Union internationale des télécommunications

UIT-T

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

X.680

(07/2002)

SÉRIE X: RÉSEAUX DE DONNÉES, COMMUNICATION ENTRE SYSTÈMES OUVERTS ET SÉCURITÉ

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

Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification de la notation de base

Recommandation UIT-T X.680

RECOMMANDATIONS UIT-T DE LA SÉRIE X

RÉSEAUX DE DONNÉES, COMMUNICATION ENTRE SYSTÈMES OUVERTS ET SÉCURITÉ

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.379
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.889
Applications génériques de l'ASN.1	X.890–X.899
TRAITEMENT RÉPARTI OUVERT	X.900–X.999
SÉCURITÉ DES TÉLÉCOMMUNICATIONS	X.1000–

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

NORME INTERNATIONALE ISO/CEI 8824-1
RECOMMANDATION UIT-T X.680

Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification de la notation de base

Résumé

La présente Recommandation | Norme internationale spécifie une notation dite notation de syntaxe abstraite numéro un (ASN.1) pour la définition de la syntaxe de données informationnelles. Elle définit un certain nombre de types de donnée simples et spécifie une notation pour y faire référence et en spécifier les valeurs.

La notation ASN.1 peut être utilisée chaque fois qu'il est nécessaire de définir la syntaxe abstraite d'informations sans imposer de contrainte sur la manière de coder ces informations en vue de leur transmission.

Source

La Recommandation UIT-T X.680 a été approuvée le 14 juillet 2002 par la Commission d'études 17 (2001-2004) de l'UIT-T selon la procédure définie dans la Recommandation UIT-T A.8. Un texte identique est publié comme Norme Internationale ISO/CEI 8824-1.

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.

Le respect de cette Recommandation se fait à titre volontaire. Cependant, il se peut que la Recommandation contienne certaines dispositions obligatoires (pour assurer, par exemple, l'interopérabilité et l'applicabilité) et considère que la Recommandation est respectée lorsque toutes ces dispositions sont observées. Le futur d'obligation et les autres moyens d'expression de l'obligation comme le verbe "devoir" ainsi que leurs formes négatives servent à énoncer des prescriptions. L'utilisation de ces formes ne signifie pas qu'il est obligatoire de respecter la Recommandation.

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 présente Recommandation | Norme internationale présente une notation normalisée pour la définition de types de donnée et de leurs valeurs. Un *type de donnée* (en abrégé, un *type*) est une catégorie informationnelle (une information numérique, textuelle, iconographique ou vidéo par exemple). Une *valeur de donnée* (en abrégé une *valeur*) est une instance d'un tel type. La présente Recommandation | Norme internationale définit plusieurs types de base et les valeurs qui leur correspondent, ainsi que les règles pour les combiner en types et valeurs plus complexes.

Dans certaines architectures de protocole, chaque message est spécifié comme la valeur binaire d'une séquence d'octets. Les rédacteurs de normes ont cependant besoin de définir des types de données vraiment complexes afin d'exprimer leurs messages, quelle que soit leur représentation binaire. Afin de spécifier ces types de données, ils ont besoin d'une notation qui ne détermine pas nécessairement la représentation de chaque valeur, ce qui est le cas de la notation de syntaxe abstraite numéro un (ASN.1). Cette notation est complétée par la spécification d'un ou de plusieurs algorithmes appelés *règles de codage*, qui déterminent la valeur des octets exprimant la sémantique applicative (appelée *syntaxe de transfert*). La Rec. UIT-T X.690 | ISO/CEI 8825-1, la Rec. UIT-T X.691 | ISO/CEI 8825-2 et la Rec. UIT-T X.693 | ISO/CEI 8825-4 spécifient trois familles de règles de codage normalisées, appelées *règles de codage de base* (BER, *basic encoding rules*), *règles de codage compact* (PER, *packed encoding rules*) et *règles de codage XML* (XER, *XML encoding rules*).

Certains utilisateurs souhaitent redéfinir leurs protocoles existants au moyen de la notation ASN.1 mais ne peuvent pas utiliser les règles de codage normalisées parce qu'ils ont besoin de conserver leurs représentations binaires existantes. D'autres utilisateurs souhaitent avoir un contrôle plus complet de la représentation exacte des bits transmis (la syntaxe de transfert). Ces exigences sont prises en compte par la Rec. UIT-T X.692 | ISO/CEI 8825-3, qui spécifie une *notation de contrôle de codage* (ECN, *encoding control notation*) pour la notation ASN.1. La notation ECN permet aux concepteurs de spécifier formellement la syntaxe abstraite d'un protocole au moyen de la notation ASN.1 mais de prendre ensuite (s'ils le souhaitent) le contrôle complet ou partiel des bits transmis en rédigeant une spécification ECN auxiliaire (qui peut faire référence à des règles de codage normalisées pour certaines parties du codage).

Une technique très générale pour définir un type complexe au niveau abstrait consiste à définir un petit nombre de *types simples* en définissant toutes leurs valeurs possibles, puis de combiner ces types simples de diverses façons. A titre d'exemple, on peut citer les procédés suivants pour définir de nouveaux types:

a) étant donné une liste (ordonnée) de types existants, une valeur peut être constituée sous la forme d'une séquence (ordonnée) de valeurs, en prenant une valeur de chacun des types existants; la collection de toutes les valeurs possibles ainsi obtenues forme un nouveau type (si les types de la liste sont tous distincts, ce mécanisme peut être étendu pour permettre l'omission de certaines valeurs de la liste);

b) étant donné un ensemble non ordonné de types (distincts) existants, une valeur peut être constituée sous la forme d'un ensemble (non ordonné) de valeurs, en prenant une valeur de chacun des types existants; la collection de tous les ensembles non ordonnés possibles ainsi obtenus forme un nouveau type (là encore, le mécanisme peut être étendu pour permettre l'omission de certaines valeurs);

c) étant donné un type simple existant, une valeur peut être constituée sous la forme d'une liste (ordonnée) ou un ensemble (non ordonné) de zéro, une ou plusieurs valeurs du type; la collection de tous les ensembles ou listes possibles ainsi obtenus forme un nouveau type;

d) étant donné une liste de types (distincts), on peut choisir une valeur de l'un quelconque de ces types; l'ensemble de toutes les valeurs possibles ainsi obtenues forme un nouveau type;

e) étant donné un type, un nouveau type peut être constitué sous la forme d'un sous-ensemble de ce type, en appliquant à ses valeurs une contrainte structurelle ou une relation d'ordre quelconque.

Un aspect important d'une telle combinaison des types est que les règles de codage doivent permettre de reconnaître les différentes structures ainsi créées, assurant ainsi un codage non ambigu de la collection de valeurs des types de base. Ainsi, une *étiquette* est affectée à chaque type défini au moyen de la notation spécifiée dans la présente Recommandation | Norme internationale pour en permettre le codage non ambigu des valeurs.

Les étiquettes sont principalement destinées au traitement machine et ne sont pas essentielles à la forme de notation lisible par l'homme, définie dans la présente Recommandation | Norme internationale. Toutefois, quand il sera nécessaire de distinguer certains types, on sera amené à leur imposer d'avoir des étiquettes distinctes. L'affectation des étiquettes constitue donc un aspect important de l'utilisation de la présente notation, mais (depuis 1994) il est possible de spécifier une affectation automatique des étiquettes.

NOTE 1 – Dans la présente Recommandation | Norme internationale, des valeurs d'étiquette sont affectées à tous les types simples et mécanismes de structuration. Les restrictions imposées à l'utilisation de la notation garantissent de pouvoir utiliser les étiquettes en transfert pour identifier les valeurs de façon non ambiguë.

Une spécification ASN.1 sera produite initialement avec un ensemble de types ASN.1 complètement définis. Il peut toutefois être nécessaire, lors d'une étape ultérieure, de modifier ces types (en général par ajout de composants supplémentaires dans un type séquence ou ensemble). Les règles de codage doivent fournir une prise en charge adéquate si cette modification doit être faite de manière à permettre à des implémentations utilisant les anciennes définitions de type de communiquer d'une manière définie avec des implémentations utilisant les nouvelles définitions. La notation ASN.1 prend en charge un *marqueur d'extension* pour un certain nombre de types. Ceci signale aux règles de codages que le concepteur a l'intention que ce type fasse partie d'une série de types apparentés (c'est-à-dire, de versions d'un même type initial) appelée *série d'extensions* et que les règles de codage doivent obligatoirement permettre le transfert d'informations entre des implémentations utilisant des types différents liés par l'appartenance à une même série d'extensions.

Les paragraphes 10 à 31 inclus définissent les types simples pris en charge par la notation ASN.1 et spécifient la notation à utiliser pour faire référence à des types simples et pour définir de nouveaux types au moyen de ces types simples. Ils spécifient également les notations à utiliser pour spécifier les valeurs de types définis en ASN.1. Deux notations de valeur sont définies. La première, appelée notation de valeur ASN.1 de base, fait partie de la notation ASN.1 depuis la première spécification de cette dernière. La deuxième, appelée notation de valeur ASN.1 XML, est une notation de valeur utilisant le langage de balisage extensible (XML, *extensible markup language*).

NOTE 2 – La notation de valeur XML permet de représenter des valeurs ASN.1 au moyen du langage XML. Par conséquent, la définition d'un type ASN.1 spécifie aussi la structure et le contenu d'un élément XML. Ainsi, la notation ASN.1 constitue un langage de schéma simple pour le langage XML.

Les paragraphes 33 à 34 inclus définissent les types pris en charge par la notation ASN.1 pour exprimer le codage complet des types ASN.1.

Les paragraphes 35 à 40 inclus définissent les types de chaîne de caractères.

Les paragraphes 41 à 44 inclus définissent certains types considérés comme étant d'utilité générale mais qui ne nécessitent aucune règle de codage supplémentaire.

Les paragraphes 45 à 47 inclus définissent une notation qui permet de définir des sous-types à partir des valeurs d'un type parent.

Le paragraphe 48 définit une notation qui permet à des types ASN.1 spécifiés dans la "version 1" d'une spécification d'être identifiés comme susceptibles d'être étendus dans la "version 2" et qui permet de faire une liste à part des ajouts faits dans les versions ultérieures et de les identifier avec leur numéro de version.

Le paragraphe 49 définit une notation qui permet à des définitions de type ASN.1 de contenir une indication du traitement d'erreur prévu au cas où des codages reçus correspondraient à des valeurs se trouvant en dehors de celles spécifiées dans la définition normalisée en vigueur.

L'Annexe A, qui fait partie intégrante de la présente Recommandation | Norme internationale, spécifie les expressions régulières en notation ASN.1.

L'Annexe B, qui fait partie intégrante de la présente Recommandation | Norme internationale, spécifie des règles applicables à la compatibilité des types et des valeurs.

L'Annexe C, qui fait partie intégrante de la présente Recommandation | Norme internationale, récapitule les valeurs d'identificateur d'objet et de descripteur d'objet affectées dans la série de Recommandations | Normes internationales relatives à la notation ASN.1.

L'Annexe D, qui ne fait pas partie intégrante de la présente Recommandation | Norme internationale, décrit les arcs de niveau supérieur de l'arbre d'enregistrement des identificateurs d'objet.

L'Annexe E, qui ne fait pas partie intégrante de la présente Recommandation | Norme internationale, fournit des exemples et des indications relatifs à l'utilisation de la notation ASN.1.

L'Annexe F, qui ne fait pas partie intégrante de la présente Recommandation | Norme internationale, est un exposé didactique sur les chaînes de caractères ASN.1.

L'Annexe G, qui ne fait pas partie intégrante de la présente Recommandation | Norme internationale, est un exposé didactique sur le modèle ASN.1 d'extension de type.

L'Annexe H, qui ne fait pas partie intégrante de la présente Recommandation | Norme internationale, fournit un récapitulatif de la notation ASN.1 en utilisant la notation du § 5.

NORME INTERNATIONALE
RECOMMANDATION UIT

Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification de la notation de base

1 Domaine d'application

La présente Recommandation | Norme internationale spécifie une notation normalisée appelée notation de syntaxe abstraite numéro un (ASN.1) servant à définir les types de donnée, les valeurs et les contraintes imposées à ces types.

La présente Recommandation | Norme internationale:

- définit un certain nombre de types simples, avec leurs étiquettes, et spécifie une notation pour faire référence à ces types et pour spécifier leurs valeurs;
- définit des mécanismes pour construire de nouveaux types à partir de types plus élémentaires, et spécifie une notation pour définir de tels types, leur affecter des étiquettes, et en spécifier les valeurs;
- définit (par référence à d'autres Recommandations | Normes internationales) les jeux de caractères à utiliser en notation ASN.1.

La notation ASN.1 peut être utilisée chaque fois qu'il est nécessaire de définir la syntaxe abstraite d'informations.

Il est fait référence à la notation ASN.1 dans d'autres normes qui définissent les règles de codage des types ASN.1.

2 Références normatives

Les Recommandations et les 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 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 de l'UIT-T actuellement en vigueur.

2.1 Recommandations | Normes internationales identiques

- Recommandation CCITT 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 les amendements).
- 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.*
- Recommandation UIT-T X.692 (2002) | ISO/CEI 8825-3:2002, *Technologies de l'information – Règles de codage ASN.1: spécification de la notation de contrôle de codage (ECN)*.
- Recommandation UIT-T X.693 (2001) | ISO/CEI 8825-4:2002, *Technologies de l'information – Règles de codage ASN.1: règles de codage XML (XER)*.

2.2 Autres références

- Recommandation UIT-R TF.460-5 (1997), *Emissions de fréquences étalon et de signaux horaires*.
- Recommandation CCITT T.100 (1988), *Echange international d'informations pour le Vidéotex interactif*.
- Recommandation UIT-T T.101 (1994), *Interfonctionnement international pour les services Vidéotex*.
- ISO *Registre international des jeux de caractères codés à utiliser avec une séquence d'échappement.*
- ISO/CEI 646:1991, *Technologies de l'information – Jeux ISO de caractères codés à 7 éléments pour l'échange d'information.*
- ISO/CEI 2022:1994, *Technologies de l'information – Structure de code de caractères et techniques d'extension*.
- ISO/CEI 6523:1998, *Technologies de l'information – Structures pour l'identification des organisations et des parties d'organisations*.
- ISO/CEI 7350:1991, *Technologies de l'information – Enregistrement des répertoires de caractères graphiques de l'ISO/CEI 10367*.
- ISO 8601:2000, *Eléments de données et formats d'échange – Echange d'information – Représentation de la date et de l'heure*.
- ISO/CEI 10646-1:2000, *Technologies de l'information – Jeu universel de caractères à plusieurs octets – Partie 1: Architecture et table multilingue.*
- The Unicode Standard, Version 3.2.0:2002. The Unicode Consortium. (Reading, MA, Addison-Wesley).

  NOTE 1 – La référence ci-dessus est incluse car elle fournit des noms pour les caractères de commande.
- W3C XML 1.0:2000, *Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, Copyright © [6 October 2000] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/2000/REC-xml-20001006.*

NOTE 2 – La référence à un document figurant dans la présente Recommandation | Norme internationale ne donne pas à ce document en tant que tel le statut de Recommandation ou de Norme internationale.

3 Définitions

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

3.1 Spécification des objets informationnels

La présente Recommandation | Norme internationale utilise les termes suivants, définis dans la Rec. UIT-T X.681 | ISO/CEI 8824-2:

a) objet informationnel;

b) classe d'objets informationnels;

c) ensemble d'objets informationnels;

d) type instance-de;

e) type champ de classe d'objets.

3.2 Spécification des contraintes

La présente Recommandation | Norme internationale utilise les termes suivants, définis dans la Rec. UIT-T X.682 | ISO/CEI 8824-3:

a) contrainte relationnelle entre composants;

b) contrainte tabulaire.

3.3 Paramétrage des spécifications ASN.1

La présente Recommandation | Norme internationale utilise les termes suivants, définis dans la Rec. UIT-T X.683 | ISO/CEI 8824-4:

a) type paramétré;

b) valeur paramétrée.

3.4 Structure pour l'identification des organisations

La présente Recommandation | Norme internationale utilise les termes suivants, définis dans l'ISO/CEI 6523:

a) organisation émettrice;

b) code d'organisation;

c) désignateur de code international (ICD, *international code designator*).

3.5 Jeu de caractères universels codés sur plusieurs octets (UCS)

La présente Recommandation | Norme internationale utilise les termes suivants, définis dans l'ISO/CEI 10646-1:

a) table multilingue (BMP, *basic multilingual plane*);

b) cellule;

c) caractère de combinaison;

d) symbole graphique;

e) groupe;

f) sous-ensemble limité;

g) plan;

h) rangée;

i) sous-ensemble sélectionné.

3.6 Autres définitions

3.6.1 **caractère abstrait**: valeur abstraite qui est utilisée pour l'organisation, le contrôle ou la représentation de données textuelles.

NOTE – L'Annexe F donne une description plus complète de l'expression "caractère abstrait".

3.6.2 **valeur abstraite**: valeur dont la définition est basée uniquement sur le type utilisé pour exprimer une certaine sémantique, indépendamment de la manière dont elle est représentée par un codage quelconque.

NOTE – Exemples de valeurs abstraites: des valeurs du type entier, du type booléen, d'un type chaîne de caractères ou d'un type qui est une séquence (ou un choix) d'un entier et d'un booléen.

3.6.3 **jeu de caractères ASN.1**: jeu de caractères spécifié au § 10 et utilisé en notation ASN.1.

3.6.4 **spécification ASN.1**: collection d'un ou plusieurs modules ASN.1.

3.6.5 **type associé**: type utilisé seulement pour définir la valeur et la notation de sous-type d'un type donné.

NOTE – Des types associés sont définis dans la présente Recommandation | Norme internationale lorsqu'il est nécessaire de bien indiquer qu'il existe une différence significative entre la façon dont le type est défini en ASN.1 et la façon de le coder. Les types associés n'apparaissent pas dans les spécifications d'utilisateurs.

3.6.6 **type chaîne binaire (bitstring type)**: type simple dont chaque valeur distinctive est une séquence ordonnée de zéro, un ou plusieurs bits.

NOTE – Lorsqu'il est nécessaire d'exprimer des codages imbriqués d'une valeur abstraite, l'utilisation d'un type chaîne binaire (ou chaîne d'octets) sans contrainte de contenu (voir le § 11 de la Rec. UIT-T X.682 | ISO/CEI 8824-3) est déconseillée. En revanche, l'utilisation du type valeur pdv imbriquée (voir § 33) constitue un mécanisme plus souple, permettant d'annoncer la syntaxe abstraite et le codage de la valeur abstraite qui est imbriquée.

3.6.7 **type booléen (boolean type)**: type simple ayant deux valeurs distinctives possibles.

3.6.8 **propriété de caractère**: ensemble des informations associées à une cellule donnée d'une table définissant un répertoire de caractères.

NOTE – Ces informations comprennent normalement tout ou partie des éléments suivants:

a) un symbole graphique;

b) le nom du caractère;

c) la définition des fonctions associées au caractère lorsqu'il est utilisé dans des environnements particuliers;

d) la question de savoir s'il représente un chiffre;

e) un caractère associé qui diffère uniquement par la casse (minuscule/majuscule).

3.6.9 syntaxe abstraite de caractères: toute syntaxe abstraite dont les valeurs sont toutes les chaînes composées de zéro, un ou plusieurs caractères appartenant à une collection de caractères donnée.

3.6.10 répertoire de caractères: caractères d'un jeu de caractères indépendamment de toute considération quant à la manière dont ces caractères sont codés.

3.6.11 types de chaîne de caractères (character string types): types simples dont les valeurs sont des chaînes de caractères pris dans un jeu donné.

3.6.12 syntaxe de transfert de caractères: toute syntaxe de transfert pour une syntaxe abstraite de caractères.

NOTE – La notation ASN.1 ne prend pas en charge les syntaxes de transfert de caractères qui ne codent pas toute chaîne de caractères sur un nombre entier d'octets.

3.6.13 types choix (choice types): types définis par l'indication d'une liste de types distincts; chaque valeur d'un type choix dérive d'une valeur de l'un quelconque des types composants.

3.6.14 type composant: un des types indiqués en référence dans une déclaration de type **`CHOICE`**, **`SET`**, **`SEQUENCE`**, **`SET OF`** ou **`SEQUENCE OF`**.

3.6.15 contrainte: notation qui, associée à un type, permet d'en définir un sous-type.

3.6.16 contrainte de contenu: contrainte associée à un type chaîne binaire ou à un type chaîne d'octets, indiquant que le contenu doit correspondre à un codage d'un type ASN.1 spécifié ou que des procédures spécifiées doivent être utilisées pour produire et traiter le contenu.

3.6.17 caractères de contrôle: caractères apparaissant dans certains répertoires de caractères et ayant reçu un nom (et éventuellement une fonction précise en relation avec certains environnements), mais qui ne se sont pas vus affecter un symbole graphique et qui ne sont pas non plus des caractères d'espacement.

NOTE – TABULATION HORIZONTALE (9) et LIGNE SUIVANTE (10) sont des exemples de caractères de contrôle qui se sont vus affecter des fonctions de formatage dans un environnement d'édition. ÉCHAPPEMENT DE LIAISON DE DONNÉES (16) est un exemple de caractère de contrôle qui s'est vu affecter une fonction dans un environnement de communication.

3.6.18 temps universel coordonné (**UTC**, ***coordinated universal time***): échelle de temps conservée par le Bureau international de l'heure, et servant de base à la diffusion coordonnée des fréquences étalon et des signaux horaires.

NOTE 1 – L'origine de cette définition est la Rec. UIT-R TF.460-5. L'UIT-R a également défini le sigle UTC du temps universel coordonné.

NOTE 2 – L'UTC et le temps moyen de Greenwich (GMT, *Greenwich mean time*) sont deux normes de temps qui indiquent le même temps pour la plupart des applications pratiques.

3.6.19 élément: valeur d'un type gouvernant ou objet informationnel d'une classe d'objets informationnels gouvernante, distinguable de toutes les autres valeurs du même type ou de tous les autres objets informationnels de la même classe, respectivement.

3.6.20 ensemble d'éléments: ensemble d'éléments, qui sont tous des valeurs d'un type gouvernant ou des objets informationnels d'une classe gouvernante.

NOTE – La classe gouvernante est définie au § 3.4.7 de la Rec. UIT-T X.681 | ISO/CEI 8824-2.

3.6.21 type valeur pdv imbriquée (embedded-pdv type): type dont l'ensemble des valeurs est, formellement, la réunion des ensembles de valeurs dans toutes les syntaxes abstraites possibles. Ce type peut être utilisé dans une spécification ASN.1 dans le protocole de laquelle on souhaite utiliser une valeur abstraite dont le type peut être défini extérieurement à cette spécification ASN.1. Il comporte un identificateur de la syntaxe abstraite (le type) de la valeur abstraite exprimée ainsi qu'un identificateur des règles de codage utilisées pour coder cette valeur abstraite.

3.6.22 codage: séquence binaire résultant de l'application d'un ensemble de règles de codage à une valeur abstraite.

3.6.23 règles de codage (ASN.1): règles qui spécifient la représentation des valeurs de types ASN.1 durant leur transfert; elles permettent aussi de retrouver les valeurs à partir de leur représentation, une fois leur type connu.

NOTE – Concernant la spécification des règles de codage, les différentes notations de types (et de valeurs) données en référence, qui peuvent constituer d'autres notations pour des types (et des valeurs) prédéfinis, ne sont pas applicables.

3.6.24 types énuméré (enumerated types): types simples dont les valeurs sont représentées par des identificateurs distincts dans le cadre de la notation du type.

3.6.25 ajout d'extension: une des notations ajoutées dans une série d'extensions. Pour les types ensemble, séquence et choix, un ajout d'extension est constitué d'un seul groupe d'ajouts d'extension ou d'un seul type composant. Pour les types énumérés, il s'agit de l'ajout d'une seule énumération supplémentaire. Pour une contrainte, il s'agit de l'ajout d'un (seul) élément sous-type.

NOTE – Les ajouts d'extension sont rangés à la fois dans un ordre textuel (à la suite du marqueur d'extension) et dans un ordre logique (dans l'ordre croissant des valeurs de l'énumération et, dans le cas des différentes formes d'un type **`CHOICE`**, dans l'ordre croissant des étiquettes).

3.6.26 groupe d'ajouts d'extension: un ou plusieurs composants d'un type ensemble, séquence ou choix apparaissant entre des crochets de version. Un groupe d'ajouts d'extension est utilisé afin d'identifier clairement les composants d'un type ensemble, séquence ou choix qui ont été ajoutés dans une version particulière d'un module ASN.1 et peut identifier cette version à l'aide d'un simple entier.

3.6.27 type d'ajout d'extension: type contenu dans un groupe d'ajouts d'extension ou type composant unique qui est lui-même un ajout d'extension (dans ce cas, le type n'est pas contenu dans un groupe d'ajouts d'extension).

3.6.28 contrainte extensible: contrainte de sous-typage avec un marqueur d'extension au niveau extérieur, ou qui est extensible grâce à l'utilisation d'opérations arithmétiques sur des ensembles de valeurs extensibles.

3.6.29 point d'insertion d'extension (ou point d'insertion): position au niveau de laquelle des ajouts d'extension sont insérés dans une définition de type. Cette position correspond à la fin de la notation du type immédiatement précédent dans la série d'extensions, s'il existe une seule occurrence de points de suspension dans la définition du type, ou immédiatement avant la deuxième occurrence de points de suspension s'il existe une paire de marqueurs d'extension dans la définition du type.

NOTE – Un point d'insertion au plus peut figurer à l'intérieur des composants d'un type choix, séquence, ou ensemble.

3.6.30 marqueur d'extension: indicateur syntaxique (points de suspension) figurant dans tous les types qui font partie d'une série d'extensions.

3.6.31 paire de marqueurs d'extension: paire de marqueurs d'extension entre lesquels les ajouts d'extension sont insérés.

3.6.32 apparenté par extension: deux types possédant la même racine d'extension dont l'un a été créé par insertion de zéro, un ou plusieurs ajouts dans l'autre.

3.6.33 racine d'extension: type extensible qui est le premier d'une série d'extensions. Il véhicule soit le marqueur d'extension sans notation supplémentaire autre que des commentaires et des espaces blancs entre le marqueur d'extension et le caractère "}" ou ")", soit une paire de marqueurs d'extension sans notation supplémentaire autre qu'une virgule unique, des commentaires et des espaces blancs entre les marqueurs d'extension.

NOTE – Le premier type d'une série d'extensions doit être une racine d'extension.

3.6.34 série d'extensions: série de types ASN.1 pouvant être rangés dans un ordre tel que chacun des types successifs de la série soit obtenu par ajout de texte au niveau du point d'insertion d'extension.

3.6.35 type extensible: type avec un marqueur d'extension ou auquel une contrainte extensible a été appliquée.

3.6.36 référence externe: référence de type, référence de valeur, objet informationnel, référence de classe, référence d'objet informationnel ou référence d'ensemble d'objets informationnels (pouvant être paramétré), qui est défini dans un module quelconque autre que celui dans lequel il y est fait référence, la référence à la définition s'effectuant en préfixant le nom du module de définition au nom de l'élément cité.

EXEMPLE – **`NomModule.RéférenceType`**

3.6.37 type externe (external type): type apparaissant dans une spécification ASN.1 et comportant une valeur dont le type peut être défini à l'extérieur de cette spécification. Le type externe comporte une identification du type de la valeur exprimée.

3.6.38 faux (false): une des deux valeurs distinctives du type booléen (voir également "Vrai").

3.6.39 (type) gouvernant; gouverneur: référence ou définition de type qui affecte l'interprétation d'une partie de la syntaxe ASN.1, nécessitant que cette partie fasse référence à des valeurs du type gouvernant.

3.6.40 définitions de types identiques: deux instances de la production "Type" ASN.1 (voir le § 16) sont définies comme des définitions de types identiques si, après application des transformations spécifiées dans l'Annexe B, elles sont constituées de listes ordonnées identiques d'unités lexicales identiques (voir le § 11).

3.6.41 type entier (integer type): type simple dont les valeurs distinctives sont les entiers relatifs (les positifs, les négatifs, et la valeur zéro en tant que valeur unique).

NOTE – Lorsque des règles de codage particulières limitent l'intervalle de variation possible des entiers, ces limites sont choisies de façon à ne gêner en rien les utilisateurs de la notation ASN.1.

3.6.42 unité lexicale: séquence nommée de caractères du jeu de caractères ASN.1, spécifiée au § 11, et utilisée dans la formation de la notation ASN.1.

3.6.43 module: une ou plusieurs instances d'utilisation de la notation ASN.1 pour la définition de types, de valeurs, d'ensembles de valeurs, de classes d'objets informationnels, d'objets informationnels et d'ensembles d'objets informationnels (ainsi que leur variante paramétrée), encapsulées au moyen de la notation de module ASN.1 (voir le § 12).

NOTE – Les expressions classe d'objets informationnels, etc., sont spécifiées dans la Rec. UIT-T X.681 | ISO/CEI 8824-2 et le paramétrage est spécifié dans la Rec. UIT-T X.683 | ISO/CEI 8824-4.

3.6.44 type néant (null type): type simple comprenant une seule valeur, appelée néant.

3.6.45 objet: information, définition ou spécification bien définie, nécessitant un nom afin de l'identifier dans une instance de communication.

NOTE – Un tel objet peut être un objet informationnel tel que défini dans la Rec. UIT-T X.681 | ISO/CEI 8824-2.

3.6.46 type descripteur d'objet (object descriptor type): type dont les valeurs distinctives sont des textes en langage naturel décrivant brièvement un objet (voir § 3.6.45).

NOTE – Une valeur de descripteur d'objet est généralement associée à un seul objet. Seule la valeur d'identificateur d'objet identifie sans ambiguïté l'objet.

3.6.47 identificateur d'objet: valeur unique globalement, associée à un objet pour l'identifier sans ambiguïté.

3.6.48 type identificateur d'objet (object identifier type): type simple dont les valeurs sont tous les identificateurs d'objet affectés conformément aux règles de la série de Rec. UIT-T X.660 | ISO/CEI 9834.

NOTE – Les règles de la série de Rec. UIT-T X.660 | ISO/CEI 9834 permettent à des autorités très diverses d'associer indépendamment les unes des autres des identificateurs à des objets.

3.6.49 type chaîne d'octets (octetstring type): type simple dont chaque valeur distinctive est une séquence ordonnée de zéro, un ou plusieurs octets (l'octet étant une séquence ordonnée de huit bits).

3.6.50 interconnexion des systèmes ouverts: architecture pour la communication entre ordinateurs, dans laquelle sont définis un certain nombre de termes qui, employés dans la présente Recommandation | Norme internationale, sont précédés de l'abréviation "OSI".

NOTE – Si nécessaire, on peut se reporter aux Rec. UIT-T de la série X.200 et aux Normes ISO/CEI équivalentes pour obtenir la signification de ces termes. Ces termes ne sont applicables que si la notation ASN.1 est utilisée dans un environnement OSI.

3.6.51 notation de type ouvert: notation ASN.1 servant à désigner un ensemble de valeurs appartenant à plus d'un type ASN.1.

NOTE 1 – Les expressions "type ouvert" et "notation de type ouvert" sont synonymes dans le corps de la présente Recommandation | Norme internationale.

NOTE 2 – Les règles de codage d'ASN.1 assurent toutes le codage non ambigu des valeurs appartenant à un type ASN.1 unique, mais elles n'assurent pas nécessairement le codage non ambigu d'une "notation de type ouvert", qui véhicule des valeurs de types ASN.1 qui ne sont pas encore normalement déterminés au moment de la spécification. Le type de valeur codée dans la "notation de type ouvert" doit être connu avant de pouvoir déterminer de manière non ambiguë la valeur abstraite de ce champ.

NOTE 3 – Dans la présente Recommandation | Norme internationale, la seule notation correspondant à un type ouvert est le type "ObjectClassFieldType" (type de champ de classe d'objets), spécifié au § 14 de la Rec. UIT-T X.681 | ISO/CEI 8824-2, et dans laquelle le nom de champ "FieldName" désigne soit un champ de type, soit un champ de valeur de type variable.

3.6.52 type parent (d'un sous-type): type dont dérive un sous-type par imposition de contraintes, et qui gouverne la notation du sous-type.

NOTE – Le type parent peut lui-même être un sous-type d'un autre type.

3.6.53 production: partie de la notation formelle (également appelée grammaire ou formalisme de Backus-Naur, BNF) utilisée pour spécifier la notation ASN.1.

3.6.54 type réel (real type): type simple dont les valeurs distinctives (spécifiées au § 20) appartiennent à l'ensemble des nombres réels.

3.6.55 définition récursive (d'un type): ensemble de définitions ASN.1 qui ne peuvent pas être réordonnées et qui sont telles que tous les types utilisés dans une structure soient définis avant la définition de cette structure.

NOTE – Les définitions récursives sont autorisées en notation ASN.1: il appartient à l'utilisateur de s'assurer que les valeurs des types résultants ont une représentation finie et que l'ensemble de valeurs associé au type contient au moins une valeur.

3.6.56 identificateur d'objet relatif: valeur qui identifie un objet en fonction de sa position par rapport à un identificateur d'objet connu (voir § 3.6.47).

3.6.57 type identificateur d'objet relatif: type simple dont les valeurs sont tous les identificateurs d'objet relatifs possibles.

3.6.58 type chaîne de caractères à alphabet restreint (restricted character string type): type de chaîne de caractères dont les caractères sont choisis dans un répertoire de caractères donné identifié dans la spécification du type.

3.6.59 types sélection (selection types): types définis par référence à un type composant d'un type choix, et dont les valeurs sont précisément celles de ce type composant.

3.6.60 types séquence (sequence types): types définis par la désignation d'une liste fixe, ordonnée, de types (dont certains peuvent être déclarés optionnels); chaque valeur d'un type séquence ainsi défini est une liste ordonnée de valeurs, une par type composant.

NOTE – Une valeur du type séquence ne contiendra pas nécessairement de valeur pour un type composant si celui-ci est déclaré optionnel.

3.6.61 types séquence-de (sequence-of types): types définis par la désignation d'un seul type composant; chaque valeur d'un type séquence-de ainsi défini est une liste ordonnée comportant zéro, une ou plusieurs valeurs du type composant.

3.6.62 application (de contraintes) en série: application d'une contrainte à un type parent qui est déjà contraint.

3.6.63 opérations arithmétiques sur des ensembles: formation de nouveaux ensembles de valeurs ou de nouveaux objets informationnels en utilisant les opérations de réunion, d'intersection et de différence entre ensembles (utilisation de **`EXCEPT`**) comme spécifié au § 46.2.

NOTE – Le résultat de l'application de contraintes en série n'est pas couvert par l'expression "opérations arithmétiques sur des ensembles".

3.6.64 types ensemble (set types): types définis par la désignation d'une liste fixe, non ordonnée, de types (dont certains peuvent être déclarés optionnels); chaque valeur d'un type ensemble est une liste non ordonnée de valeurs, une par type composant.

NOTE – Une valeur du type ensemble ne contiendra pas nécessairement de valeur pour un type composant si celui-ci est déclaré optionnel.

3.6.65 types ensemble-de (set-of types): types définis par la désignation d'un seul type composant; chaque valeur d'un type ensemble-de est une liste non ordonnée comportant zéro, une ou plusieurs valeurs du type composant.

3.6.66 types simples: types définis en spécifiant directement l'ensemble de leurs valeurs.

3.6.67 caractère d'espacement: caractère d'un répertoire destiné à être inclus en impression avec une chaîne de caractères graphiques, mais qui est représenté matériellement par un vide; il n'est généralement pas considéré comme un caractère de contrôle (voir § 3.6.17).

NOTE – Un répertoire de caractères peut comporter un seul caractère d'espacement, ou plusieurs de différentes largeurs.

3.6.68 sous-type (d'un type parent): type dont les valeurs sont un sous-ensemble (ou l'ensemble complet) des valeurs d'un autre type (le type parent).

3.6.69 étiquette: dénomination de type associée à chaque type ASN.1.

3.6.70 types étiquetés (tagged types): types définis par la désignation d'un type existant et d'une étiquette; le type étiqueté ainsi formé et le type existant sont isomorphes mais distincts.

3.6.71 étiquetage: remplacement de l'étiquette existante (éventuellement l'étiquette par défaut) d'un type par une étiquette spécifiée.

3.6.72 syntaxe de transfert: ensemble de chaînes binaires utilisées pour échanger les valeurs abstraites d'une syntaxe abstraite, généralement obtenues par application de règles de codage à une syntaxe abstraite.

NOTE – L'expression "syntaxe de transfert " est synonyme de "codage".

3.6.73 Vrai (true): une des deux valeurs distinctives du type booléen (voir également "Faux").

3.6.74 type: ensemble nommé de valeurs.

3.6.75 nom de référence de type: nom associé de manière unique à un type dans un contexte donné.

NOTE – Des noms de référence sont affectés aux types définis dans la présente Recommandation | Norme internationale; ils sont disponibles universellement en notation ASN.1. D'autres noms de référence sont définis dans diverses Recommandations | Normes internationales et ne sont alors applicables que dans le contexte de celles-ci.

3.6.76 type chaîne de caractères à alphabet non restreint (unrestricted character string type): type dont les valeurs abstraites sont celles d'une syntaxe abstraite de caractères, conjointement avec une identification de la syntaxe abstraite de caractères et de la syntaxe de transfert de caractères à utiliser pour son codage.

3.6.77 utilisateur (de la notation ASN.1): personne physique ou morale qui définit la syntaxe abstraite d'une information particulière en notation ASN.1.

3.6.78 correspondance entre valeurs: relation biunivoque entre des valeurs de deux types, qui permet d'utiliser une référence à une valeur d'un type comme référence à une valeur de l'autre type. On peut l'utiliser, par exemple, pour spécifier des sous-types et des valeurs par défaut (voir Annexe B).

3.6.79 nom de référence de valeur: nom associé de manière unique à une valeur dans un contexte donné.

3.6.80 ensemble de valeurs: collection de valeurs d'un type donné; cet ensemble est sémantiquement équivalent à un sous-type.

3.6.81 crochets de version: paire de crochets gauches ou droits adjacents ("[[" ou "]]") utilisés pour délimiter le début ou la fin d'un groupe d'ajouts d'extension. La paire de crochets gauches peut facultativement être suivie par un nombre donnant un numéro de version pour le groupe d'ajouts d'extension.

3.6.82 numéro de version: numéro qui peut être associé à un crochet de version (voir § G.1.8).

NOTE – Un numéro de version ne peut être ajouté ni pour un ajout d'extension qui ne fait pas partie d'un groupe d'ajouts d'extension, ni pour des ajouts d'extension à n'importe quel type autre que choix, séquence ou ensemble.

3.6.83 espace blanc: toute action de formatage se traduisant par un espace sur la page d'impression, par exemple un espace ou une tabulation.

4 Abréviations

Pour les besoins de la présente Recommandation | Norme internationale, les abréviations suivantes sont utilisées:

ASN.1	Notation de syntaxe abstraite numéro un (*abstract syntax notation one*)
BER	Règles de codage de base d'ASN.1 (*basic encoding rules of ASN.1*)
BMP	Table multilingue (*basic multilingual plane*)
CEI	Commission électrotechnique internationale
DCC	Indicatif de pays pour les données (*data country code*)
DNIC	Code d'identification de réseau pour données (*data network identification code*)
ECN	Notation de contrôle de codage de l'ASN.1 (*encoding control notation of ASN.1*)
ER	Exploitation reconnue
ICD	Désignateur de code international (*international code designator*)
ISO	Organisation internationale de normalisation (*International organization for standardization*)
OID	Identificateur d'objet (*object identifier*)
OSI	Interconnexion des systèmes ouverts (*open systems interconnection*)
PER	Règles de codage compact de l'ASN.1 (*packed encoding rules of ASN.1*)
UCS	Jeu de caractères universels codés sur plusieurs octets *(universal multiple-octet coded character set*)
UTC	Temps universel coordonné (*coordinated universal time*)
UIT-T	Union internationale des télécommunications – Secteur de la normalisation des télécommunications
XML	Langage de balisage extensible (*extensible markup language*)

5 Notation

5.1 Généralités

5.1.1 Une notation ASN.1 consiste en une séquence de caractères pris dans le jeu de caractères ASN.1 spécifié au § 10.

5.1.2 Chaque instance de notation ASN.1 contient des caractères du jeu ASN.1 regroupés en unités lexicales. Le § 11 spécifie toutes les séquences de caractères formant des unités lexicales, ainsi que le nom de ces unités lexicales.

5.1.3 La notation ASN.1 est définie au § 12 (et les paragraphes suivants), par la spécification et le nommage des séquences d'unités lexicales qui forment des instances valides de la notation ASN.1 et par la spécification de la sémantique ASN.1 de chaque séquence.

5.1.4 Pour la spécification des séquences autorisées d'unités lexicales, la présente Recommandation | Norme internationale utilise une notation formelle définie dans les paragraphes suivants.

5.2 Productions

5.2.1 Toutes les unités lexicales sont nommées (voir § 11) et les séquences autorisées d'unités lexicales sont nommées.

5.2.2 Une nouvelle séquence autorisée d'unités lexicales (plus complexe) est définie au moyen d'une production. Celle-ci utilise les noms d'unités lexicales et de séquences autorisées d'unités lexicales pour former une nouvelle séquence autorisée d'unités lexicales nommée.

5.2.3 Chaque production comporte les parties suivantes, dans l'ordre, sur une ou plusieurs lignes:

a) le nom de la nouvelle séquence autorisée d'unités lexicales;

b) les caractères:

::=

c) une ou plusieurs séquences d'unités lexicales correspondant aux formes possibles, telles qu'elles sont définies au § 5.3, séparées par le caractère:

|

5.2.4 Une séquence d'unités lexicales figure dans la nouvelle séquence autorisée d'unités lexicales si elle figure dans une ou plusieurs des formes possibles. La nouvelle séquence autorisée d'unités lexicales est désignée dans la présente Recommandation | Norme internationale par le nom mentionné au § 5.2.3 a) ci-dessus.

NOTE – Si la même séquence d'unités lexicales apparaît dans plusieurs formes possibles, toute ambiguïté sémantique dans la notation résultante est résolue par le texte associé.

5.3 Formes possibles

5.3.1 Chaque forme possible d'une production (voir § 5.2.3 c)) est spécifiée par une liste de noms. Chaque nom est soit le nom d'une unité lexicale, soit le nom d'une séquence autorisée d'unités lexicales définie et nommée par une autre production.

5.3.2 La séquence autorisée d'unités lexicales définie par chaque forme possible est constituée de toutes les séquences obtenues en prenant l'une quelconque des séquences (ou l'unité lexicale) associée au premier nom, en combinaison avec (et suivie de) l'une quelconque des séquences (ou l'unité lexicale) associée au deuxième nom, en combinaison avec (et suivie de) l'une quelconque des séquences (ou l'unité lexicale) associée au troisième nom, etc., jusqu'au dernier nom inclus (ou jusqu'à la dernière unité lexicale incluse) de la forme possible.

5.4 Indicateur d'absence d'espacement

Si l'indicateur d'absence d'espacement "**&**" (ESPERLUETTE) est inséré entre deux unités lexicales dans des séquences de production, l'unité lexicale qui le précède et l'unité lexicale qui le suit ne doivent pas être séparées par un espace blanc.

NOTE – Cet indicateur est utilisé uniquement dans les productions qui décrivent la notation de valeur XML. Il est par exemple utilisé pour spécifier que l'unité lexicale "<" doit être suivie immédiatement d'un nom d'étiquette XML.

5.5 Exemple de production

5.5.1 La production:

```
ExampleProduction    ::=
            bstring
        |   hstring
        |   "{" IdentifierList "}"
```

associe le nom "ExampleProduction" aux séquences d'unités lexicales suivantes:

a) soit une chaîne binaire quelconque "bstring" (une unité lexicale);

b) soit une chaîne hexadécimale quelconque "hstring" (une unité lexicale);

c) soit une séquence quelconque d'unités lexicales associée à la liste d'identificateurs "IdentifierList", précédée du caractère "**{**" et suivie du caractère "**}**".

NOTE – "**{**" et "**}**" sont les noms des unités lexicales contenant respectivement le caractère unique **{** et le caractère unique **}** (voir § 11.26).

5.5.2 Dans cet exemple, "IdentifierList" serait défini par une autre production, placée avant ou après la production définissant "ExampleProduction".

5.6 Mise en page

Chaque production de la présente Recommandation | Norme internationale est précédée et suivie d'une ligne blanche. Il n'y a pas de ligne blanche à l'intérieur des productions. Les productions peuvent occuper une ou plusieurs lignes. La mise en page n'est pas significative.

5.7 Récursivité

Les productions de la présente Recommandation | Norme internationale sont souvent récursives. Dans ce cas, les productions doivent être appliquées autant de fois qu'il est nécessaire, jusqu'à ce qu'aucune nouvelle séquence ne soit engendrée.

NOTE – Dans de nombreux cas, cette procédure récursive produit un ensemble infini de séquences autorisées d'unités lexicales. Tout ou partie des séquences de l'ensemble peuvent elles-mêmes contenir un nombre non borné d'unités lexicales. Ceci n'est pas une erreur.

5.8 Désignation des séquences autorisées d'unités lexicales

La présente Recommandation | Norme internationale permet de désigner une séquence autorisée d'unités lexicales (faisant partie de la notation ASN.1) en citant le nom figurant à gauche du signe **::=** dans une production; le nom est mis entre GUILLEMETS (34) (") pour le distinguer du texte en langage naturel, sauf s'il apparaît à l'intérieur d'une production.

5.9 Désignation d'une unité lexicale

La présente Recommandation | Norme internationale permet de désigner une unité lexicale en utilisant son nom; le nom est mis entre GUILLEMETS (34) (") lorsqu'il apparaît dans du texte en langage naturel afin d'éviter tout risque de confusion avec ce texte.

5.10 Notations abrégées

Pour rendre les productions plus concises et en faciliter la lecture, les notations abrégées suivantes sont utilisées dans les définitions des séquences autorisées d'unités lexicales dans la présente Recommandation | Norme internationale ainsi que dans les Rec. UIT-T X.681 | ISO/CEI 8824-2, Rec. UIT-T X.682 | ISO/CEI 8824-3 et Rec. UIT-T X.683 | ISO/CEI 8824-4:

a) un astérisque (*) placé après deux noms "A" et "B" indique soit l'unité lexicale "empty" (vide) (voir § 11.7), soit l'une des séquences autorisées d'unités lexicales associées à "A", soit une série alternée comportant l'une des séquences d'unités lexicales associées à "A" et l'une des séquences d'unités lexicales associées à "B", commençant et se terminant par une séquence associée à "A". Ainsi:

   C ::= A B *

   équivaut à:

   C ::= D | empty
   D ::= A | A B D

   "D" étant une variable auxiliaire n'apparaissant pas ailleurs dans les productions.

   EXEMPLE – "C::= A B *" est une notation abrégée désignant l'une quelconque des formes suivantes pour C:

   empty
   A
   A B A
   A B A B A
   A B A B A B A
   ...

b) un signe plus (+) est analogue à l'astérisque en a), sauf que l'unité lexicale "empty" (vide) est exclue. Ainsi:

   E ::= A B +

   équivaut à:

   E ::= A | A B E

EXEMPLE – "E::= A B +" est une notation abrégée désignant l'une quelconque des formes suivantes pour E:

A
A B A
A B A B A
A B A B A B A
...

c) un point d'interrogation (?) placé à la suite d'un nom indique soit l'unité lexicale "empty" (vide) (voir § 11.7), soit une séquence autorisée d'unités lexicales associées à "A". Ainsi:

F ::= A ?

équivaut à:

F ::= empty | A

NOTE – Ces notations abrégées prévalent sur la juxtaposition d'unités lexicales dans les séquences de production (voir § 5.2.2).

5.11 Références de valeurs et typage de valeurs

5.11.1 La notation d'affectation de valeur ASN.1 permet de donner un nom à une valeur d'un type spécifié. Ce nom peut être utilisé chaque fois qu'il est nécessaire de faire référence à cette valeur. L'Annexe B décrit et spécifie le mécanisme de mappage entre valeurs qui permet à un nom de référence de valeur affecté à une valeur d'un type d'identifier une valeur d'un second type (analogue). Ainsi, on peut utiliser une référence à une valeur du premier type chaque fois qu'une référence à une valeur du second type est nécessaire.

5.11.2 Dans le corps des normes ASN.1, un texte définit la légalité (ou non) des constructions mettant en jeu plusieurs types. Ces spécifications de légalité nécessitent généralement la "compatibilité" entre deux types ou plus. Par exemple, le type utilisé pour définir une référence de valeur doit être "compatible avec" le type gouvernant lorsque cette référence de valeur est utilisée. L'Annexe B normative utilise le concept de mappage entre valeurs pour donner une indication précise sur la question de savoir si une construction ASN.1 donnée est légale ou non.

6 Modèle ASN.1 d'extension de type

Un décodeur peut détecter l'une des situations suivantes lorsqu'il décode un type extensible:

a) absence d'ajouts d'extension attendus dans un type séquence ou ensemble;

b) présence d'ajouts d'extension arbitraires non attendus en plus de ceux qui sont éventuellement définis dans un type séquence ou ensemble, présence d'une forme inconnue dans un type choix, présence d'une énumération inconnue dans un type énuméré ou présence d'une valeur ou d'une longueur non attendue pour un type dont la contrainte est extensible.

D'un point de vue formel, une syntaxe abstraite définie par le type extensible **`X`** contient non seulement les valeurs du type **`X`**, mais également les valeurs de tous les types qui sont apparentés à **`X`** par extension. Il s'ensuit que le processus de décodage ne signale jamais d'erreur lorsque l'une des situations précédentes (a ou b) est détectée. L'action à effectuer dans chacune de ces situations est déterminée par le spécificateur ASN.1.

NOTE – L'action consistera souvent à ignorer la présence d'extensions supplémentaires non attendues et à utiliser une valeur par défaut ou un indicateur "manquant" pour les ajouts d'extension attendus qui sont absents.

Des ajouts d'extension non attendus détectés par un décodeur dans un type extensible peuvent ensuite faire partie d'un codage ultérieur de ce type (à des fins de retransmission vers l'émetteur ou vers un tiers), à condition que la même syntaxe de transfert soit utilisée dans la transmission ultérieure.

7 Conditions imposées aux règles de codage par l'extensibilité

NOTE – Ces conditions s'appliquent aux règles de codage normalisées. Elles ne s'appliquent pas aux règles de codage définies au moyen de la notation ECN (voir la Rec. UIT-T X.692 | ISO/CEI 8825-3).

7.1 Toutes les règles de codage ASN.1 doivent permettre de coder les valeurs d'un type extensible **`X`** d'une manière telle qu'elles puissent être décodées au moyen d'un type extensible **`Y`** qui est apparenté au type **`X`** par extension. Les règles de codage doivent en outre permettre de coder à nouveau (au moyen du type **`Y`**) les valeurs qui avaient été décodées au moyen du type **`Y`** et de les décoder ensuite au moyen d'un troisième type extensible **`Z`** qui est apparenté au type **`Y`** par extension (et donc également au type **`X`**).

NOTE – Les types **`X`**, **`Y`** et **`Z`** peuvent apparaître dans un ordre quelconque dans la série d'extensions.

Si une valeur d'un type extensible **X** est codée puis relayée (directement ou par le biais d'une application relais utilisant un type **Z** apparenté par extension) vers une autre application qui décode la valeur au moyen d'un type extensible **Y** apparenté au type **X** par extension, le décodeur utilisant le type **y** obtiendra alors une valeur abstraite constituée des éléments suivants:

a) valeur abstraite du type correspondant à la racine d'extension;

b) valeurs abstraites de chacun des ajouts d'extension qui figurent à la fois dans les types **X** et **Y**;

c) codage délimité pour tout ajout d'extension éventuel figurant dans le type **X** et non dans le type **Y**.

Le codage dans le cas c) doit être en mesure d'être inclus dans un codage ultérieur d'une valeur de type **Y**, si l'application le nécessite. Ce codage doit être un codage valide pour une valeur de type **X**.

Exemple didactique: si le système A utilise un type racine extensible (type **X**) qui est un type séquence ou ensemble avec un ajout d'extension d'un type entier optionnel, alors que le système B utilise un type apparenté par extension (type **Y**) qui possède deux ajouts d'extension qui sont chacun de type entier optionnel, le système A ne devra pas confondre une transmission faite par B pour une valeur de **Y** qui omet le premier ajout d'extension et qui contient le deuxième avec une transmission où figure (seulement) le premier ajout d'extension de **X** dont il connaît l'existence. Le système A doit en outre être en mesure de coder à nouveau la valeur de **X** avec une valeur figurant dans le premier type entier, suivie de la deuxième valeur d'entier reçue de B, si le protocole d'application le nécessite.

7.2 Toutes les règles de codage ASN.1 doivent spécifier le codage et le décodage de la valeur d'un type énuméré et d'un type choix d'une manière telle que, si une valeur transmise se trouve dans l'ensemble d'ajouts d'extension communs au codeur et au décodeur, cette valeur soit décodée avec succès; autrement, le décodeur doit avoir la possibilité d'en délimiter le codage et de l'identifier comme étant une valeur correspondant à un ajout d'extension (non connu).

7.3 Toutes les règles de codage ASN.1 doivent spécifier le codage et le décodage de types avec des contraintes extensibles d'une manière telle que, si une valeur transmise appartient à l'ensemble d'ajouts d'extension communs au codeur et au décodeur, cette valeur soit décodée avec succès; autrement, le décodeur doit avoir la possibilité d'en délimiter le codage et de l'identifier comme étant une valeur correspondant à un ajout d'extension (non connu).

Dans tous les cas, la présence d'ajouts d'extension n'affectera pas la capacité de reconnaissance ultérieure de l'information lorsqu'un type avec un marqueur d'extension est imbriqué dans un autre type.

NOTE 1 – Toutes les variantes des règles de codage de base et des règles de codage compact de l'ASN.1 satisfont à toutes ces prescriptions. Les règles de codage définies au moyen de la notation ECN peuvent ou non satisfaire à toutes ces prescriptions.

NOTE 2 – Les règles PER et BER n'identifient pas le numéro de version dans le codage d'un ajout d'extension. Les codages spécifiés au moyen de la notation ECN peuvent ou non fournir cette identification.

8 Etiquettes

8.1 On spécifie une étiquette en indiquant une classe et un numéro dans cette classe. L'étiquette peut appartenir à l'une des classes suivantes:

– universelle;

– application;

– privée;

– propre au contexte.

8.2 Le numéro est un entier non négatif, spécifié en notation décimale.

8.3 Le paragraphe 30 spécifie les restrictions applicables aux étiquettes affectées par l'utilisateur de la notation ASN.1.

NOTE – Le paragraphe 30 comprend la restriction suivante: les utilisateurs de cette notation ne sont pas autorisés à spécifier explicitement des étiquettes de la classe universelle dans leurs spécifications ASN.1. D'un point de vue formel, l'utilisation des étiquettes des trois autres classes est identique pour les trois classes. Lorsqu'une étiquettes de la classe application est employée, il est généralement possible d'employer à la place une étiquette de la classe privée ou de la classe propre au contexte, au choix de l'utilisateur. La présence des trois classes est en grande partie due à des raisons historiques, mais le § E.2.12 fournit des indications sur la manière dont les classes sont généralement employées.

8.4 Le Tableau 1 récapitule les étiquettes de la classe universelle affectées dans la présente Recommandation | Norme internationale.

Tableau 1 – Affectation des étiquettes de la classe universelle

UNIVERSAL 0	Utilisation réservée aux règles de codage
UNIVERSAL 1	Type booléen
UNIVERSAL 2	Type entier
UNIVERSAL 3	Type chaîne binaire
UNIVERSAL 4	Type chaîne d'octets
UNIVERSAL 5	Type néant
UNIVERSAL 6	Type identificateur d'objet
UNIVERSAL 7	Type descripteur d'objet
UNIVERSAL 8	Type externe et type instance-de
UNIVERSAL 9	Type réel
UNIVERSAL 10	Type énuméré
UNIVERSAL 11	Type valeur pdv imbriquée
UNIVERSAL 12	Type UFT8String
UNIVERSAL 13	Type identificateur d'objet relatif
UNIVERSAL 14-15	Réservées pour de futures éditions de la présente Recommandation | Norme internationale
UNIVERSAL 16	Types séquence et séquence-de
UNIVERSAL 17	Types ensemble et ensemble-de
UNIVERSAL 18-22, 25-30	Types chaîne de caractères
UNIVERSAL 23-24	Types temps
UNIVERSAL 31-…	Réservées aux additifs à la présente Recommandation | Norme internationale

8.5 Certaines règles de codage nécessitent un ordre canonique des étiquettes. Dans un souci d'uniformité, un ordre canonique des étiquettes est défini au § 8.6.

8.6 L'ordre canonique des étiquettes, fondé sur l'étiquette la plus à l'extérieur de chaque type, est défini de la manière suivante:

a) les éléments ou formes avec des étiquettes de la classe universelle apparaîtront en premier, suivis de ceux avec des étiquettes de la classe application, suivis de ceux avec des étiquettes propres au contexte, suivis de ceux avec des étiquettes de la classe privée;

b) à l'intérieur de chaque classe d'étiquettes, les éléments ou formes apparaîtront dans l'ordre croissant de leur numéro d'étiquette.

9 Utilisation de la notation ASN.1

9.1 Un type est défini en ASN.1 par la notation "Type" (voir § 16.1).

9.2 Une valeur d'un type donné est déclarée en ASN.1 par la notation "Value" (voir § 16.7).

NOTE – Il est généralement impossible d'interpréter la notation d'une valeur sans en connaître le type.

9.3 Un type est affecté à un nom de référence de type en ASN.1 par la notation "TypeAssignment" (voir § 15.1), "ValueSetTypeAssignment" (voir § 15.6), "ParameterizedTypeAssignment" (voir § 8.2 de la Rec. UIT-T X.683 | ISO/CEI 8824-4) ou "ParameterizedValueSetTypeAssignment" (voir § 8.2 de la Rec. UIT-T X.683 | ISO/CEI 8824-4).

9.4 Une valeur est affectée à un nom de référence de valeur en ASN.1 par la notation "ValueAssignment" (voir § 15.2) ou "ParameterizedValueAssignment" (voir § 8.2 de la Rec. UIT-T X.683 | ISO/CEI 8824-4).

9.5 Les différentes productions de notation d'affectation "Assignment" ne seront utilisées que dans la notation "ModuleDefinition" (définition de module) (sauf dans les cas spécifiés dans la NOTE 2 du 12.1).

10 Jeu de caractères ASN.1

10.1 Une unité lexicale consiste en une séquence de caractères pris dans le Tableau 2, sauf dans les cas spécifiés aux § 10.2 et 10.3. Dans le Tableau 2, les caractères sont identifiés par les noms qui leur sont attribués dans l'ISO/CEI 10646-1.

Tableau 2 – Caractères ASN.1

A à Z	(ALPHABET LATIN: LETTRE MAJUSCULE A à LETTRE MAJUSCULE Z)
a à z	(ALPHABET LATIN: LETTRE MINUSCULE a à LETTRE MINUSCULE z)
0 à 9	(CHIFFRE ZÉRO à CHIFFRE 9)
!	(POINT D'EXCLAMATION)
"	(GUILLEMET)
&	(ESPERLUETTE)
'	(APOSTROPHE)
(	(PARENTHÈSE GAUCHE)
)	(PARENTHÈSE DROITE)
*****	(ASTÉRISQUE)
,	(VIRGULE)
-	(TRAIT D'UNION – SIGNE MOINS)
.	(POINT)
/	(BARRE OBLIQUE)
:	(DEUX-POINTS)
;	(POINT-VIRGULE)
<	(SIGNE INFÉRIEUR À)
=	(SIGNE ÉGAL)
>	(SIGNE SUPÉRIEUR À)
@	(AROBAS)
[	(CROCHET GAUCHE)
]	(CROCHET DROIT)
^	(ACCENT CIRCONFLEXE)
_	(TIRET SOULIGNÉ)
{	(ACCOLADE GAUCHE)
|	(BARRE VERTICALE)
}	(ACCOLADE DROITE)

NOTE – Quand des normes dérivées équivalentes sont élaborées par des organismes de normalisation nationaux, des caractères supplémentaires peuvent apparaître dans les unités lexicales suivantes:

- typereference (référence de type) (voir § 11.2);
- identifier (identificateur) (voir § 11.3);
- valuereference (référence de valeur) (voir § 11.4);
- modulereference (référence de module) (voir § 11.5).

Quand des caractères supplémentaires sont introduits pour pouvoir utiliser une langue dans laquelle il n'y a pas de distinction entre majuscules et minuscules, la distinction syntaxique assurée en imposant au premier caractère de certaines unités lexicales d'être une majuscule ou une minuscule sera obtenue d'une autre façon. Cette disposition est introduite pour permettre d'écrire des spécifications ASN.1 valides dans différentes langues.

10.2 Lorsque la notation est utilisée pour spécifier la valeur d'un type chaîne de caractères, tous les caractères du jeu défini peuvent apparaître entre GUILLEMETS (34) (") dans la notation ASN.1 (voir § 11.14).

10.3 Des symboles graphiques supplémentaires arbitraires peuvent apparaître dans l'unité lexicale "comment" (commentaire) (voir § 11.6).

10.4 Aucune signification particulière ne sera accordée au style, à la taille, à la couleur, à la graisse ou aux autres caractéristiques typographiques d'affichage.

10.5 Les majuscules et les minuscules seront considérées comme distinctes.

10.6 Les définitions ASN.1 peuvent aussi contenir des caractères espace blanc (voir 11.1.6) entre les unités lexicales.

11 Unités lexicales ASN.1

11.1 Règles générales

11.1.1 Les paragraphes qui suivent spécifient les caractères utilisés dans les unités lexicales. Dans chaque cas, le nom de l'unité lexicale est donné avec la définition des séquences de caractères qui la forment.

11.1.2 Les unités lexicales spécifiées dans les sous-paragraphes du présent § 11 (exception faite du commentaire "comment" de plusieurs lignes et des chaînes "bstring", "hstring" et "cstring") ne contiendront pas d'espace blanc (voir les § 11.6, 11.10, 11.12 et 11.14).

11.1.3 La longueur d'une ligne n'est pas limitée.

11.1.4 Les unités lexicales peuvent être séparées par un ou plusieurs espaces blancs (voir le § 11.1.6) ou commentaires (voir le § 11.6) sauf lorsque l'indicateur d'absence d'espacement "**&**" (voir le § 5.4) est utilisé. Dans une production "XMLTypedValue" (voir le § 15.2), un espace blanc peut figurer entre deux unités lexicales, mais l'unité lexicale "comment" ne doit pas être présente.

NOTE – Il s'agit d'éviter toute ambiguïté résultant de la présence d'un double trait d'union ou d'un astérisque et d'une barre oblique adjacents dans une unité lexicale "xmlcstring". Ces caractères n'indiquent jamais le début d'une unité lexicale "comment" lorsqu'ils figurent dans une production "XMLTypedValue".

11.1.5 Si le ou les caractères initiaux d'une unité lexicale font partie des caractères qu'il est permis d'ajouter à la suite des caractères de l'unité lexicale précédente, ces deux unités lexicales devront être séparées par une ou plusieurs instances d'espace blanc ou de commentaire.

11.1.6 La présente Recommandation | Norme internationale utilise les termes "nouvelle ligne" et "espace blanc". Pour représenter un espace blanc ou une nouvelle ligne (fin de ligne) dans des spécifications lisibles par une machine, on peut utiliser l'un quelconque des caractères suivants ou une combinaison quelconque de ces caractères (pour chaque caractère, on donne son nom et son code spécifiés dans la Norme Unicode):

Pour un espace blanc:

TABULATION HORIZONTALE (9)

LIGNE SUIVANTE (10)

TABULATION VERTICALE (11)

PAGE SUIVANTE (12)

RETOUR CHARIOT (13)

ESPACE (32)

Pour une nouvelle ligne:

LIGNE SUIVANTE (10)

TABULATION VERTICALE (11)

PAGE SUIVANTE (12)

RETOUR CHARIOT (13)

NOTE – Tout caractère ou toute séquence de caractères qui constitue une nouvelle ligne valide constitue également un espace blanc valide.

11.2 Référence de type

Nom de l'unité lexicale – typereference

11.2.1 Une référence "typereference" est constituée d'un nombre arbitraire (un ou plusieurs) de lettres, chiffres et traits d'union. Le premier caractère doit être une majuscule. Le dernier caractère ne peut pas être un trait d'union. Un trait d'union ne doit pas être immédiatement suivi par un autre trait d'union.

NOTE – Les règles concernant le trait d'union sont destinées à éviter toute confusion possible avec un commentaire (éventuellement placé à la suite).

11.2.2 Une référence "typereference" ne doit pas être l'une des séquences de caractères réservées indiquées au § 11.27.

11.3 Identificateur

Nom de l'unité lexicale – identifier

Un identificateur "identifier" est constitué d'un nombre arbitraire (un ou plusieurs) de lettres, chiffres et traits d'union. Le premier caractère doit être une minuscule. Le dernier caractère ne peut pas être un trait d'union. Un trait d'union ne peut pas être immédiatement suivi par un autre trait d'union.

NOTE – Les règles concernant le trait d'union sont destinées à éviter toute confusion possible avec un commentaire (éventuellement placé à la suite).

11.4 Référence de valeur

Nom de l'unité lexicale – valuereference

Une référence "valuereference" est constituée de la séquence de caractères spécifiée pour un identificateur au § 11.3. Lors de l'analyse d'une instance de cette notation, une référence "valuereference" est distinguée d'un identificateur "identifier" par le contexte dans lequel elle apparaît.

11.5 Référence de module

Nom de l'unité lexicale – modulereference

Une référence "modulereference" est constituée de la séquence de caractères spécifiée pour une référence "typereference" au § 11.2. Lors de l'analyse d'une instance de cette notation, une référence "modulereference" est distinguée d'une référence "typereference" par le contexte dans lequel elle apparaît.

11.6 Commentaire

Nom de l'unité lexicale – comment

11.6.1 Un commentaire "comment" n'a pas de référence dans la définition d'une notation ASN.1. Il peut toutefois apparaître n'importe où entre d'autres unités lexicales, et n'a pas de signification syntaxique.

NOTE – Dans le contexte d'une Recommandation | Norme internationale comportant des déclarations ASN.1, un commentaire ASN.1 peut néanmoins contenir un texte ayant force de norme se rapportant à la sémantique de l'application ou à des contraintes syntaxiques.

11.6.2 L'unité lexicale "comment" peut avoir deux formes:

- commentaire d'une ligne commençant par "`--`" comme défini au § 11.6.3;
- commentaire de plusieurs lignes commençant par "`/*`" comme défini au § 11.6.4.

11.6.3 Chaque fois qu'un commentaire "comment" commence par un double trait d'union, il se termine au premier double trait d'union rencontré ou en fin de ligne. Un commentaire ne doit pas contenir de doubles traits d'union autres que ceux par lesquels il commence et éventuellement se termine. Si un commentaire commençant par les caractères "`--`" inclut les caractères adjacents "`/*`" ou "`*/`", ces caractères n'ont pas de signification spéciale et sont considérés comme faisant partie du commentaire. Le commentaire peut inclure des symboles graphiques qui n'appartiennent pas au jeu de caractères spécifié au § 10.1 (voir § 10.3).

11.6.4 Chaque fois qu'un commentaire "comment" commence par "`/*`", il se termine par les caractères correspondants "`*/`", que ceux-ci soient sur la même ligne ou non. S'il y a d'autres caractères "`/*`" avant des caractères "`*/`", le commentaire se termine lorsque des caractères "`*/`" sont associés aux différents caractères "`/*`". Si un commentaire commençant par des caractères "`/*`" contient un double trait d'union "`--`", ces traits d'union n'ont pas de signification spéciale et sont considérés comme faisant partie du commentaire. Le commentaire peut inclure des symboles graphiques qui n'appartiennent pas au jeu de caractères spécifié au § 10.1 (voir § 10.3).

NOTE – Cela permet à l'utilisateur de faire des commentaires sur des parties d'un module ASN.1 qui contiennent déjà des commentaires (commençant par "`--`" ou par "`/*`") en insérant simplement les caractères "`/*`" au début de la partie devant faire l'objet d'un commentaire et les caractères "`*/`" à la fin, sous réserve que la partie devant faire l'objet d'un commentaire ne contienne pas de valeurs de chaîne de caractères contenant "`/*`" ou "`*/`".

11.7 Unité lexicale vide

Nom de l'unité lexicale – empty

L'unité lexicale vide "empty" ne contient aucun caractère. Elle est utilisée dans la notation du § 5 pour inclure l'ensemble vide dans les formes possibles lorsque plusieurs ensembles de séquences de productions sont spécifiés.

11.8 Numéro

Nom de l'unité lexicale – number

Un numéro "number" comprend un ou plusieurs chiffres. Le premier chiffre doit être différent de zéro, sauf si le numéro n'a qu'un seul chiffre.

NOTE – L'unité lexicale "number" est toujours interprétée comme un entier en base dix.

11.9 Nombre réel

Nom de l'unité lexicale – realnumber

Un nombre réel "realnumber" se compose d'une partie entière constituée d'un ou de plusieurs chiffres avec, facultativement, une virgule décimale (,). La virgule décimale peut éventuellement être suivie d'une partie décimale constituée d'un ou de plusieurs chiffres. La partie entière, la virgule décimale ou la partie décimale (c'est-à-dire celui de ces trois éléments qui est présent en dernier) peut éventuellement être suivi d'un e ou d'un E ainsi que d'un exposant, le cas échéant signé, constitué d'un ou de plusieurs chiffres. Le premier chiffre de l'exposant doit être différent de zéro sauf s'il est composé d'un seul chiffre.

11.10 Chaîne binaire

Nom de l'unité lexicale – bstring

Une chaîne binaire "bstring" se compose d'un nombre arbitraire (éventuellement zéro) de caractères parmi les caractères suivants:

0 1

éventuellement mélangés avec des espaces blancs, précédés d'un APOSTROPHE (39) (**'**) et suivis des deux caractères:

'B

EXEMPLE – **'01101100'B**

Les espaces blancs qui figurent à l'intérieur d'une unité lexicale chaîne binaire n'ont pas de signification.

11.11 Unité lexicale chaîne binaire XML

Nom de l'unité lexicale – xmlbstring

Une chaîne "xmlbstring" est composée d'un nombre arbitraire (éventuellement zéro) de 0, 1 et espaces blancs. Les éventuels caractères espace blanc qui figurent à l'intérieur d'une unité lexicale chaîne binaire n'ont pas de signification.

EXEMPLE – 01101100

Cette séquence de caractères est également une instance valable des chaînes "xmlhstring" et "xmlcstring". Lors de l'analyse d'une instance de cette notation, une chaîne "xmlbstring" est distinguée d'une chaîne "xmlhstring" ou "xmlcstring" par le contexte dans lequel elle figure.

11.12 Chaîne hexadécimale

Nom de l'unité lexicale – hstring

11.12.1 Une chaîne "hstring" est composée d'un nombre arbitraire (éventuellement zéro) de caractères parmi les caractères suivants:

A B C D E F 0 1 2 3 4 5 6 7 8 9

éventuellement mélangés avec des espaces blancs, précédés d'un APOSTROPHE (39) (**'**) et suivis des deux caractères:

'H

EXEMPLE – **'AB0196'H**

Les espaces blancs qui figurent à l'intérieur d'une unité lexicale chaîne hexadécimale n'ont pas de signification.

11.12.2 Chaque caractère représente la valeur d'un demi-octet en hexadécimal.

11.13 Unité lexicale chaîne hexadécimale XML

Nom de l'unité lexicale – xmlhstring

11.13.1 Une chaîne "xmlhstring" est composée d'un nombre arbitraire (éventuellement zéro) de caractères parmi les caractères suivants:

0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f

et l'espace blanc. Les éventuels caractères espace blanc qui figurent à l'intérieur d'une unité lexicale chaîne hexadécimale n'ont pas de signification.

EXEMPLE – Ab0196

11.13.2 Chaque caractère représente la valeur d'un demi-octet en hexadécimal.

11.13.3 Certaines instances de "xmlhstring" sont également des instances valables de "xmlbstring" et "xmlcstring". Lors de l'analyse d'une instance de cette notation, une chaîne "xmlhstring" est distinguée d'une chaîne "xmlbstring" ou "xmlcstring" par le contexte dans lequel elle figure.

11.14 Chaîne de caractères

Nom de l'unité lexicale – cstring

11.14.1 Une chaîne de caractères "cstring" est composée d'un nombre arbitraire (éventuellement zéro) de symboles graphiques et caractères d'espacement du jeu désigné par le type de la chaîne de caractères, placés entre GUILLEMETS (34) ("). Si le jeu de caractères comprend le GUILLEMET (34), celui-ci (s'il est présent dans la chaîne de caractères représentée par la chaîne "cstring") sera représenté dans la chaîne "cstring" par deux GUILLEMETS (34) consécutifs sur la même ligne et sans caractère d'espacement intermédiaire. La chaîne "cstring" peut occuper plusieurs lignes de texte, auquel cas la chaîne de caractères représentée ne comportera pas de caractère d'espacement dans la position située juste avant ou après les fins de ligne de la chaîne "cstring". Les éventuels caractères d'espacement situés immédiatement avant ou après une fin de ligne d'une chaîne "cstring" n'ont pas de signification.

NOTE 1 – Une chaîne "cstring" ne peut être utilisée que pour représenter sans ambiguïté (sur une page imprimée) des chaînes dont chaque caractère est soit un symbole graphique soit un caractère d'espacement. Il existe d'autres syntaxes ASN.1 pour représenter sur une page imprimée des chaînes comportant des caractères de contrôle (voir § 35).

NOTE 2 – Une chaîne "cstring" est constituée des caractères associés aux symboles graphiques et aux caractères d'espacement. Les caractères d'espacement placés immédiatement avant ou après une fin de ligne de la chaîne "cstring" ne font pas partie de la chaîne représentée (ils sont ignorés). Lorsque la chaîne "cstring" comporte des caractères d'espacement ou lorsque les symboles graphiques du répertoire de caractères comportent des ambiguïtés dans une représentation imprimée, la chaîne "cstring" pourra être ambiguë dans cette représentation.

EXEMPLE 1 – **"屎 屍 市 弒"**

EXEMPLE 2 – La chaîne "cstring"

```
"ABCDE   FGH
IJK""XYZ"
```

peut être utilisée pour représenter une valeur de chaîne de caractères du type **IA5String.** La valeur représentée est constituée des caractères:

```
ABCDE   FGHIJK"XYZ
```

où le nombre exact d'espaces voulus entre **E** et **F** peut être ambigu dans une représentation imprimée si une police de caractères à espacement proportionnel est utilisée dans la spécification imprimée (ce qui est le cas dans l'exemple) ou si le répertoire de caractères contient plusieurs caractères d'espacement de différentes largeurs.

11.14.2 Un caractère de combinaison (voir l'Annexe F) apparaîtra dans une représentation imprimée de la chaîne "cstring" comme un caractère à part entière et ne recevra pas en surimpression le caractère avec lequel il se combine. (Ceci garantit que l'ordre des caractères de combinaison dans la valeur de la chaîne est défini sans ambiguïté dans la version imprimée.)

EXEMPLE – La minuscule "**é**" et l'accent aigu de combinaison sont deux caractères de l'ISO/CEI 10646-1, et il convient donc d'imprimer la chaîne "cstring" correspondante sous la forme de deux caractères et non sous la forme du caractère unique **é**.

11.15 Unité lexicale chaîne de caractères XML

Nom de l'unité lexicale – xmlcstring

11.15.1 Une chaîne "xmlcstring" est constituée d'un nombre arbitraire (éventuellement zéro) de caractères parmi les caractères suivants de l'ISO/CEI 10646-1:

a) TABULATION HORIZONTALE (9);

b) LIGNE SUIVANTE (10);

c) RETOUR CHARIOT (13);

d) tout caractère dont le code ISO/CEI 10646-1 est compris entre 32 (20 hex) et 55295 (D7FF hex) inclus;

e) tout caractère dont le code ISO/CEI 10646-1 est compris entre 57344 (E000 hex) et 65533 (FFFD hex) inclus;

f) tout caractère dont le code ISO/CEI 10646-1 est compris entre 65536 (10000 hex) et 1114111 (10FFFF hex) inclus.

NOTE – Des restrictions supplémentaires sont imposées par le fait que toute instance de la chaîne "xmlcstring" ne peut contenir que des caractères autorisés par le type de chaîne de caractères gouvernant.

11.15.2 Les caractères "&" (ESPERLUETTE), "<" (SIGNE INFERIEUR A) et ">" (SIGNE SUPERIEUR A) ne doivent apparaître que dans les séquences de caractères spécifiées aux § 11.15.4 et 11.15.5.

11.15.3 Une chaîne "xmlcstring" sert à représenter la valeur d'une chaine de caractères à alphabet restreint (voir § 37.9) et peut servir à représenter toutes les combinations de caractères ISO/CEI 10646-1, soit directement, soit en utilisant les séquences d'échappement spécifiées ci-dessous.

NOTE 1 – Une chaîne "xmlcstring" ne peut pas être utilisée pour représenter des caractères qui ne figurent pas dans l'ISO/CEI 10646-1 (par exemple certains des caractères de contrôle qui peuvent figurer dans une chaîne **`GeneralString`**) ni pour représenter des caractères qui pourraient être définis avec un code de caractère ISO/CEI 10646-1 supérieur à 10FFFF hex.

NOTE 2 – Aucune distinction n'est faite entre les caractères LIGNE SUIVANTE (10) et RETOUR CHARIOT (13) et la paire RETOUR CHARIOT + LIGNE SUIVANTE lorsqu'ils sont traités par des processeurs XML conformes.

11.15.4 Si des caractères "&" (ESPERLUETTE), "<" (SIGNE INFERIEUR A) ou ">" (SIGNE SUPERIEUR A) sont présents dans une valeur abstraite de chaîne de caractères "xmlcstring" (voir § 37.9), ils doivent être représentés dans la chaîne "xmlcstring":

a) soit par les séquences d'échappement spécifiées au § 11.15.8;

b) soit par les séquences d'échappement "&", "<" ou ">" respectivement. Ces séquences d'échappement ne doivent pas contenir d'espace blanc (voir § 11.1.6).

11.15.5 Si un caractère dont le code ISO/CEI 10646-1 figure dans la colonne 1 du Tableau 3 est présent dans une valeur abstraite de chaîne de caractères "xmlcstring" (voir § 37.9), il devra être représenté par la séquence de caractères de la colonne 2 du Tableau 3. Ces séquences de caractères ne doivent pas contenir d'espace blanc (voir § 11.1.6).

NOTE – Ce tableau ne comprend pas les caractères dont le code décimal vaut 9, 10 ou 13, et toutes les lettres présentes dans ces séquences de caractères sont des minuscules.

Tableau 3 – Séquences d'échappement pour les caractères de contrôle dans une chaîne "xmlcstring"

Code de caractère ISO/CEI 10646-1	Représentation "xmlcstring"	Code de caractère ISO/CEI 10646-1	Représentation "xmlcstring"
0 (0 hex)	<nul/>	17 (11 hex)	<dc1/>
1 (1 hex)	<soh/>	18 (12 hex)	<dc2/>
2 (2 hex)	<stx/>	19 (13 hex)	<dc3/>
3 (3 hex)	<etx/>	20 (14 hex)	<dc4/>
4 (4 hex)	<eot/>	21 (15 hex)	<nak/>
5 (5 hex)	<enq/>	22 (16 hex)	<syn/>
6 (6 hex)	<ack/>	23 (17 hex)	<etb/>
7 (7 hex)	<bel/>	24 (18 hex)	<can/>
8 (8 hex)	<bs/>	25 (19 hex)	
11 (B hex)	<vt/>	26 (1A hex)	<sub/>
12 (C hex)	<ff/>	27 (1B hex)	<esc/>
14 (E hex)	<so/>	28 (1C hex)	<is4/>
15 (F hex)	<si/>	29 (1D hex)	<is3/>
16 (10 hex)	<dle/>	30 (1E hex)	<is2/>
		31 (1F hex)	<is1/>

11.15.6 Lorsqu'une chaîne "xmlcstring" est utilisée dans une valeur "XMLTypedValue" (voir § 15.2) faisant partie d'un codage XER (voir la Rec. UIT-T X.693 | ISO/CEI 8825-4), elle peut contenir des TRAITS D'UNION – SIGNES MOINS (45) adjacents. Lorsqu'elle est utilisée dans une instance de notation de valeur XML dans un module ASN.1,

elle ne doit pas contenir deux TRAITS D'UNION – SIGNES MOINS adjacents. Si cette séquence de caractères est présente dans une valeur abstraite de chaîne de caractères "xmlcstring" dans un module ASN.1, au moins un des TRAITS D'UNION – SIGNES MOINS adjacents doit être représenté par les séquences d'échappement spécifiées au § 11.15.8.

11.15.7 Lorsqu'une chaîne "xmlcstring" est utilisée dans une valeur "XMLTypedValue" faisant partie d'un codage XER (voir la Rec. UIT-T X.693 | ISO/CEI 8825-4), elle peut contenir un ASTERISQUE (42) et une BARRE OBLIQUE (47) adjacents dans n'importe quel ordre. Lorsqu'elle est utilisée dans une instance de notation de valeur XML dans un module ASN.1, elle ne doit pas contenir d'ASTERISQUE et de BARRE OBLIQUE adjacents (quel que soit l'ordre). Si cette séquence de caractères est présente dans une valeur abstraite de chaîne de caractères "xmlcstring", au moins un des caractères ASTERISQUE et BARRE OBLIQUE adjacents doit être représenté par les séquences d'échappement spécifiées au § 11.15.8.

11.15.8 Tout caractère figurant directement dans une chaîne "xmlcstring" peut aussi être représenté dans la chaîne "xmlcstring" par une séquence d'échappement de la forme "&#n;" (où n est le code de caractère ISO/CEI 10646-1 en notation décimale) ou de la forme "&#xn;" (où n est le code de caractère ISO/CEI 10646-1 en notation hexadécimale). Ces séquences d'échappement ne doivent pas contenir d'espace blanc (voir § 11.1.6).

NOTE 1 – Les zéros d'en-tête sont autorisés dans les valeurs décimale et hexadécimale de "n" et les lettres "A"-"F" peuvent être utilisées aussi bien en minuscules qu'en majuscules dans la valeur hexadécimale.

NOTE 2 – Si les séquences d'échappement "&#n" et "&#xn" sont utilisées pour des caractères ISO/CEI 10646-1 qui ne figurent pas dans la table multilingue (BMP, *basic multilingual plane*), la valeur de "n" sera supérieure à 65535 (FFFF hex).

EXEMPLE – La chaîne "xmlcstring":

ABCDé FGHîJK&XYZ

peut être utilisée pour représenter une valeur de chaîne de caractères de type UTF8String. La valeur représentée comprend les caractères:

ABCDé FGHîJK&XYZ

où le nombre exact de caractères espace entre é et F peut être ambigu sur un support imprimé si une police à espacement proportionnel est utilisée dans la spécification, comme c'est le cas ci-dessus.

11.16 Unité lexicale affectation

Nom de l'unité lexicale – "**::=**"

Cette unité lexicale est composée de la séquence de caractères:

::=

NOTE – Cette séquence ne doit pas contenir d'espace blanc (voir § 11.1.2).

11.17 Séparateur d'intervalles de valeurs

Nom de l'unité lexicale – "**..**"

Cette unité lexicale se compose de la séquence de caractères:

..

NOTE – Cette séquence ne doit pas contenir d'espace blanc (voir § 11.1.2).

11.18 Points de suspension

Nom de l'unité lexicale – "**...**"

Cette unité lexicale se compose de la séquence de caractères:

...

NOTE – Cette séquence ne doit pas contenir d'espace blanc (voir § 11.1.2).

11.19 Crochets gauches de version

Nom de l'unité lexicale – "**[[**"

Cette unité lexicale se compose de la séquence de caractères:

[[

NOTE – Cette séquence ne doit pas contenir d'espace blanc (voir § 11.1.2).

11.20 Crochets droits de version

Nom de l'unité lexicale – "]]"

Cette unité lexicale se compose de la séquence de caractères:

]]

NOTE – Cette séquence ne doit pas contenir d'espace blanc (voir § 11.1.2).

11.21 Unité lexicale début d'étiquette unique XML

Nom de l'unité lexicale – "</"

Cette unité lexicale se compose de la séquence de caractères:

</

NOTE – Cette séquence ne doit pas contenir d'espace blanc (voir § 11.1.2).

11.22 Unité lexicale fin d'étiquette unique XML

Nom de l'unité lexicale – "/>"

Cette unité lexicale se compose de la séquence de caractères:

/>

NOTE – Cette séquence ne doit pas contenir d'espace blanc (voir § 11.1.2).

11.23 Unité lexicale vrai booléen XML

Nom de l'unité lexicale – "true"

11.23.1 Cette unité lexicale se compose de la séquence de caractères:

true

11.23.2 Lors de l'analyse d'une instance de cette notation, une valeur "true" est distinguée d'une référence "valuereference" ou d'un identificateur "identifier" par le contexte dans lequel elle apparaît.

NOTE – Cette séquence ne doit pas contenir d'espace blanc (voir § 11.1.2).

11.24 Unité lexicale faux booléen XML

Nom de l'unité lexicale – "false"

11.24.1 Cette unité lexicale se compose de la séquence de caractères:

false

11.24.2 Lors de l'analyse d'une instance de cette notation, une valeur "false" est distinguée d'une référence "valuereference" ou d'un identificateur "identifier" par le contexte dans lequel elle apparaît.

NOTE – Cette séquence ne doit pas contenir d'espace blanc (voir § 11.1.2).

11.25 Noms d'étiquette XML pour les types ASN.1

Nom de l'unité lexicale – xmlasn1typename

11.25.1 La présente Recommandation | Norme internationale utilise l'unité lexicale "xmlasn1typename" lorsque des types prédéfinis ASN.1 doivent être utilisés comme noms d'étiquette XML.

11.25.2 Le Tableau 4 énumère les séquences de caractères qui doivent constituer le nom "xmlasn1typename" pour chacun des types prédéfinis ASN.1 énumérés au § 16.2. Le type prédéfini ASN.1 est identifié dans la colonne 1 du Tableau 4 par son nom de production. La séquence de caractères à utiliser pour le nom "xmlasn1typename" est identifiée dans la colonne 2 du Tableau 4, sans espace blanc avant comme après ces séquences de caractères.

11.25.3 Le nom "xmlasn1typename" pour les types "UsefulType" (voir § 41.1) correspond à la référence "typereference" utilisée dans leur définition.

11.25.4 La séquence de caractères de l'unité lexicale "xmlasn1typename" pour les types "ObjectClassFieldType" et "InstanceOfType" est spécifiée au § 14.1 et dans l'Annexe C de la Rec. UIT-T X.681 | ISO/CEI 8824-2.

11.25.5 Si le type prédéfini ASN.1 est un type "TaggedType", le type qui détermine le nom "xmlasn1typename" doit être le type "Type" du type "TaggedType" (voir § 30.1). Si ce type est lui-même un type "TaggedType", ce § 11.25.5 est appliqué de façon récursive.

Tableau 4 – Caractères du nom "xmlasn1typename"

Nom de production du type ASN.1	Caractères du nom "xmlasn1typename"
BitStringType	BIT_STRING
BooleanType	BOOLEAN
ChoiceType	CHOICE
EmbeddedPDVType	SEQUENCE
EnumeratedType	ENUMERATED
ExternalType	SEQUENCE
InstanceOfType	SEQUENCE
IntegerType	INTEGER
NullType	NULL
ObjectClassFieldType	*Voir la Rec. UIT-T X.681 | ISO/CEI8824-2, § 14.10 et 14.11*
ObjectIdentifierType	OBJECT_IDENTIFIER
OctetStringType	OCTET_STRING
RealType	REAL
RelativeOIDType	RELATIVE_OID
RestrictedCharacterStringType	*Le nom de type (par exemple* `IA5String`*)*
SequenceType	SEQUENCE
SequenceOfType	SEQUENCE_OF
SetType	SET
SetOfType	SET_OF
TaggedType	*Voir 11.25.5*
UnrestrictedCharacterStringType	SEQUENCE

11.26 Unités lexicales à caractère unique

Noms des unités lexicales:

"{"
"}"
"<"
">"
","
"."
"("
")"
"["
"]"
"-" (TRAIT D'UNION – SIGNE MOINS)
":"
"="
""" (GUILLEMET)
"'" (APOSTROPHE)
" " (ESPACE)
"."
"@"
"|"
"!"
"^"

Une unité lexicale dont le nom est l'un de ceux indiqués ci-dessus est composée du seul caractère sans les guillemets.

11.27 Mots réservés

Noms des mots réservés:

ABSENT	**ENCODED**	**INTEGER**	**RELATIVE-OID**
ABSTRACT-SYNTAX	**END**	**INTERSECTION**	**SEQUENCE**
ALL	**ENUMERATED**	**ISO646String**	**SET**
APPLICATION	**EXCEPT**	**MAX**	**SIZE**
AUTOMATIC	**EXPLICIT**	**MIN**	**STRING**
BEGIN	**EXPORTS**	**MINUS-INFINITY**	**SYNTAX**
BIT	**EXTENSIBILITY**	**NULL**	**T61String**
BMPString	**EXTERNAL**	**NumericString**	**TAGS**
BOOLEAN	**FALSE**	**OBJECT**	**TeletexString**
BY	**FROM**	**ObjectDescriptor**	**TRUE**
CHARACTER	**GeneralizedTime**	**OCTET**	**TYPE-IDENTIFIER**
CHOICE	**GeneralString**	**OF**	**UNION**
CLASS	**GraphicString**	**OPTIONAL**	**UNIQUE**
COMPONENT	**IA5String**	**PATTERN**	**UNIVERSAL**
COMPONENTS	**IDENTIFIER**	**PDV**	**UniversalString**
CONSTRAINED	**IMPLICIT**	**PLUS-INFINITY**	**UTCTime**
CONTAINING	**IMPLIED**	**PRESENT**	**UTF8String**
DEFAULT	**IMPORTS**	**PrintableString**	**VideotexString**
DEFINITIONS	**INCLUDES**	**PRIVATE**	**VisibleString**
EMBEDDED	**INSTANCE**	**REAL**	**WITH**

Les unités lexicales dont les noms figurent ci-dessus sont composées de la séquence des caractères du nom et sont des séquences réservées.

NOTE 1 – Aucun espace blanc ne doit figurer dans ces séquences.

NOTE 2 – Les mots clés **CLASS**, **CONSTRAINED**, **CONTAINING**, **ENCODED**, **INSTANCE**, **SYNTAX** et **UNIQUE** ne sont pas utilisés dans la présente Recommandation | Norme internationale; ils le sont dans les Rec. UIT-T X.681 | ISO/CEI 8824-2, Rec. UIT-T X.682 | ISO/CEI 8824-3 et Rec. UIT-T X.683 | ISO/CEI 8824-4.

12 Définition de module

12.1 Une définition "ModuleDefinition" est spécifiée par les productions suivantes:

```
ModuleDefinition ::=
        ModuleIdentifier
        DEFINITIONS
        TagDefault
        ExtensionDefault
        "::="
        BEGIN
        ModuleBody
        END

ModuleIdentifier ::=
        modulereference
        DefinitiveIdentifier

DefinitiveIdentifier ::=
        "{" DefinitiveObjIdComponentList "}"
    |   empty

DefinitiveObjIdComponentList  ::=
        DefinitiveObjIdComponent
    |   DefinitiveObjIdComponent DefinitiveObjIdComponentList
```

```
DefinitiveObjIdComponent ::=
        NameForm
    |   DefinitiveNumberForm
    |   DefinitiveNameAndNumberForm

DefinitiveNumberForm    ::= number

DefinitiveNameAndNumberForm ::= identifier "(" DefinitiveNumberForm ")"

TagDefault ::=
        EXPLICIT TAGS
    |   IMPLICIT TAGS
    |   AUTOMATIC TAGS
    |   empty

ExtensionDefault ::=
        EXTENSIBILITY IMPLIED
    |   empty

ModuleBody ::=
        Exports Imports AssignmentList
    |   empty

Exports ::=
        EXPORTS SymbolsExported ";"
    |   EXPORTS ALL ";"
    |   empty

SymbolsExported ::=
        SymbolList
    |   empty

Imports ::=
        IMPORTS SymbolsImported ";"
    |   empty

SymbolsImported ::=
        SymbolsFromModuleList
    |   empty

SymbolsFromModuleList ::=
        SymbolsFromModule
    |   SymbolsFromModuleList  SymbolsFromModule

SymbolsFromModule ::=
        SymbolList FROM GlobalModuleReference

GlobalModuleReference ::=
        modulereference AssignedIdentifier

AssignedIdentifier ::=
        ObjectIdentifierValue
    |   DefinedValue
    |   empty

SymbolList ::=
        Symbol
    |   SymbolList "," Symbol

Symbol ::=
        Reference
    |   ParameterizedReference

Reference ::=
        typereference
    |   valuereference
    |   objectclassreference
    |   objectreference
    |   objectsetreference
```

AssignmentList ::=
    Assignment
  | **AssignmentList Assignment**

Assignment ::=
    TypeAssignment
  | **ValueAssignment**
  | **XMLValueAssignment**
  | **ValueSetTypeAssignment**
  | **ObjectClassAssignment**
  | **ObjectAssignment**
  | **ObjectSetAssignment**
  | **ParameterizedAssignment**

NOTE 1 – L'utilisation d'une référence paramétrée "ParameterizedReference" dans les listes "Exports" et "Imports" est spécifiée dans la Rec. UIT-T X.683 | ISO/CEI 8824-4.

NOTE 2 – Lorsque des notations sont données à titre d'exemple (et lors de la définition de types avec des étiquettes de la classe universelle dans la présente Recommandation | Norme internationale), le corps de module "ModuleBody" peut être utilisé à l'extérieur d'une définition de module "ModuleDefinition".

NOTE 3 – Les productions "TypeAssignment", "ValueAssignment", "XMLValueAssignment" et "ValueSetTypeAssignment" sont spécifiées au § 15.

NOTE 4 – La valeur de l'étiquette par défaut "TagDefault" de la définition d'un module porte uniquement sur les types définis explicitement dans ce module. Elle n'a aucun effet sur l'interprétation des types importés.

NOTE 5 – Le point-virgule ";" n'apparaît ni dans la spécification de la liste d'affectation "AssignmentList" ni dans l'une quelconque de ses productions subordonnées; son utilisation est réservée aux développeurs d'outils ASN.1.

12.2 Si l'étiquette par défaut "TagDefault" vaut "empty", on considère qu'elle vaut **`EXPLICIT TAGS`**.

NOTE – Le paragraphe 30 donne la signification des étiquetages explicites, implicites et automatiques: **`EXPLICIT TAGS`**, **`IMPLICIT TAGS`** et **`AUTOMATIC TAGS`**.

12.3 Si l'étiquette par défaut "TagDefault" a la valeur **`AUTOMATIC TAGS`**, on dit que l'étiquetage automatique a été choisi pour le module; sinon, on dit qu'il n'a pas été choisi. L'étiquetage automatique est une transformation syntaxique qui est appliquée (avec des conditions additionnelles) aux productions "ComponentTypeLists" et "AlternativeTypeLists" apparaissant dans la définition du module. Cette transformation est spécifiée formellement aux § 24.7 à 24.9, 26.3 et 28.2 à 28.5 pour ce qui est respectivement des notations pour les types séquences, les types ensembles et les types choix.

12.4 L'option "**`EXTENSIBILITY IMPLIED`**" est équivalente à l'insertion textuelle d'un marqueur d'extension ("**`...`**") dans la définition de chaque type contenu dans le module pour lequel ce marqueur est autorisé. L'emplacement du marqueur d'extension implicite est la dernière position du type pour lequel un marqueur d'extension spécifié d'une manière explicite est autorisé. L'absence de l'option "**`EXTENSIBILITY IMPLIED`**" signifie que l'extensibilité n'est fournie que pour ceux des types du module pour lesquels un marqueur d'extension est présent d'une manière explicite.

NOTE – L'option "**`EXTENSIBILITY IMPLIED`**" porte uniquement sur les types. Elle n'a aucun effet sur les ensembles d'objets et sur les contraintes de sous-typage.

12.5 La référence "modulereference" apparaissant dans la production "ModuleIdentifier" est appelée nom du module.

NOTE – La possibilité de définir un même module ASN.1 comportant plusieurs corps de module "ModuleBody" partageant une même référence de module "modulereference" était (d'une certaine façon) autorisée dans les spécifications précédentes. Cette possibilité est interdite par la présente Recommandation | Norme internationale.

12.6 Les noms de module ne doivent être utilisés qu'une seule fois (sauf dans les cas spécifiés au § 12.9) dans le domaine de visibilité de la définition du module.

12.7 Si l'identificateur définitif "DefinitiveIdentifier" n'est pas vide, la valeur de l'identificateur d'objet indiquée identifie de manière unique et non ambiguë le module ainsi défini. Aucune valeur définie ne peut être utilisée pour définir la valeur de l'identificateur d'objet.

NOTE – Le problème de savoir quelles modifications d'un module nécessitent l'utilisation d'un nouvel identificateur "DefinitiveIdentifier" n'est pas traité dans la présente Recommandation | Norme internationale.

12.8 Si l'identificateur affecté "AssignedIdentifier" n'est pas vide, le module à partir duquel les noms de référence sont importés sera identifié de manière unique et non ambiguë soit par la valeur d'identificateur d'objet "ObjectIdentifierValue", soit par la valeur définie "DefinedValue". Si c'est la valeur "DefinedValue" qui assure l'identification, elle devra être du type identificateur d'objet. Chaque référence de valeur "valuereference" apparaissant textuellement dans un identificateur "AssignedIdentifier" satisfera à l'une des règles suivantes:

a) elle est définie dans la liste d'affectation "AssignmentList" du module ainsi défini, et toutes les références de valeur "valuereference" apparaissant textuellement dans le membre droit de la déclaration d'affectation satisfont aussi à la présente règle (règle "a") ou à la suivante (règle "b");

b) elle apparaît en tant que symbole "Symbol" d'une liste "SymbolsFromModule" dont l'identificateur affecté "AssignedIdentifier" ne contient textuellement pas de référence de valeur "valuereference".

NOTE – Il est recommandé d'affecter au module un identificateur d'objet de manière à ce que d'autres modules puissent s'y référer sans ambiguïté.

12.9 La référence globale de module "GlobalModuleReference" d'une liste "SymbolsFromModule" apparaîtra dans la définition "ModuleDefinition" d'un autre module, sauf si elle comporte un identificateur "DefinitiveIdentifier" non vide, dans ce cas la référence de module "modulereference" peut être différente.

NOTE – On ne doit utiliser une référence de module "modulereference" différente de celle utilisée dans l'autre module que lorsqu'il faut importer des symboles de deux modules portant le même nom (l'appellation des modules n'ayant pas respecté les dispositions du § 12.6). L'utilisation d'autres noms distincts permet d'utiliser ces noms dans le corps du module (voir § 12.15).

12.10 Lorsqu'on utilise à la fois une référence de module "modulereference" et un identificateur affecté "AssignedIdentifier" non vide pour désigner un module, l'identificateur sera considéré comme l'identificateur définitif du module.

12.11 Lorsque le module désigné possède un identificateur définitif "DefinitiveIdentifier" non vide, la référence globale "GlobalModuleReference" désignant ce module possédera un identificateur affecté "AssignedIdentifier" non vide.

12.12 Lorsque la liste "SymbolsExported" de la déclaration "Exports" est choisie:

a) chaque symbole "Symbol" de la liste "SymbolsExported" satisfera à une et une seule des deux conditions suivantes:

i) soit être défini dans le module en cours de constitution;

ii) soit apparaître une seule fois dans la liste "SymbolsImported" de la déclaration "Imports";

b) tous les symboles "Symbol" qu'il serait utile de pouvoir désigner depuis l'extérieur du module seront inclus dans la liste "SymbolsExported", et seront les seuls à pouvoir être ainsi désignés (sous réserve de l'assouplissement spécifié au § 12.13);

c) lorsqu'il n'existe pas de tels symboles "Symbol", la liste "SymbolsExported" (et non pas la déclaration "Exports") se verra affecter la valeur vide.

12.13 Lorsque la déclaration "Exports" se voit affecter la valeur "empty" (vide) ou la valeur "**`EXPORTS ALL`**", tous les symboles "Symbol" définis dans le module ou importés par le module peuvent être désignés par d'autres modules, sous réserve de la contrainte spécifiée au § 12.12 a).

NOTE – La valeur "empty" (vide) de la déclaration "Exports" est incluse pour des raisons de compatibilité en amont.

12.14 Les identificateurs apparaissant dans une liste "NamedNumberList", "Enumeration" ou "NamedBitList" sont implicitement exportés si la référence de type "typereference" les définissant est exportée ou apparaît en tant que composant (ou sous-composant) d'un type exporté.

12.15 Lorsque la liste "SymbolImported" de la déclaration "Imports" est choisie:

a) chaque symbole "Symbol" de la liste "SymbolsFromModule" soit sera défini dans le corps du module désigné par la référence globale "GlobalModuleReference" de la déclaration "SymbolsFromModule", soit figurera dans la déclaration "Imports" de ce module. L'importation d'un symbole "Symbol" figurant dans la déclaration "Imports" du module désigné n'est autorisée que s'il existe une seule occurrence de ce symbole dans cette déclaration et que le symbole n'est pas défini dans ce module;

NOTE 1 – Ceci n'interdit pas d'importer dans un module deux symboles de même nom définis dans deux autres modules différents. Mais si le même nom de symbole figure plus d'une fois dans la déclaration "Imports" d'un module A, ce symbole ne peut plus être exporté de A pour être importé par un autre module B.

b) si la forme "SymbolExported" de la déclaration "Exports" est choisie dans la définition du module désigné par la référence "GlobalModuleReference" dans la déclaration "SymbolsFromModule", le symbole "Symbol" devra apparaître dans la liste "SymbolsExported";

c) seuls les symboles "Symbol" figurant dans une liste "SymbolList" d'une déclaration "SymbolsFromModule" peuvent apparaître dans une déclaration de référence externe

"External<X>Reference" (<X> remplaçant ici "Value", "Type", "Object", "Objectclass" ou "Objectset") pour laquelle la référence "modulereference" correspond à la déclaration "GlobalModuleReference" de cette déclaration "SymbolsFromModule";

d) lorsqu'il n'existe pas de symbole "Symbol" à importer, la forme "empty" (vide) de la déclaration "SymbolsImported" sera choisie;

NOTE 2 – Une des conséquences de c) et d) est que la déclaration "**IMPORTS**"; implique que le module ne peut pas contenir de référence externe "External<X>Reference".

e) les déclarations "SymbolsFromModule" de la liste "SymbolsFromModuleList" comporteront chacune une référence globale "GlobalModuleReference" telle que:

i) les références "modulereference" qui y sont indiquées soient toutes différentes les unes des autres, et différentes également de la référence "modulereference" associé au module importateur;

ii) les identificateurs affectés "AssignedIdentifier" désignent, lorsqu'ils ne sont pas vides, des valeurs d'identificateurs d'objet toutes différentes les unes des autres, et différentes également de la valeur de l'identificateur d'objet (s'il y en a une) associée au module importateur.

12.16 Lorsque la forme "empty" (vide) de la déclaration "Imports" est choisie, le module peut quand même faire référence à des symboles "Symbol" définis dans d'autres modules au moyen de déclarations "External<X>Reference".

NOTE – La valeur "empty" (vide) de la déclaration "Imports" est incluse pour des raisons de compatibilité en amont.

12.17 Les identificateurs apparaissant dans une liste "NamedNumberList", "Enumeration" ou "NamedBitList" sont implicitement importés si la référence de type "typereference" les définissant est importée ou apparaît en tant que composant (ou sous-composant) d'un type importé.

12.18 Un symbole "Symbol" d'une déclaration "SymbolsFromModule" peut apparaître dans le corps de module "ModuleBody" en tant que référence "Reference". La signification qui lui est associée est celle qu'il a dans le module désigné par la référence "GlobalModuleReference" correspondante.

12.19 Lorsque le symbole "Symbol" donné apparaît également dans une liste d'affectation "AssignmentList" (ce qui est déconseillé) ou apparaît dans une ou plusieurs autres instances de déclaration "SymbolsFromModule", il ne sera utilisé que dans une référence "External<X>Reference". Autrement, il peut être utilisé directement sous la forme d'une référence "Reference".

12.20 Sauf mention contraire, les différentes formes d'affectation "Assignment" sont définies dans les paragraphes suivants de la présente Recommandation | Norme internationale:

Forme d'affectation	*Paragraphe de définition*
"TypeAssignment"	15.1
"ValueAssignment"	15.2
"XMLValueAssignment"	15.2
"ValueSetTypeAssignment"	15.6
"ObjectClassAssignment"	Rec. UIT-T X.681 | ISO/CEI 8824-2, 9.1
"ObjectAssignment"	Rec. UIT-T X.681 | ISO/CEI 8824-2, 11.1
"ObjectSetAssignment"	Rec. UIT-T X.681 | ISO/CEI 8824-2, 12.1
"ParameterizedAssignment"	Rec. UIT-T X.683 | ISO/CEI 8824-4, 8.1

Le premier symbole de chaque affectation "Assignment" est l'une des formes possibles de la référence "Reference", indiquant le nom de la référence à définir. Un même nom de référence ne peut être donné à deux affectations d'une même liste d'affectation "AssignmentList".

13 Références des définitions de types et de valeurs

13.1 Les productions suivantes relatives aux types et valeurs définis:

```
DefinedType ::=
        ExternalTypeReference
    |   Typereference
    |   ParameterizedType
    |   ParameterizedValueSetType
```

DefinedValue ::=
ExternalValueReference
| **Valuereference**
| **ParameterizedValue**

spécifient les séquences qui seront utilisées pour référencer les définitions de types et de valeurs. Le type identifié par un type paramétré "ParameterizedType" ou un type d'ensemble de valeurs paramétrées "ParameterizedValueSetType" et la valeur identifiée par une valeur paramétrée "ParameterizedValue" sont spécifiés dans la Rec. UIT-T X.683 | ISO/CEI 8824-4.

13.2 La production "NonParameterizedTypeName":

NonParameterizedTypeName ::=
ExternalTypeReference
| **typereference**
| **xmlasn1typename**

est utilisée lorsqu'un nom d'étiquette XML est nécessaire pour représenter un type ASN.1.

13.3 La troisième forme ne doit pas être utilisée comme nom "NonParameterizedTypeName" dans la valeur "XMLTypedValue" de "XMLValueAssignment" (voir § 15.2) ou de "XMLOpenTypeFieldVal" (voir le § 14.6 de la Rec. UIT-T X.681 | ISO/CEI 8824-2) lorsque la notation de valeur XML est utilisée dans un module ASN.1 si le nom "xmlasn1typename" vaut "CHOICE", "ENUMERATED", "SEQUENCE", "SEQUENCE_OF", "SET" ou "SET_OF".

NOTE – Cette restriction imposée à la notation de valeur XML utilisée dans un module ASN.1 vient du fait que ces noms "xmlasn1typename" ne définissent pas un type ASN.1. Cette restriction est absente lorsque cette notation est utilisée dans des règles de codage (par exemple XER, voir la Rec. UIT-T X.693 | ISO/CEI 8825-4) car les étiquettes XML formées à partir de noms "xmlasn1typename" ne sont pas utilisées pour déterminer les types qui sont codés.

13.4 En-dehors des cas spécifiés au § 12.18, les formes "typereference", "valuereference", "ParameterizedType", "ParameterizedValueSetType" et "ParameterizedValue" ne devront être utilisées que si la référence indiquée provient du corps de module "ModuleBody" dans lequel un type ou une valeur est affecté (voir § 15.1 et 15.2) à la référence de type "typereference" ou de valeur "valuereference".

13.5 Les références externes de type "ExternalTypeReference" (respectivement de valeur "ExternalValueReference") ne seront utilisées que si la référence de type "typereference" (respectivement de valeur "valuereference") correspondante:

a) s'est déjà vue affecter un type (respectivement une valeur) (voir § 15.1 et 15.2); ou

b) est présente dans la déclaration "Imports"

dans le corps de module "ModuleBody" utilisé pour définir la référence "modulereference" correspondante. Il n'est permis de faire référence à un nom dans la déclaration "Imports" d'un autre module que s'il n'existe pas plus d'une occurrence de ce symbole "Symbol" dans la déclaration "Imports".

NOTE – Ceci n'empêche pas d'importer dans un module deux symboles "Symbol" identiques définis dans deux autres modules différents. Mais si un même symbole "Symbol" apparaît plus d'une fois dans la déclaration **IMPORTS** d'un module **A**, il n'est plus possible de faire référence à ce symbole à partir du module **A** dans une référence externe.

13.6 Une référence externe ne sera utilisée dans un module que pour désigner un nom de référence défini dans un autre module; cette référence est spécifiée par les productions suivantes:

ExternalTypeReference ::=
modulereference
"."
typereference

ExternalValueReference ::=
modulereference
"."
valuereference

NOTE – Des productions supplémentaires de définition de référence externe ("ExternalClassReference", "ExternalObjectReference" et "ExternalObjectSetReference") sont spécifiées dans la Rec. UIT-T X.681 | ISO/CEI 8824-2.

13.7 Lorsque dans le module importateur, la déclaration d'importation "Imports" a la forme "SymbolsImported", la référence "modulereference" de la référence externe apparaîtra dans la référence "GlobalModuleReference" d'une et une seule liste "SymbolsFromModule" de la déclaration "SymbolsImported". Lorsque dans le module importateur, la déclaration d'importation "Imports" est vide, la référence "modulereference" de la référence externe apparaîtra dans la définition "ModuleDefinition" du module (différent du module importateur) dans lequel la référence "Reference" invoquée est définie.

13.8 Lorsqu'un type "DefinedType" est utilisé dans une notation gouvernée par un type "Type" (par exemple une contrainte "SubtypeConstraint"), le type "DefinedType" doit être compatible avec le type "Type" gouvernant, comme spécifié au § B.6.2.

13.9 Chaque occurrence d'une valeur "DefinedValue" dans une spécification ASN.1 est gouvernée par un type "Type" et cette valeur "DefinedValue" doit faire référence à une valeur d'un type compatible avec le type "Type" gouvernant, comme spécifié au § B.6.2.

14 Notation permettant de faire référence à des composants ASN.1

14.1 Il est nécessaire de se référer formellement à des composants de types, valeurs, etc., ASN.1 à de multiples fins, par exemple, lorsqu'on écrit un texte, pour identifier un type particulier dans un module ASN.1 donné. Le présent paragraphe définit une notation qui peut être utilisée pour effectuer de telles références.

14.2 Cette notation permet d'identifier n'importe quel composant d'un type ensemble ou séquence (qu'il soit présent à titre obligatoire ou optionnel dans le type).

14.3 Il est possible de faire référence à toutes les parties d'une définition de type ASN.1 en utilisant la structure syntaxique de référence absolue "AbsoluteReference":

```
AbsoluteReference ::= "@" ModuleIdentifier
                      "."
                      ItemSpec

ItemSpec ::=
                      typereference
                |     ItemId "." ComponentId

ItemId ::= ItemSpec

ComponentId ::=
                      identifier
                |     number
                |     "*"
```

NOTE – La production "AbsoluteReference" n'est pas utilisée ailleurs dans la présente Recommandation | Norme internationale. Elle est établie aux fins indiquées au § 14.1.

14.4 L'identificateur de module "ModuleIdentifier" identifie un module ASN.1 (voir § 12.1).

14.5 Lorsque la première forme de l'identificateur définitif "DefinitiveIdentifier" est utilisée dans l'identificateur de module "ModuleIdentifier", l'identificateur définitif "DefinitiveIdentifier" identifie de manière unique et non ambiguë le module à partir duquel il est fait référence à un nom.

14.6 La référence "typereference" est celle d'un type ASN.1 quelconque défini dans le module identifié par "ModuleIdentifier".

14.7 L'identificateur de composant "ComponentId" de chaque spécification "ItemSpec" identifie un composant du type désigné par "ItemId". Il s'agira du dernier identificateur "ComponentId" si le composant identifié n'est pas un type ensemble, séquence, ensemble-de, séquence-de ou choix.

14.8 La forme "identifier" de "ComponentId" peut être utilisée si l'identificateur "ItemId" parent est du type ensemble ou séquence; cet identificateur doit alors être l'un des identificateurs "identifier" du type nommé "NamedType" de la liste des types de composants "ComponentTypeLists" de cet ensemble ou séquence. Il peut aussi être utilisé si "ItemId" désigne un type choix; il doit alors être l'un des identificateurs "identifier" de type nommé "NamedType" de la liste des formes possibles "AlternativeTypeLists" de ce type choix. Il ne peut pas être utilisé dans d'autres cas.

14.9 La forme "number" de "ComponentId" ne peut être utilisée que si l'identificateur "ItemId" est un type séquence-de ou ensemble-de. La valeur du numéro identifie l'instance du type donné dans la structure séquence-de ou ensemble-de, le numéro "1" correspondant à la première instance. Le numéro "0" identifie un composant conceptuel de type entier (non présent explicitement dans le transfert) qui indique le nombre d'instances du type dans la structure séquence-de ou ensemble-de présentes dans la valeur du type contenant.

14.10 La forme "*" de "ComponentId" ne peut être utilisée que si l'identificateur "ItemId" est un type séquence-de ou ensemble-de. Toute sémantique associée à l'utilisation de la forme "*" s'applique à tous les composants de la structure séquence-de ou ensemble-de.

NOTE – Dans l'exemple suivant:

```
M DEFINITIONS ::= BEGIN
      T ::= SEQUENCE {
            a     BOOLEAN,
            b     SET OF INTEGER
      }
END
```

il est possible de faire référence aux composants de "**T**" depuis un texte hors d'un module ASN.1 (ou depuis un commentaire) de la manière suivante par exemple:

```
-- si (@M.T.b.0 est impair) alors:
--        (@M.T.b.* est un entier impair)
```

dans laquelle il est déclaré que si le nombre de composants de **b** est impair, alors tous les composants de **b** doivent être impairs.

15 Affectation des types et des valeurs

15.1 Un type sera affecté à une référence de type "typereference" par la notation spécifiée dans la production "TypeAssignment":

TypeAssignment ::=
typereference
"::="
Type

La référence "typereference" ne devra pas être l'un des mots ASN.1 réservés (voir § 11.27).

15.2 Une valeur sera affectée à une référence de valeur "valuereference" par la notation spécifiée dans la production "ValueAssignment" ou "XMLValueAssignment":

ValueAssignment ::=
valuereference
Type
"::="
Value

XMLValueAssignment ::=
valuereference
"::="
XMLTypedValue

XMLTypedValue ::=
"<" & NonParameterizedTypeName ">"
XMLValue
"</" & NonParameterizedTypeName ">"
| **"<" & NonParameterizedTypeName "/>"**

La valeur affectée à la référence "valuereference" dans la production "ValueAssignment" est "Value", elle est gouvernée par le "Type" et sera une notation de valeur associée au type défini par "Type" (comme spécifié au § 15.3). La valeur affectée à la référence "valuereference" dans la production "XMLValueAssignment" est "XMLValue" (voir § 16.7) et sera une notation de valeur associée au type défini par le nom "NonParameterizedTypeName" (comme spécifié au § 15.4). S'il s'agit de l'unité lexicale "xmlasn1typename", celle-ci identifie le type prédéfini ASN.1 à la ligne correspondante du Tableau 4 (voir aussi § 13.3).

15.3 "Value" est une notation de valeur associée à un type donné comme spécifié au § 16.7.

15.4 "XMLValue" est une notation de valeur associée à un type donné si "XMLValue" est une notation "XMLBuiltinValue" pour le type (voir § 16.10).

15.5 La deuxième forme de "XMLTypedValue" (utilisation d'une étiquette d'élément vide XML) ne peut être utilisée que si une instance de la production "XMLValue" est vide.

NOTE – Si la production "XMLValue" est une chaîne "xmlcstring" contenant uniquement un espace blanc, elle n'est pas vide et la deuxième forme ne peut pas être utilisée.

15.6 La notation spécifiée par la production "ValueSetTypeAssignment" permet d'affecter un ensemble de valeurs à une référence de type "typereference":

ValueSetTypeAssignment ::=
typereference
Type
"::="
ValueSet

Cette notation affecte à "typereference" le type défini comme un sous-type du type "Type", et contenant exactement les valeurs spécifiées ou autorisées par "ValueSet". La référence "typereference" ne doit pas être un mot ASN.1 réservé (voir § 11.27), et il sera possible d'y faire référence comme à un type. L'ensemble de valeurs "ValueSet" est défini au § 15.7.

15.7 Un ensemble de valeurs d'un type donné est spécifié par la notation "ValueSet":

ValueSet ::= "{" ElementSetSpecs "}"

L'ensemble de valeurs comprend toutes les valeurs, au nombre d'une au moins, spécifiées par la production "ElementSetSpecs" (voir le § 46).

15.8 La production "ValueSetTypeAssignment" devient alors:

typereference
Type
"::="
"{" ElementSetSpecs "}"

Quel que soit l'objectif, y compris l'application de règles de codage, cette production est exactement équivalente à l'utilisation de la production:

typereference
"::="
Type
"(" ElementSetSpecs ")"

avec les mêmes spécifications "Type" et "ElementSetSpecs".

16 Définition des types et des valeurs

16.1 Un type est spécifié par la notation "Type":

Type ::= BuiltinType | ReferencedType | ConstrainedType

16.2 Les types prédéfinis de la notation ASN.1 sont spécifiés par la notation "BuiltinType" définie comme suit:

BuiltinType ::=
BitStringType
| **BooleanType**
| **CharacterStringType**
| **ChoiceType**
| **EmbeddedPDVType**
| **EnumeratedType**
| **ExternalType**
| **InstanceOfType**
| **IntegerType**
| **NullType**
| **ObjectClassFieldType**
| **ObjectIdentifierType**
| **OctetStringType**
| **RealType**
| **RelativeOIDType**
| **SequenceType**
| **SequenceOfType**
| **SetType**
| **SetOfType**
| **TaggedType**

Les diverses notations de "BuiltinType" sont définies dans les paragraphes suivants (de la présente Recommandation | Norme internationale sauf indication contraire):

BitStringType	21
BooleanType	17
CharacterStringType	36
ChoiceType	28
EmbeddedPDVType	33
EnumeratedType	19
ExternalType	34
InstanceOfType	Rec. UIT-T X.681 | ISO/CEI 8824-2, Annexe C
IntegerType	18
NullType	23
ObjectClassFieldType	Rec. UIT-T X.681 | ISO/CEI 8824-2, 14.1
ObjectIdentifierType	31
OctetStringType	22
RealType	20
RelativeOIDType	32
SequenceType	24
SequenceOfType	25
SetType	26
SetOfType	27
TaggedType	30

16.3 Les types de la notation ASN.1 auxquels il est fait référence sont spécifiés par la notation "ReferencedType":

ReferencedType ::=
DefinedType
| **UsefulType**
| **SelectionType**
| **TypeFromObject**
| **ValueSetFromObjects**

La notation "ReferencedType" offre une manière différente de faire référence à un autre type (et en dernier lieu un type prédéfini). Les diverses notations de "ReferencedType" et la manière utilisée pour déterminer le type auquel elles renvoient sont spécifiées dans les paragraphes suivants (de la présente Recommandation | Norme internationale sauf indication contraire):

DefinedType	13.1
UsefulType	41.1
SelectionType	29
TypeFromObject	Rec. UIT-T X.681 | ISO/CEI 8824-2, § 15
ValueSetFromObjects	Rec. UIT-T X.681 | ISO/CEI 8824-2, § 15

16.4 Le type contraint "ConstrainedType" est défini au § 45.

16.5 Conformément à la présente Recommandation | Norme internationale, il faut utiliser la notation "NamedType" suivante pour spécifier les composants des types ensemble, séquence et choix:

NamedType ::= identifier Type

16.6 L'identificateur "identifier" est utilisé pour désigner sans ambiguïté les composants d'un type ensemble, séquence ou choix dans les notations de valeurs, dans les contraintes de sous-typage interne et dans les contraintes relationnelles entre composants (voir la Rec. UIT-T X.682 | ISO/CEI 8824-3). Il ne fait pas partie du type et n'a pas d'effet sur celui-ci.

16.7 Une valeur d'un type donné est spécifiée par la notation "Value" ou par la notation "XMLValue":

Value ::=
BuiltinValue
| **ReferencedValue**
| **ObjectClassFieldValue**

XMLValue ::=
XMLBuiltinValue
| **XMLObjectClassFieldValue**

NOTE 1 – "ObjectClassFieldValue" et "XMLObjectClassFieldValue" sont définis au § 14.6 de la Rec. UIT-T X.681 | ISO/CEI 8824-2.

NOTE 2 – "XMLValue" est utilisé uniquement dans "XMLTypedValue".

16.8 Si une partie quelconque de la production "XMLValue" se traduit par une étiquette de début XML immédiatement suivie d'une étiquette de fin XML, éventuellement séparées par un espace blanc inséré comme autorisé au § 11.1.4 (par exemple, <champ1></champ1>), ces deux étiquettes XML, et l'éventuel espace blanc, peuvent être remplacés par une seule étiquette d'élément vide XML (<champ1/>).

NOTE – Si un caractère espace blanc, à l'exception d'un espace blanc inséré comme autorisé au § 11.1.4, est présent entre le caractère ">" final de l'étiquette de début et le caractère "<" initial de l'étiquette de fin, la condition ci-dessus n'est pas satisfaite.

16.9 Les valeurs des types prédéfinis de la notation ASN.1 peuvent être spécifiées par la notation "XMLBuiltinValue" (voir § 16.10) ou par la notation "BuiltinValue" définie comme suit:

BuiltinValue ::=
BitStringValue
| **BooleanValue**
| **CharacterStringValue**
| **ChoiceValue**
| **EmbeddedPDVValue**
| **EnumeratedValue**
| **ExternalValue**
| **InstanceOfValue**
| **IntegerValue**
| **NullValue**
| **ObjectIdentifierValue**
| **OctetStringValue**
| **RealValue**
| **RelativeOIDValue**
| **SequenceValue**
| **SequenceOfValue**
| **SetValue**
| **SetOfValue**
| **TaggedValue**

Chacune des diverses notations de "BuiltinValue" est définie dans le même paragraphe que la notation de "BuiltinType" correspondante, selon la liste du § 16.2.

16.10 La valeur "XMLBuiltinValue" est définie comme suit:

XMLBuiltinValue ::=
XMLBitStringValue
| **XMLBooleanValue**
| **XMLCharacterStringValue**
| **XMLChoiceValue**
| **XMLEmbeddedPDVValue**
| **XMLEnumeratedValue**
| **XMLExternalValue**
| **XMLInstanceOfValue**
| **XMLIntegerValue**
| **XMLNullValue**
| **XMLObjectIdentifierValue**
| **XMLOctetStringValue**
| **XMLRealValue**
| **XMLRelativeOIDValue**
| **XMLSequenceValue**
| **XMLSequenceOfValue**
| **XMLSetValue**
| **XMLSetOfValue**
| **XMLTaggedValue**

Chacune des diverses notations de "XMLBuiltinValue" est définie dans le même paragraphe que la notation de "BuiltinType" correspondante, selon la liste du § 16.2.

16.11 Les valeurs de la notation ASN.1 auxquelles il est fait référence sont spécifiées par la notation "ReferencedValue" suivante:

```
ReferencedValue ::=
        DefinedValue
    |   ValueFromObject
```

La notation "ReferencedValue" offre une manière différente de faire référence à une autre valeur (et en dernier lieu une valeur prédéfinie). Les différentes notations de "ReferencedValue" et la manière utilisée pour déterminer la valeur à laquelle elles renvoient sont spécifiées dans les paragraphes suivants (de la présente Recommandation | Norme internationale sauf indication contraire):

DefinedValue	13.1
ValueFromObject	Rec. UIT-T X.681 | ISO/CEI 8824-2, § 15

16.12 Indépendamment du fait qu'un type soit un type "BuiltinType", "ReferencedType" ou "ConstrainedType", ses valeurs peuvent être spécifiées par une valeur "BuiltinValue" ou "ReferencedValue" de ce type.

16.13 La valeur d'un type désigné à l'aide de la notation "NamedType" sera définie par la notation "NamedValue" ou, lorsqu'elle est utilisée dans le cadre d'une valeur "XMLValue", par la notation "XMLNamedValue". Ces productions sont les suivantes:

```
NamedValue ::= identifier Value

XMLNamedValue ::= "<" & identifier  ">" XMLValue "</" & identifier ">"
```

où l'identificateur "identifier" est le même que celui utilisé dans la notation "NamedType".

NOTE – L'identificateur "identifier" fait partie de la notation et non de la valeur proprement dite. Il sert à désigner sans ambiguïté les composants d'un type ensemble, séquence ou choix.

16.14 La présence implicite (voir § 12.4) ou explicite d'un marqueur d'extension (voir § 6) dans la définition d'un type n'a aucun effet sur la notation de la valeur. Ceci signifie que la notation de valeur associée à un type avec un marqueur d'extension est exactement la même que si le marqueur d'extension était absent.

NOTE – Conformément au § 46.8, une notation de valeur utilisée dans une contrainte de sous-typage ne doit pas faire référence à une valeur qui n'est pas dans la racine d'extension du type parent.

17 Notation du type booléen (boolean type)

17.1 Le type booléen (voir § 3.6.7) est déclaré par la notation "BooleanType":

```
BooleanType ::= BOOLEAN
```

17.2 L'étiquette des types définis par cette notation est le numéro 1 de la classe universelle.

17.3 La valeur d'un type booléen (voir § 3.6.73 et 3.6.38) est définie par la notation "BooleanValue" ou, lorsqu'elle est utilisée comme une valeur "XMLValue", par la notation "XMLBooleanValue". Ces productions sont les suivantes:

```
BooleanValue ::= TRUE | FALSE

XMLBooleanValue ::=
        "<" & "true" "/>"
    |   "<" & "false" "/>"
```

18 Notation du type entier (integer type)

18.1 Le type entier (voir § 3.6.41) est déclaré par la notation "IntegerType":

```
IntegerType ::=
        INTEGER
    |   INTEGER "{" NamedNumberList "}"

NamedNumberList ::=
        NamedNumber
    |   NamedNumberList "," NamedNumber

NamedNumber ::=
        identifier "(" SignedNumber ")"
    |   identifier "(" DefinedValue ")"
```

SignedNumber ::=
number
| **"-" number**

18.2 La seconde forme possible de "SignedNumber" n'est pas utilisée si "number" est nul.

18.3 Dans la déclaration de type, la liste "NamedNumberList" n'est pas significative. Elle est seulement utilisée dans la notation de valeur spécifiée au § 18.9.

18.4 La référence de valeur "valuereference" de la valeur définie "DefinedValue" sera du type entier.

NOTE – Un identificateur "identifier" ne pouvant servir à spécifier la valeur associée à "NamedNumber", la valeur définie "DefinedValue" ne peut jamais être prise pour un entier "IntegerValue". Ainsi, dans le cas suivant:

```
a INTEGER ::= 1
T1 ::= INTEGER { a(2) }
T2 ::= INTEGER { a(3), b(a) }
c T2 ::= b
d T2 ::= a
```

c désigne la valeur 1 puisqu'il ne peut être une référence à la deuxième ou troisième occurrence de **a**, et **d** désigne la valeur 3.

18.5 Les valeurs des nombres signés "SignedNumber" et des valeurs définies "DefinedValue" apparaissant dans la liste "NamedNumberList" seront toutes différentes et représentent les valeurs distinctives du type entier.

18.6 Les identificateurs "identifier" apparaissant dans la liste "NamedNumberList" seront tous différents.

18.7 L'ordre des nombres "NamedNumber" de la liste "NamedNumberList" n'est pas significatif.

18.8 L'étiquette des types définis par cette notation est le numéro 2 de la classe universelle.

18.9 La valeur d'un type entier est déclarée par la notation "IntegerValue" ou, lorsqu'elle est utilisée comme une valeur "XMLValue", par la notation "XMLIntegerValue". Ces productions sont:

IntegerValue ::=
SignedNumber
| **identifier**

XMLIntegerValue ::=
SignedNumber
| **"<" & identifier "/>"**

18.10 L'identificateur "identifier" de la valeur "IntegerValue" ou de la valeur "XMLIntegerValue" sera l'un des identificateurs du type "IntegerType" auquel la valeur est associée, et représentera le nombre correspondant.

NOTE – Pour faire référence à un entier pour lequel un identificateur "identifier" a été défini, il est préférable d'utiliser l'identificateur "identifier" de la valeur "IntegerValue" ou de la valeur "XMLIntegerValue".

18.11 Dans une instance de notation de valeur pour un type entier avec une liste "NamedNumberList", toute occurrence d'un nom qui est à la fois un identificateur "identifier" de la liste "NamedNumberList" et un nom de référence sera interprétée comme l'identificateur "identifier".

19 Notation du type énuméré (enumerated type)

19.1 Le type énuméré (voir § 3.6.24) est déclaré par la notation "EnumeratedType":

EnumeratedType ::=
ENUMERATED "{" Enumerations "}"

Enumerations ::=
RootEnumeration
| **RootEnumeration "," "..." ExceptionSpec**
| **RootEnumeration "," "..." ExceptionSpec "," AdditionalEnumeration**

RootEnumeration ::= Enumeration

AdditionalEnumeration ::= Enumeration

Enumeration ::= EnumerationItem | EnumerationItem "," Enumeration

EnumerationItem ::= identifier | NamedNumber

NOTE 1 – Chaque valeur d'un type énuméré a un identificateur associé à un entier distinct. Toutefois, les valeurs elles-mêmes ne sont pas censées avoir une sémantique d'entier. Si "EnumerationItem" a la forme "NamedNumber", il rend possible le contrôle de la représentation de la valeur pour faciliter des extensions compatibles.

NOTE 2 – Les valeurs numériques des nombres nommés "NamedNumber" d'une énumération "RootEnumeration" ne sont pas nécessairement ordonnées ni consécutives et les valeurs numériques des nombres "NamedNumber" d'une énumération "AdditionalEnumeration" sont ordonnées mais pas nécessairement consécutives.

19.2 L'identificateur "identifier" et le nombre signé "SignedNumber" des différents nombres "NamedNumber" seront différents de tous les autres identificateurs et nombres signés de l'énumération "Enumeration". Les § 18.2 et 18.4 s'appliquent aussi à chaque nombre nommé "NamedNumber".

19.3 Chaque élément "EnumerationItem" (dans un type énuméré "EnumeratedType") défini par un identificateur "identifier" se voit affecter un entier non négatif distinct. Pour l'énumération "RootEnumeration", on utilise les entiers successifs à partir de 0, à l'exception de ceux qui sont utilisés dans les unités lexicales "EnumerationItem" définies comme nombres nommés "NamedNumber".

NOTE – Un entier est associé à l'unité lexicale "EnumerationItem" pour aider à en définir les règles de codage. Mais cet artifice n'est pas utilisé ailleurs dans la spécification de la notation ASN.1.

19.4 La valeur associée à chaque nouvel élément "EnumerationItem" sera supérieure à celle de toute énumération "AdditionalEnumeration" définie précédemment dans le type.

19.5 Lorsqu'un nombre nommé "NamedNumber" est utilisé pour définir un élément "EnumerationItem" de l'énumération "AdditionalEnumeration", la valeur qui lui est associée sera différente de la valeur de tout élément "EnumerationItem" défini précédemment (dans ce type), indépendamment du fait que l'élément "EnumerationItem" défini précédemment figure ou non dans la racine de l'énumération. A titre d'exemple:

```
A ::= ENUMERATED {a, b, ..., c(0)}      -- non valide, car 'a' et 'c' sont tous
                                        -- deux nuls
B ::= ENUMERATED {a, b, ..., c, d(2)}   -- non valide, car 'c' et 'd' sont tous
                                        -- deux égaux à 2
C ::= ENUMERATED {a, b(3), ..., c(1)}   -- valide, 'c' = 1
D ::= ENUMERATED {a, b, ..., c(2)}      -- valide, 'c' = 2
```

19.6 La valeur associée au premier élément "EnumerationItem" de "AdditionalEnumeration" qui est défini par un identificateur "identifier" (et non par un nombre nommé "NamedNumber") sera la plus petite valeur à laquelle n'est associé aucun élément "EnumerationItem" dans "RootEnumeration" telle que la valeur associée à tout élément "EnumerationItem" précédent de "AdditionalEnumeration" (s'il en existe) soit inférieure. Les exemples suivants sont tous valides:

```
A ::= ENUMERATED {a, b, ..., c}         -- c = 2
B ::= ENUMERATED {a, b, c(0), ..., d}   -- d = 3
C ::= ENUMERATED {a, b, ..., c(3), d}   -- d = 4
D ::= ENUMERATED {a, z(25), ..., d}     -- d = 1
```

19.7 L'étiquette du type énuméré est le numéro 10 de la classe universelle.

19.8 La valeur d'un type énuméré est déclarée par la notation "EnumeratedValue" ou, lorsqu'elle est utilisée comme une valeur "XMLValue", par la notation "XMLEnumeratedValue". Ces productions sont:

EnumeratedValue ::= identifier

XMLEnumeratedValue ::= "<" & identifier "/>"

19.9 L'identificateur "identifier" d'une valeur "EnumeratedValue" ou d'une valeur "XMLEnumeratedValue" sera identique à l'identificateur présent dans la séquence "EnumeratedType" et auquel la valeur est associée.

19.10 Dans une instance de notation de valeur pour un type énuméré, toute occurrence d'un nom qui est à la fois un identificateur "identifier" de l'énumération "Enumeration" et un nom de référence sera interprétée comme l'identificateur "identifier".

20 Notation du type réel

20.1 Le type réel (voir § 3.6.54) est déclaré par la notation "RealType":

RealType ::= `REAL`

20.2 L'étiquette du type réel est le numéro 9 de la classe universelle.

20.3 Les valeurs du type réel sont les valeurs **PLUS-INFINITY** et **MINUS-INFINITY**, ainsi que les nombres réels qui peuvent être représentés par la formule suivante faisant intervenir trois entiers M, B et E:

$$M \times B^E$$

où M est la mantisse, B la base et E l'exposant.

20.4 Le type réel possède un type associé qui sert à préciser la définition des valeurs abstraites de type réel et qui sert également à prendre en charge les notations de valeurs et de sous-types du type réel.

NOTE – Les règles de codage peuvent définir un type différent servant à spécifier les codages; elles peuvent aussi spécifier les codages sans faire référence au type associé. En particulier, les règles de codage de base BER et compact PER fournissent un codage décimal codé binaire (BCD, *binary-coded decimal*) si la "base" est égale à 10, et un codage permettant une transformation efficace vers et depuis les représentations en virgule flottante sur matériel informatique si la "base" est égale à 2.

20.5 Le type associé servant à la définition des valeurs et au sous-typage est le suivant (les commentaires ont force de norme):

```
SEQUENCE {
      mantissa    INTEGER,
      base  INTEGER (2|10),
      exponent    INTEGER
-- Le nombre mathématique réel associé est égal à "mantissa" fois la
-- "base" élévé à la puissance "exponent"
}
```

NOTE 1 – Les valeurs différentes de zéro qui sont représentées en "base" 2 ou en "base" 10 sont considérées comme des valeurs abstraites distinctes même si elles correspondent à la même valeur de nombre réel. Elles peuvent exprimer des sémantiques d'application différentes.

NOTE 2 – La notation **REAL (WITH COMPONENTS { ... , base (10)})** peut être utilisée pour restreindre l'ensemble des valeurs aux valeurs abstraites en base 10 (ou de manière similaire aux valeurs abstraites en base 2).

NOTE 3 – Ce type peut véhiculer une représentation finie exacte de tout nombre pouvant être enregistré sur un matériel standard en virgule flottante, ainsi que tout nombre à représentation en caractères décimaux finie.

20.6 La valeur d'un type réel est définie par la notation "RealValue" ou, lorsqu'elle est utilisée dans une valeur "XMLValue", par la notation "XMLRealValue":

```
RealValue ::=
        NumericRealValue
    |   SpecialRealValue

NumericRealValue ::=
        realnumber
    |   "-" realnumber
    |   SequenceValue      -- Valeur du type séquence associé

SpecialRealValue ::=
        PLUS-INFINITY
    |   MINUS-INFINITY
```

Les deuxième et troisième formes de la notation "NumericRealValue" ne doivent pas être utilisées pour les valeurs nulles.

```
XMLRealValue ::=
        XMLNumericRealValue | XMLSpecialRealValue

XMLNumericRealValue ::=
        realnumber
    |   "-" realnumber
```

La deuxième forme de la notation "XMLNumericRealValue" ne doit pas être utilisée pour les valeurs nulles.

```
XMLSpecialRealValue ::=
        "<" & PLUS-INFINITY "/>" | "<" & MINUS-INFINITY "/>"
```

20.7 Lorsque la notation "realnumber" est utilisée, elle désigne la valeur abstraite correspondante en "base" 10. Si le type réel "RealType" est contraint à la "base" 2, la notation "realnumber" désigne la valeur abstraite en "base" 2 qui correspond soit à la valeur décimale spécifiée par "realnumber" soit à une précision définie localement si une représentation exacte n'est pas possible.

21 Notation du type chaîne binaire (bitstring type)

21.1 Le type chaîne binaire (voir § 3.6.6) est déclaré par la notation "BitStringType":

BitStringType ::=
    BIT STRING
  | **BIT STRING "{" NamedBitList "}"**

NamedBitList ::=
    NamedBit
  | **NamedBitList "," NamedBit**

NamedBit ::=
    identifier "(" number ")"
  | **identifier "(" DefinedValue ")"**

21.2 Dans une chaîne binaire, le premier bit est le bit de début et le dernier bit est appelé le bit de fin.

NOTE – Cette terminologie est utilisée pour spécifier la notation des valeurs et définir les règles de codage.

21.3 La valeur "DefinedValue" fait référence à une valeur non négative de type entier.

21.4 Les valeurs "number" ou "DefinedValue" de la liste "NamedBitList" seront toutes différentes, et correspondront chacune au numéro d'un bit distinct dans une chaîne binaire. Le bit de début de la chaîne binaire est identifiée par le numéro "number" zéro, les bits successifs ayant des valeurs successives.

21.5 Les identificateurs "identifier" apparaissant dans la liste "NamedBitList" seront tous différents.

NOTE 1 – L'ordre des productions "NamedBit" dans la liste "NamedBitList" n'est pas significatif.

NOTE 2 – Etant donné qu'un identificateur "identifier" de la liste "NamedBitList" ne peut pas servir à spécifier la valeur associée à un bit nommé "NamedBit", la valeur "DefinedValue" ne peut jamais être interprétée comme étant une valeur d'entier "IntegerValue". Par conséquent, dans la production suivante:

```
a  INTEGER ::= 1
T1 ::= INTEGER { a(2) }
T2 ::= BIT STRING { a(3), b(a) }
```

la dernière occurrence de **a** représente la valeur 1, puisqu'elle ne peut correspondre ni à la deuxième ni à la troisième occurrence de **a**.

21.6 La présence d'une liste "NamedBitList" n'a aucun effet sur l'ensemble des valeurs abstraites du type déclaré: les valeurs pourront comporter des bits de valeur 1 autres que les bits nommés.

21.7 Lorsqu'une liste de bits nommés "NamedBitList" est utilisée dans la définition d'un type de chaîne binaire, les règles de codage ASN.1 peuvent ajouter (ou supprimer) un nombre arbitraire de bits de fin égaux à 0 aux valeurs en cours de codage ou de décodage. Les concepteurs d'applications doivent donc s'assurer que des valeurs ne différant que par le nombre de "0" en bout de chaîne ne sont pas sémantiquement différentes.

21.8 L'étiquette de ce type est le numéro 3 de la classe universelle.

21.9 La valeur d'un type chaîne binaire "bitstring" est déclarée par la notation "BitStringValue" ou, lorsqu'elle est utilisée comme une valeur "XMLValue", par la notation "XMLBitStringValue". Ces productions sont:

BitStringValue ::=
    bstring
  | **hstring**
  | **"{" IdentifierList "}"**
  | **"{" "}"**
  | **CONTAINING Value**

IdentifierList ::=
    identifier
  | **IdentifierList "," identifier**

XMLBitStringValue ::=
    XMLTypedValue
  | **Xmlbstring**
  | **XMLIdentifierList**
  | **empty**

XMLIdentifierList ::=
"<" & identifier "/>"
| **XMLIdentifierList "<" & identifier "/>"**

21.10 La forme "XMLTypedValue" n'est utilisée que si la chaîne binaire a une contrainte de contenu qui inclut un type ASN.1 et qui n'inclut pas **ENCODED BY**. Si cette forme est utilisée, la valeur "XMLTypedValue" est une valeur du type ASN.1 figurant dans la contrainte de contenu.

21.11 La forme "XMLIdentifierList" n'est utilisée que si la chaîne binaire a une liste "NamedBitList".

21.12 Chaque identificateur "identifier" d'une valeur "BitStringValue" ou d'une valeur "XMLBitStringValue" sera identique à l'identificateur de la production "BitStringType" auquel cette valeur est associée.

21.13 La forme "empty" désigne une chaîne binaire sans aucun bit.

21.14 Si la chaîne binaire possède des bits nommés, la notation "BitStringValue" ou "XMLBitStringValue" représente une valeur de chaîne binaire dont les bits de numéros correspondant aux identificateurs "identifier" contiennent des "1", tous les autres bits étant à "0".

NOTE – Pour un type "BitStringType" qui a une liste "NamedBitList", la séquence de productions "**{**" "**}**" dans une valeur "BitStringValue" et la valeur "empty" dans une valeur "XMLBitStringValue" servent à représenter une chaîne binaire ne contenant pas de "1".

21.15 Dans la notation "bstring" ou "xmlbstring", le bit de début de la valeur de la chaîne binaire est à gauche et le bit de fin de la valeur de la chaîne binaire est à droite.

21.16 Dans la notation "hstring", le bit le plus significatif de chaque chiffre hexadécimal est celui de gauche.

NOTE – Cette notation n'impose aucune contrainte quant à la manière dont les règles de codage convertissent une chaîne binaire en octets pour le transfert.

21.17 La notation "hstring" ne sera utilisée que si la valeur de chaîne binaire comprend un nombre de bits multiple de quatre.

EXEMPLE

'A98A'H

et

'1010100110001010'B

sont les deux notations possibles d'une même valeur de chaîne binaire. Si le type a été défini à l'aide d'une liste "NamedBitList", le bit de fin, qui est nul, ne fait pas partie de la valeur, dont la longueur est donc de 15 bits. Si le type a été défini sans liste "NamedBitList", le bit de fin nul fait partie de la valeur, qui compte alors 16 bits.

21.18 La forme **CONTAINING** ne peut être utilisée que s'il existe une contrainte de contenu sur le type de chaîne binaire qui inclut **CONTAINING**. La valeur "Value" est alors la notation d'une valeur du "Type" figurant dans la contrainte "ContentsConstraint" (Rec. UIT-T X.682 | ISO/CEI 8824-3, § 11).

NOTE – Cette notation de valeur n'apparaît jamais dans une contrainte de sous-typage car, d'après la Rec. UIT-T X.682 | ISO/CEI 8824-3, § 11.3, toute autre contrainte après une contrainte "ContentsConstraint" est interdite et d'après le texte ci-dessus, l'utilisation de cette notation est interdite sauf si le gouverneur a une contrainte "ContentsConstraint".

21.19 La forme **CONTAINING** doit être utilisée s'il existe une contrainte de contenu sur le type de chaîne binaire qui ne contient pas **ENCODED BY**.

22 Notation du type chaîne d'octets (octetstring type)

22.1 Le type chaîne d'octets (voir § 3.6.49) est déclaré par la notation "OctetStringType":

OctetStringType ::= OCTET STRING

22.2 L'étiquette de ce type est le numéro 4 de la classe universelle.

22.3 Une valeur d'un type chaîne d'octets est déclarée par la notation "OctetStringValue" ou, lorsqu'elle est utilisée comme une valeur "XMLValue", par la notation "XMLOctetStringValue". Ces productions sont:

OctetStringValue ::=
bstring
| **hstring**
| **CONTAINING Value**

```
XMLOctetStringValue ::=
        XMLTypedValue
    |   xmlhstring
```

22.4 La forme "XMLTypedValue" n'est utilisée que si la chaîne d'octets a une contrainte de contenu qui inclut un type ASN.1 et qui n'inclut pas **ENCODED BY**. Si cette forme est utilisée, la valeur "XMLTypedValue" est une valeur du type ASN.1 figurant dans la contrainte de contenu.

22.5 Lors de la spécification des règles de codage d'une chaîne d'octets, les octets extrêmes sont désignés par les expressions premier octet et octet de fin, et les bits extrêmes d'un octet sont désignés par les expressions bit de plus fort poids et bit de plus faible poids.

22.6 Dans la notation "bstring", le bit le plus à gauche de cette notation est le bit de plus fort poids du premier octet de la valeur de la chaîne d'octets. Si la chaîne "bstring" ne contient pas un nombre entier d'octets, elle est interprétée comme se terminant par des bits complémentaires tous nuls, la complétant au prochain nombre entier d'octets.

22.7 Dans la notation "hstring" ou "xmlstring", le chiffre hexadécimal le plus à gauche est le demi-octet le plus significatif du premier octet.

22.8 Si le nombre de chiffres hexadécimaux d'une chaîne "hstring" est impair, cette chaîne est interprétée comme si elle se terminait par un seul chiffre hexadécimal complémentaire nul. Le nombre de chiffres hexadécimaux d'une chaîne "xmlhstring" ne doit pas être impair.

22.9 La forme **CONTAINING** ne peut être utilisée que s'il existe une contrainte de contenu sur le type de chaîne d'octets qui inclut **CONTAINING**. La valeur "Value" est alors la notation d'une valeur du "Type" figurant dans la contrainte "ContentsConstraint" (voir la Rec. UIT-T X.682 | ISO/CEI 8824-3, § 11).

NOTE – Cette notation de valeur n'apparaît jamais dans une contrainte de sous-typage car, d'après la Rec. UIT-T X.682 | ISO/CEI 8824-3, § 11.3, toute autre contrainte après une contrainte "ContentsConstraint" est interdite et d'après le texte ci-dessus, l'utilisation de cette notation est interdite sauf si le gouverneur a une contrainte "ContentsConstraint".

22.10 La forme **CONTAINING** doit être utilisée s'il existe une contrainte de contenu sur le type de chaîne d'octets qui ne contient pas **ENCODED BY**.

23 Notation du type néant (null type)

23.1 Le type néant (voir § 3.6.44) est déclaré par la notation "NullType":

```
NullType ::= NULL
```

23.2 L'étiquette de ce type est le numéro 5 de la classe universelle.

23.3 Une valeur du type néant est déclarée par la notation "NullValue" ou, lorsqu'elle est utilisée comme une valeur "XMLValue", par la notation "XMLNullValue". Ces productions sont:

```
NullValue ::= NULL

XMLNullValue ::= empty
```

24 Notation des types séquence (sequence types)

24.1 Un type séquence (voir § 3.6.60) est déclaré par la notation "SequenceType":

```
SequenceType ::=
        SEQUENCE "{" "}"
    |   SEQUENCE "{" ExtensionAndException OptionalExtensionMarker "}"
    |   SEQUENCE "{" ComponentTypeLists "}"

ExtensionAndException ::= "..." | "..." ExceptionSpec

OptionalExtensionMarker ::= "," "..." | empty

ComponentTypeLists ::=
        RootComponentTypeList
    |   RootComponentTypeList "," ExtensionAndException ExtensionAdditions
            OptionalExtensionMarker
    |   RootComponentTypeList "," ExtensionAndException ExtensionAdditions
            ExtensionEndMarker "," RootComponentTypeList
```

```
        |   ExtensionAndException ExtensionAdditions ExtensionEndMarker ","
              RootComponentTypeList
        |   ExtensionAndException ExtensionAdditions OptionalExtensionMarker

RootComponentTypeList ::= ComponentTypeList

ExtensionEndMarker ::= "," "..."

ExtensionAdditions ::=
            "," ExtensionAdditionList
        |   empty

ExtensionAdditionList ::=
            ExtensionAddition
        |   ExtensionAdditionList "," ExtensionAddition

ExtensionAddition ::=
            ComponentType
        |   ExtensionAdditionGroup

ExtensionAdditionGroup ::= "[[" VersionNumber ComponentTypeList "]]"

VersionNumber ::= empty | number ":"

ComponentTypeList ::=
            ComponentType
        |   ComponentTypeList "," ComponentType

ComponentType ::=
        NamedType
        |   NamedType OPTIONAL
        |   NamedType DEFAULT Value
        |   COMPONENTS OF Type
```

24.2 Lorsque la production "ComponentTypeList" apparaît à l'intérieur de la définition d'un module pour lequel a été choisi l'étiquetage automatique (voir § 12.3), et qu'aucune des occurrences de "NamedType" correspondant aux trois premières formes possibles de déclaration de "ComponentType" ne contient de type étiqueté "TaggedType", alors la transformation d'étiquetage automatique est adoptée pour l'ensemble de la liste des composants "ComponentTypeLists"; sinon, l'étiquetage automatique n'est pas appliqué.

NOTE 1 – L'utilisation de la notation de type étiqueté dans la définition de la liste des composants d'un type séquence permet à l'auteur de la spécification de garder le contrôle de l'étiquetage, contrairement à ce qui se passe avec l'affectation par un mécanisme d'étiquetage automatique. Ainsi, dans le cas suivant:

```
T ::= SEQUENCE { a INTEGER, b [1] BOOLEAN, c OCTET STRING }
```

aucun étiquetage automatique n'est appliqué à la liste des composants **a**, **b**, **c**, même si cette définition du type de séquence **T** apparaît à l'intérieur d'un module pour lequel l'étiquetage automatique a été sélectionné.

NOTE 2 – Seules les occurrences de productions "ComponentTypeList" apparaissant à l'intérieur d'un module pour lequel l'étiquetage automatique a été choisi sont des candidats possibles pour une transformation par étiquetage automatique.

24.3 La décision d'appliquer l'étiquetage automatique est prise individuellement pour chaque occurrence de la liste "ComponentTypeLists" *avant* la transformation **COMPONENTS OF** spécifiée au § 24.4. Toutefois, comme l'indiquent les § 24.7 à 24.9, l'étiquetage automatique, s'il s'applique, est appliqué *après* la transformation **COMPONENTS OF**.

NOTE – L'effet de cette disposition est que l'étiquetage automatique n'a pas lieu là où des étiquettes explicites sont présentes dans la liste "ComponentTypeLists", mais qu'il a lieu même si des étiquettes sont présentes dans le type déclaré à la suite d'une transformation **COMPONENTS OF**.

24.4 Le "Type" dans la notation "**COMPONENTS OF** Type" est un type séquence. La notation "**COMPONENTS OF** Type" est utilisée pour définir l'inclusion, à cet emplacement dans la liste des composants, de tous les types composants contenus dans le type indiqué, à l'exception de tout marqueur d'extension et de tout ajout d'extension pouvant figurer dans le "Type". (Seul l'élément "RootComponentTypeList" du "Type" indiqué par la notation "**COMPONENTS OF** Type" est inclus; les marqueurs d'extension et les ajouts d'extension éventuels sont ignorés par la notation "**COMPONENTS OF** Type".) Toute contrainte de sous-typage appliquée au type référencé n'est pas prise en considération par cette transformation.

NOTE – Cette transformation est logiquement effectuée avant l'exécution des dispositions des paragraphes suivants.

24.5 Chacun des paragraphes suivants identifie une série d'occurrences de productions "ComponentType" dans la racine, dans les ajouts d'extension ou dans les deux. La règle 24.5.1 s'appliquera pour toute série de cette sorte.

24.5.1 Lorsqu'il existe une ou plusieurs occurrences consécutives de la production "ComponentType" qui sont marquées comme **OPTIONAL** ou **DEFAULT**, leurs étiquettes et les étiquettes de tout type de composant immédiatement consécutif dans la série doivent être distinctes (voir le § 30). La prescription d'étiquettes distinctes ne s'applique qu'une fois l'étiquetage automatique effectué et sera toujours satisfaite lorsque cet étiquetage aura été appliqué.

24.5.2 Le paragraphe 24.5.1 s'applique aux séries de productions "ComponentType" dans la racine.

24.5.3 Le paragraphe 24.5.1 s'applique aux séries complètes de productions "ComponentType" dans la racine ou dans les ajouts d'extension, dans l'ordre textuel de leur apparition au sein de la définition du type (en ignorant toute notation de crochet de version et de points de suspension) (voir aussi § 48.7).

24.6 Lorsque la troisième ou la quatrième valeur de la production "ComponentTypeLists" est utilisée, tous les composants "ComponentType" dans les ajouts "ExtensionAddition" auront des étiquettes qui différeront des étiquettes de tous les composants "ComponentType" suivants dans l'ordre textuel jusques et y compris le premier composant "ComponentType" de cette sorte qui n'est pas marqué comme **OPTIONAL** ou **DEFAULT** dans la dernière liste "RootComponentTypeList" (s'il en existe) (Voir aussi § 48.7).

24.7 L'étiquetage automatique d'une occurrence de la liste "ComponentTypeLists" est logiquement effectué *après* la transformation spécifiée au § 24.4, mais seulement si le § 24.2 établit qu'il s'applique bien à cette occurrence. L'étiquetage automatique agit sur chaque composant "ComponentType" de la liste "ComponentTypeLists" en remplaçant le "Type" déclaré à l'origine dans la production "NamedType" par une occurrence "TaggedType" telle que celle-ci est spécifiée au § 24.9.

24.8 Si l'étiquetage automatique est en vigueur et que les composants "ComponentType" de la racine d'extension n'aient pas d'étiquettes, alors aucun composant "ComponentType" de la liste "ExtensionAdditionList" ne pourra être un type étiqueté "TaggedType".

24.9 Si l'étiquetage automatique est en service, le type étiqueté de remplacement "TaggedType" est spécifié comme suit:

a) la notation "TaggedType" de remplacement utilise la forme de production "Tag Type";

b) la classe "Class" du type de remplacement "TaggedType" est vide (c'est-à-dire que l'étiquetage est propre au contexte);

c) le numéro dans la classe "ClassNumber" du type de remplacement "TaggedType" est nul pour le premier composant "ComponentType" dans la liste "RootComponentTypeList"; il est égal à un pour le deuxième composant et ainsi de suite avec des numéros d'étiquettes croissants;

d) le numéro dans la classe "ClassNumber" du type de remplacement "TaggedType" est nul pour le premier composant "ComponentType" présent dans la liste "ExtensionAdditionList" si la liste "RootComponentTypeList" est absente; dans le cas contraire, il est supérieur d'une unité au plus grand numéro dans la classe "ClassNumber" des composants de la liste "RootComponentTypeList" et le composant "ComponentType" suivant dans la liste "ExtensionAdditionList" possède un numéro dans la classe "ClassNumber" supérieur d'une unité à celui du premier et ainsi de suite avec des numéros d'étiquettes croissants;

e) le "Type" du type de remplacement "TaggedType" est le même que le "Type" d'origine.

NOTE 1 – Le paragraphe 30.6 indique les règles qui régissent la spécification de l'étiquetage implicite ou explicite pour les types étiquetés de remplacement "TaggedType". L'étiquetage automatique est toujours implicite à moins que le "Type" soit une notation de type choix ou de type ouvert, ou une référence muette "DummyReference" (voir la Rec. UIT-T X.683 | ISO/CEI 8824-4, § 8.3), auquel cas l'étiquetage est explicite.

NOTE 2 – Une fois le § 24.7 appliqué, les étiquettes des différents composants sont complètement déterminées et ne sont plus modifiées même lorsque le type séquence est utilisé dans la définition d'un composant dans une autre liste "ComponentTypeLists" à laquelle l'étiquetage automatique s'applique. Ainsi, dans le cas suivant:

```
T ::= SEQUENCE { a   Ta,  b   Tb,  c   Tc }
E ::= SEQUENCE { f1  E1,  f2  T,   f3  E3 }
```

l'étiquetage automatique, qui est appliqué aux composants de **E**, n'affectera jamais les étiquettes attachées à **a**, **b** et **c** de **T**, quel que soit l'environnement d'étiquetage de **T**. Si **T** est défini dans un environnement d'étiquetage automatique et si **E** ne l'est pas, l'étiquetage automatique s'appliquera toujours aux composants **a**, **b** et **c** de **T**.

NOTE 3 – Lorsqu'un type séquence est utilisé comme "Type" dans une notation "**COMPONENTS OF** Type", chaque composant "ComponentType" de ce type séquence est dupliqué par application du 24.4 avant l'étiquetage automatique éventuel du type séquence qui y fait référence. Ainsi, dans le cas suivant:

```
T ::= SEQUENCE { a  Ta,  b  SEQUENCE { b1  T1,  b2  T2,  b3  T3},  c Tc }
W ::= SEQUENCE { x  Wx,  COMPONENTS OF  T, y  Wy }
```

les étiquettes de **a**, **b** et **c** dans l'expression de **T** ne sont pas nécessairement identiques aux étiquettes de **a**, **b** et **c** dans l'expression de **W** si **W** a été défini dans un environnement d'étiquetage automatique, mais les étiquettes de **b1**, **b2** et **b3** sont les

mêmes dans **T** et **W**. En d'autres termes, l'étiquetage automatique n'est appliqué qu'une seule fois à une liste "ComponentTypeList" donnée.

NOTE 4 – Le sous-typage n'a aucun impact sur l'étiquetage automatique.

NOTE 5 – En étiquetage automatique, l'insertion de nouveaux composants à n'importe quel endroit autre que le point d'insertion d'extension (voir § 3.6.29) peut induire des modifications des autres composants par suite d'effets collatéraux de la modification des étiquettes, causant ainsi des problèmes d'interfonctionnement avec une version antérieure de la présente spécification.

24.10 Si une production comporte une déclaration **OPTIONAL** ou **DEFAULT**, la valeur correspondante peut être omise dans la valeur du nouveau type.

24.11 Si une production comporte une déclaration **DEFAULT**, l'omission de la valeur correspondante est équivalente à l'insertion de la valeur définie par "Value", qui doit être la notation d'une valeur du type défini par "Type" dans la séquence de production "NamedType".

24.12 La valeur correspondant à un groupe "ExtensionAdditionGroup" (avec tous ses composants) est optionnelle. Si toutefois une telle valeur est présente, la valeur de tout composant de la liste entre crochets "ComponentTypeList" qui n'est pas marqué **OPTIONAL** ou **DEFAULT** devra être présente.

24.13 Les identificateurs "identifier" des séquences de production "NamedType" de la liste "ComponentTypeList" (plus les identificateurs obtenus par le développement de **COMPONENTS OF**) seront tous distincts.

24.14 Une valeur pour un type donné d'ajout d'extension ne sera pas spécifiée, à moins qu'il n'existe des valeurs spécifiées pour tous les types d'ajout d'extension non marqués **OPTIONAL** ou **DEFAULT** qui sont situés logiquement entre le type de l'ajout d'extension et la racine d'extension.

NOTE 1 – Lorsque le type est passé de la racine d'extension (version 1) vers la version 2 puis vers la version 3 par l'insertion d'ajouts d'extension, la présence d'un codage de tout ajout appartenant à la version 3 nécessite un codage de tous les ajouts appartenant à la version 2 qui ne sont pas marqués **OPTIONAL** ou **DEFAULT**.

NOTE 2 – Les types "ComponentType" qui sont des ajouts d'extension mais qui ne sont pas contenus dans un groupe "ExtensionAdditionGroup" doivent toujours être codés comme s'ils n'étaient pas marqués **OPTIONAL** ou **DEFAULT**, sauf lorsque la valeur abstraite est relayée à partir d'un émetteur utilisant une version antérieure de la syntaxe abstraite dans laquelle le type "ComponentType" n'est pas défini.

NOTE 3 – L'utilisation de la production "ExtensionAdditionGroup" est recommandée parce que:

a) elle peut fournir des codages plus concis, en fonction des règles de codage (par exemple pour les règles PER);

b) la syntaxe est plus précise dans la mesure où elle indique clairement qu'une valeur d'un type défini dans la liste "ExtensionAdditionList" et qui n'est pas marqué **OPTIONAL** ou **DEFAULT** doit toujours figurer dans un codage si le groupe d'ajouts d'extension, dans lequel elle est définie, est lui-même codé (voir la Note 1);

c) la syntaxe indique clairement quels sont les types d'une liste "ExtensionAdditionList" qui doivent être pris en charge sous la forme d'un groupe par une application.

24.15 Un numéro "VersionNumber" n'est utilisé que si toutes les productions "ExtensionAdditions" et "ExtensionAdditionAlternatives" à l'intérieur du module font partie d'un groupe "ExtensionAdditionGroup" ou "ExtensionAdditionAlternativesGroup" avec le numéro "VersionNumber". Le numéro "number" de chaque production "VersionNumber" d'un groupe "ExtensionAdditionGroup" doit être supérieur ou égal à deux et doit être supérieur au numéro "number" de n'importe quel groupe "ExtensionAdditionGroup" précédent dans un point d'insertion.

NOTE 1 – La convention utilisée ici est que la spécification sans groupe d'ajouts d'extension correspond à la version 1. Par conséquent, le premier groupe d'ajouts d'extension inséré aura un numéro supérieur ou égal à 2. Lorsqu'un seul ajout "ExtensionAddition" est nécessaire, on peut utiliser un groupe "ExtensionAdditionGroup" avec un seul ajout "ExtensionAddition".

NOTE 2 – Les restrictions relatives à l'utilisation du numéro "VersionNumber" s'appliquent uniquement à l'intérieur d'un seul module et n'imposent aucune contrainte aux types importés.

24.16 L'étiquette de tous les types séquence est le numéro 16 de la classe universelle.

NOTE – Les types séquence-de portent la même étiquette que les types séquence (voir § 25.2).

24.17 Une valeur de type séquence est définie par la notation "SequenceValue" ou, lorsqu'elle est utilisée comme une valeur "XMLValue", par la notation "XMLSequenceValue". Ces productions sont:

```
SequenceValue ::=
        "{" ComponentValueList "}"
    |   "{" "}"

ComponentValueList ::=
        NamedValue
    |   ComponentValueList "," NamedValue
```

XMLSequenceValue ::=
    XMLComponentValueList
  | **empty**

XMLComponentValueList ::=
    XMLNamedValue
  | **XMLComponentValueList XMLNamedValue**

24.18 La notation "**{**" "**}**" ou "empty" n'est utilisée que si:

a) tous les types "ComponentType" de la production "SequenceType" comportent la déclaration **DEFAULT** ou **OPTIONAL**, et toutes les valeurs sont omises;

b) la notation du type est **SEQUENCE{}**.

24.19 Une valeur "NamedValue" ou "XMLNamedValue" figure pour chaque type "NamedType" du type "SequenceType" ne comportant pas la déclaration **OPTIONAL** ou **DEFAULT**, et ces valeurs apparaissent dans le même ordre que les types "NamedType" correspondants.

25 Notation des types séquence-de (sequence-of types)

25.1 Un type séquence-de (voir § 3.6.61) est défini à partir d'un autre type par la notation "SequenceOfType":

SequenceOfType ::= SEQUENCE OF Type | SEQUENCE OF NamedType

NOTE – Si une lettre initiale en majuscule est nécessaire pour un nom d'étiquette XML utilisé dans la notation de valeur XML pour le type "SequenceOfType", il convient d'utiliser la première forme. (Le nom d'étiquette XML est ensuite formé à partir du nom du type "Type".)

25.2 L'étiquette de tous les types séquence-de est le numéro 16 de la classe universelle.

NOTE – Les types séquence portent la même étiquette que les types séquence-de (voir § 24.16).

25.3 Une valeur de type séquence-de est déclarée par la notation "SequenceOfValue" ou, lorsqu'elle est utilisée comme une valeur "XMLValue", par la notation "XMLSequenceOfValue". Ces productions sont:

SequenceOfValue ::=
    "{" ValueList "}"
  | **"{" NamedValueList "}"**
  | **"{" "}"**

ValueList ::=
    Value
  | **ValueList "," Value**

NamedValueList ::=
    NamedValue
  | **NamedValueList "," NamedValue**

XMLSequenceOfValue ::=
    XMLValueList
  | **XMLDelimitedItemList**
  | **XMLSpaceSeparatedList**
  | **empty**

XMLValueList ::=
    XMLValueOrEmpty
  | **XMLValueOrEmpty XMLValueList**

XMLValueOrEmpty ::=
    XMLValue
  | **"<" & NonParameterizedTypeName "/>"**

XMLSpaceSeparatedList ::=
    XMLValueOrEmpty
  | **XMLValueOrEmpty " " XMLSpaceSeparatedList**

XMLDelimitedItemList ::=
    XMLDelimitedItem
  | **XMLDelimitedItem XMLDelimitedItemList**

```
XMLDelimitedItem ::=
        "<" & NonParameterizedTypeName ">" XMLValue
            "</" & NonParameterizedTypeName ">"
    |   "<" & identifier ">" XMLValue "</" & identifier ">"
```

La notation "**{**" "**}**" ou "empty" est utilisée quand la valeur "SequenceOfValue" ou "XMLSequenceOfValue" est une liste vide.

NOTE 1 – L'ordre de ces valeurs peut avoir une signification sémantique.

NOTE 2 – La production "XMLSpaceSeparatedList" n'est pas utilisée dans la présente Recommandation | Norme internationale et elle n'est pas utilisée dans la notation de valeur XML. Elle est fournie afin de pouvoir spécifier l'utilisation de la liste "XMLSpaceSeparatedList" dans des codages des types "IntegerType", "RealType", "ObjectIdentifierType", "RelativeOIDType" ainsi que des types utiles **`GeneralizedTime`** et **`UTCTime`**. Il est également possible de spécifier l'utilisation de la liste "XMLValueList" au lieu de la liste "XMLDelimitedItemList" pour certaines instances de "**`SEQUENCE OF SEQUENCE`**" et "**`SEQUENCE OF SET`**".

25.4 Si la valeur "XMLValue" pour le composant est "empty", la deuxième forme de "XMLValueOrEmpty" est choisie pour représenter cette valeur du composant.

25.5 Les productions "XMLValueList" et "XMLDelimitedItemList" sont utilisées conformément à la colonne 2 du Tableau 5, où le type "Type" du composant est donné dans la colonne 1.

Tableau 5 – Notation "XMLSequenceOfValue" et "XMLSetOfValue" pour les types ASN.1

Type ASN.1	Notation de valeur XML
BitStringType	XMLDelimitedItemList
BooleanType	XMLValueList
CharacterStringType	XMLDelimitedItemList
ChoiceType	XMLValueList
ConstrainedType	*Voir 25.7*
DefinedType	*Voir 25.9*
EmbeddedPDVType	XMLDelimitedItemList
EnumeratedType	XMLValueList
ExternalType	XMLDelimitedItemList
InstanceOfType	*Voir Rec. UIT-T X.681 | ISO/CEI 8824-2, § C.9*
IntegerType	XMLDelimitedItemList
NullType	XMLValueList
ObjectClassFieldType	*Voir Rec. UIT-T X.681 | ISO/CEI 8824-2, § 14.10 et 14.11*
ObjectIdentifierType	XMLDelimitedItemList
OctetStringType	XMLDelimitedItemList
RealType	XMLDelimitedItemList
RelativeOIDType	XMLDelimitedItemList
SelectionType	*Voir 25.8*
SequenceType	XMLDelimitedItemList
SequenceOfType	XMLDelimitedItemList
SetType	XMLDelimitedItemList
SetOfType	XMLDelimitedItemList
TaggedType	*Voir 25.6*
UsefulType (GeneralizedTime)	XMLDelimitedItemList
UsefulType (UTCTime)	XMLDelimitedItemList
UsefulType (ObjectDescriptor)	XMLDelimitedItemList
TypeFromObject	*Voir Rec. UIT-T X.681 | ISO/CEI 8824-2, § 15.6*
ValueSetFromObjects	*Voir Rec. UIT-T X.681 | ISO/CEI 8824-2, § 15.6*

25.6 Si le type "Type" du composant est un type "TaggedType", le type qui détermine la notation "XMLSequenceOfValue" est le type "Type" du type "TaggedType" (voir § 30.1). S'il est lui-même un type "TaggedType", le présent § 25.6 est appliqué de façon récursive.

25.7 Si le type "Type" du composant est un type "ConstrainedType", le type qui détermine la notation "XMLSequenceOfValue" est le type "Type" du type "ConstrainedType" (voir § 45.1). S'il est lui-même un type "ConstrainedType", le présent § 25.7 est appliqué de façon récursive.

25.8 Si le type "Type" du composant est un type "SelectionType", le type qui détermine la notation "XMLSequenceOfValue" est le type désigné par le type "SelectionType" (voir § 29).

25.9 Si le type "Type" du composant est un type "DefinedType", le type qui détermine la notation "XMLSequenceOfValue" est le type désigné par le type "DefinedType" (voir § 13.1).

25.10 La deuxième forme de "XMLDelimitedItem" est utilisée si et seulement si le type "SequenceOfType" contient un identificateur "identifier", et l'identificateur "identifier" de "XMLDelimitedItem" doit être cet identificateur "identifier".

25.11 Si la première forme de "XMLDelimitedItem" est utilisée et si le composant du type séquence-de (après avoir ignoré les éventuelles étiquettes) est une référence "typereference" ou "ExternalTypeReference", le nom "NonParameterizedTypeName" est cette référence "typereference" ou "ExternalTypeReference", sinon c'est le nom "xmlasn1typename" spécifié dans le Tableau 4 correspondant au type prédéfini du composant.

25.12 Si la première forme de type "SequenceOfType" est utilisée, la première forme de "SequenceOfValue" doit être utilisée. Chaque valeur "Value" de la liste "ValueList" de "SequenceOfValue" est du type spécifié dans "SequenceOfType", de même que chaque valeur "XMLValue" des différentes formes de "XMLSequenceOfValue".

25.13 Si la deuxième forme de type "SequenceOfType" est utilisée, la deuxième forme de "SequenceOfValue" doit être utilisée. Chaque valeur "NamedValue" de la liste "NamedValueList" doit contenir une valeur "Value" du type spécifié dans le type "NamedType" de "SequenceOfType". L'identificateur "identifier" des valeurs "NamedValue" est l'identificateur "identifier" du type "NamedType" de "SequenceOfType".

26 Notation des types ensemble (set types)

26.1 Un type ensemble (voir § 3.6.64) est défini à partir d'autres types par la notation "SetType":

```
SetType ::=
        SET "{" "}"
    |   SET "{" ExtensionAndException OptionalExtensionMarker "}"
    |   SET "{" ComponentTypeLists "}"
```

"ComponentTypeLists", "ExtensionAndException" et "OptionalExtensionMarker" sont spécifiés au § 24.1.

26.2 Le "Type" dans la notation "**COMPONENTS OF** Type" est un type ensemble. La notation "**COMPONENTS OF** Type" est utilisée pour définir l'inclusion, au niveau de ce point dans la liste de composants, de tous les types composants contenus dans le type indiqué, à l'exception de tout marqueur d'extension et d'ajout d'extension pouvant figurer dans le "Type". (Seul l'élément "RootComponentTypeList" du "Type" indiqué par la notation "**COMPONENTS OF** Type" est inclus, les marqueurs d'extension et les ajouts d'extension éventuels sont ignorés par la notation "**COMPONENTS OF** Type".) Toute contrainte de sous-typage appliquée au type référencé n'est pas prise en considération par cette transformation.

NOTE – Cette transformation doit être logiquement effectuée avant d'appliquer les dispositions des paragraphes suivants.

26.3 Les types "ComponentType" d'un type ensemble porteront tous des étiquettes différentes (voir le § 30). Chaque nouveau type "ComponentType" inséré dans les ajouts "ExtensionAdditions" portera une étiquette canoniquement plus grande (voir § 8.6) que celles des autres composants figurant dans les ajouts "ExtensionAdditions".

NOTE – Lorsque l'étiquetage par défaut "TagDefault" du module dans lequel cette notation apparaît est **AUTOMATIC TAGS**, il résulte des dispositions du § 24.7 que l'étiquetage automatique est appliqué quels que soient les types "ComponentType" effectifs. (Voir aussi 48.7.)

26.4 Les paragraphes 24.2 et 24.7 à 24.13 s'appliquent également aux types ensemble.

26.5 L'étiquette de tous les types ensemble est le numéro 17 de la classe universelle.

NOTE – Les types ensemble-de portent la même étiquette que les types ensemble (voir § 27.2).

26.6 Dans un type ensemble, l'ordre des valeurs n'est porteur d'aucun contenu sémantique.

26.7 Une valeur de type ensemble est déclarée par la notation "SetValue" ou, lorsqu'elle est utilisée comme une valeur "XMLValue", par la notation "XMLSetValue". Ces productions sont:

```
SetValue ::=
        "{" ComponentValueList "}"
    |   "{" "}"

XMLSetValue ::=
        XMLComponentValueList
    |   empty
```

Les listes "ComponentValueList" et "XML ComponentValueList" sont spécifiées au § 24.17.

26.8 La valeur "SetValue" (respectivement "XMLSetValue") vaut "`{`" "`}`" (respectivement "empty") que si:

a) tous les types "ComponentType" de la production "SetType" comportent la déclaration **`DEFAULT`** ou **`OPTIONAL`**, et toutes les valeurs sont omises;

b) la notation du type est **`SET{}`**.

26.9 Une valeur "NamedValue" ou "XMLNamedValue" figure pour chaque type "NamedType" de "SetType" ne comportant pas la déclaration **`OPTIONAL`** ou **`DEFAULT`**.

NOTE – Ces valeurs "NamedValue" ou "XMLNamedValue" peuvent figurer dans n'importe quel ordre.

27 Notation des types ensemble-de (set-of types)

27.1 Un type ensemble-de (voir § 3.6.65) est défini à partir d'un autre type par la notation "SetOfType":

```
SetOfType ::=
        SET OF Type
    |   SET OF NamedType
```

NOTE – Si une lettre initiale en majuscule est nécessaire pour un nom d'étiquette XML utilisé dans la notation de valeur XML pour le type "SetOfType", il convient d'utiliser la première forme. (Le nom d'étiquette XML est ensuite formé à partir du nom du type "Type".)

27.2 L'étiquette de tous les types ensemble-de est le numéro 17 de la classe universelle.

NOTE – Les types ensemble portent la même étiquette que les types ensemble-de (voir § 26.5).

27.3 Une valeur de type ensemble-de est déclarée par la notation "SetOfValue" ou, lorsqu'elle est utilisée comme valeur "XMLValue", par la notation "XMLSetOfValue". Ces productions sont:

```
SetOfValue ::=
        "{" ValueList "}"
    |   "{" NamedValueList "}"
    |   "{" "}"

XMLSetOfValue ::=
        XMLValueList
    |   XMLDelimitedItemList
    |   XMLSpaceSeparatedList
    |   empty
```

La liste "ValueList", la liste "NamedValueList" et les différentes formes de "XMLSetOfValue"sont spécifiées au § 25.3. La notation "`{`" "`}`" ou "empty" est utilisée quand la valeur "SetOfValue" ou "XMLSetOfValue"est une liste vide.

NOTE 1 – L'ordre de ces valeurs ne doit pas avoir de signification sémantique.

NOTE 2 – Il n'est pas demandé aux règles de codage de préserver l'ordre de ces valeurs.

NOTE 3 – Le type ensemble-de n'est pas un ensemble de valeurs au sens mathématiques. Ainsi, pour un **`SET OF INTEGER`**, les valeurs `{ 1 }` et `{ 1 1 }` sont distinctes.

27.4 Si la première forme de "SetOfType" est utilisée, la première forme de "SetOfValue" doit être utilisée. Chaque valeur "Value" de la liste "ValueList" de "SetOfValue" est du type spécifié dans "SetOfType", de même que chaque valeur "XMLValue" des différentes formes de "XMLSetOfValue".

27.5 Si la deuxième forme de "SetOfType" est utilisée, la deuxième forme de "SetOfValue" doit être utilisée. Chaque valeur "NamedValue" de la liste "NamedValueList" doit contenir une valeur "Value" du type spécifié dans le type "NamedType" de "SetOfType". L'identificateur "identifier" des valeurs "NamedValue" est l'identificateur "identifier" du type "NamedType" de "SetOfType".

28 Notation des types choix (choice types)

28.1 Un type choix (voir § 3.6.13) est défini à partir d'autres types par la notation "ChoiceType":

```
ChoiceType ::= CHOICE "{" AlternativeTypeLists "}"

AlternativeTypeLists ::=
        RootAlternativeTypeList
    |   RootAlternativeTypeList "," ExtensionAndException ExtensionAdditionAlternatives
        OptionalExtensionMarker

RootAlternativeTypeList ::= AlternativeTypeList

ExtensionAdditionAlternatives ::=
        "," ExtensionAdditionAlternativesList
    |   empty

ExtensionAdditionAlternativesList ::=
        ExtensionAdditionAlternative
    |   ExtensionAdditionAlternativesList "," ExtensionAdditionAlternative

ExtensionAdditionAlternative ::=
        ExtensionAdditionAlternativesGroup
    |   NamedType

ExtensionAdditionAlternativesGroup ::=
        "[[" VersionNumber AlternativeTypeList "]]"

AlternativeTypeList ::=
        NamedType
    |   AlternativeTypeList "," NamedType
```

NOTE – "`T ::= CHOICE { a A }`" et **A** ne sont pas des types identiques, et peuvent être codés différemment par les règles de codage.

28.2 Lorsque la production "AlternativeTypeLists" apparaît dans la définition d'un module pour lequel l'étiquetage automatique a été choisi (voir § 12.3), et si aucune des occurrences "NamedType" dans n'importe quelle liste "AlternativeTypeList" ne contient un type étiqueté "TaggedType", alors l'étiquetage automatique est adopté pour l'ensemble de la liste "AlternativeTypeLists"; sinon, l'étiquetage automatique n'est pas appliqué.

28.3 Les types définis dans les productions "AlternativeTypeList" d'une production "AlternativeTypeLists" porteront des étiquettes distinctes (voir § 30 et 48.7). Si l'étiquetage automatique a été choisi, la spécification selon laquelle toutes les étiquettes doivent être distinctes ne s'appliquera qu'une fois l'étiquetage automatique effectué et sera toujours satisfaite.

28.4 Si l'étiquetage automatique est en vigueur et que les types nommés "Nametype" dans la racine d'extension n'ont pas d'étiquette alors aucun type "Nametype" de la liste "ExtensionAdditionAlternativeslist" ne sera un type étiqueté.

28.5 L'étiquetage automatique agit sur chaque production "NamedType" de la liste "AlternativeTypeLists" en remplaçant le "Type" déclaré à l'origine dans la production "NamedType" par un type étiqueté de remplacement "TaggedType". Ce dernier est spécifié de la façon suivante:

a) la notation "TaggedType" de remplacement utilise la forme de production "Tag Type";

b) la classe "Class" du type de remplacement "TaggedType" est vide (c'est-à-dire que l'étiquetage est propre au contexte);

c) le numéro dans la classe "ClassNumber" du type de remplacement "TaggedType" est nul pour la première forme "NamedType" dans la liste "RootAlternativeTypeList"; il est égal à un pour la deuxième forme et ainsi de suite avec des numéros d'étiquettes croissants;

d) le numéro dans la classe "ClassNumber" du type de remplacement "TaggedType" pour la première forme "NamedType" dans la liste "ExtensionAdditionAlternativesList" est supérieur d'une unité au plus grand numéro dans la classe "ClassNumber" de la liste "RootAlternativeTypeList", la forme "NamedType" suivante dans la liste "ExtensionAdditionAlternativesList" possédant un numéro dans la classe "ClassNumber" supérieur d'une unité à celui du premier et ainsi de suite avec des numéros d'étiquettes croissants;

e) le "Type" du type de remplacement "TaggedType" est le même que le "Type" d'origine.

NOTE 1 – Le paragraphe 30.6 indique les règles qui régissent la spécification de l'étiquetage implicite ou explicite pour les types étiquetés de remplacement "TaggedType". L'étiquetage automatique est toujours implicite à moins que le "Type" ne soit une notation de type choix non étiqueté ou de type ouvert non étiqueté, ou une référence muette "DummyReference" non étiquetée (voir la Rec. UIT-T X.683 | ISO/CEI 8824-4, § 8.3), auquel cas l'étiquetage est explicite.

NOTE 2 – Une fois l'étiquetage automatique appliqué, les étiquettes des différents composants sont complètement déterminées et ne sont plus modifiées même lorsque le type choix est utilisé dans la définition d'une forme dans une autre liste "AlternativeTypeLists" à laquelle l'étiquetage automatique s'applique. Ainsi, dans le cas suivant:

```
T ::= CHOICE { a Ta, b Tb, c Tc }
E ::= CHOICE {f1 E1, f2 T, f3 E3}
```

l'étiquetage automatique, qui est appliqué aux composants de **E**, n'affectera jamais les étiquettes attachées à **a**, **b** et **c** de **T**, quel que soit l'environnement d'étiquetage de **T**. Si **T** est défini dans un environnement d'étiquetage automatique et si **E** ne l'est pas, l'étiquetage automatique s'appliquera toujours aux composants **a**, **b** et **c** de **T**.

NOTE 3 – Le sous-typage n'a aucune incidence sur l'étiquetage automatique.

NOTE 4 – En étiquetage automatique, l'insertion de nouvelles formes à n'importe quel endroit autre que le point d'insertion d'extension (voir § 3.6.29) peut induire des modifications des autres formes par suite d'effets collatéraux de la modification des étiquettes, causant ainsi des problèmes d'interfonctionnement avec une version antérieure de la présente spécification.

28.6 Le numéro "VersionNumber" est défini au § 24.1 et les restrictions relatives à son utilisation cohérente dans la totalité d'un module qui sont spécifiées au § 24.15 s'appliquent à l'utilisation de numéros "number" dans cette production.

28.7 L'étiquette de chaque nouveau type "NamedType" ajouté à la liste "ExtensionAdditionAlternativesList" sera canoniquement supérieure (voir § 8.6) à celle des autres types de la liste "ExtensionAdditionAlternativesList" et sera celle du dernier type "NamedType" de la liste "ExtensionAdditionAlternativesList".

28.8 Le type choix contient des valeurs qui ne portent pas tous la même étiquette. (L'étiquette dépend du choix effectué pour définir la valeur du type choix.)

28.9 Lorsque le type choix ne possède pas de marqueur d'extension et qu'il est utilisé dans un cas où la présente Recommandation | Norme internationale impose aux types de porter des étiquettes distinctes (voir § 28.3), la disposition s'appliquera à toutes les étiquettes possibles des valeurs du type choix. Les exemples suivants, dans lesquels on suppose que l'étiquetage par défaut "TagDefault" n'est pas l'étiquetage automatique **AUTOMATIC TAGS**, illustrent cette prescription.

EXEMPLES

1
```
A ::= CHOICE {
b     B,
c     NULL}

B ::= CHOICE {
d     [0] NULL,
e     [1] NULL}
```

2
```
A ::= CHOICE {
b     B,
c     C}

B ::= CHOICE {
d     [0] NULL,
e     [1] NULL}

C ::= CHOICE {
f     [2] NULL,
g     [3] NULL}
```

3 (Incorrect)
```
      A ::= CHOICE {
b     B,
c     C}

      B ::= CHOICE {
d     [0] NULL,
e     [1] NULL}

      C ::= CHOICE {
f     [0] NULL,
g     [1] NULL}
```

Les exemples 1 et 2 correspondent à une utilisation correcte de la notation. L'exemple 3 n'est pas correct en l'absence d'étiquetage automatique, car les étiquettes des composants **d** et **f** d'une part, et **e** et **g** d'autre part, sont identiques.

28.10 Les identificateurs "identifier" de toutes les productions "NamedType" d'une même liste "AlternativeTypeLists" seront tous distincts.

28.11 Une valeur de type choix est déclarée par la notation "ChoiceValue" ou, lorsqu'elle est utilisée comme une valeur "XMLValue", par la notation "XMLChoiceValue". Ces productions sont:

ChoiceValue ::= identifier ":" Value

XMLChoiceValue ::= "<" & identifier ">" XMLValue "</" & identifier ">"

28.12 "Value" ou "XMLValue" est une notation de valeur associée au type de la liste "AlternativeTypeLists" qui est désigné par l'identificateur "identifier".

29 Notation des types sélection (selection types)

29.1 Un type sélection (voir § 3.6.59) est défini par la notation "SelectionType":

SelectionType ::= identifier "<" Type

où "Type" désigne un type choix, et où "identifier" est l'identificateur d'une production "NamedType" figurant dans la liste "AlternativeTypeLists" de ce type choix.

29.2 Lorsque "Type" désigne un type contraint, la sélection est effectuée sur le type parent, sans tenir compte de l'éventuelle contrainte de sous-typage imposée au type parent.

29.3 Lorsque le type "SelectionType" est utilisé comme type nommé "NamedType", l'identificateur "identifier" de "NamedType" est présent, de même que l'identificateur "identifier" de "SelectionType".

29.4 Lorsque le type "SelectionType" est utilisé comme type "Type", l'identificateur "identifier" est conservé et le type désigné est celui de la forme sélectionnée.

29.5 La notation de valeur associée à un type sélection est la notation de valeur associée au type désigné par le type "SelectionType".

30 Notation des types étiquetés (tagged types)

Un type étiqueté "TaggedType" (voir § 3.6.70) est un type nouveau, isomorphe d'un type existant, mais portant une étiquette différente. Le type étiqueté est principalement utilisé lorsque la présente Recommandation | Norme internationale nécessite l'emploi de types portant des étiquettes distinctes (voir les § 24.5 à 24.6, 26.3 et 28.3). L'adoption de l'étiquetage automatique **`AUTOMATIC TAGS`** comme étiquetage par défaut "TagDefault" dans un module permet d'accomplir un tel étiquetage sans que la notation de type étiqueté n'apparaisse explicitement dans le module.

NOTE – Lorsqu'un protocole détermine qu'à un certain moment des valeurs de différents types de donnée peuvent être transmises, des étiquettes distinctes peuvent s'avérer nécessaires pour permettre au destinataire de les décoder correctement.

30.1 Un type étiqueté est défini par la notation "TaggedType":

TaggedType ::=
Type
| **Tag `IMPLICIT` Type**
| **Tag `EXPLICIT` Type**

Tag ::= "[" Class ClassNumber "]"

ClassNumber ::=
number
| **DefinedValue**

Class ::=
`UNIVERSAL`
| **`APPLICATION`**
| **`PRIVATE`**
| **empty**

30.2 La référence "valuereference" de la valeur "DefinedValue" sera du type entier et se verra affecter une valeur non négative.

30.3 Le type nouveau est isomorphe de l'ancien, mais porte une étiquette de la classe "Class" et de numéro "ClassNumber", à moins que la classe ne soit vide (option "empty"), auquel cas la classe est propre au contexte et le numéro vaut "ClassNumber".

30.4 Seuls les types définis dans la présente Recommandation | Norme internationale peuvent porter une étiquette de la classe universelle **UNIVERSAL**.

NOTE 1 – L'utilisation d'étiquettes de la classe universelle fait l'objet d'accords périodiques entre l'ISO et l'UIT-T.

NOTE 2 – Le paragraphe E.2.12 comporte des conseils et des indications concernant l'utilisation des classes d'étiquettes.

30.5 L'étiquetage est toujours implicite ou explicite. L'étiquetage implicite indique, pour les règles de codage qui en offrent le choix, qu'il n'est pas nécessaire d'identifier explicitement en transfert l'étiquette d'origine du "Type" dans le type étiqueté.

NOTE – Il peut être utile de conserver l'ancienne étiquette si celle-ci est de la classe universelle et qu'elle identifie donc de façon non ambiguë l'ancien type sans connaître la définition en notation ASN.1 du nouveau type. Toutefois, l'utilisation de l'étiquetage **IMPLICIT** permet de minimiser le nombre d'octets à transférer. Un exemple de codage utilisant la déclaration d'étiquetage **IMPLICIT** est donné dans la Rec. UIT-T X.690 | ISO/CEI 8825-1.

30.6 La structure d'étiquetage spécifie un étiquetage explicite si l'une des conditions suivantes est vérifiée:

a) la forme "Tag **EXPLICIT** Type" est utilisée;

b) la forme "Tag Type" est utilisée et l'étiquetage par défaut "TagDefault" du module est explicite **EXPLICIT TAGS** ou vide "empty";

c) la forme "Tag Type" est utilisée et l'étiquetage par défaut "TagDefault" du module est implicite **IMPLICIT TAGS** ou automatique **AUTOMATIC TAGS**, mais le type défini par "Type" est un type choix non étiqueté, un type ouvert non étiqueté ou une référence muette "DummyReference" non étiquetée (voir la Rec. UIT-T X.683 | ISO/CEI 8824-4, § 8.3).

Dans tous les autres cas, la structure d'étiquetage spécifie un étiquetage implicite.

30.7 Si la classe "Class" est vide (option "empty"), l'utilisation de l'étiquette "Tag" n'est soumise à aucune autre restriction que l'obligation spécifiée par les § 24.5 à 24.6, 26.3 et 28.3 imposant d'avoir des étiquettes distinctes.

30.8 La déclaration d'étiquetage implicite **IMPLICIT** ne sera pas utilisée si le "Type" indiqué est un type choix non étiqueté, un type ouvert non étiqueté ou une référence muette "DummyReference" non étiquetée (voir la Rec. UIT-T X.683 | ISO/CEI 8824-4, § 8.3).

30.9 Une valeur de type "TaggedType" est déclarée par la notation "TaggedValue" ou, lorsqu'elle est utilisée comme une valeur "XMLValue", par la notation "XMLTaggedValue". Ces productions sont:

TaggedValue ::= Value

XMLTaggedValue ::= XMLValue

où "Value" ou "XMLValue" est une notation de valeur associée au type "Type" dans la notation "TaggedType".

NOTE – L'étiquette "Tag" n'apparaît pas dans cette notation.

31 Notation du type identificateur d'objet (object identifier type)

31.1 Le type identificateur d'objet (voir § 3.6.48) est déclaré par la notation "ObjectIdentifierType":

```
ObjectIdentifierType ::=
        OBJECT IDENTIFIER
```

31.2 L'étiquette de ce type est le numéro 6 de la classe universelle.

31.3 Une valeur d'identificateur d'objet est déclarée par la notation "ObjectIdentifierValue" ou, lorsqu'elle est utilisée comme une valeur "XMLValue", par la notation "XMLObjectIdentifierValue". Ces productions sont:

```
ObjectIdentifierValue ::=
        "{" ObjIdComponentsList "}"
    |   "{" DefinedValue ObjIdComponentsList "}"

ObjIdComponentsList ::=
        ObjIdComponents
    |   ObjIdComponents ObjIdComponentsList
```

```
ObjIdComponents ::=
        NameForm
    |   NumberForm
    |   NameAndNumberForm
    |   DefinedValue

NameForm ::= identifier

NumberForm ::= number | DefinedValue

NameAndNumberForm ::=
        identifier "(" NumberForm ")"

XMLObjectIdentifierValue ::=
        XMLObjIdComponentList

XMLObjIdComponentList ::=
        XMLObjIdComponent
    |   XMLObjIdComponent & "." & XMLObjIdComponentList

XMLObjIdComponent ::=
        NameForm
    |   XMLNumberForm
    |   XMLNameAndNumberForm

XMLNumberForm ::= number

XMLNameAndNumberForm ::=
        identifier & "(" & XMLNumberForm & ")"
```

31.4 La référence "valuereference" de la valeur "DefinedValue"de la notation "NumberForm" est du type entier et une valeur non négative lui est affectée.

31.5 La référence "valuereference" de la valeur "DefinedValue" de la notation "ObjectIdentifierValue" est du type identificateur d'objet.

31.6 La valeur "DefinedValue" de "ObjIdComponents" est du type identificateur d'objet relatif et identifie un ensemble ordonné d'arcs depuis un nœud de départ vers un nœud ultérieur dans l'arbre d'identificateurs d'objets. Le nœud de départ est identifié par les composants "ObjIdComponents" antérieurs, et les composants "ObjIdComponents" ultérieurs (s'il y en a) identifient les arcs à partir du nœud ultérieur. Le nœud de départ ne doit être ni la racine, ni un nœud situé immédiatement au-dessous de la racine.

NOTE – Une valeur d'identificateur d'objet relatif doit être associée à une valeur d'identificateur d'objet spécifique afin qu'un objet puisse être identifié sans ambiguïté. Les valeurs d'identificateurs d'objets doivent (voir § 31.10) contenir au moins deux composants. C'est pour cela qu'il y a une restriction sur le nœud de départ.

31.7 La forme "NameForm" ne sera utilisée que pour les composants d'identificateur d'objet dont la valeur numérique et l'identificateur sont spécifiés dans la Rec. UIT-T X.660 | ISO/CEI 9834-1, Annexes A à C (voir aussi l'Annexe D de la présente Recommandation | Norme internationale) et correspondra à l'un des identificateurs spécifiés dans la Rec. UIT-T X.660 | ISO/CEI 9834-1, Annexes A à C. Lorsque la Rec. UIT-T X.660 | ISO/CEI 9834-1 spécifie des identificateurs synonymes, tout synonyme peut être utilisé avec la même sémantique. Lorsqu'un même nom est à la fois un identificateur spécifié dans la Rec. UIT-T X.660 | ISO/CEI 9834-1 et une référence de valeur ASN.1 dans le module contenant la forme "NameForm", le nom présent dans la valeur d'identificateur d'objet sera traité comme un identificateur de la Rec. UIT-T X.660 | ISO/CEI 9834-1.

31.8 Le numéro "number" de la notation "NumberForm" ou "XMLNumberForm" sera la valeur numérique affectée au composant de l'identificateur d'objet.

31.9 L'identificateur "identifier" de la notation "NameAndNumberForm" ou "XMLNameAndNumberForm" sera spécifié quand une valeur numérique est affectée au composant de l'identificateur d'objet.

NOTE – Les autorités affectant des valeurs numériques aux composants d'identificateur d'objet sont identifiées dans la Rec. UIT-T X.660 | ISO/CEI 9834-1.

31.10 La sémantique associée à une valeur d'identificateur d'objet est définie dans la Rec. UIT-T X.660 | ISO/CEI 9834-1.

NOTE – Conformément à la Rec. UIT-T X.660 | ISO/CEI 9834-1, une valeur d'identificateur d'objet doit contenir au moins deux arcs.

31.11 La partie significative du composant de l'identificateur d'objet est la forme "NameForm" ou "NumberForm" ou "XMLNumberForm" à laquelle il se réduit et qui fournit la valeur numérique pour le composant de l'identificateur

d'objet. La valeur numérique du composant de l'identificateur d'objet figure toujours dans une instance de la notation de valeur de l'identificateur d'objet, sauf pour les arcs spécifiés dans la Rec. UIT-T X.660 | ISO/CEI 9834-1, Annexes A à C (voir aussi l'Annexe D de la présente Recommandation | Norme internationale).

31.12 Quand la notation "ObjectIdentifierValue" comprend une valeur "DefinedValue" pour une valeur d'identificateur d'objet, la liste des composants de l'identificateur d'objet à laquelle elle se réfère s'ajoute par préfixation aux composants figurant explicitement dans la valeur.

NOTE – Conformément à la Rec. UIT-T X.660 | ISO/CEI 9834-1, il est recommandé d'attribuer également un descripteur d'objet chaque fois qu'une valeur d'identificateur d'objet est attribuée dans le but d'identifier un objet.

EXEMPLES

Avec les identificateurs affectés selon les dispositions de la Rec. UIT-T X.660 | ISO/CEI 9834-1, les valeurs:

```
{ iso standard 8571 pci (1) }
```

et

```
{ 1 0 8571 1 }
```

identifient toutes deux un même objet, **pci**, défini dans l'ISO 8571, de même que:

```
iso.standard.8571.pci(1)
```

et

```
1.0.8571.1
```

dans une valeur "XMLObjectIdentifierValue".

Avec la définition additionnelle suivante:

```
ftam OBJECT IDENTIFIER ::= { iso standard 8571 }
```

la valeur suivante est équivalente aux deux premières valeurs:

```
{ ftam pci(1) }
```

32 Notation du type identificateur d'objet relatif

32.1 Le type identificateur d'objet relatif (voir § 3.6.57) est déclaré par la notation "RelativeOIDType":

RelativeOIDType ::= `RELATIVE-OID`

32.2 L'étiquette de ce type est le numéro 13 de la classe universelle.

32.3 La valeur d'un identificateur d'objet relatif est déclarée par la notation "RelativeOIDValue" ou, lorsqu'elle est utilisée comme une valeur "XMLValue", par la notation "XMLRelativeOIDValue". Ces productions sont:

```
RelativeOIDValue ::=
        "{" RelativeOIDComponentsList "}"

RelativeOIDComponentsList ::=
        RelativeOIDComponents
    |   RelativeOIDComponents RelativeOIDComponentsList

RelativeOIDComponents ::=
        NumberForm
    |   NameAndNumberForm
    |   DefinedValue

XMLRelativeOIDValue ::=
        XMLRelativeOIDComponentList

XMLRelativeOIDComponentList ::=
        XMLRelativeOIDComponent
    |   XMLRelativeOIDComponent & "." & XMLRelativeOIDComponentList

XMLRelativeOIDComponent ::=
        XMLNumberForm
    |   XMLNameAndNumberForm
```

32.4 Les productions "NumberForm", "NameAndNumberForm," "XMLNumberForm", "XMLNameAndNumberForm" et leur sémantique sont définies aux § 31.3 à 31.11.

32.5 La valeur "DefinedValue" de "RelativeOIDComponents" est du type identificateur d'objet relatif et identifie un ensemble ordonné d'arcs depuis un nœud de départ vers un nœud ultérieur dans l'arbre d'identificateurs d'objet. Le nœud de départ est identifié par les composants "RelativeOIDComponents" antérieurs (s'il y en a), et les composants "RelativeOIDComponents" ultérieurs (s'il y en a) identifient les arcs à partir du nœud ultérieur.

32.6 La première notation "RelativeOIDComponents" ou "XMLRelativeOIDComponents" identifie un ou plusieurs arcs à partir d'un nœud de départ vers un nœud ultérieur dans l'arbre d'identificateurs d'objet. Le nœud de départ peut être défini par des commentaires relatifs à la définition du type. En l'absence de définition du nœud de départ dans les commentaires relatifs à la définition de type, cette définition doit être transmise en tant que valeur d'identificateur d'objet dans une instance de communication (voir § E.2.19). Le nœud de départ ne doit être ni la racine, ni un nœud situé immédiatement au-dessous de la racine.

NOTE – Une valeur d'identificateur d'objet relatif doit être associée à une valeur d'identificateur d'objet spécifique afin qu'un objet puisse être identifié sans ambiguïté. Les valeurs d'identificateurs d'objets doivent (voir § 31.10) contenir au moins deux composants. C'est pour cela qu'il y a une restriction sur le nœud de départ.

EXEMPLE

Avec les définitions suivantes:

```
thisUniversity  OBJECT IDENTIFIER       ::=
            {iso member-body country(29) universities(56) thisuni(32)}

firstgroup   RELATIVE-OID   ::=  {science-fac(4) maths-dept(3)}
```

ou en notation de valeur XML:

```
thisUniversity ::= <OBJECT_IDENTIFIER>1.2.29.56.32</OBJECT_IDENTIFIER>

firstgroup ::= <RELATIVE_OID>4.3</RELATIVE_OID>
```

l'identificateur d'objet relatif:

```
relOID   RELATIVE-OID  ::=  {firstgroup room(4) socket(6)}
```

ou en notation de valeur XML:

```
relOID ::= <RELATIVE_OID>4.3.4.6</RELATIVE_OID>
```

peut être utilisé à la place de la valeur d'identificateur d'objet **{1 2 29 56 32 4 3 4 6}** si la racine actuelle (connue de l'application ou transmise par l'application) est **thisUniversity**.

33 Notation du type valeur pdv imbriquée (embedded-pdv type)

33.1 Le type valeur pdv imbriquée (voir § 3.6.21) est déclaré par la notation "EmbeddedPDVType":

EmbeddedPDVType ::= EMBEDDED PDV

NOTE – Le terme "Embedded PDV" désigne une valeur abstraite d'une syntaxe abstraite éventuellement différente (essentiellement, la valeur et le codage d'un message défini dans un protocole distinct mais identifié) qui est imbriquée dans un message. A l'origine, il signifiait "valeur de donnée de présentation imbriquée" de par son utilisation dans la couche Présentation OSI, mais cette signification n'est plus utilisée aujourd'hui et il convient d'interpréter ce terme comme s'agissant d'une "valeur imbriquée".

33.2 L'étiquette de ce type est le numéro 11 de la classe universelle.

33.3 Ce type regroupe des valeurs constituées:

a) du codage d'une valeur de donnée unique, pouvant être une valeur d'un type ASN.1 mais pas nécessairement;

b) de l'identification (jointe ou séparée):

   1) d'une syntaxe abstraite;

   2) de la syntaxe de transfert.

NOTE 1 – La valeur de donnée peut être une valeur d'un type ASN.1, ou être par exemple le codage d'une image fixe ou animée. L'identification consiste en un ou deux identificateurs d'objet, ou (dans un environnement OSI) fait référence à un identificateur de contexte de présentation OSI qui spécifie la syntaxe abstraite et la syntaxe de transfert.

NOTE 2 – L'identification de la syntaxe abstraite et/ou le codage peuvent également être déterminés par le concepteur de l'application sous la forme d'une valeur fixe, auquel cas elle n'est pas codée dans une instance de communication.

33.4 Le type valeur pdv imbriquée possède un type associé, qui sert de base à la notation des valeurs et des sous-types du type valeur pdv imbriquée.

33.5 Dans l'hypothèse d'un environnement d'étiquetage automatique, le type associé servant à la définition des valeurs et au sous-typage est le suivant (les commentaires ont force de norme):

```
SEQUENCE {
      identification                          CHOICE {
            syntaxes                             SEQUENCE {
                  abstract                          OBJECT IDENTIFIER,
                  transfer                          OBJECT IDENTIFIER }
            -- Identificateurs d'objet de la syntaxe abstraite et de la
            -- syntaxe de transfert --,

            syntax                               OBJECT IDENTIFIER
            -- Identificateur d'objet unique identifiant la syntaxe abstraite
            -- et la syntaxe de transfert --,

            presentation-context-id              INTEGER
            -- (Ne s'applique que dans les environnements OSI)
            -- Le contexte de présentation OSI négocié identifie la syntaxe
            -- abstraite et la syntaxe de transfert --,

            context-negotiation                  SEQUENCE {
                  presentation-context-id           INTEGER,
                  transfer-syntax                   OBJECT IDENTIFIER }
            -- (Ne s'applique que dans les environnements OSI)
            -- Négociation de contexte en cours, presentation-context-id
            -- identifie uniquement la syntaxe abstraite,
            -- la syntaxe de transfert doit donc être spécifiée --,

            transfer-syntax                      OBJECT IDENTIFIER
            -- Le type de la valeur (une spécification indiquant par exemple
            -- qu'il s'agit d'une valeur d'un type ASN.1)
            -- est fixé par le concepteur de l'application (et est donc connu à la
            -- fois de l'expéditeur et du destinataire). Ce cas est prévu avant
            -- tout pour prendre en charge le
            -- chiffrement sélectif par champ (ou d'autres transformations de
            -- codage) d'un type ASN.1 --,

            fixed                                NULL
            -- La valeur de donnée est une valeur d'un type ASN.1 fixe
            -- (qui est donc connue à la fois de l'expéditeur et du
            -- destinataire) -- },

      data-value-descriptor                  ObjectDescriptor  OPTIONAL
      -- Il s'agit de l'identification en langage naturel de la classe
      -- de la valeur --,
      data-value                             OCTET STRING }

( WITH COMPONENTS {
            ... ,
            data-value-descriptor  ABSENT } )
```

NOTE – Le type valeur pdv imbriquée ne permet pas d'inclure une valeur de descripteur de valeur de donnée **data-value-descriptor**. Toutefois, la définition du type associé donnée ici souligne les points communs existant entre le type valeur pdv imbriquée, le type externe et le type chaîne de caractères à alphabet non restreint.

33.6 La forme **presentation-context-id** n'est applicable que dans un environnement OSI, lorsque la valeur entière est l'identificateur d'un contexte de présentation OSI dans l'ensemble de contextes définis OSI. Cette forme ne sera pas utilisée pendant la négociation du contexte OSI.

33.7 La forme **context-negotiation** n'est applicable que dans un environnemtn OSI et ne sera utilisée que pendant la négociation du contexte OSI. La valeur entière sera l'identificateur d'un contexte de présentation OSI qu'il est proposé d'ajouter à l'ensemble de contextes définis OSI. L'identificateur d'objet **transfer-syntax** identifiera une syntaxe de transfert pour ce contexte de présentation OSI, à utiliser pour coder la valeur.

33.8 Une valeur de type valeur pdv imbriquée est déclarée par la notation de valeur du type associé défini au § 33.5, où la valeur du composant **data-value** du type chaîne d'octets **OCTET STRING** représente un codage utilisant la syntaxe de transfert spécifiée dans **identification**.

```
EmbeddedPdvValue ::= SequenceValue          -- valeur du type associé défini au § 33.5

XMLEmbeddedPDVValue ::= XMLSequenceValue    -- valeur du type associé défini au § 33.5
```

EXEMPLE – S'il faut forcer le choix d'une seule option, par exemple celui de la forme **syntaxes**, on peut y parvenir en écrivant:

```
EMBEDDED PDV (WITH COMPONENTS {
                  ... ,
                  identification  (WITH COMPONENTS {
                        syntaxes  PRESENT } ) } )
```

34 Notation du type externe (external type)

34.1 Le type externe (voir § 3.6.37) est déclaré par la notation "ExternalType":

ExternalType ::= EXTERNAL

34.2 L'étiquette de ce type est le numéro 8 de la classe universelle.

34.3 Ce type regroupe des valeurs constituées:

a) du codage d'une valeur de donnée simple, pouvant être une valeur d'un type ASN.1 mais pas nécessairement;

b) de l'identification:

1) d'une syntaxe abstraite;

2) de la syntaxe de transfert;

c) (optionnellement) d'un descripteur d'objet donnant une description en langage naturel de la catégorie de la valeur de donnée. Le descripteur d'objet optionnel ne sera présent que si le commentaire associé à l'utilisation de la notation "ExternalType" le permet explicitement.

NOTE – La Note 1 du § 33.3 s'applique également au type externe.

34.4 Le type externe possède un type associé, qui sert à préciser la définition des valeurs abstraites du type externe, et qui sert également de base à la notation des valeurs et des sous-types du type externe.

NOTE – Les règles de codage peuvent définir un type différent utilisé pour déterminer les codages, ou peuvent spécifier les codages sans faire référence à un quelconque type associé. Par exemple, les règles de codage de base (BER) utilisent un type séquence différent pour des raisons historiques.

34.5 Dans l'hypothèse d'un environnement d'étiquetage automatique, le type associé servant à la définition des valeurs et au sous-typage est le suivant (les commentaires ont force de norme):

```
SEQUENCE {
      identification                          CHOICE {
            syntaxes                             SEQUENCE {
                  abstract                           OBJECT IDENTIFIER,
                  transfer                           OBJECT IDENTIFIER }
            -- Identificateurs d'objet de la syntaxe abstraite et de la
            -- syntaxe de transfert --,

            syntax                               OBJECT IDENTIFIER
            -- Identificateur d'objet unique identifiant la syntaxe abstraite et la
            -- syntaxe de transfert --,

            presentation-context-id              INTEGER
            -- (Ne s'applique que dans les environnements OSI)
            -- Le contexte de présentation OSI négocié identifie la syntaxe
            -- abstraite et la syntaxe de transfert --,

            context-negotiation                  SEQUENCE {
                  presentation-context-id           INTEGER,
                  transfer-syntax                   OBJECT IDENTIFIER }
            -- (Ne s'applique que dans les environnements OSI)
            -- Négociation de contexte en cours, presentation-context-id
            -- identifie uniquement la syntaxe abstraite,
            -- la syntaxe de transfert doit donc être spécifiée  --,

            transfer-syntax                      OBJECT IDENTIFIER
            -- Le type de la valeur (une spécification indiquant par exemple qu'il
            -- s'agit d'une valeur
            -- d'un type ASN.1) est fixé par le concepteur de l'application (et est
            -- donc connu à la fois
            -- de l'expéditeur et du destinataire). Ce cas est prévu avant tout
            -- pour prendre en charge le
```

```
            -- chiffrement sélectif par champ (ou d'autres transformations de
            -- codage) d'un type ASN.1 --,

            fixed                                   NULL
            -- La valeur de donnée est une valeur d'un type ASN.1 fixe
            -- (qui est donc connu à la fois de l'expéditeur et du
            -- destinataire) -- },

      data-value-descriptor                 ObjectDescriptor  OPTIONAL
            -- Il s'agit de l'identification en langage naturel de la classe de la
            -- valeur --,

      data-value                            OCTET STRING }
            ( WITH COMPONENTS {
                  ... ,
                  identification  (WITH COMPONENTS {
                        ... ,
                        syntaxes                                ABSENT,
                        transfer-syntax                         ABSENT,
                        fixed                           ABSENT } ) } )
```

NOTE – Pour des raisons historiques, le type externe ne permet pas l'utilisation des formes **syntaxes**, **transfer-syntax** et **fixed** de la notation **identification**. Les concepteurs d'applications ayant besoin de faire appel à ces formes devront utiliser le type valeur pdv imbriquée. La définition du type associé donnée ici souligne les points communs existant entre le type externe, le type chaîne de caractères à alphabet non restreint et le type valeur pdv imbriquée.

34.6 Les dispositions des § 33.6 et 33.7 s'appliquent également au type externe.

34.7 Une valeur de type externe est déclarée par la notation de valeur du type associé défini au § 34.5, où la valeur du composant **data-value** de type chaîne d'octets **OCTET STRING** représente un codage utilisant la syntaxe de transfert spécifiée dans **identification**.

ExternalValue ::= SequenceValue *-- valeur du type associé défini au § 34.5*

XMLExternalValue ::= XMLSequenceValue *-- valeur du type associé défini au § 34.5*

NOTE – Pour des raisons historiques, les règles de codage permettent de transférer des valeurs imbriquées de type externe **EXTERNAL** codés sur un nombre de bits qui n'est pas multiple de huit. De telles valeurs ne peuvent pas être représentées par une notation de valeur au moyen du type associé ci-dessus.

35 Les types chaînes de caractères (character string types)

Ces types représentent les chaînes de caractères constituées à partir d'un répertoire de caractères spécifié donné. Il est normal de définir un répertoire de caractères et son codage en les représentant sous la forme de cellules disposées en une ou plusieurs grilles, chaque cellule correspondant à un caractère du répertoire. D'habitude, on affecte également à chaque cellule un symbole graphique et un nom de caractère, bien que dans certains répertoires, des cellules soient laissées vides ou qu'elles aient un nom mais pas de forme imprimable (on peut citer comme exemples de telles cellules ayant un nom mais pas de forme affectée les caractères de contrôle comme EOF de l'ISO/CEI 646 et les caractères d'espacement comme THIN-SPACE and EN-SPACE de l'ISO/CEI 10646-1).

En général, l'information associée à une cellule décrit un caractère abstrait distinct même si cette information est vide (pas de symbole graphique ni de nom affecté à la cellule).

La notation de valeur de base ASN.1 des types chaînes de caractères possède trois variantes (combinables) spécifiées comme suit:

a) une représentation des caractères de la chaîne utilisant des symboles graphiques attribués aux caractères, y compris éventuellement les caractères d'espacement; c'est la notation "cstring";

   NOTE 1 – Une telle représentation peut être ambiguë en représentation imprimée lorsqu'un même symbole graphique est attribué à plus d'un caractère du répertoire.

   NOTE 2 – Une telle représentation peut être ambiguë en représentation imprimée lorsque des caractères d'espacement de largeurs différentes sont présents dans le répertoire ou que la spécification est imprimée dans une police à espacement proportionnel.

b) une liste des caractères représentant la valeur de la chaîne sous la forme d'une série des références de valeurs ASN.1 affectées aux caractères; un ensemble de telles références de valeurs est défini dans le module **ASN1-CHARACTER-MODULE** du § 38 pour les répertoires de caractères de l'ISO/CEI 10646-1 et **IA5String**; cette forme n'est utilisable pour les autres répertoires de caractères que si l'utilisateur leur affecte des références de valeurs au moyen de la notation décrite au point a) ci-dessus ou au point c) ci-dessous;

c) une liste des caractères représentant la valeur de la chaîne sous la forme d'une suite de positions de cellules dans la ou les grilles du répertoire; cette forme n'est utilisable que pour les chaînes **IA5String**, **UniversalString**, **UTF8String** et **BMPString**.

La notation de valeur XML ASN.1 des types chaînes de caractères utilise la notation "xmlcstring", qui offre la possibilité d'utiliser des séquences d'échappement pour certains caractères spéciaux ainsi que pour la spécification de caractères en décimal ou hexadécimal (voir § 11.15).

36 Notation des types chaînes de caractères

36.1 Un type chaîne de caractères (voir § 3.6.11) est déclaré par la notation:

CharacterStringType ::=
RestrictedCharacterStringType
| **UnrestrictedCharacterStringType**

"RestrictedCharacterStringType" est la notation d'un type chaîne de caractères à alphabet restreint défini au § 37. "UnrestrictedCharacterStringType" est la notation du type chaîne de caractères à alphabet non restreint défini au § 40.1.

36.2 Les étiquettes des différents types chaîne de caractères à alphabet restreint sont spécifiées au § 37.1. L'étiquette du type chaîne de caractères à alphabet non restreint est spécifiée au § 40.2.

36.3 Une valeur de chaîne de caractères est déclarée par la notation suivante:

CharacterStringValue ::=
RestrictedCharacterStringValue
| **UnrestrictedCharacterStringValue**

XMLCharacterStringValue ::=
XMLRestrictedCharacterStringValue
| **XMLUnrestrictedCharacterStringValue**

Les valeurs "RestrictedCharacterStringValue" et "XMLRestrictedCharacterStringValue" sont respectivement définies au § 37.8 et au § 37.9. "UnrestrictedCharacterStringValue" et "XMLUnrestrictedCharacterStringValue" sont les notations d'une valeur de chaîne de caractères à alphabet non restreint; elles sont définies au § 40.7.

37 Définition des types chaîne de caractères à alphabet restreint

Le présent paragraphe définit les types dont les valeurs sont limitées aux séquences de zéro, un ou plusieurs caractères appartenant à un ensemble de caractères donné. Un type chaîne de caractères à alphabet restreint est déclaré par la notation "RestrictedCharacterStringType":

RestrictedCharacterStringType ::=
BMPString
| **GeneralString**
| **GraphicString**
| **IA5String**
| **ISO646String**
| **NumericString**
| **PrintableString**
| **TeletexString**
| **T61String**
| **UniversalString**
| **UTF8String**
| **VideotexString**
| **VisibleString**

Chaque forme de "RestrictedCharacterStringType" est définie en spécifiant:

a) l'étiquette qui lui est affectée;

b) un nom (par exemple **NumericString**) par lequel le type est désigné;

c) les caractères de l'ensemble utilisé pour définir le type, par référence à une table énumérant les différentes formes graphiques des caractères, ou par référence à un numéro d'enregistrement du registre international des jeux de caractères codés de l'ISO (voir le *Registre international de l'ISO des jeux de caractères codés à utiliser avec une séquence d'échappement*), ou par référence à l'ISO/CEI 10646-1.

37.1 Le Tableau 6 donne la liste des noms désignant chacun des types chaîne de caractères à alphabet restreint, le numéro de l'étiquette de la classe universelle affectée à ce type, le tableau, paragraphe ou numéro d'enregistrement de définition, et, le cas échéant, le numéro de la Note relative à cette entrée du tableau. Lorsqu'un nom de type possède un synonyme, celui-ci est indiqué entre parenthèses.

Tableau 6 – Liste des types de chaînes de caractères à alphabet restreint

Nom désignant le type de chaîne	Numéro de la classe universelle	Numéro d'enregistrement, numéro du tableau ou paragraphe de la Rec. UIT-T X.680 | ISO/CEI 8824-1 de définition[a)]	Notes
`UTF8String` (format de transformation UCS 8)	12	§ 37.16	
`NumericString` (chaîne numérique)	18	Tableau 7	(Note 1)
`PrintableString` (chaîne imprimable)	19	Tableau 8	(Note 1)
`TeletexString` (T61String) [chaîne télétex (ou chaîne T61)]	20	6, 87, 102, 103, 106, 107, 126, 144, 150, 153, 156, 164, 165, 168 + SPACE + DELETE	(Note 2)
`VideotexString` (chaîne vidéotex)	21	1, 13, 72, 73, 87, 89, 102, 108, 126, 128, 129, 144, 150, 153, 164, 165, 168 + SPACE + DELETE	(Note 3)
`IA5String` (chaîne alphabet international nº 5)	22	1, 6 + SPACE + DELETE	
`GraphicString` (chaîne graphique)	25	Tous les jeux graphiques + SPACE	
`VisibleString` (ISO646String) [chaîne visible (chaîne ISO 646)]	26	6 + SPACE	
`GeneralString` (chaîne générale)	27	Tous les jeux graphiques et caractères + SPACE + DELETE	
`UniversalString` (chaîne universelle)	28	voir § 37.6	
`BMPString` (chaîne multilingue)	30	voir § 37.15	

a) Les numéros d'enregistrement de définition sont énumérés dans le Registre international de l'ISO des jeux de caractères codés à utiliser avec les séquences d'échappement.

NOTE 1 – Le style typographique, le corps, la couleur, la graisse et les autres caractéristiques d'impression ou d'affichage ne sont pas significatifs.

NOTE 2 – Les entrées de registre 6 et 156 peuvent être utilisées à la place des entrées 102 et 103.

NOTE 3 – Les entrées correspondant à ces numéros d'enregistrement assurent les fonctionnalités définies dans les Rec.T.100 du CCITT et Rec. UIT-T T.101.

37.2 Le Tableau 7 donne la liste des caractères susceptibles d'apparaître dans les chaînes numériques **`NumericString`** et dans leur syntaxe abstraite de caractères.

Tableau 7 – Chaînes numériques "NumericString"

Nom	Caractère graphique
Chiffres	**`0`**, **`1`**, ... **`9`**
Espace	(espace)

37.3 Les valeurs d'identificateur d'objet et de descripteur d'objet suivantes sont affectées pour identifier et décrire la syntaxe abstraite de caractères de **`NumericString`**:

```
{ joint-iso-itu-t asn1(1) specification(0) characterStrings(1) numericString(0) }
```

et

```
"NumericString character abstract syntax"
```

NOTE 1 – Cette valeur d'identificateur d'objet peut être utilisée dans des valeurs de type chaîne de caractères **`CHARACTER STRING`** lorsqu'il est nécessaire de transmettre l'identificateur du type de chaîne de caractères indépendamment de la valeur.

NOTE 2 – Une valeur de syntaxe abstraite du type chaîne numérique **`NumericString`** peut être codée:

a) par une des règles données dans l'ISO/CEI 10646-1 pour le codage des caractères abstraits. Dans ce cas, la syntaxe de transfert de caractères est identifiée par l'identificateur d'objet associé aux règles dans l'ISO/CEI 10646-1, Annexe N;

b) par les règles de codage ASN.1 du type prédéfini **NumericString**. Dans ce cas, la syntaxe de transfert de caractères est identifiée par la valeur d'identificateur d'objet **{joint-iso-itu-t asn1(1) basic-encoding(1)}**.

37.4 Le Tableau 8 donne la liste des caractères susceptibles d'apparaître dans les chaînes imprimables **PrintableString** et dans leur syntaxe abstraite de caractères.

Tableau 8 – Chaînes imprimables "PrintableString"

Nom	Caractère graphique
Majuscules latines	**A, B, ... Z**
Minuscules latines	**a, b, ... z**
Chiffres	**0, 1, ... 9**
ESPACE	(espace)
APOSTROPHE	**'**
PARENTHÈSE GAUCHE	**(**
PARENTHÈSE DROITE	**)**
SIGNE PLUS	**+**
VIRGULE	**,**
TRAIT D'UNION – SIGNE MOINS	**-**
POINT	**.**
BARRE OBLIQUE	**/**
DEUX-POINTS	**:**
SIGNE ÉGAL	**=**
POINT D'INTERROGATION	**?**

37.5 Les valeurs d'identificateur d'objet et de descripteur d'objet suivantes ont été affectées pour identifier et décrire la syntaxe abstraite de caractères **PrintableString**:

{ joint-iso-itu-t asn1(1) specification(0) characterStrings(1) printableString(1) }

et

"PrintableString character abstract syntax"

NOTE 1 – Cette valeur d'identificateur d'objet peut être utilisée dans des valeurs de type chaîne de caractères **CHARACTER STRING** lorsqu'il est nécessaire de transmettre l'identificateur du type de chaîne de caractères indépendamment de la valeur.

NOTE 2 – Une valeur de syntaxe abstraite du type chaîne imprimable **PrintableString** peut être codée:

a) par une des règles données dans l'ISO/CEI 10646-1 pour le codage des caractères abstraits. Dans ce cas, la syntaxe de transfert de caractères est identifiée par l'identificateur d'objet associé aux règles dans l'ISO/CEI 10646-1, Annexe N;

b) par les règles de codage ASN.1 du type prédéfini **PrintableString**. Dans ce cas, la syntaxe de transfert de caractères est identifiée par la valeur d'identificateur d'objet **{ joint-iso-itu-t asn1(1) basic-encoding(1) }**.

37.6 Les caractères susceptibles d'apparaître dans les chaînes de type chaîne universelle **UniversalString** sont tous les caractères autorisés par l'ISO/CEI 10646-1.

37.7 L'utilisation de ce type impose le respect des spécifications de conformité indiquées dans l'ISO/CEI 10646-1.

NOTE – Le paragraphe 38 contient un module ASN.1 définissant un certain nombre de sous-types de ce type pour les "collections de caractères graphiques des sous-jeux" définies dans l'ISO/CEI 10646-1, Annexe A.

37.8 Une valeur "RestrictedCharacterStringValue" de type chaîne de caractères à alphabet restreint est déclarée par la notation "cstring" (voir § 11.14), "CharacterStringList", "Quadruple" ou "Tuple". "Quadruple" ne peut définir qu'une chaîne de caractères de longueur 1, et ne peut être utilisée que pour noter une valeur de type **UniversalString**, **UTF8String** ou **BMPString**. "Tuple" ne peut définir qu'une chaîne de caractères de longueur 1, et ne peut être utilisée que pour noter une valeur de type **IA5String**.

RestrictedCharacterStringValue ::=
cstring
| **CharacterStringList**
| **Quadruple**
| **Tuple**

```
CharacterStringList ::=  "{" CharSyms "}"

CharSyms ::=
        CharsDefn
    |   CharSyms "," CharsDefn

CharsDefn ::=
        cstring
    |   Quadruple
    |   Tuple
    |   DefinedValue

Quadruple ::= "{" Group "," Plane "," Row "," Cell "}"

Group    ::= number

Plane    ::= number

Row      ::= number

Cell     ::= number

Tuple    ::= "{" TableColumn "," TableRow "}"

TableColumn ::= number

TableRow ::= number
```

NOTE 1 – La notation "cstring" ne peut être utilisée sans ambiguïté que sur un dispositif capable d'afficher les symboles graphiques des caractères figurant dans la valeur. Réciproquement, si le dispositif utilisé n'offre pas une telle capacité, le seul moyen de spécifier sans ambiguïté une valeur de chaîne de caractères utilisant de tels symboles est la notation "CharacterStringList", sous réserve que la chaîne soit de type **`UniversalString`**, **`UTF8String`**, **`BMPString`** ou **`IA5String`**, et seulement si la forme "DefinedValue" de "CharsDefn" est utilisée (voir § 38.1.2).

NOTE 2 – Le paragraphe 38 définit un certain nombre de références "valuereference" désignant des caractères simples (chaînes de longueur 1) du type **`BMPString`** (et donc **`UniversalString`** et **`UTF8String`**) et **`IA5String`**.

EXEMPLE – Supposons que l'on souhaite spécifier la valeur "abcΣdef" de type **`UniversalString`** dans laquelle le caractère "Σ" ne peut pas être représenté sur le dispositif utilisé. Cette valeur peut alors s'écrire comme suit:

```
IMPORTS BasicLatin, greekCapitalLetterSigma FROM ASN1-CHARACTER-MODULE
      { joint-iso-itu-t asn1(1) specification(0) modules(0) iso10646(0) };

MyAlphabet ::= UniversalString (FROM (BasicLatin | greekCapitalLetterSigma))

mystring MyAlphabet ::= { "abc" , greekCapitalLetterSigma , "def" }
```

NOTE 3 – Lorsqu'on spécifie une valeur du type **`UniversalString`**, **`UTF8String`** ou **`BMPString`**, la notation "cstring" ne sera pas utilisée à moins que n'aient été résolues les ambiguïtés résultant de la similitude de forme de caractères différents.

EXEMPLE – La notation "cstring" suivante ne doit pas être utilisée car les symboles graphiques "P", "O", "I", "N" et "T" existent dans les alphabets BASIC LATIN, CYRILLIC et BASIC GREEK et sont donc ambigus.

```
IMPORTS BasicLatin, Cyrillic, BasicGreek FROM ASN1-CHARACTER-MODULE
      { joint-iso-itu-t asn1(1) specification(0) modules(0) iso10646(0) };

MyAlphabet ::= UniversalString (FROM (BasicLatin | Cyrillic | BasicGreek))

mystring MyAlphabet ::= "HOPE"
```

Une définition alternative non ambiguë de **`mystring`** serait:

```
mystring  MyAlphabet(BasicLatin) ::= "HOPE"
```

Formellement, **`mystring`** est une référence à une valeur d'un sous-ensemble de **`MyAlphabet`**, mais elle peut, en vertu des règles de mappage entre valeurs de l'Annexe B, être utilisée pour référencer la même chaîne mais en tant que valeur du type **`MyAlphabet`**.

37.9 La notation "XMLRestrictedCharacterStringValue" est définie comme suit:

```
XMLRestrictedCharacterStringValue ::= xmlcstring
```

37.10 Certains caractères ne peuvent pas être représentés directement en notation "xmlcstring". Ils doivent donc être représentés au moyen des séquences d'échappement spécifiées au § 11.15.

NOTE – Si la valeur de type chaîne de caractères à alphabet restreint contient des caractères qui ne sont pas des caractères ISO/CEI 10646-1 spécifiés au § 11.15.1, ceux-ci ne peuvent pas être représentés en notation "xmlcstring", et la valeur ne peut pas être transférée au moyen des règles de codage XML (voir la Rec. UIT-T X.693 | ISO/CEI 8825-4).

37.11 La valeur "DefinedValue" de "CharsDefn" renverra à une valeur de ce type.

37.12 Les numéros "number" seront inférieurs à 256 dans les productions "Plane", "Row" et "Cell" et à 128 dans la production "Group".

37.13 "Group" spécifie un groupe de l'espace de codage du système de codage universel UCS, "Plane" spécifie un plan du groupe, "Row" spécifie une rangée dans le plan, et "Cell" spécifie une cellule dans la rangée. Le caractère abstrait identifié par cette notation est celui de la cellule spécifiée par les valeurs "Group", "Plane", "Row", et "Cell". Dans tous les cas, l'ensemble des caractères autorisés peut être restreint par sous-typage.

NOTE – Les concepteurs d'applications doivent faire attention aux problèmes de conformité qui pourraient se poser lorsqu'ils utilisent des types chaînes de caractères ouverts tels que **GeneralString**, **GraphicString**, and **UniversalString**, sans leur appliquer de contraintes. Un texte précis sur la conformité est aussi nécessaire pour les types chaînes de caractères de taille limitée mais grande, tels que **TeletexString**.

37.14 Le numéro "number" dans la production "TableColumn" sera compris entre zéro et sept, et le numéro "number" dans la production "TableRow" sera compris entre zéro et quinze. "TableColumn" désigne une colonne et "TableRow" une rangée de la grille des codes des caractères conformément à la disposition représentée Figure 1 de l'ISO/CEI 2022. Cette notation n'est utilisée que pour les chaînes du type **IA5String** lorsque la grille de codage contient l'entrée de registre 1 en colonnes 0 et 1 et l'entrée de registre 6 dans les colonnes 2 à 7 (voir le *Registre international de l'ISO des jeux de caractères codés à utiliser avec les séquences d'échappement*).

37.15 Le type chaîne multilingue **BMPString** est un sous-type du type chaîne universelle **UniversalString** qui possède sa propre étiquette et contient uniquement les caractères de la table multilingue BMP (*basic multilingual plane*) (ceux correspondant aux 64K-2 premières cellules, moins les cellules dont le codage est utilisé pour désigner des caractères en dehors de la table multilingue BMP) de la grille ISO/CEI 10646-1. Son type associé est:

```
UniversalString (Bmp)
```

où **Bmp** est défini dans le module ASN-1 **ASN1-CHARACTER-MODULE** (voir le § 38) comme le sous-type du type **UniversalString** correspondant au nom de collection "BMP" défini dans l'ISO/CEI 10646-1, Annexe A.

NOTE 1 – Comme **BMPString** est un type prédéfini, il n'est pas défini dans le module **ASN1-CHARACTER-MODULE**.

NOTE 2 – La définition du type **BMPString** en tant que type prédéfini a pour but de permettre aux règles de codage (comme les règles de base BER) obéissant à des contraintes d'utiliser un codage sur 16 bits plutôt que sur 32 bits.

NOTE 3 – Toutes les valeurs du type **BMPString** sont également des notations de valeurs valides des types **UniversalString** et **UTF8String**.

37.16 Le type **UTF8String** est synonyme du type **UniversalString** au niveau abstrait et peut être utilisé chaque fois que ce dernier est utilisé (compte tenu des règles exigeant des étiquettes distinctes), mais il possède une étiquette différente et constitue un type distinct.

NOTE – Le codage de **UTF8String** utilisé par les règles BER et PER est différent de celui de **UniversalString** et sera moins verbeux pour la plupart des textes.

38 Dénomination des caractères et collections de caractères définis dans l'ISO/CEI 10646-1

Le présent paragraphe définit un module prédéfini ASN.1 qui donne la définition d'un nom de référence de valeur pour chaque caractère défini dans l'ISO/CEI 10646-1. Chacun de ces noms renvoie à une valeur de type **UniversalString** de taille 1. Ce module définit également un nom de référence de type pour chacune des collections de caractères de l'ISO/CEI 10646-1, chacun de ces noms renvoyant à un sous-ensemble du type **UniversalString**.

NOTE – Ces valeurs peuvent être utilisées pour noter les valeurs du type **UniversalString** et des sous-types qui en dérivent. Toutes les références de valeurs et de types spécifiées au § 38.1 sont exportées, et doivent être importées par tout module qui les utilise.

38.1 Spécification du module ASN-1 "ASN1-CHARACTER-MODULE"

Ce module n'est pas reproduit ici en entier. En revanche, on spécifie la manière dont il est défini.

38.1.1 Le module commence comme suit:

```
ASN1-CHARACTER-MODULE { joint-iso-itu-t asn1(1) specification(0) modules(0)
iso10646(0) }
      DEFINITIONS ::= BEGIN
```

```
-- Toutes les références de valeurs et de types définies dans ce module sont
-- implicitement exportées et peuvent être importées par tout module.
-- Caractères de contrôle de l'ISO/CEI 646:

nul   IA5String ::= {0, 0}
soh   IA5String ::= {0, 1}
stx   IA5String ::= {0, 2}
etx   IA5String ::= {0, 3}
eot   IA5String ::= {0, 4}
enq   IA5String ::= {0, 5}
ack   IA5String ::= {0, 6}
bel   IA5String ::= {0, 7}
bs    IA5String ::= {0, 8}
ht    IA5String ::= {0, 9}
lf    IA5String ::= {0,10}
vt    IA5String ::= {0,11}
ff    IA5String ::= {0,12}
cr    IA5String ::= {0,13}
so    IA5String ::= {0,14}
si    IA5String ::= {0,15}
dle   IA5String ::= {1, 0}
dc1   IA5String ::= {1, 1}
dc2   IA5String ::= {1, 2}
dc3   IA5String ::= {1, 3}
dc4   IA5String ::= {1, 4}
nak   IA5String ::= {1, 5}
syn   IA5String ::= {1, 6}
etb   IA5String ::= {1, 7}
can   IA5String ::= {1, 8}
em    IA5String ::= {1, 9}
sub   IA5String ::= {1,10}
esc   IA5String ::= {1,11}
is4   IA5String ::= {1,12}
is3   IA5String ::= {1,13}
is2   IA5String ::= {1,14}
is1   IA5String ::= {1,15}
del   IA5String ::= {7,15}
```

38.1.2 Pour chaque entrée de chaque liste de noms de caractères graphiques (glyphes) figurant dans les § 24 et 25 de l'ISO/CEI 10646-1, le modèle inclut une déclaration de la forme:

```
<namedcharacter> BMPString ::= <tablecell>
-- représentant le caractère <iso10646name>, voir l'ISO/CEI 10646-1
```

où:

a) <*iso10646name*> est le nom du caractère dérivé de la liste donnée dans l'ISO/CEI 10646-1;

b) <*namedcharacter*> est une chaîne obtenue en appliquant les procédures spécifiées au § 38.2 au caractère <*iso10646name*> ;

c) <*tablecell*> est le glyphe de la cellule du tableau de l'ISO/CEI 10646-1 correspondant à l'entrée dans la liste.

EXEMPLE

```
latinCapitalLetterA  BMPString  ::=  {0, 0, 0, 65}
-- représente le caractère A MAJUSCULE LATIN, voir l'ISO/CEI 10646-1

greekCapitalLetterSigma BMPString  ::=  {0, 0, 3, 163}
-- représente le caractère SIGMA MAJUSCULE GREC, voir l'ISO/CEI 10646-1
```

38.1.3 Pour chaque nom de collection de caractères graphiques spécifié dans l'ISO/CEI 10646-1, Annexe A, le module comporte une déclaration de la forme:

```
<namedcollectionstring> ::= BMPString
      (FROM (<alternativelist>))
      -- représente la collection de caractères <collectionstring>,
      -- voir l'ISO/CEI 10646-1.
```

où:

a) <*collectionstring*> est le nom de la collection de caractères affecté par l'ISO/CEI 10646-1;

b) <*namedcollectionstring*> est formé en appliquant à <*collectionstring*> les procédures du § 38.3;

c) <*alternativelist*> est formé en utilisant les caractères <*namedcharacter*> engendrés comme l'indique le § 38.2 pour chacun des caractères spécifiés dans l'ISO/CEI 10646-1.

La référence de type résultante <*namedcollectionstring*> forme un sous-ensemble limité de la collection (voir les exemples didactiques de l'Annexe F).

NOTE – Un sous-ensemble limité est une liste de caractères appartenant à un sous-ensemble spécifié, par opposition à un sous-ensemble sélectionné, qui est une des collections de caractères énumérées dans l'ISO/CEI 10646-1, Annexe A, plus la collection latine de base.

EXEMPLE (partiel):

```
space BMPString              ::= {0, 0, 0, 32}
exclamationMark BMPString    ::= {0, 0, 0, 33}
quotationMark BMPString      ::= {0, 0, 0, 34}
...                          -- et ainsi de suite
tilde BMPString              ::= {0, 0, 0, 126}

BasicLatin ::= BMPString
      (FROM (space
      | exclamationMark
      | quotationMark
      | ...       -- et ainsi de suite
      | tilde)
      )
-- représente la collection de caractères latins de base, voir l'ISO/CEI 10646-1.
-- Les points de suspension, utilisés dans cet exemple par souci d'abréviation,
-- signifient "et ainsi de suite"; il ne faut pas les utiliser sous cette forme
-- dans un véritable module ASN.1.
```

38.1.4 L'ISO/CEI 10646-1 définit trois niveaux d'implémentation. Tous les types définis dans le module **ASN1-CHARACTER-MODULE** à l'exception de **Level1** et de **Level2** sont par défaut conformes à une implémentation de niveau 3, car de tels types n'imposent aucune restriction quant à l'utilisation des caractères de combinaison. **Level1** indique qu'une implémentation de niveau 1 est requise; **Level2** indique qu'une implémentation de niveau 2 est requise, et **Level3** indique qu'une implémentation de niveau 3 est requise. Ces types sont définis dans le module **ASN1-CHARACTER-MODULE**:

```
Level1 ::= BMPString (FROM (ALL EXCEPT CombiningCharacters))

Level2 ::= BMPString (FROM (ALL EXCEPT CombiningCharactersType-2))

Level3 ::= BMPString
```

NOTE 1 – **CombiningCharacters** et **CombiningCharactersType-2** sont les chaînes de collections nommées <namedcollectionstring> correspondant respectivement aux caractères de combinaison (COMBINING CHARACTERS) et aux caractères de combinaison de type 2 (COMBINING CHARACTERS B-2) définies dans l'ISO/CEI 10646-1, Annexe A.

NOTE 2 – **Level1** et **Level2** sont utilisés à la suite d'un signe "IntersectionMark" (voir le § 46) ou comme seule contrainte dans une spécification "ConstraintSpec". (Un exemple en est donné au § E.2.7.1.)

NOTE 3 – Pour plus de détails, voir § F.2.5.

38.1.5 Le module se termine par la déclaration:

```
END
```

38.1.6 Un équivalent défini par l'utilisateur de l'exemple donné au § 38.1.3 pourrait être:

```
BasicLatin ::= BMPString (FROM (space..tilde))
-- représente la collection de caractères latins de base (BASIC LATIN) voir
-- l'ISO/CEI 10646-1.
```

38.2 Un nom <*namedcharacter*> est une chaîne dérivée d'un nom <*iso10646name*> (voir § 38.1.2) en lui appliquant l'algorithme suivant:

a) chaque majuscule de <*iso10646name*> est mise en minuscule, sauf si elle est précédée par un ESPACE, auquel cas elle reste inchangée;

b) les chiffres et les TRAITS D'UNION – SIGNES MOINS restent inchangés;

c) les ESPACES sont supprimés.

NOTE – L'algorithme ci-dessus, appliqué conjointement avec les directives d'appellation des caractères énoncées dans l'ISO/CEI 10646-1, Annexe K, produira toujours une notation de valeur non ambiguë pour chacun des noms de caractère figurant dans l'ISO/CEI 10646-1.

EXEMPLE – Le caractère ISO/CEI 10646-1, rangée 0, cellule 60, appelé "SIGNE INFERIEUR A", dont la représentation graphique est "<", peut être désigné par la chaîne de type "DefinedValue":

```
less-thanSign
```

38.3 Le nom <*namedcollectionstring*> est la chaîne dérivée du nom <*collectionstring*> en lui appliquant l'algorithme suivant:

a) chaque majuscule du nom de la collection de l'ISO/CEI 10646-1 est mise en minuscule, sauf si elle est précédée par un ESPACE ou est la première lettre du nom, auquel cas elle reste inchangée;

b) les chiffres et les TRAITS D'UNION – SIGNES MOINS restent inchangés;

c) les ESPACES sont supprimés.

EXEMPLES

1) La collection identifiée dans l'ISO/CEI 10646-1, Annexe A, sous le nom:

BASIC LATIN

a la référence de type ASN.1:

```
BasicLatin
```

2) Un type chaîne de caractères composé des caractères de la collection BASIC LATIN, ainsi que de la collection BASIC ARABIC peut être défini de la manière suivante:

```
My-Character-String ::= BMPString (FROM (BasicLatin | BasicArabic) )
```

NOTE – La formulation ci-dessus est nécessaire car la formulation apparemment plus simple:

```
My-Character-String ::= BMPString (BasicLatin | BasicArabic)
```

ne permettrait d'écrire que des chaînes dont tous les caractères appartiennent à la collection soit latine BASIC LATIN soit arabe BASIC ARABIC mais pas à un mélange des deux.

39 Ordre canonique des caractères

39.1 A des fins de sous-typage par la notation d'intervalle de valeurs "ValueRange" et d'utilisation possible par les règles de codage, un ordre canonique des caractères a été spécifié pour les types de chaîne **`UniversalString`**, **`UTF8String`**, **`BMPString`**, **`NumericString`**, **`PrintableString`**, **`VisibleString`** et **`IA5String`**.

39.2 Aux fins de ce seul paragraphe, les caractères sont en correspondance biunivoque avec les cellules d'une grille de codage, qu'un nom ou une forme de caractère ait été ou non affecté à ces cellules, qu'il s'agisse de caractères de contrôle ou de caractères imprimables, et qu'il s'agisse ou non de caractères de combinaison.

39.3 L'ordre canonique d'un caractère abstrait est défini par l'ordre canonique de sa valeur dans la représentation à 32 bits de l'ISO/CEI 10646-1, les petits nombres figurant en premier et les grands nombres en dernier dans l'ordre canonique.

39.4 Les extrémités des intervalles de valeurs "ValueRange" dans les notations d'alphabet permis "PermittedAlphabet" (ou les différents caractères pris individuellement) peuvent être désignées soit en utilisant la référence de valeur ASN.1 définie dans le module **`ASN1-CHARACTER-MODULE`**, soit (lorsque le symbole graphique n'est pas ambigu dans le contexte de la spécification et du support utilisé pour le représenter) en indiquant le symbole graphique dans une chaîne "cstring" (le module **`ASN1-CHARACTER-MODULE`** est défini au § 38.1) ou en utilisant la notation "Quadruple" ou "Tuple" du § 37.8.

39.5 Pour les chaînes de type **`NumericString`**, l'ordre canonique est défini dans un ordre croissant de gauche à droite par (voir le Tableau 7 du § 37.2):

(espace) **`0 1 2 3 4 5 6 7 8 9`**

Le jeu de caractères complet contient exactement 11 caractères. L'extrémité d'un intervalle de valeur "ValueRange" (ou un caractère isolé) peut être désignée en indiquant le symbole graphique correspondant dans une chaîne "cstring".

NOTE – Cet ordre est le même que les caractères correspondants de la collection latine de base BASIC LATIN de l'ISO/CEI 10646-1.

39.6 Pour les chaînes de type **PrintableString**, l'ordre canonique est défini dans un ordre croissant de gauche à droite puis de haut en bas par (voir le Tableau 8 du § 37.4):

(ESPACE) (APOSTROPHE) (PARENTHESE GAUCHE) (PARENTHESE DROITE) (SIGNE PLUS) (VIRGULE) (TRAIT D'UNION – SIGNE MOINS) (POINT) (BARRE OBLIQUE) **0123456789** (DEUX-POINTS) (SIGNE EGAL) (POINT D'INTERROGATION)
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

Le jeu de caractères complet contient exactement 74 caractères. L'extrémité d'un intervalle de valeur "ValueRange" (ou un caractère isolé) peut être désignée en indiquant le symbole graphique correspondant dans une chaîne "cstring".

NOTE – Cet ordre est le même que les caractères correspondants de la collection latine de base "BASIC LATIN" de l'ISO/CEI 10646-1.

39.7 Pour les chaînes de type **VisibleString**, l'ordre canonique des cellules est défini à partir du codage ISO/CEI 646 (appelé ISO 646 ENCODING) de la manière suivante:

(ISO 646 ENCODING) - 32

NOTE – C'est-à-dire que l'ordre canonique est celui des caractères correspondants dans les cellules 2/0 à 7/14 de la grille de codage ISO/CEI 646.

Le jeu de caractères complet contient exactement 95 caractères. L'extrémité d'un intervalle de valeur "ValueRange" (ou un caractère isolé) peut être désignée en indiquant le symbole graphique correspondant dans une chaîne "cstring".

39.8 Pour les chaînes de type **IA5String**, l'ordre canonique des cellules est défini à partir du codage ISO/CEI 646 de la manière suivante:

(ISO 646 ENCODING)

Le jeu de caractères complet contient exactement 128 caractères. L'extrémité d'un intervalle de valeur "ValueRange" (ou un caractère isolé) peut être désignée en indiquant le symbole graphique correspondant dans une chaîne "cstring" ou une référence de valeur de caractère de contrôle ISO 646 définie au § 38.1.1.

40 Définition du type chaîne de caractères à alphabet non restreint

Le présent paragraphe définit un type dont les valeurs sont celles de n'importe quelle syntaxe abstraite de caractères. Dans un environnement OSI, cette syntaxe abstraite peut faire partie de l'ensemble de contextes définis OSI. Autrement, elle est indiquée en référence directement dans chaque instance d'utilisation du type chaîne de caractères à alphabet non restreint.

NOTE 1 – Une syntaxe abstraite de caractères (et une ou plusieurs syntaxes correspondantes de transfert de caractères) peut être définie par tout organisme habilité à attribuer des identificateurs d'objets **OBJECT IDENTIFIER** ASN.1.

NOTE 2 – Les profils établis par une communauté d'intérêt détermineront normalement les syntaxes abstraites de caractères et les syntaxes de transfert de caractères devant être prises en charge pour des instances ou groupes d'instances donnés de chaînes de caractères **CHARACTER STRING**. Dans les applications OSI, on indiquera généralement les syntaxes admises dans un formulaire de déclaration de conformité d'une instance de protocole OSI.

40.1 Le type chaîne de caractères à alphabet non restreint (voir § 3.6.76) est déclaré par la notation "UnrestrictedCharacterStringType":

UnrestrictedCharacterStringType ::= CHARACTER STRING

40.2 L'étiquette de ce type est le numéro 29 de la classe universelle.

40.3 Ce type est composé de valeurs représentant:

a) une chaîne de caractères qui peut être du type chaîne de caractères ASN.1 mais pas nécessairement;

b) l'identification (jointe ou séparée):

1) d'une syntaxe abstraite de caractères;

2) de la syntaxe de transfert de caractères.

40.4 Le type chaîne de caractères à alphabet non restreint possède un type associé qui sert à spécifier la notation de ses valeurs et sous-types.

40.5 Dans l'hypothèse d'un environnement d'étiquetage automatique, le type associé servant à la définition des valeurs et au sous-typage est le suivant (les commentaires ont force de norme):

```
SEQUENCE {
      identification                              CHOICE {
            syntaxes                                SEQUENCE {
                  abstract                            OBJECT IDENTIFIER,
                  transfer                            OBJECT IDENTIFIER }
                  -- Identificateurs d'objet de la syntaxe abstraite et de la
                  -- syntaxe de transfert --,

            syntax                                    OBJECT IDENTIFIER
                  -- Identificateur d'objet unique pour identifier la syntaxe
                  -- abstraite et la syntaxe de transfert --,
            presentation-context-id                   INTEGER
                  -- (Ne s'applique que dans les environnements OSI)
                  -- Le contexte de présentation OSI négocié identifie la syntaxe
                  -- abstraite et la syntaxe de transfert  --,

            context-negotiation                       SEQUENCE {
                  presentation-context-id               INTEGER,
                  transfer-syntax                       OBJECT IDENTIFIER }
            -- (Ne s'applique que dans les environnements OSI)
            -- Négociation de contexte en cours, presentation-context-id identifie
            -- uniquement la syntaxe abstraite. La syntaxe de transfert doit donc
            -- être spécifiée --,

            transfer-syntax                             OBJECT IDENTIFIER
            --  Le type de la valeur (une spécification indiquant par exemple qu'il
            --  s'agit d'une valeur d'un type ASN.1) est fixé par le concepteur de
            --  l'application (et est donc connu à la fois de l'expéditeur et du
            --  destinataire). Ce cas est prévu avant tout pour prendre en
            --  charge le chiffrement sélectif par champ (ou d'autres
            --  transformations de codage) d'un type ASN.1 --,

            fixed                                       NULL
            -- La valeur de donnée est une valeur d'un type ASN.1 fixe
            -- (qui est donc connu à la fois de l'expéditeur et du
            -- destinataire) -- },

      data-value-descriptor                     ObjectDescriptor  OPTIONAL
            -- Il s'agit de l'identification en langage naturel de la classe
            -- de la valeur --,
      string-value                              OCTET STRING }
      ( WITH COMPONENTS {
            ... ,
            data-value-descriptor  ABSENT } )
```

NOTE – Le type chaîne de caractères à alphabet non restreint ne permet pas d'inclure une valeur de descripteur **data-value-descriptor** en même temps qu'une **identification**. Toutefois, la définition du type associé donnée ici souligne les points communs existant entre le type valeur pdv imbriquée, le type externe et le type chaîne de caractères à alphabet non restreint.

40.6 Le texte du § 33.6 et du § 33.7 s'applique aussi au type chaîne de caractères à alphabet non restreint.

40.7 La notation de la valeur sera celle du type associé, et la valeur du composant **string-value** de type **OCTET STRING** représente un codage utilisant la syntaxe de transfert spécifiée dans **identification**.

UnrestrictedCharacterStringValue ::= SequenceValue *-- valeur du type associé défini au § 40.5*

XMLUnrestrictedCharacterStringValue ::=
XMLSequenceValue *-- valeur du type associé défini au § 40.5*

40.8 Un exemple du type chaîne de caractères à alphabet non restreint est donné au § E.2.8.

41 Notation des types définis dans les § 42 à 44

41.1 La notation permettant de faire référence à un type défini dans les § 42 à 44 est la suivante:

UsefulType ::= typereference

où "typereference" est l'une des références définies dans les § 42 à 44 en notation ASN.1.

41.2 L'étiquette de chaque type "UsefulType" est spécifiée dans les § 42 à 44.

42 Temps généralisé

42.1 Ce type est désigné par le nom:

`GeneralizedTime`

42.2 Ce type est composé de valeurs représentant:

a) une date calendaire, telle que celle-ci est définie dans l'ISO 8601;

b) une heure, à une des précisions définies dans l'ISO 8601, à l'exception de la valeur horaire 24 qui n'est pas utilisée;

c) le facteur de décalage horaire local, tel qu'il est défini dans l'ISO 8601.

42.3 Ce type est défini en ASN.1 comme suit:

```
GeneralizedTime ::=    [UNIVERSAL 24] IMPLICIT VisibleString
```

les valeurs du type **`VisibleString`** étant restreintes à l'une des chaînes de caractères suivantes:

a) une chaîne représentant la date calendaire selon les spécifications de l'ISO 8601, l'année étant représentée par quatre chiffres, le mois par deux chiffres et le jour par deux chiffres, sans caractères séparateurs, suivie d'une chaîne représentant l'heure selon les spécifications de l'ISO 8601, sans autre caractère séparateur que la virgule ou le point décimal, (comme prévu dans l'ISO 8601) et sans lettre finale Z (comme le prévoit l'ISO 8601);

b) les caractères de l'alinéa a) ci-dessus, suivis d'une lettre **`Z`** majuscule;

c) les caractères de l'alinéa a) ci-dessus, suivis d'une chaîne représentant un décalage horaire local selon les spécifications de l'ISO 8601, sans caractère séparateur.

Dans le cas a), l'heure correspond à l'heure locale. Dans le cas b), le temps représente l'heure universelle coordonnée (heure UTC). Dans le cas c), la partie de la chaîne formée comme dans le cas a) correspond à l'heure locale (t_1), le décalage horaire (t_2) permettant de déterminer l'heure UTC de la manière suivante:

temps universel coordonné (UTC) = $t_1 - t_2$

EXEMPLES:

Cas a)

"**`19851106210627.3`**"
heure locale 21 heures 6 minutes 27,3 secondes, le 6 novembre 1985.

Cas b)

"**`19851106210627.3Z`**"
temps universel coordonné correspondant à l'heure ci-dessus.

Cas c)

"**`19851106210627.3-0500`**"
même heure locale que dans le cas a), avec un retard local de 5 heures sur l'heure UTC.

42.4 L'étiquette de ce type est celle qui est définie au § 42.3.

42.5 Une valeur est déclarée par la notation de valeur du type **`VisibleString`** définie au § 42.3.

43 Temps universel

43.1 Ce type est désigné par le nom:

`UTCTime`

43.2 Le type est composé de valeurs représentant:

a) une date calendaire;

b) une heure, à la précision de la minute ou de la seconde;

c) et optionnellement, le décalage de l'heure locale par rapport au temps universel coordonné.

43.3 Le type est défini en ASN.1 de la manière suivante:

```
UTCTime ::= [UNIVERSAL 23] IMPLICIT VisibleString
```

les valeurs du type **VisibleString** étant limitées aux chaînes de caractères résultant de la juxtaposition des éléments suivants:

a) six chiffres AAMMJJ, où AA représente les deux chiffres de plus faible poids de l'année grégorienne, MM le mois (01 désignant le mois de janvier) et JJ le quantième du mois (de 01 à 31);

b) l'une des deux chaînes suivantes au choix:

1) quatre chiffres hhmm où hh représente l'heure (00 à 23) et mm les minutes (00 à 59);

2) six chiffres hhmmss où hh et mm sont comme en 1) ci-dessus, et ss représente les secondes (00 à 59);

c) l'une des deux chaînes suivantes au choix:

1) le caractère **Z**;

2) le caractère **+** ou le caractère **-**, suivi de hhmm, hh représentant les heures et mm les minutes.

Les deux formes de l'alinéa b) ci-dessus permettent de spécifier le temps avec différentes précisions.

Dans la forme c) 1), le temps représenté est le temps universel coordonné (UTC). Dans la forme c) 2), le temps t_1 indiqué par les chaînes a) et b) est l'heure locale; le décalage horaire (t_2) indiqué par la chaîne c) 2) permet de déterminer le temps UTC de la manière suivante:

$$\text{Temps universel coordonné} = t_1 - t_2$$

EXEMPLE 1 – Si le temps local est 7 heures du matin du 2 janvier 1982, et que le temps universel coordonné est midi du 2 janvier 1982, la valeur du temps **UTCTime** est représenté par l'une des chaînes suivantes:

– **"8201021200Z"**; ou
– **"8201020700-0500"**.

EXEMPLE 2 – Si le temps local est 7 heures du matin du 2 janvier 2001, et que le temps universel coordonné est midi du 2 janvier 2001, la valeur du temps **UTCTime** est représenté par l'une des chaînes suivantes:

– **"0101021200Z"**; ou
– **"0101020700-0500"**.

43.4 L'étiquette de ce type est celle qui est définie au § 43.3.

43.5 Une valeur est déclarée par la notation de valeur du type **VisibleString** définie au § 43.3.

44 Type descripteur d'objets

44.1 Ce type est désigné par le nom:

```
ObjectDescriptor
```

44.2 Ce type est constitué d'un texte en langage naturel décrivant un objet donné. Ce texte ne constitue pas une identification non ambiguë de l'objet, mais des objets différents ne devraient logiquement pas être décrits par un même texte.

NOTE – Il est recommandé que l'autorité affectant un identificateur **OBJECT IDENTIFIER** à un objet lui affecte également une valeur de descripteur d'objet **ObjectDescriptor**.

44.3 Ce type est défini en notation ASN.1 de la manière suivante:

```
ObjectDescriptor ::= [UNIVERSAL 7] IMPLICIT GraphicString
```

La chaîne graphique **GraphicString** contient le texte décrivant l'objet.

44.4 L'étiquette est celle qui est définie au § 44.3.

44.5 Une valeur est déclarée par la notation de valeur du type **GraphicString** définie au § 44.3.

45 Types contraints

45.1 La notation du type contraint "ConstrainedType" permet soit d'imposer une contrainte à un type (parent) pour en restreindre l'ensemble des valeurs à un sous-type donné, soit d'imposer, dans un type ensemble ou un type séquence,

des contraintes relationnelles entre composants qui s'appliquent aux valeurs du type parent et aux valeurs d'un autre composant du même type ensemble ou séquence. Elle permet aussi d'associer un identificateur d'exception à une contrainte.

ConstrainedType ::=
Type Constraint
| **TypeWithConstraint**

Dans la première forme, le type parent est "Type", et la contrainte est spécifiée par "Constraint", elle est définie au § 45.6. La seconde forme est définie au § 45.5.

45.2 Lorsqu'une contrainte est indiquée à la suite d'une notation de type ensemble-de ou séquence-de, elle s'applique au "Type" intérieur à la notation ensemble-de ou séquence-de, et non au type ensemble-de ou séquence-de lui-même.

NOTE – Dans la notation suivante par exemple, la contrainte de taille **(SIZE(1..64))** s'applique au type **VisibleString** et non au type **SEQUENCE OF**:

```
NamesOfMemberNations ::= SEQUENCE OF VisibleString (SIZE(1..64))
```

45.3 Lorsqu'une contrainte est indiquée à la suite de la notation de type sélection, elle s'applique au type choix et non au type de la forme sélectionnée. Une telle contrainte est ignorée (voir § 29.2).

NOTE – Dans l'exemple suivant, la contrainte **(WITH COMPONENTS {..., a ABSENT})** s'applique au type **CHOICE T**, et non au type **SEQUENCE** sélectionné, et n'a aucune incidence sur les valeurs de **V**.

```
T ::= CHOICE {
      a SEQUENCE {
            a INTEGER OPTIONAL,
            b BOOLEAN
      },
      b NULL
}

V ::= a < T (WITH COMPONENTS {..., a ABSENT})
```

45.4 Lorsqu'une contrainte est indiquée à la suite d'une notation de type étiqueté "TaggedType", l'interprétation de la notation globale est la même, que l'on considère le type étiqueté ou le type non étiqueté comme le type parent.

45.5 Par suite de l'interprétation donnée au § 45.2, une notation spéciale est prévue pour permettre d'appliquer une contrainte à un type ensemble-de ou séquence-de. Il s'agit de la contrainte "TypeWithConstraint":

TypeWithConstraint ::=
SET Constraint OF Type
| **SET SizeConstraint OF Type**
| **SEQUENCE Constraint OF Type**
| **SEQUENCE SizeConstraint OF Type**
| **SET Constraint OF NamedType**
| **SET SizeConstraint OF NamedType**
| **SEQUENCE Constraint OF NamedType**
| **SEQUENCE SizeConstraint OF NamedType**

Dans la première et la deuxième formes, le type parent est "**SET OF** Type", dans la troisième et la quatrième forme, c'est "**SEQUENCE OF** Type", dans la cinquième et la sixième forme, c'est "**SET OF** NamedType" et dans la septième et la huitième forme, c'est "**SEQUENCE OF** NamedType". Dans la première, la troisième, la cinquième et la septième forme, la contrainte est "Constraint" (voir § 45.6) et dans la deuxième, la quatrième, la sixième et la huitième forme, c'est "SizeConstraint" (voir § 47.5).

NOTE – Bien que la forme "Constraint" recouvre la forme "SizeConstraint", cette dernière forme est fournie pour des raisons historiques.

45.6 Une contrainte est spécifiée par la notation "Constraint":

Constraint ::= "(" ConstraintSpec ExceptionSpec ")"

ConstraintSpec ::=
SubtypeConstraint
| **GeneralConstraint**

La spécification d'exception "ExceptionSpec" est définie dans le § 49. Sauf si elle est utilisée conjointement à un "marqueur d'extension" (voir le § 48), elle n'apparaîtra que lorsqu'une occurrence de référence muette "DummyReference" (voir le § 8.3 de la Rec. UIT-T X.683 | ISO/CEI 8824-4) ou une contrainte définie par l'utilisateur

"UserDefinedConstraint" (voir le § 9 de la Rec. UIT-T X.682 | ISO/CEI 8824-3) figurera dans la spécification de contrainte "ConstraintSpec". La contrainte générale "GeneralConstraint" est définie au § 8.1 de la Rec. UIT-T X.682 | ISO/CEI 8824-3.

45.7 La notation "SubtypeConstraint" est la notation d'ordre général "ElementSetSpecs" (voir le § 46):

SubtypeConstraint ::= ElementSetSpecs

Dans ce contexte, les éléments sont des valeurs du type parent (le gouvernant de l'ensemble est le type parent). L'ensemble comprend au moins un élément.

46 Spécification d'un ensemble d'éléments

46.1 Dans certaines notations, on peut spécifier un ensemble d'éléments d'un certain type ou d'une certaine classe d'objets informationnels (le gouverneur). A cette fin, on utilise la notation "ElementSetSpec":

```
ElementSetSpecs ::=
    RootElementSetSpec
    |   RootElementSetSpec "," "..."
    |   RootElementSetSpec "," "..." "," AdditionalElementSetSpec

RootElementSetSpec ::= ElementSetSpec

AdditionalElementSetSpec ::= ElementSetSpec

ElementSetSpec ::= Unions
    |   ALL Exclusions

Unions ::= Intersections
    |   UElems UnionMark Intersections

UElems ::= Unions

Intersections ::= IntersectionElements
    |   IElems IntersectionMark IntersectionElements

IElems ::= Intersections

IntersectionElements ::= Elements | Elems Exclusions

Elems ::= Elements

Exclusions ::= EXCEPT Elements

UnionMark ::= "|" | UNION

IntersectionMark ::= "^" | INTERSECTION
```

NOTE 1 – Le symbole accent circonflexe "^" et le mot **INTERSECTION** sont synonymes. Le symbole barre "|" et le mot **UNION** sont synonymes. Dans un souci de cohérence stylistique, il est recommandé d'utiliser la notation soit par symboles soit par mots dans l'ensemble de la spécification. La déclaration **EXCEPT** peut être utilisée avec l'une ou l'autre notation.

NOTE 2 – L'ordre de priorité décroissant des opérateurs est: **EXCEPT**, "^", "|". A noter que la déclaration **ALL EXCEPT** est spécifiée de telle manière qu'elle ne puisse être insérée au milieu d'autres contraintes sans que l'expression " **ALL EXCEPT** xxx" ne figure entre parenthèses.

NOTE 3 – A chaque occurrence "Elements" peut correspondre soit à une contrainte non parenthésée [par exemple **INTEGER (1..4)**] soit à une contrainte parenthésée de sous-typage [par exemple **INTEGER ((1..4 | 9))**].

NOTE 4 – A noter que deux opérateurs **EXCEPT** seront toujours séparés par un signe "|", "^", "**(**" ou "**)**". Ainsi, l'expression **(A EXCEPT B EXCEPT C)** est incorrecte; elle devrait être remplacée par **((A EXCEPT B) EXCEPT C)** ou par **(A EXCEPT (B EXCEPT C))**.

NOTE 5 – A noter que l'expression **((A EXCEPT B) EXCEPT C)** est équivalente à **(A EXCEPT (B | C))**.

NOTE 6 – Les éléments auxquels fait référence la spécification "ElementSetSpecs" correspondent à la réunion des éléments auxquels font référence les spécifications "RootElementSetSpec" et "AdditionalElementSetSpec" (lorsqu'elles sont présentes).

NOTE 7 – Lorsque les éléments sont des objets informationnels (autrement dit le gouverneur est une classe d'objets informationnels), la notation "ObjectSetElements" définie dans la Rec. UIT-T X.681 | ISO/CEI 8824-2, § 12.3, est utilisée.

46.2 Les éléments composant l'ensemble sont:

a) si la première forme de "ElementSetSpec" est utilisée, les éléments spécifiés par la notation "Union" [voir alinéa b)], sinon les éléments du gouverneur à l'exception des éléments spécifiés par la notation "Elements" de l'expression "Exclusions";

b) si la première forme de "Unions" est utilisée, les éléments spécifiés par la notation "Intersections" [voir alinéa c)], sinon les éléments figurant au moins une fois dans "UElems" ou dans "Intersections";

c) si la première forme de "Intersections" est utilisée, les éléments spécifiés dans "IntersectionElements" [voir alinéa d)], sinon les éléments spécifiés à la fois dans "IElems" et dans "IntersectionElements";

d) si la première forme de "IntersectionElements" est utilisée, les éléments spécifiés dans "Elements", sinon les éléments spécifiés dans "Elems" à l'exception de ceux spécifiés dans "Exclusions".

46.3 L'ensemble des valeurs est défini comme étant extensible si les conditions suivantes sont satisfaites:

a) pour "Elements": il existe un marquer d'extension au niveau extérieur;

NOTE – Ceci s'applique même si toutes les valeurs du parent sont incluses dans la racine du nouveau type contraint.

b) pour "Unions": au moins un des "UElems" est extensible;

c) pour "Intersections": au moins un des "IElems" est extensible;

d) pour "Exclusions": l'ensemble des éléments précédant **`EXCEPT`** est extensible.

Dans les autres cas, l'ensemble des valeurs n'est pas extensible (voir aussi § G.4).

46.4 Si l'ensemble des valeurs est extensible, on peut déterminer les valeurs de la racine en réalisant les opérations arithmétiques uniquement sur les valeurs de la racine des ensembles de valeurs intervenant dans ces opérations arithmétiques, comme spécifiés au § 46.2. Pour déterminer les ajouts d'extension, on peut réaliser les opérations arithmétiques sur les valeurs de la racine complétées par les ajouts d'extension des différents ensembles de valeurs intervenant dans ces opérations arithmétiques, puis exclure les valeurs qui ont été déterminées comme étant des valeurs de la racine.

46.5 La notation "Elements" est définie comme suit:

```
Elements ::=
        SubtypeElements
    |   ObjectSetElements
    |   "(" ElementSetSpec ")"
```

Les éléments spécifiés par cette notation sont:

a) comme décrit dans le § 47 ci-dessous si l'expression utilise la forme "SubtypeElements". Cette notation ne sera utilisée que lorsque le gouverneur est un type, et que le type concerné doit servir à imposer des contraintes supplémentaires aux possibilités de la notation. Dans un tel contexte, le gouverneur est considéré comme le type parent;

b) ceux décrits dans la Rec. UIT-T X.681 | ISO/CEI 8824-2, § 12.10, si l'expression utilise la forme "ObjectSetElements". Cette notation ne sera utilisée que lorsque le gouverneur est une classe d'objets informationnels;

c) les éléments spécifiés par la notation "ElementSetSpec" si la troisième forme est utilisée.

46.6 Quand on procède à des opérations arithmétiques sur des ensembles avec une contrainte de sous-typage ou un ensemble de valeurs dont le type gouvernant est extensible, seules les valeurs abstraites figurant dans la racine d'extension du type gouvernant sont utilisées dans ces opérations. Dans ce cas, toutes les instances de notation de valeur (y compris les références de valeur) utilisées dans les opérations arithmétiques sont nécessaires pour faire référence à une valeur abstraite de la racine d'extension du type gouvernant. Les extrémités d'une contrainte d'intervalle doivent faire référence aux valeurs qui sont présentes dans la racine d'extension du type gouvernant et la spécification d'intervalle proprement dite fait référence à toutes les valeurs de l'intervalle qui se trouvent dans la racine d'extension du type gouvernant (et uniquement à ces valeurs).

46.7 Quand on procède à des opérations arithmétiques sur des ensembles d'objets informationnels, tous les objets informationnels sont utilisés dans ces opérations. Si l'un des ensembles d'objets informationnels utilisés est extensible, ou s'il y a un marqueur d'extension à l'extérieur d'une production "ElementSetSpecs", le résultat des opérations arithmétiques est extensible.

46.8 Si une contrainte de sous-typage est appliquée en série à un type parent qui est extensible par le biais de l'application d'une contrainte extensible, la notation de valeur qui y est utilisée ne doit pas faire référence à des valeurs qui ne se trouvent pas dans la racine d'extension du type parent. Le résultat de la deuxième contrainte (appliquée en série) est défini comme étant le même que le résultat qui serait obtenu si la contrainte avait été appliquée au type parent sans son marqueur d'extension et les éventuels ajouts d'extension.

EXEMPLE

```
Foo ::= INTEGER ( 1..6, ..., 73..80)
Bar ::= Foo (73) -- illégal
foo Foo ::= 73 -- légal car c'est une notation de valeur pour Foo,
               -- ne faisant pas partie d'une contrainte
```

Bar est illégal car 73 ne se trouve pas dans la racine d'extension de **Foo**. Si 73 avait été dans la racine d'extension de **Foo**, l'exemple aurait été légal et **Bar** aurait contenu la valeur unique 73.

47 Eléments de sous-typage

47.1 Généralités

Plusieurs formes sont prévues pour noter les éléments de sous-typage "SubtypeElements". Elles sont identifiées ci-dessous, leur syntaxe et leur sémantique étant définies dans les paragraphes qui suivent. Le Tableau 9 récapitule pour chaque type parent les notations qui peuvent lui être appliquées.

SubtypeElements ::=
SingleValue
| **ContainedSubtype**
| **ValueRange**
| **PermittedAlphabet**
| **SizeConstraint**
| **TypeConstraint**
| **InnerTypeConstraints**
| **PatternConstraint**

Tableau 9 – Applicabilité de la notation de sous-typage selon le type parent

Type (ou dérivé du type par étiquetage ou sous-typage)	Valeur unique (Single value)	Sous-type contenu (Contained subtype)	Intervalle de valeurs (Value range)	Contrainte de taille (Size constraint)	Alphabet permis (Permitted alphabet)	Contrainte de type (Type constraint)	Sous-typage interne (Inner subtyping)	Contrainte de structure (Pattern constraint)
Chaîne binaire (Bit string)	Oui	Oui	Non	Oui	Non	Non	Non	Non
Booléen (Boolean)	Oui	Oui	Non	Non	Non	Non	Non	Non
Choix (Choice)	Oui	Oui	Non	Non	Non	Non	Oui	Non
Valeur pdv imbriquée (Embedded-pdv)	Oui	Non	Non	Non	Non	Non	Oui	Non
Enuméré (Enumerated)	Oui	Oui	Non	Non	Non	Non	Non	Non
Externe (External)	Oui	Non	Non	Non	Non	Non	Oui	Non
Instance-de (Instance-of)	Oui	Oui	Non	Non	Non	Non	Oui	Non
Entier (Integer)	Oui	Oui	Oui	Non	Non	Non	Non	Non
Néant (Null)	Oui	Oui	Non	Non	Non	Non	Non	Non
Type champ de classe d'objets (Object class field type)	Oui	Oui	Non	Non	Non	Non	Non	Non
Descripteur d'objet (Object descriptor)	Oui	Oui	Non	Oui	Oui	Non	Non	Non
Identificateur d'objet (Object identifier)	Oui	Oui	Non	Non	Non	Non	Non	Non
Chaîne d'octets (Octet string)	Oui	Oui	Non	Oui	Non	Non	Non	Non

Tableau 9 – Applicabilité de la notation de sous-typage selon le type parent

Type (ou dérivé du type par étiquetage ou sous-typage)	Valeur unique (Single value)	Sous-type contenu (Contained subtype)	Intervalle de valeurs (Value range)	Contrainte de taille (Size constraint)	Alphabet permis (Permitted alphabet)	Contrainte de type (Type constraint)	Sous-typage interne (Inner subtyping)	Contrainte de structure (Pattern constraint)
Type ouvert (open type)	Non	Non	Non	Non	Non	Oui	Non	Non
Réel (Real)	Oui	Oui	Oui	Non	Non	Non	Oui	Non
Identificateur d'objet relatif (Relative object identifier)	Oui[b)]	Oui[b)]	Non	Non	Non	Non	Non	Non
Types chaîne de caractères à alphabet restreint (Restricted character string types)	Oui	Oui	Oui[a)]	Oui	Oui	Non	Non	Oui
Séquence (Sequence)	Oui	Oui	Non	Non	Non	Non	Oui	Non
Séquence-de (Sequence-of)	Oui	Oui	Non	Oui	Non	Non	Oui	Non
Ensemble (Set)	Oui	Oui	Non	Non	Non	Non	Oui	Non
Ensemble-de (Set-of)	Oui	Oui	Non	Oui	Non	Non	Oui	Non
Types temps (Time types)	Oui	Oui	Non	Non	Non	Non	Non	Non
Type de chaîne de caractères à alphabet non restreint (Unrestricted character string type)	Oui	Non	Non	Oui	Non	Non	Oui	Non

a) Autorisé seulement avec l'alphabet permis "PermittedAlphabet" des types **`BMPString`**, **`IA5String`**, **`NumericString`**, **`PrintableString`**, **`VisibleString`**, **`UTF8String`** et **`UniversalString`**.

b) Le nœud de départ pour tous les types d'identificateur d'objet relatif ou les valeurs intervenant dans les contraintes ou les ensembles de valeurs doit être identique au nœud de départ utilisé pour le gouverneur.

47.2 Valeur unique

47.2.1 La notation de valeur unique "SingleValue" est la suivante:

SingleValue ::= Value

où "Value" est la notation de la valeur du type parent.

47.2.2 La notation "SingleValue" spécifie la valeur unique du type parent spécifiée par "Value".

47.3 Sous-type contenu

47.3.1 La notation de sous-type contenu "ContainedSubtype" est la suivante:

ContainedSubtype ::= Includes Type

Includes ::= `INCLUDES` | empty

La forme vide "empty" de la production "Includes" ne sera pas utilisée si le "Type" du sous-type contenu "ContainedSubtype" est la notation du type néant.

47.3.2 La notation "ContainedSubtype" spécifie toutes les valeurs de la racine du type parent qui sont aussi dans la racine du "Type". Le "Type" doit dériver du même type prédéfini que le type parent.

47.3.3 L'ensemble des valeurs auxquelles fait référence un "Type" extensible utilisé dans une contrainte de sous-type contenu n'hérite pas du marqueur d'extension du "Type". Toutes les valeurs du "Type" qui ne sont pas dans la racine d'extension de ce type sont ignorées et ne font pas partie des valeurs du type contraint.

NOTE – Ce n'est pas parce qu'on utilise un "Type" extensible que le type constraint est lui aussi extensible.

47.4 Intervalle de valeurs

47.4.1 La notation d'intervalle de valeurs "ValueRange" est la suivante:

ValueRange ::= LowerEndpoint " . . " UpperEndpoint

47.4.2 La notation "ValueRange" spécifie les valeurs d'un intervalle désigné en spécifiant les valeurs de ses extrémités. Cette notation ne peut être utilisée qu'avec le type entier, l'alphabet "PermittedAlphabet" de certains types de chaînes de caractères à alphabet restreint (**IA5String**, **NumericString**, **PrintableString**, **VisibleString**, **BMPString**, **UniversalString** et **UTF8String** seulement) et le type réel. Toutes les valeurs spécifiées dans l'intervalle "ValueRange" doivent se trouver dans la racine du type parent.

NOTE – Pour les besoins du sous-typage, **PLUS-INFINITY** est supérieur à toute valeur réelle et **MINUS-INFINITY** est inférieur à toute valeur réelle.

47.4.3 Chaque extrémité de l'intervalle peut être soit autorisée (auquel cas elle appartient à l'intervalle), soit non autorisée (auquel cas elle n'appartient pas à l'intervalle). Quand l'extrémité n'appartient pas à l'intervalle, la définition inclut un symbole inférieur-à "<":

LowerEndpoint ::= LowerEndValue | LowerEndValue "<"

UpperEndpoint ::= UpperEndValue | "<" UpperEndValue

47.4.4 Une extrémité peut également ne pas être spécifiée, auquel cas l'intervalle s'étend dans ce sens aussi loin que le type parent l'autorise:

LowerEndValue ::= Value | MIN

UpperEndValue ::= Value | MAX

NOTE – Lorsqu'une construction "ValueRange" est utilisée dans une contrainte de limitation d'alphabet "PermittedAlphabet", les valeurs "LowerEndValue" et "UpperEndValue" doivent être de taille 1.

47.5 Contrainte de taille

47.5.1 La notation de contrainte de taille "SizeConstraint" est la suivante:

SizeConstraint ::= SIZE Constraint

47.5.2 Une contrainte de taille "SizeConstraint" ne peut s'appliquer qu'aux types chaîne binaire, chaîne d'octets, chaîne de caractères, ensemble-de ou séquence-de.

47.5.3 La contrainte "Constraint" spécifie les valeurs entières autorisées pour la longueur des valeurs spécifiées; elle a la forme de n'importe quelle contrainte pouvant être appliquée au type parent suivant:

```
INTEGER (0 .. MAX)
```

La contrainte "Constraint" sera exprimée au moyen de la forme "SubtypeConstraint" de la spécification "ConstraintSpec".

47.5.4 L'unité de mesure dépend du type parent, comme suit:

Type	*Unité de mesure*
chaîne binaire	bit
chaîne d'octets	octet
chaîne de caractères	caractère
ensemble-de	valeur de composant
séquence-de	valeur de composant

NOTE – On distinguera clairement le décompte des caractères utilisé dans le présent paragraphe pour déterminer la taille des chaînes de caractères, d'un quelconque décompte d'octets. Le décompte des caractères sera interprété selon la définition de la collection de caractères utilisée dans le type, et plus particulièrement par rapport aux normes, grilles ou numéros d'enregistrement dans un registre pouvant apparaître dans une telle définition.

47.6 Contrainte de type

47.6.1 La notation de contrainte de type "TypeConstraint" est la suivante:

TypeConstraint ::= Type

47.6.2 Cette notation ne s'applique qu'à la notation d'un type ouvert et restreint ce type aux valeurs du type "Type".

47.7 Alphabet permis

47.7.1 La notation d'alphabet permis "PermittedAlphabet" est la suivante:

PermittedAlphabet ::= `FROM` Constraint

47.7.2 La notation "PermittedAlphabet" spécifie toutes les valeurs qui peuvent être obtenues en utilisant un sous-alphabet de la chaîne parent. Cette notation ne s'applique qu'aux types chaînes de caractères à alphabet restreint.

47.7.3 La contrainte "Constraint" est n'importe quelle contrainte pouvant s'appliquer au type parent (voir le Tableau 9), sauf qu'elle doit utiliser la forme "SubtypeConstraint" de la production "ConstraintSpec". Le sous-alphabet inclut tous les caractères qui figurent dans une ou plusieurs valeurs du type chaîne parent et qui sont autorisés par la contrainte "Constraint" et seulement ces caractères.

47.7.4 Si la contrainte "Constraint" est extensible, l'ensemble de valeurs sélectionné par la contrainte de l'alphabet permis est extensible. L'ensemble des valeurs de la racine correspond aux valeurs permises par la racine de la contrainte "Constraint" et les ajouts d'extension correspondent aux valeurs permises par la racine conjointement avec les ajouts d'extension de la contrainte "Constraint", à l'exception des valeurs se trouvant déjà dans la racine.

47.8 Sous-typage interne

47.8.1 La notation de contrainte de type interne "InnerTypeConstraints" est la suivante:

InnerTypeConstraints ::=
`WITH COMPONENT` SingleTypeConstraint
| `WITH COMPONENTS` MultipleTypeConstraints

48.8.2 La notation "InnerTypeConstraints" spécifie uniquement les valeurs qui satisfont à un ensemble de contraintes portant sur la présence et la valeur des composants du type parent. Une valeur du type parent ne satisfait la contrainte que si elle satisfait à toutes les contraintes explicites ou implicites (voir § 47.8.6). Cette notation peut être appliquée aux types ensemble-de, séquence-de, ensemble, séquence et choix.

NOTE – Une notation "InnerTypeConstraints" appliquée à un type ensemble ou séquence n'est pas prise en considération par la transformation **`COMPONENTS OF`** (voir les § 24.4 et 26.2).

47.8.3 Une contrainte ayant la forme d'une spécification de valeur de sous-type est prévue pour les types (ensemble-de, séquence-de) qui sont définis en termes d'un seul autre type (interne). La notation de cette contrainte de type unique "SingleTypeConstraint" est la suivante:

SingleTypeConstraint ::= Constraint

La contrainte "Constraint" définit un sous-type du type unique (interne). Une valeur du type parent satisfait la contrainte si et seulement si chaque valeur interne appartient au sous-type obtenu en appliquant la contrainte au type interne.

47.8.4 Lorsqu'un type (choix, ensemble, séquence) est défini en termes de plusieurs autres types (internes), plusieurs contraintes peuvent s'appliquer aux types internes qui le composent. La notation de ces contraintes de types multiples "MultipleTypeConstraints" est la suivante:

MultipleTypeConstraints ::=
FullSpecification
| PartialSpecification

FullSpecification ::= "{" TypeConstraints "}"

PartialSpecification ::= "{" "..." "," TypeConstraints "}"

TypeConstraints ::=
NamedConstraint
| NamedConstraint "," TypeConstraints

NamedConstraint ::=
identifier ComponentConstraint

47.8.5 Les contraintes "TypeConstraints" contiennent une liste de contraintes s'appliquant aux types composants du type parent. Pour un type séquence, les contraintes doivent figurer dans l'ordre des composants de la séquence. Le type interne auquel les contraintes s'appliquent est repéré par son identificateur. Il y aura au plus une contrainte nommée "NamedConstraint" par composant.

47.8.6 Les contraintes "MultipleTypeConstraints" comprennent soit une spécification complète "FullSpecification", soit une spécification partielle "PartialSpecification". Lorsqu'une spécification complète "FullSpecification" est fournie, il existe une contrainte de présence implicite selon laquelle la déclaration **ABSENT** est apposée à tous les types internes qui peuvent recevoir une telle contrainte (voir § 47.8.9) et qui ne sont pas explicitement énumérés. Lorsqu'une spécification partielle "PartialSpecification" est fournie, il n'y a pas de contrainte implicite, et tout type interne peut être omis de la liste.

47.8.7 Les contraintes s'appliquant à un type interne donné peuvent concerner sa présence (exprimée à partir du type parent), ses valeurs, ou les deux. La notation de la contrainte de composant "ComponentConstraint" est la suivante:

ComponentConstraint ::= ValueConstraint PresenceConstraint

47.8.8 Une contrainte sur la valeur d'un type interne est définie par la notation "ValueConstraint":

ValueConstraint ::= Constraint | empty

La contrainte est satisfaite par une valeur du type parent si et seulement si la valeur interne appartient au sous-type spécifié par la contrainte "Constraint" appliquée au type interne.

47.8.9 Une contrainte portant sur la présence d'un type interne est exprimée par la notation "PresenceConstraint":

PresenceConstraint ::= PRESENT | ABSENT | OPTIONAL | empty

La signification de ces formes et les situations dans lesquelles elles sont autorisées sont définies du § 47.8.9.1 au § 47.8.9.3.

47.8.9.1 Si le type parent est une séquence ou un ensemble, un type composant déclaré **OPTIONAL** peut être contraint par la déclaration **PRESENT** (auquel cas la contrainte est satisfaite si et seulement si la valeur du composant correspondant est présent), par la déclaration **ABSENT** (auquel cas la contrainte est satisfaite si et seulement si la valeur du composant correspondant est absente) ou par la déclaration **OPTIONAL** (auquel cas aucune contrainte n'est imposée quant à la présence de la valeur du composant correspondant).

47.8.9.2 Si le type parent est un choix, un type composant peut être contraint par la déclaration **ABSENT** (auquel cas la contrainte est satisfaite si et seulement si le type composant correspondant n'est pas utilisé dans la valeur), ou par la déclaration **PRESENT** (auquel cas la contrainte est satisfaite si et seulement si le type composant correspondant figure dans la valeur); il y aura au plus un mot-clé **PRESENT** dans une notation "MultipleTypeConstraints".

NOTE – Le paragraphe E.4.6 fournit un exemple illustrant ce texte.

47.8.9.3 La signification d'une contrainte de présence "PresenceConstraint" vide dépend de la question de savoir si une spécification "FullSpecification" ou "PartialSpecification" est employée:

a) dans une spécification complète "FullSpecification", elle est équivalente à la contrainte de présence **PRESENT** pour un composant d'ensemble ou de séquence déclarée **OPTIONAL** et n'impose autrement aucune autre contrainte;

b) dans une spécification partielle "PartialSpecification", aucune contrainte n'est imposée.

47.9 Contrainte de structure

47.9.1 La notation "PatternConstraint" est la suivante:

PresenceConstraint ::= PRESENT | ABSENT | OPTIONAL | empty

47.9.2 L'élément "Value" doit être une chaîne "cstring" de type **UniversalString** (ou une référence à une telle chaîne de caractères) contenant une expression régulière ASN.1 comme défini dans l'Annexe A. La notation "PatternConstraint" sélectionne les valeurs du type parent qui satisfont à l'expression régulière ASN.1. La valeur entière doit satisfaire à l'expression régulière ASN.1 entière, c'est-à-dire que "PatternConstraint" ne sélectionne pas les valeurs dont les caractères de début répondent à l'expression régulière ASN.1 (entière) mais qui contiennent d'autres caractères de fin.

NOTE – La valeur "Value" est défini formellement comme étant une valeur de type **UniversalString**; mais les ensembles de valeurs de type **UniversalString** et **UTF8String** sont les mêmes (voir § 37.16). Une définition totalement équivalente aurait donc pu consister à énoncer que "Value" est une valeur de type **UTF8String**.

48 Marqueur d'extension

NOTE – Comme c'est le cas d'une manière générale pour la notation de contrainte, le marqueur d'extension n'a aucun effet sur certaines règles de codage de la notation ASN.1, telles que les règles de codage de base, mais en a sur certaines autres, telles que les règles de codage compact. Son effet sur les codages définis au moyen de la notation ECN est déterminé par la spécification de la notation ECN.

48.1 Le marqueur d'extension, représenté par des points de suspension, est une indication que des ajouts d'extension sont attendus. Il ne fait aucune déclaration au sujet de la manière de traiter de tels ajouts, à part le fait qu'ils ne seront pas traités comme des erreurs lors du processus de décodage.

48.2 L'utilisation conjointe du marqueur d'extension et d'un identificateur d'exception (voir § 49) permet à la fois d'indiquer que des ajouts d'extension sont attendus et de fournir un moyen d'identifier l'action que l'application doit effectuer si une contrainte est transgressée. L'utilisation de cette notation est recommandée pour des situations dans lesquelles un stockage avec retransmission ou toute autre forme de relais est utilisé, de manière à indiquer (par exemple) que tout ajout d'extension non reconnu doit être renvoyé à l'application pour un nouveau codage et un relais éventuel.

48.3 Le résultat d'opérations arithmétiques sur des ensembles impliquant des contraintes de sous-typage, des ensembles de valeurs ou des ensembles d'objets informationnels qui sont extensibles, est décrit au § 46.

48.4 Si un type défini avec une contrainte extensible est référencé dans un sous-type "ContainedSubtype", le type nouvellement défini n'hérite ni du marqueur d'extension ni d'aucun des ajouts d'extension (voir § 47.3.3). Le type nouvellement défini peut être rendu extensible par l'inclusion d'un marqueur d'extension au niveau extérieur dans ses spécifications "ElementSetSpecs" (voir aussi § 46.3). A titre d'exemple:

```
A ::= INTEGER (0..10, ..., 12) -- A est extensible.
B ::= INTEGER (A)              -- B n'est pas extensible et
                               -- limité à l'intervalle 0-10.
C ::= INTEGER (A, …)           -- C est extensible et limité
                               -- à l'intervalle 0-10.
```

48.5 Si un type défini avec une contrainte extensible est soumis en plus à une contrainte avec une spécification "ElementSetSpecs", le type résultant n'hérite ni du marqueur d'extension ni d'aucun des ajouts d'extension pouvant être présents dans la première contrainte (voir § 46.8). A titre d'exemple:

```
A ::= INTEGER (0..10, ...)  -- A est extensible.
B ::= A (2..5)              -- B n'est pas extensible.
C ::= A                     -- C est extensible.
```

48.6 Les composants d'un type ensemble, séquence ou choix qui sont soumis à la contrainte d'être absents ne pourront pas être présents, que le type ensemble, séquence ou choix soit extensible ou non.

NOTE – Des contraintes de type interne n'ont aucun effet sur l'extensibilité.

Par exemple:

```
A ::= SEQUENCE   {
a     INTEGER
b     BOOLEAN OPTIONAL,
...
}

B ::= A (WITH COMPONENTS {b ABSENT})
                              -- B est extensible, mais "b" ne
                              -- figurera dans aucune de ses valeurs.
```

48.7 Les transformations suivantes doivent être appliquées d'une manière conceptuelle avant d'effectuer la vérification d'unicité des étiquettes lorsque la présente Recommandation | Norme internationale prescrit des étiquettes distinctes (voir § 24.5 et § 24.6, § 26.3 et § 28.3).

48.7.1 Un nouvel élément ou une nouvelle forme (appelé élément ajouté conceptuellement, voir § 48.7.2) est ajouté conceptuellement au point d'insertion d'extension dans l'un des cas suivants:

a) il n'existe pas de marqueurs d'extension, mais l'extensibilité est implicitement prévue dans l'en-tête du module, puis un marqueur d'extension est ajouté et le nouvel élément est inséré comme premier ajout après ce marqueur d'extension;

b) il existe un seul marqueur d'extension dans une notation **CHOICE**, **SEQUENCE** ou **SET** et le nouvel élément est ajouté à la fin de la notation en question immédiatement avant l'accolade fermante;

c) il existe deux marqueurs d'extension dans une notation **CHOICE**, **SEQUENCE** ou **SET** et le nouvel élément est ajouté immédiatement avant le deuxième marqueur d'extension.

48.7.2 L'élément ajouté conceptuellement existe uniquement à des fins de vérification de légalité par application des règles imposant des étiquettes distinctes (voir § 24.5 et § 24.6, § 26.3 et § 28.3). Il est ajouté conceptuellement *après* l'application éventuelle de l'étiquetage automatique et le développement de **COMPONENTS OF**.

48.7.3 L'élément ajouté conceptuellement est défini comme ayant une étiquette distincte de l'étiquette de tout type ASN.1 normal, mais qui correspond à l'étiquette de tous les éléments ajoutés conceptuellement ainsi qu'à l'étiquette indéterminée du type ouvert, comme le spécifie la Rec. UIT-T X.681 | ISO/CEI 8824-2, § 14.2, Note 2.

NOTE – En ce qui concerne les éléments conceptuellement ajoutés et le type ouvert, les règles relatives à l'unicité de l'étiquette ainsi que les règles imposant des étiquettes distinctes (voir § 24.5 et § 24.6, § 26.3 et § 28.3) sont nécessaires et suffisantes pour garantir:

a) que tout ajout d'extension inconnu puisse être attribué sans ambiguïté à un seul point d'insertion lors du décodage d'un élément codé selon les règles de codage de base;

b) que les ajouts d'extension inconnus ne pourront jamais être confondus avec des éléments **OPTIONAL**.

Dans les codages compacts, les règles ci-dessous sont suffisantes mais ne sont pas nécessaires pour garantir ces propriétés. Elles sont néanmoins imposées comme règles ASN.1 pour garantir l'indépendance de la notation par rapport aux règles de codage.

48.7.4 La spécification utilise d'une manière illégale la notation d'extensibilité si les éléments ajoutés conceptuellement transgressent les règles imposant des types distincts.

NOTE – Les règles précédentes ont pour but de fournir des restrictions précises au sujet de l'utilisation des points d'insertion (en particulier pour ceux qui ne sont pas situés à la fin de types **SEQUENCE**, **SET** ou **CHOICE**). Ces restrictions ont été conçues afin de garantir qu'il est possible d'affecter sans ambiguïté un élément inconnu reçu par un système en version 1 à un point d'insertion spécifique lors de l'utilisation des règles de codage BER, DER et CER. Ce point est important si le traitement d'exception de tels éléments ajoutés n'est pas le même pour différents points d'insertion.

48.8 Exemples

48.8.1 Exemple 1

```
A ::= SET {
      a     A,
      b     CHOICE {
            c     C,
            d     D,
            ...
      }
}
```

est légal, car il n'existe pas d'ambiguïté sur le fait que toute information ajoutée doit faire partie de **b**.

48.8.2 Exemple 2

```
A ::= SET {
      a     A,
      b     CHOICE {
            c     C,
            d     D,
            ...
      },
      ... ,
      d     D
}
```

est illégal, car les informations ajoutées peuvent faire partie de **b** ou peuvent être à l'extérieur de **A**; un système en version 1 ne peut pas lever l'ambiguïté.

48.8.3 Exemple 3

```
A ::= SET {
      a     A,
      b     CHOICE {
            c     C,
            ...
      } ,
      d     CHOICE {
            e     E,
            ...
      }
}
```

est également illégal, car les informations ajoutées peuvent appartenir à **b** ou **d**.

48.8.4 Il est possible de construire des exemples plus complexes, avec des choix extensibles à l'intérieur d'autres choix extensibles ou dans des d'éléments d'une séquence marquée **OPTIONAL** ou **DEFAULT**; les règles précédentes sont toutefois nécessaires et suffisantes pour garantir qu'un élément qui n'est pas présent dans la version 1 peut être attribué sans ambiguïté à un point d'insertion et un seul par un système en version 1.

49 Identificateur d'exception

49.1 Dans une spécification ASN.1 complexe, il existe plusieurs situations dans lesquelles il est admis que les décodeurs auront à traiter des éléments incomplètement spécifiés. Ces situations résultent notamment de l'utilisation d'une contrainte définie au moyen d'un paramètre de la syntaxe abstraite (voir la Rec. UIT-T X.683 | ISO/CEI 8824-4, § 10).

49.2 Dans de telles situations, les concepteurs d'applications doivent déterminer les mesures à prendre lorsqu'une contrainte donnée dépendant de l'application est transgressée. L'identificateur d'exception est prévu comme un moyen non ambigu de faire référence à des éléments de spécification ASN.1 indiquant les actions à exécuter dans ces cas. L'identificateur est constitué du caractère "**!**", suivi d'un type ASN.1 optionnel et d'une valeur de ce type. Si le type optionnel n'est pas mentionné, la valeur est supposée être du type entier **INTEGER**.

49.3 Si une spécification d'exception "ExceptionSpec" est présente, cela signifie qu'il existe un texte dans le corps de la norme disant comment traiter la transgression de contrainte associé au caractère "**!**". En l'absence d'un telle spécification et si une transgression de contrainte a lieu, les concepteurs devront soit identifier le texte décrivant les mesures à prendre, soit décider de ces mesures en fonction de l'application.

49.4 La notation de spécification d'exception "ExceptionSpec" est définie comme suit:

```
ExceptionSpec ::= "!"  ExceptionIdentification | empty

ExceptionIdentification ::=
      SignedNumber
|     DefinedValue
|     Type ":" Value
```

Les deux premières formes correspondent à des identificateurs d'exception de type entier. La troisième correspond à un identificateur d'exception "Value" de type arbitraire "Type".

49.5 Lorsqu'un type est soumis à des contraintes multiples dont plus d'une possède un identificateur d'exception, l'identificateur d'exception de la contrainte la plus externe sera considéré comme l'identificateur d'exception pour ce type.

49.6 L'identificateur d'exception est ignoré et n'est pas hérité par le type qui est soumis à une contrainte résultant d'opérations arithmétiques sur des ensembles, lorsqu'un marqueur d'exception est présent pour des types utilisés dans de telles opérations.

Annexe A

Expressions régulières en notation ASN.1

(Cette annexe fait partie intégrante de la présente Recommandation | Norme internationale)

A.1 Définition

A.1.1 Une expression régulière ASN.1 est une structure qui décrit un ensemble de chaînes dont le format est conforme à cette structure. Une expression régulière est elle-même une chaîne; elle est construite de façon analogue aux expressions arithmétiques, au moyen de divers opérateurs combinant des expressions plus petites. Les plus petites expressions, constituées (habituellement) de 1 ou 2 caractères, sont des structures génériques qui représentent un jeu de caractères.

Les expressions régulières présentées ici sont très semblables à celles des langages d'information en code machine comme Perl et à celles du schéma XML, où l'on peut trouver quelques autres exemples d'utilisation.

A.1.2 La plupart des caractères, y compris toutes les lettres et tous les chiffres, sont des expressions régulières qui correspondent à elles-mêmes.

EXEMPLE

L'expression régulière **"fred"** ne correspond qu'à la chaîne **"fred"**.

A.1.3 Deux expressions régulières peuvent être concaténées: l'expression régulière résultante correspond à toute chaîne formée par concaténation de deux sous-chaînes qui correspondent respectivement aux sous-expressions concaténées.

A.2 Métacaractères

A.2.1 Une séquence de métacaractères (ou un métacaractère) est un ensemble d'un ou de plusieurs caractères contigus qui ont une signification spéciale dans le contexte d'une expression régulière. La liste suivante contient toutes les séquences de métacaractères. Leur signification est expliquée dans les paragraphes qui suivent.

[ ]		Correspond à tout caractère du jeu où les intervalles sont indiqués par "-". Un "^" après le premier crochet complète le jeu qui le suit
{g,p,r,c}		Quadruplet qui identifie un caractère de l'ISO/CEI 10646-1 (voir § 37.8)
\N{name}		Correspond au caractère nommé (ou à tout caractère du jeu de caractères nommé) comme défini au § 38.1
.		Correspond à tout caractère (à moins qu'il ne soit un des caractères de nouvelle ligne définis au § 11.1.6)
\d		Correspond à un chiffre quelconque (équivalent à "**[0-9]**")
\w		Correspond à n'importe quel caractère alphanumérique (équivalent à "**[a-zA-Z0-9]**")
\t		Correspond au caractère TABULATION HORIZONTALE (9) (voir § 11.1.6)
\n		Correspond à l'un quelconque des caractères de nouvelle ligne définis au § 11.1.6
\r		Correspond au caractère RETOUR CHARIOT (13) (voir § 11.1.6)
\s		Correspond à l'un quelconque des caractères espace blanc (voir § 11.1.6)
\b		Correspond à une limite de mot
****	(préfixe)	Cite le métacaractère suivant et lui attribue une interprétation littérale
****		Correspond au caractère BARRE OBLIQUE INVERSE (92) "****"
""		Correspond au caractère GUILLEMET (34) (**"**)
|	(infixe)	Choix entre deux expressions
( )		Grouping of the enclosed expression
*****	(suffixe)	Groupement de l'expression enclose
+	(suffixe)	Correspond à l'expression précédente zéro, une ou plusieurs fois
?	(suffixe)	Correspond à l'expression précédente une ou plusieurs fois
#n	(suffixe)	Correspond à l'expression précédente exactement n fois (où n désigne un seul chiffre)
#(n)	(suffixe)	Correspond à l'expression précédente exactement n fois
#(n,)	(suffixe)	Correspond à l'expression précédente au moins n fois
#(n,m)	(suffixe)	Correspond à l'expression précédente au moins n mais au plus m fois
#(,m)	(suffixe)	Correspond à l'expression précédente au plus m fois

NOTE 1 – Les caractères ACCENT CIRCONFLEXE (94) "^" et TRAIT D'UNION-SIGNE MOINS (45) "-" sont des métacaractères additionnels dans certaines positions de la chaîne définies au § A.2.2.

NOTE 2 – La valeur entre parenthèses après un nom de caractère dans la présente annexe est la valeur décimale du caractère contenu dans l'ISO/CEI 10646-1.

NOTE 3 – Cette notation ne spécifie pas que les métacaractères "^" et "$" correspondent respectivement au début et à la fin d'une chaîne. Une chaîne doit donc correspondre à une expression régulière dans sa totalité sauf si cette expression contient les caractères ".*" à son début, à sa fin ou à la fois à son début et à sa fin.

NOTE 4 – Les séquences de métacaractères suivantes ne peuvent contenir d'espace blanc (voir § 11.1.6) sauf si celui-ci est situé immédiatement avant ou après une nouvelle ligne:

```
{g,p,r,c}
\N{name}
#n
#(n)
#(n,)
#(n,m)
#(,m)
```

Si une expression régulière contient une nouvelle ligne, tout caractère d'espacement situé immédiatement avant ou après la nouvelle ligne n'a pas de signification et ne correspond à rien (voir § 11.14.1).

A.2.2 Une liste de caractères entre crochets "[" et "]" correspond à tout caractère isolé de cette liste. Si le premier caractère de la liste est l'accent circonflexe "^", il correspond à tout caractère qui n'est pas dans la liste. Une série de caractères peut être spécifiée par l'indication des premier et dernier caractères, séparés par un trait d'union (selon la relation d'ordre définie au § 39.3). Toutes les séquences de métacaractères, sauf "]" et "\", perdent leur signification spéciale dans une liste. Pour inclure un littéral ACCENT CIRCONFLEXE (94) "^", il convient de le placer n'importe où sauf dans la première position ou de le faire précéder d'une barre oblique inverse. Pour inclure un littéral TRAIT D'UNION-SIGNE MOINS (45) "-", il convient de le placer en premier ou en dernier dans la liste, ou de le faire précéder d'une barre oblique inverse. Pour inclure un littéral CROCHET FERMANT (93) "]", il convient de le placer en premier. Si le premier caractère dans une liste est l'accent circonflexe "^" alors les caractères "-" et "]" correspondent à eux-mêmes lorsqu'ils suivent immédiatement cet accent circonflexe. Les séquences de métacaractères définies aux § A.2.3, A.2.4, A.2.6 et A.2.7 peuvent être utilisées entre les crochets où elles conservent leur signification.

EXEMPLES

L'expression régulière "[0123456789]", ou son équivalent "[0-9]", correspond à tout chiffre isolé.

L'expression régulière "[^0]" correspond à tout caractère isolé sauf 0.

L'expression régulière "[\d^.-]" correspond à tout chiffre isolé, un accent circonflexe, un trait d'union ou un point.

A.2.3 Afin d'éviter toute ambiguïté entre deux caractères ISO/CEI 10646-1 ayant le même glyphe, deux notations sont prévues. Une notation de la forme "{groupe,plan,ligne,cellule}" renvoie à un caractère (isolé) selon la production "Quadruple" définie au § 37.8.

A.2.4 Une notation de la forme "\N{valuereference}" correspond au caractère référencé si "valuereference" est une référence à une valeur de chaîne de caractères à alphabet restreint de longueur 1 (voir § 37) qui est définie ou importée dans le module courant. Une notation de la forme "\N{typereference}" correspond à tout caractère du jeu de caractères référencé si "typereference" est une référence à un sous-type d'un type "RestrictedCharacterStringType" qui est défini dans le module courant, ou est l'un des types "RestrictedCharacterStringType" définis au § 37.

NOTE – En particulier, "valuereference" ou "typereference" peut être une des références définies dans le module **ASN1-CHARACTER-MODULE** (voir 38.1) et importée dans le module courant (voir 37.8).

EXEMPLES

L'expression régulière "\N{greekCapitalLetterSigma}" correspond à la LETTRE MAJUSCULE GRECQUE SIGMA.

L'expression régulière "\N{BasicLatin}" correspond à tout caractère (isolé) du jeu de caractères LATIN DE BASE.

"[\N{BasicLatin}\N{Cyrillic}\N{BasicGreek}]+" et son équivalent "(\N{BasicLatin} | \N{Cyrillic} | \N{BasicGreek})+" sont des expressions régulières qui correspondent à une chaîne constituée de tout nombre (non nul) de caractères provenant des trois jeux de caractères spécifiés.

A.2.5 Le point "." correspond à tout caractère isolé, sauf si c'est un des caractères de nouvelle ligne définis au § 11.1.6.

A.2.6 Le symbole "**\d**" est un synonyme de "**[0-9]**", c'est-à-dire qu'il correspond à tout chiffre isolé. Le symbole "**\t**" correspond au caractère TABULATION HORIZONTALE (9). Le symbole "**\w**" est un synonyme de "**[a-zA-Z0-9]**", c'est-à-dire qu'il correspond à tout caractère isolé (minuscule ou majuscule) ou à tout chiffre isolé.

EXEMPLE

L'expression régulière "**\w+(\s\w+)*\.**" correspond à une phrase constituée d'au moins un mot (alphanumérique). Les mots sont séparés par un caractère espace blanc tel que défini au § 11.1.6. Il n'y a pas de caractère espace blanc avant le point final.

A.2.7 Le symbole "**\r**" correspond au caractère RETOUR CHARIOT (13). Le symbole "**\n**" correspond à l'un quelconque des caractères de nouvelle ligne définis au § 11.1.6. Le symbole "**\s**" correspond à l'un quelconque des caractères espace blanc définis au § 11.1.6. Le symbole "**\b**" correspond à une chaîne vide au début ou à la fin d'un mot.

EXEMPLE

L'expression régulière "**.*\bfred\b.***" correspond à toute chaîne qui inclut le mot "**fred**" (ce mot n'est pas seulement une série de quatre caractères; il est délimité). Il correspond donc à des chaînes comme "**fred**" ou "**je suis fred le premier**", mais non à des chaînes comme "**Mon nom est freddy**" ou "**I am afred I don't know how to spell 'afraid'!**" (je crains que vous ne sachiez comment épeler 'afraid').

A.2.8 Un caractère qui normalement fonctionne en tant que métacaractère peut être interprété littéralement par attribution à ce caractère d'un préfixe "****". Si l'expression régulière inclut un GUILLEMET (34), ce caractère doit être représenté par une paire de GUILLEMETS.

EXEMPLES

L'expression régulière **"\."** correspond à la chaîne (isolée) **"."**, mais non à une chaîne quelconque contenant un caractère isolé quelconque.

L'expression régulière **""""** correspond à la chaîne qui contient un seul GUILLEMET.

L'expression régulière **"\)"** correspond à la chaîne **")"**.

'expression régulière **"\a"**correspond au caractère **"a"**.

NOTE – Le quatrième exemple montre que la barre oblique inverse peut précéder des caractères qui ne sont pas des métacaractères. Toutefois, il est déconseillé de l'utiliser, du fait que d'autres métacaractères pourraient être autorisés dans des versions ultérieures de la présente Recommandation | Norme internationale.

A.2.9 Deux expressions régulières ou plus peuvent être jointes par l'opérateur infixe "**|**". L'expression régulière qui en résulte correspond à toute chaîne qui correspond à l'une ou l'autre des sous-expressions.

A.2.10 Une expression régulière peut être suivie par un opérateur de répétition. Si l'opérateur est "**?**", l'élément précédent est facultatif et il est présent au plus une fois. Si l'opérateur est "*****", l'élément précédent est présent zéro, une ou plusieurs fois. Si l'opérateur est "**+**", l'élément précédent est présent une ou plusieurs fois. Si l'opérateur est de la forme "**#(n)**", l'élément précédent est présent exactement n fois; dans ce cas particulier, les parenthèses peuvent être omises si n comprend un seul chiffre. S'il est de la forme "**#(n,)**", l'élément est présent n fois ou plus. S'il est de la forme "**#(,m)**", l'élément est facultatif et est présent au plus m fois. Enfin, s'il est de la forme "**#(n,m)**", l'élément est présent au moins n fois, mais pas plus que m fois.

NOTE – Il est illégal d'utiliser les métacaractères "*****", "**+**", "**?**" ou "**#**" comme premier caractère d'une expression régulière. Il est également illégal d'utiliser les métacaractères "**#**" ou "**|**" comme dernier caractère d'une expression régulière.

EXEMPLES

Un numéro de téléphone comme **"555-1212"** est une valeur à laquelle correspond l'expression régulière **"\d#3-\d#4"**, ou son équivalent **"\d#(3)-\d#(4)"**.

Un prix en dollars comme **"$12345.90"** est une valeur à laquelle correspond l'expression régulière **"$\d#(1,)(\.\d#(1,2))?"**. Il est à noter que des parenthèses sont requises après le symbole '**#**' si celui-ci est suivi d'une série.

Un numéro de sécurité sociale comme **"123-45-5678"** est une valeur à laquelle correspond l'expression régulière **"\d#3-?\d#2-?\d#4"**.

A.2.11 La répétition (voir § A.2.10) a priorité sur la concaténation (voir § A.1.3), qui à son tour a priorité sur l'alternance (voir § A.2.9). Une sous-expression peut être mise entre parenthèses pour outrepasser ces règles de priorité.

A.2.12 Lorsqu'une expression régulière contient des sous-expressions entre parenthèses, on attribue successivement à chaque parenthèse ouvrante (non citée) un nombre entier distinct (strictement positif) en allant de gauche à droite de

l'expression régulière. Chaque sous-expression peut alors être référencée à l'intérieur d'un commentaire avec une notation comme "*\1*", "*\2*" qui utilise l'entier associé. La sous-expression vide "**()**" n'est pas permise.

EXEMPLE

```
"((\d#2)(\d#2)(\d#4))" -- \1 est une date dans laquelle \2 est le mois,
                       -- \3 le jour et \4 l'année.
```

NOTE – Une référence formelle aux sous-expressions d'une expression régulière est nécessaire à de nombreuses fins, comme la nécessité d'écrire un texte pour détailler l'expression régulière contenue dans le module ASN.1. Il s'agit là d'une notation qui peut être utilisée afin de fournir de telles références. Cette notation n'est pas utilisée ailleurs dans la présente Recommandation | Norme internationale.

Annexe B

Règles applicables à la compatibilité des types et des valeurs

(Cette annexe fait partie intégrante de la présente Recommandation | Norme internationale)

La présente annexe s'adresse principalement aux concepteurs d'outils en vue de leur permettre d'interpréter la notation ASN.1 de la même façon. Elle a pour objet de spécifier clairement ce qu'est une définition ASN.1 légale et une définition ASN.1 non légale, et d'indiquer la valeur précise identifiée par un nom de référence de valeur ainsi que l'ensemble de valeurs précis identifié par un nom de référence d'ensemble de valeurs ou par un nom de référence de type. Elle ne vise pas à donner une définition des transformations valides des définitions ASN.1 à d'autres fins que celles indiquées ci-dessus.

B.1 Nécessité du concept de correspondance entre valeurs (introduction didactique)

B.1.1 Considérons les définitions ASN.1 suivantes:

```
A ::= INTEGER

B ::= [1] INTEGER

C ::= [2] INTEGER (0..6,...)

D ::= [2] INTEGER (0..6,...,7)

E ::= INTEGER (7..20)

F ::= INTEGER {red(0), white(1), blue(2), green(3), purple(4)}

a A ::= 3

b B ::= 4

c C ::= 5

d D ::= 6

e E ::= 7

f F ::= green
```

B.1.2 Il est clair que les références de valeurs **a**, **b**, **c**, **d**, **e** et **f** peuvent être utilisées dans la notation de valeur gouvernée respectivement par **A**, **B**, **C**, **D**, **E** et **F**. Par exemple:

```
W ::= SEQUENCE {w1  A  DEFAULT a}
```

et

```
x A ::= a
```

et

```
Y ::= A(1..a)
```

sont toutes des expressions valables compte tenu des définitions données au § B.1.1. Toutefois, si le type **A** ci-dessus était remplacé par **B**, **C**, **D**, **E** ou **F**, les déclarations qui en découlent seraient-elles illégales? De même, si la référence de valeur **a** ci-dessus était remplacée dans chacun de ces cas par **b**, **c**, **d**, **e** ou **f**, les déclarations qui en découlent seraient-elles légales?

B.1.3 Il serait plus délicat d'envisager dans chaque cas de remplacer la référence de type par le texte explicite à droite de sa définition, par exemple:

```
f INTEGER {red(0), white(1), blue(2), green(3), purple(4)} ::= green

W ::= SEQUENCE {
      w1    INTEGER {red(0), white(1), blue(2), green(3),  purple(4)}
      DEFAULT f}

x INTEGER {red(0), white(1), blue(2), green(3),  purple(4)} ::= f

Y ::= INTEGER {red(0), white(1), blue(2), green(3),  purple(4)}(1..f)
```

La notation ci-dessus serait-elle une notation ASN.1 légale?

B.1.4 Les utilisateurs seraient mal avisés de rédiger un texte semblable à certains des exemples donnés ci-dessus, même si la plupart sont légaux (voir le dernier exemple), car ces exemples sont pour le moins obscurs, voire au pire confus. Toutefois, on utilise souvent une référence à une valeur d'un type (pas nécessairement un type **INTEGER**) comme valeur par défaut de un type, un étiquetage ou un sous-typage étant appliqué dans le gouverneur. Le concept de *mappage entre valeurs* est introduit en vue de donner un moyen clair et précis de déterminer les structures qui, comme celles citées ci-dessus, sont légales.

B.1.5 Examinons de nouveau:

```
C ::= [2] INTEGER (0..6,...)

E ::= INTEGER (7..20)

F ::= INTEGER {red(0), white(1), blue(2), green(3), purple(4)}
```

Dans chaque cas, un nouveau type est créé. Pour le type **F**, nous pouvons clairement identifier une correspondance biunivoque entre les valeurs qu'il contient et les valeurs du type universel **INTEGER**. Pour les types **C** et **E**, nous pouvons identifier clairement une correspondance biunivoque entre les valeurs qu'ils contiennent et un sous-ensemble de valeurs du type universel **INTEGER**. Nous appelons cette relation un *mappage entre valeurs* des deux types. En outre, étant donné que les valeurs des types **F**, **C** et **E** sont toutes en mappage biunivoque avec des valeurs du type **INTEGER**, nous pouvons utiliser ces mappages pour définir des mappages entre les valeurs des types **F**, **C** et **E**. La Figure B.1 illustre cet exemple pour les types **F** et **C**.

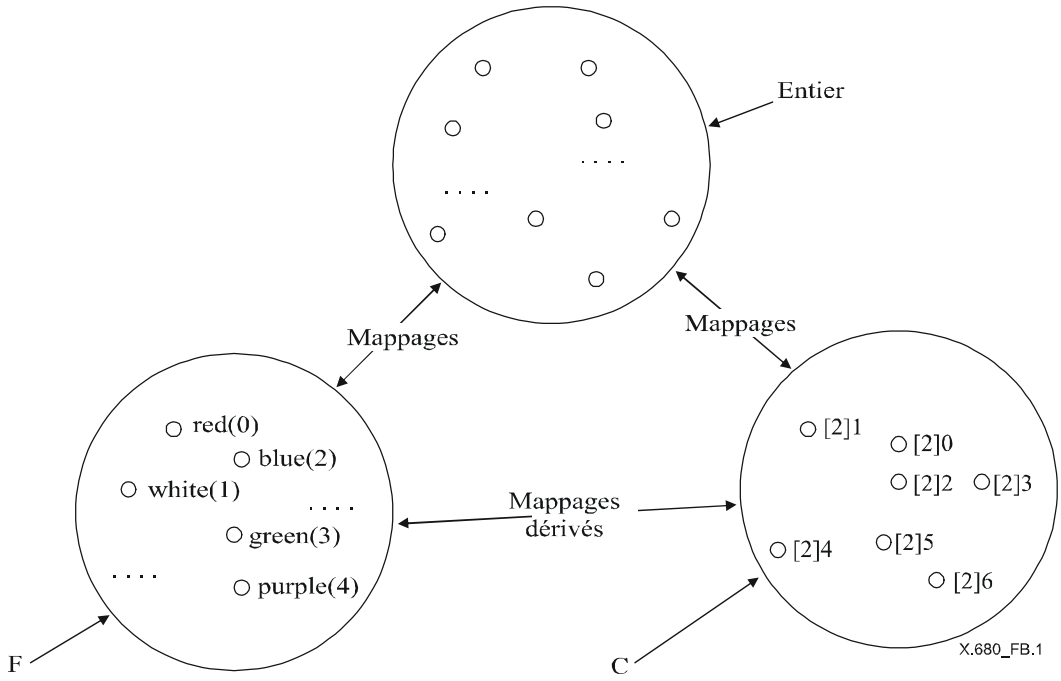

Figure B.1

B.1.6 Lorsque nous avons une référence à une valeur telle que:

```
c C ::= 5
```

qui, dans un certain contexte doit être utilisée pour identifier une valeur du type **F**, s'il existe un mappage entre cette valeur du type **C** et une (seule) valeur du type **F**, nous pouvons définir (et nous définissons) **c** comme une référence légale à une valeur du type **F**. Ce cas est illustré par la Figure B.2, où la référence de valeur c sert à identifier une valeur du type **F**, et peut être utilisée à la place d'une référence directe **f1** qu'il serait sans cela nécessaire de définir:

```
f1 F ::= 5
```

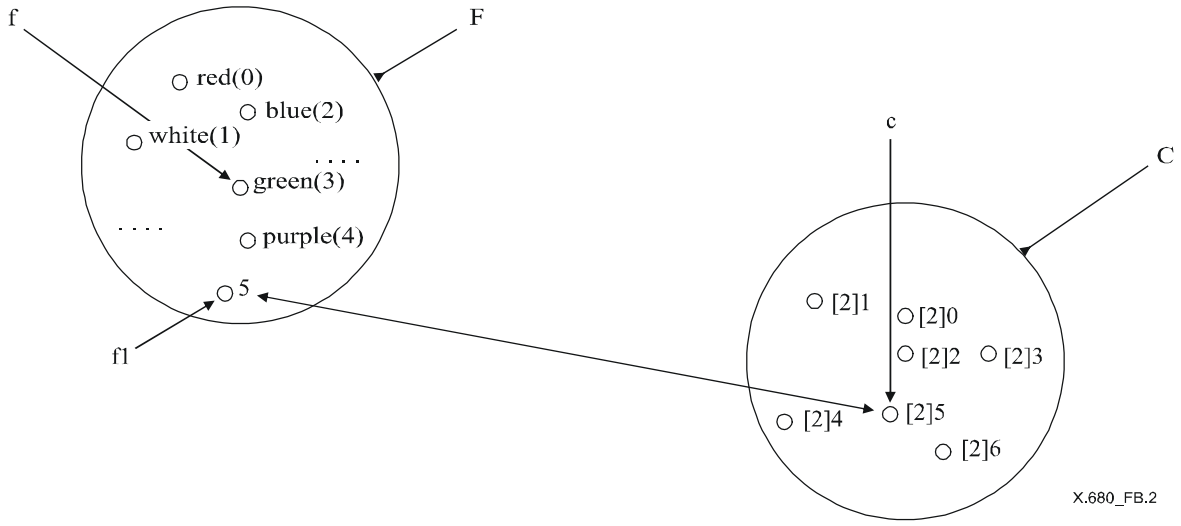

Figure B.2

B.1.7 Il convient de noter que dans certains cas, des valeurs d'un type (les valeurs 7 à 20 du type **A** du § B.1.1 par exemple) seront en mappage avec des valeurs d'un autre type (les valeurs 7 à 20 du type **E** du B.1.1 par exemple), mais que d'autres valeurs du même type (à partir de la valeur 21 du type **A**) ne le seront pas. Une référence à ces valeurs du type **A** ne serait pas une référence valable à une valeur du type **E**. (Dans cet exemple, l'ensemble des valeurs du type **E** est en mappage avec un sous-ensemble des valeurs du type **A**. En règle générale, un sous-ensemble de valeurs d'un type est en mappage avec un sous-ensemble de valeurs d'un autre type et les autres valeurs du premier type ne sont pas en mappage avec les autres valeurs du deuxième type.)

B.1.8 Dans le corps des normes ASN.1, un texte est utilisé pour spécifier la légalité concernant les cas présentés ci-dessus ou des cas similaires. Le paragraphe B.6 indique les conditions précises de légalité et il convient de s'y référer en cas de doute au sujet d'une construction complexe.

NOTE – Le fait qu'un mappage entre valeurs soit défini entre deux occurrences de constructions de "Type" permet d'utiliser des références de valeur établies à l'aide d'une construction de "Type" pour identifier des valeurs dans une autre construction de "Type" suffisamment semblable. Cela permet de typer des paramètres fictifs et réels en utilisant deux constructions de "Type" textuellement distinctes sans enfreindre les règles de compatibilité des paramètres fictifs et réels. Cela permet aussi de spécifier les champs de classes d'objets informationnels en utilisant une construction de "Type" et la valeur correspondante dans la définition d'un objet informationnel à l'aide d'une construction de "Type" distincte suffisamment similaire. (La liste de ces exemples n'est pas exhaustive.) Il est toutefois recommandé de n'avoir recours à ces possibilités que pour des cas simples comme **SEQUENCE OF INTEGER**, ou **CHOICE {int INTEGER, id OBJECT IDENTIFIER}**, et non pour des constructions de "Type" plus complexes.

B.2 Mappages entre valeurs

B.2.1 Le modèle sous-jacent est constitué de types, considérés comme des conteneurs qui ne se chevauchent pas et qui contiennent des valeurs, chaque occurrence de la construction de "Type" ASN.1 définissant un nouveau type distinct (voir Figures B.1 et B.2). La présente annexe indique dans quelles conditions des *mappages* existent entre les valeurs de ces types, ce qui permet d'utiliser une référence à une valeur d'un type lorsqu'une référence à une valeur d'un autre type est nécessaire.

EXEMPLE: Considérer:

```
X ::= INTEGER
Y ::= INTEGER
```

X et **Y** sont des noms de référence (pointeurs) de deux types distincts, mais un mappage existe entre les valeurs de ces types, de sorte que l'on peut utiliser n'importe quelle référence à une valeur de **X** pour une valeur gouvernée par **Y** (par exemple, après le mot-clé **DEFAULT**).

B.2.2 Dans l'ensemble de toutes les valeurs possibles de la notation ASN.1, un mappage entre valeurs se rapporte à une paire de valeurs. Un mappage entre valeurs est une relation mathématique qui a les propriétés suivantes: il est réflexif (chaque valeur ASN.1 est en mappage avec elle-même), symétrique (si une valeur **x1** est en mappage avec une valeur **x2**, la valeur **x2** est automatiquement en mappage avec la valeur **x1**) et transitive (s'il y a, d'une part, un mappage

entre une valeur **x1** et une valeur **x2**, d'autre part, un mappage entre une valeur **x2** et une valeur **x3**, il existe automatiquement un mappage entre **x1** et **x3**).

B.2.3 En outre, étant donné deux types **X1** et **X2**, considérés comme des ensembles de valeurs, le mappage entre les valeurs de **X1** et les valeurs de **X2** est une relation univoque, c'est-à-dire que pour toutes les valeurs **x1** de **X1** et **x2** de **X2**, s'il y a un mappage entre **x1** et **x2**, alors:

a) il n'y a pas de mappage entre **x1** et une autre valeur de **X2** différente de **x2**;

b) il n'y a pas de mappage entre une valeur quelconque de **X1** (autre que **x1**) et **x2**.

B.2.4 Lorsqu'il y a un mappage entre une valeur **x1** et une valeur **x2**, une référence à l'une de ces valeurs peut automatiquement servir de référence à l'autre si le type gouvernant l'exige.

NOTE – Le fait que des mappages entre valeurs soient définis dans certaines constructions de "Type" vise simplement à accorder une plus grande souplesse dans l'utilisation de la notation ASN.1. L'existence de ces mappages entre valeurs n'implique en aucune façon que les deux types acheminent les mêmes sémantiques du point de vue de l'application, mais il est recommandé de n'utiliser des constructions ASN.1 qui seraient illégales sans mappages entre valeurs, que si les types correspondants ont réellement la même sémantique du point de vue de l'application. Il est à noter que l'on trouvera souvent, dans des spécifications comportant de nombreuses définitions, des mappages entre valeurs de types qui sont des constructions ASN.1 identiques, mais qui ont une sémantique totalement différente du point de vue de l'application; toutefois, ces mappages entre valeurs ne sont jamais utilisés pour déterminer la légalité de l'ensemble de la spécification.

B.3 Définition de types identiques

B.3.1 Le concept de définition de types identiques permet de définir des mappages entre valeurs entre deux instances de "Type" identiques ou suffisamment similaires pour être normalement interchangeables. Afin de préciser la signification de "suffisamment similaires", le présent paragraphe indique une série de transformations appliquées à chacune des instances de "Type" en vue de produire une *forme normale* pour ces instances de "Type". Les deux instances de "Type" sont définies comme étant des définitions de types identiques si et seulement si leurs formes normales sont des listes ordonnées identiques des mêmes unités lexicales (voir § 11).

B.3.2 Chaque occurrence de "Type" dans une spécification ASN.1 est une liste ordonnée d'unités lexicales définies au § 11. La forme normale est obtenue par application des transformations définies du § B.3.2.1 au § B.3.2.6 (dans cet ordre).

B.3.2.1 Tous les commentaires (voir § 11.6) doivent être supprimés.

B.3.2.2 Les transformations suivantes ne sont pas récursives et ne doivent donc être appliquées qu'une seule fois, dans un ordre quelconque:

a) pour un type défini par une production "ValueSetTypeAssignment", sa définition est remplacée par une production "TypeAssignment" utilisant le même "Type" et une contrainte de sous-typage correspondant au contenu de "ValueSet" comme spécifié au § 15.6;

b) pour chaque type entier: l'éventuelle liste "NamedNumberList" (voir § 18.1) est réordonnée de façon que les identificateurs ("identifier") soient dans l'ordre alphabétique ("a" en premier, "z" en dernier);

c) pour chaque type énuméré: des numéros sont ajoutés, comme spécifié au § 19.3, à tout élément "EnumerationItem" (voir 19.1) qui est un identificateur "identifier" (sans numéro); puis l'énumération "RootEnumeration" est réordonnée de façon que les identificateurs ("identifier") soient dans l'ordre alphabétique ("a" en premier, "z" en dernier);

d) pour chaque type de chaîne binaire: l'éventuelle liste "NamedBitList" (voir § 21.1) est réordonnée de façon que les identificateurs ("identifier") soient dans l'ordre alphabétique ("a" en premier, "z" en dernier);

e) pour chaque valeur d'identificateur d'objet: chaque composant "ObjIdComponents" est transformé en sa forme "NumberForm" correspondante, conformément à la sémantique du § 31 (voir l'exemple du § 31.12);

f) pour chaque valeur d'identificateur d'objet relatif (voir § 32.3): chaque composant "RelativeOIDComponents" est transformé en sa forme "NumberForm" correspondante, conformément à la sémantique du § 32;

g) pour les types séquence (voir § 24) et ensemble (voir § 26): toute extension de la forme "ExtensionAndException", "ExtensionAdditions" est coupée et collée à la fin de la production "ComponentTypeLists"; la production "OptionalExtensionMarker" est supprimée si elle est présente.

Si l'option "TagDefault" est égale à **IMPLICIT TAGS**, le mot clé **IMPLICIT** est ajouté à toutes les instances de "Tag" (voir § 30) sauf dans les cas suivants:

- le mot clé **IMPLICIT** est déjà présent;

- le mot réservé **EXPLICIT** est présent;
- le type étiqueté est un type **CHOICE**;
- il s'agit d'un type ouvert.

Si l'option "TagDefault" est égale à **AUTOMATIC TAGS**, la décision d'appliquer l'étiquetage automatique est prise conformément au § 24.2 (l'étiquetage automatique sera effectué ultérieurement).

NOTE – Les paragraphes 24.3 et 26.2 indiquent que la présence d'une étiquette "Tag" dans une notation "ComponentType" qui a été insérée à la suite du remplacement de "Components of Type" n'empêche pas en soi l'étiquetage automatique.

Si l'option "ExtensionDefault" est égale à **EXTENSIBILITY IMPLIED**, des points de suspension ("...") sont ajoutés après la notation "ComponentTypeLists", s'ils ne sont pas déjà présents;

h) pour le type choix (voir § 28): la liste "RootAlternativeType" est réordonnée de façon que les identificateurs des types "NameType" soient dans l'ordre alphabétique ("a" en premier, "z" en dernier). Si la production "OptionalExtensionMarker" est présente, elle est supprimée. Si l'option "TagDefault" est égale à **IMPLICIT TAGS**, le mot clé **IMPLICIT** est ajouté à toutes les instances de "Tag" (voir § 30) sauf dans les cas suivants:

- le mot clé **IMPLICIT** est présent;
- le mot réservé **EXPLICIT** est présent;
- le type étiqueté est un type **CHOICE**;
- il s'agit d'un type ouvert.

Si l'option "TagDefault" est égale à **AUTOMATIC TAGS**, la décision d'appliquer l'étiquetage automatique est prise conformément au 28.5 (l'étiquetage automatique sera effectué ultérieurement). Si l'option "ExtensionDefault" est égale à **EXTENSIBILITY IMPLIED**, des points de suspension ("...") sont ajoutés après la notation "AlternativeTypeLists", s'ils ne sont pas déjà présents.

B.3.2.3 Les transformations suivantes doivent faire l'objet d'une application récursive dans l'ordre spécifié, jusqu'à ce qu'une situation fixe soit atteinte:

a) pour chaque valeur d'identificateur d'objet (voir § 31.3): si la définition de valeur commence par une valeur définie ("DefinedValue"), celle-ci est remplacée par sa définition;

b) pour chaque valeur d'identificateur d'objet reatif (voir § 32.3): si la définition de valeur contient des valeurs définies ("DefinedValue"), celles-ci sont remplacées par leur définition;

c) pour les types séquence et ensemble: toutes les instances de "**COMPONENTS OF** types" (voir § 24) sont transformées conformément aux § 24 et 26;

d) pour les types séquence, ensemble et choix: s'il a été antérieurement décidé d'effectuer un étiquetage automatique [voir les points g) et h) du § B.3.2.2], l'étiquetage automatique est appliqué conformément aux § 24, 26 et 28;

e) pour le type sélection: la construction est remplacée par la forme choisie conformément au § 29;

f) toutes les références de type sont remplacées par leurs définitions conformément aux règles suivantes:

- si le type remplaçant est une référence au type qui subit la transformation, la référence de type est remplacée par un élément spécial ne correspondant à aucun autre élément que lui-même;
- si le type remplaçant est un type séquence de ou ensemble de, les contraintes faisant éventuellement suite au type remplacé sont transférées devant le mot clé **OF**;
- si le type remplacé est un type paramétré ou un ensemble de valeurs paramétrées (voir la Rec. UIT-T X.683 | ISO/CEI 8824-4, § 8.2), chaque référence "DummyReference" est remplacée par le paramètre "ActualParameter" correspondant;

g) toutes les références de valeur sont remplacées par leurs définitions, si la valeur remplacée est une valeur paramétrée (voir la Rec. UIT-T X.683 | ISO/CEI 8824-4, § 8.2), chaque référence "DummyReference" est remplacée par le paramètre "ActualParameter" correspondant.

NOTE – Avant de remplacer une quelconque référence de valeur, les procédures de la présente annexe doivent être appliquées afin de s'assurer que cette référence de valeur désigne, directement ou par le biais de mappages, une valeur de son type gouvernant.

B.3.2.4 Pour le type ensemble: la liste "RootComponentTypeList" est réordonnée de façon que les types "ComponentType" soient dans l'ordre alphabétique ("a" en premier, "z" en dernier).

B.3.2.5 Les transformations suivantes doivent s'appliquer aux définitions de valeur:

a) si une valeur entière est définie avec un identificateur, celui-ci est remplacé par le numéro associé;

b) si une valeur de chaîne binaire est définie au moyen d'identificateurs, elle est remplacée par la chaîne "bstring" correspondante dans laquelle tous les bits nuls de fin sont supprimés;

c) tout espace blanc situé immédiatememt avant ou après chaque nouvelle ligne (y compris la nouvelle ligne) dans une chaîne "cstring" est supprimé;

d) tout espace blanc situé dans des chaînes "bstring" et "hstring" est supprimé;

e) chaque valeur réelle définie avec la base 2 est normalisée de sorte que la mantisse soit impaire, et chaque valeur réelle définie avec la base 10 est normalisée de sorte que le dernier chiffre de la mantisse soit différent de 0;

f) chaque valeur **`GeneralizedTime`** et **`UTCTime`** est remplacée par une chaîne qui est conforme aux règles utilisées lors du codage selon les règles DER et CER (voir la Rec. UIT-T X.690 | ISO/CEI 8825-1, § 11.7 et 11.8);

g) après application du c), chaque valeur **`UTF8String`**, **`NumericString`**, **`PrintableString`**, **`IA5String`**, **`VisibleString`** (**`ISO646String`**), **`BMPString`** et **`UniversalString`** est remplacée par la valeur équivalente de type **`UniversalString`** écrite au moyen de la notation "Quadruple" (voir § 37.8).

B.3.2.6 Chaque occurrence de "realnumber" doit être transformée en une valeur de séquence "SequenceValue" associée à la "base" 10. Chaque occurrence de "RealValue" associée à une valeur "SequenceValue" doit être transformée en la valeur "SequenceValue" associée de la même "base", de façon que le dernier chiffre de la mantisse soit différent de zéro.

B.3.3 Si deux instances de "Type", transformées en forme normale, sont des listes identiques d'unités lexicales (voir § 11), ce sont des définitions de types identiques sauf dans le cas suivant: si une notation "objectclassreference" (voir la Rec. UIT-T X.681 | ISO/CEI 8824-2, § 7.1) "objectreference" (voir la Rec. UIT-T X.681 | ISO/CEI 8824-2, § 7.2) ou "objectsetreference" (voir la Rec. UIT-T X.681 | ISO/CEI 8824-2, § 7.3) apparaît dans la forme normalisée du "Type", les deux types **ne** sont **pas** considérés comme étant des définitions de types identiques, et il n'y aura pas de mappage entre leurs valeurs (voir § B.4 ci-dessous).

NOTE – Cette exception a été ajoutée pour éviter de devoir fournir des règles de transformation en forme normale pour les éléments de syntaxe concernant la notation des classes d'objets informationnels, des objets informationnels et des ensembles d'objets informationnels. De même, la spécification pour la normalisation de la notation de toutes les valeurs et de la notation des opérations arithmétiques sur des ensembles n'a pas été insérée pour le moment. Si cette spécification s'avère nécessaire, elle pourra être fournie dans une future version de la présente Recommandation | Norme internationale. Le concept de définitions de types identiques et de mappages entre valeurs a été introduit pour faire en sorte que l'on puisse utiliser des constructions ASN.1 simples soit en ayant recours au nom de référence, soit en recopiant le texte. Il n'a pas été jugé utile de fournir cette fonctionnalité pour des instances plus complexes de "Type" comprenant des classes d'objets informationnels, etc.

B.4 Spécification des mappages entre valeurs

B.4.1 Si deux occurrences de "Type" sont des définitions de types identiques conformément aux règles du § B.3, il existe, pour chaque valeur de l'un des types, un mappage avec une valeur de l'autre type.

B.4.2 Pour un type **`X1`** créé à partir d'un type quelconque **`X2`** par étiquetage (voir § 30), il existe pour toute valeur de **`X1`** un mappage avec une valeur de **`X2`**.

NOTE – Bien que des mappages entre valeurs soient définis entre les valeurs de **`X1`** et de **`X2`** dans le présent § B.4.2, et entre les valeurs de **`X3`** et de **`X4`** au § B.4.3, si ces types sont imbriqués dans la définition d'autres types (comme **`SEQUENCE`** ou **`CHOICE`**), les définitions de ces types (**`SEQUENCE`** ou **`CHOICE`**) ne seront pas des définitions de types identiques et il n'y aura pas de mappage entre leurs valeurs.

B.4.3 Pour un type **`X3`** créé en sélectionnant des valeurs d'un type gouvernant quelconque **`X4`** par la définition d'un ensemble de valeurs ou par sous-typage, un mappage existe entre les valeurs du nouveau type **`X3`** et les valeurs du type **`X4`** qui ont été choisies par la contrainte de sous-typage ou par l'ensemble de valeurs. La présence ou l'absence d'un marqueur d'extension n'a aucune incidence sur cette règle.

B.4.4 Le paragraphe B.5 définit des mappages supplémentaires entre valeurs relatives à certains types de chaînes de caractères.

B.4.5 Un mappage est défini entre toutes les valeurs d'un type entier défini avec des valeurs nommées et celles d'un type entier défini sans valeurs nommées, ou avec des valeurs nommées différentes, ou encore avec des valeurs nommées différemment, ou les deux à la fois.

NOTE – L'existence de ce mappage entre valeurs n'a aucune incidence sur les exigences des règles de visibilité quant à l'utilisation des noms des valeurs nommées. On ne peut les utiliser que dans le domaine de visibilité du type associé, ou en utilisant une référence à ce type.

B.4.6 Un mappage entre valeurs est défini entre toutes les valeurs d'un type chaîne binaire définie avec des bits nommés et celles d'un type chaîne binaire définie sans bits nommés, ou avec des bits nommés différents, ou encore avec des bits nommés différemment, ou les deux à la fois.

NOTE – L'existence de ce mappage entre valeurs n'a aucune incidence sur les exigences des règles de visibilité quant à l'utilisation des noms des bits nommés. On ne peut les utiliser que dans le domaine de visibilité du type associé, ou en utilisant une référence à ce type.

B.5 Mappages supplémentaires définis entre valeurs des types de chaînes de caractères

B.5.1 Il y a deux groupes de types de chaînes de caractères à alphabet restreint: le groupe A (voir § B.5.2) et le groupe B (voir § B.5.3). Des mappages entre valeurs sont définis entre tous les types du groupe A et les références aux valeurs de ces types peuvent être utilisées lorsqu'elles sont gouvernées par l'un des autres types. Il n'y a jamais de mappage entre les valeurs des types du groupe B, ni entre les valeurs d'un type du groupe A et les valeurs d'un type du groupe B.

B.5.2 Le groupe A comprend:

`UTF8String`
`NumericString`
`PrintableString`
`IA5String`
`VisibleString` (**`ISO646String`**)
`UniversalString`
`BMPString`

B.5.3 Le groupe B comprend:

`TeletexString` (**`T61String`**)
`VideotexString`
`GraphicString`
`GeneralString`

B.5.4 Pour spécifier les mappages entre valeurs dans le groupe A, on utilise le mappage entre les valeurs de chaînes de caractères de chacun des types et les valeurs du type **`UniversalString`**, puis on utilise la transitivité du mappage entre valeurs. Le mappage entre les valeurs de l'un des types du groupe A et les valeurs du type **`UniversalString`**, est obtenu par remplacement d'une chaîne donnée par une chaîne **`UniversalString`** de même longueur, chaque caractère étant remplacé par le caractère correspondant comme indiqué ci-dessous.

B.5.5 Formellement, l'ensemble des valeurs abstraites du type **`UTF8String`** est le même que celui du type **`UniversalString`** mais avec une étiquette différente (voir § 37.16) et chaque valeur abstraite du type **`UTF8String`** est définie de sorte qu'il y ait mappage avec la valeur abstraite du type **`UniversalString`** qui lui est associée.

B.5.6 Les glyphes (formes de caractères imprimées) pour les caractères servant à former les types **`NumericString`** et **`PrintableString`** sont en mappage d'une manière reconnaissable et non ambiguë avec un sous-ensemble des glyphes attribués aux 128 premiers caractères de l'ISO/CEI 10646-1. Le mappage entre les valeurs de ces types est défini à l'aide de ce mappage entre glyphes.

B.5.7 Les chaînes **`IA5String`** et **`VisibleString`** sont en mappage avec **`UniversalString`**, chaque caractère du type **`IA5String`** ou **`VisibleString`** est associé au caractère du type **`UniversalString`** qui a la même valeur (32 bits) dans le codage BER de **`UniversalString`** que la valeur (8 bits) du codage BER de **`IA5String`** ou **`VisibleString`**.

B.5.8 Le type **`BMPString`** est formellement un sous-ensemble du type **`UniversalString`** et les valeurs abstraites de ces deux types sont donc en mappage.

B.6 Conditions particulières de la compatibilité des types et des valeurs

Dans le présent paragraphe, on utilise le concept de mappage entre valeurs pour fournir un texte précis pour la légalité de certaines constructions ASN.1.

B.6.1 Toute occurrence de "Value", *x-notation*, avec un type gouvernant **`Y`**, identifie la valeur, *y-val*, du type gouvernant **`Y`** qui est en mappage avec la valeur *x-val* spécifiée par *x-notation*. Toutefois, il faut que cette valeur existe.

Par exemple, considérons l'occurrence de **x** dans la dernière ligne des expressions:

```
X   ::= [0] INTEGER (0..30)

x X ::= 29

Y   ::= [1] INTEGER (25..35)

Z1  ::= Y (x | 30)
```

Ces constructions ASN.1 sont légales et dans la dernière définition, la *x-notation* **x** fait référence à *x-val* 29 du type **X** et, grâce au mappage entre valeurs, identifie *y-val* 29 du type **Y**. La *x-notation* 30 fait référence à *y-val* 30 de type **Y**, et **Z1** est l'ensemble des valeurs 29 et 30. Par ailleurs, la définition:

```
Z2 ::= Y (x | 20)
```

est illégale car il n'y a pas de valeur *y-val* à laquelle la *x-notation* 20 puisse être associée.

B.6.2 Toute occurrence de "Type", *t-notation*, qui a un type gouvernant **V**, identifie l'ensemble complet de valeurs se trouvant dans la racine du type gouvernant **V** pour lesquelles il existe un mappage avec l'une quelconque des valeurs se trouvant dans la racine de la *t-notation* "Type". Cet ensemble doit contenir au moins une valeur.

Par exemple, considérons l'occurrence de **W** dans la dernière ligne des expressions suivantes:

```
V  ::= [0] INTEGER (0..30)

W  ::= [1] INTEGER (25..35)

Y  ::= [2] INTEGER (31..35)

Z1 ::= V (W | 24)
```

W fournit les valeurs 25 à 30 aux opérations arithmétiques sur les ensembles, ce qui fait que **Z1** a les valeurs 24 à 30. Par ailleurs, la définition:

```
Z2 ::= V (Y | 24)
```

est illégale car il n'y a pas de valeur du type **Y** qui soit en mappage avec une valeur de type **V.**

B.6.3 Le type de toute valeur fournie comme paramètre effectif doit être tel qu'il y ait un mappage entre cette valeur et l'une des valeurs du type qui gouverne le paramètre fictif, et c'est une valeur de ce type gouvernant qui est identifiée.

B.6.4 Si un "Type" est fourni comme paramètre effectif quand le paramètre fictif désigne un ensemble de valeurs, toutes les valeurs de ce "Type" doivent être en mappage avec des valeurs du type qui gouverne le paramètre fictif. Le paramètre effectif sera l'ensemble des valeurs du type gouvernant qui sont en mappage avec les valeurs de "Type".

B.6.5 L'utilisation de valeurs d'un type **A**, à la place de valeurs d'un paramètre fictif qui désigne une valeur ou un ensemble de valeurs, est illégale sauf si pour toutes les valeurs de **A**, et pour chaque utilisation de **A** dans le membre de droite de la définition, cette valeur de **A** peut être appliquée légalement à la place du paramètre fictif.

B.7 Exemples

B.7.1 Le présent paragraphe donne des exemples pour illustrer les § B.3 et B.4.

B.7.2 Exemple 1

```
X ::= SEQUENCE
      {name VisibleString,
      age  INTEGER}

X2 ::= [8] SEQUENCE
      {name VisibleString,
      age  INTEGER}
```

```
X1 ::=  SEQUENCE
   {name VisibleString,
   -- commentaire --
         age  INTEGER}

X3 ::=  SEQUENCE
   {name VisibleString,
   age  AgeType}

AgeType ::= INTEGER
```

X, **X1**, **X2** et **X3** sont toutes des définitions de types identiques. Les différences d'espaces blancs et de commentaires ne sont pas visibles et l'utilisation de la référence de type **AgeType** dans **X3** n'affecte pas la définition de ce type. Il est toutefois à noter que si l'un des identificateurs des éléments de la séquence était modifié, les types cesseraient d'être des définitions identiques et il n'y aurait plus de mappage entre leurs valeurs.

B.7.3 Exemple 2

```
B ::= SET                          B1 ::= SET
      {name VisibleString,                {age INTEGER,
      age  INTEGER}                       name VisibleString}
```

sont des définitions de types identiques à condition que ni l'une ni l'autre ne soit dans un module dont l'en-tête contient **AUTOMATIC TAGS**, sinon il ne s'agit plus de définitions de types identiques, et il n'y a plus de mappage entre leurs valeurs. On peut donner des exemples similaires avec **CHOICE** et **ENUMERATED** (au moyen de la forme "identifier" de "EnumerationItem").

B.7.4 Exemple 3

```
C ::= SET                          C1 ::= SET
      {name [0]VisibleString,             {name VisibleString,
      age INTEGER}                        age INTEGER (1..64)}
```

ne sont pas des définitions de types identiques, et ne sont ni l'une ni l'autre des définitions de types identiques à celles de **B** ou **B1**, et il n'y a pas de mappage entre les valeurs de **C** et **C1** ni entre leurs valeurs et celles de **B** ou **B1**.

B.7.5 Exemple 4

```
x  INTEGER { y (2) } ::= 3

z  INTEGER ::= x
```

est légal, et attribue la valeur 3 à **z** au moyen du mappage entre valeurs définie au § B.4.5.

B.7.6 Exemple 5

```
b1 BIT STRING ::= '101'B

b2 BIT STRING {version1(0), version2(1), version3(2)} ::= b1
```

est légal, et attribue la valeur **{version1, version3}** à **b2**.

B.7.7 Exemple 6

D'après les définitions du § B.1.1, l'utilisation dans des types **SEQUENCE** de notations:

```
X DEFAULT y
```

est légale, lorsque **X** désigne **A**, **B**, **C**, **D**, **E** ou **F** ou le texte figurant dans la partie droite des définitions ainsi nommées, et **y** désigne **a**, **b**, **c**, **d**, **e** ou **f**, sauf dans les cas suivants: **E DEFAULT y** est illégal pour **a**, **b**, **c**, **d**, **f** et **C DEFAULT e** est illégal car, en pareils cas, il n'existe pas de mappage entre la référence de valeur par défaut et l'une des valeurs du type par défaut.

Annexe C

Valeurs d'identificateur d'objet affectées

(Cette annexe fait partie intégrante de la présente Recommandation | Norme internationale)

La présente annexe rappelle les valeurs d'identificateur d'objet et de descripteur d'objet affectées dans la série de Recommandations | Normes internationales relatives à la notation ASN.1 et contient un module ASN.1 à utiliser pour faire reference à ces valeurs d'identificateur d'objet.

C.1 Valeurs d'identificateur d'objet affectées dans la présente Recommandation | Norme internationale

Les valeurs suivantes sont affectées dans la présente Recommandation | Norme internationale:

Paragraphe 37.3

Valeur d'identificateur d'objet:
{ joint-iso-itu-t asn1(1) specification(0) characterStrings(1) numericString(0) }

Valeur de descripteur d'objet: **"NumericString ASN.1 type"**

Paragraphe 37.5

Valeur d'identificateur d'objet:
{ joint-iso-itu-t asn1(1) specification(0) characterStrings(1) printableString(1) }

Valeur de descripteur d'objet: **"PrintableString ASN.1 type"**

Paragraphe 38.1

Valeur d'identificateur d'objet:
{ joint-iso-itu-t asn1(1) specification(0) modules(0) iso10646(0) }

Valeur de descripteur d'objet: **"ASN.1 Character Module"**

Paragraphe C.2

Valeur d'identificateur d'objet:
{ joint-iso-itu-t asn1(1) specification(0) modules(0) object-identifiers(1) }

Valeur de descripteur d'objet: **"ASN.1 Object Identifier Module"**

C.2 Valeurs d'identificateur d'objet dans les normes relatives à l'ASN.1 et aux règles de codage

Le présent paragraphe spécifie un module ASN.1 contenant la définition d'un nom de référence pour chaque valeur d'identificateur d'objet définie dans les normes relatives à l'ASN.1 (Rec. UIT-T X.680 | ISO/CEI 8824-1 à Rec. UIT-T X.693 | ISO/CEI 8825-4).

NOTE – Ces valeurs peuvent être utilisées dans la notation de valeur du type OBJECT IDENTIFIER et des types qui en sont dérivés. Toutes les références de valeur définies dans le module spécifié dans le présent paragraphe sont exportées et doivent être importées par tout module dans lequel on souhaite les utiliser.

```
ASN1-Object-Identifier-Module { joint-iso-itu-t asn1(1) specification(0) modules(0)
object-identifiers(1) }
      DEFINITIONS ::= BEGIN

      -- type ASN.1 NumericString (voir 37.3) --
      numericString OBJECT IDENTIFIER ::=
            { joint-iso-itu-t asn1(1) specification(0) characterStrings(1)
      numericString(0) }

      -- type ASN.1 PrintableString (voir 37.5) --
      printableString OBJECT IDENTIFIER ::=
            { joint-iso-itu-t asn1(1) specification(0) characterStrings(1)
      printableString(1) }

      -- module des caractères ASN.1 (voir 38.1) --
      asn1CharacterModule OBJECT IDENTIFIER ::=
            { joint-iso-itu-t asn1(1) specification(0) modules(0) iso10646(0) }
```

```
    -- module des identificateurs d'objets ASN.1 (le présent module) --
    asn1ObjectIdentifierModule OBJECT IDENTIFIER ::=
          { joint-iso-itu-t asn1(1) specification(0) modules(0)
    object-identifiers(1) }

    -- codage BER d'un seul type ASN.1 --
    ber OBJECT IDENTIFIER ::=
          { joint-iso-itu-t asn1(1) basic-encoding(1) }

    -- codage CER d'un seul type ASN.1 --
    cer OBJECT IDENTIFIER ::=
          { joint-iso-itu-t asn1(1) ber-derived(2) canonical-encoding(0) }

    -- codage DER d'un seul type ASN.1 --
    der OBJECT IDENTIFIER ::=
          { joint-iso-itu-t asn1(1) ber-derived(2) distinguished-encoding(1) }

    -- codage PER d'un seul type ASN.1 (de base avec alignement) --
    perBasicAligned OBJECT IDENTIFIER ::=
          { joint-iso-itu-t asn1(1) packed-encoding(3) basic(0) aligned(0) }

    -- codage PER d'un seul type ASN.1 (de base sans alignement) --
    perBasicUnaligned OBJECT IDENTIFIER ::=
          { joint-iso-itu-t asn1(1) packed-encoding(3) basic(0) unaligned(1) }

    -- codage PER d'un seul type ASN.1 (canonique avec alignement) --
    perCanonicalAligned OBJECT IDENTIFIER ::=
          { joint-iso-itu-t asn1(1) packed-encoding(3) canonical(1) aligned(0) }

    -- codage PER d'un seul type ASN.1 (canonique sans alignement) --
    perCanonicalUnaligned OBJECT IDENTIFIER ::=
          { joint-iso-itu-t asn1(1) packed-encoding(3) canonical(1) unaligned(1) }

    -- codage XER d'un seul type ASN.1 (de base) --
    xerBasic OBJECT IDENTIFIER ::=
          {joint-iso-itu-t asn1(1) xml-encoding(5) basic(0) }

    -- codage XER d'un seul type ASN.1 (canonique) --
    xerCanonical OBJECT IDENTIFIER ::=
          {joint-iso-itu-t asn1(1) xml-encoding(5) canonical(1) }

END -- module des identificateurs d'objets ASN1 --
```

Annexe D

Affectation des valeurs de composant d'identificateur d'objet

(Cette annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

La présente annexe décrit les arcs de niveau supérieur de l'arbre d'enregistrement des identificateurs d'objet. Elle ne donne aucune explication sur la manière dont les nouveaux arcs sont ajoutés ni sur les règles que les autorités d'enregistrement doivent suivre. Pour cela, on se reportera à la Rec. UIT-T X.660 | ISO/CEI 9834-1.

D.1 Affectation des valeurs de composant d'identificateur d'objet à partir du nœud racine

D.1.1 Trois arcs sont spécifiés à partir du nœud racine. Les valeurs et les identificateurs affectés ainsi que l'autorité chargée d'affecter les valeurs subséquentes de composant sont les suivants:

Valeur	*Identificateur*	*Autorité chargée de l'affectation des valeurs subséquentes*
0	**itu-t**	UIT-T (voir § D.2)
1	**iso**	ISO (voir § D.3)
2	**joint-iso-itu-t**	Voir § D.4

D.1.2 Chacun des identificateurs **itu-t**, **iso** et **joint-iso-itu-t**, affectés ci-dessus, peut être utilisé en tant que forme "NameForm" (voir § 31.3).

D.1.3 Les identificateurs **ccitt** et **joint-iso-ccitt** sont respectivement synonymes de **itu-t** et de **joint-iso-itu-t** et peuvent donc figurer dans des valeurs d'identificateur d'objet.

D.2 Affectation des valeurs de composant d'identificateur d'objet à partir du nœud ITU-T

D.2.1 Cinq arcs sont spécifiés à partir du nœud identifié par **itu-t**. Les valeurs et les identificateurs affectés sont les suivants:

Valeur	*Identificateur*	*Autorité chargée de l'affectation des valeurs subséquentes*
0	**recommendation**	Voir § D.2.2
1	**question**	Voir § D.2.3
2	**administration**	Voir § D.2.4
3	**network-operator**	Voir § D.2.5
4	**identified-organization**	Voir § D.2.6

Ces identificateurs peuvent être utilisés en tant que forme "NameForm" (voir § 31.3).

D.2.2 Les arcs au-dessous de **recommendation** ont les valeurs **1** à **26**, correspondant aux identificateurs affectés **a** à **z**. Les arcs figurant au-dessous correspondent aux numéros des Recommandations de l'UIT-T (et du CCITT) dans la série identifiée par la lettre. Les arcs figurant au-dessous sont déterminés en fonction des besoins découlant des Recommandations de l'UIT-T (et du CCITT). Les identificateurs **a** à **z** peuvent être utilisés en tant que forme "NameForm".

D.2.3 Les arcs au-dessous de **question** ont des valeurs correspondant aux commissions d'études de l'UIT-T et à la période d'études. Chaque valeur est calculée par la formule:

Numéro de commission d'études + (période * 32)

où "période" a la valeur 0 pour 1984-1988, 1 pour 1988-1992, etc., et le multiplicateur est la valeur décimale 32.

Les arcs au-dessous de chaque commission d'études ont des valeurs correspondant aux questions confiées à cette commission d'études. Les arcs figurant au-dessous sont déterminés en fonction des besoins du groupe (par exemple groupe de travail ou groupe spécial du rapporteur) chargé d'étudier la question.

D.2.4 Les arcs au-dessous de **administration** ont pour valeur les indicatifs DCC X.121. Les arcs figurant au-dessous sont déterminés en fonction des besoins de l'Administration du pays identifié par l'indicatif DCC X.121 considéré.

D.2.5 Les arcs au-dessous de **network-operator** ont pour valeur les codes DNIC X.121. Les arcs figurant au-dessous sont déterminés en fonction des besoins de l'administration ou de l'exploitation reconnue identifiée par le code DNIC considéré.

D.2.6 Les arcs au-dessous de **identified-organization** ont des valeurs affectées par le Bureau de la normalisation des télécommunications (TSB) de l'UIT. Les arcs figurant au-dessous sont déterminés en fonction des besoins des organisations identifiées.

NOTE – Les organisations pour lesquelles cet arc peut être utile sont les suivantes:

– exploitations reconnues n'exploitant pas un réseau public de données;
– organismes scientifiques et industriels;
– organisations de normalisation régionales;
– organisations multinationales.

D.3 Affectation des valeurs de composant d'identificateur d'objet à partir du nœud ISO

D.3.1 Trois arcs sont spécifiés à partir du nœud identifié par **iso(1)**. Les valeurs et les identificateurs affectés sont les suivants:

Valeur	*Identificateur*	*Autorité chargée de l'affectation des valeurs subséquentes*
0	**standard**	Voir § D.3.2
2	**member-body**	Voir § D.3.3
3	**identified-organization**	Voir § D.3.4

Ces identificateurs peuvent être utilisés en tant que forme "NameForm".

NOTE – L'utilisation de l'arc **registration-authority(1)** a été supprimée.

D.3.2 Chacun des arcs au-dessous de **standard** a pour valeur le numéro d'une Norme internationale. Lorsque la Norme internationale est en plusieurs parties, il faut un arc supplémentaire pour le numéro de la partie, sauf indication expresse contraire dans le texte de la Norme internationale. Les valeurs des arcs qui suivent seront telles que définies dans la Norme internationale considérée.

D.3.3 Chacun des arcs immédiatement au-dessous de **member-body** a pour valeur un indicatif de pays numérique à trois chiffres, tel que spécifié dans l'ISO 3166, qui identifie l'organisme national ISO de ce pays. La forme "NameForm" du composant d'identificateur d'objet n'est pas autorisée avec ces identificateurs.

D.3.4 Chacun des arcs immédiatement au-dessous de **identified-organization** a pour valeur un désignateur de code international (ICD, *international code designator*) affecté par l'autorité d'enregistrement pour l'ISO/CEI 6523 et qui identifie une organisation émettrice expressément enregistrée par cette autorité comme affectant des composants d'identificateur d'objet. Chacun des arcs immédiatement au-dessous a pour valeur un "code d'organisation" affecté par l'organisation émettrice conformément à l'ISO/CEI 6523.

D.4 Affectation conjointe de valeurs de composant d'identificateur d'objet

D.4.1 Les arcs au-dessous de **joint-iso-itu-t** ont des valeurs qui sont affectées et adoptées périodiquement par une autorité d'enregistrement établie par l'ISO/CEI et l'UIT-T afin d'identifier les domaines dans lesquels la normalisation est opérée conjointement par l'ISO/CEI et par l'UIT-T, conformément à la Rec. UIT-T X.662 | ISO/CEI 9834-3.

Annexe E

Exemples et indications

(Cette annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

La présente annexe présente des exemples d'utilisation de la notation ASN.1 pour décrire des structures de données (fictives). Elle comporte également des indications concernant l'utilisation des diverses caractéristiques de la notation ASN.1. Sauf mention contraire, on suppose se trouver dans un environnement d'étiquetage automatique **AUTOMATIC TAGS**.

E.1 Exemple d'un enregistrement "salarié"

L'utilisation de la notation ASN.1 est illustrée par un enregistrement simple concernant un salarié fictif.

E.1.1 Description informelle de l'enregistrement "salarié"

La structure de l'enregistrement "salarié" est décrite ci-dessous, avec sa valeur pour un individu donné.

Nom:	Jean P Martin
Fonction:	Directeur
Matricule:	51
Date d'embauche:	17 septembre 1971
Nom du conjoint:	Marie T Martin
Nombre d'enfants:	2
Renseignements enfant	
Nom:	Marc T Martin
Date de naissance:	11 novembre 1957
Renseignements enfant	
Nom:	Anne B Dubois
Date de naissance:	17 juillet 1959

E.1.2 Description en notation ASN.1 de la structure de l'enregistrement

La structure de chaque enregistrement "salarié" est décrite formellement au moyen de la notation normalisée des types de données.

```
PersonnelRecord ::= [APPLICATION 0] SET
{     name              Name,
      title             VisibleString,
      number            EmployeeNumber,
      dateOfHire        Date,
      nameOfSpouse      Name,
      children          SEQUENCE OF ChildInformation DEFAULT {}
}

ChildInformation ::= SET
{     name              Name,
      dateOfBirth       Date
}

Name ::= [APPLICATION 1] SEQUENCE
{     givenName         VisibleString,
      initial           VisibleString,
      familyName        VisibleString
}

EmployeeNumber ::= [APPLICATION 2] INTEGER

Date ::= [APPLICATION 3] VisibleString  --  AAAA MMJJ
```

Cet exemple illustre un aspect d'analyse de la syntaxe ASN.1. La structure syntaxique **DEFAULT** ne peut être appliquée qu'à un composant de type **SEQUENCE** ou **SET**; elle ne peut pas être appliquée à un composant de type **SEQUENCE-OF**. La

déclaration **DEFAULT { }** de **PersonnelRecord** s'applique par conséquent à la composante **children** et non à la notation **ChildInformation**.

E.1.3 Description en notation ASN.1 d'une valeur d'enregistrement

La valeur de l'enregistrement "salarié" de Jean P. Martin est décrite formellement ci-dessous en utilisant la notation normalisée des valeurs de données.

```
{     name             {givenName "John", initial "P", familyName "Smith"},
      title            "Director",
      number           51,
      dateOfHire       "19710917",
      nameOfSpouse     {givenName "Mary", initial "T", familyName "Smith"},
      children
      {  {name  {givenName "Ralph", initial "T", familyName "Smith"} ,
          dateOfBirth "19571111"},
         {name  {givenName "Susan", initial "B", familyName "Jones"} ,
          dateOfBirth "19590717" }
      }
}
```

ou, en notation de valeur XML:

```
person ::=
      <PersonnelRecord>
         <name>
            <givenName>John</givenName>
            <initial>P</initial>
            <familyName>Smith</familyName>
         </name>
         <title>Director</title>
         <number>51</number>
         <dateOfHire>19710917</dateOfHire>
         <nameOfSpouse>
            <givenName>Mary</givenName>
            <initial>T</initial>
            <familyName>Smith</familyName>
         </nameOfSpouse>
         <children>
            <ChildInformation>
               <name>
                  <givenName>Ralph</givenName>
                  <initial>T</initial>
                  <familyName>Smith</familyName>
               </name>
               <dateOfBirth>19571111</dateOfBirth>
            </ChildInformation>
            <ChildInformation>
               <name>
                  <givenName>Susan</givenName>
                  <initial>B</initial>
                  <familyName>Jones</familyName>
               </name>
               <dateOfBirth>19590717</dateOfBirth>
            </ChildInformation>
         </children>
      </PersonnelRecord>
```

E.2 Indications pour l'utilisation de la notation

La souplesse des types de données et de la notation formelle définis dans la présente Recommandation | Norme internationale permet de conceptualiser une grande variété de protocoles. Toutefois, cette souplesse peut parfois porter à confusion, en particulier lors d'une première utilisation de la notation. La présente annexe vise à minimiser les risques de confusion, en donnant des indications et des exemples concernant l'utilisation de la notation. Pour chacun des types de données prédéfinis, une ou plusieurs indications sont données concernant son emploi. Les types de chaînes de caractères (**VisibleString** par exemple) et les types définis dans les § 42 à 44 ne sont pas traités ici.

E.2.1 Type booléen (BooleanType)

E.2.1.1 Utiliser un type booléen pour modéliser les valeurs d'une variable logique (c'est-à-dire à deux états), par exemple la réponse par oui ou par non à une question.

EXEMPLE

```
Employed ::= BOOLEAN
```

E.2.1.2 Pour affecter un nom de référence à un type booléen, en choisir un qui représente l'état *vrai*.

EXEMPLE

```
Married ::= BOOLEAN
```

et non:

```
MaritalStatus ::= BOOLEAN
```

E.2.2 Type entier (IntegerType)

E.2.2.1 Utiliser un type entier pour modéliser les valeurs (de grandeur illimitée en pratique) d'une variable cardinale ou entière.

EXEMPLE

```
CheckingAccountBalance ::= INTEGER --  en centimes; négatif signifie
                                   --  découvert

balance CheckingAccountBalance ::= 0
```

ou, en notation de valeur XML:

```
balance ::= <CheckingAccountBalance>0</CheckingAccountBalance>
```

E.2.2.2 Définir les valeurs minimale et maximale autorisées d'un type entier comme nombres nommés.

EXEMPLE

```
DayOfTheMonth ::= INTEGER {first(1), last(31)}

today DayOfTheMonth ::= first

unknown DayOfTheMonth ::= 0
```

ou, en notation de valeur XML:

```
today ::= <DayOfTheMonth><first/></DayOfTheMonth>

unknown ::= <DayOfTheMonth>0</DayOfTheMonth>
```

A noter que les nombres nommés **first** et **last** ont été choisis à cause de leur contenu sémantique pour le lecteur, et n'excluent pas la possibilité d'avoir des **DayOfTheMonth** d'une valeur inférieure à 1, supérieure à 31 ou comprise entre 1 et 31.

Pour restreindre les valeurs de **DayOfTheMonth** aux seules valeurs **first** et **last**, il faudrait écrire:

```
DayOfTheMonth ::= INTEGER {first(1), last(31)} (first | last)
```

et pour restreindre les valeurs de **DayOfTheMonth** à toutes les valeurs comprises entre 1 et 31 inclusivement, il faudrait écrire:

```
DayOfTheMonth ::= INTEGER {first(1), last(31)} (first .. last)

dayOfTheMonth DayOfTheMonth ::= 4
```

ou, en notation de valeur XML:

```
dayOfTheMonth ::= <DayOfTheMonth>4</DayOfTheMonth>
```

E.2.3 Type énuméré (EnumeratedType)

E.2.3.1 Utiliser un type énuméré pour modéliser les valeurs d'une variable qui possède trois états ou plus. Attribuer des valeurs en partant de zéro si la seule contrainte est qu'elles soient distinctes.

EXEMPLE

```
DayOfTheWeek ::= ENUMERATED {sunday(0), monday(1), tuesday(2),
                             wednesday(3), thursday(4), friday(5),
                             saturday(6)}

firstDay DayOfTheWeek ::= sunday
```

ou, en notation de valeur XML:

```
firstDay ::= <DayOfTheWeek><sunday/></DayOfTheWeek>
```

A noter que les valeurs énumérées **sunday**, **monday**, etc., ont été choisies à cause de leur contenu sémantique pour le lecteur, et qu'à partir de ce moment, le type **DayOfTheWeek** est restreint à ces seules valeurs. Ainsi, une valeur ne peut se voir affecter que les seuls noms **sunday**, **monday**, etc., et non par exemple les entiers équivalents.

E.2.3.2 Utiliser un type énuméré extensible pour modéliser les valeurs d'une variable qui ne possède que deux états pour le moment, mais qui pourra en avoir d'autres dans une version ultérieure du protocole.

EXEMPLE

```
MaritalStatus ::= ENUMERATED {single, married}
          -- première version de la situation de famille
```

en prévision de:

```
MaritalStatus ::= ENUMERATED {single, married, …, widowed}
          -- deuxième version de la situation de famille
```

et par la suite:

```
MaritalStatus ::= ENUMERATED {single, married, …, widowed, divorced}
          -- troisième version de la situation de famille
```

E.2.4 Type réel

E.2.4.1 Utiliser un type réel pour représenter un nombre irrationnel.

EXEMPLE

```
AngleInRadians ::=  REAL

pi REAL ::= {mantissa 31415926535897932384626433832 79, base 10, exponent –30}
```

ou, dans l'autre notation de valeur pour **REAL**:

```
pi   REAL ::= 3.14159265358979323846264338327
```

ou, en notation de valeur XML:

```
pi ::=
      <REAL>
            3.14159265358979323846264338327
      </REAL>
```

E.2.4.2 Les concepteurs d'applications peuvent souhaiter garantir une interfonctionnement complet avec les valeurs réelles malgré les différences de représentation des nombres en virgule flottante dans les différents matériels, et décider par exemple d'utiliser des nombres en virgule flottante de longueur simple ou double dans une application X donnée. Ceci peut être réalisé de la manière suivante:

```
App-X-Real ::= REAL (WITH COMPONENTS {
                       mantissa (–16777215..16777215),
                       base (2),
                       exponent (–125..128) } )
/*
  Les expéditeurs ne transmettront pas de valeurs hors de ces intervalles;
  les destinataires conformes seront capables de recevoir et de traiter
  toutes les valeurs respectant ces limites d'intervalles.
*/

circonference Reel-App-X ::= {mantisse 16, base 2, exposant 1}
```

ou, en notation de valeur XML:

```
girth ::=
      <App-X-Real>
         32
      </App-X-Real>
```

E.2.5 Type chaîne binaire (BitStringType)

E.2.5.1 Utiliser un type chaîne binaire pour modéliser des données binaires dont le format et la longueur ne sont pas spécifiés, ou sont spécifiés ailleurs, et dont la longueur n'est pas nécessairement un multiple de huit bits.

EXEMPLE

```
G3FacsimilePage ::= BIT STRING
--  séquence binaire conforme à la Rec. UIT-T T.4

image G3FacsimilePage ::= '100110100100001110110'B

trailer BIT STRING ::= '0123456789ABCDEF'H

body1 G3FacsimilePage ::= '1101'B

body2 G3FacsimilePage ::= '1101000'B
```

ou, en notation de valeur XML:

```
image ::= <G3FacSimile>100110100100001110110</G3FacSimile>

trailer ::=
            <BIT_STRING>
                  0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011
                  1100 1101 1110 1111
            </BIT_STRING>

body1 ::= <G3FacSimile>1101</G3FacSimile>

body2 ::= <G3FacSimile>1101000</G3FacSimile>
```

A noter que **body1** et **body2** sont des valeurs abstraites différentes car les bits à 0 en fin de chaîne sont significatifs (en raison de l'absence d'une liste "NamedBitList" dans la définition de **G3FacsimilePage**).

E.2.5.2 Utiliser un type chaîne binaire avec une contrainte de taille pour modéliser les valeurs d'un champ binaire de taille fixe.

EXEMPLE

```
BitField ::= BIT STRING (SIZE (12))

map1 BitField ::= '100110100100'B

map2 BitField ::= '9A4'H

map3 BitField ::= '1001101001'B    --  forme illicite: elle transgresse
                                   --  la contrainte de taille
```

ou, en notation de valeur XML:

```
map1 ::= <BitField>100110100100</BitField>
```

A noter que **map1** et **map2** sont une même valeur abstraite, les 4 bits en fin de **map2** ne sont pas significatifs.

E.2.5.3 Utiliser un type chaîne binaire pour modéliser les valeurs d'un **champ d'indicateurs**, d'un ensemble ordonné de variables logiques correspondant à une collection ordonnée d'objets et indiquant chacun si une condition particulière est remplie par l'objet correspondant de la collection.

```
DaysOfTheWeek ::= BIT STRING {
            sunday(0), monday (1), tuesday(2),
            wednesday(3), thursday(4), friday(5),
            saturday(6) } (SIZE (0..7))

sunnyDaysLastWeek1 DaysOfTheWeek ::= {sunday, monday, wednesday}
      sunnyDaysLastWeek2 DaysOfTheWeek ::= '1101'B
      sunnyDaysLastWeek3 DaysOfTheWeek ::= '1101000'B

sunnyDaysLastWeek4 DaysOfTheWeek ::= '11010000'B -- Illicite
```

ou, en notation de valeur XML:

```
sunnyDaysLastWeek1 ::=
      <DaysOfTheWeek>
          <sunday/><monday/><wednesday/>
      </DaysOfTheWeek>

sunnyDaysLastWeek2 ::= <DaysOfTheWeek>1101</DaysOfTheWeek>

sunnyDaysLastWeek3 ::= <DaysOfTheWeek>1101000</DaysOfTheWeek>
```

A noter que si la longueur de la chaîne binaire est inférieure à 7, cela signifie que les jours correspondant aux bits manquants ont été maussades; les trois premières valeurs ont donc la même valeur abstraite.

E.2.5.4 Utiliser un type chaîne binaire pour modéliser les valeurs d'un *champ d'indicateurs*, d'un ensemble ordonné de taille fixe de variables logiques correspondant à une collection ordonnée d'objets et indiquant chacun si une condition particulière est remplie par l'objet correspondant de la collection.

```
DaysOfTheWeek ::= BIT STRING {
            sunday(0), monday (1), tuesday(2),
            wednesday(3), thursday(4), friday(5),
            saturday(6) } (SIZE (7))
sunnyDaysLastWeek1 DaysOfTheWeek ::= {sunday, monday, wednesday}
sunnyDaysLastWeek2 DaysOfTheWeek ::= '1101'B -- forme illicite
                                     -- elle transgresse la contrainte
                                     -- de taille
sunnyDaysLastWeek3 DaysOfTheWeek ::= '1101000'B

sunnyDaysLastWeek4 DaysOfTheWeek ::= '11010000'B --  forme illicite
                                        -- elle transgresse la contrainte
                                        -- de taille
```

A noter que la première et la troisième valeur ont la même valeur abstraite.

E.2.5.5 Utiliser un type chaîne binaire avec des bits nommés pour modéliser les valeurs d'un ensemble de variables logiques liées.

EXEMPLE

```
PersonalStatus ::= BIT STRING
            {married(0), employed(1), veteran(2), collegeGraduate(3)}

billClinton PersonalStatus ::= {married, employed, collegeGraduate}

hillaryClinton PersonalStatus ::= '110100'B
```

ou, en notation de valeur XML:

```
billClinton ::=
      <PersonalStatus>
         <married/>
         <employed/>
         <collegeGraduate/>
      </PersonalStatus>

hillaryClinton ::= <PersonalStatus>110100</PersonalStatus>
```

A noter que **billClinton** et **hillaryClinton** ont les mêmes valeurs abstraites.

E.2.6 Type chaîne d'octets (OctetStringType)

E.2.6.1 Utiliser un type chaîne d'octets pour modéliser des données binaires dont le format et la longueur ne sont pas spécifiés, ou sont spécifiés ailleurs, et dont la longueur est un multiple de huit bits.

EXEMPLE

```
G4FacsimileImage ::= OCTET STRING
--  séquence d'octets conforme à la Rec. UIT-T T.5 et la Rec. CCITT T.6

image G4FacsimilePage ::= '3FE2EBAD471005'H
```

ou, en notation de valeur XML:

```
image ::= <G4FacSimileImage>3FE2EBAD471005</G4FacSimileImage>
```

E.2.6.2 Utiliser un type chaîne de caractères à alphabet restreint approprié de préférence à un type chaîne d'octets lorsque cela est possible.

EXEMPLE

```
Surname ::= PrintableString

president Surname ::= "Clinton"
```

ou, en notation de valeur XML:

```
president ::= <Surname>Clinton</Surname>
```

E.2.7 Types UniversalString, BMPString et UTF8String

Utiliser le type **BMPString** ou le type **UTF8String** pour modéliser toute chaîne informationnelle constituée des seuls caractères de la table multilingue (BMP, *basic multilingual plane*) de l'ISO/CEI 10646-1, et le type **UniversalString** ou **UTF8String** pour modéliser toute chaîne constituée de caractères de l'ISO/CEI 10646-1 mais non limitée à la table BMP.

E.2.7.1 Utiliser les déclarations **Level1** ou **Level2** pour indiquer si des restrictions sont imposées à l'utilisation de caractères de combinaison.

EXEMPLE

```
RussianName ::= Cyrillic (Level1)
      -- Un NomRusse n'utilise pas de caractères de combinaison

SaudiName ::= BasicArabic (SIZE (1..100) ^ Level2)
      --  NomSaoudien utilise un sous-ensemble de caractères de combinaison
```

Représentation de la lettre Σ:

```
greekCapitalLetterSigma BMPString ::= {0, 0, 3, 163}
```

ou, en notation de valeur XML:

```
greekCapitalLetterSigma ::= <BMPString>&#x03a3;</BMPString>
```

Représentation de la chaîne "f → ∝":

```
rightwardsArrow UTF8String ::= {0, 0, 33, 146}

infinity UTF8String ::= {0, 0, 34, 30}

property UTF8String ::= {"f  ", rightwardsArrow, " ", infinity}
```

ou, en notation de valeur XML:

```
property ::= <UTF8String>f &#x2192; &#x221E;</UTF8String>
```

E.2.7.2 Une collection de caractères peut être étendue et transformée en un sous-ensemble sélectionné (c'est-à-dire en lui ajoutant tous les caractères du jeu latin de base) au moyen du symbole de réunion "UnionMark" (voir le § 46).

EXEMPLE

```
KatakanaAndBasicLatin ::= UniversalString (FROM (Katakana | BasicLatin))
```

E.2.8 Type chaîne de caractères (CharacterStringType)

Utiliser le type chaîne de caractères à alphabet non restreint pour modéliser toute chaîne informationnelle qui ne peut être modélisée par l'un des types de chaînes de caractères à alphabet restreint. Ne pas oublier de spécifier le répertoire de caractères et leur codage en octets.

EXEMPLE

```
PackedBCDString ::= CHARACTER STRING (WITH COMPONENTS {
                                                    identification  (WITH
COMPONENTS {
                                                       fixed  PRESENT })
      /* Les syntaxes abstraite et de transfert correspondent aux
         identificateurs
         packedBCDString-AbstractSyntaxId et
         packedBCDString-TransferSyntaxId définis ci-dessous.
      */
                            } )
```

```
        /* valeur d'identificateur d'objet pour une syntaxe abstraite de
           caractères (jeu de caractères) dont l'alphabet est constitué des
            chiffres de 0 à 9.
        */
        PackedBCDString-AbstractSyntaxId  OBJECT  IDENTIFIER ::=
              { joint-iso-itu-t asn1(1) examples(123) packedBCD(2) charSet(0) }

        /* valeur d'identificateur d'objet pour une syntaxe de transfert de
           caractères qui condense deux chiffres par octet, chaque octet étant
           codé de 0000 à 1001, 1111₂ servant au remplissage.
        */
        PackedBCDString-TransferSyntaxId  OBJECT IDENTIFIER ::=
              { joint-iso-itu-t asn1(1) examples(123) packedBCD(2)
                    characterTransferSyntax(1) }
/* Le codage d'une valeur du type PackedBCDString ne comportera que le codage
   défini des caractères avec le champ de longueur nécessaire, ainsi que le
   champ étiquette en cas de codage selon les règles de codage de base BER.
   Les valeurs d'identificateurs d'objets ne sont pas transmises, la notation
   spécifiant le mode "fixe".
*/
```

ou, en notation de valeur XML:

```
packedBCDString-AbstractSyntaxId ::=
      <OBJECT_IDENTIFIER>
            joint-iso-itu-t.asn1(1).examples(123).packedBCD(2).charSet(0)
      </OBJECT_IDENTIFIER>


packedBCDString-TransferSyntaxId ::=
 <OBJECT_IDENTIFIER>

joint-iso-itu-t.asn1(1).examples(123).packedBCD(2).characterTransferSyntax(1)
 </OBJECT_IDENTIFIER>
```

ou:

```
packedBCDString-AbstractSyntaxId ::=
      <OBJECT_IDENTIFIER>2.1.123.2.0</OBJECT_IDENTIFIER>

PackedBCDString-TransferSyntaxId ::=
      <OBJECT_IDENTIFIER>2.1.123.2.1</OBJECT_IDENTIFIER>
```

NOTE – Les règles de codage ne codent pas toujours les valeurs du type chaîne de caractères **CHARACTER STRING** sous une forme qui comporte nécessairement des valeurs d'identificateurs d'objets, bien que de tels identificateurs garantissent la préservation de la valeur abstraite dans le codage.

E.2.9 Type néant (NullType)

Utiliser un type néant pour indiquer l'absence effective d'un élément d'une séquence.

EXEMPLE

```
PatientIdentifier ::= SEQUENCE {
      name              VisibleString,
      roomNumber        CHOICE {
            room        INTEGER,
            outPatient  NULL  --  s'il s'agit d'un malade non
                              --  hospitalisé  --
      }
}

lastPatient PatientIdentifier ::= {
      name  "Jane Doe",
      roomNumber  outPatient : NULL
}
```

ou, en notation de valeur XML:

```
lastPatient ::=
      <PatientIdentifier>
         <name>Jane Doe</name>
         <roomNumber><outPatient/></roomNumber>
      </PatientIdentifier>
```

E.2.10 Types séquence et séquence-de (Sequence, SequenceOfType)

E.2.10.1 Utiliser un type séquence-de pour modéliser une collection de variables appartenant à un même type, en nombre élevé ou imprévisible, et dont l'ordre est significatif.

EXEMPLE

```
NamesOfMemberNations ::= SEQUENCE OF VisibleString
--  par ordre alphabétique

firstTwo  NamesOfMemberNations ::= {"Australia", "Austria"}
```

ou, en utilisant l'identificateur optionnel:

```
NamesOfMemberNations2 ::= SEQUENCE OF memberNation VisibleString
--  par ordre alphabétique

firstTwo2  NamesOfMemberNations2 ::=
      {memberNation "Australia", memberNation "Austria"}
```

En notation de valeur XML, les deux valeurs ci-dessus sont les suivantes:

```
firstTwo ::=
      <NamesOfMemberNations>
          <VisibleString>Australia</VisibleString>
          <VisibleString>Austria</VisibleString>
      </NamesOfMemberNations>

firstTwo2 ::=
      <NamesOfMemberNations2>
          <memberNation>Australia</memberNation>
          <memberNation>Austria</memberNation>
      </NamesOfMemberNations2>
```

E.2.10.2 Utiliser un type séquence pour modéliser une collection de variables de types différents, en nombre limité et connu, et dont l'ordre est significatif, la composition de la collection ayant peu de chance de changer d'une version du protocole à la suivante.

EXEMPLE

```
NamesOfOfficers ::= SEQUENCE {
            president               VisibleString,
            vicePresident           VisibleString,
            secretary               VisibleString}

acmeCorp NamesOfOfficers ::= {
            president               "Jane Doe",
            vicePresident           "John Doe",
            secretary               "Joe Doe"}
```

ou, en notation de valeur XML:

```
acmeCorp ::=
      <NamesOfOfficers>
          <president>Jane Doe</president>
           <vicePresident>John Doe</vicePresident>
           <secretary>Joe Doe</secretary>
      </NamesOfOfficers>
```

E.2.10.3 Utiliser un type séquence inextensible pour modéliser une collection de variables de types différents, en nombre connu et limité, et dont l'ordre est significatif, la composition de cette collection ayant peu de chance de changer d'une version du protocole à la suivante.

EXEMPLE

```
Credentials ::= SEQUENCE {
            userName                VisibleString,
            password                VisibleString,
            accountNumber           INTEGER}
```

E.2.10.4 Utiliser un type séquence extensible pour modéliser une collection de variables dont l'ordre est significatif et dont le nombre actuel est faible et connu mais dont il est prévu qu'il puisse être augmenté.

EXEMPLE

```
Record ::= SEQUENCE {  -- première version de protocole contenant
                       -- "Enregistrement" (Record)
      userName                    VisibleString,
      password                    VisibleString,
      accountNumber               INTEGER,
      ...,
      ...
}
```

en prévision de:

```
Record ::= SEQUENCE {  -- deuxième version de protocole contenant
                       -- "Enregistrement" (Record)

userName                      VisibleString,
      password                    VisibleString,
      accountNumber               INTEGER,
      ...,
      [[2:             -- ajout d'extension inséré en version 2 de protocole
         lastLoggedIn             GeneralizedTime OPTIONAL,
         minutesLastLoggedIn      INTEGER
      ]],
      ...
}
```

et par la suite (dans la version 3 du protocole, aucun ajout n'est fait à **Record**):

```
Record ::= SEQUENCE {  -- deuxième version de protocole contenant
                       -- "Enregistrement" (Record)
      userName                    VisibleString,
      password                    VisibleString,
      accountNumber               INTEGER,
      ...,
      [[2:             -- ajout d'extension inséré en version 2 de protocole
         lastLoggedIn             GeneralizedTime OPTIONAL,
         minutesLastLoggedIn      INTEGER
      ]],
      [[4:             -- ajout d'extension inséré en version 3 de protocole

         certificate              Certificate,
         thumb                    ThumbPrint OPTIONAL
      ]],
      ...
      }
```

E.2.11 Types ensemble et ensemble-de (SetType, SetOfType)

E.2.11.1 Utiliser un type ensemble pour modéliser une collection de variables en nombre connu et limité, et dont l'ordre n'est pas significatif. En l'absence d'étiquetage automatique, identifier chaque variable par un étiquetage spécifique au contexte comme le montre l'exemple suivant (en étiquetage automatique, les étiquettes ne sont pas nécessaires).

EXEMPLE

```
UserName ::= SET {
           personalName                [0] VisibleString,
           organizationName            [1] VisibleString,
           countryName                 [2] VisibleString}

user UserName ::= {
           countryName                 "Nigeria",
           personalName                  "Jonas Maruba",
           organizationName            "Meteorology, Ltd."}
```

ou, en notation de valeur XML:

```
user ::=
      <UserName>
         <countryName>Nigeria</countryName>
         <personalName>Jonas Maruba</personalName>
         <organizationName>Meteorology, Ltd.</organizationName>
      </UserName>
```

E.2.11.2 Utiliser un type ensemble avec déclaration **OPTIONAL** pour modéliser une collection de variables qui est un sous-ensemble (strict ou non strict) d'une autre collection de variables, dont le nombre est connu et raisonnablement petit, et dont l'ordre n'est pas significatif. En l'absence d'étiquetage automatique, identifier chaque variable par un étiquetage spécifique au contexte comme le montre l'exemple suivant (en étiquetage automatique, les étiquettes ne sont pas nécessaires).

EXEMPLE

```
UserName ::= SET {
      personalName                    [0] VisibleString,
      organizationName                [1] VisibleString OPTIONAL
          --  par défaut, nom de l'organisation locale  -- ,
      countryName                     [2] VisibleString OPTIONAL
          --  par défaut, nom du pays local  --  }
```

E.2.11.3 Utiliser un type ensemble extensible pour modéliser une collection de variables dont la composition risque de changer d'une version du protocole à la suivante. On fait l'hypothèse, dans ce qui suit, qu'une déclaration **AUTOMATIC TAGS** a été faite lors de la définition du module.

EXEMPLE

```
UserName ::= SET {
      personalName          VisibleString,        -- première version de
                                                  -- "NomUtilisateur"
                                                  -- (UserName)
      organizationName      VisibleString OPTIONAL ,
      countryName           VisibleString OPTIONAL,
      ...,
      ...
}

user UserName ::= { personalName  "Jonas Maruba" }
```

ou, en notation de valeur XML:

```
user ::=
      <UserName>
         <personalName>Jonas Maruba</personalName>
      </UserName>
```

en prévision de:

```
UserName ::= SET {          -- deuxième version de "NomUtilisateur"
                            -- (UserName)
      personalName               VisibleString,
      organizationName           VisibleString OPTIONAL,
      countryName                VisibleString OPTIONAL,
      ...,
      [[2:                  -- ajout d'extension inséré en version 2 de
                            -- protocole
         internetEmailAddress    VisbleString,
         faxNumber               VisibleString OPTIONAL
      ]],
      ...
}

user  UserName ::= {
      personalName              "Jonas Maruba",
      internetEmailAddress      "jonas@meteor.ngo.com"
}
```

ou, en notation de valeur XML:

```
 user ::=
<UserName>
   <personalName>Jonas Maruba</personalName>
   <internetEmailAddress>jonas@meteor.ngo.com</internetEmailAddress>
</UserName>
```

et par la suite (dans les versions 3 et 4 du protocole, aucun ajout n'a été apporté à **UserName**):

```
UserName ::= SET {           -- cinquième version de protocole contenant
                             -- "nomUtilisateur"
      personalName                VisibleString,
      organizationName            VisibleString OPTIONAL,
      countryName                 VisibleString OPTIONAL,
      ...,
      [[2:                        -- ajout d'extension inséré en version 2
         internetEmailAddress     VisbleString,
         faxNumber                VisibleString OPTIONAL
      ]],
      [[5:                        -- ajout d'extension inséré en version 5
      phoneNumber                 VisibleString OPTIONAL
      ]],
      ...
}

user  UserName ::= {
      personalName                "Jonas Maruba",
      internetEmailAddress        "jonas@meteor.ngo.com"
}
```

ou, en notation de valeur XML:

```
user ::=
      <UserName>
         <personalName>Jonas Maruba</personalName>
         <internetEmailAddress>jonas@meteor.ngo.com</internetEmailAddress>
      </UserName>
```

E.2.11.4 Utiliser un type ensemble-de pour modéliser une collection de variables appartenant à un même type et dont l'ordre n'est pas significatif.

EXEMPLE

```
Keywords ::= SET OF VisibleString  --  en ordre arbitraire

someASN1Keywords Keywords ::= {"INTEGER", "BOOLEAN", "REAL"}
```

ou, en utilisant l'identificateur optionnel:

```
Keywords2 ::= SET OF keyword VisibleString  --  en ordre arbitraire

someASN1Keywords2 Keywords2 ::= {keyword "INTEGER", keyword "BOOLEAN",
      keyword "REAL"}
```

En utilisant la notation de valeur XML, les deux valeurs ci-dessus sont les suivantes:

```
someASN1Keywords ::=
      <Keywords>
         <VisibleString>INTEGER</VisibleString>
         <VisibleString>BOOLEAN</VisibleString>
         <VisibleString>REAL</VisibleString>
      </Keywords>

someASN1Keywords2 ::=
      <Keywords2>
         <keyword>INTEGER</keyword>
         <keyword>BOOLEAN</keyword>
         <keyword>REAL</keyword>
      </Keywords2>
```

E.2.12 Type étiqueté (TaggedType)

Avant l'introduction de la déclaration d'étiquetage automatique **AUTOMATIC TAGS**, les spécifications ASN.1 comportaient souvent des étiquettes. Les points suivants décrivent la manière dont l'étiquetage était typiquement

appliqué. Avec l'introduction de l'étiquetage automatique, les auteurs de spécifications ASN.1 n'ont plus besoin d'utiliser la notation d'étiquettes, mais ceux qui apportent des modifications à des spécifications anciennes doivent en comprendre le fonctionnement. Les nouveaux utilisateurs de la notation ASN.1 sont encouragés à utiliser l'étiquetage automatique car celui-ci rend la notation plus lisible.

E.2.12.1 Les étiquettes de la classe universelle ne sont utilisées que dans la présente Recommandation | Norme internationale. La notation **[UNIVERSAL 30]** (par exemple) n'est indiquée que pour assurer la précision de la définition des types utiles "UsefulTypes" (voir § 41.1). Elle ne doit pas être utilisée ailleurs.

E.2.12.2 Un cas fréquent d'utilisation des étiquettes est l'affectation d'une seule étiquette de la classe application à l'ensemble de la spécification, cette étiquette servant à identifier un type qui s'avère être largement utilisé un peu partout dans la spécification. On utilise souvent aussi une seule étiquette de la classe application (une seule fois) pour étiqueter les types de la structure **CHOICE** la plus externe d'une application, en assurant l'identification des différents messages par l'étiquette de la classe application. Le premier cas est illustré dans l'exemple ci-dessous:

EXEMPLE

```
FileName ::= [APPLICATION 8] SEQUENCE {
            directoryName                VisibleString,
            directoryRelativeFileName    VisibleString}
```

E.2.12.3 L'étiquetage propre au contexte est fréquemment appliqué de manière algorithmique à tous les composants d'une structure **SET**, **SEQUENCE** ou **CHOICE**. A noter toutefois que l'étiquetage automatique **AUTOMATIC TAGS** décharge le concepteur de cette tâche.

EXEMPLE

```
CustomerRecord ::= SET {
            name                         [0] VisibleString,
            mailingAddress               [1] VisibleString,
            accountNumber                [2] INTEGER,
            balanceDue                   [3] INTEGER  --  en centimes  --}

CustomerAttribute ::= CHOICE {
            name                         [0] VisibleString,
            mailingAddress               [1] VisibleString,
            accountNumber                [2] INTEGER,
            balanceDue                   [3] INTEGER  --  en centimes  --}
```

E.2.12.4 L'étiquetage en classe privée ne devrait normalement pas être utilisé dans les spécifications internationales (bien que ceci ne puisse être interdit). Les applications développées par les entreprises devraient normalement utiliser les classes d'étiquetage application et propres au contexte. Mais une situation particulière peut se présenter dans laquelle une spécification propre à une entreprise apporte une extension à une spécification internationale; dans une telle circonstance, l'utilisation d'étiquettes de la classe privée a l'avantage de protéger partiellement la spécification propre à l'entreprise d'une modification de la spécification internationale.

EXEMPLE

```
AcmeBadgeNumber ::= [PRIVATE 2] INTEGER

badgeNumber AcmeBadgeNumber ::= 2345
```

ou, en notation de valeur XML:

```
badgeNumber ::= <AcmeBadgeNumber>2345</AcmeBadgeNumber>
```

E.2.12.5 L'ajout de la déclaration **IMPLICIT** à chaque étiquette n'est généralement trouvé que sur les spécifications les plus anciennes. Les règles de codage de base (BER) produisent une représentation moins compacte avec l'étiquetage explicite qu'avec l'étiquetage implicite. Les règles de codage compact (PER) aboutissent à la même compacité de codage dans les deux cas. Avec les règles BER et l'étiquetage explicite, il y a une meilleure visibilité du type sous-jacent (**INTEGER**, **REAL**, **BOOLEAN**, etc.) dans les données codées. Les présentes directives utilisent l'étiquetage implicite dans les exemples chaque fois qu'il est licite de le faire. Selon les règles de codage, cela peut aboutir à une représentation compacte, particulièrement recherchée dans certaines applications. Dans d'autres applications, la compacité peut être moins importante par exemple que la possibilité d'effectuer une solide vérification de type. Dans une telle situation, on peut recourir à l'étiquetage explicite.

EXEMPLE

```
CustomerRecord ::= SET {
            name                    [0] IMPLICIT VisibleString,
            mailingAddress          [1] IMPLICIT VisibleString,
            accountNumber           [2] IMPLICIT INTEGER,
            balanceDue              [3] IMPLICIT INTEGER  --  en centimes  --}

CustomerAttribute ::= CHOICE {
            name                    [0] IMPLICIT VisibleString,
            mailingAddress          [1] IMPLICIT VisibleString,
            accountNumber           [2] IMPLICIT INTEGER,
            balanceDue              [3] IMPLICIT INTEGER  --  en centimes  --}
```

E.2.12.6 Concernant l'étiquetage pour les nouvelles spécifications ASN.1 d'utilisateur faisant référence à la présente Recommandation | Norme internationale, l'indication à suivre est très simple: NE PAS ETIQUETER. Mettre la déclaration **AUTOMATIC TAGS** dans l'en-tête du module, et ne plus s'en occuper. S'il s'avère nécessaire d'ajouter un nouveau composant à une structure **SET**, **SEQUENCE** ou **CHOICE** dans une version ultérieure, l'ajouter à la fin.

E.2.13 Type choix

E.2.13.1 Utiliser un type choix **CHOICE** pour modéliser une variable choisie dans un ensemble de variables en nombre connu et limité.

EXEMPLE

```
FileIdentifier ::= CHOICE {
            relativeName     VisibleString,
          --  nom du fichier (par exemple, "MarchProgressReport"
          --  "RapportAvancementMars")
            absoluteName     VisibleString,
          --  noms du fichier et du répertoire qui le contient
          --  (par exemple,
          --  "<Williams>MarchProgressReportRapportAvancementMars")
            serialNumber     INTEGER
          --  identificateur affecté par le système au fichier  --}

file FileIdentifier ::= serialNumber : 106448503
```

ou, en notation de valeur XML:

```
fileIdentifier ::=
          <FileIdentifier>
             <serialNumber>106448503</serialNumber>
          </FileIdentifier>
```

E.2.13.2 Utiliser un type choix **CHOICE** extensible pour modéliser une variable choisie dans un ensemble de variables dont la composition risque de changer d'une version du protocole à la suivante.

EXEMPLE

```
FileIdentifier ::= CHOICE {                        -- première version
FileIdentifier d'IdentificateurDeFichier
            relativeName    VisibleString,

            absoluteName    VisibleString,

            ...,   ...
}

fileId1  FileIdentifier ::= relativeName : "MarchProgressReport.doc"
```

ou, en notation de valeur XML:

```
fileId1 ::=
      <FileIdentifier>
         <relativeName>MarchProgressReport.doc</relativeName>
      </FileIdentifier>
```

en prévision de:

```
      FileIdentifier ::= CHOICE {       -- deuxième version FileIdentifier
d'IdentificateurDeFichier
            relativeName     VisibleString,
            absoluteName     VisibleString,
            ...,
            serialNumber     INTEGER,    -- ajout d'extension inséré en version 2
            ...
      }

      fileId1  FileIdentifier ::= relativeName : "MarchProgressReport.doc"

      fileId2  FileIdentifier ::= serialNumber : 214
```

ou, en notation de valeur XML:

```
fileId1 ::=
      <FileIdentifier>
         <relativeName>MarchProgressReport.doc</relativeName>
      </FileIdentifier>

fileId2 ::=
      <FileIdentifier>
         <serialNumber>214</serialNumber>
      </FileIdentifier>
```

et par la suite:

```
      FileIdentifier ::= CHOICE {       -- troisième version FileIdentifier
d'IdentificateurDeFichier
            relativeName     VisibleString,
            absoluteName     VisibleString,
            ...,
            serialNumber     INTEGER,    -- ajout d'extension inséré en version 2
            [[                           -- ajout d'extension inséré en version 3
               vendorSpecificVendorExt,
               unidentified  NULL
            ]],
            ...
      }

      fileId1  FileIdentifier ::= relativeName : "MarchProgressReport.doc"

      fileId2  FileIdentifier ::= serialNumber : 214

      fileId3  FileIdentifier ::= unidentified : NULL
```

ou, en notation de valeur XML:

```
      fileId1 ::=
            <FileIdentifier>
               <relativeName>MarchProgressReport.doc</relativeName>
            </FileIdentifier>

      fileId2 ::=
            <FileIdentifier>
               <serialNumber>214</serialNumber>
            </FileIdentifier>

      fileId3 ::=
            <FileIdentifier>
               <unidentified/>
            </FileIdentifier>
```

E.2.13.3 Utiliser un type choix extensible avec un seul type lorsqu'il est envisagé d'avoir ultérieurement plusieurs types possibles.

EXEMPLE

```
      Greeting ::= CHOICE {             -- première version de "Vœux"
            postCard         VisibleString,
            ...,
            ...
      }
```

en prévision de:

```
Greeting ::= CHOICE {                -- deuxième version de "Vœux"
      postCard    VisibleString,
      ...,
      [[2:                           -- ajout d'extension inséré en version 2
         audio          Audio,
         video          Video
      ]],
      ...
}
```

E.2.13.4 Utiliser plusieurs signes "deux-points" lorsqu'un type choix est imbriqué dans un autre type choix.

EXEMPLE

```
Greeting ::= [APPLICATION 12] CHOICE {
            postCard          VisibleString,
            recording         Voice }

Voice ::= CHOICE {
            english           OCTET STRING,
            swahili           OCTET STRING }

myGreeting Greeting ::= recording : english : '019838547E0'H
```

ou, en notation de valeur XML:

```
myGreeting ::=
      <Greeting>
         <recording><english>019838547E0</english></recording>
      </Greeting>
```

E.2.14 Type sélection

E.2.14.1 Utiliser un type sélection pour modéliser une variable dont le type est celui d'une des formes particulières d'un choix **CHOICE** défini antérieurement.

E.2.14.2 Considérons la définition:

```
FileAttribute ::= CHOICE {
            date-last-used    INTEGER,
            file-name         VisibleString}
```

La définition suivante est alors possible:

```
AttributeList ::= SEQUENCE {
            first-attribute   date-last-used  < FileAttribute,
            second-attribute  file-name < FileAttribute }
```

avec la notation de valeur possible suivante:

```
listOfAttributes AttributeList ::=  {
            first-attribute   27,
            second-attribute  "PROGRAM" }
```

ou, en notation de valeur XML:

```
listOfAttributes ::=
      <AttributeList>
         <first-attribute>27</first-attribute>
         <second-attribute>PROGRAM</second-attribute>
      </AttributeList>
```

E.2.15 Type champ de classe d'objets

E.2.15.1 Utiliser un type champ de classe d'objets pour identifier un type défini au moyen d'une classe d'objets informationnels (voir la Rec. UIT-T X.681 | ISO/CEI 8824-2). Par exemple, les champs de la classe d'objets informationnels **ATTRIBUTE** peuvent être utilisés pour définir un type **Attribute**.

EXEMPLE

```
ATTRIBUTE ::= CLASS {
      &AttributeType,
      &attributeId            OBJECT IDENTIFIER UNIQUE
            }
```

```
Attribute ::= SEQUENCE {
      attributeID      ATTRIBUTE.&attributeId,   --  normalement, une contrainte
      est imposée
      attributeValue   ATTRIBUTE.&AttributeType     --  normalement, une contrainte
      est imposée
            }
```

ATTRIBUTE.&attributeId et **ATTRIBUTE.&AttributeType** sont tous deux des types de champ de classe d'objets, en ce sens qu'il s'agit de types définis par référence à une classe d'objets informationnels **ATTRIBUTE**. Le type **ATTRIBUTE.&attributeId** est fixé parce qu'il est explicitement défini dans ATTRIBUTE comme identificateur d'objet **OBJECT IDENTIFIER**. Par contre, le type **ATTRIBUTE.&AttributeType** peut contenir une valeur de n'importe quel type défini en ASN.1, puisque son type n'est pas fixé dans la définition de la classe d'objets informationnels **ATTRIBUTE**. Une notation qui possède cette propriété de pouvoir véhiculer une valeur de n'importe quel type est dite "notation de type ouvert". Ainsi, **ATTRIBUTE.&AttributeType** est un type ouvert.

E.2.16 Type valeur pdv imbriquée (EmbeddedPDVType)

E.2.16.1 Utiliser un type valeur pdv imbriquée pour modéliser une variable dont le type n'est pas spécifié, ou l'est ailleurs sans restriction quant à la notation utilisée pour le spécifier.

EXEMPLE

```
FileContents ::= EMBEDDED PDV

DocumentList ::= SEQUENCE OF document EMBEDDED PDV
```

E.2.17 Type externe (ExternalType)

Le type externe est similaire au type valeur pdv imbriquée, mais dispose d'un nombre plus restreint de possibilités d'identification. On préférera généralement dans les nouvelles spécifications utiliser le type valeur pdv imbriquée à cause de sa plus grande souplesse et parce que certaines règles de codage codent les valeurs de manière plus efficace.

E.2.18 Type instance-de (InstanceOfType)

E.2.18.1 Utiliser une déclaration instance-de pour spécifier un type contenant un champ identificateur d'objet et une valeur de type ouvert dont le type est déterminé par cet identificateur d'objet. Le type instance-de ne peut être utilisé que si l'association entre la valeur d'identificateur d'objet et le type est spécifiée au moyen d'un objet informationnel d'une classe dérivée de **TYPE-IDENTIFIER** (voir la Rec. UIT-T X.681 | ISO/CEI 8824-2, Annexes A et C).

EXEMPLE

```
ACCESS-CONTROL-CLASS ::= TYPE-IDENTIFIER

Get-Invoke ::= SEQUENCE {
      objectClass      ObjectClass,
      objectInstance   ObjectInstance,
      accessControl    INSTANCE OF ACCESS-CONTROL-CLASS, --  normalement, une
                                                         --  contrainte est
                                                         --  imposée
      attributeID      ATTRIBUTE.&attributeId
      }
```

Get-Invoke est alors équivalent à:

```
Get-Invoke ::= SEQUENCE {
      objectClass      ObjectClass,
      objectInstance   ObjectInstance,
      accessControl    [UNIVERSAL 8] IMPLICIT SEQUENCE {
            type-id    ACCESS-CONTROL-CLASS.&id,         --  normalement, une
                                                         --  contrainte est
                                                         --  imposée
            value      [0] ACCESS-CONTROL-CLASS.&Type    --  normalement, une
                                                         --  contrainte est
                                                         --  imposée
      },
      attributeID ATTRIBUTE.&attributeId
}
```

La véritable utilité du type instance-de n'apparaît que lorsqu'il est contraint au moyen d'un ensemble d'objets informationnels, mais un tel exemple sort du cadre de la présente Recommandation | Norme internationale. Se reporter à la Rec. UIT-T X.682 | ISO/CEI 8824-3 pour la définition de l'ensemble d'objets informationnels, et à l'Annexe A de la

Rec. UIT-T X.682 | ISO/CEI 8824-3 pour la manière d'utiliser un ensemble d'objets informationnels pour contraindre un type instance-de.

E.2.19 Identificateur d'objet relatif

E.2.19.1 Utiliser un type identificateur d'objet relatif pour transmettre des valeurs d'identificateur d'objet sous une forme plus compacte lorsque la partie antérieure de la valeur d'identificateur d'objet est connue. Trois situations peuvent se présenter:

a) La partie antérieure de la valeur d'identificateur d'objet est fixe pour une spécification donnée (il s'agit d'une norme propre à l'industrie) et tous les identificateurs d'objet (OID) sont relatifs à un identificateur d'objet attribué à un organisme de normalisation. En pareil cas, utiliser:

```
RELATIVE-OID      --  La valeur d'identificateur d'objet relatif est
                  --  relative à {iso identified-organization set(22)}
```

b) La partie antérieure de la valeur d'identificateur d'objet est souvent une valeur qui est connue au moment de la spécification, mais qui peut parfois être une valeur plus générale. En pareil cas, utiliser:

```
CHOICE
      {a    RELATIVE-OID     -- La valeur est relative à {1 3 22} --,
       b    OBJECT IDENTIFIER  -- Toute autre valeur d'identificateur
                               -- d'objet --}
```

c) La partie antérieure de la valeur d'identificateur d'objet n'est pas connue avant le moment de la communication, mais sera souvent commune aux nombreuses valeurs qui doivent être envoyées et sera très souvent une valeur connue au moment de la spécification. En pareil cas, utiliser (par exemple):

```
SEQUENCE
      {oid-root   OBJECT IDENTIFIER DEFAULT {1  3  22},
       reloids    SEQUENCE OF RELATIVE-OID -- relatif à l'identificateur
                                           -- d'objet racine --}
```

E.3 Identification des syntaxes abstraites

E.3.1 Souvent, on définit les protocoles en associant une sémantique à chacune des valeurs d'un seul type ASN.1, généralement un type choix. (Ce type ASN.1 est parfois appelé de façon informelle "le type de niveau supérieur de l'application".) Cet ensemble de valeurs abstraites est appelé de façon formelle la syntaxe abstraite de l'application. On peut identifier une syntaxe abstraite en lui donnant un nom de syntaxe abstraite du type ASN.1 identificateur d'objet.

E.3.2 L'affectation d'un identificateur d'objet à une syntaxe abstraite peut être effectuée au moyen de la classe d'objets informationnels prédéfinie **ABSTRACT-SYNTAX** définie dans la Rec. UIT-T X.681 | ISO/CEI 8824-2, ce qui sert par ailleurs à identifier clairement le type de niveau supérieur de l'application.

E.3.3 Ce qui suit est un exemple de texte pouvant apparaître dans la spécification d'une application:

EXEMPLE

```
Application-ASN1 DEFINITIONS ::=
      BEGIN
      EXPORTS Application-PDU;

      Application-PDU ::= CHOICE {
                  connect-pdu    ..... ,
                  data-pdu    CHOICE {
                        ..... ,
                        .....
                  },
                  .....
            }
            ......
      END

Abstract-Syntax-Module DEFINITIONS ::=
      BEGIN
      IMPORTS Application-PDU FROM Application-ASN1;

--  Cette application définit la syntaxe abstraite suivante:

      Abstract-Syntax ABSTRACT-SYNTAX ::=
                  { Application-PDU  IDENTIFIED BY
                        application-abstract-syntax-object-id }
```

```
        application-abstract-syntax-object-id OBJECT IDENTIFIER ::=
                    {joint-iso-itu-t asn1(1) examples(123)
                          application-abstract-syntax(3) }
--  Le descripteur d'objet correspondant est:

        application-abstract-syntax-descriptor ObjectDescriptor ::=
                          "Example Application Abstract Syntax"

--  Les valeurs d'identificateur d'objet et de descripteur d'objet ASN.1:
                    --  identificateur d'objet de règles de codage
                    --  descripteur d'objet de règles de codage
--  affectées aux règles de codage dans les Rec. UIT-T X.690 | ISO/CEI 8825-1
-- et Rec. UIT-T X.691 | ISO/CEI 8825-2 peuvent être utilisées comme
-- identificateur de la syntaxe de
-- transfert conjointement avec cette syntaxe abstraite.

END
```

E.3.4 Pour garantir l'interfonctionnement, la norme peut de surcroît identifier une syntaxe de transfert obligatoire (généralement l'une de celles qui sont définies dans les règles de codage de la Rec. UIT-T X.690 | ISO/CEI 8825-1 ou de la Rec. UIT-T X.691 | ISO/CEI 8825-2 ou de la Rec. UIT-T X.692 | ISO/CEI 8825-3).

E.4 Sous-types

E.4.1 Utiliser des sous-types pour restreindre un type existant à des valeurs particulières.

EXEMPLE

```
AtomicNumber    ::=   INTEGER (1..104)

TouchToneString ::=  IA5String
                     (FROM ("0123456789" | "*" | "#")) (SIZE (1..63))

ParameterList   ::=  SET SIZE (1..63) OF Parameter

SmallPrime   ::=  INTEGER (2|3|5|7|11|13|17|19|23|29)
```

E.4.2 Utiliser une contrainte de sous-type extensible pour modéliser un type **INTEGER** dont l'ensemble de valeurs autorisées est petit et bien défini, mais dont il est prévu qu'il puisse croître.

EXEMPLE

```
SmallPrime ::= INTEGER (2 | 3, ...)
-- première version de SmallPrime PetitNombrePremier
```

en prévision de:

```
SmallPrime ::= INTEGER (2 | 3, ..., 5 | 7 | 11)
-- deuxième version de SmallPrime PetitNombrePremier
```

et par la suite:

```
SmallPrime ::= INTEGER (2 | 3, ..., 5 | 7 | 11 | 13 | 17 | 19)
-- troisième version de SmallPrime PetitNombrePremier
```

NOTE – Certaines règles de codage (par exemple les règles PER) fournissent, pour certains types, un codage hautement optimisé pour les valeurs de racine d'extension de contrainte de sous-type (c'est-à-dire pour les valeurs apparaissant avant la notation "**...**") et un codage moins optimisé des valeurs d'ajout d'extension de contrainte de sous-type (c'est-à-dire pour les valeurs apparaissant après la notation "**...**"), tandis que les contraintes de sous-type n'ont aucun effet sur le codage effectué par d'autres règles (par exemple les règles BER).

E.4.3 Lorsque deux types apparentés ou plus ont de nombreux points communs, envisager de définir explicitement leur parent commun comme un type, et d'en dériver chacun des types par sous-typage. Cette solution met en relief leurs relations et leurs points communs et permet (sans aucune obligation) de maintenir ces relations au fur et à mesure de l'évolution des types. Elle facilite donc l'adoption d'approches communes pour le traitement des valeurs de ces types.

EXEMPLE

```
Envelope    ::=  SET {
                        typeA  TypeA,
                        typeB  TypeB   OPTIONAL,
                        typeC  TypeC   OPTIONAL}
                --  le parent commun

ABEnvelope  ::=  Envelope (WITH  COMPONENTS
                        {... ,
                              typeB   PRESENT, typeC   ABSENT})
                --  le typeB devant toujours apparaître mais pas le typeC

ACEnvelope  ::=  Envelope (WITH  COMPONENTS
                        {... ,
                              typeB   ABSENT, typeC   PRESENT})
                --  le typeC devant toujours apparaître mais pas le typeB
```

Ces dernières définitions peuvent aussi être exprimées de la manière suivante:

```
ABEnvelope  ::= Envelope (WITH COMPONENTS {typeA, typeB})

ACEnvelope  ::= Envelope (WITH COMPONENTS {typeA, typeC})
```

Le choix entre les différentes formes dépend de facteurs comme le nombre de composants du type parent et, parmi ceux-ci, le nombre de ceux qui sont optionnels, l'ampleur des différences entre les types concernés et la stratégie d'évolution probable.

E.4.4 Recourir au sous-typage pour définir partiellement une valeur, lorsqu'une unité de données protocolaire par exemple doit être soumise à un test de conformité ne portant que sur certains de ses composants.

EXEMPLE

Soit:

```
PDU  ::=  SET
        {alpha          INTEGER,
         beta           IA5String  OPTIONAL,
         gamma          SEQUENCE OF Parameter,
         delta          BOOLEAN}
```

alors, pour établir un test exigeant que le booléen soit faux et l'entier négatif, écrire:

```
TestPDU   ::=  PDU (WITH COMPONENTS
           {... ,
            delta (FALSE),
            alpha (MIN..<0)})
```

et si, de plus, la chaîne **IA5String**, **beta**, doit être présente et longue de 5 ou 12 caractères, écrire:

```
FurtherTestPDU ::= TestPDU (WITH COMPONENTS {... , beta (SIZE (5|12)) PRESENT
} )
```

E.4.5 Si un type de données d'usage général a été défini comme un type **SEQUENCE OF**, recourir au sous-typage pour définir un sous-type restreint de ce type général:

EXEMPLE

```
Text-block ::=  SEQUENCE OF VisibleString

Address   ::=  Text-block (SIZE (1..6)) (WITH COMPONENT (SIZE (1..32)))
```

E.4.6 Si un type de données d'usage général a été défini comme un type **CHOICE**, recourir au sous-typage pour définir un sous-type restreint de ce type général:

EXEMPLE

```
Z ::= CHOICE {
            a           A,
            b           B,
            c           C,
            d           D,
            e           E
      }
```

```
V ::= Z (WITH COMPONENTS { ..., a  ABSENT,  b  ABSENT })   --  "a" et "b"
                              -- doivent être absents, et
                              -- soit "c", soit "d", soit "e" peut être
                              -- présent dans la valeur

W ::= Z (WITH COMPONENTS { ..., a  PRESENT })         --  seul "a" doit être
                                    -- présent (voir § 47.8.9.2)

X ::= Z (WITH COMPONENTS { a  PRESENT })        --  seul "a" doit être
                                 -- présent (voir § 47.8.9.2)

Y ::= Z (WITH COMPONENTS { a  ABSENT,  b,  c })         --  "a", "d" et "e"
                                 -- doivent être absents, et
                                 -- soit "b", soit "c" peut être présent
                                 -- dans la valeur
```

NOTE – Les types **W** et **X** sont sémantiquement identiques.

E.4.7 Utiliser des sous-typages contenus pour former de nouveaux sous-types à partir de sous-types existants:

EXEMPLE

```
Months      ::= ENUMERATED {
                        january          (1),
                        february         (2),
                        march            (3),
                        april            (4),
                        may              (5),
                        june             (6),
                        july             (7),
                        august           (8),
                        september        (9),
                        october          (10),
                        november         (11),
                        december         (12) }

First-quarter ::= Months ( january | february  | march )

Second-quarter ::= Months ( april | may | june )

Third-quarter ::= Months ( july | august | september )

Fourth-quarter ::= Months ( october | november | december )

First-half ::= Months ( First-quarter | Second-quarter )

Second-half ::= Months ( Third-quarter | Fourth-quarter )
```

Annexe F

Exposé didactique sur les chaînes de caractères ASN.1

(Cette annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

F.1 Prise en charge des chaînes de caractères en notation ASN.1

F.1.1 La notation ASN.1 prend en charge quatre groupes de chaînes de caractères, à savoir:

a) les types de chaînes de caractères basés sur le *Registre international des jeux de caractères codés à utiliser avec une séquence d'échappement de l'ISO* (c'est-à-dire basés sur la structure de l'ISO/CEI 646) et le Registre international associé des jeux de caractères codés, constitués par les types **VisibleString**, **IA5String**, **TeletexString**, **VideotexString**, **GraphicString** et **GeneralString**;

b) les types de chaînes de caractères basés sur l'ISO/CEI 10646-1, obtenus par restriction des types **UniversalString**, **UTF8String** ou **BMPString** à des sous-ensembles définis dans l'ISO/CEI 10646-1 ou en utilisant des caractères nommés;

c) les types de chaînes de caractères fournissant une petite collection simple de caractères, spécifiés dans la présente Recommandation | Norme internationale, et destinés à des utilisations particulières; il s'agit des types **NumericString** et **PrintableString**;

d) le type **CHARACTER STRING** utilisé avec négociation du jeu de caractères à utiliser (ou l'annonce du jeu utilisé); cette possibilité permet à une application d'utiliser tous les jeux de caractères et codages auxquels des identificateurs d'objets **OBJECT IDENTIFIER** ont été affectés, y compris ceux du *Registre international des jeux de caractères codés à utiliser avec les séquences d'échappement de l'ISO*, de l'ISO/CEI 7350 et de l'ISO/CEI 10646-1, et les jeux de caractères et codages privés (les profils peuvent imposer des spécifications ou des restrictions aux jeux de caractères – les syntaxes abstraites de caractères – à utiliser).

F.2 Les types UniversalString, UTF8String et BMPString

F.2.1 Les types **UniversalString** et **UTF8String** contiennent tous les caractères définis dans l'ISO/CEI 10646-1. Ce jeu est généralement trop grand pour imposer en pratique une contrainte de conformité, et il doit normalement être restreint à des sous-ensembles combinant les jeux de caractères normalisés définis dans l'Annexe A de l'ISO/CEI 10646-1.

F.2.2 Le type **BMPString** contient tous les caractères de la grille ISO/CEI 10646-1. Ce jeu est généralement restreint à une combinaison des jeux de caractères normalisés définis dans l'Annexe A de l'ISO/CEI 10646-1.

F.2.3 Il existe pour les jeux définis dans l'Annexe A de l'ISO/CEI 10646-1 des références de type définies dans le module prédéfini **ASN1-CHARACTER-MODULE** (voir le § 38). Le mécanisme de contrainte de sous-typage permet de définir de nouveaux sous-types de **UniversalString** par combinaison de sous-types existants.

F.2.4 Parmi les références de type définies dans le module **ASN1-CHARACTER-MODULE** et le nom de collection correspondant dans l'ISO/CEI 10646-1, on peut citer à titre d'exemple:

BasicLatin	BASIC LATIN (Latin de base)
Latin-1Supplement	LATIN-1 SUPPLEMENT (Supplément Latin-1)
LatinExtended-a	LATIN EXTENDED-A (Latin étendu-a)
LatinExtended-b	LATIN EXTENDED-B (Latin étendu-b)
IpaExtensions	IPA EXTENSIONS (Extensions IPA)
SpacingModifierLetters	SPACING MODIFIER LETTERS (Lettres modifiant les espacements)
CombiningDiacriticalMarks	COMBINING DIACRITICAL MARKS (Signes diacritiques de combinaison)

F.2.5 L'ISO/CEI 10646-1 spécifie trois "niveaux d'application", et impose dans toute utilisation de l'ISO/CEI 10646-1 de spécifier le niveau de l'application.

Le niveau d'implémentation a trait à l'étendue de la prise en charge des *caractères de combinaison* dans le répertoire de caractères, et par là, en termes ASN.1, il définit un sous-ensemble des types de chaînes de caractères à alphabet restreint **UniversalString** et **BMPString**.

Dans le niveau 1 d'implémentation, les caractères de combinaison sont interdits, et il existe normalement une correspondance biunivoque entre les caractères abstraits des chaînes de caractères ASN.1 et les caractères matériels imprimés ou affichés de la chaîne.

Dans le niveau 2 d'implémentation, il est possible d'utiliser certains caractères de combinaison (énumérés dans l'Annexe B de l'ISO/CEI 10646-1), mais d'autres sont interdits.

Dans le niveau 3 d'implémentation, il n'y a aucune restriction à l'utilisation des caractères de combinaison.

F.2.6 Un type **BMPString** ou **UniversalString** peut faire l'objet d'une limitation de manière à exclure toutes les fonctions de commande en utilisant comme suit la notation de sous-type:

```
VanillaBMPString ::= BMPString (FROM (ALL EXCEPT ({0,0,0,0}..{0,0,0,31} |
                                                  {0,0,0,128}..{0,0,0,159})))
```

ou, de manière équivalente:

```
C0 ::= BMPString (FROM ({0,0,0,0} .. {0,0,0,31}))  -- fonctions de
commande C0
C1 ::= BMPString (FROM ({0,0,0,128} .. {0,0,0,159}))  -- fonctions de
commande C1
VanillaBMPString ::= BMPString (FROM (ALL EXCEPT (C0 | C1)))
```

F.3 A propos des prescriptions de conformité à l'ISO/CEI 10646-1

L'utilisation d'un type (ou d'un sous-type de) **UniversalString**, **BMPString** ou **UTF8String** dans une définition de type ASN.1 impose d'étudier les prescriptions de conformité de l'ISO/CEI 10646-1.

Ces prescriptions de conformité imposent aux développeurs d'une certaine norme (disons la norme X) utilisant de tels types ASN.1 de produire une déclaration (dans la déclaration de conformité d'une implémentation de protocole) précisant le sous-ensemble de caractères de l'ISO/CEI 10646-1 qu'ils utilisent dans leur implémentation de la norme X, ainsi que le niveau d'application (prise en charge des caractères de combinaison) de cette instance.

L'utilisation d'un sous-type de **UniversalString**, de **UTF8String** ou de **BMPString** dans une spécification impose la prise en charge par l'implémentation de tous les caractères de l'ISO/CEI 10646-1 inclus dans ce sous-type ASN.1, et donc que ces caractères (au moins) soient présents dans le sous-ensemble adopté pour l'implémentation de protocole. De même, il est nécessaire que le niveau de mise en œuvre déclaré soit pris en charge par tous les sous-types ASN.1 ainsi définis.

NOTE – En l'absence de paramètres dans la syntaxe abstraite et de spécifications d'exceptions, une spécification ASN.1 détermine à la fois le jeu (maximal) de caractères pouvant être utilisé en émission et le jeu (minimal) de caractères qui doit pouvoir être traité en réception. L'adoption du jeu ISO/CEI 10646-1 signifie qu'aucun caractère en dehors de ce jeu ne doit être transmis et que tous les caractères de ce jeu sont pris en charge à la réception. Le jeu adopté doit donc correspondre très précisément à l'ensemble de tous les caractères autorisés par la spécification ASN.1. Le cas où intervient un paramètre de syntaxe abstraite est discuté ci-dessous.

F.4 Recommandations aux utilisateurs ASN.1 à propos de la conformité à l'ISO/CEI 10646-1

Les utilisateurs ASN.1 doivent indiquer clairement le jeu de caractères ISO/CEI 10646-1 (et le niveau d'implémentation correspondant) qui devra être adopté par les instances de protocole pour que celles-ci puissent être conformes aux prescriptions de leur norme.

Une bonne façon de le faire est de définir un sous-type ASN.1 du type de **UniversalString**, **UTF8String** ou **BMPString** contenant tous les caractères nécessaires à la norme, et d'en restreindre s'il y a lieu le niveau au niveau **Level1** ou **Level2**. On pourra par exemple désigner ce type par **ISO-10646-String**.

EXEMPLE

```
ISO-10646-String ::= BMPString
      (FROM (Level2 INTERSECTION (BasicLatin UNION HebrewExtended UNION Hiragana)))
      --  Il s'agit du type qui définit le jeu minimal de caractères dans la
      --  collection adoptée pour les instances de cette norme.
      --  Le niveau d'application requis est le niveau 2 au moins.
```

Dans un environnement OSI, la déclaration de conformité d'une implémentation de protocole OSI contiendrait alors une simple déclaration indiquant que le sous-ensemble de caractères ISO/CEI 10646-1 adopté est le jeu (et le niveau) défini par le type **ISO-10646-String**, et ce type (ou un sous-type de) **ISO-10646-String** sera utilisé partout dans la norme où apparaîtront des chaînes de caractères ISO/CEI 10646-1.

EXEMPLE DE DECLARATION DE CONFORMITE

Le sous-jeu de caractères ISO/CEI 10646-1 adopté est le sous-ensemble limité constitué de tous les caractères du type ASN.1 **ISO-10646-String** défini dans le module <mettez ici le nom de votre module>, avec un niveau d'implémentation de 2.

EXEMPLE D'UTILISATION DANS UN PROTOCOLE

```
Message ::=  SEQUENCE {
      first-field ISO-10646-String,--  tous les caractères du sous-jeu peuvent
                                   -- être utilisés
      second-field     ISO-10646-String
            (FROM (latinSmallLetterA .. latinSmallLetterZ)),   --  minuscules
                                                          -- latines seulement
      third-field ISO-10646-String
            (FROM (digitZero .. digitNine)) --  chiffres seulement

}
```

F.5 Sous-jeux adoptés comme paramètres de la syntaxe abstraite

L'ISO/CEI 10646-1 impose de définir *explicitement* le sous-jeu de caractères et le niveau d'implémentation adoptés. Lorsqu'un utilisateur ASN.1 ne souhaite pas imposer de contrainte au jeu de caractères ISO/CEI 10646-1 dans une partie quelconque de la norme qu'il définit, il peut l'exprimer en définissant **ISO-10646-String** (par exemple) comme un sous-type de **UniversalString**, **BMPString** ou **UTF8String** avec une contrainte de sous-typage constituée de (ou comportant) un **ImplementorsSubset** qui est laissé comme paramètre de la syntaxe abstraite.

Les utilisateurs de la notation ASN.1 doivent savoir que dans ce cas, un expéditeur conforme peut transmettre à un destinataire conforme des caractères qui ne pourront pas être traités par le destinataire parce qu'ils n'appartiennent pas au sous-jeu ou au niveau d'application (dépendants de l'implémentation) adopté par le destinataire; il est recommandé dans ce cas d'inclure dans la définition du type **ISO-10646-String** une spécification de traitement d'exception.

EXEMPLE

```
ISO-10646-String {UniversalString : ImplementorsSubset, ImplementationLevel} ::=
            UniversalString (FROM((ImplementorsSubset UNION BasicLatin)
                        INTERSECTION ImplementationLevel) !characterSetProblem)
-- Le sous-jeu pris dans l'ISO/CEI 10646-1 comprendra le jeu "BasicLatin",
-- mais pourra également inclure tout caractère spécifié dans "ImplementorsSubset",
-- qui est un paramètre de la syntaxe abstraite. "ImplementationLevel", qui est
-- aussi un paramètre de la syntaxe abstraite, définit le niveau de
-- l'implémentation. Un destinataire conforme devra être préparé à recevoir des
-- caractères n'appartenant pas au sous-jeu de caractères et au niveau de son
-- implémentation. Dans un tel cas, le système invoquera le
-- traitement d'exception spécifié au paragraphe <ajouter ici votre numéro de
-- paragraphe> pour "characterSetProblem". A noter qu'un tel traitement ne sera
-- jamais invoqué par un destinataire conforme si les caractères utilisés dans la
-- communication sont limités au jeu "BasicLatin".

My-Level2-String ::= ISO-10646-String { { HebrewExtended UNION Hiragana }, Level2 }
```

F.6 Le type chaîne de caractères CHARACTER STRING

F.6.1 Le type **CHARACTER STRING** offre une totale souplesse de choix du jeu de caractères et de la méthode de codage.

NOTE – Lorsqu'une connexion simple assure un transfert de données de bout en bout (sans relais) et que les protocoles OSI sont utilisés, la négociation des jeux de caractères à utiliser et de leur codage peut s'effectuer dans le cadre de la définition des contextes de présentation OSI pour les syntaxes abstraites de caractères. Dans les autres cas, la syntaxe abstraite et la syntaxe de transfert de caractères (répertoire de caractères et codages) sont annoncés par une paire de valeurs d'identificateur d'objet.

F.6.2 D'un point de vue formel, une syntaxe abstraite de caractères est une syntaxe abstraite ordinaire avec des restrictions apportées à ses valeurs possibles (ce sont toutes des chaînes de caractères formées à partir d'une certaine collection de caractères). L'affectation des valeurs d'identificateur d'objet pour les sysntaxes abstraite et de transfert de caractères s'effectue donc de manière normale.

F.6.3 Le codage du type **CHARACTER STRING** permet d'annoncer les syntaxes abstraite et de transfert du répertoire de caractères utilisé (autrement dit le jeu de caractère et son codage). Dans les environnements OSI, la négociation de ces deux syntaxes est possible.

F.6.4 Des syntaxes abstraites de caractères (et les syntaxes de transfert correspondantes) sont définies dans plusieurs Recommandations UIT-T et Normes internationales; des syntaxes abstraites (ou des syntaxes de transfert) de caractères supplémentaires peuvent également être définies par tout organisme habilité à affecter des identificateurs d'objets.

F.6.5 Dans l'ISO/CEI 10646-1, une syntaxe abstraite de caractères est définie (et des identificateurs d'objets affectés) pour la collection complète de caractères, pour chacune des collections de caractères définies comme sous-ensemble de la collection complète (BASIC LATIN, BASIC SYMBOLS, etc.) et pour chaque combinaison possible des collections de caractères définies. Deux syntaxes de transfert de caractères sont aussi définies pour identifier les différentes options (en particulier les représentations sur 16 bits et sur 32 bits) dans l'ISO/CEI 10646-1.

Annexe G

Exposé didactique sur le modèle ASN.1 d'extension de type

(Cette annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

G.1 Aperçu général

G.1.1 Un type ASN.1 peut évoluer dans le temps à partir d'un type **racine d'extension** par le biais d'une série d'extensions appelée **ajouts d'extension**.

G.1.2 Un type ASN.1 disponible pour une implémentation donnée peut être le type racine d'extension ou peut être ce type complété par un ou plusieurs ajouts d'extension. Tout type ASN.1 qui contient un ajout d'extension contient également la totalité des ajouts d'extension définis précédemment.

G.1.3 Les définitions de type ASN.1 de ces séries sont dites "**apparentées par extension**" (voir § 3.6.32 pour une définition plus précise de ce terme) et des règles de codage sont nécessaires afin de coder des types apparentés par extension de manière telle que si deux systèmes utilisent deux types qui sont apparentés par extension, une transmission entre les deux systèmes réussira à transmettre le contenu informationnel des parties des types apparentés par extension qui sont communes aux deux systèmes. Il est également nécessaire que les parties qui ne sont pas communes aux deux systèmes puissent être délimitées et retransmises (éventuellement à un tiers) lors d'une transmission ultérieure, à condition que la syntaxe de transfert utilisée soit la même.

NOTE – L'émetteur peut utiliser un type antérieur ou postérieur dans la série d'ajouts d'extension.

G.1.4 La série des types obtenus par des ajouts successifs à un type racine est appelée **série d'extensions.** De tels types (y compris le type racine d'extension) doivent être repérés par un indicateur syntaxique, afin que les règles de codage puissent prendre les dispositions idoines pour la transmission de types apparentés par extension (qui peuvent nécessiter plus de bits sur la ligne). Cet indicateur constitué de points de suspension (`...`) est appelé **marqueur d'extension**.

EXEMPLE

```
Type racine          première            deuxième            troisième
d'extension          extension           extension           extension

A ::= SEQUENCE {     A ::= SEQUENCE {    A ::= SEQUENCE {    A ::= SEQUENCE {
      a INTEGER,       a INTEGER,          a INTEGER,          a INTEGER,
        ...              ...,                ...,                ...,
      }                  b BOOLEAN,          b BOOLEAN,          b BOOLEAN,
                         c INTEGER           c INTEGER,          c INTEGER,
                       }                     d SEQUENCE {        d SEQUENCE {
                                               e INTEGER,          e INTEGER,
                                                 ...,                ...,
                                                 ...,              g BOOLEAN OPTIONAL,
                                               f IA5String         h BMPString,
                                             }                       ...,
                                           }                       f IA5String
                                                                 }
                                                               }
```

G.1.5 Pour les types séquence, ensemble et choix, tous les ajouts d'extension sont insérés entre des paires de marqueurs d'extension. Un marqueur d'extension unique est autorisé si (dans le type racine d'extension) il apparaît comme dernier élément dans le type, auquel cas on fait l'hypothèse qu'un marqueur d'extension correspondant existe immédiatement avant l'accolade fermante du type; touts les ajouts d'extension sont alors insérés à la fin du type.

G.1.6 Un type possédant un marqueur d'extension peut être imbriqué au sein d'un type qui n'en possède pas, être imbriqué au sein d'un type dans une racine d'extension ou être imbriqué dans un type d'ajout d'extension. Les séries d'extensions sont traitées dans de tels cas d'une manière indépendante et le type imbriqué avec le marqueur d'extension n'a aucun effet sur le type au sein duquel il est imbriqué. Un seul **point d'insertion d'extension** peut exister dans toute expression spécifique; il est situé à la fin du type si un seul marqueur d'extension est utilisé, ou immédiatement avant le deuxième marqueur d'extension si une paire de marqueurs d'extension est utilisée.

G.1.7 Un nouvel ajout d'extension dans la série d'extensions est défini sous la forme d'un seul **groupe d'ajouts d'extension** (un ou plusieurs types imbriqués dans une structure "`[[`" "`]]`") ou sous la forme d'un type unique ajouté au niveau du point d'insertion d'extension. Dans l'exemple qui suit, la première extension définit un groupe d'ajouts d'extension pour lequel les éléments `b` et `c` doivent être tous deux présents ou absents dans une valeur de type `A`. La deuxième extension définit un type composant unique `d` qui peut être absent dans une valeur de type `A`. La troisième

extension définit un groupe d'ajouts d'extension dans lequel l'élément **h** doit être présent dans une valeur de type **A** chaque fois que le groupe d'ajouts d'extension figure dans une valeur.

EXEMPLE

```
Type racine          première            deuxième            troisième
d'extension          extension           extension           extension

A ::= SEQUENCE {     A ::= SEQUENCE {    A ::= SEQUENCE {    A ::= SEQUENCE {
      a INTEGER,       a INTEGER,          a INTEGER,          a INTEGER,
        ...              ...,                ...,                ...,
      }                [[                  [[                  [[
                         b BOOLEAN,          b BOOLEAN,          b BOOLEAN,
                         c INTEGER           c INTEGER           c INTEGER
                       ]]                  ]],                 ]],
                     }                     d SEQUENCE {        d SEQUENCE {
                                             e INTEGER,          e INTEGER,
                                               ...,                ...,
                                               ...,                [[
                                             f IA5String         g BOOLEAN OPTIONAL,
                                           }                     h BMPString
                                         }                       ]],
                                                                   ...,
                                                                 f IA5String
                                                               }
                                                             }
```

G.1.8 Il est également possible d'ajouter le numéro de version aux crochets de version, mais uniquement s'il est présent pour tous les crochets à l'intérieur d'un module et si toutes les extensions du module sont à l'intérieur de crochets de version. Il est recommandé d'utiliser des numéros de version. La possibilité d'omettre les numéros et les crochets de version provient d'anciennes dispositions. (Les crochets de version et les numéros de version étaient interdits dans les premières versions de la présente Recommandation | Norme internationale.) (Voir aussi § G.3.)

G.1.9 Bien que la démarche normale consiste à insérer des ajouts d'extension au cours du temps, le modèle ASN.1 sous-jacent et les spécifications n'impliquent pas de facteur temps. Deux types sont apparentés par extension si l'un des deux peut être produit à partir de l'autre au moyen d'ajouts d'extension. Ceci signifie que le premier contient tous les composants présents dans le deuxième. Il est possible (mais peu probable) qu'il existe des types qui doivent évoluer dans le sens opposé. Il est même possible qu'un type *démarre* avec un grand nombre d'ajouts d'extension qui sont supprimés progressivement. Dans le cadre de la notation ASN.1 et de ses règles de codage, on s'intéresse uniquement au fait que deux spécifications de type sont apparentées par extension ou non. Si elles le sont, *toutes* les règles de codage ASN.1 assureront un interfonctionnement entre leurs utilisateurs.

G.1.10 Nous partons d'un type donné et décidons ensuite si nous souhaitons pouvoir assurer un interfonctionnement avec des implémentations de versions antérieures lorsque nous devons procéder à des extensions. Si c'est le cas, le marqueur d'extension sera inclus *à cet instant*. Nous pourrons alors insérer ultérieurement des ajouts d'extension à ce type avec des procédures définies pour le traitement de valeurs étendues par les anciens systèmes. Il est toutefois important de noter que l'ajout d'un marqueur d'extension à un type qui n'en possédait pas précédemment (ou la suppression d'un marqueur d'extension) pourra faire obstacle à l'interfonctionnement.

NOTE – Lorsque la notation ECN est utilisée, il est possible d'ajouter des extensions dans la version 2 en des endroits où il n'y avait pas de marqueur d'extension dans la version 1, tout en conservant l'interfonctionnement entre les versions 1 et 2.

G.1.11 Le Tableau G.1 indique les types ASN.1 qui peuvent servir de type de racine d'extension pour une série d'extensions ASN.1, ainsi que la nature de l'ajout d'extension unique qui est autorisé pour ce type (les ajouts d'extension multiples sont effectués successivement, ou ensemble dans le cadre d'un groupe d'extensions).

Tableau G.1 – Ajouts d'extension

Type de racine d'extension	Nature de l'ajout d'extension
ENUMERATED	Ajout d'une seule énumération supplémentaire à la fin des énumérations "AdditionalEnumeration" avec une valeur d'énumération supérieure à celle de toute énumération déjà ajoutée.
SEQUENCE et **SET**	Ajout d'un seul type ou d'un seul groupe d'ajouts d'extension à la fin de la liste "ExtensionAdditionList". Les types de composants "ComponentType" qui constituent les ajouts d'extension (n'appartenant pas à un groupe d'ajouts d'extension) ne sont pas obligatoirement marqués comme **OPTIONAL** ou **DEFAULT** bien que ce soit souvent le cas.
CHOICE	Ajout d'un seul type "NamedType" à la fin de la liste "ExtensionAdditionAlternativesList".
Notation de contrainte	Ajout d'un seul élément "AdditionalElementSetSpec" à la notation "ElementSetSpecs".

G.2 Signification des numéros de version

G.2.1 Les numéros de version ne sont pas utilisés dans les codages BER et PER. Leur éventuelle utilisation dans les codages ECN est déterminée par la spécification de la notation ECN.

G.2.2 Les numéros de version sont très utiles lorsqu'ils se rapportent au moyen de décodage d'une unité PDU complète, et non à un type individuel. Lorsqu'un type est utilisé comme composant de plusieurs protocoles et contribue ainsi à différentes unités PDU complètes, un ajout à ce type nécessitera normalement une incrémentation du numéro de version pour toutes les unités PDU auxquelles il contribue.

G.2.3 Lorsque les numéros de version servent à assurer un interfonctionnement entre systèmes déployés, ils devraient être utilisés pour des groupes d'ajouts d'extension de telle sorte que les systèmes déployés aient connaissance de la syntaxe et de la sémantique de tous les groupes d'ajouts d'extension avec un numéro de version donné (peu importe l'endroit où ils figurent dans le protocole) et de tous les groupes d'ajouts d'extension avec un numéro de version antérieur. Les spécificateurs ECN supposeront généralement que les numéros de version sont attribués (à toutes les parties de types auxquelles la notation ECN est appliquée) conformément à ce principe.

G.3 Prescriptions concernant les règles de codage

G.3.1 Une syntaxe abstraite peut être définie par les valeurs d'un type ASN.1 unique qui est un type extensible. Elle contient alors toutes les valeurs qui peuvent être obtenues par insertion ou suppression d'ajouts d'extension. Une telle syntaxe abstraite est appelée une syntaxe abstraite apparentée par extension.

G.3.2 Un ensemble de règles de codage bien formées pour une syntaxe abstraite apparentée par extension satisfait aux prescriptions supplémentaires formulées dans les § G.3.3 à G.3.5.

NOTE – Toutes les règles de codage ASN.1 satisfont à ces prescriptions.

G.3.3 La définition des procédures de transformation d'une valeur abstraite en un codage à des fins de transfert et des procédures de transformation d'un codage reçu en une valeur abstraite doit tenir compte du fait que l'émetteur et le récepteur utilisent des syntaxes abstraites qui ne sont pas identiques, mais qui sont apparentés par extension.

G.3.4 Les règles de codage garantiront que lorsque l'émetteur utilise une spécification de type qui est plus ancienne dans la série d'extensions que la spécification utilisée par le récepteur, les valeurs de l'émetteur seront transférées de telle manière que le récepteur puisse déterminer que des ajouts d'extension ne sont pas présents.

G.3.5 Les règles de codage garantiront que lorsque l'émetteur utilise une spécification de type qui est plus récente dans la série d'extensions que la spécification utilisée par le récepteur, un transfert de valeurs de ce type à destination du récepteur sera possible.

G.4 Combinaison de contraintes (éventuellement extensibles)

G.4.1 Modèle

G.4.1.1 Le modèle ASN.1 de base pour l'application de contraintes est simple: un type est un ensemble de valeurs abstraites et une contrainte appliquée à ce type sélectionne un sous-ensemble de ces valeurs abstraites. Si le type non contraint n'est pas extensible, le type résultant est défini comme étant extensible si et seulement si la contrainte appliquée est définie comme étant extensible.

G.4.1.2 Même dans ce cas simple, il faut apporter la précisison suivante: un type peut être extensible d'un point de vue formel, même s'il ne peut jamais y avoir d'ajouts d'extension. Considérons par exemple:

```
A ::= INTEGER (MIN .. MAX, ... , 1..10)
```

Comme dans nombreux exemples donnés dans la présente annexe, c'est une production que personne n'écrira jamais, mais pour laquelle les fabricants d'outils doivent écrire un code car la norme ASN.1 a été voulue simple et générale et cet exemple correspond à une déclaration ASN.1 licite. Dans cet exemple, A est d'un point de vue formel un entier **INTEGER** extensible, l'intervalle complet de valeurs entières étant indiqué dans la racine.

G.4.1.3 Les complexités proviennent des trois principales situations suivantes:

- L'application d'une contrainte à un type auquel une contrainte extensible a déjà été appliquée (application de contraintes en série – voir § G.4.2).
- La combinaison de contraintes extensibles à l'aide de **UNION**, **INTERSECTION** et **EXCEPT** (opérations arithmétiques sur les ensembles – voir § G.4.3).
- L'utilisation d'une référence de type (un sous-type contenu) dans les opérations arithmétiques sur les ensembles relatives à une contrainte, lorsque la référence de type ne fait plus référence à un type extensible (éventuellement avec des ajouts d'extension effectifs – voir § G.4.4).

G.4.2 Application de contraintes en série

G.4.2.1 L'application de contraintes en série se produit lorsqu'un type est contraint (dans une affectation à une référence de type) et que la référence de type est ensuite utilisée avec une autre contrainte appliquée au type.

G.4.2.2 Elle peut aussi se produire, mais c'est moins courant, lorsque plusieurs contraintes sont appliquées directement à un type et ce, en série. C'est cette forme qui est utilisée pour de nombreux exemples de la présente annexe (dans un souci de simplicité de la présentation), mais le cas où une référence de type relie les deux contraintes (ou davantage) est la forme sous laquelle l'application en série se produit normalement dans les spécifications réelles.

G.4.2.3 Les deux points suivants sont essentiels en ce qui concerne l'application de contraintes en série:

– si un type contraint est extensible (et éventuellement étendu), le fanion "extensible" et tous les ajouts d'extension sont ignorés si une autre contrainte est ensuite appliquée en série. L'extensibilité d'un type contraint (et des éventuels ajouts d'extension) dépend uniquement de la dernière contrainte qui est appliquée, qui peut faire référence uniquement aux valeurs de la racine du type auquel une autre contrainte est appliquée (le type parent). Les valeurs incluses dans la racine ou dans les ajouts d'extension du type résultant peuvent uniquement être des valeurs qui se trouvent dans la racine du type parent;

– l'application de contraintes en série est (pour les cas complexes) différent d'une intersection d'ensembles, même en l'absence d'extensibilité. D'une part, l'environnement dans lequel **MIN** et **MAX** sont interprétés et, d'autre part, les valeurs abstraites auxquelles il peut être fait référence dans la deuxième contrainte sont très différents dans l'application en série et dans la situation où les deux contraintes sont spécifiées comme une intersection de valeurs provenant d'un parent commun.

NOTE – L'utilisation d'un intervalle tel que **20..28** dans une contrainte imposée à un type entier est légale si (et seulement si) les valeurs **20** et **28** figurent toutes deux dans le ([la] racine du) type parent, mais les valeurs auxquelles fait référence cette spécification d'intervalle sont uniquement celles qui se trouvent dans le ([la] racine du) parent. Ainsi, si une contrainte visant à exclure les valeurs **24** et **25** a déjà été appliquée au parent, l'intervalle **20..28** fait référence uniquement aux valeurs **20** à **23** et **26** à **28**.

Voici quelques exemples:

```
A1 ::= INTEGER (1..32, ... , 33..128)
      -- A1 est extensible et contient les valeurs 1 à 128, les valeurs 1 à 32
      -- étant dans la racine et les valeurs 33 à 128 étant des ajouts d'extension

B1 ::= A1 (1..128)
      -- ou, de façon équivalente

B1 ::= INTEGER (1..32, ... , 33..128) (1..128)
      -- Ces déclarations sont illégales, car 128 n'est pas dans le parent,
      -- qui a perdu ses ajouts d'extension lorsqu'une nouvelle contrainte lui a
      -- été appliquée

B2 ::= A1 (1..16)
      -- Cette déclaration est légale. B2 n'est pas extensible et contient les
      -- valeurs 1 à 16.

A2 ::= INTEGER (1..32) (MIN .. 63)
      -- MIN vaut 1 et la valeur 63 est illégale

A3 ::= INTEGER ( (1..32) INTERSECTION (MIN..63) )
      -- Cette déclaration est légale. MIN vaut moins l'infini et A3 contient les
      -- valeurs 1 à 32.
```

G.4.3 Utilisation d'opérations arithmétiques sur les ensembles

G.4.3.1 Les résultats sont largement intuitifs et obéissent aux règles mathématiques normales concernant l'intersection, la réunion et la différence entre ensembles (**EXCEPT**). En particulier, l'intersection et la réunion sont commutatives, autrement dit:

```
( <some set 1 of values> INTERSECTION <some set 2 of values> )
```

est identique à

```
( <some set 2 of values> INTERSECTION <some set 1 of values> )
```

Il en est de même pour **UNION**.

G.4.3.2 La commutativité est Vraie, que les ensembles de valeurs soient extensibles ou non et que des ajouts d'extension soient présents ou non.

G.4.3.3 Des malentendus peuvent se produire si une intersection rend impossible pour toujours la présence de valeurs d'ajouts d'extension. Ceci est analogue au cas de **INTEGER (MIN..MAX, ...)**.

G.4.3.4 Par exemple:

```
A ::= INTEGER ((1..256, ... , B) INTERSECTION (1..256))
-- A contient toujours les (seules) valeurs 1..256, quelles que soient les
-- valeurs que B contient, mais est néanmoins extensible d'un point de vue
-- formel.
```

G.4.3.5 Il est important d'avoir à l'esprit que les parents perdent leur extensibilité et leurs ajouts d'extension lorsqu'une nouvelle contrainte leur est appliquée et que les sous-types contenus perdent leur extensibilité et leurs ajouts d'extension, mais que les ensembles de valeurs spécifiés directement dans des opérations artihmétiques sur les ensembles ne perdent ni leur extensibilité ni leurs ajouts d'extension.

G.4.3.6 Les règles d'extensibilité des ensembles de valeurs produits par les opérations arithmétiques sur les ensembles sont définies clairement aux § 46.3 et 46.4 et ne dépendent pas de la question de savoir si ces opérations rendent possibles ou impossibles les ajouts d'extension.

G.4.3.7 Les règles sont récapitulées ici dans un souci d'exhaustivité, **E** désignant un ensemble de valeurs avec le fanion "extensible" mis à 1 et **N** désignant un ensemble de valeurs non extensible d'un point de vue formel. Les valeurs se trouvant dans la racine de chaque ensemble sont désignées par **R** et les éventuels ajouts d'extension par **X**. Le contenu du résultat est donné dans chaque cas.

NOTE 1 – Pour les besoins de la présente annexe et dans un souci de simplicité de la présentation, si un ensemble de valeurs n'est pas extensible, nous décrivons toutes ses valeurs comme étant des valeurs de la racine.

NOTE 2 – Si la racine d'un ensemble résultant de valeurs utilisées pour l'application d'une contrainte en série est vide, cela signifie que la spécification est illégale.

NOTE 3 – Dans un souci de concision, on utilise dans ce qui suit l'expression "Extensions" au lieu de l'expression "Ajouts d'extension", qui est plus correcte.

G.4.3.8 Les règles sont les suivantes:

```
N1 INTERSECTION N2 => N
        Root: R1 INTERSECTION R2
N1 INTERSECTION E2 => E
        Root: R1 INTERSECTION R2, Extensions: R1 INTERSECTION X2
E1 INTERSECTION E2 => E
        Root: R1 INTERSECTION R2, Extensions: ((R1 UNION X1)
                                               INTERSECTION
                                               (R2 UNION X2))
                                               EXCEPT
                                               (R1 INTERSECTION R2)
N1 UNION N2 => N
        Root: R1 UNION R2
N1 UNION E2 => E
        Root: R1 UNION R2, Extensions: X2
E1 UNION E2 => E
        Root: R1 UNION R2, Extensions: (R1 UNION X1 UNION R2 UNION X2)
                                       EXCEPT
                                       (R1 UNION R2)
N1 EXCEPT N2 => N
        Root: R1 EXCEPT R2
N1 EXCEPT E2 => N
        Root: R1 EXCEPT R2
E1 EXCEPT N2 => E
        Root: R1 EXCEPT R2, Extensions: (X1 EXCEPT R2)
                                        EXCEPT
                                        (R1 EXCEPT R2)
E1 EXCEPT E2 => E
        Root: R1 EXCEPT R2, Extensions: (X1 EXCEPT (R2 UNION X2) )
                                        EXCEPT
                                        (R1 EXCEPT R2)
N1 ... N2 => E
        Root: R1, Extensions: R2 EXCEPT R1
E1 ... N2 => E
        Root: R1, Extensions: X1 UNION R2
                              EXCEPT
                              R1
```

```
N1 ... E2 => E
        Root: R1, Extensions: R2 UNION X2
                              EXCEPT
                              R1
E1 ... E2 => E
        Root: R1, Extensions: X1 UNION R2 UNION E2
                              EXCEPT
                              R1
```

NOTE – Si le résultat d'une opération arithmétique sur des ensembles de valeurs extensibles n'a pas d'ajouts d'extension effectifs ou ne peut jamais avoir d'ajouts d'extension effectifs (quels que soient les ajouts d'extension qui sont insérés dans les entrées extensibles), le résultat reste défini d'un point de vue formel comme étant extensible pour les résultats **E** ci-dessus.

G.4.4 Utilisation de la notation de sous-type contenu

Un sous-type contenu peut être extensible ou non, mais lorsqu'il est utilisé dans une opération arithmétique sur des ensembles, il est toujours considéré comme n'étant pas extensible et tous ses ajouts d'extension sont ignorés.

Annexe H

Récapitulatif de la notation ASN.1

(Cette annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

Les unités lexicales ci-dessous sont définies dans le § 11:

typereference
identifier
valuereference
modulereference
comment
empty
number
realnumber
bstring
hstring
cstring
xmlbstring
xmlhstring
xmlcstring
xmlasn1typename
"true"
"false"
"::="
"[["
"]]"
".."
"..."
"</"
"/>"
"{"
"}"
"<"
">"
","
"."
"("
")"
"["
"]"
"-"
":"
"="
""" (QUOTATION MARK)
"'" (APOSTROPHE)
" " (SPACE)
";"
"@"
"|"
"!"
"^"

ABSENT
ABSTRACT-SYNTAX
ALL
APPLICATION
AUTOMATIC
BEGIN
BIT
BMPString
BOOLEAN
BY
CHARACTER
CHOICE
CLASS
COMPONENT
COMPONENTS
CONSTRAINED
CONTAINING
DEFAULT
DEFINITIONS
EMBEDDED
ENCODED
END
ENUMERATED
EXCEPT
EXPLICIT
EXPORTS
EXTENSIBILITY
EXTERNAL
FALSE
FROM
GeneralizedTime
GeneralString
GraphicString
IA5String
IDENTIFIER
IMPLICIT
IMPLIED
IMPORTS
INCLUDES
INSTANCE
INTEGER
INTERSECTION
ISO646String
MAX
MIN
MINUS-INFINITY
NULL
NumericString
OBJECT
ObjectDescriptor
OCTET
OF
OPTIONAL
PATTERN
PDV
PLUS-INFINITY
PRESENT
PrintableString
PRIVATE
REAL
RELATIVE-OID
SEQUENCE
SET
SIZE
STRING

SYNTAX	TYPE-IDENTIFIER	UTCTime
T61String	UNION	UTF8String
TAGS	UNIQUE	VideotexString
TeletexString	UNIVERSAL	VisibleString
TRUE	UniversalString	WITH

Les productions suivantes sont utilisées dans la présente Recommandation | Norme internationale, avec les unités lexicales ci-dessus comme symboles terminaux:

```
ModuleDefinition ::= ModuleIdentifier
                     DEFINITIONS
                     TagDefault
                     ExtensionDefault
                     "::="
                     BEGIN
                     ModuleBody
                     END

ModuleIdentifier ::= modulereference
                         DefinitiveIdentifier

DefinitiveIdentifier ::= "{" DefinitiveObjIdComponentList "}"
    |                    empty

DefinitiveObjIdComponentList ::=
          DefinitiveObjIdComponent
    |     DefinitiveObjIdComponent DefinitiveObjIdComponentList

DefinitiveObjIdComponent ::=
          NameForm
    |     DefinitiveNumberForm
    |     DefinitiveNameAndNumberForm

DefinitiveNumberForm      ::= number

DefinitiveNameAndNumberForm ::= identifier "(" DefinitiveNumberForm ")"

TagDefault ::=
          EXPLICIT TAGS
    |     IMPLICIT TAGS
    |     AUTOMATIC TAGS
    |     empty

ExtensionDefault ::=
          EXTENSIBILITY IMPLIED | empty

ModuleBody ::=   Exports Imports AssignmentList
    |                empty

Exports ::=      EXPORTS SymbolsExported ";"
    |            EXPORTS ALL ";"
    |            empty

SymbolsExported ::= SymbolList
    |               empty

Imports ::=      IMPORTS SymbolsImported ";"
    |            empty

SymbolsImported ::= SymbolsFromModuleList
    |               empty

SymbolsFromModuleList ::=
                    SymbolsFromModule
    |               SymbolsFromModuleList  SymbolsFromModule

SymbolsFromModule ::=   SymbolList FROM GlobalModuleReference

GlobalModuleReference ::= modulereference AssignedIdentifier
```

```
AssignedIdentifier ::=  ObjectIdentifierValue
      |                 DefinedValue
      |                 empty

SymbolList   ::= Symbol | SymbolList "," Symbol

Symbol  ::= Reference | ParameterizedReference

Reference ::=
            typereference
      |     valuereference
      |     objectclassreference
      |     objectreference
      |     objectsetreference

AssignmentList ::=          Assignment  | AssignmentList Assignment

Assignment ::=
            TypeAssignment
      |     ValueAssignment
      |     XMLValueAssignment
      |     ValueSetTypeAssignment
      |     ObjectClassAssignment
      |     ObjectAssignment
      |     ObjectSetAssignment
      |     ParameterizedAssignment

DefinedType ::=
            ExternalTypeReference
      |     typereference
      |     ParameterizedType
      |     ParameterizedValueSetType

ExternalTypeReference ::=
                  modulereference
                  "."
                  typereference

NonParameterizedTypeName ::=
            ExternalTypeReference
      |     typereference
      |     xmlasn1typename

DefinedValue ::=
            ExternalValueReference
      |     valuereference
      |     ParameterizedValue

ExternalValueReference ::=
                  modulereference
                  "."
                  valuereference

AbsoluteReference ::=
                  "@" ModuleIdentifier
                  "."
                  ItemSpec

ItemSpec ::=
            typereference
      |     ItemId "." ComponentId

ItemId ::= ItemSpec

ComponentId ::=
            identifier | number | "*"

TypeAssignment ::=    typereference
                           "::="
```

```
                                Type

ValueAssignment ::=   valuereference
                                Type
                                ":::="
                                Value

XMLValueAssignment ::=
        valuereference
        ":::="
        XMLTypedValue

XMLTypedValue ::=
        "<" & NonParameterizedTypeName  ">"
        XMLValue
        "</" & NonParameterizedTypeName ">"
    |   "<" & NonParameterizedTypeName "/>"

ValueSetTypeAssignment ::=      typereference
                                Type
                                ":::="
                                ValueSet

ValueSet ::= "{" ElementSetSpecs "}"

Type ::= BuiltinType | ReferencedType | ConstrainedType

BuiltinType ::=
        BitStringType
    |   BooleanType
    |   CharacterStringType
    |   ChoiceType
    |   EmbeddedPDVType
    |   EnumeratedType
    |   ExternalType
    |   InstanceOfType
    |   IntegerType
    |   NullType
    |   ObjectClassFieldType
    |   ObjectIdentifierType
    |   OctetStringType
    |   RealType
    |   RelativeOIDType
    |   SequenceType
    |   SequenceOfType
    |   SetType
    |   SetOfType
    |   TaggedType

NamedType ::= identifier   Type

ReferencedType ::=
        DefinedType
    |   UsefulType
    |   SelectionType
    |   TypeFromObject
    |   ValueSetFromObjects

Value  ::=   BuiltinValue | ReferencedValue | ObjectClassFieldValue

XMLValue ::= XMLBuiltinValue | XMLObjectClassFieldValue

BuiltinValue ::=
        BitStringValue
    |   BooleanValue
    |   CharacterStringValue
    |   ChoiceValue
    |   EmbeddedPDVValue
```

```
        |   EnumeratedValue
        |   ExternalValue
        |   InstanceOfValue
        |   IntegerValue
        |   NullValue
        |   ObjectIdentifierValue
        |   OctetStringValue
        |   RealValue
        |   RelativeOIDValue
        |   SequenceValue
        |   SequenceOfValue
        |   SetValue
        |   SetOfValue
        |   TaggedValue

XMLBuiltinValue ::=
            XMLBitStringValue
        |   XMLBooleanValue
        |   XMLCharacterStringValue
        |   XMLChoiceValue
        |   XMLEmbeddedPDVValue
        |   XMLEnumeratedValue
        |   XMLExternalValue
        |   XMLInstanceOfValue
        |   XMLIntegerValue
        |   XMLNullValue
        |   XMLObjectIdentifierValue
        |   XMLOctetStringValue
        |   XMLRealValue
        |   XMLRelativeOIDValue
        |   XMLSequenceValue
        |   XMLSequenceOfValue
        |   XMLSetValue
        |   XMLSetOfValue
        |   XMLTaggedValue

ReferencedValue ::=
            DefinedValue
        |   ValueFromObject

NamedValue ::= identifier Value

XMLNamedValue ::=
            "<" & identifier  ">" XMLValue "</" & identifier ">"

BooleanType  ::=BOOLEAN

BooleanValue::=  TRUE | FALSE

XMLBooleanValue ::=
            "<" & "true" "/>"
        |   "<" & "false" "/>"

IntegerType ::=
            INTEGER
        |   INTEGER "{" NamedNumberList "}"

NamedNumberList ::=
            NamedNumber
        |   NamedNumberList "," NamedNumber

NamedNumber ::=
            identifier "(" SignedNumber ")"
        |   identifier "(" DefinedValue ")"

SignedNumber ::= number | "-" number

IntegerValue ::=   SignedNumber | identifier
```

```
XMLIntegerValue ::=
        SignedNumber
    |   "<" & identifier "/>"

EnumeratedType ::=
        ENUMERATED "{" Enumerations "}"

Enumerations ::= RootEnumeration
    |   RootEnumeration  ","  "..." ExceptionSpec
    |   RootEnumeration  ","  "..." ExceptionSpec  ","  AdditionalEnumeration

RootEnumeration ::= Enumeration

AdditionalEnumeration ::= Enumeration

Enumeration ::= EnumerationItem | EnumerationItem "," Enumeration

EnumerationItem ::= identifier | NamedNumber

EnumeratedValue  ::= identifier

XMLEnumeratedValue ::= "<" & identifier "/>"

RealType ::= REAL

RealValue ::=
        NumericRealValue | SpecialRealValue

NumericRealValue ::=
        realnumber
    |   "-" realnumber
    |   SequenceValue              -- Valeur du type séquence associé

SpecialRealValue ::=
        PLUS-INFINITY | MINUS-INFINITY

XMLRealValue ::=
        XMLNumericRealValue | XMLSpecialRealValue

XMLNumericRealValue ::=
        realnumber
    |   "-" realnumber

XMLSpecialRealValue ::=
        "<" & PLUS-INFINITY "/>" | "<" & MINUS-INFINITY "/>"

BitStringType   ::=     BIT STRING | BIT STRING "{" NamedBitList "}"

NamedBitList::=     NamedBit | NamedBitList "," NamedBit

NamedBit    ::=     identifier "(" number ")"
    |           identifier "(" DefinedValue ")"

BitStringValue  ::=  bstring | hstring | "{" IdentifierList "}" | "{"  "}" | CONTAINING Value

IdentifierList  ::=     identifier | IdentifierList "," identifier

XMLBitStringValue ::=
        XMLTypedValue
    |   xmlbstring
    |   XMLIdentifierList
    |   empty

XMLIdentifierList ::=
        "<" & identifier "/>"
    |   XMLIdentifierList "<" & identifier "/>"

OctetStringType  ::=  OCTET STRING

OctetStringValue ::=  bstring | hstring | CONTAINING Value

XMLOctetStringValue ::=
        XMLTypedValue
```

```
    |   xmlhstring

NullType    ::=  NULL

NullValue   ::=  NULL

XMLNullValue ::= empty

SequenceType ::=
        SEQUENCE "{" "}"
    |   SEQUENCE "{" ExtensionAndException OptionalExtensionMarker "}"
    |   SEQUENCE "{" ComponentTypeLists "}"

ExtensionAndException ::= "..." | "..." ExceptionSpec

OptionalExtensionMarker ::= "," "..." | empty

ComponentTypeLists ::=
        RootComponentTypeList
    |   RootComponentTypeList "," ExtensionAndException ExtensionAdditions
            OptionalExtensionMarker
    |   RootComponentTypeList "," ExtensionAndException ExtensionAdditions
            ExtensionEndMarker "," RootComponentTypeList
    |   ExtensionAndException ExtensionAdditions ExensionEndMarker ","
            RootComponentTypeList
    |   ExtensionAndException ExtensionAdditions OptionalExtensionMarker

RootComponentTypeList ::= ComponentTypeList

ExtensionEndMarker ::= "," "..."

ExtensionAdditions ::= "," ExtensionAdditionList | empty

ExtensionAdditionList ::= ExtensionAddition
    |   ExtensionAdditionList "," ExtensionAddition

ExtensionAddition ::= ComponentType | ExtensionAdditionGroup

ExtensionAdditionGroup ::= "[[" VersionNumber ComponentTypeList "]]"

VersionNumber ::= empty | number ":"

ComponentTypeList ::=   ComponentType
    |                   ComponentTypeList "," ComponentType

ComponentType ::=
        NamedType
    |   NamedType OPTIONAL
    |   NamedType DEFAULT Value
    |   COMPONENTS OF Type

SequenceValue   ::=  "{" ComponentValueList "}" | "{" "}"

ComponentValueList ::=
        NamedValue
    |   ComponentValueList "," NamedValue

XMLSequenceValue ::=
        XMLComponentValueList
    |   empty

XMLComponentValueList ::=
        XMLNamedValue
    |   XMLComponentValueList XMLNamedValue

SequenceOfType ::=  SEQUENCE OF Type | SEQUENCE OF NamedType

SequenceOfValue ::=  "{" ValueList "}" | "{" NamedValueList "}" | "{" "}"

ValueList     ::=     Value | ValueList "," Value
```

```
XMLSequenceOfValue ::=
        XMLValueList
    |   XMLDelimitedItemList
    |   XMLSpaceSeparatedList
    |   empty

XMLValueList ::=
        XMLValueOrEmpty
    |   XMLValueOrEmpty XMLValueList

XMLValueOrEmpty ::=
        XMLValue
    |   "<" & NonParameterizedTypeName "/>"

XMLSpaceSeparatedList ::=
        XMLValueOrEmpty
    |   XMLValueOrEmpty " " XMLSpaceSeparatedList

XMLDelimitedItemList ::=
        XMLDelimitedItem
    |   XMLDelimitedItem XMLDelimitedItemList

XMLDelimitedItem ::=
        "<" & NonParameterizedTypeName ">" XMLValue
            "</" & NonParameterizedTypeName ">"
    |   "<" & identifier ">" XMLValue "</" & identifier ">"

SetType ::=  SET "{" "}"
    |        SET "{" ExtensionAndException OptionalExtensionMarker "}"
    |        SET "{" ComponentTypeLists "}"

SetValue    ::=   "{" ComponentValueList "}" | "{" "}"

XMLSetValue ::= XMLComponentValueList   |   empty

SetOfType   ::=  SET OF Type | SET OF NamedType

SetOfValue  ::=  "{" ValueList "}" | "{" NamedValueList "}" | "{" "}"

XMLSetOfValue ::=
        XMLValueList
    |   XMLDelimitedItemList
    |   XMLSpaceSeparatedList
    |   empty

ChoiceType   ::= CHOICE "{" AlternativeTypeLists "}"

AlternativeTypeLists ::=
        RootAlternativeTypeList
    |   RootAlternativeTypeList ","
            ExtensionAndException ExtensionAdditionAlternatives OptionalExtensionMarker

RootAlternativeTypeList ::= AlternativeTypeList

ExtensionAdditionAlternatives ::= "," ExtensionAdditionAlternativesList | empty

ExtensionAdditionAlternativesList ::= ExtensionAdditionAlternative
    |       ExtensionAdditionAlternativesList "," ExtensionAdditionAlternative

ExtensionAdditionAlternative ::= ExtensionAdditionAlternativesGroup | NamedType

ExtensionAdditionAlternativesGroup ::= "[[" VersionNumber AlternativeTypeList "]]"

AlternativeTypeList ::=             NamedType
    |                               AlternativeTypeList "," NamedType

ChoiceValue ::=       identifier ":" Value

XMLChoiceValue ::= "<" & identifier ">" XMLValue "</" & identifier ">"

SelectionType   ::=         identifier "<" Type
```

```
TaggedType   ::=      Tag Type
     |                Tag IMPLICIT Type
     |                Tag EXPLICIT Type

Tag     ::=           "[" Class ClassNumber "]"

ClassNumber  ::= number | DefinedValue

Class   ::=   UNIVERSAL
     |        APPLICATION
     |        PRIVATE
     |        empty

TaggedValue  ::=  Value

XMLTaggedValue ::= XMLValue

EmbeddedPDVType ::=            EMBEDDED PDV

EmbeddedPDVValue ::=           SequenceValue

XMLEmbeddedPDVValue ::= XMLSequenceValue

ExternalType ::= EXTERNAL

ExternalValue ::= SequenceValue

XMLExternalValue ::= XMLSequenceValue

ObjectIdentifierType  ::=  OBJECT IDENTIFIER

ObjectIdentifierValue ::=  "{" ObjIdComponentsList "}"
     |                     "{" DefinedValue ObjIdComponentsList "}"

ObjIdComponentsList  ::=  ObjIdComponents
     |                    ObjIdComponents ObjIdComponentsList

ObjIdComponents  ::= NameForm
     |               NumberForm
     |               NameAndNumberForm
     |               DefinedValue

NameForm        ::=      identifier

NumberForm      ::=      number | DefinedValue

NameAndNumberForm ::=      identifier "(" NumberForm ")"

XMLObjectIdentifierValue ::=
          XMLObjIdComponentList

XMLObjIdComponentList ::=
          XMLObjIdComponent
     |    XMLObjIdComponent & "." & XMLObjIdComponentList

XMLObjIdComponent ::=
          NameForm
     |    XMLNumberForm
     |    XMLNameAndNumberForm

XMLNumberForm ::= number

XMLNameAndNumberForm ::=
          identifier & "(" & XMLNumberForm & ")"

RelativeOIDType ::=
               RELATIVE-OID

RelativeOIDValue ::=
               "{" RelativeOIDComponentsList "}"

RelativeOIDComponentsList ::=
          RelativeOIDComponents
```

```
    |   RelativeOIDComponents RelativeOIDComponentsList

RelativeOIDComponents ::=   NumberForm
    |                       NameAndNumberForm
    |                       DefinedValue

XMLRelativeOIDValue ::=
        XMLRelativeOIDComponentList

XMLRelativeOIDComponentList ::=
        XMLRelativeOIDComponent
    |   XMLRelativeOIDComponent & "." & XMLRelativeOIDComponentList

XMLRelativeOIDComponent ::=
        XMLNumberForm
    |   XMLNameAndNumberForm

CharacterStringType ::= RestrictedCharacterStringType | UnrestrictedCharacterStringType

RestrictedCharacterStringType ::=
        BMPString
    |   GeneralString
    |   GraphicString
    |   IA5String
    |   ISO646String
    |   NumericString
    |   PrintableString
    |   TeletexString
    |   T61String
    |   UniversalString
    |   UTF8String
    |   VideotexString
    |   VisibleString

RestrictedCharacterStringValue ::= cstring | CharacterStringList | Quadruple | Tuple

CharacterStringList  ::=  "{" CharSyms "}"

CharSyms  ::=  CharsDefn | CharSyms "," CharsDefn

CharsDefn  ::=  cstring | Quadruple | Tuple | DefinedValue

Quadruple    ::= "{"  Group  ","  Plane  ","  Row  ","  Cell "}"

Group        ::= number

Plane        ::= number

Row          ::= number

Cell         ::= number

Tuple ::= "{" TableColumn "," TableRow "}"

TableColumn ::= number

TableRow ::= number

XMLRestrictedCharacterStringValue ::= xmlcstring

UnrestrictedCharacterStringType ::= CHARACTER STRING

CharacterStringValue ::= RestrictedCharacterStringValue | UnrestrictedCharacterStringValue

XMLCharacterStringValue ::=
        XMLRestrictedCharacterStringValue
    |   XMLUnrestrictedCharacterStringValue

UnrestrictedCharacterStringValue ::= SequenceValue

XMLUnrestrictedCharacterStringValue ::= XMLSequenceValue
```

```
UsefulType   ::= typereference
```

Les types de chaînes de caractères suivants sont définis au § 37.1:

```
NumericString     VisibleString
PrintableString   ISO646String
TeletexString     IA5String
T61String         GraphicString
VideotexString    GeneralString
UniversalString   BMPString
```

Les types utiles suivants sont définis aux § 42 à 44:

```
GeneralizedTime

UTCTime

ObjectDescriptor
```

Les productions suivantes sont utilisées dans les § 45 à 47:

```
ConstrainedType ::=
        Type   Constraint
    |   TypeWithConstraint

TypeWithConstraint ::=
        SET Constraint OF Type
    |   SET SizeConstraint OF Type
    |   SEQUENCE Constraint OF Type
    |   SEQUENCE SizeConstraint OF Type
    |   SET Constraint OF NamedType
    |   SET SizeConstraint OF NamedType
    |   SEQUENCE Constraint OF NamedType
    |   SEQUENCE SizeConstraint OF NamedType

Constraint ::= "(" ConstraintSpec  ExceptionSpec ")"

ConstraintSpec ::=    SubtypeConstraint
    |                 GeneralConstraint

ExceptionSpec ::= "!"  ExceptionIdentification | empty

ExceptionIdentification ::= SignedNumber
    |                       DefinedValue
    |                       Type ":" Value

SubtypeConstraint ::= ElementSetSpecs

ElementSetSpecs ::=
        RootElementSetSpec
    |   RootElementSetSpec  ","  "..."
    |   RootElementSetSpec  ","  "..."  ","  AdditionalElementSetSpec

RootElementSetSpec ::= ElementSetSpec

AdditionalElementSetSpec ::= ElementSetSpec

ElementSetSpec ::= Unions | ALL  Exclusions

Unions ::=    Intersections
    |         UElems UnionMark Intersections

UElems ::= Unions

Intersections ::=   IntersectionElements
    |               IElems IntersectionMark IntersectionElements

IElems ::= Intersections

IntersectionElements ::= Elements | Elems Exclusions
```

```
Elems ::= Elements

Exclusions ::= EXCEPT Elements

UnionMark ::= "|"   |   UNION

IntersectionMark ::= "^" |   INTERSECTION

Elements ::= SubtypeElements
    |        ObjectSetElements
    |        "(" ElementSetSpec ")"

SubtypeElements ::=
        SingleValue
    |   ContainedSubtype
    |   ValueRange
    |   PermittedAlphabet
    |   SizeConstraint
    |   TypeConstraint
    |   InnerTypeConstraints
    |   PatternConstraint

SingleValue   ::= Value

ContainedSubtype ::= Includes Type

Includes ::= INCLUDES | empty

ValueRange ::= LowerEndpoint ".." UpperEndpoint

LowerEndpoint ::= LowerEndValue | LowerEndValue "<"

UpperEndpoint ::= UpperEndValue | "<" UpperEndValue

LowerEndValue ::= Value | MIN

UpperEndValue ::= Value | MAX

SizeConstraint ::= SIZE Constraint

PermittedAlphabet ::= FROM Constraint

TypeConstraint ::= Type

InnerTypeConstraints ::=
        WITH COMPONENT SingleTypeConstraint
    |   WITH COMPONENTS MultipleTypeConstraints

SingleTypeConstraint::= Constraint

MultipleTypeConstraints ::= FullSpecification | PartialSpecification

FullSpecification   ::= "{" TypeConstraints "}"

PartialSpecification ::= "{" "..." "," TypeConstraints "}"

TypeConstraints ::=   NamedConstraint
    |                 NamedConstraint "," TypeConstraints

NamedConstraint ::=   identifier ComponentConstraint

ComponentConstraint ::= ValueConstraint PresenceConstraint

ValueConstraint ::= Constraint | empty

PresenceConstraint ::= PRESENT | ABSENT | OPTIONAL | empty

PatternConstraint ::= PATTERN Value
```

SÉRIES DES RECOMMANDATIONS UIT-T

Série A	Organisation du travail de l'UIT-T
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	Gestion des télécommunications y compris le RGT et maintenance des réseaux
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, communication entre systèmes ouverts et sécurité**
Série Y	Infrastructure mondiale de l'information, protocole Internet et réseaux de prochaine génération
Série Z	Langages et aspects généraux logiciels des systèmes de télécommunication

Imprimé en Suisse
Genève, 2006