

B.2.2 Références de champ de liste ordonnée d'objets de codage

Nom de l'item – orderedencodingobjectlistfieldreference

Une référence "orderedencodingobjectlistfieldreference" doit se composer d'un perluète ("**&**") immédiatement suivi par une séquence de caractères comme spécifié pour la référence "objectsetreference" dans la Rec. UIT-T X.681 | ISO/CEI 8824-2, § 7.3.

B.2.3 Références de champ de classe de codage

Nom de l'item – encodingclassfieldreference

Une référence "encodingclassfieldreference" doit se composer d'un perluète ("**&**") immédiatement suivi par une séquence de caractères comme spécifié pour une référence "encodingclassreference au § 8.3.

B.3 Addition de "ENCODING-CLASS"

NOTE – Cette modification est introduite pour le seul objectif de l'article 23.

Remplacer le mot réservé "CLASS" par "ENCODING-CLASS" dans la Rec. UIT-T X.681 | ISO/CEI 8824-2, § 9.3.

B.4 Additions de "FieldSpec"

NOTE – Cette modification est introduite pour le seul objectif de l'article 23.

La Rec. UIT-T X.681 | ISO/CEI 8824-2, § 9.4, est modifiée comme suit:

```
FieldSpec ::=
                FixedTypeValueFieldSpec
            |   FixedTypeValueSetFieldSpec
            |   FixedTypeOrderedValueListFieldSpec
            |   FixedClassEncodingObjectFieldSpec
            |   VariableClassEncodingObjectFieldSpec
            |   FixedClassEncodingObjectSetFieldSpec
            |   FixedClassOrderedEncodingObjectListFieldSpec
            |   EncodingClassFieldSpec
```

B.5 Spécification du champ "liste de valeurs ordonnées de type fixe"

NOTE – Cette modification est introduite pour le seul objectif de l'article 23.

Une spécification "FixedTypeOrderedValueListFieldSpec" spécifie que le champ contient une liste de valeurs ordonnées de type fixe (voir § B.1 de la présente Recommandation | Norme internationale):

```
FixedTypeOrderedValueListFieldSpec ::=
        orderedvaluelistfieldreference
        DefinedType
        ORDERED
        FixedTypeOrderedValueListOptionalitySpec ?

FixedTypeOrderedValueListOptionalitySpec ::= OPTIONAL | DEFAULT OrderedValueList
```

Le nom du champ est "orderedvaluelistfieldreference". La production "DefinedType" fait référence au type de valeurs contenu dans le champ. La production "FixedTypeOrderedValueListOptionalitySpec", si présente, spécifie que le champ peut ne pas être spécifié dans une définition d'objet de codage, ou, dans le cas "DEFAULT", que son omission produit la liste "OrderedValueList" suivante (voir Rec. UIT-T X.680 | ISO/CEI 8824-1, § 25.3), valeurs qui doivent toutes être de type "DefinedType".

B.6 Spécification du champ "objet de codage de classe fixe"

NOTE – Cette modification est introduite pour le seul objectif de l'article 23.

Une spécification "FixedClassEncodingObjectFieldSpec" spécifie que le champ contient un objet de codage de classe fixe (voir § B.1 de la présente Recommandation | Norme internationale):

```
FixedClassEncodingObjectFieldSpec ::=
        objectfieldreference
        DefinedOrBuiltinEncodingClass
        EncodingObjectOptionalitySpec?

EncodingObjectOptionalitySpec ::= OPTIONAL | DEFAULT EncodingObject
```

Le nom du champ est "objectfieldreference". La production "DefinedOrBuiltinEncodingClass" fait référence à la classe de codage de l'objet de codage contenu dans le champ (qui peut être la classe "EncodingClass" en cours de définition). La spécification "EncodingObjectOptionalitySpec", si présente, spécifie que le champ peut ne pas être spécifié dans une définition d'objet de codage, ou, dans le cas **`DEFAULT`**, que son omission produit l'objet "EncodingObject" suivant (voir § 17.1.5 de la présente Recommandation | Norme internationale) qui doit être de la classe "DefinedOrBuiltinEncodingClass".

B.7 Spécification du champ "objet de codage de classe variable"

Une spécification "VariableClassEncodingObjectFieldSpec" spécifie que le champ contient un objet de codage de classe variable (voir § B.1 de la présente Recommandation | Norme internationale):

```
VariableClassEncodingObjectFieldSpec ::=
        objectfieldreference
        encodingclassfieldreference
        EncodingObjectOptionalitySpec?
```

Le nom du champ est "objectfieldreference". La référence "encodingclassfieldreference" renvoie à un champ de classe de codage de la classe de codage spécifiée. La spécification "EncodingObjectOptionalitySpec", si présente, spécifie que l'objet de codage peut être omis dans une définition d'objet de codage, ou, dans le cas **DEFAULT**, que son omission produit l'objet "EncodingObject" suivant. La spécification "EncodingObjectOptionalitySpec" doit être telle que:

a) si le champ de type indiqué par la référence "encodingclassfieldreference" a une spécification "EncodingClassOptionalitySpec" de valeur **OPTIONAL**, dans ce cas la spécification "EncodingObjectOptionalitySpec" doit également être de valeur **OPTIONAL**; et

b) si la spécification "EncodingObjectOptionalitySpec" a la valeur "**DEFAULT** EncodingObject", dans ce cas le champ de classe de codage indiqué par la référence "encodingclassfieldreference" doit avoir une spécification "EncodingClassOptionalitySpec" de classe " **DEFAULT** DefinedOrBuiltinEncodingClass", et l'objet "EncodingObject" doit être un objet de codage de cette classe.

B.8 Spécification du champ "ensemble d'objets de codage de classe fixe"

NOTE – Cette modification est introduite pour le seul objectif de l'article 23.

Une spécification "FixedClassEncodingObjectSetFieldSpec" spécifie que le champ contient un ensemble d'objets de codage de classe fixe (voir § B.1 de la présente Recommandation | Norme internationale):

FixedClassEncodingObjectSetFieldSpec ::=
objectsetfieldreference
DefinedOrBuiltinEncodingClass
EncodingObjectSetOptionalitySpec?

EncodingObjectSetOptionalitySpec ::= OPTIONAL | DEFAULT EncodingObjectSet

Le nom du champ est "objectsetfieldreference". La production "DefinedOrBuiltinEncodingClass" fait référence à la classe des objets de codage contenus dans le champ. La production "EncodingObjectSetOptionalitySpec", si présente, spécifie que le champ peut ne pas être spécifié dans une définition d'objet de codage, ou, dans le cas **DEFAULT**, que son omission produit l'ensemble "EncodingObjectSet" suivant (voir l'article 18), objets qui doivent tous être de la classe "DefinedOrBuiltinEncodingClass".

B.9 Spécification du champ "liste ordonnée d'objets de codage de classe fixe"

NOTE – Cette modification est introduite pour le seul objectif de l'article 23.

Une spécification "FixedClassOrderedEncodingObjectListFieldSpec" spécifie que le champ contient une liste ordonnée d'objets de codage de classe fixe (voir § B.1 de la présente Recommandation | Norme internationale):

FixedClassOrderedEncodingObjectListFieldSpec ::=
orderedencodingobjectlistfieldreference
DefinedOrBuiltinEncodingClass
ORDERED
OrderedEncodingObjectListOptionalitySpec?

OrderedEncodingObjectListOptionalitySpec ::= OPTIONAL | DEFAULT OrderedEncodingObjectList

Le nom du champ est "orderedencodingobjectlistfieldreference". La production "DefinedOrBuiltinEncodingClass" fait référence à la classe des objets de codage contenus dans le champ. La production "OrderedEncodingObjectListOptionalitySpec", si présente, spécifie que le champ peut ne pas être spécifié dans une définition d'objet de codage, ou, dans le cas **DEFAULT**, que son omission produit la liste d'objets "OrderedEncodingObjectList" suivante (voir § B.11 de la présente Recommandation | Norme internationale), objets qui doivent tous être de la classe "DefinedOrBuiltinEncodingClass".

B.10 Spécification du champ de classe de codage

NOTE – Cette modification est introduite pour le seul objectif de l'article 23.

Une spécification "EncodingClassFieldSpec" spécifie que le champ est un champ de classe de codage (voir § B.1 de la présente Recommandation | Norme internationale):

EncodingClassFieldSpec ::=
encodingclassfieldreference
EncodingClassOptionalitySpec?

EncodingClassOptionalitySpec ::= OPTIONAL | DEFAULT DefinedOrBuiltinEncodingClass

Le nom du champ est "encodingclassfieldreference". Si la spécification "EncodingClassOptionalitySpec" est absente, toutes les définitions d'objet de codage pour cette classe doivent comprendre une spécification d'une classe de codage

pour ce champ. Si le mot **OPTIONAL** est présent, dans ce cas le champ peut être laissé indéfini. Si **DEFAULT** est présent, dans ce cas la spécification "DefinedOrBuiltinEncodingClass" suivante offre le réglage par défaut pour le champ si cela est omis dans une définition.

B.11 Notation de liste ordonnée de valeurs

OrderedValueList ::= "{" Value "," + "}"

La production "OrderedValueList" est une liste ordonnée d'une ou de plusieurs valeurs du type gouverneur. Elle est utilisée lorsque l'application applique la sémantique à l'ordre des valeurs dans la liste.

NOTE – Une liste de valeurs ne peut être spécifiée par notation en ligne (qui est gouvernée par un champ de type, par un champ d'ensemble de valeurs de type fixe, ou par un champ de liste ordonnée de valeurs de type fixe).

B.12 Notation de liste ordonnée d'objets de codage

OrderedEncodingObjectList ::= "{" EncodingObject "," + "}"

La production "OrderedEncodingObjectList" est une liste ordonnée d'un ou de plusieurs objets de codage de la classe gouvernante. Elle est utilisée lorsque l'application applique la sémantique à l'ordre des objets de codage dans la liste.

Exemple: Une liste d'objets de codage de classe **#TRANSFORM** est appliquée dans l'ordre indiqué.

NOTE – Les restrictions suivantes proviennent du texte normatif et des productions BNF: une liste ordonnée d'objets de codage ne peut être spécifiée par notation en ligne (qui est gouvernée par un champ de liste ordonnée d'objets de codage); les objets de codage contenus dans cette liste peuvent être spécifiés au moyen d'un nom de référence ou d'une notation en ligne; le gouverneur ne peut pas être de la classe **#ENCODINGS**.

B.13 Noms de champ primitif

La Rec. UIT-T X.681 | ISO/CEI 8824-2, § 9.13, est modifiée comme suit:

9.13 La construction "PrimitiveFieldName" est utilisée afin de désigner un champ relatif à la classe de codage contenant sa spécification:

PrimitiveFieldName ::=
valuefieldreference
| **valuesetfieldreference**
| **orderedvaluelistfieldreference**

B.14 Mots réservés additionnels

Les § 10.6 et 10.7 de la Rec. UIT-T X.681 | ISO/CEI 8824-2 sont modifiés comme suit:

10.6 Un item lexical "word", utilisé comme "Literal" ne peut pas être un des suivants:

BEGIN	**MINUS-INFINITY**	**PER-CANONICAL-UNALIGNED**
BER	**NON-ECN-BEGIN**	**PLUS-INFINITY**
CER	**NULL**	**TRUE**
DER	**OPTIONS**	**UNION**
ENCODE	**OUTER**	**USE**
ENCODE-DECODE	**PER-BASIC-ALIGNED**	**USE-SET**
END	**PER-BASIC-UNALIGNED**	
FALSE	**PER-CANONICAL-UNALIGNED**	

NOTE – Cette liste contient seulement les mots réservés ASN.1 qui peuvent apparaître comme premier item d'une spécification "Value", "EncodingObject", ou "EncodingObjectSet", ainsi que le mot réservé **END**. L'utilisation d'autres mots réservés ECN ne donne pas lieu à ambiguïté et est permise. Lorsque la syntaxe définie est utilisée dans un environnement dans lequel un mot "word" est également une référence "encodingobjectsetreference", l'utilisation en tant que mot "word" a priorité.

10.7 Un "Literal" spécifie l'inclusion réelle de ce littéral "Literal", qui est doit être un mot "word", à cette position dans la syntaxe définie.

B.15 Définition d'objets de codage

La restriction imposée par la Rec. UIT-T X.681 | ISO/CEI 8824-2, § 10.12.d, est supprimée.

NOTE – Cela affecte la syntaxe définie pour la définition des objets de codage de certaines classes (voir les articles 23 et 24). Cela signifie, par exemple, que, pour une syntaxe définie telle que:

```
[BOOL-TO-INT [AS &bool-to-int]]
```

l'utilisateur est autorisé à écrire:

```
BOOL-TO-INT
```

lors de la définition d'un objet de codage de cette classe. Dans un tel cas, la valeur **DEFAULT** associée au paramètre "**&bool-to-int**" (c'est-à-dire, "**false-zero**") est utilisée dans la définition de la transformée "**BOOL-TO-INT**".

B.16 Compléments à "Setting"

La Rec. UIT-T X.681 | ISO/CEI 8824-2, § 11.7, est modifiée comme suit:

11.7 Un élément "Setting" spécifie le réglage de certains champs à l'intérieur d'un objet de codage à définir:

```
Setting ::=
        Value
      | ValueSet
      | OrderedValueList
      | EncodingObject
      | EncodingObjectSet
      | OrderedEncodingObjectList
      | DefinedOrBuiltinEncodingClass
      | OUTER
```

Si le champ est:

a) un champ de valeur, l'option "Value";

b) un champ d'ensemble de valeurs de type fixe, l'option "ValueSet";

c) un champ de liste ordonnée de valeurs de type fixe, l'option "OrderedValueList";

d) un champ d'objet de codage, l'option "EncodingObject";

e) un champ d'ensemble d'objets de codage, l'option "EncodingObjectSet";

f) un champ de liste ordonnée d'objets de codage, l'option "OrderedEncodingObjectList";

g) un champ de classe de codage, l'option "DefinedOrBuiltinEncodingClass";

h) un champ de référence, l'option "Value" ou **OUTER**

doit être sélectionnée. Pour un champ de référence spécifié au moyen de la syntaxe des articles 20 à 25, le champ "Value" doit être un paramètre fictif. L'option **OUTER** peut être utilisée chaque fois qu'une référence est requise et identifie un conteneur qui est le codage entier.

NOTE – Le réglage est encore restreint comme décrit dans la Rec. UIT-T X.681 | ISO/CEI 8824-2, § 9.5 à 9.12, et § 11.8 à 11.9.

B.17 Type de champ de classe de codage

Le type qui est cité en référence par cette notation dépend de la catégorie du nom de champ. Pour les différents catégories de nom de champ, les § B.17.2 à B.17.4 ci-dessous spécifient le type qui est cité en référence.

B.17.1 La notation pour un type de champ de classe de codage doit être "EncodingClassFieldType":

```
EncodingClassFieldType ::=
        DefinedOrBuiltinEncodingClass
        "."
        FieldName
```

où la production "FieldName" est comme spécifié dans la Rec. UIT-T X.681 | ISO/CEI 8824-2, § 9.14, relative à la classe de codage identifiée par la spécification "DefinedOrBuiltinEncodingClass".

B.17.2 Pour une valeur de type fixe, pour un champ d'ensemble de valeurs de type fixe, ou pour un champ de liste ordonnée de valeurs de type fixe, la notation indique le "Type" qui apparaît dans la spécification de ce champ dans la définition de la classe de l'objet de codage.

B.17.3 Cette notation n'est pas permise si le champ contient un objet de codage, un ensemble d'objets de codage ou un champ de liste ordonnée d'objets de codage.

B.17.4 La notation visant à définir une valeur de ce type doit être "FixedTypeFieldVal" telle que définie dans la Rec. UIT-T X.681 | ISO/CEI 8824-2, § 14.6.

Annexe C

Addendum à la Rec. UIT-T X.683 | ISO/CEI 8824-4

(Cette annexe fait partie intégrante de la présente Recommandation | Norme internationale)

La présente annexe spécifie les modifications qui doivent être appliquées lorsque des productions et/ou des clauses issues de la Rec. UIT-T X.683 | ISO/CEI 8824-4 sont citées en référence dans la présente Recommandation | Norme internationale.

C.1 Attributions paramétrées

Les § 8.1 et 8.3 de la Rec. UIT-T X.683 | ISO/CEI 8824-4 sont modifiés comme suit:

8.1 Il y a des instructions d'attribution paramétrée correspondant à chacune des instructions d'attribution spécifiées dans la présente Recommandation | Norme internationale. La construction "ParameterizedAssignment" est la suivante:

```
ParameterizedAssignment ::=
        ParameterizedEncodingObjectAssignment
|       ParameterizedEncodingClassAssignment
|       ParameterizedEncodingObjectSetAssignment
```

8.3 **ParameterList ::= "{<" Parameter "," + ">}"**

```
Governor ::=
            EncodingClassFieldType
        |   REFERENCE
        |   DefinedOrBuiltinEncodingClass
        |   #ENCODINGS
```

Une référence "DummyReference" contenue dans un paramètre "Parameter" peut représenter:

a) une classe de codage, auquel cas il ne doit y avoir aucune construction "ParamGovernor";

b) une valeur ASN.1, un ensemble de valeurs, ou une liste ordonnée de valeurs de type fixe, auquel cas la construction "ParamGovernor" doit être présente en tant que gouverneur "Governor" qui est un type extrait d'une classe de codage ("EncodingClassFieldType");

c) un identificateur "identifier", auquel cas la construction "ParamGovernor" doit être présente en tant que gouverneur "Governor" qui est une référence **`REFERENCE`**;

d) un objet de codage, ou une liste ordonnée d'objets de codage, auquel cas la construction "ParamGovernor" doit être présente en tant que gouverneur "Governor" qui est une classe de codage ("DefinedOrBuiltinEncodingClass");

e) un ensemble d'objets de codage, auquel cas la construction "ParamGovernor" doit être présente en tant que gouverneur "Governor" qui est la classe "**`#ENCODINGS`**".

NOTE – Les constructions "DummyGovernor" ne sont pas permises en notation ECN.

C.2 Attributions de codage paramétrées

Les productions suivantes sont ajoutées à la Rec. UIT-T X.683 | ISO/CEI 8824-4, § 8.2:

```
ParameterizedEncodingClassAssignment ::=
        encodingclassreference
        ParameterList
        "::="
        EncodingClass

ParameterizedEncodingObjectAssignment ::=
        encodingobjectreference
        ParameterList
        DefinedOrBuiltinEncodingClass
        "::="
        EncodingObject

ParameterizedEncodingObjectSetAssignment ::=
        encodingobjectsetreference
        ParameterList
```

#ENCODINGS
"::="
EncodingObjectSet

La Rec. UIT-T X.683 | ISO/CEI 8824-4, § 8.4, est modifiée comme suit:

8.4 Le domaine d'application d'une référence "DummyReference" apparaissant dans une liste "ParameterList" est la liste "ParameterList" elle-même, ainsi qu'une partie du champ "ParameterizedAssignment" qui suit la notation "**::=**". Dans le cas d'une construction "ParameterizedEncodingObjectAssignment", le domaine d'application s'étend jusqu'à la classe "DefinedOrBuiltinEncodingClass" qui précède la notation "**::=**". La production "DummyReference" masque toute autre "Reference" ayant le même nom dans ce domaine d'application.

NOTE – Le cas particulier de l'attribution "ParameterizedEncodingObjectAssignment" est destiné à être utilisé en association avec les clauses de renommage (voir § D.3.3.3). Cette construction autorise à écrire une attribution telle que la suivante, dans laquelle le paramètre fictif "**#Any-Class**" de l'objet de codage "**new-component-encoding**" est utilisé comme paramètre réel pour la classe de codage "**#New-component**":

```
new-component-encoding {< #Any-class >} #New-component {< #Any-class >} ::=
    { -- encoding object definition -- }
```

C.3 Référence à des définitions paramétrées

La production "ParameterizedReference" de la Rec. UIT-T X.683 | ISO/CEI 8824-4, § 9.1, est modifiée comme suit:

ParameterizedReference ::=
Reference
| **Reference "{<" ">}"**

Les productions suivantes sont ajoutées à la Rec. UIT-T X.683 | ISO/CEI 8824-4, § 9.2:

ParameterizedEncodingObject ::=
SimpleDefinedEncodingObject
ActualParameterList

SimpleDefinedEncodingObject ::=
ExternalEncodingObjectReference
| **encodingobjectreference**

ParameterizedEncodingObjectSet ::=
SimpleDefinedEncodingObjectSet
ActualParameterList

SimpleDefinedEncodingObjectSet ::=
ExternalEncodingObjectSetReference
| **encodingobjectsetreference**

ParameterizedEncodingClass ::=
SimpleDefinedEncodingClass
ActualParameterList

SimpleDefinedEncodingClass ::=
ExternalEncodingClassReference
| **encodingclassreference**

C.4 Liste des paramètres réels

La Rec. UIT-T X.683 | ISO/CEI 8824-4, § 9.5, est modifiée comme suit:

9.5 La production "ActualParameterList" est le suivant:

ActualParameterList ::=
"{<" ActualParameter "," + ">}"

ActualParameter ::=
Value
| **ValueSet**
| **OrderedValueList**
| **DefinedOrBuiltinEncodingClass**
| **EncodingObject**
| **EncodingObjectSet**
| **OrderedEncodingObjectList**
| **identifier**
| **STRUCTURE**
| **OUTER**

Si le paramètre fictif correspondant est:

a) une valeur, l'option "Value";

b) un ensemble de valeurs, l'option "ValueSet";

c) une liste ordonnée de valeurs de type fixe, l'option "OrderedValueList";

d) une classe de codage, l'option "DefinedOrBuiltinEncodingClass";

e) un objet de codage, l'option "EncodingObject";

f) un ensemble d'objets de codage, l'option "EncodingObjectSet";

g) une liste ordonnée d'objets de codage, l'option "OrderedEncodingObjectList";

h) une référence, l'option "identifier", **`STRUCTURE`** ou **`OUTER`**

doit être sélectionnée. L'option **`STRUCTURE`** ne doit être sélectionnée que lorsque le paramètre réel est utilisé comme spécifié au § 17.5.15. L'option **`OUTER`** peut être utilisée chaque fois qu'une référence est requise afin de désigner un conteneur, et identifie le conteneur du codage entier.

Annexe D

Exemples

(Cette annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

La présente annexe contient des exemples de l'utilisation de la notation ECN. Les exemples sont subdivisés en cinq groupes:

- exemples généraux, qui montrent l'allure générale des définitions en notation ECN (§ D.1);
- exemples de spécialisation, qui montrent la façon de modifier certaines parties d'un codage normalisé. Chaque exemple comporte une description des exigences de codage et une description de la solution sélectionnée et des solutions offertes en variante (§ D.2);
- exemples de structure produite explicitement, qui montrent l'utilisation de structures produites explicitement lorsque le même codage spécialisé est utilisé plusieurs fois (§ D.2.15);
- un exemple de protocole existant qui montre trois façons de traiter le problème d'une approche traditionnelle par "bit d'extension" pour une terminaison de type séquence d'items (§ D.3.8);
- un deuxième exemple de protocole existant, qui montre la façon de construire des définitions ECN pour un protocole dont les codages de message ont été spécifiés au moyen d'une notation tabulaire (§ D.5).

D.1 Exemples généraux

Les exemples décrits dans les § D.1.1 à D.1.14 font partie d'une spécification ECN complète dont les modules ASN.1, EDM, et ELM sont présentés dans leurs grandes lignes aux § D.1.15, D.1.16 et D.1.17 et sont exposés complètement dans une version de cette annexe qui est disponible à partir du site Web cité en Annexe F.

D.1.1 Un objet de codage pour un type booléen

D.1.1.1 L'attribution ASN.1 est la suivante:

```
Married ::= BOOLEAN
```

D.1.1.2 L'attribution d'objet de codage (voir § 23.3.1) est la suivante:

```
booleanEncoding #BOOLEAN ::= {
        ENCODING-SPACE
                SIZE 1
                MULTIPLE OF bit
         TRUE-PATTERN bits:'1'B
         FALSE-PATTERN bits:'0'B}

marriedEncoding-1 #Married ::= booleanEncoding
```

D.1.1.3 Il n'y a pas de préalignement et l'espace de codage est d'un bit, de sorte que "`Married`" est codé en tant que champ binaire de longueur 1. Les séquences pour les valeurs `TRUE` et `FALSE` (dans ce cas un bit isolé) sont '1'B et '0'B respectivement.

D.1.1.4 Les valeurs spécifiées ci-dessus sont celles qui seraient fixées par défaut (voir § 23.3.1) si les propriétés de codage correspondantes étaient omises, de sorte que le même codage peut être réalisé avec moins de verbialité comme suit:

```
marriedEncoding-2 #Married ::= {
        ENCODING-SPACE
                SIZE 1}
```

D.1.1.5 Ce codage pour un booléen est, évidemment, exactement ce que les règles PER offrent et une autre option vise à spécifier le codage au moyen de l'objet de codage PER de booléens par la syntaxe fournie par le § 17.3.1.

```
marriedEncoding-3 #Married ::= {
        ENCODE WITH PER-BASIC-UNALIGNED}
```

D.1.1.6 Comme le montrent ces exemples, il y a souvent des cas où la notation ECN offre plusieurs façons de définir un codage. C'est à l'utilisateur qu'il appartient de choisir l'option à utiliser, en mettant la verbialité (indiquant explicitement les valeurs qui peuvent être prises par défaut) en regard de la lisibilité et de la clarté.

D.1.2 Un objet de codage pour un type d'entier

D.1.2.1 Les attributions ASN.1 sont les suivantes:

```
EvenPositiveInteger ::= INTEGER (1..MAX) (CONSTRAINED  BY {-- doit être pair --})

EvenNegativeInteger ::= INTEGER (MIN..-1) (CONSTRAINED BY {-- doit être pair --})
```

D.1.2.2 Les attributions d'objet de codage sont les suivantes:

```
evenPositiveIntegerEncoding #EvenPositiveInteger ::= {
        USE #NonNegativeInt
MAPPING TRANSFORMS {{INT-TO-INT divide:2}}
        WITH PER-BASIC-UNALIGNED}

#NonNegativeInt ::= #INT(0..MAX)

evenNegativeIntegerEncoding #EvenNegativeInteger ::= {
        USE #NonPositiveInt
MAPPING TRANSFORMS {{INT-TO-INT divide:2
                                    -- Note: -1 / 2 = 0 - voir § 24.3.6 -- }}
        WITH PER-BASIC-UNALIGNED}

#NonPositiveInt ::= #INT(MIN..0)
```

D.1.2.3 Une valeur paire est divisée par deux et est ensuite codée au moyen des règles de codage PER normalisées pour les types d'entier positifs et négatifs.

D.1.3 Un autre objet de codage pour un type d'entier

D.1.3.1 Il s'agit ici de l'exigence visant à définir un objet de codage qui code un entier dans un champ de deux octets commençant à une limite d'octet.

D.1.3.2 L'attribution ASN.1 est la suivante:

```
Altitude ::= INTEGER (0..65535)
```

D.1.3.3 L'attribution d'objet de codage (voir § 23.6.1 et 23.7.1) est la suivante:

```
integerRightAlignedEncoding #Altitude ::= {
        ENCODING {
            ALIGNED TO NEXT octet
            ENCODING-SPACE
                    SIZE 16}}
```

D.1.4 Un objet de codage pour un type d'entier avec trous

D.1.4.1 L'attribution ASN.1 est la suivante:

```
IntegerWithHole ::= INTEGER (-256..-1 | 32..1056)
```

D.1.4.2 L'attribution d'objet de codage (voir § 19.5.2) est la suivante:

```
integerWithHoleEncoding #IntegerWithHole ::= {
        USE #IntFrom0To1280
        MAPPING ORDERED VALUES
        WITH PER-BASIC-UNALIGNED}

#IntFrom0To1280 ::= #INT (0..1280)
```

D.1.4.3 La production "`IntegerWithHole`" est codée en tant qu'entier positif. Les valeurs dans l'étendue -256..-1 sont mappées sur les valeurs dans l'étendue 0..255 et les valeurs dans l'étendue 32..1056 sont mappées sur 256..1280.

D.1.5 Un objet de codage plus complexe pour un type d'entier

D.1.5.1 Les attributions ASN.1 sont les suivantes:

```
PositiveInteger ::= INTEGER (1..MAX)

NegativeInteger ::= INTEGER (MIN..-1)
```

D.1.5.2 Les attributions d'objet de codage sont les suivantes:

```
positiveIntegerEncoding #PositiveInteger ::=
        integerEncoding
```

```
negativeIntegerEncoding #NegativeInteger ::=
        integerEncoding
```

D.1.5.3 Les valeurs des types "**`PositiveInteger`**" et "**`NegativeInteger`**" sont codées par l'objet de codage "**`integerEncoding`**" en tant qu'entier positif ou en tant qu'entier en complément à deux respectivement. Elles sont définies ci-dessous et offrent différents codages selon les limites du type auquel elles sont appliquées.

D.1.5.4 L'objet de codage "**`integerEncoding`**" défini ici est très puissant, mais tout à fait complexe. Il contient cinq objets de codage de la classe **`#CONDITIONAL-INT`** qui tous définissent un codage de **type aligné** par les octets. Lorsque les valeurs entières en cours de codage sont limitées, le nombre de bits est fixe; lorsque les valeurs ne sont pas limitées, le type doit être le dernier dans une unité PDU et la valeur est justifiée à droite dans les octets restants de l'unité PDU.

D.1.5.5 La définition de l'objet de codage (voir § 23.6.1 et 23.7.1) est la suivante:

```
integerEncoding #INT ::= {ENCODINGS {
        { IF unbounded-or-no-lower-bound
            ENCODING-SPACE
                    SIZE variable-with-determinant
                    DETERMINED BY container
                    USING OUTER
            ENCODING twos-complement} ,
        { IF bounded-with-negatives
            ENCODING-SPACE
                    SIZE fixed-to-max
            ENCODING twos-complement} ,
        { IF semi-bounded-with-negatives
            ENCODING-SPACE
                    SIZE variable-with-determinant
                    DETERMINED BY container
                    USING OUTER
            ENCODING twos-complement} ,
        { IF semi-bounded-without-negatives
            ENCODING-SPACE
                    SIZE variable-with-determinant
                    DETERMINED BY container
                    USING OUTER
            ENCODING positive-int} ,
        { IF bounded-without-negatives
            ENCODING-SPACE
                    SIZE fixed-to-max
            ENCODING positive-int}}}
```

D.1.6 Entiers positifs codés en BCD

D.1.6.1 Cet exemple montre la façon de coder un entier positif en codage binaire décimal (BCD, *binary coded decimal*) par des transformées successives: d'un entier à une chaîne de caractères, puis d'une chaîne de caractères à une chaîne de bits.

D.1.6.2 L'attribution ASN.1 est la suivante:

```
PositiveIntegerBCD ::= INTEGER(0..MAX)
```

D.1.6.3 L'attribution d'objet de codage (voir § 19.4, 24.1 et 23.4.1) est la suivante:

```
positiveIntegerBCDEncoding #PositiveIntegerBCD ::= {
        USE #CHARS
        MAPPING TRANSFORMS{{
                INT-TO-CHARS
                    -- L'on convertit en caractères (par exemple, l'entier 42
                    -- devient la chaîne de caractères "42") et l'on code les caractères
                    -- avec l'objet de codage "numeric-chars-to-bcdEncoding"
                        SIZE variable
                        PLUS-SIGN FALSE}}
        WITH numeric-chars-to-bcdEncoding }

numeric-chars-to-bcdEncoding #CHARS ::= {
        ALIGNED TO NEXT nibble
                TRANSFORMS {{
                        CHAR-TO-BITS
                            -- L'on convertit chaque caractère en une chaîne de bits
                            -- (par exemple, le caractère "4" devient '0100'B et "2" devient '0010'B)
```

```
                    AS mapped
                    CHAR-LIST { "0","1","2","3",
                                "4","5","6","7",
                                "8","9"}
                    BITS-LIST { '0000'B, '0001'B, '0010'B, '0011'B,
                                '0100'B, '0101'B, '0110'B, '0111'B,
                                '1000'B, '1001'B }}}
        REPETITION-ENCODING {
            REPETITION-SPACE
            -- Nous déterminons la concaténation des chaînes de bits pour les
            -- caractères et ajoutons un caractère de fin de chaîne (par exemple,
            -- '0100'B + '0010'B devient '0100 0010 1111'B)
                    SIZE variable-with-determinant
                    DETERMINED BY pattern
          PATTERN bits:'1111'B}}
```

D.1.6.4 Le nombre positif est d'abord transformé en une chaîne de caractères par la transformée "int-to-chars" au moyen des options de longueur variable et d'absence de signe plus, et par ailleurs de l'option par défaut d'absence de bourrage, qui donne une chaîne contenant les caractères "`0`" à "`9`". Puis la chaîne de caractères est codée de façon que chaque caractère soit transformé en séquence binaire: `'0000'B` pour "`0`", `'0001'B` pour "`1`"…, `'1001'B` pour "`9`". La chaîne de bits est alignée sur une limite de quartet et se termine avec une séquence spécifique '1111'B.

D.1.6.5 Une option plus complexe, non indiquée ici, mais généralement utilisée, consisterait à imbriquer le codage BCD dans une chaîne d'octets, avec un booléen externe précisant s'il y a un quartet inutilisé à l'extrémité ou non.

D.1.7 Un objet de codage de classe `#BITS`

D.1.7.1 Cet exemple définit un objet de codage de classe **`#BITS`** (voir § 23.2.1) pour une chaîne de bits qui est alignée par les octets, bourrée avec 0 et terminée par un champ de 8 bits contenant `'00000000'B` (il est admis qu'une valeur abstraite ne contient jamais huit zéros successifs):

D.1.7.2 L'attribution ASN.1 est la suivante:

```
Fax ::= BIT STRING (CONSTRAINED BY {-- ne doit jamais contenir huit bits zéro successifs --})
```

D.1.7.3 L'attribution d'objet de codage (voir § 23.2.1, 23.12.1 et 23.13.1) est la suivante:

```
faxEncoding #Fax ::= {
        ALIGNED TO NEXT octet
        REPETITION-ENCODING {
            REPETITION-SPACE
                    SIZE variable-with-determinant
                    DETERMINED BY pattern
                PATTERN bits:'00000000'B}}
```

D.1.7.4 Cet objet de codage (de classe **`#BITS`**) contient un objet de codage imbriqué de classe **`#CONDITIONAL-REPETITION`** qui spécifie le mécanisme et la séquence de terminaison.

D.1.7.5 Comme avec n'importe lequel des exemples dans cette annexe, il y a ici une dépendance étroite avec les valeurs par défaut fournies dans l'article 23, et avantage est tiré de la capacité afin de définir des objets de codage en ligne plutôt que de leur attribuer séparément des noms de référence qui seront ensuite utilisés dans d'autres attributions.

D.1.8 Un objet de codage pour une chaîne d'octets type

D.1.8.1 L'attribution ASN.1 est la suivante:

```
BinaryFile ::= OCTET STRING
```

D.1.8.2 L'attribution d'objet de codage (voir § 23.9.1) est la suivante:

```
binaryFileEncoding #BinaryFile ::= {
        ALIGNED TO NEXT octet
                PADDING one
        REPETITION-ENCODING {
            REPETITION-SPACE
                    SIZE variable-with-determinant
                DETERMINED BY container
                        USING OUTER}}
```

D.1.8.3 La valeur est alignée par les octets au moyen du bourrage avec des unités et se termine par l'extrémité de l'unité PDU.

D.1.9 Un objet de codage pour une chaîne de caractères type

D.1.9.1 L'attribution ASN.1 est la suivante:

```
Password ::= PrintableString
```

D.1.9.2 L'attribution d'objet de codage (voir § 23.4.1 et 23.13.1) est la suivante:

```
passwordEncoding #Password ::= {
        ALIGNED TO NEXT octet
                TRANSFORMS {{CHAR-TO-BITS
                                        AS compact
                                        SIZE fixed-to-max
                                        MULTIPLE OF bit }}
        REPETITION-ENCODING {
            REPETITION-SPACE
                SIZE variable-with-determinant
                DETERMINED BY container
                        USING OUTER}}
```

D.1.9.3 La chaîne est alignée par les octets au moyen du bourrage avec "0" et se termine par l'extrémité de l'unité PDU; le codage de caractères est spécifié en tant que "`compact`", de sorte que chaque caractère est codé sur 7 bits au moyen de `'0000000'B` pour le premier caractère ASCII de type `PrintableString`, de `'0000001'B` pour la suivante, et ainsi de suite.

D.1.10 Mappage de valeurs de caractère sur des valeurs binaires

D.1.10.1 L'attribution ASN.1 est la suivante:

```
CharacterStringToBit ::= IA5String ("FIRST" | "SECOND" | "THIRD")
```

D.1.10.2 L'attribution d'objet de codage (voir § 19.2) est la suivante:

```
characterStringToBitEncoding #CharacterStringToBit ::= {
        USE #IntFrom0To2
        MAPPING VALUES {
            "FIRST"   TO 0,
            "SECOND"    TO 1,
            "THIRD"  TO 2}
        WITH integerEncoding}

#IntFrom0To2 ::= #INT (0..2)
```

où "integerEncoding" est défini au § D.1.5.5.

D.1.10.3 Les trois valeurs abstraites possibles sont mappées sur trois nombres entiers puis ces nombres sont codés dans un champ de 2 bits.

D.1.11 Un objet de codage pour un type de séquence

D.1.11.1 L'on code ici une séquence type qui possède un champ "`a`" qui contient la sémantique applicative (c'est-à-dire qui est visible à l'application), mais l'on souhaite également l'utiliser en tant que déterminant de présence pour un second (et facultatif) champ d'entier "`b`". Il y a donc une chaîne d'octets qui est alignée par les octets, délimitée par l'extrémité de l'unité PDU. Il est nécessaire de fournir des codages spécialisés pour les offres d'options de "`b`" et l'on utilise le codage spécialisé défini au § D.1.8 (par référence à l'objet de codage "binaryFileEncoding") pour la chaîne d'octets "`c`". L'on souhaite coder tout le reste avec les règles PER de base non alignées.

D.1.11.2 L'attribution ASN.1 est la suivante:

```
Sequence1       ::= SEQUENCE {
        a       BOOLEAN,
        b       INTEGER OPTIONAL,
        c       BinaryFile
                -- "BinaryFile" est défini au § D.1.8.1 --}
```

D.1.11.3 Les attributions ECN (voir § 17.5 et 23.10.1) sont les suivantes:

```
sequence1Encoding #Sequence1 ::= {
        ENCODE STRUCTURE {
            b USE-SET OPTIONAL-ENCODING parameterizedPresenceEncoding {< a >},
            c binaryFileEncoding
            -- "binaryFileEncoding" est défini au § D.1.8.2 -- }
      WITH PER-BASIC-UNALIGNED}
```

```
parameterizedPresenceEncoding {< REFERENCE:reference >}  #OPTIONAL ::= {
        PRESENCE
            DETERMINED BY field-to-be-used
            USING reference}
```

D.1.11.4 Noter qu'il n'est pas nécessaire d'offrir la propriété de codage "**DECODERS-TRANSFORMS**" dans l'objet de codage "**parameterizedPresenceEncoding**", parce que le composant "**a**" était un booléen, et il est admis que **TRUE** indiquait que "**b**" était présent. Si, cependant, "**a**" avait été un champ d'entier, ou si la valeur applicative de **TRUE** pour "**a**" signifiait réellement que "**b**" était absent, dans ce cas l'on aurait inclus une propriété de codage "**DECODER-TRANSFORMS**" comme au § D.2.6**.**

D.1.12 Un objet de codage pour un type "choix"

D.1.12.1 Un type "choix" avec trois options est codé au moyen du numéro d'étiquette du contexte de classe, codé dans un champ de trois bits, en tant que sélecteur. L'objet de codage de classe **#ALTERNATIVES** spécifie que le pointeur d'identification "**Tag**" est utilisé en tant que déterminant; l'objet de codage de classe **#TAG** définit la position du pointeur d'identification (trois bits). Pour chaque option, la valeur est codée avec les règles PER de base non alignées.

D.1.12.2 L'attribution ASN.1 est la suivante:

```
Choice ::= CHOICE {
        boolean    [1] BOOLEAN,
integer [3] INTEGER,
string  [5] IA5String}
```

D.1.12.3 Les attributions ECN (voir § 23.1.1 et 23.14.1) sont les suivantes:

```
choiceEncoding #Choice ::= {
        ENCODE STRUCTURE {
            boolean [tagEncoding] USE-SET,
            integer [tagEncoding] USE-SET,
            string [tagEncoding] USE-SET
            STRUCTURED WITH {
                ALTERNATIVE
                    DETERMINED BY handle
                        HANDLE "Tag"}}
WITH PER-BASIC-UNALIGNED}

tagEncoding #TAG ::= {
        ENCODING-SPACE
            SIZE 3
            MULTIPLE OF bit
        EXHIBITS HANDLE "Tag" AT {0 | 1 | 2}}
```

D.1.12.4 Une façon plus correcte de fournir la première attribution du § D.1.12.3 serait peut-être de définir un nouvel ensemble d'objets de codage et de l'appliquer comme suit:

```
MyEncodings #ENCODINGS ::= { tagEncoding } COMPLETED BY PER-BASIC-UNALIGNED

choiceEncoding #Choice ::= {
        ENCODE STRUCTURE {
            STRUCTURED WITH {
                ALTERNATIVE
                    DETERMINED BY handle
                        HANDLE "Tag"}}
WITH MyEncodings}
```

D.1.13 Codage d'une chaîne de bits contenant un autre codage

D.1.13.1 Une valeur de chaîne de bits codée avec les règles PER de base non alignées contient le codage d'une séquence en tant que nombre entier d'octets (bourré avec des zéros) mais non nécessairement aligné sur une limite d'octet.

D.1.13.2 L'attribution ASN.1 est la suivante:

```
Sequence2 ::= SEQUENCE {
        a   BOOLEAN,
        b   BIT STRING (CONTAINING Sequence3) }

Sequence3 ::= SEQUENCE {
        a   INTEGER(0..10),
        b   BOOLEAN }
```

D.1.13.3 Les attributions ECN (voir § 25.1) sont les suivantes:

```
sequence2Encoding #Sequence2 ::= {
        ENCODE STRUCTURE {
            b    { REPETITION-ENCODING
                     REPETITION-SPACE
                         SIZE 8
                         MULTIPLE OF bit
                             CONTENTS-ENCODING {containerEncoding}
                                 COMPLETED BY PER-BASIC-UNALIGNED}}
        WITH PER-BASIC-UNALIGNED}

containerEncoding #OUTER ::= {
        PADDING
                MULTIPLE OF octet}
```

D.1.14 Un ensemble d'objets de codage

Cet ensemble d'objets de codage contient des définitions de codage pour certains types spécifiés dans le module ASN.1 du § D.1.15.

```
Example1Encodings #ENCODINGS ::= {
                    marriedEncoding-1
                |   integerRightAlignedEncoding
                |   evenPositiveIntegerEncoding
                |   evenNegativeIntegerEncoding
                |   integerRightAlignedEncoding
                |   integerWithHoleEncoding
                |   positiveIntegerEncoding
                |   negativeIntegerEncoding
                |   positiveIntegerBCDEncoding
                |   faxEncoding
                |   binaryFileEncoding
                |   passwordEncoding
                |   characterStringToBitEncoding
                |   sequence1Encoding
                |   choiceEncoding
                |   sequence2Encoding}
```

D.1.15 Définitions ASN.1

D.1.15.1 Ce module ASN.1 regroupe toutes les définitions ASN.1 du § D.1.1 au § D.1.12.4. Elles seront codées conformément aux objets de codage définis dans le module EDM du § D.1.16, ainsi qu'aux règles de codage PER de base non alignées.

Exemple1-ASN1-Module {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module1(2)}

```
DEFINITIONS AUTOMATIC TAGS : :=
BEGIN
MyPDU ::= CHOICE {
        marriedMessage          Married,
        altitudeMessage         Altitude
        -- etc.
}

Married ::= BOOLEAN

Altitude ::= INTEGER (0..65535)

 -- etc.

  END
```

D.1.16 EDM définitions

D.1.16.1 Ce module EDM regroupe toutes les définitions ECN du § D.1.1 au § D.1.12.4.

Example1-EDM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module1(3)}

```
ENCODING-DEFINITIONS ::=
BEGIN
EXPORTS Example1Encodings;
IMPORTS #Married, #Altitude, #EvenPositiveInteger, #EvenNegativeInteger, #IntegerRightAligned,
```

```
        #IntegerWithHole, #PositiveInteger, #NegativeInteger, #PositiveIntegerBCD, #Fax,
        #BinaryFile, #Password, #CharacterStringToBit, #Sequence1, #Choice, #Sequence2
        FROM Example1-ASN1-Module { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module1(2) };

Example1Encodings #ENCODINGS ::= {
        marriedEncoding-1    |
            -- etc
        sequence2Encoding}

-- etc

END
```

D.1.17 ELM définitions

Le module ELM suivant code le module ASN.1 défini au § D.1.15, au moyen des objets spécifiés dans le module EDM défini au § D.1.16.

```
Exemple1-ELM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) elm-module1(1)}

LINK-DEFINITIONS ::=
BEGIN
IMPORTS
        Example1Encodings FROM Example-EDM
                {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module1(3)}
        #MyPDU FROM Example1-ASN1-Module
                {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module1(2)};

ENCODE #MyPDU WITH Example1Encodings
        COMPLETED BY PER-BASIC-UNALIGNED

END
```

D.2 Exemples de spécialisation

Les exemples de ce paragraphe montrent la façon de modifier des parties sélectionnées d'un codage pour des types donnés afin de minimiser la taille de messages codés. Les codages selon les règles PER de base non alignées produisent normalement des codages aussi compacts que possible. Cependant, il y a certains cas où des codages spécialisés pourraient être souhaités:

- une sémantique spéciale est associée à des composants de message qui permettent de supprimer certains des champs auxiliaires produits par les règles PER;
- l'utilisateur souhaite des codages différents pour des champs auxiliaires PER qui sont produits par défaut, tels que les champs de déterminant de longueur variable.

D.2.1 Codage par répartition de valeurs dans une structure de codage à option

D.2.1.1 L'attribution ASN.1 est la suivante:

```
NormallySmallValues ::= INTEGER (0..1000)
-- Habituellement, les valeurs sont dans l'étendue 0..63, mais parfois l'étendue totale des valeurs est utilisée.
```

D.2.1.2 Les règles PER coderaient le type au moyen de 10 bits. L'on souhaite minimiser la taille du codage de façon que le cas normal soit codé au moyen d'aussi peu de bits que possible.

NOTE – Dans cet exemple, l'on adopte une méthode simple et directe. Une approche plus élaborée, utilisant des codages de Huffman, est donnée au § E.1.

D.2.1.3 L'attribution d'objet de codage (voir § 19.6) est la suivante:

```
normallySmallValuesEncoding-1 #NormallySmallValues ::= {
        USE     #NormallySmallValuesStruct
        MAPPING DISTRIBUTION {
                0..63           TO small,
                REMAINDER   TO large }
        WITH PER-BASIC-UNALIGNED}
```

D.2.1.4 L'attribution de structure de codage est la suivante:

```
#NormallySmallValuesStruct ::= #CHOICE {
        small   #INT (0..63),
        large   #INT (64..1000)}
```

D.2.1.5 Les valeurs qui sont normalement utilisées sont codées au moyen du champ "**small**" et celles qui ne sont utilisées qu'occasionnellement sont codées au moyen du champ "**large**". Le choix entre les deux est effectué par un champ sélecteur de 1 bit produit par les règles PER. La longueur du champ "**small**" est de 6 bits et la longueur du champ "**large**" est de 10 bits, de sorte que le cas normal est codé au moyen de 7 bits et que le cas rare est codé au moyen de 11 bits.

D.2.2 Codage par mappage de valeurs abstraites ordonnées dans une structure de codage à option

D.2.2.1 L'exemple D.2.1 utilisait une définition explicite de la façon dont les étendues de valeurs sont mappées sur des champs de la structure de codage. Le même effet peut être obtenu plus simplement au moyen du "mappage par valeurs abstraites ordonnées". Cependant, à titre d'illustration, l'on modifiera également ici l'exigence: des valeurs arbitrairement grandes pourront apparaître occasionnellement et l'attribution ASN.1 est censée avoir ses contraintes supprimées.

D.2.2.2 L'attribution des objets de codage (voir § 19.5) est la suivante:

```
normallySmallValuesEncoding-2 #NormallySmallValues ::= {
        USE    #NormallySmallValuesStruct2
        MAPPING ORDERED VALUES
        WITH      NormallySmallValuesTag-encoding-plus-PER}

normallySmallValuesTag-encoding   #TAG ::= {
        ENCODING-SPACE
                SIZE 1}

NormallySmallValuesTag-encoding-plus-PER #ENCODINGS ::= {normallySmallValuesTag-encoding}
        COMPLETED BY PER-BASIC-UNALIGNED
```

D.2.2.3 L'attribution de structure de codage est la suivante:

```
#NormallySmallValuesStruct2 ::= #CHOICE {
        small   [#TAG(0)] #INT (0..63),
        large   [#TAG(1)] #INT (0..MAX) }
```

D.2.2.4 Le résultat est très semblable au § D.2.1, mais ici les valeurs supérieures à 64 qui sont mappées sur le champ "**large**" sont codées à partir de zéro en ordre croissant. Les deux options sont distinguées par un index d'un bit. Une autre difference est que le champ "**large**" est laissé sans limite, de sorte que l'objet de codage peut coder arbitrairement de grands entiers, mais au prix d'un champ de longueur dans le cas "**large**". Cet exemple peut également être utilisé s'il n'y a pas de limite supérieure applicable aux valeurs qui pourraient occasionnellement apparaître (le champ "**large**" n'est pas limité dans la structure de remplacement). Cela illustre encore la flexibilité offerte aux spécificateurs ECN afin de concevoir des codages adaptés à leurs exigences particulières.

D.2.3 Compression d'étendues de valeurs non continues

D.2.3.1 Cet exemple utilise également un mappage de valeurs abstraites ordonnées. Dans ce cas le mappage sert à comprimer des valeurs dispersées dans une spécification ASN.1 de base. La compression pourrait également avoir été réalisée par définition de la valeur abstraite ASN.1 "x" afin d'obtenir la sémantique applicative de "2x", puis par utilisation d'une contrainte plus simple concernant le type d'entier ASN.1. Dans cet exemple, cependant, l'hypothèse est que le concepteur ASN.1 n'a pas choisi de faire cela et que l'on est contraint d'appliquer la compression pendant le mappage de valeurs abstraites sur des codages.

D.2.3.2 L'attribution ASN.1 est la suivante:

```
SparseEvenlyDistributedValueSet ::= INTEGER (2 | 4 | 6 | 8 | 10 | 12 | 14 | 16)
```

D.2.3.3 Les règles PER de base non alignées ne prennent en compte que les limites inférieures et supérieures lors de la détermination du nombre de bits nécessaire pour coder un entier. Cela aboutit à des séquences binaires inutilisées dans le codage. Celui-ci peut être comprimé de façon que les séquences binaires inutilisées soient omises et que chaque valeur soit codée au moyen du nombre minimal de bits.

D.2.3.4 L'attribution d'objet de codage (voir § 19.5) est la suivante:

```
sparseEvenlyDistributedValueSetEncoding #SparseEvenlyDistributedValueSet ::= {
        USE    #IntFrom0To7
        MAPPING ORDERED VALUES
        WITH PER-BASIC-UNALIGNED}

#IntFrom0To7 ::= #INT (0..7)
```

D.2.3.5 Les huit valeurs abstraites possibles ont été mappées sur l'étendue **0..7** et seront codées dans un champ de trois bits.

D.2.4 Compression d'étendues de valeurs discontinues au moyen d'une transformée

D.2.4.1 L'exemple du § D.2.3 utilisait le mappage de valeurs abstraites ordonnées. Le même effet peut être obtenu au moyen de la classe **`#TRANSFORM`**.

D.2.4.2 L'attribution d'objet de codage (voir § 19.4) est la suivante:

```
sparseEvenlyDistributedValueSetEncoding-2 #SparseEvenlyDistributedValueSet ::= {
        USE    #IntFrom0To7
        MAPPING TRANSFORMS {{INT-TO-INT divide: 2}, {INT-TO-INT decrement:1}}
        WITH PER-BASIC-UNALIGNED}
```

D.2.4.3 De nouveau, les huit valeurs abstraites possibles sont mappées sur l'étendue **`0..7`** et codées dans un champ de trois bits.

D.2.5 Compression d'un ensemble de valeurs non uniformément réparties par mappage de valeurs abstraites ordonnées

D.2.5.1 L'attribution ASN.1 est la suivante:

```
SparseUnevenlyDistributedValueSet ::= INTEGER (0|3|5|6|11|8)
-- L'absence d'ordre vise à illustrer le fait que l'ordre n'a pas d'importance dans la contrainte
```

D.2.5.2 Le codage devrait être tel qu'il n'y ait pas de trous dans les séquences de codage utilisées.

D.2.5.3 L'attribution d'objet de codage est la suivante:

```
sparseUnevenlyDistributedValueSetEncoding #SparseUnevenlyDistributedValueSet ::= {
        USE    #IntFrom0To5
        MAPPING ORDERED VALUES
        WITH PER-BASIC-UNALIGNED}

#IntFrom0To5 ::= #INT (0..5)
```

D.2.5.4 Les six valeurs abstraites possibles sont mappées sur l'étendue **`0..5`** et codées dans un champ de trois bits. Le mappage est comme suit: 0→0 , 3→1, 5→2, 6→3, 8→4 et 11→5.

D.2.6 Présence d'un composant facultatif selon la valeur d'un autre composant

D.2.6.1 L'attribution ASN.1 est la suivante:

```
ConditionalPresenceOnValue ::= SEQUENCE {
        a   INTEGER (0..4),
        b   INTEGER (1..10),
        c   BOOLEAN OPTIONAL
                -- Condition: "c" est présent si "a" est 0; sinon "c" est absent --,
        d   BOOLEAN OPTIONAL
                -- Condition: "d" est absent si "a" est 1; sinon "d" est présent -- }
        -- Noter les contraintes de présence impliquées dans les commentaires.
        -- Noter également que le champ d'entier "a" achemine la sémantique applicative et a des valeurs
        -- autres que zéro et un. Si "a" a la valeur 0, aussi bien "c" que "d" sont présents. Si "a" a
        -- la valeur 1, aussi bien "c" que "d" sont absents. Si "a" a une valeur 3 ou 4, "c" est absent
        -- et "d" est présent. Ces conditions sont très difficiles à exprimer formellement au moyen
        -- de la seule notation ASN.1.
```

D.2.6.2 Le composant "**`a`**" agit en tant que déterminant de présence pour les deux composants "**`c`**" et "**`d`**", mais un codage PER produirait deux bits auxiliaires pour les composants facultatifs. L'on a besoin d'un codage dans lequel ces bits auxiliaires sont absents.

D.2.6.3 L'attribution d'objet de codage est la suivante:

```
conditionalPresenceOnValueEncoding #ConditionalPresenceOnValue ::= {
        ENCODE STRUCTURE {
            c   USE-SET OPTIONAL-ENCODING is-c-present{< a >},
            d   USE-SET OPTIONAL-ENCODING is-d-present{< a >}}
        WITH PER-BASIC-UNALIGNED}

is-c-present {< REFERENCE : a >} #OPTIONAL ::= {
        PRESENCE
            DETERMINED BY field-to-be-used
            USING a
                DECODER-TRANSFORMS {{INT-TO-BOOL TRUE-IS {0}}}}
```

```
is-d-present {< REFERENCE : a >} #OPTIONAL ::= {
        PRESENCE
            DETERMINED BY field-to-be-used
            USING a
                DECODER-TRANSFORMS {{INT-TO-BOOL TRUE-IS {0 | 2 | 3 | 4}}}}
```

D.2.6.4 L'on a ici une spécification simple, formelle et claire des conditions de présence en "`c`" et "`d`" qui peuvent être interprétées par des utilitaires de codeur-décodeur. Les commentaires ASN.1 ne peuvent pas être traités par utilitaires. La fourniture d'un codage d'offres d'options pour "`c`" et "`d`" implique que le codage PER pour `OPTIONAL` n'est pas utilisé dans ce cas et il n'y a pas de bits auxiliaires.

D.2.6.5 Les objets de codage paramétrés "`is-c-present`" et "`is-d-present`" spécifient la façon dont la présence des composants est déterminée pendant le décodage. Noter qu'aucune transformation n'est nécessaire (ni permise) pour le codage parce que le déterminant possède une sémantique applicative (c'est-à-dire visible dans la définition de type ASN.1). Cependant, un bon utilitaire de codage gérera le réglage de "`a`" par l'application, afin de garantir que sa valeur est compatible avec la présence ou l'absence de "`c`" et "`d`" que le code applicatif a déterminée.

D.2.7 La présence d'un composant facultatif dépend d'une condition externe

D.2.7.1 L'attribution ASN.1 est la suivante:

```
ConditionalPresenceOnExternalCondition ::= SEQUENCE {
        a   BOOLEAN OPTIONAL
            -- Condition: "a" est présent si la condition externe "C" est vérifiée,
            -- sinon "a" absent -- }
        -- Noter que la contrainte de présence ne peut être fournie que dans un commentaire.
```

D.2.7.2 Le code applicatif, pour aussi bien un émetteur qu'un récepteur, peut évaluer la condition "`c`" à partir de certaines informations extérieures au message. Le spécificateur ECN recherche des utilitaires permettant d'invoquer un tel code afin de déterminer la présence de "`a`", plutôt qu'au moyen d'un bit dans le codage.

D.2.7.3 L'attribution d'objet de codage est la suivante:

```
conditionalPresenceOnExternalConditionEncoding #ConditionalPresenceOnExternalCondition ::= {
        ENCODE STRUCTURE {
            a   USE-SET OPTIONAL-ENCODING is-a-present}
        WITH PER-BASIC-UNALIGNED}

is-a-present #OPTIONAL ::=
        NON-ECN-BEGIN {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) user-notation(7)}
        extern C;
        extern channel;
        /* a n'est présent que si "channel" est égal à une valeur “C” */
        int is_a_present() {
            if(channel == C) return 1;
            else return 0; }
        NON-ECN-END
```

D.2.7.4 Etant donné que la condition est extérieure au message, l'objet de codage permettant de déterminer la présence du composant "`a`" ne peut être spécifié que par une définition non ECN d'un objet de codage. Cependant, bien que cela économise des bits en ligne, de nombreux concepteurs jugeront préférable d'inclure le bit dans le message afin de réduire la possibilité d'erreur et de faciliter les essais et les contrôles. Ces choix appartiennent au spécificateur ECN.

D.2.8 Une liste de longueur variable

D.2.8.1 L'attribution ASN.1 est la suivante:

```
EnclosingStructureForList ::= SEQUENCE {
        list VariableLengthList}

VariableLengthList ::= SEQUENCE (SIZE (0..1023) ) OF INTEGER (1..2)
-- Normalement, la liste ne contient que peu d'éléments (0..31), mais elle pourrait en contenir un grand nombre.
```

D.2.8.2 Les règles PER de base non alignées codent la longueur de la liste au moyen de 10 bits même si normalement la longueur est dans l'étendue `0..31`. L'on souhaite minimiser la taille du codage du déterminant de longueur dans le cas normal tout en continuant à autoriser des valeurs qui apparaissent rarement.

D.2.8.3 L'attribution d'objet de codage est la suivante:

```
enclosingStructureForListEncoding #EnclosingStructureForList ::= {
        USE   #EnclosingStructureForListStruct
        MAPPING FIELDS
```

```
            WITH {
               ENCODE STRUCTURE {
                  aux-length list-lengthEncoding,
                  list  {
                     ENCODE STRUCTURE {
                        STRUCTURED WITH {
                                                REPETITION-ENCODING {
                                                      REPETITION-SPACE
                                                            DETERMINED BY field-to-be-set
                                                      USING aux-length}}}
                                          WITH PER-BASIC-UNALIGNED }}
                  WITH PER-BASIC-UNALIGNED}}
                     -- premier mappage: utilisation d'une structure de codage avec un déterminant
                     -- explicite de longueur.

list-lengthEncoding #AuxVariableListLength ::= {
            USE   #AuxVariableListLengthStruct      -- Voir § D.2.8.4.
            MAPPING ORDERED VALUES
            WITH PER-BASIC-UNALIGNED}
            -- Second mappage: la longueur de liste est codée en tant que choix entre une forme courte "normally" et
            -- une forme longue "sometimes".
```

D.2.8.4 Les attributions de structure de codage sont les suivantes:

```
#EnclosingStructureForListStruct ::= #CONCATENATION {
            aux-length#AuxVariableListLength,
            list      #VariableLengthList}

#AuxVariableListLength ::= #INT (0..1023)

#AuxVariableListLengthStruct ::= #ALTERNATIVES {
            normally       #INT (0..31),
            sometimes #INT (32..1023)}
```

D.2.8.5 Le déterminant de longueur pour le composant "`list`" est variable. Le déterminant de longueur pour les valeurs de liste courte est codé au moyen de 1 bit pour le déterminant de choix et 5 bits pour le déterminant de longueur. Le déterminant de longueur pour les valeurs de liste longue est codé au moyen de 1 bit pour le déterminant de choix et de 10 bits pour le déterminant de longueur.

D.2.9 Listes de longueur égale

D.2.9.1 L'attribution ASN.1 est la suivante:

```
EqualLengthLists ::= SEQUENCE {
            list1    List1,
            list2    List2}
            (CONSTRAINED BY {
              -- "list1" et "list2" ont toujours le même nombre d'éléments. --
            })

List1 ::= SEQUENCE (SIZE (0..1023)) OF BOOLEAN

List2 ::= SEQUENCE (SIZE (0..1023)) OF INTEGER (1..2)
```

D.2.9.2 Aussi bien "`list1`" que "`list2`" ont le même nombre d'éléments et le spécificateur ECN souhaite utiliser un seul déterminant de longueur pour les deux listes. (Les règles PER coderont la longueur des champs pour les deux composants.)

D.2.9.3 Les attributions des objets de codage sont les suivantes:

```
equalLengthListsEncoding #EqualLengthLists ::= {
            USE    #EqualLengthListsStruct
            MAPPING FIELDS
            WITH {
               ENCODE STRUCTURE {
                  list1     list1Encoding{< aux-length >},
                  list2     list2Encoding{< aux-length >}}
                    WITH PER-BASIC-UNALIGNED}}
```

Le premier objet de codage est défini avec deux objets de codage paramétrés de classes `#List1` et `#List2` respectivement au moyen du champ de longueur en tant que paramètre réel. Ces deux objets de codage utilisent un objet de codage paramétré commun, de classe `#REPETITION`.

```
list1Encoding {< REFERENCE : length >} #List1 ::= {
        ENCODE STRUCTURE { USE-SET
                STRUCTURED WITH list-with-determinantEncoding {< length >}}
        WITH PER-BASIC-UNALIGNED}

list2Encoding {< REFERENCE : length >} #List2 ::= {
        ENCODE STRUCTURE { USE-SET
                STRUCTURED WITH list-with-determinantEncoding {< length >}}
        WITH PER-BASIC-UNALIGNED}

list-with-determinantEncoding {< REFERENCE : length-determinant >} #REPETITION ::= {
        REPETITION-ENCODING {
            REPETITION-SPACE
                SIZE variable-with-determinant
                    MULTIPLE OF repetitions
                DETERMINED BY field-to-be-set
                USING length-determinant}}
```

D.2.9.4 Les attributions de structure de codages sont les suivantes:

```
#EqualLengthListsStruct ::= #CONCATENATION {
        aux-length#AuxListLength,
        list1      #List1,
        list2      #List2}

#AuxListLength ::= #INT (0..1023)
```

D.2.10 Probabilités d'option lors d'un choix irrégulier

D.2.10.1 L'attribution ASN.1 est la suivante:

```
EnclosingStructureForChoice ::= SEQUENCE {
        choice     UnevenChoiceProbability }

UnevenChoiceProbability ::= CHOICE {
        frequent1  INTEGER (1..2),
        frequent2  BOOLEAN,
        common1    INTEGER (1..2),
        common2    BOOLEAN,
        common3    BOOLEAN,
        rare1      BOOLEAN,
        rare2      INTEGER (1..2),
        rare3      INTEGER (1..2)}
```

D.2.10.2 Les options du type choix ont différentes probabilités de sélection. Il y a des options qui apparaissent très fréquemment ("`frequent1`" et "`frequent2`"), ou qui sont assez courant ("`common1`", "`common2`" et "`common3`"), ou qui n'apparaissent que rarement ("`rare1`", "`rare2`" et "`rare3`"). Le codage pour le déterminant d'option devrait être tel que les options qui apparaissent fréquemment aient des champs de déterminant plus courts que celles qui apparaissent rarement.

D.2.10.3 Les attributions de structure de codage sont les suivantes:

```
#EnclosingStructureForChoiceStruct ::= #CONCATENATION {
        aux-selector   #AuxSelector,
        choice     #UnevenChoiceProbability }
        -- Déterminant d'option explicite auxiliaire pour "choice".

#AuxSelector ::= #INT (0..7)
```

D.2.10.4 Les attributions des objets de codage sont les suivantes:

```
enclosingStructureForChoiceEncoding #EnclosingStructureForChoice ::= {
        USE   #EnclosingStructureForChoiceStruct
        MAPPING FIELDS
        WITH {
            ENCODE STRUCTURE {
                aux-selector   auxSelectorEncoding,
                choice {
                    ENCODE STRUCTURE  {
                        STRUCTURED WITH  {
                            ALTERNATIVE
                                DETERMINED BY field-to-be-set
                                USING aux-selector}}
```

```
                    WITH PER-BASIC-UNALIGNED }}
                  WITH PER-BASIC-UNALIGNED} }
                  -- Premier mappage: insère un déterminant d'option auxiliaire explicite.
                  -- Cet objet de codage spécifie qu'un déterminant auxiliaire est utilisé
                  -- en tant que déterminant d'option.

auxSelectorEncoding #AuxSelector ::= {
        USE    #BITS
        -- codages Huffman en ECN
        -- RANGE (0..7)
        -- (0..1) est 60%
        -- (2..4) est 30%
        -- (5..7) est 10%
        -- Fin de définition
        -- Mappages produits par "ECN Public Domain Software for Huffman encodings, version 1"
        -- (voir § E.8)
        MAPPING TO BITS {
                0 ..  1 TO '10'B   .. '11'B,
                2 ..  4 TO '001'B .. '011'B,
                5  TO '0001'B,
                6 ..  7 TO '00000'B .. '00001'B}
        WITH PER-BASIC-UNALIGNED }
        -- Second mappage: mappe des indices de déterminant sur des chaînes de bits
```

D.2.10.5 Dans ce qui précède, l'on a quantifié "frequent", "common" et "rare" à 60%, 30% et 10%, respectivement, et utilisé le générateur de codages Huffman en notation ECN du domaine public (voir § E.8) afin de déterminer les séquences binaires optimales à utiliser pour chaque étendue d'entiers.

D.2.10.6 Ce qui précède est optimal dans le sens mathématique, mais le pourcentage du trafic total exprimant la différence ainsi créée dépend de ce dont les autres parties du protocole se composent. Bien que l'effort de produire et d'utiliser des codages optimaux ne coûte rien (car des utilitaires peuvent être utilisés), les gains ultimes peuvent ne pas être significatifs.

D.2.11 Un message de version 1

D.2.11.1 Attribution ASN.1

```
Version1Message ::= SEQUENCE {
        ie-1        BOOLEAN,
        ie-2        INTEGER (0..20)}
```

L'on souhaite utiliser les règles PER de base non alignées, mais l'on a l'intention d'ajouter encore des champs en version 2 et de spécifier que les systèmes de version 1 doivent accepter et ignorer toutes données additionnelles dans l'unité PDU.

D.2.11.2 L'on utilise deux structures de codage pour coder le message: l'une est la structure de codage produite implicitement qui ne contient que les champs de la version 1 et le second est une structure que l'on définit comme contenant les champs de version 1 plus un champ de bourrage de longueur variable qui s'étend jusqu'à l'extrémité de l'unité PDU. Le système de version 1 utilise la première structure afin de coder et la seconde afin de décoder. En dehors de cette approche de l'extensibilité, tous les codages sont conformes aux règles PER de base non alignées. La structure de décodage de la version 1 est la suivante:

```
#Version1DecodingStructure ::= #CONCATENATION {
        ie-1        #BOOL,
        ie-2        #INT (0..20),
        future-additions   #PAD}
```

D.2.11.3 Les attributions des objets de codage sont les suivantes:

```
version1MessageEncoding  #Version1Message ::= {
        ENCODE-DECODE
            {ENCODE WITH PER-BASIC-UNALIGNED }
        DECODE AS IF decodingSpecification}
decodingSpecification #Version1Message ::= {
        USE #Version1DecodingStructure
        MAPPING FIELDS
                WITH {
              ENCODE STRUCTURE {
              future-additions    additionsEncoding{< OUTER >} }
              WITH PER-BASIC-UNALIGNED}}
```

```
additionsEncoding {< REFERENCE:determinant >} #PAD ::= {
        ENCODING-SPACE
            SIZE   encoder-option-with-determinant
            DETERMINED BY container
            USING determinant}
```

D.2.12 L'ensemble d'objets de codage

Cet ensemble d'objets de codage contient des définitions de codage pour certains des types spécifiés dans le module ASN.1 nommé "**`Example2-ASN1-Module`**" (le reste est codé au moyen des règles PER de base non alignées).

```
Example2Encodings #ENCODINGS ::= {
        normallySmallValuesEncoding-1                       |
        sparseEvenlyDistributedValueSetEncoding             |
        sparseUnevenlyDistributedValueSetEncoding           |
        conditionalPresenceOnValueEncoding                  |
        conditionalPresenceOnExternalConditionEncoding      |
        enclosingStructureForListEncoding                   |
        equalLenghListsEncoding                             |
        enclosingStructureForChoiceEncoding                 |
        version1MessageEncoding }
```

D.2.13 Définitions ASN.1

Ce module regroupe toutes les définitions ASN.1 du § D.2.1 au § D.2.11 qui seront codées conformément aux objets de codage définis dans le module EDM. Il énumère également les autres définitions ASN.1 qui seront codées avec les règles de codage PER de base non alignées.

```
Example2-ASN1-Module {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module2(5)}
        DEFINITIONS AUTOMATIC TAGS : :=
        BEGIN

        ExampleMessages ::= CHOICE {
                firstExample        NormallySmallValues,
                secondExample       SparseEvenlyDistributedValueSet
                -- etc.
    }

        NormallySmallValues ::= INTEGER (0..1024)

        SparseEvenlyDistributedValueSet ::= INTEGER (2 | 4 | 6 | 8 | 10 | 12 | 14 | 16)


-- etc.

        END
```

D.2.14 Définitions EDM

```
Example2-EDM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module2(6)}
        ENCODING-DEFINITIONS ::=
        BEGIN

        EXPORTS Example2Encodings;

        IMPORTS #NormallySmallValues, #SparseEvenlyDistributedValue,
                #SparseUnevenlyDistributedValueSet, #ConditionalPresenceOnValueSet,
                #ConditionalPresenceOnExternalCondition,
                #EnclosingStructureForList, #EqualLengthLists, #EnclosingStructureForChoice,
                #Version1Message
                FROM Example2-ASN1-Module
                {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module2(5)};

        Example2Encodings #ENCODINGS ::= {
                    normallySmallValuesEncoding   |
                        -- etc.
                    extensibleMessageEncoding}
        -- etc.

        END
```

D.2.15 Définitions ELM

Le module ELM suivant est associé au module ASN.1 défini au § D.2.13 et au module EDM défini au § D.2.14.

```
Example2-ELM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) elm-module2(4)}
        LINK-DEFINITIONS ::=
        BEGIN

        IMPORTS
                Example2Encodings FROM Example2-EDM
                        {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module2(6)}
                #ExampleMessages FROM Example2-ASN1-Module
                        {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module2(5)};

        ENCODE #ExampleMessages WITH Example2Encodings
                COMPLETED BY PER-BASIC-UNALIGNED

        END
```

D.3 Exemples de structure produite explicitement

Les exemples décrits du § D.3.1 au D.3.4 montrent l'utilisation de structures produites explicitement afin de remplacer une classe de codage contenue dans une structure de codage produite implicitement par une classe synonyme. L'on produit ensuite des codages spécialisés en insérant dans l'ensemble d'objets de codage un objet de la classe synonyme.

Les exemples sont présentés au moyen du format suivant:

- la production "attribution de type ASN.1" qui indique la définition originale de type ASN.1;
- l'exigence qui énumère les changements requis à partir des codages fournis par les règles PER de base non alignées;
- modification de la structure de codage produite implicitement afin de créer une nouvelle structure de codage;
- les attributions de classe de codage et d'objets de codage.

D.3.1 Séquence avec composants facultatifs définis par un pointeur

D.3.1.1 L'attribution ASN.1 est la suivante:

```
Sequence1 ::= SEQUENCE{
  component1   INTEGER      OPTIONAL,
  component2   INTEGER      OPTIONAL,
  component3   VisibleString }
```

D.3.1.2 Au lieu d'utiliser le mappage de bits PER pour les deux composants de type entier marqués **`OPTIONAL`**, la présence et la position de ces composants sont déterminées par des pointeurs au début du codage de la séquence. Chaque pointeur contient 0 (composant absent) ou un décalage relatif au codage du composant qui commence sur une limite d'octet.

D.3.1.3 La classe de codage **`#Integer`** est remplacée par la classe "**`#Integer-with-pointer-concat`**" dans l'objet de codage de "**`sequence1-encoding`**". La classe "**`#Integer-with-pointer-concat`**" est définie comme une structure de concaténation contenant un élément qui est l'élément remplacé combiné avec une classe de la catégorie des offres d'options "**`#Integer-optionality`**".

D.3.1.4 Puis deux objets de codage sont définis. Le premier, "**`integer-with-pointer-concat-encoding`**" de classe "**`#Integer-with-pointer-concat`**" reçoit trois paramètres: l'élément remplacé, le pointeur et l'ensemble courant d'objets de codage combinés (voir § 22.1.3.7). Le second objet, "**`integer-optionality-encoding`**" de classe "**`#Integer-optionality`**" reçoit un paramètre, le pointeur, qui est utilisé afin de déterminer la présence du composant. Comme la production **`PER-BASIC-UNALIGNED`** ne contient pas d'objet de codage de classe **`#CONCATENATION`** avec composants facultatifs, un troisième objet de codage de classe **`#CONCATENATION`** doit être défini. Cet objet "concat" utilise les réglages par défaut.

D.3.1.5 Les attributions de classe de codage et d'objets de codage sont les suivantes:

```
sequence1-encoding #SEQUENCE ::= {
        REPLACE OPTIONALS
                WITH #Integer-with-pointer-concat
                  ENCODED BY integer-with-pointer-concat-encoding
                                INSERT AT HEAD #Pointer
```

```
            ENCODING-SPACE
                SIZE variable-with-determinant
                DETERMINED BY container
                USING OUTER }

#Pointer ::= #INTEGER

#Integer-with-pointer-concat {< #Element >} ::= #CONCATENATION {
        element    #Element OPTIONAL-ENCODING #Integer-optionality }

#Integer-optionality ::= #OPTIONAL

integer-optionality-encoding{< REFERENCE: start-pointer>} #Integer-optionality ::= {
        ALIGNED TO ANY octet
        START-POINTER start-pointer
        PRESENCE DETERMINED BY pointer}

integer-with-pointer-concat-encoding {< #Element, REFERENCE:pointer, #ENCODINGS:EncodingObjectSet >}
        #Integer-with-pointer-concat{< #Element >} ::= {
        ENCODE STRUCTURE {
            element USE-SET OPTIONAL-ENCODING integer-optionality-encoding{< pointer >}}
        WITH EncodingObjectSet}

concat #CONCATENATION ::= {
        ENCODING-SPACE }
```

D.3.2 Addition d'un type booléen en tant que déterminant de présence

D.3.2.1 L'attribution ASN.1 est la suivante:

```
Sequence2 ::= SEQUENCE{
  component1    BOOLEAN     OPTIONAL,
  component2    INTEGER,
  component3    VisibleString   OPTIONAL }
```

D.3.2.2 Au lieu d'utiliser le mappage de bits PER pour les composants marqués "`OPTIONAL`", la présence d'un composant facultatif est associée à la valeur d'un fanion de présence égal à 1 (composant absent), ou 0 (composant présent). Dans ce cas, le fanion de présence est inversé.

D.3.2.3 Les structures de codage et les objets de codage sont définis comme suit:

La classe de codage `#OPTIONAL` est renommée `#Sequence2-optional` dans la clause "`RENAMES`" (voir § D.3.7). La classe "`#Sequence2`" est donc implicitement remplacée par:

```
#Sequence2 ::= #SEQUENCE   {
  component1    #BOOL          OPTIONAL-ENCODING #Sequence2-optional,
  component2    #INTEGER,
component3    #VisibleString OPTIONAL-ENCODING #Sequence2-optional}
```

où:

```
#Sequence2-optional ::= #OPTIONAL
```

Puis un objet de codage de classe "`#Sequence2-optional`" est défini; cet objet, au moyen du groupe de remplacement, remplace la définition de codage du composant (voir § 23.10.3.2) par la classe "`Optional-with-determinant`".

```
sequence2-optional-encoding #Sequence2-optional ::= {
        REPLACE STRUCTURE
        WITH #Optional-with-determinant
                ENCODED BY optional-with-determinant-encoding}
```

Cette classe, qui est paramétrée par le composant original, appartient à la catégorie des concaténations et a deux composants: le déterminant (booléen) et le composant original.

```
#Optional-with-determinant{< #Element >} ::= #CONCATENATION {
  determinant   #BOOLEAN,
  component     #Element OPTIONAL-ENCODING #Presence-determinant}
```

où:

```
#Presence-determinant ::= #OPTIONAL
```

Puis un objet de codage de classe "`#Optional-with-determinant`" est défini; cet objet a deux paramètres fictifs: la classe du composant et un ensemble d'objets de codage utilisé pour coder tout sauf le déterminant et l'offres d'options du composant:

```
optional-with-determinant-encoding {< #Element, #ENCODINGS: Sequence2-combined-encoding-object-set >}
        #Optional-with-determinant {< #Element >}      ::= {
  ENCODE STRUCTURE {
      determinant    determinant-encoding,
  component USE-SET
                                OPTIONAL-ENCODING if-component-present-encoding{< determinant >} }
        WITH Sequence2-combined-encoding-object-set }
```

Le codage est complètement spécifié par la définition d'objets de codage "`if-component-present-encoding`" et "`determinant-encoding`":

```
if-component-present-encoding {<REFERENCE:presence-bit>} #Presence-determinant ::= {
        PRESENCE
                DETERMINED BY field-to-be-set
                USING presence-bit}

determinant-encoding #BOOLEAN ::= {
        ENCODING-SPACE
            SIZE 1
            MULTIPLE OF bit
        TRUE-PATTERN bits:'0'B
        FALSE-PATTERN bits:'1'B}
```

D.3.3 Séquence avec composants facultatifs identifiés par une étiquette unique et délimités par un champ de longueur

D.3.3.1 Les attributions ASN.1 sont les suivantes:

```
Octet3 ::= OCTET STRING (CONTAINING Sequence3)

Sequence3 ::=SEQUENCE {
  component1    [0] BIT STRING (SIZE(0..2047)) OPTIONAL,
  component2    [1] OCTET STRING (SIZE(0..2047))     OPTIONAL,
  component3    [2] VisibleString (SIZE(0..2047)) OPTIONAL }
```

D.3.3.2 Chaque composant est identifié par une étiquette sur quatre bits et la longueur totale de la séquence est spécifiée avec un champ de onze bits qui précède le codage du premier composant.

D.3.3.3 Les classes de codage `#OCTETS`, `#OPTIONAL` et `#TAG` sont renommées respectivement `#Octets3`, `#Sequence3-optional` et `#TAG-4-bits` dans la clause "`RENAMES`" (voir § D.3.7). Puis les objets de codage des nouvelles classes de codage sont définis.

D.3.3.4 Les attributions de classe de codage et d'objets de codage pour la chaîne d'octets sont les suivantes:

```
#Octets3 ::= #OCTET-STRING

octets3-encoding #Octets3 ::= {
        REPETITION-ENCODING {
                REPLACE STRUCTURE
                        WITH #Octets-with-length
              ENCODED BY octets-with-length-encoding}}

#Octets-with-length{< #Element >} ::= #CONCATENATION {
        length  #INT(0..2047),
        octets  #Element}

octets-with-length-encoding{< #Element >} #Octets-with-length{< #Element >} ::= {
        ENCODE STRUCTURE {
            octets octets-encoding{< length >}}
        WITH PER-BASIC-UNALIGNED}

octets-encoding{< REFERENCE:length >} #OCTETS ::= {
        REPETITION-ENCODING {
                REPETITION-SPACE
                  SIZE variable-with-determinant
              DETERMINED BY field-to-be-set
              USING length} }
```

D.3.3.5 Les attributions de classe de codage et d'objets de codage pour la séquence sont les suivantes:

```
#Sequence3-optional ::= #OPTIONAL

sequence3-optional-encoding  #Sequence3-optional ::= {
  PRESENCE
            DETERMINED BY container
                USING OUTER}

#TAG-4-bits ::= #TAG

tag-4-bits-encoding  #TAG-4-bits   ::= {
  ENCODING-SPACE
      SIZE 4}
```

D.3.4 Type "séquence-de" avec décompte

D.3.4.1 L'attribution ASN.1 est la suivante:

```
SequenceOfIntegers ::= SEQUENCE(SIZE(0..63)) OF INTEGER(0..1023)
```

D.3.4.2 Le nombre d'éléments est codé dans un champ de 6 bits précédant le codage du premier élément.

D.3.4.3 La classe de codage **`#SEQUENCE-OF`** est renommée **`#Sequenceof`** dans la clause "**`RENAMES`**" (voir § D.3.7). Un objet de codage de la nouvelle classe de codage est défini. Les attributions de classe de codage et d'objets de codage sont les suivantes:

```
#SequenceOf ::= #REPETITION

sequenceOf-encoding #SequenceOf ::= {
  REPETITION-ENCODING {
                REPLACE STRUCTURE
                WITH #SequenceOf-with-count
                ENCODED BY sequenceOf-with-count-encoding}}


#SequenceOf-with-count{< #Element >} ::= #CONCATENATION {
        count   #INT(0..63),
        elements   #Element }


sequenceOf-with-count-encoding{< #Element >} #SequenceOf-with-count{< #Element >} ::= {
        ENCODE STRUCTURE {
            elements {
                        ENCODE STRUCTURE {
                                STRUCTURED WITH elements-encoding{< count >}}
                        WITH PER-BASIC-UNALIGNED}}
        WITH PER-BASIC-UNALIGNED}


elements-encoding{< REFERENCE:count >} #REPETITION ::= {
        REPETITION-ENCODING {
            REPETITION-SPACE
                SIZE variable-with-determinant
        MULTIPLE OF repetitions
                DETERMINED BY field-to-be-set
                USING count}}
```

D.3.4.4 Le champ de décompte est codé au moyen des règles de codage PER pour un type d'entier avec la contrainte d'étendue de valeurs (0..63), qui indique un champ de 6 bits.

D.3.5 Ensemble d'objets de codage

L'ensemble d'objets de codage contient des objets de codage de classes définies dans le module EDM.

```
Example3Encodings     #ENCODINGS ::= {
                sequence1-encoding                  |
                concat                              |
                sequence2-optional-encoding         |
                octets3-encoding                    |
                sequence3-optional-encoding         |
                tag-4-bits-encoding                 |
                sequenceOf-encoding }
```

D.3.6 Définitions ASN.1

Ce module regroupe les définitions ASN.1 du § D.3.1 au § D.3.4 qui seront codées conformément aux objets de codage définis dans le module EDM du § D.3.7.

```
Example3-ASN1-Module {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module3(9)}
        DEFINITIONS
        AUTOMATIC TAGS ::=
        BEGIN

        Sequence1 ::=       SEQUENCE    {
              component1    BOOLEAN             OPTIONAL,
              component2    INTEGER             OPTIONAL,
              component3    VisibleString       OPTIONAL }

 -- etc.

        END
```

D.3.7 Définitions EDM

```
Example3-EDM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module3(10)}
ENCODING-DEFINITIONS ::=
BEGIN

EXPORTS Example3Encodings;
RENAMES
  #OPTIONAL AS #Sequence2-optional
      IN #Sequence2
  #OCTET-STRING AS #Octets3
      IN ALL
  #OPTIONAL AS #Sequence3-optional
      IN #Sequence3
  #TAG AS #TAG-4-bits
      IN #Sequence3
FROM Example3-ASN1-Module { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module3(9)};

Example3Encodings #ENCODINGS   ::= {
  sequence1-optional-encoding |
            -- etc.
  sequenceOf-encoding }
 -- etc.
END
```

D.3.8 Définitions ELM

Le module ELM suivant est associé au module ASN.1 défini au § D.3.6 et au module EDM défini au § D.3.7.

```
Example3-ELM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) elm-module3(8)}
LINK-DEFINITIONS ::=
BEGIN

IMPORTS Example3Encodings, #Sequence1, #Sequence2, #Octet3, #Sequence3, #SequenceOfIntegers
FROM Example3-EDM { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module3(10) };

ENCODE #Sequence1, #Sequence2, #Octet3, #Sequence3, #SequenceOfIntegers
WITH Example3Encodings
COMPLETED BY PER-BASIC-UNALIGNED

END
```

D.4 Exemple de codage par bit d'extension

D.4.1 Description du problème

D.4.1.1 Cet exemple est repris de la Rec. UIT-T Q.763 (*Système de signalisation n° 7 – Formats et codes du sous-système utilisateur du RNIS*).

D.4.1.2 Il est prescrit de produire le codage suivant sous la forme d'une série d'octets:

8	7 6	5 4 3 2 1
indicateur d'extension	**en réserve**	**profil du protocole**

D.4.1.3 Le bit 8 est un "indicateur d'extension". Si sa valeur est 0, il y a un octet suivant dans le même format. Si sa valeur est 1, c'est le dernier octet de la série.

NOTE – Le codage PER du booléen est 1 pour **TRUE** et 0 pour **FALSE** et la notation ECN nécessite que le dernier élément renvoie **FALSE** et que les éléments antérieurs renvoient **TRUE**. Donc, si l'on utilise un booléen à codage PER pour le bit d'extension, il faut appliquer la transformation "**not**".

D.4.1.4 C'est l'utilisation traditionnelle d'un "bit d'extension", bien qu'avec l'éventuel zéro inhabituel pour "more" et un pour "last".

D.4.1.5 L'exemple serait simplifié si l'utilisation du champ "indicateur d'extension" effectuait l'interversion de zéro et un, et s'il n'y avait pas de bits "de réserve", mais l'utilisation de l'exemple réel a été préférée ici.

D.4.1.6 Il y a quatre moyens de résoudre ce problème.

D.4.1.7 Le premier consiste à inclure un composant dans la spécification ASN.1 afin d'offrir le déterminant de bit d'extension (voir § D.4.2). Ce moyen est déconseillé pour deux raisons. La première est que la définition de type ASN.1 contient un composant qui ne donne pas la sémantique applicative. La seconde est qu'il nécessite que l'application fixe (en redondance) ce champ correctement dans chaque élément de la répétition du bit d'extension.

D.4.1.8 Le second moyen consiste à utiliser des mappages de valeur d'une structure produite implicitement sur une structure de codage définie par l'usager qui contient le déterminant de bit d'extension (voir § D.4.3).

D.4.1.9 Le troisième moyen consiste à utiliser le mécanisme de remplacement afin d'inclure le déterminant de bit d'extension (voir § D.4.4).

D.4.1.10 Le quatrième moyen consiste à utiliser l'insertion en tête de séquence du déterminant de bit d'extension. (Cela n'est pas illustré ici.)

D.4.1.11 Ces trois dernières approches ont chacune leurs propres avantages et le choix entre elles est surtout une question de style.

D.4.2 Utilisation de la notation ASN.1 afin d'offrir le déterminant de bit d'extension

D.4.2.1 Dans cette méthode, la notation ASN.1 reflète tous les champs contenus dans le codage. Cette méthode est généralement considérée comme "chargée", car les champs qui ne devraient être visibles que dans le codage sont visibles par l'application, ce qui réduit le "masquage d'informations" qui est la force de la notation ASN.1. Dans ce cas la notation ASN.1 est la suivante:

```
ProfileIndication ::= SEQUENCE OF
        SEQUENCE {
                more-bit            BOOLEAN,
                reserved            BIT STRING (SIZE (2)),
                protocol-Profile-ID    INTEGER (0..32) }
```

D.4.2.2 La structure de codage produite implicitement est la suivante:

```
#ProfileIndication ::= #SEQUENCE-OF {
        #SEQUENCE {
                more-bit            #BOOLEAN,
                reserved            #BIT-STRING (SIZE (2)),
                protocol-Profile-ID    #INTEGER (0..32) } }
```

D.4.2.3 Tout d'abord, l'on produit un objet de codage générique pour la classe **#SEQUENCE-OF** qui utilise un bit d'extension dans un champ identifié comme un paramètre de l'objet de codage, et avec la valeur **BOOLEAN TRUE** (codée en tant que fanion "1" par les règles PER) pour le dernier élément:

```
more-bit-encoding {< REFERENCE:more-bit >} #SEQUENCE-OF ::= {
        REPETITION-ENCODING {
                REPETITION-SPACE
                        SIZE variable-with-determinant
                        DETERMINED BY flag-to-be-set
                                USING more-bit
                                        ENCODER-TRANSFORMS
                                                { { BOOL-TO-BOOL AS logical:not } } } }
```

D.4.2.4 Cet objet de codage est également utilisé aux § D.4.3 et D.4.4, car il offre la description fondamentale du codage nécessaire pour la répétition.

D.4.2.5 Avec la première approche (simple mais chargée!), l'on peut maintenant définir notre objet de codage pour la classe **`#ProfileIndication`** au moyen de la production **`ENCODE STRUCTURE`**, et appliquer cet objet de codage dans le module ELM, ce qui achève l'exemple. L'objet de codage est défini comme suit:

```
profileIndicationEncoding #ProfileIndication ::= {
        ENCODE STRUCTURE {
                STRUCTURED WITH
                        more-bit-encoding {< more-bit >} }
        WITH PER-BASIC-UNALIGNED }
```

D.4.3 Utilisation de mappages de valeur afin d'offrir le déterminant de bit d'extension

D.4.3.1 Dans cette approche, l'on masque la structure de codage contenue dans une définition ECN d'une structure de codage définie par l'usager et l'on utilise le mappage de valeurs par champs appariés afin de permettre un codage de la structure définie par l'utilisateur et de coder une définition de type ASN.1 simplifiée.

D.4.3.2 La définition de type ASN.1 est donc:

```
ProfileIndication2 ::= SEQUENCE OF
        protocol-Profile-ID  INTEGER (0..32)
```

D.4.3.3 Ce type a une structure de codage produite implicitement (à laquelle s'appliquent nos codages contenus dans le module ELM) de:

```
#ProfileIndication2 ::=  #SEQUENCE-OF {
        protocol-Profile-ID  #INTEGER (0..32) }
```

D.4.3.4 Nous définissons une structure pour le codage dont nous avons besoin, semblable à la notation ASN.1 que nous avons rédigée dans la première approche (voir § D.4.2.1), sauf que l'on utilise la classe **`#PAD`** pour les bits réservés:

```
#ProfileIndicationStruct ::= #SEQUENCE-OF {
        #SEQUENCE {
                more-bit-field          #BOOLEAN,
                reserved            #PAD,
                protocol-Profile-ID     #INTEGER (0..32) } }
```

D.4.3.5 Nous aurons donc besoin d'un objet de codage pour la classe **`#PAD`** de deux bits, avant de pouvoir achever le codage:

```
pad-encoding #PAD ::= {
        ENCODING-SPACE SIZE 2
        PATTERN bits:'00'B }
```

NOTE – Le § 23.11.4.2 spécifie que les décodeurs doivent normalement accepter toute valeur pour les bits de classe **`#PAD`**, ce qui est ce que nous recherchons ici, de sorte que nous n'avons pas besoin d'un codage/décodage différentiel.

D.4.3.6 Nous définisssons un objet de codage pour notre structure, tout à fait comme dans la première approche (voir § D.4.2.5):

```
profileIndicationStructEncoding #ProfileIndicationStruct ::= {
        ENCODE STRUCTURE {
                STRUCTURED WITH
                        more-bit-encoding {< more-bit-field >} }
        WITH {pad-encoding} COMPLETED BY PER-BASIC-UNALIGNED }
```

D.4.3.7 Finalement, l'on utilise le mappage de valeur à partir de la structure produite implicitement, sur notre structure produite explicitement afin de définir notre codage final:

```
profileIndication2Encoding #ProfileIndication2 ::= {
        USE #ProfileIndicationStruct
                MAPPING FIELDS
        WITH profileIndicationStructEncoding }
```

D.4.4 Utilisation du mécanisme de remplacement afin d'offrir le déterminant de bit d'extension

D.4.4.1 Dans notre approche finale, nous définissons un codage générique de type "séquence-de", qui peut s'appliquer à toute séquence d'items. A cette fin, nous avons besoin d'une structure de codage paramétrée:

```
#SequenceOfStruct {< #Component >} ::=
        #SEQUENCE {
                more-bit-field          #BOOLEAN,
```

```
                    reserved        #PAD,
                    sequence-of-component#Component }
```

D.4.4.2 Nous définissons notre codage de type "séquence-de" afin d'effectuer un remplacement du composant avec cette structure, en spécifiant un codage de bit d'extension et au moyen du codage de bourrage défini:

```
sequence-of-encoding #SEQUENCE-OF ::= {
    REPETITION-ENCODING {
        REPLACE COMPONENT WITH #SequenceOfStruct
            REPETITION-SPACE
                SIZE variable-with-determinant
                DETERMINED BY flag-to-be-set
                USING more-bit-field
                    ENCODER-TRANSFORMS
                        { { BOOL-TO-BOOL AS logical:not } } } }
```

D.4.4.3 Lorsque ce codage est appliqué dans le module ELM, la production "**`COMPLETED BY PER-BASIC-UNALIGNED`**" est utilisée en tant qu'ensemble d'objets de codage combinés afin de réaliser le codage, ce qui donne l'effet recherché.

D.5 Protocole existant spécifié en notation tabulaire

D.5.1 Introduction

D.5.1.1 L'objet de l'exemple contenu dans ce paragraphe est de montrer la façon de construire des définitions ECN pour un protocole dont les codages de message ont été spécifiés au moyen de représentations graphiques de "bits et octets" en notation tabulaire. Les tableaux suivants décrivent le contenu de ces messages (seul le "**`Message1`**" a été montré complètement):

Message 1:

	8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Octet 1	Identificateur de message							
Octet 2	A			b-flag	c-len			réservé
Octet 3	b1		b2	réservé	b3		réservé	
…								
Octet Y	c1				c2			
Octet Y+1	c3						réservé	
…								
Octet Z	d1	d2			d3			réservé

Message 2:

	8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Octet 1	Identificateur de message							
Octet 2…	Quelque chose – 1							

Message 3:

	8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Octet 1	Message id							
Octet 2…	Quelque chose – 2							

D.5.1.2 Tous les messages ont une partie d'en-tête commune (représentée en gris dans les tableaux). Dans cet exemple cette partie n'est utilisée que pour l'identification du message.

D.5.1.3 Le message 1 a trois sortes de champs:

- des champs obligatoires ("a");
- des champs obligatoires qui sont déterminants pour d'autres champs ("b-flag", "c-len");
- des champs facultatifs ("b", "c" et "d").

D.5.1.4 Les champs "b", "c" et "d" sont tous tenus de commencer sur une limite d'octet.

D.5.1.5 Les champs "b", "c" et "d" sont composés de sous-champs ("b1", "b2", "b3", "c1", etc.) de longueur fixe. Par ailleurs les champs "c" et "d" peuvent apparaître de nombreuses fois (mais une seule instance est représentée ci-dessus). Le champ "b2" est tenu de commencer à une limite de quartet.

D.5.1.6 La présence d'un composant facultatif est indiquée au moyen de différentes méthodes:

- le champ "b" est présent si la valeur du champ "b-flag" est 1;
- le champ "d" est présent s'il reste des octets dans le message.

D.5.1.7 La longueur d'un champ qui peut apparaître plusieurs fois est déterminée au moyen de différentes méthodes:

- le nombre de répétitions du champ "c" est gouverné par le champ de déterminant "c-len";
- le nombre de répétitions du champ "d" est déterminé par l'extrémité du message.

D.5.1.8 Le module ASN.1 suivant contient des définitions pour les structures de message présentées ci-dessus. Les hypothèses suivantes ont été faites:

- il y a un type encapsulateur qui contient les définitions communes à tous les messages;
- les champs de déterminant auxiliaire dans les messages sont visibles au niveau ASN.1. Noter que cette opération est effectuée pour simplifier l'exposé dans cet exemple, mais il devrait être de pratique normale de tenir de tels champs en dehors de la définition ASN.1 à moins qu'ils ne contiennent la sémantique applicative réelle;
- l'extensibilité est exprimée sous la forme de commentaires;
- le bourrage n'est pas visible.

D.5.1.9 Le module ASN.1 est le suivant:

```
LegacyProtocol-ASN1-Module {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module4(11)}

    DEFINITIONS AUTOMATIC TAGS ::=
    BEGIN

    LegacyProtocolMessages ::= SEQUENCE {
            message-id     ENUMERATED {message1, message2, message3},
            messages   CHOICE {
                message1       Message1,
                message2       Message2,
                message3       Message3}}
            -- La production CHOICE est contrainte par la valeur de l'identification de message.

    Message1 ::= SEQUENCE {
            a   A,
            b-flag  BOOLEAN,
            c-len   INTEGER (0..max-c-len),
            b   B   OPTIONAL,  -- déterminé par "b-flag"
            c   C,             -- déterminé par "c-len"
            d   D   OPTIONAL}  -- déterminé par la fin de l'unité PDU

    A ::= INTEGER (0..7)       -- les valeurs 5..7 sont réservées pour usage futur. Les systèmes de version 1 devraient traiter
                                  5 à 7 comme 4.

    B ::= SEQUENCE {
            b1  ENUMERATED { e0, e1, e2, e3 },
            b2  BOOLEAN,
            b3  INTEGER (0..3) }

    C ::= SEQUENCE (SIZE (0..max-c-len)) OF C-elem

    C-elem ::= SEQUENCE {
            c1  BIT STRING (SIZE (4)),
            c2  INTEGER (0..1024) }

    D ::= SEQUENCE (SIZE (0..max-d-len)) OF D-elem

    D-elem ::= SEQUENCE {
            d1  BOOLEAN,
            d2  ENUMERATED { f0, f1, f2, f3, f4, f5, f6, f7 },
            d3  INTEGER (0..7) }

    max-c-len INTEGER ::= 7

    max-d-len INTEGER ::= 20
```

```
Message2 ::= SEQUENCE {
        -- quelque chose 1 -- }

Message3 ::= SEQUENCE {
        -- quelque chose 2 -- }

END
```

D.5.1.10 Le module EDM au § D.5.7 contient des définitions de codage pour les messages spécifiés dans le module ASN.1 "**`LegacyProtocol-ASN1-Module`**". Les hypothèses suivantes ont été faites:

- le bourrage à l'intérieur des octets est explicitement spécifié en tant que champs de bourrage;
- le bourrage d'alignement n'est pas spécifié en tant que champs de bourrage explicites.

D.5.2 Définition de codage pour la structure de message de niveau supérieur

D.5.2.1 L'objet de codage "**`legacyProtocolMessagesEncoding`**" spécifie la façon dont les parties communes des messages du protocole existant sont codées. L'identificateur de message est spécifié dans la notation ASN.1 sous la forme d'un type énuméré. Les règles PER de base non alignées codent le champ "**`message-id`**" au moyen du nombre minimal de bits (c'est-à-dire, 2) mais ici l'on souhaite qu'il soit codé au moyen de 8 bits. Par ailleurs, l'on doit spécifier que le champ "**`message-id`**" est à utiliser en tant que déterminant pour "**`messages`**".

D.5.2.2 L'objet de codage "**`legacyProtocolMessagesEncoding`**" est le suivant:

```
legacyProtocolMessagesEncoding #LegacyProtocolMessages ::= {
        ENCODE STRUCTURE {
                message-id {
                        ENCODING {
                                ENCODING-SPACE
                                        SIZE 8}},
            messages {
                         ENCODE STRUCTURE {
                                 STRUCTURED WITH {
                            ALTERNATIVE
                                DETERMINED BY field-to-be-used
                                USING message-id}}
                                 WITH PER-BASIC-UNALIGNED}}
        WITH PER-BASIC-UNALIGNED}
```

D.5.3 Définition de codage pour une structure de message

D.5.3.1 L'objet de codage "**`message1Encoding`**" spécifie la façon dont les valeurs de "**`Message1`**" doivent être codées:

- le champ "**`b`**" est présent si le champ "**`b-flag`**" contient la valeur **`TRUE`**;
- le champ "**`c`**" est présent si le champ "**`c-len`**" ne contient pas la valeur 0. "**`c-len`**" gouverne également le nombre d'éléments contenus dans le champ "**`c`**";
- le champ "**`d`**" est présent s'il y a encore des octets dans un codage pour le message.

D.5.3.2 L'objet de codage pour "Message1" est le suivant:

```
message1Encoding #Message1 ::= {
        ENCODE STRUCTURE {
            b   b-encoding
                OPTIONAL-ENCODING {
                    PRESENCE
                        DETERMINED BY field-to-be-used
                        USING b-flag},
            c   octet-aligned-seq-of-with-ext-determinant{< c-len >},
            d   octet-aligned-seq-of-until-end-of-container
                                OPTIONAL-ENCODING USE-SET}
        WITH PER-BASIC-UNALIGNED}
```

D.5.4 Codage pour la séquence de type "B"

D.5.4.1 Un bourrage de un bit est inséré entre les champs "**`b2`**" et "**`b3`**" ("**`aux-reserved`**"). Le codage de "**`B`**" est aligné en octets.

D.5.4.2 Le codage pour "**B**" est le suivant:

```
b-encoding #B ::= {
        ENCODE STRUCTURE {
                -- Composants
                b3 {
                        ALIGNED TO NEXT nibble
                        ENCODING {
                                ENCODING-SPACE
                                        SIZE 2
                                                MULTIPLE OF bit }}
                -- Structure
                STRUCTURED WITH {
                        ALIGNED TO NEXT octet
                        ENCODING-SPACE
                                SIZE self-delimiting-values
                                        MULTIPLE OF bit }}
        -- Le reste
        WITH PER-BASIC-UNALIGNED}
```

D.5.5 Codage pour un type aligné en octets "séquence-de" avec déterminant de longueur

D.5.5.1 Un seul des types "séquence-de" utilisés dans le protocole existant a un déterminant explicite de longueur.

D.5.5.2 Le codage est aligné en octets. Le nombre d'éléments décompté est déterminé par le champ "**len**".

```
octet-aligned-seq-of-with-ext-determinant{< REFERENCE : len >} #REPETITION ::= {
        REPETITION-ENCODING {
            ALIGNED TO NEXT octet
            REPETITION-SPACE
                SIZE variable-with-determinant
                DETERMINED BY field-to-be-used
                USING len}}
```

D.5.6 Codage pour un type aligné en octets "séquence-de" qui continue jusqu'à l'extrémité de l'unité PDU

D.5.6.1 Le codage est aligné en octets. Le nombre d'éléments est déterminé par l'extrémité de l'unité PDU.

D.5.6.2 L'objet de codage est le suivant:

```
octet-aligned-seq-of-with-ext-determinant{< REFERENCE : len >} #REPETITION ::= {
        REPETITION-ENCODING {
            ALIGNED TO NEXT octet
            REPETITION-SPACE
                SIZE variable-with-determinant
                DETERMINED BY field-to-be-used
                USING len}}
```

D.5.7 Définitions de module EDM

Les définitions de module EDM sont les suivantes:

```
LegacyProtocol-EDM-Module {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module4(13)}
        ENCODING-DEFINITIONS ::=
        BEGIN

        EXPORTS LegacyProtocolEncodings;

        IMPORTS #LegacyProtocolMessages
        FROM   LegacyProtocol-ASN1-Module
        { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module4(11) };

        LegacyProtocolEncodings #ENCODINGS ::= {
                legacyProtocolMessagesEncoding    |
                message1Encoding }

        -- etc.

        END
```

D.5.8 Définitions de module ELM

Le module ELM pour le protocole existant est le suivant:

```
LegacyProtocol-ELM-Module { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) elm-module4(12) }
LINK-DEFINITIONS ::=
BEGIN

IMPORTS
        LegacyProtocolEncodings FROM LegacyProtocol-EDM-Module
                { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module4(13) }
        #LegacyProtocolMessages FROM LegacyProtocol-ASN1-Module
                { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module4(11) };

ENCODE #LegacyProtocolMessages WITH LegacyProtocolEncodings
        COMPLETED BY PER-BASIC-UNALIGNED

END
```

Annexe E

Prise en charge des codages de Huffman

(Cette annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

E.1 Les codages de Huffman sont les codages optimaux pour un ensemble fini de valeurs entières, lorsque la fréquence à laquelle chaque valeur sera transmise est connue.

E.2 Les codages sont autodélimitateurs (aucun déterminant de longueur n'est nécessaire) et utilisent un petit nombre de bits pour les valeurs fréquentes et un plus grand nombre de bits pour les valeurs moins fréquentes.

E.3 Il y a de nombreux codages de Huffman possibles. Par exemple, compte tenu d'un quelconque de ces codages, il suffit de remplacer tous les "1" par des "0" et vice versa afin d'obtenir un codage de Huffman différent (mais tout aussi efficace). Des modifications plus subtiles peuvent également être effectuées afin de produire d'autres codages de Huffman qui soient tout aussi efficaces.

E.4 De façon que les codages de Huffman soient efficaces pour les décodeurs, il est souhaitable que, lorsque des valeurs entières successives codent dans le même nombre de bits, ceux-ci définissent des valeurs entières successives lors de leur interprétation en tant que codage d'entier positif.

E.5 Un codage de Huffman en notation ECN a été défini comme possédant cette propriété, et une macro Microsoft Word 97 a été construite afin de produire la syntaxe d'un mappage "MappingIntToBits" (voir § 19.7) qui soit aussi bien optimal que facile à décoder.

E.6 Une version de cette annexe est disponible avec un bouton de macro qui prendra une spécification des valeurs entières à coder et leur fréquence et qui produira en ligne la spécification formelle du mappage conforme à la notation ECN. (La version de cette annexe contenant la macro associée est disponible gratuitement sur le site Web de l'UIT à l'adresse: http://www.itu.int/ITU-T/publications/recs.html sous la Recommandation X.692, et sur le site Web de l'ISO à l'adresse: http://www.iso.ch/iso/en/ittf/PubliclyAvailableStandards/c034390_ISO_8825-3_2003(E)_Annex_E.html).

E.7 Le texte suivant contient trois exemples de la spécification de codage Huffman en notation ECN.

E.8 Dans la version avec macro, un double cliquage sur le bouton ci-dessous:

ECN Huffman

ajoutera au texte le codage de Huffman en notation ECN des spécifications de mappage.

E.9 L'utilisateur de la version avec macro peut souhaiter une modification de la spécification des valeurs à mapper et de la spécification de leurs fréquences afin de voir les codages qui sont produits en différents cas.

NOTE – Dans la version avec macro, les spécifications de codage peuvent, une fois produites, être supprimées, la spécification Huffman en ECN peut être modifiée et le bouton de macro peut de nouveau être cliqué.

E.10 La syntaxe informelle pour une spécification de codage Huffman en notation ECN devrait ressortir clairement des exemples suivants. Toutes les lignes commencent par un marqueur ASN.1 de commentaires ("--").

E.11 La première ligne (si la macro doit être utilisée) doit contenir exactement les mots "ECN Huffman" précédés par deux tirets et un espace, mais les lignes suivantes ne sont pas sensibles à la casse et peuvent contenir plus ou moins d'espaces.

E.12 La deuxième ligne est requise et les valeurs les plus basses et les plus élevées qui doivent être mappées. L'étendue (limite supérieure moins limite inférieure) est limitée à 1000, mais peut comprendre des valeurs négatives. Toutes les valeurs comprises dans l'étendue n'ont pas besoin d'être mappées.

E.13 Les pourcentages sont donnés pour soit des valeurs isolées soit pour des étendues de valeurs. Il n'est pas nécessaire que la somme des pourcentages soit de 100%, mais un avertissement est donné si ce n'est pas le cas.

E.14 La ligne "REST" est facultative. Elle offre des fréquences pour toutes les valeurs comprises dans l'étendue qui ne sont pas explicitement énumérées. Si cette ligne est absente, dans ce cas les valeurs mappées seront seulement celles qui sont explicitement spécifiées.

E.15 La ligne finale est obligatoire, et doit contenir "End Définition" (en majuscules ou en minuscules). La spécification formelle de codage est insérée (par la macro) après cette ligne.

E.15.1 Le premier exemple est le suivant:

```
my-int-encoding1 #My-Special-1 ::=
{ USE  #BITS
      -- ECN Huffman
      -- RANGE (-1..10)
      -- -1 est 20%
      -- 1 est 25%
      -- 0 est 15%
      -- (3..6) est 10%
      -- le reste est 2%
      -- fin de définition
      -- mappages produits par "ECN Public Domain Software for Huffman encodings, version 1"
        MAPPING TO BITS {
              -1 TO '11'B,
              0 ..  1 TO '01'B .. '10'B,
              2 TO '0000001'B ,
              3 ..  5 TO '0001'B .. '0011'B,
              6 TO '00001'B,
              7 ..  8 TO '0000010'B .. '0000011'B,
              9 ..  10 TO '00000000'B .. '00000001'B
        }
WITH  my-self-delim-bits-encoding }
```

E.15.2 Le second exemple est le suivant:

```
my-int-encoding2 #My-Special-2 ::=
{ USE  #BITS
      -- ECN Huffman
      -- RANGE (-10..10)
      -- -10 est 20%
      -- 1 est 25%
      -- 5 est 15%
      -- (7..10) est 10%
      -- fin de définition
      -- mappages produits par "ECN Public Domain Software for Huffman encodings, version 1"
        MAPPING TO BITS {
              -10 TO '11'B ,
              1 TO '10'B ,
              5 TO '01'B ,
              7 ..  10 TO '0000'B .. '0011'B
        }
WITH my-self-delim-bits-encoding }
```

E.15.3 Le troisième exemple est le suivant:

```
my-int-encoding3 #My-Special-3 ::=
{ USE  #BITS
      -- ECN Huffman
      -- RANGE  (0..1000)
      -- (0..63) est 100%
      -- LE RESTE est 0%
      -- fin de définition
      -- mappages produits par "ECN Public Domain Software for Huffman encodings"
        MAPPING TO BITS {
              0 ..  62 TO '000001'B .. '111111'B,
              63 TO '0000001'B ,
              64 ..  150 TO '0000000110101001'B .. '0000000111111111'B,
              151 ..  1000 TO '00000000000000000'B .. '00000001101010001'B
        }
WITH my-self-delim-bits-encoding }
```

Annexe F

Informations complémentaires sur la notation de contrôle de codage (ECN)

(Cette annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

Des informations complémentaires et des liens sur la notation de contrôle de codage peuvent être trouvés sur le site Web suivant:

- http://asn1.elibel.tm.fr/ecn

Annexe G

Résumé de la notation ECN

(Cette annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

G.1 Symboles terminaux

Les symboles terminaux sont utilisés dans la présente Recommandation | Norme internationale.

G.1.1 Les items suivants sont définis dans l'article 8:

anystringexceptnonecend
encodingobjectreference
encodingobjectsetreference
encodingclassreference
"::="
".."
"{"
"}"
"("
")"
","
"."
"|"
ALL
AS
BEGIN
BER
BITS
BY
CER
COMPLETED
DECODE
DER
DISTRIBUTION
ENCODE
ENCODE-DECODE
ENCODING-CLASS
ENCODING-DEFINITIONS
END
EXCEPT
EXPORTS
FALSE
FIELDS
FROM
GENERATES
IF
IMPORTS
IN
LINK-DEFINITIONS
MAPPING
MAX
MIN
MINUS-INFINITY
NON-ECN-BEGIN
NON-ECN-END
NULL
OPTIONAL-ENCODING
OPTIONS
ORDERED
OUTER
PER-BASIC-ALIGNED
PER-BASIC-UNALIGNED
PER-CANONICAL-ALIGNED
PER-CANONICAL-UNALIGNED
PLUS-INFINITY
REFERENCE
REMAINDER
RENAMES
SIZE
STRUCTURE
STRUCTURED
TO
TRANSFORMS
TRUE
UNION
USE
USE-SET
VALUES
WITH

G.1.2 L'item suivant est défini en Annexe A:

REFERENCE

G.1.3 Les items suivants sont définis dans la Rec. UIT-T X.680 | ISO/CEI 8824-1:

bstring
cstring
hstring
identifier
modulereference
number
realnumber
typereference
"-"
";"
":"

ALL
EXCEPT
EXPORTS
FALSE
FROM
IMPORTS
MINUS-INFINITY
NULL
PLUS-INFINITY
TRUE

G.1.4 Les items suivants sont définis dans la Rec. UIT-T X.681 | ISO/CEI 8824-2 :

word
valuefieldreference
valuesetfieldreference

G.1.5 Les items suivants sont définis dans la Rec. UIT-T X.683 | ISO/CEI 8824-4:

"{<"
">}"

G.2 Productions

G.2.1 Les productions suivantes sont utilisées dans la présente Recommandation | Norme internationale, avec les items définis au § G.1 en tant que symboles terminaux:

```
ELMDefinition ::=
        ModuleIdentifier
        LINK-DEFINITIONS
        "::="
        BEGIN
        ELMModuleBody
        END

ELMModuleBody ::=
        Imports ?
        EncodingApplicationList

EncodingApplicationList ::=
        EncodingApplication
        EncodingApplicationList ?

EncodingApplication ::=
        ENCODE
        SimpleDefinedEncodingClass "," +
        CombinedEncodings

CombinedEncodings ::=
        WITH
        PrimaryEncodings
        CompletionClause ?

CompletionClause ::=
        COMPLETED BY
        SecondaryEncodings

PrimaryEncodings ::= EncodingObjectSet

SecondaryEncodings ::= EncodingObjectSet

EDMDefinition ::=
        ModuleIdentifier
```

```
            ENCODING-DEFINITIONS
            "::="
            BEGIN
            EDMModuleBody
            END

EDMModuleBody ::=
            Exports ?
            RenamesAndExports ?
            Imports ?
            EDMAssignmentList ?

EDMAssignmentList ::=
            EDMAssignment
            EDMAssignmentList ?

EDMAssignment ::=
                EncodingClassAssignment
            |   EncodingObjectAssignment
            |   EncodingObjectSetAssignment
            |   ParameterizedAssignment

RenamesAndExports ::=
            RENAMES
            ExplicitGenerationList ";"

ExplicitGenerationList ::=
            ExplicitGeneration
            ExplicitGenerationList ?

ExplicitGeneration ::=
            OptionalNameChanges
            FROM GlobalModuleReference

OptionalNameChanges ::=
            NameChanges |  GENERATES

NameChanges ::= NameChange NameChanges ?

NameChange ::=
            OriginalClassName
            AS
            NewClassName
            IN
            NameChangeDomain

OriginalClassName ::= SimpleDefinedEncodingClass | BuiltinEncodingClassReference

NewClassName ::= encodingclassreference

NameChangeDomain ::=
            IncludedRegions
            Exception ?

Exception ::=
            EXCEPT
            ExcludedRegions

IncludedRegions ::=
            ALL | RegionList

ExcludedRegions ::= RegionList

RegionList ::=
            Region "," +

Region ::=
            SimpleDefinedEncodingClass |
            ComponentReference

ComponentReference ::=
            SimpleDefinedEncodingClass
            "."
            ComponentIdList

ComponentIdList ::=
```

```
        identifier "." +

EncodingClassAssignment ::=
        encodingclassreference
        "::="
        EncodingClass

EncodingClass ::=
        BuiltinEncodingClassReference  |
        EncodingStructure

EncodingObjectAssignment ::=
        encodingobjectreference
        DefinedOrBuiltinEncodingClass
        "::="
        EncodingObject

EncodingObjectSetAssignment ::=
        encodingobjectsetreference
        #ENCODINGS
        "::="
        EncodingObjectSet
        CompletionClause ?

EncodingObjectSet ::=
        DefinedOrBuiltinEncodingObjectSet |
        EncodingObjectSetSpec

EncodingStructure ::=
        TaggedStructure       |
        UntaggedStructure

TaggedStructure ::=
        "["
        TagClass
        TagValue ?
        "]"
        UntaggedStructure

UntaggedStructure ::=
            DefinedEncodingClass
        |   EncodingStructureField
        |   EncodingStructureDefn

TagClass ::=
            DefinedEncodingClass
        |   TagClassReference

TagValue ::=
        "(" number ")"

EncodingStructureDefn ::=
            AlternativesStructure
        |   RepetitionStructure
        |   ConcatenationStructure

AlternativesStructure ::=
        AlternativesClass
        "{"
        NamedFields
        "}"

AlternativesClass ::=
            DefinedEncodingClass
        |   AlternativesClassReference

NamedFields ::= NamedField "," +

NamedField ::=
        identifier
        EncodingStructure

RepetitionStructure ::=
        RepetitionClass
        "{"
```

```
            identifier ?
            EncodingStructure
            "}"
            Size?

RepetitionClass ::=
                DefinedEncodingClass
            |   RepetitionClassReference

ConcatenationStructure ::=
            ConcatenationClass
            "{"
            ConcatComponents
            "}"

ConcatenationClass ::=
                DefinedEncodingClass
            |   ConcatenationClassReference

ConcatComponents ::=
            ConcatComponent "," *

ConcatComponent ::=
            NamedField
            ConcatComponentPresence ?

ConcatComponentPresence ::=
            OPTIONAL-ENCODING
            OptionalClass

OptionalClass ::=
                DefinedEncodingClass
            |   OptionalityClassReference

DefinedEncodingClass ::=
                encodingclassreference
            |   ExternalEncodingClassReference
            |   ParameterizedEncodingClass

DefinedOrBuiltinEncodingClass ::=
                DefinedEncodingClass
            |   BuiltinEncodingClassReference

DefinedEncodingObject ::=
                encodingobjectreference
            |   ExternalEncodingObjectReference
            |   ParameterizedEncodingObject

DefinedEncodingObjectSet ::=
                encodingobjectsetreference
            |   ExternalEncodingObjectSetReference
            |   ParameterizedEncodingObjectSet

DefinedOrBuiltinEncodingObjectSet ::=
                DefinedEncodingObjectSet
            |   BuiltinEncodingObjectSetReference

BuiltinEncodingObjectSetReference ::=
                PER-BASIC-ALIGNED
            |   PER-BASIC-UNALIGNED
            |   PER-CANONICAL-ALIGNED
            |   PER-CANONICAL-UNALIGNED
            |   BER
            |   CER
            |   DER

ExternalEncodingClassReference ::=
                modulereference "." encodingclassreference
            |   modulereference "." BuiltinEncodingClassReference

ExternalEncodingObjectReference ::=
            modulereference "." encodingobjectreference

ExternalEncodingObjectSetReference ::=
            modulereference "." encodingobjectsetreference
```

```
EncodingObjectSetSpec ::=
        "{"
        EncodingObjects UnionMark *
        "}"

EncodingObjects ::=
            DefinedEncodingObject
        |   DefinedEncodingObjectSet

UnionMark ::=
        "|"|
        UNION

EncodingObject ::=
            DefinedEncodingObject
        |   DefinedSyntax
        |   EncodeWith
        |   EncodeByValueMapping
        |   EncodeStructure
        |   DifferentialEncodeDecodeObject
        |   EncodingOptionsEncodingObject
        |   NonECNEncodingObject

EncodeWith ::=
        "{" ENCODE CombinedEncodings "}"

EncodeByValueMapping ::=
        "{"
        USE
        DefinedOrBuiltinEncodingClass
        MAPPING
        ValueMapping
        WITH
        ValueMappingEncodingObjects
        "}"

ValueMappingEncodingObjects ::=
            EncodingObject
        |   DefinedOrBuiltinEncodingObjectSet

DifferentialEncodeDecodeObject ::=
        "{"
        ENCODE-DECODE
        SpecForEncoding
        DECODE AS IF
        SpecForDecoders
        "}"

SpecForEncoding ::= EncodingObject

SpecForDecoders ::= EncodingObject

EncodingOptionsEncodingObject ::=
        "{"
        OPTIONS
        EncodingOptionsList
        WITH
        AlternativesEncodingObject
        "}"

EncodingOptionsList ::= OrderedEncodingObjectList

AlternativesEncodingObject ::= EncodingObject

NonECNEncodingObject::=
        NON-ECN-BEGIN
        AssignedIdentifier
        anystringexceptnonecnend
        NON-ECN-END

EncodeStructure ::=
        "{"
        ENCODE STRUCTURE
        "{"
        ComponentEncodingList
```

```
            StructureEncoding ?
            "}"
            CombinedEncodings ?
            "}"

StructureEncoding ::=
            STRUCTURED WITH
            TagEncoding ?
            EncodingOrUseSet

ComponentEncodingList ::=
            ComponentEncoding "," *

ComponentEncoding ::=
                NonOptionalComponentEncodingSpec
            |   OptionalComponentEncodingSpec

NonOptionalComponentEncodingSpec ::=
            identifier ?
            TagAndElementEncoding

OptionalComponentEncodingSpec ::=
            identifier
            TagAndElementEncoding
            OPTIONAL-ENCODING
            OptionalEncoding

TagAndElementEncoding ::=
            TagEncoding ?
            EncodingOrUseSet

TagEncoding ::= "[" EncodingOrUseSet "]"

OptionalEncoding ::= EncodingOrUseSet

EncodingOrUseSet ::=
                EncodingObject
            |   USE-SET

BuiltinEncodingClassReference ::=
                BitfieldClassReference
            |   AlternativesClassReference
            |   ConcatenationClassReference
            |   RepetitionClassReference
            |   OptionalityClassReference
            |   TagClassReference
            |   EncodingProcedureClassReference

BitfieldClassReference ::=
                #NUL
            |   #BOOL
            |   #INT
            |   #BITS
            |   #OCTETS
            |   #CHARS
            |   #PAD
            |   #BIT-STRING
            |   #BOOLEAN
            |   #CHARACTER-STRING
            |   #EMBEDDED-PDV
            |   #ENUMERATED
            |   #EXTERNAL
            |   #INTEGER
            |   #NULL
            |   #OBJECT-IDENTIFIER
            |   #OCTET-STRING
            |   #OPEN-TYPE
            |   #REAL
            |   #RELATIVE-OID
            |   #GeneralizedTime
            |   #UTCTime
            |   #ObjectDescriptor
            |   #BMPString
            |   #GeneralString
```

```
        |   #GraphicString
        |   #IA5String
        |   #NumericString
        |   #PrintableString
        |   #TeletexString
        |   #UniversalString
        |   #UTF8String
        |   #VideotexString
        |   #VisibleString

AlternativesClassReference ::=
            #ALTERNATIVES
        |   #CHOICE

ConcatenationClassReference ::=
            #CONCATENATION
        |   #SEQUENCE
        |   #SET

RepetitionClassReference ::=
            #REPETITION
        |   #SEQUENCE-OF
        |   #SET-OF

OptionalityClassReference ::=
            #OPTIONAL

TagClassReference ::=
            #TAG

EncodingProcedureClassReference ::=
            #TRANSFORM
        |   #CONDITIONAL-INT
        |   #CONDITIONAL-REPETITION
        |   #OUTER

EncodingStructureField ::=
            #NUL
        |   #BOOL
        |   #INT                    Bounds?
        |   #BITS                   Size?
        |   #OCTETS                 Size?
        |   #CHARS                  Size?
        |   #PAD
        |   #BIT-STRING  Size?
        |   #BOOLEAN
        |   #CHARACTER-STRING
        |   #EMBEDDED-PDV
        |   #ENUMERATED             Bounds?
        |   #EXTERNAL
        |   #INTEGER                Bounds?
        |   #NULL
        |   #OBJECT-IDENTIFIER
        |   #OCTET-STRING           Size?
        |   #OPEN-TYPE
        |   #REAL
        |   #RELATIVE-OID
        |   #GeneralizedTime
        |   #UTCTime
        |   #ObjectDescriptor       Size?
        |   #BMPString              Size?
        |   #GeneralString          Size?
        |   #GraphicString          Size?
        |   #IA5String              Size?
        |   #NumericString          Size?
        |   #PrintableString        Size?
        |   #TeletexString          Size?
        |   #UniversalString        Size?
        |   #UTF8String             Size?
        |   #VideotexString         Size?
        |   #VisibleString          Size?

Bounds ::= "(" EffectiveRange ")"
```

```
EffectiveRange ::=
        MinMax
    |   Fixed

Size    ::= "(" SIZE SizeEffectiveRange ")"

SizeEffectiveRange ::=
        "(" EffectiveRange ")"

MinMax ::=
        ValueOrMin
        ".."
        ValueOrMax

ValueOrMin    ::=
        SignedNumber
    |   MIN

ValueOrMax    ::=
        SignedNumber
    |   MAX

Fixed ::=  SignedNumber

ValueMapping ::=
        MappingByExplicitValues
    |   MappingByMatchingFields
    |   MappingByTransformEncodingObjects
    |   MappingByAbstractValueOrdering
    |   MappingByValueDistribution
    |   MappingIntToBits

MappingByExplicitValues ::=
        VALUES
        "{"
        MappedValues "," +
        "}"

MappedValues ::=
        MappedValue1
        TO
        MappedValue2

MappedValue1 ::= Value

MappedValue2 ::= Value

MappingByMatchingFields ::=
        FIELDS

MappingByTransformEncodingObjects ::=
        TRANSFORMS
        "{"
        OrderedTransformList
        "}"

OrderedTransformList ::= Transform "," +

Transform ::= EncodingObject

MappingByAbstractValueOrdering ::=
        ORDERED VALUES

MappingByValueDistribution ::=
        DISTRIBUTION
        "{"
        Distribution "," +
        "}"

Distribution ::=
        SelectedValues
        TO
        identifier

SelectedValues ::=
        SelectedValue
```

```
        |   DistributionRange
        |   REMAINDER

DistributionRange ::=
        DistributionRangeValue1
        ".."
        DistributionRangeValue2

SelectedValue ::= SignedNumber

DistributionRangeValue1 ::= SignedNumber

DistributionRangeValue2 ::= SignedNumber

MappingIntToBits ::=
        TO BITS
        "{"
        MappedIntToBits "," +
        "}"

MappedIntToBits ::=
            SingleIntValMap
        |   IntValRangeMap

SingleIntValMap ::=
        IntValue
        TO
        BitValue

IntValue ::= SignedNumber

BitValue ::=
        bstring     |
        hstring

IntValRangeMap ::=
        IntRange
        TO
        BitRange

IntRange ::=
        IntRangeValue1
        ".."
        IntRangeValue2

BitRange ::=
        BitRangeValue1
        ".."
        BitRangeValue2

IntRangeValue1 ::= SignedNumber

IntRangeValue2 ::= SignedNumber

BitRangeValue1 ::=
        bstring |
        hstring

BitRangeValue2 ::=
        bstring |
        hstring
```

G.2.2 Les productions suivantes sont définies dans la Rec. UIT-T X.680 | ISO/CEI 8824-1, telle que modifiée par l'Annexe A, avec les items définis au § G.1 en tant que symboles terminaux:

NOTE – Les productions arbitraires ne sont pas permises en notation ECN.

```
        ModuleIdentifier ::=
            modulereference
            DefinitiveIdentifier ?

        DefinitiveIdentifier ::=
                "{" DefinitiveObjIdComponentList "}"

        DefinitiveObjIdComponentList ::=
                DefinitiveObjIdComponent
```

```
    |   DefinitiveObjIdComponent DefinitiveObjIdComponentList

DefinitiveObjIdComponent ::=
        NameForm
    |   DefinitiveNumberForm
    |   DefinitiveNameAndNumberForm

NameForm ::= identifier

DefinitiveNumberForm ::= number

DefinitiveNameAndNumberForm ::= identifier "(" DefinitiveNumberForm ")"

Exports ::=
            EXPORTS SymbolsExported? ";" |
            EXPORTS ALL ";"

SymbolsExported ::= SymbolList

Imports ::= IMPORTS SymbolsImported? ";"

SymbolsImported ::= SymbolsFromModuleList

SymbolsFromModuleList ::=
    SymbolsFromModule               |
    SymbolsFromModuleList  SymbolsFromModule

SymbolsFromModule ::=
    SymbolList
            FROM
            GlobalModuleReference

GlobalModuleReference ::=
    modulereference AssignedIdentifier

AssignedIdentifier ::=
            DefinitiveIdentifier
        |   empty

SymbolList ::=
    Symbol              |
    SymbolList "," Symbol

Symbol ::=
            Reference
        |   ParameterizedReference
        |   BuiltinEncodingClassReference

Reference ::=
            encodingclassreference
        |   ExternalEncodingClassReference
        |   encodingobjectreference
        |   encodingobjectsetreference

Value ::=
            BuiltinValue
           | ReferencedValue
           | ObjectClassFieldValue

BuiltinValue ::=
            BitStringValue
            | BooleanValue
            | CharacterStringValue
            | ChoiceValue
            | EmbeddedPDVValue
            | EnumeratedValue
            | ExternalValue
            | InstanceOfValue
            | IntegerValue
            | NullValue
```

```
            | ObjectIdentifierValue
            | OctetStringValue
            | RealValue
            | RelativeOIDValue
            | SequenceValue
            | SequenceOfValue
            | SetValue
            | SetOfValue
            | TaggedValue

BitStringValue ::=
              bstring
            | hstring
            | "{" IdentifierList "}"
            |"{" "}"

BooleanValue ::=
              TRUE
            | FALSE

CharacterStringValue ::=
              RestrictedCharacterStringValue
            | UnrestrictedCharacterStringValue

RestrictedCharacterStringValue ::=
              cstring
            | CharacterStringList
            | Quadruple
            | Tuple

CharacterStringList  ::=  "{" CharSyms "}"

CharSyms  ::=
              CharsDefn
            | CharSyms "," CharsDefn

CharsDefn  ::=
              cstring
            | Quadruple
            | Tuple
            | AbsoluteCharReference

Quadruple     ::= "{"  Group  ","  Plane  ","  Row  ","  Cell "}"
Group       ::= number
Plane       ::= number
Row         ::= number
Cell        ::= number

Tuple ::= "{" TableColumn "," TableRow "}"
TableColumn ::= number
TableRow      ::= number

AbsoluteCharReference ::=
            ModuleIdentifier
            "."
             valuereference

UnrestrictedCharacterStringValue ::= SequenceValue

ChoiceValue ::= identifier ":" Value

EmbeddedPDVValue ::= SequenceValue

EnumeratedValue ::= identifier

ExternalValue ::= SequenceValue

IntegerValue ::=
              SignedNumber
            | identifier
```

```
SignedNumber ::=
        number      |
        "-" number

NullValue ::= NULL

ObjectIdentifierValue ::=
          "{" ObjIdComponentsList "}"
      | "{" DefinedValue ObjIdComponentsList "}"

ObjIdComponentsList ::=
        ObjIdComponents                |
        ObjIdComponents ObjIdComponentsList

ObjIdComponents ::=
        NameForm |
        NumberForm   |
        NameAndNumberForm

NameForm ::= identifier

NumberForm ::=
          number
        | DefinedValue

NameAndNumberForm ::= identifier "(" NumberForm ")"

OctetStringValue ::=
        bstring  |
        hstring

RealValue ::=
          NumericRealValue
        | SpecialRealValue

NumericRealValue ::=
          0
        | realnumber
        | "-" realnumber
        | SequenceValue

SpecialRealValue ::=
          PLUS-INFINITY
        | MINUS-INFINITY

RelativeOIDValue ::= "{" RelativeOidComponentsList "}"

RelativeOidComponentsList ::=
    RelativeOidComponents
|   RelativeOidComponents RelativeOidComponentsList

RelativeOidComponents ::=
    NumberForm
|   NameAndNumberForm
|   DefinedValue

SequenceValue ::=
        "{" ComponentValueList "}"  |
        "{"  "}"

ComponentValueList ::=
            NamedValue
          |   ComponentValueList "," NamedValue

NamedValue ::=
        identifier Value

SequenceOfValue ::=
        "{" ValueList "}"
```

```
        | "{" "}"

ValueList ::=
        Value
    | ValueList "," Value

SetValue ::=
        "{" ComponentValueList "}" |
        "{" "}"

SetOfValue ::=
        "{" ValueList "}" |
        "{" "}"

ValueSet ::= "{" ElementSetSpecs "}"

ElementSetSpecs ::=
            RootElementSetSpec
          | RootElementSetSpec "," "..."
        | "..." "," AdditionalElementSetSpec
        | RootElementSetSpec "," "..." "," AdditionalElementSetSpec

RootElementSetSpec ::= ElementSetSpec

ElementSetSpec ::=
          Unions
        | ALL Exclusions

Exclusions ::= EXCEPT  Elements

Unions ::=
          Intersections
        | UElems UnionMark Intersections

UElems ::= Unions

Intersections ::=
          IntersectionElements
        | IElems IntersectionMark IntersectionElements

IntersectionElements ::= Elements | Elems Exclusions

UnionMark  ::=
        "|"   |
        UNION


Elements  ::=
          SubtypeElements
        | ObjectSetElements
        |"(" ElementSetSpec ")"

SubtypeElements ::=
          SingleValue
        | ContainedSubtype
        |ValueRange
        | PermittedAlphabet
        | SizeConstraint
        | TypeConstraint
        | InnerTypeConstraints

SingleValue ::= Value
```

G.2.3 Les productions suivantes sont définies dans la Rec. UIT-T X.681 | ISO/CEI 8824-2, telle que modifiée par l'Annexe B, avec les items définis au § G.1 en tant que symboles terminaux:

```
DefinedSyntax ::= "{" DefinedSyntaxList ? "}"

DefinedSyntaxList ::= DefinedSyntaxToken  DefinedSyntaxList  ?
```

```
DefinedSyntaxToken ::=
        Literal
    |   Setting
Literal ::=
        word
    |   ","
Setting ::=
        Value
    |   ValueSet
    |   OrderedValueList
    |   EncodingObject
    |   EncodingObjectSet
    |   OrderedEncodingObjectList
    |   DefinedOrBuiltinEncodingClass
    |   OUTER

OrderedValueList ::= "{" Value "," + "}"

OrderedEncodingObjectList ::= "{" EncodingObject "," + "}"

InstanceOfValue ::= Value

EncodingClassFieldType ::=
    DefinedEncodingClass
    "."
    FieldName

FieldName ::= PrimitiveFieldName "." +

PrimitiveFieldName ::=
        valuefieldreference
    |   valuesetfieldreference
|   orderedvaluelistfieldreference
```

G.2.4 Les productions suivantes sont définies dans la Rec. UIT-T X.683 | ISO/CEI 8824-4 telle que modifiée par l'Annexe C, avec les items définis au § G.1 en tant que symboles terminaux:

```
ParameterizedAssignment ::=
    ParameterizedEncodingObjectAssignment
|   ParameterizedEncodingClassAssignment
|   ParameterizedEncodingObjectSetAssignment

ParameterizedEncodingObjectAssignment ::=
    encodingobjectreference
    ParameterList
    DefinedOrBuiltinEncodingClass
    "::="
    EncodingObject

ParameterizedEncodingClassAssignment ::=
    encodingclassreference
    ParameterList
    "::="
    EncodingClass

ParameterizedEncodingObjectSetAssignment ::=
    encodingobjectsetreference
    ParameterList
    #ENCODINGS
    "::="
    EncodingObjectSet

ParameterList ::= "{<" Parameter "," + ">}"

Parameter ::=
            ParamGovernor ":" DummyReference
          | DummyReference
```

```
ParamGovernor ::=
            Governor
          | DummyGovernor

Governor ::=
        EncodingClassFieldType
    |   REFERENCE
    |   DefinedOrBuiltinEncodingClass
    |   #ENCODINGS

DummyGovernor ::= DummyReference

DummyReference ::=
            encodingclassreference
        |   valuereference
        |   typereference
        |   identifier
        |   encodingobjectreference
        |   encodingobjectsetreference

ParameterizedReference ::=
                Reference
            |   Reference "{<"  ">}"

ParameterizedEncodingObject ::=
            SimpleDefinedEncodingObject
            ActualParameterList

SimpleDefinedEncodingObject ::=
        ExternalEncodingObjectReference
    |   encodingobjectreference

ParameterizedEncodingObjectSet ::=
    SimpleDefinedEncodingObjectSet
    ActualParameterList

SimpleDefinedEncodingObjectSet ::=
        ExternalEncodingObjectSetReference
    |   encodingobjectsetreference

ParameterizedEncodingClass ::=
    SimpleDefinedEncodingClass
    ActualParameterList

SimpleDefinedEncodingClass ::=
        ExternalEncodingClassReference
    |   encodingclassreference

ActualParameterList ::= "{<" ActualParameter "," + ">}"

ActualParameter ::=
            Value
        |   ValueSet
        |   OrderedValueList
        |   DefinedOrBuiltinEncodingClass
        |   EncodingObject
        |   EncodingObjectSet
        |   OrderedEncodingObjectList
        |   identifier
        |   STRUCTURE
        |   OU
```

SÉRIES DES RECOMMANDATIONS UIT-T

Série A	Organisation du travail de l'UIT-T
Série B	Moyens d'expression: définitions, symboles, classification
Série C	Statistiques générales des télécommunications
Série D	Principes généraux de tarification
Série E	Exploitation générale du réseau, service téléphonique, exploitation des services et facteurs humains
Série F	Services de télécommunication non téléphoniques
Série G	Systèmes et supports de transmission, systèmes et réseaux numériques
Série H	Systèmes audiovisuels et multimédias
Série I	Réseau numérique à intégration de services
Série J	Réseaux câblés et transmission des signaux radiophoniques, télévisuels et autres signaux multimédias
Série K	Protection contre les perturbations
Série L	Construction, installation et protection des câbles et autres éléments des installations extérieures
Série M	RGT et maintenance des réseaux: systèmes de transmission, circuits téléphoniques, télégraphie, télécopie et circuits loués internationaux
Série N	Maintenance: circuits internationaux de transmission radiophonique et télévisuelle
Série O	Spécifications des appareils de mesure
Série P	Qualité de transmission téléphonique, installations téléphoniques et réseaux locaux
Série Q	Commutation et signalisation
Série R	Transmission télégraphique
Série S	Equipements terminaux de télégraphie
Série T	Terminaux des services télématiques
Série U	Commutation télégraphique
Série V	Communications de données sur le réseau téléphonique
Série X	**Réseaux de données et communication entre systèmes ouverts**
Série Y	Infrastructure mondiale de l'information et protocole Internet
Série Z	Langages et aspects généraux logiciels des systèmes de télécommunication

Genève, 2003