UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T X.692

(03/2002)

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

SERIE X: REDES DE DATOS Y COMUNICACIÓN ENTRE SISTEMAS ABIERTOS

Gestión de redes de interconexión de sistemas abiertos y aspectos de sistemas – Notación de sintaxis abstracta uno

Tecnología de la información – Reglas de codificación de notación de sintaxis abstracta uno: Especificación de la notación de control de codificación

Recomendación UIT-T X.692

RECOMENDACIONES UIT-T DE LA SERIE X

REDES DE DATOS Y COMUNICACIÓN ENTRE SISTEMAS ABIERTOS

REDES PÚBLICAS DE DATOS	
Servicios y facilidades	X.1–X.19
Interfaces	X.20–X.49
Transmisión, señalización y conmutación	X.50–X.89
Aspectos de redes	X.90–X.149
Mantenimiento	X.150–X.179
Disposiciones administrativas	X.180–X.199
INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Modelo y notación	X.200–X.209
Definiciones de los servicios	X.210–X.219
Especificaciones de los protocolos en modo conexión	X.220–X.229
Especificaciones de los protocolos en modo sin conexión	X.230–X.239
Formularios para declaraciones de conformidad de implementación de protocolo	X.240–X.259
Identificación de protocolos	X.260–X.269
Protocolos de seguridad	X.270–X.279
Objetos gestionados de capa	X.280–X.289
Pruebas de conformidad	X.290–X.299
INTERFUNCIONAMIENTO ENTRE REDES	
Generalidades	X.300–X.349
Sistemas de transmisión de datos por satélite	X.350–X.369
Redes basadas en el protocolo Internet	X.370–X.399
SISTEMAS DE TRATAMIENTO DE MENSAJES	X.400–X.499
DIRECTORIO	X.500–X.599
GESTIÓN DE REDES DE INTERCONEXIÓN DE SISTEMAS ABIERTOS Y ASPECTOS DE SISTEMAS	
Gestión de redes	X.600–X.629
Eficacia	X.630–X.639
Calidad de servicio	X.640–X.649
Denominación, direccionamiento y registro	X.650–X.679
Notación de sintaxis abstracta uno	**X.680–X.699**
GESTIÓN DE INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Marco y arquitectura de la gestión de sistemas	X.700–X.709
Servicio y protocolo de comunicación de gestión	X.710–X.719
Estructura de la información de gestión	X.720–X.729
Funciones de gestión y funciones de arquitectura de gestión distribuida abierta	X.730–X.799
SEGURIDAD	X.800–X.849
APLICACIONES DE INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Compromiso, concurrencia y recuperación	X.850–X.859
Procesamiento de transacciones	X.860–X.879
Operaciones a distancia	X.880–X.899
PROCESAMIENTO DISTRIBUIDO ABIERTO	X.900–X.999

Para más información, véase la Lista de Recomendaciones del UIT-T.

NORMA INTERNACIONAL ISO/CEI 8825-3
RECOMENDACIÓN UIT-T X.692

Tecnología de la información – Reglas de codificación de notación de sintaxis abstracta uno: Especificación de la notación de control de codificación

Resumen

La presente Recomendación | Norma Internacional define la notación de control de codificación (ECN) utilizada para especificar codificaciones (de tipos ASN.1) que difieren de las obtenidas aplicando reglas de codificación normalizadas tales como las reglas de codificación básica (BER) y las reglas de codificación compactada (PER).

Orígenes

La Recomendación UIT-T X.692, preparada por la Comisión de Estudio 17 (2001-2004) del UIT-T, fue aprobada el 8 de marzo de 2002. Se publica también un texto idéntico como Norma Internacional ISO/CEI 8825-3.

PREFACIO

La UIT (Unión Internacional de Telecomunicaciones) es el organismo especializado de las Naciones Unidas en el campo de las telecomunicaciones. El UIT-T (Sector de Normalización de las Telecomunicaciones de la UIT) es un órgano permanente de la UIT. Este órgano estudia los aspectos técnicos, de explotación y tarifarios y publica Recomendaciones sobre los mismos, con miras a la normalización de las telecomunicaciones en el plano mundial.

La Asamblea Mundial de Normalización de las Telecomunicaciones (AMNT), que se celebra cada cuatro años, establece los temas que han de estudiar las Comisiones de Estudio del UIT-T, que a su vez producen Recomendaciones sobre dichos temas.

La aprobación de Recomendaciones por los Miembros del UIT-T es el objeto del procedimiento establecido en la Resolución 1 de la AMNT.

En ciertos sectores de la tecnología de la información que corresponden a la esfera de competencia del UIT-T, se preparan las normas necesarias en colaboración con la ISO y la CEI.

NOTA

En esta Recomendación, la expresión "Administración" se utiliza para designar, en forma abreviada, tanto una administración de telecomunicaciones como una empresa de explotación reconocida de telecomunicaciones.

PROPIEDAD INTELECTUAL

La UIT señala a la atención la posibilidad de que la utilización o aplicación de la presente Recomendación suponga el empleo de un derecho de propiedad intelectual reivindicado. La UIT no adopta ninguna posición en cuanto a la demostración, validez o aplicabilidad de los derechos de propiedad intelectual reivindicados, ya sea por los miembros de la UIT o por terceros ajenos al proceso de elaboración de Recomendaciones.

En la fecha de aprobación de la presente Recomendación, la UIT no ha recibido notificación de propiedad intelectual, protegida por patente, que puede ser necesaria para aplicar esta Recomendación. Sin embargo, debe señalarse a los usuarios que puede que esta información no se encuentre totalmente actualizada al respecto, por lo que se les insta encarecidamente a consultar la base de datos sobre patentes de la TSB.

ÍNDICE

Página

Introducción

La notación de control de codificación (ECN) es una notación utilizada para especificar codificaciones de tipos ASN.1 que difieren de las obtenidas aplicando reglas de codificación normalizadas. La ECN se puede utilizar para codificar todos los tipos de una codificación ASN.1, pero se puede utilizar también con reglas de codificación normalizadas tales como las BER o las PER (Rec. UIT-T X.690 | ISO/CEI 8825-1 y Rec. UIT-T X.691 | ISO/CEI 8825-2) para especificar solamente las codificaciones de tipos que tienen requisitos especiales.

Un tipo ASN.1 especifica un conjunto de valores abstractos. Las reglas de codificación especifican la representación de esos valores abstractos como una serie de bits. La ECN está concebida de manera que satisfaga los requisitos de codificación siguientes:

a) La necesidad de escribir tipos ASN.1 (y obtener el soporte de herramientas de la ASN.1 en las implementaciones) para protocolos (de "legado") establecidos cuando la codificación ya está determinada y difiere de todas las reglas de codificación normalizadas.

b) La necesidad de producir codificaciones que son pequeñas variaciones de reglas normalizadas.

La vinculación con una especificación ASN.1 proporcionada en una especificación ECN está bien definida y es procesable por ordenador, por lo que los codificadores y decodificadores pueden ser generados automáticamente a partir de las especificaciones combinadas. Éste es un factor importante a efectos de reducción de la carga de trabajo y la posibilidad de cometer errores al hacer que los sistemas sean interoperables. Otra ventaja significativa es la posibilidad de proporcionar el soporte automático de herramientas para la realización de pruebas.

Las ventajas anteriores sólo se pueden aprovechar con ASN.1 cuando las reglas de codificación normalizadas son suficientes. El trabajo con ECN permite disponer de esas ventajas en circunstancias en las que no bastan las reglas de codificación normalizadas.

NOTA 1 – En la actualidad, la ECN admite únicamente codificaciones binarias, pero podría ampliarse en el futuro para abarcar codificaciones basadas en caracteres.

El anexo A es parte de la presente Recomendación | Norma Internacional y detalla las modificaciones que es preciso introducir en la Rec. UIT-T X.680 | ISO/CEI 8824-1 para admitir la notación utilizada en esta Recomendación | Norma Internacional.

El anexo B es parte de la presente Recomendación | Norma Internacional y detalla las modificaciones que es preciso introducir en la Rec. UIT-T X.681 | ISO/CEI 8824-2 para admitir la notación utilizada en esta Recomendación | Norma Internacional.

El anexo C es parte de la presente Recomendación | Norma Internacional y detalla las modificaciones que es preciso introducir en la Rec. UIT-T X.683 | ISO/CEI 8824-4 para admitir la notación utilizada en esta Recomendación | Norma Internacional.

NOTA 2 – No se pretende que los anexos A, B y C se conviertan en enmiendas a las Recomendaciones | Normas Internacionales referenciadas. Las modificaciones sólo tienen por objeto la definición de la ECN (véanse la cláusula 5 y 9.26).

El anexo D no es parte integrante de la presente Recomendación | Norma Internacional y contiene ejemplos de utilización de la ECN.

El anexo E no es parte integrante de la presente Recomendación | Norma Internacional y da más detalles sobre el soporte de las codificaciones Huffman en la ECN.

El anexo F no es parte integrante de la presente Recomendación | Norma Internacional e identifica un sitio web que permite el acceso a más información sobre la ECN y enlaces propios de la misma.

El anexo G no es parte integrante de la presente Recomendación | Norma Internacional y contiene un resumen de la ECN utilizando la notación de la cláusula 5.

NORMA INTERNACIONAL
RECOMENDACIÓN UIT-T

Tecnología de la información – Reglas de codificación de notación de sintaxis abstracta uno: Especificación de la notación de control de codificación

1 Alcance

La presente Recomendación | Norma Internacional define una notación para la especificación de codificaciones de tipos o partes de tipos ASN.1.

Proporciona varios mecanismos aplicables a dicha especificación, a saber:

- especificación directa de la codificación utilizando notación normalizada;
- especificación de la codificación por referencia a reglas de codificación normalizadas;
- especificación de la codificación de un tipo ASN.1 por referencia a una estructura de codificación;
- especificación de la codificación utilizando una notación distinta de la ECN.

Proporciona además la manera de vincular la especificación de codificaciones con las definiciones de los tipos a los que se han de aplicar.

2 Referencias normativas

Las siguientes Recomendaciones y Normas Internacionales contienen disposiciones que, mediante su referencia en este texto, constituyen disposiciones de la presente Recomendación | Norma Internacional. Al efectuar esta Recomendación, estaban en vigor las ediciones indicadas. Todas las Recomendaciones y Normas Internacionales son objeto de revisiones por lo que se preconiza que los usuarios de esta Recomendación | Norma Internacional investiguen la posibilidad de aplicar las ediciones más recientes de las Recomendaciones y Normas citadas a continuación. Los miembros de la CEI y la ISO mantienen registros de las Normas Internacionales válidas actualmente. La oficina de Normalización de las Telecomunicaciones de la UIT mantiene una lista de las Recomendaciones UIT-T actualmente vigentes.

2.1 Recomendaciones | Normas Internacionales idénticas

- Recomendación UIT-T X.660 (1992) | ISO/CEI 9834-1:1993, *Tecnología de la información – Interconexión de sistemas abiertos – Procedimientos para la operación de autoridades de registro para interconexión de sistemas abiertos: Procedimientos generales (más enmiendas).*
- Recomendación UIT-T X.680 (2002) | ISO/CEI 8824-1:2002, *Tecnología de la información – Notación de sintaxis abstracta uno: Especificación de la notación básica*.
- Recomendación UIT-T X.681 (2002) | ISO/CEI 8824-2:2002, *Tecnología de la información – Notación de sintaxis abstracta uno: Especificación de objetos de información.*
- Recomendación UIT-T X.682 (2002) | ISO/CEI 8824-3:2002, *Tecnología de la información – Notación de sintaxis abstracta uno: Especificación de constricciones.*
- Recomendación UIT-T X.683 (2002) | ISO/CEI 8824-4:2002, *Tecnología de la información – Notación de sintaxis abstracta uno: Parametrización de especificaciones de notación de sintaxis abstracta uno.*
- Recomendación UIT-T X.690 (2002) | ISO/CEI 8825-1:2002, *Tecnología de la información – Reglas de codificación de notación de sintaxis abstracta uno: Especificación de las reglas de codificación básica, de las reglas de codificación canónica y de las reglas de codificación distinguida.*
- Recomendación UIT-T X.691 (2002) | ISO/CEI 8825-2:2002, *Tecnología de la información – Reglas de codificación de notación de sintaxis abstracta uno: Especificación de las reglas de codificación compactada*.

NOTA 1 – Con independencia de la fecha de publicación ISO, normalmente se hace referencia a las especificaciones anteriores como "ASN.1:2002".

NOTA 2 – Las referencias anteriores deberán interpretarse como referencias a las Recomendaciones | Normas Internacionales junto con todas sus enmiendas y corrigendos técnicos publicados.

2.2 Referencias adicionales

– ISO/CEI 10646-1:1993, *Information technology – Universal Multiple-Octet Coded Character Set (UCS) Part 1: Architecture and Basic Multilingual Plane*.

NOTA – La referencia anterior deberá interpretarse como una referencia a ISO/CEI 10646-1 junto con todas sus enmiendas y corrigendos técnicos publicados.

3 Definiciones

Para los fines de esta Recomendación | Norma Internacional, se aplican las definiciones siguientes.

3.1 Definiciones ASN.1

Esta Recomendación | Norma Internacional utiliza los términos definidos en la cláusula 3 de la Rec. UIT-T X.680 | ISO/CEI 8824-1, la Rec. UIT-T X.681 | ISO/CEI 8824-2, la Rec. UIT-T X.682 | ISO/CEI 8824-3, la Rec. UIT-T X.683 | ISO/CEI 8824-4, la Rec. UIT-T X.690 | ISO/CEI 8825-1 y la Rec. UIT-T X.691 | ISO/CEI 8825-2.

3.2 Definiciones específicas de la ECN

3.2.1 punto de alineación: Punto de una codificación (normalmente su comienzo) que sirve como punto de referencia cuando una especificación de codificación requiere la alineación con alguna frontera.

3.2.2 campo auxiliar: Campo de una estructura de sustitución (que se añade en la especificación ECN) cuyo valor lo fija directamente el codificador sin utilizar ningún valor abstracto proporcionado por la aplicación.

NOTA – Un ejemplo de campo auxiliar es el determinante de longitud de una codificación de entero o de una repetición.

3.2.3 campo de bits: Bits u octetos contiguos en una codificación que se decodifican como un todo y que representan un valor abstracto o proporcionan información (tal como el determinante de longitud de algún otro campo, véase 3.2.30) necesaria para que la decodificación se haga de manera satisfactoria, o bien ambas cosas.

NOTA – Lo de "o bien ambas cosas" se aplica, en su caso, en los protocolos de legado.

3.2.4 clase de campo de bits: Clase de codificación cuyos objetos especifican la codificación de valores abstractos (de algún tipo ASN.1) en bits.

NOTA – Otras clases de codificación se refieren a procedimientos de codificación más generales, tales como los requeridos para determinar el final de repeticiones de codificaciones de clase de campo de bits, o para determinar cuál de las codificaciones de un conjunto de codificaciones de campos de bits alternativas está presente.

3.2.5 condición de límites: Condición relativa a la existencia de límites de un campo de entero (y a si se permiten o no valores negativos) que, si se cumple, significa que se han de aplicar las reglas de codificación especificadas.

3.2.6 determinante de elección: Campo de bits que determina qué codificación, de entre varias posibles (cada una de las cuales representa valores abstractos diferentes), está presente en algún otro campo de bits.

3.2.7 conjunto de objetos de codificación combinados: Conjunto temporal de objetos de codificación producido por la combinación de dos conjuntos de objetos de codificación para la aplicación de codificaciones.

3.2.8 codificación condicional: Codificación que sólo se ha de aplicar si se cumple alguna condición de límites o condición de gama de tamaños especificada.

3.2.9 tipo conteniendo: Tipo ASN.1 (o campo de estructura de codificación) cuando se ha aplicado una constricción de contenido a los valores de ese tipo (o a los valores asociados a ese campo de estructura de codificación).

NOTA – Los tipos ASN.1 a los que se puede aplicar una constricción de contenido (utilizando **`CONTAINING`**/**`ENCODED BY`**) son los tipos cadena de bits y cadena de octetos.

3.2.10 punto de aplicación actual: Punto de una estructura de codificación en el que se aplica un conjunto de objetos de codificación combinados.

3.2.11 codificación–decodificación diferencial: Especificación de las reglas de un decodificador que requieren la aceptación de codificaciones que no pueden ser producidas por un codificador conforme a la especificación actual.

NOTA – La codificación-decodificación diferencial soporta la especificación de la decodificación por un decodificador, conforme a la versión inicial de una norma, que tiene por objeto permitirle decodificar de manera satisfactoria codificaciones producidas por una versión posterior de esa norma. A veces, para referirse a esto, se habla de soporte de la extensibilidad.

3.2.12 clase de codificación: Conjunto de todas las codificaciones posibles de una parte específica de los procedimientos necesarios para efectuar la codificación o decodificación de un tipo ASN.1.

NOTA – Se definen clases de codificación para codificar tipos ASN.1 primitivos, pero se definen también para los procedimientos asociados a la notación de rótulo ASN.1, la utilización de **`OPTIONAL`** y los constructores de codificación.

3.2.13 categoría de clases de codificación: Clases de codificación con algunas características comunes.

NOTA – Ejemplos al respecto son la categoría entero, la categoría booleano y la categoría concatenación.

3.2.14 constructor de codificación: Clase de codificación cuyos objetos de codificación definen procedimientos para combinar, seleccionar o repetir partes de una codificación. (Ejemplos al respecto son las clases **`#ALTERNATIVES`**, **`#CHOICE`**, **`#CONCATENATION`**, **`#SEQUENCE`**, etc.).

3.2.15 módulos de definición de codificación (EDM, *encoding definition modules*): Módulos que definen codificaciones de aplicación en el módulo de enlace de codificación.

3.2.16 módulo de enlace de codificación (ELM, *encoding link module*): Módulo (único, cualquiera que sea la aplicación de que se trate) que asigna codificaciones a tipos ASN.1.

3.2.17 objeto de codificación: Especificación de alguna parte de los procedimientos necesarios para efectuar la codificación o la decodificación de un tipo ASN.1.

NOTA – Los objetos de codificación pueden especificar la codificación de tipos ASN.1 primitivos, pero también pueden especificar los procedimientos asociados a la notación de rótulo ASN.1, la utilización de **`OPTIONAL`** y los constructores de codificación.

3.2.18 conjunto de objetos de codificación: Un conjunto de objetos de codificación.

NOTA – Normalmente se emplea un conjunto de objetos de codificación en el módulo de enlace de codificación para determinar la codificación de todos los tipos de nivel máximo utilizados en una aplicación.

3.2.19 propiedad de codificación: Elemento de información utilizado para definir una codificación empleando la notación especificada en las cláusulas 23, 24 y 25.

3.2.20 espacio de codificación: Número de bits (u octetos, palabras u otras unidades) utilizados para codificar un valor abstracto en un campo de bits (véase 9.21.5).

3.2.21 estructura de codificación: Estructura de una codificación definida a partir de la estructura de la definición de un tipo ASN.1, o en un EDM utilizando clases de campo de bits y constructores de codificación.

NOTA 1 – La utilización de una estructura de codificación es sólo uno de los varios mecanismos (si bien un mecanismo importante) que la notación de control de codificación proporciona para la definición de codificaciones de tipos ASN.1.

NOTA 2 – La definición de una estructura de codificación es también la definición de una clase de codificación correspondiente.

3.2.22 estructura de codificación generada explícitamente: Estructura de codificación derivada de una estructura de codificación generada implícitamente utilizando la cláusula de redenominaciones de un EDM.

3.2.23 extensibilidad: Disposiciones de una versión anterior de una norma que tienen por objeto maximizar el interfuncionamiento de las implementaciones de esa versión anterior con las implementaciones previstas de una versión posterior de la norma.

3.2.24 nombre totalmente cualificado: Referencia a una clase, objeto o conjunto de objetos de codificación que incluye el nombre del módulo EDM en el que se definió esa clase o ese objeto o ese conjunto de objetos de codificación, o (en el caso de una clase de codificación generada implícitamente) el nombre del módulo ASN.1 en el que se generó. (Véase también 3.2.42).

NOTA – Se ha de utilizar un nombre totalmente cualificado (véase la producción "ExternalEncodingClassReference" en 10.6) en el cuerpo de un módulo si la clase de codificación es una estructura de codificación generada implícitamente cuyo nombre es el mismo que el de una clase reservada, o si la utilización del nombre sólo produjera ambigüedad debido a las múltiples importaciones de clases con ese nombre. (Véase A.1/12.15).

3.2.25 estructura de codificación generada: Estructura de codificación generada implícita o explícitamente cuyo objetivo es definir las codificaciones del tipo ASN.1 correspondiente mediante la aplicación de codificaciones en el ELM.

3.2.26 gobernador: Parte de una especificación ECN que determina la forma sintáctica (y la semántica) de alguna otra parte de la especificación ECN.

NOTA – Un gobernador es una referencia de clase de codificación, y determina la sintaxis que se ha de utilizar para la definición de un objeto de codificación (de esa clase). El concepto es el mismo que el de una referencia de tipo en ASN.1 que actúa como el gobernador de la notación del valor ASN.1.

3.2.27 asa de identificación: Parte de una codificación que sirve para distinguir las codificaciones de una clase de codificación de las de otras clases de codificación.

NOTA – Las reglas de codificación básica de la ASN.1 utilizan rótulos para proporcionar asas de identificación en las codificaciones BER.

3.2.28 estructura de codificación generada implícitamente: Estructura de codificación que es generada implícitamente y exportada siempre que se define un tipo en un módulo ASN.1.

3.2.29 punto de aplicación inicial: Punto de cualquier estructura de codificación en el que se aplica primero cualquier conjunto de objetos de codificación combinados (en el ELM y en los EDM).

3.2.30 determinante de longitud: Campo de bits que determina la longitud de algún otro campo de bits.

3.2.31 valor entero negativo: Un valor inferior a cero.

3.2.32 valor entero no negativo: Un valor superior o igual a cero.

3.2.33 valor entero no positivo: Un valor inferior o igual a cero.

3.2.34 campo de bits opcional: Campo de bits que a veces se incluye (para codificar un valor abstracto) y a veces se omite.

3.2.35 valor entero positivo: Un valor superior a cero.

3.2.36 determinante de presencia: Campo de bits que determina si está presente o no un campo de bits opcional.

3.2.37 clase primitiva: Clase de codificación que no es una estructura de codificación, y que no se puede desreferenciar a ninguna otra clase (véase 16.1.14).

3.2.38 definición recursiva (de un nombre de referencia): Nombre de referencia para el que la resolución del nombre de referencia, o del gobernador de la definición del nombre de referencia, requiere la resolución del nombre de referencia original.

NOTA – Se permite la definición recursiva de una clase de codificación (incluyendo una estructura de codificación). La definición recursiva de un objeto de codificación o de un conjunto de objetos de codificación está prohibida por 17.1.4 y 18.1.3, respectivamente.

3.2.39 instanciación recursiva (de un nombre de referencia parametrizada): Instanciación de un nombre de referencia, cuando la resolución de los parámetros reales requiere la resolución del nombre de referencia original.

NOTA – Se permite la instanciación recursiva de una clase de codificación (incluyendo una estructura de codificación). La instanciación recursiva de un objeto de codificación o de un conjunto de objetos de codificación está prohibida por 17.1.4 y 18.1.3, respectivamente.

3.2.40 estructura de sustitución: Estructura parametrizada utilizada para sustituir algunas o todas las partes de una construcción antes de codificar la construcción.

3.2.41 codificación autodelimitante: Codificación de un conjunto de valores abstractos de tal manera que no haya ningún valor abstracto con una codificación que sea una subcadena inicial de la codificación de cualquier otro valor abstracto del conjunto.

NOTA – Se incluyen aquí no sólo las codificaciones de longitud fija de un entero limitado, sino también las codificaciones descritas por lo general como "codificaciones Huffman" (véase el anexo E).

3.2.42 nombre de referencia simple: Referencia a una clase, objeto o conjunto de objetos de codificación que no incluye el nombre del módulo EDM en el que se definió esa clase o ese objeto o conjunto de objetos de codificación ni (en el caso de una clase de codificación generada implícitamente) el nombre del módulo ASN.1 en el que se generó.

NOTA – Un nombre de referencia simple sólo se puede utilizar cuando la referencia a la clase de codificación es inequívoca; de no ser así se ha de utilizar un nombre totalmente cualificado (véase 3.2.24) en el cuerpo de un módulo.

3.2.43 condición de gama de tamaños: Condición relativa a la existencia de constricciones de tamaño efectivo en un campo de cadena o repetición (y a si la constricción incluye cero, y/o si permite múltiples tamaños), que, si se cumple, significa que se han de aplicar las reglas de codificación especificadas.

3.2.44 gobernador origen (o clase origen): Gobernador que determina la notación para especificar valores abstractos asociados a una clase origen cuando se establece la correspondencia entre ellos y una clase destino.

3.2.45 puntero de comienzo: Campo auxiliar que indica la presencia o ausencia de un campo de bits opcional, y en caso de presencia, contiene el desplazamiento del campo de bits con respecto a la posición actual.

3.2.46 gobernador destino (o clase destino): Gobernador que determina la notación para especificar valores abstractos asociados a una clase destino cuando se establece la correspondencia con ellos desde una clase origen.

3.2.47 tipo(s) de nivel máximo: El tipo o los tipos ASN.1 de una aplicación utilizados por la aplicación para fines distintos del de definir los componentes de otros tipos ASN.1.

NOTA 1 – Los tipos de nivel máximo se pueden utilizar también (aunque no es lo normal) como componentes de otros tipos ASN.1.

NOTA 2 – A veces se hace referencia a los tipos de nivel máximo como "los mensajes de la aplicación" o "las PDU". Esos tipos se tratan normalmente aplicando herramientas especiales, ya que constituyen el nivel máximo de las estructuras de datos del lenguaje de programación que se presentan a la aplicación.

3.2.48 transformadas: Objetos de codificación de la clase **`#TRANSFORM`** que especifican que la codificación de los valores abstractos asociados a alguna clase (o de compuestos de transformadas, véase 3.2.49) ha de ser la codificación de diferentes valores abstractos asociados a la misma clase o a una clase diferente (o de compuestos de transformadas).

NOTA – Las transformadas se pueden utilizar, por ejemplo, para especificar operaciones aritméticas sencillas con valores enteros, o para establecer la correspondencia entre valores enteros y cadenas de caracteres o cadenas de bits.

3.2.49 compuesto de transformadas: Lista ordenada de elementos que puede ser, ella misma, el origen o el resultado de transformadas.

NOTA – Es preciso que todos los elementos de un compuesto tengan la misma clasificación (véase 9.18.2).

3.2.50 codificación de valor: Manera de utilizar un espacio de codificación para representar un valor abstracto (véase 9.21.5).

4 Abreviaturas

A los efectos de esta Recomendación | Norma Internacional se aplicarán las siglas siguientes:

ASN.1 Notación de sintaxis abstracta uno (*abstract syntax notation one*)

BCD Decimal codificado en binario (*binary coded decimal*)

BER Reglas básicas de codificación de ASN.1 (*basic encoding rules of ASN.1*)

CER Reglas de codificación canónica de ASN.1 (*canonical encoding rules of ASN.1*)

DER Reglas de codificación distinguida de ASN.1 (*distinguished encoding rules of ASN.1*)

ECN Notación de control de codificación para ASN.1 (*encoding control notation for ASN.1*)

EDM Módulo de definición de codificación (*encoding definition module*)

ELM Módulo de enlace de codificación (*encoding link module*)

PDU Unidad de datos de protocolo (*protocol data unit*)

PER Reglas de codificación compactada de ASN.1 (*packed encoding rules of ASN.1*)

5 Definición de la sintaxis de la ECN

5.1 Esta Recomendación | Norma Internacional emplea el convenio notacional definido en la cláusula 5 de la Rec. UIT-T X.680 | ISO/CEI 8824-1.

5.2 Esta Recomendación | Norma Internacional emplea la notación de clases de objetos de información definida en la Rec. UIT-T X.681 | ISO/CEI 8824-2 modificada por el anexo B.

5.3 Esta Recomendación | Norma Internacional hace referencia a producciones definidas en la Rec. UIT-T X.680 | ISO/CEI 8824-1 modificada por el anexo A, la Rec. UIT-T X.681 | ISO/CEI 8824-2 modificada por el anexo B y la Rec. UIT-T X.683 | ISO/CEI 8824-4 modificada por el anexo C.

6 Convenios de codificación y notación

6.1 Esta Recomendación | Norma Internacional define el valor de cada octeto de una codificación utilizando las expresiones "bit más significativo" y "bit menos significativo".

NOTA – Las especificaciones de capa más baja utilizan la misma notación para definir el orden de transmisión de los bits por una línea en serie, o la asignación de los bits a canales en paralelo.

6.2 Para los fines de esta Recomendación | Norma Internacional, los bits de un octeto se numeran de 8 a 1, siendo el bit 8 el "bit más significativo" y el bit 1 el "bit menos significativo".

6.3 Para los fines de esta Recomendación | Norma Internacional, las codificaciones se definen como una cadena de bits que va de un "bit inicial" a un "bit final". En la transmisión, los ocho primeros bits de esa cadena de bits, que empieza con el "bit inicial", se colocarán en el primer octeto transmitido, siendo el bit inicial el bit más significativo de ese octeto. Los ocho bits siguientes se colocarán en el octeto siguiente, y así sucesivamente. Si la codificación no comprende un número de bits que sea múltiplo de ocho, los bits restantes se transmitirán como si fueran ocho bits colocados en orden descendente en el octeto subsiguiente.

NOTA – Una codificación ECN completa no necesariamente ha de ser siempre un múltiplo de ocho bits, pero una especificación ECN puede determinar la adición de relleno para asegurar esta propiedad.

6.4 Cuando en esta Recomendación | Norma Internacional se representen números binarios, el "bit inicial" aparecerá siempre a la izquierda de la figura.

7 Conjunto de caracteres de la ECN

7.1 La utilización del término "carácter" empleado a lo largo del presente documento hace referencia a los caracteres especificados en ISO/CEI 10646-1, y el pleno soporte de todas las especificaciones ECN posibles quizá requiera la representación de todos esos caracteres.

7.2 Las especificaciones ECN sólo utilizan los caracteres indicados en el cuadro 1, salvo en los comentarios (definidos en 11.6 de la Rec. UIT-T X.680 | ISO/CEI 8824-1), la definición no ECN de objetos de codificación (véase 17.8) y los valores cadenas de caracteres.

7.3 Los elementos de léxico definidos en la cláusula 8 consisten en una secuencia de los caracteres indicados en el cuadro 1.

NOTA – En la cláusula 8 se especifican otras limitaciones a propósito de los caracteres permitidos para cada elemento de léxico.

Cuadro 1 – Caracteres de la ECN

0 a 9	(DÍGITO 0 a 9)
A a Z	(A (MAYÚSCULA) a Z (MAYÚSCULA))
a a z	(a (MINÚSCULA) a z (MINÚSCULA))
"	(COMILLAS)
#	(SIGNO DE NÚMERO)
&	(SIGNO DE "Y" COMERCIAL)
'	(APÓSTROFO)
(	(PARÉNTESIS IZQUIERDO)
)	(PARÉNTESIS DERECHO)
,	(COMA)
-	(GUIÓN-SIGNO DE MENOS)
.	(PUNTO)
:	(DOS PUNTOS)
;	(PUNTO Y COMA)
=	(SIGNO DE IGUAL)
{	(LLAVE IZQUIERDA DE APERTURA)
|	(BARRA VERTICAL)
}	(LLAVE DERECHA DE CIERRE)

7.4 No se dará relevancia al estilo tipográfico, el tamaño, el color, la intensidad u otras características de la presentación visual.

7.5 Las letras mayúsculas y minúsculas se considerarán distintas.

8 Elementos de léxico de la ECN

Además de los elementos de léxico de la ASN.1 especificados en la cláusula 11 de la Rec. UIT-T X.680 | ISO/CEI 8824-1, la presente Recomendación | Norma Internacional utiliza los elementos de léxico especificados en la subcláusulas que siguen. En esta cláusula son aplicables las reglas generales especificadas en 11.1 de la Rec. UIT-T X.680 | ISO/CEI 8824-1.

NOTA – El anexo G contiene la relación de todos los elementos de léxico utilizados en esta Recomendación | Norma Internacional, identificando los que se definen en la Rec. UIT-T X.680 | ISO/CEI 8824-1, en la Rec. UIT-T X.681 | ISO/CEI 8824-2 y en la Rec. UIT-T X.683 | ISO/CEI 8824-4.

8.1 Referencias de objeto de codificación

Nombre del elemento: encodingobjectreference

Una "encodingobjectreference" estará formada por la secuencia de caracteres especificada para una "valuereference" en 11.4 de la Rec. UIT-T X.680 | ISO/CEI 8824-1. Al analizar una instancia del empleo de esta notación, una "encodingobjectreference" se distingue de un "identifier" por el contexto en que aparece.

8.2 Referencias de conjunto de objetos de codificación

Nombre del elemento: encodingobjectsetreference

Una "encodingobjectsetreference" estará formada por la secuencia de caracteres especificada para una "typereference" en 11.2 de la Rec. UIT-T X.680 | ISO/CEI 8824-1. No deberá ser una de las secuencias de caracteres indicadas en 8.4.

8.3 Referencias de clase de codificación

Nombre del elemento: encodingclassreference

Una "encodingclassreference" estará formada por el carácter "#" seguido por la secuencia de caracteres especificada para una "typereference" en 11.2 de la Rec. UIT-T X.680 | ISO/CEI 8824-1. No deberá ser una de las secuencias de caracteres indicadas en 8.5 salvo en una lista de importación de EDM (véase 12.19 de la Rec. UIT-T X.680 | ISO/CEI 8824-1, modificada por A.1) o en una "ExternalEncodingClassReference" (véase la nota de 14.11).

8.4 Elementos de palabras reservadas

Nombres de elementos de palabras reservadas:

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

Los elementos con los nombres anteriores estarán formados por la secuencia de caracteres del nombre.

NOTA – Las palabras (véase 7.9 de la Rec. UIT-T X.681 | ISO/CEI 8824-2) utilizadas para definir clases de codificación (dentro de una declaración "WITH SYNTAX") en la cláusula 23 no son palabras reservadas (véase también B.14).

8.5 Elementos de nombres de clase de codificación reservados

Nombres de elementos de nombres de clase de codificación reservados:

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

```
#UTCTime                    #VideotexString
#UTF8String                 #VisibleString
```

Los elementos con los nombres anteriores estarán formados por la secuencia de caracteres del nombre.

8.6 Elemento no ECN

Nombre del elemento: anystringexceptnonecnend

Un "anystringexceptnonecnend" estará formado por uno o más caracteres del conjunto de caracteres ISO/CEI 10646-1, con la salvedad de que no será la secuencia de caracteres "**NON-ECN-END**", ni esa secuencia de caracteres aparecerá dentro de él.

9 Conceptos de la ECN

Esta cláusula describe los principales conceptos en los que se basa la presente Recomendación UIT-T | Norma Internacional.

9.1 Especificaciones de la notación de control de codificación (ECN)

9.1.1 Las especificaciones ECN constan de uno o más módulos de definición de codificación (los EDM) que definen reglas de codificación para tipos ASN.1, y un módulo de enlace de codificación (ELM) único que aplica esas reglas de codificación a los tipos ASN.1.

9.1.2 Lo más importante de la ECN es el concepto de **definición de estructura de codificación**. La ASN.1 se emplea para definir valores abstractos complejos utilizando constructores y tipos primitivos. Del mismo modo, las codificaciones complejas se pueden definir empleando una notación similar en la que se utilizan mecanismos de construcción para combinar campos de bits simples en codificaciones más complejas y, llegado el caso, en mensajes completos. Esto es lo que se denomina definición de una estructura de codificación. Cuando se utiliza la ECN con la ASN.1 se necesita en principio:

a) definir la sintaxis abstracta (el conjunto de valores abstractos que se ha de comunicar y su semántica); y

b) la estructura de codificación (la estructura de los campos) utilizada para llevar esos valores abstractos; y

c) relacionar los componentes de los valores abstractos con los campos de la estructura de codificación; y

d) definir la codificación de cada campo de la estructura de codificación y los mecanismos de identificación de repeticiones de campos e identificación de alternativas, etc.

9.1.3 El proceso anterior se desarrolla normalmente en varias etapas. Primero se produce una definición ASN.1 con la que se detalla la sintaxis abstracta. A continuación se genera automáticamente una estructura de codificación cruda (conceptualmente, dentro del módulo ASN.1). Esta estructura generada implícitamente sólo contiene campos que llevan la semántica de la aplicación y no contiene campo alguno para cosas tales como la determinación de longitud, la selección de alternativa, etc.

9.1.4 Esa estructura puede ser transformada por una serie de mecanismos en la estructura de campos que se requiere de hecho, incluyendo todos los cambios necesarios para el soporte de la actividad decodificadora (determinantes). Todos esos mecanismos implican alguna forma de sustitución de un campo sencillo que lleva semántica de aplicación por una estructura más compleja. Esas sustituciones constituyen una parte importante de la especificación ECN.

9.1.5 Es posible definir además **objetos de codificación** para cada uno de los campos de la estructura final. Así se determina no sólo la codificación de los campos sino también la manera en que un campo determina, por ejemplo, la longitud de otro campo o se resuelve su opcionalidad.

9.1.6 Las definiciones anteriores se producen en módulos de definición de codificación (los EDM). El último paso consiste en aplicar un conjunto de objetos de codificación definidos a la estructura de codificación final para determinar por completo una codificación. Esto es lo que se hace en el módulo de enlace de codificación (ELM).

9.2 Clases de codificación

9.2.1 Una clase de codificación es una propiedad implícita de todos los tipos ASN.1, y representa el conjunto de todas las especificaciones de codificación posibles para ese tipo. Proporciona una referencia que permite a los módulos de definición de codificación definir reglas de codificación de los campos de la estructura de codificación correspondientes al tipo. Los nombres de las clases de codificación empiezan con el carácter "**#**".

Ejemplo: Las reglas de codificación del tipo ASN.1 incorporado "**INTEGER**" se definen por referencia a la clase de codificación **#INTEGER**, y las reglas de codificación del tipo "**My-Type**" definido por el usuario se definen por referencia a la clase de codificación **#My-Type**.

9.2.2 Existen varios tipos de clases de codificación:

9.2.2.1 Clases de codificación incorporadas. Hay clases de codificación incorporadas con nombres tales como **#INTEGER** y **#BOOLEAN**. Esto permite la definición de codificaciones especiales para tipos ASN.1 primitivos. Hay además clases de codificación incorporadas para constructores de codificación tales como **#SEQUENCE**, **#SEQUENCE-OF** y **#CHOICE** (véase también 9.3.2) y para la definición de las reglas de codificación del tratamiento de la opcionalidad mediante **#OPTIONAL**. La codificación de rótulos corre a cargo de la clase **#TAG**. Hay, por último, varias clases incorporadas (**#OUTER**, **#TRANSFORM** y otras) que permiten definir procedimientos de codificación que forman parte del proceso de codificación/decodificación, pero que no están relacionadas directamente con ningún campo de bits real o constructivo ASN.1.

9.2.2.2 Clases de codificación de estructuras de codificación generadas implícitamente. Estas clases tienen nombres que constan del carácter "**#**" seguido por la "typereference" que aparece en una "TypeAssignment" de un módulo ASN.1. Son clases de codificación generadas implícitamente siempre que se asigna una "typereference" (no parametrizado) en un módulo ASN.1, y pueden ser importadas en un módulo de definición de codificación para hacer posible la definición de codificaciones especiales del tipo ASN.1 correspondiente. Estas clases de codificación representan la estructura de una codificación ASN.1, y se forman a partir de las clases de codificación incorporadas que reflejan la estructura de la definición del tipo ASN.1.

9.2.2.3 Clases de codificación de estructuras de codificación definidas por el usuario. Se trata de clases de codificación que define el usuario de la ECN especificando una estructura de codificación (véase 9.3) como una estructura formada por campos de bits y constructores de codificación. Estas estructuras de codificación son similares a las estructuras de codificación generadas implícitamente, pero el usuario de la ECN tiene pleno control de su estructura. Estas clases permiten definir reglas de codificación complejas, y son importantes a efectos de la utilización de la ASN.1 con la ECN al especificar protocolos de legado en donde se necesitan campos de bits adicionales para codificar determinantes.

9.2.2.4 Clases de codificación de estructuras de codificación generadas explícitamente. Se trata de clases de codificación producidas a partir de una estructura de codificación generada implícitamente, cambiando de manera selectiva los nombres de determinadas clases para indicar los lugares en los que se necesitan codificaciones especializadas para opcionalidad, terminación de secuencia de (*sequence-of*), etc.

9.3 Estructuras de codificación

9.3.1 Las definiciones de las estructuras de codificación son en cierto modo semejantes a las definiciones de tipos ASN.1, y tienen un nombre que comienza con el carácter "**#**" al que sigue una letra mayúscula. Cada definición de una estructura de codificación define una clase de codificación nueva (el conjunto de todas las codificaciones posibles de esa estructura de codificación). Las estructuras de codificación se forman a partir de campos que son las clases de codificación incorporadas o los nombres de otras estructuras de codificación, combinados utilizando constructores de codificación (que representan el conjunto de todas las reglas de codificación posibles que admiten su tipo de mecanismo de construcción, y se llaman por tanto clases de codificación). (Véase en D.2.8.4 un ejemplo de definición de estructura de codificación).

9.3.2 Los constructores de codificación más elementales son **#CONCATENATION**, **#REPETITION** y **#ALTERNATIVES**, que corresponden aproximadamente a los tipos ASN.1 secuencia (y conjunto), secuencia de (y conjunto de) y elección. Hay además una clase de codificación **#OPTIONAL** que representa la presencia opcional de codificaciones y corresponde aproximadamente a los marcadores **DEFAULT** y **OPTIONAL** de la ASN.1.

9.3.3 Una definición de estructura de codificación define una clase de codificación basada en la estructura. Esas clases no pueden tener los mismos nombres que las clases de codificación que se importan en el módulo. (Véase 12.12 de la Rec. UIT-T X.680 | ISO/CEI 8824-1, modificada por A.1 de la presente Recomendación | Norma Internacional).

9.3.4 Los nombres de las estructuras de codificación pueden ser exportados e importados entre módulos de definición de codificación y se pueden utilizar siempre que se necesiten en el grupo de categorías campo de bits (véase 9.6).

9.3.5 Se puede establecer la correspondencia entre valores de tipos ASN.1 (primitivos o definidos por el usuario) y campos de una estructura de codificación, y las reglas de codificación de esa estructura proporcionan entonces las codificaciones del tipo ASN.1. (Los valores cuya correspondencia con estructuras de codificación se ha establecido se pueden hacer corresponder a continuación con campos de estructura de codificación más compleja). De esta manera se dispone de un potente mecanismo de definición de reglas de codificación complejas.

9.4 Objetos de codificación

9.4.1 Los objetos de codificación representan la definición específica de reglas de codificación de una clase de codificación dada. Normalmente, las reglas se refieren a los bits que se han de producir realmente, pero también pueden especificar procedimientos relacionados con la codificación y la decodificación, por ejemplo, la manera de determinar la presencia o la ausencia de componentes opcionales.

9.4.2 Para definir por completo la codificación de tipos ASN.1 (normalmente, el tipo o los tipos de nivel máximo de una aplicación) es necesario definir (u obtener mediante reglas de codificación normalizadas) objetos de codificación de todas las clases que corresponden a componentes de esos tipos ASN.1 y de los constructores de codificación que se utilizan.

9.4.3 En el caso de protocolos de legado, es posible que esto se tenga que hacer definiendo un objeto de codificación separado por cada componente de un tipo ASN.1, pero lo más probable es que se utilicen objetos de codificación definidos por reglas de codificación normalizadas (por ejemplo, las PER).

9.4.4 Aunque las especificaciones de codificación BER y PER son anteriores a la ECN, dentro del modelo ECN definen simplemente objetos de codificación de todas las clases correspondientes a los constructores y tipos ASN.1 primitivos (es decir, de todas las clases de codificación incorporadas). También se tienen en cuenta las BER y las PER para proporcionar los objetos de codificación de las clases de codificación utilizadas en la definición de estructuras de codificación (véase 18.2).

9.5 Conjuntos de objetos de codificación

9.5.1 Los objetos de codificación se pueden agrupar en conjuntos de la misma manera que los objetos de información en ASN.1, y son esos conjuntos de objetos de codificación los que (en un ELM) se aplican a un tipo ASN.1 para determinar su codificación. El gobernador utilizado cuando se forman estos conjuntos de objetos de codificación es la palabra reservada **`#ENCODINGS`**. (Véase un ejemplo en D.1.14).

9.5.2 Una regla fundamental de la construcción de conjuntos de objetos de codificación es que cualquier conjunto sólo puede contener un objeto de codificación de una clase de codificación dada (véase también 9.6.2). Así pues, no hay ambigüedad alguna cuando se aplica un conjunto de objetos de codificación a un tipo para definir su codificación.

9.5.3 Hay conjuntos de objetos de codificación incorporados para todas las variantes de BER y PER, y se pueden utilizar para completar conjuntos de objetos de codificación definidos por el usuario.

9.6 Definición de clases de codificación nuevas

9.6.1 Las personas familiarizadas con la ASN.1 saben que se puede utilizar una asignación de tipo para crear nombres nuevos (tipos nuevos) a partir, por ejemplo, de los tipos **`INTEGER`** o **`BOOLEAN`**. Los nombres nuevos identifican tipos que son los mismos que **`INTEGER`** o **`BOOLEAN`**" pero llevan una semántica diferente. Este concepto se amplía en la ECN para hacer posible la creación (en una asignación de clase, véase 16.1.1) de nombres nuevos (clases nuevas) para constructores tales como **`#SEQUENCE`**. Los nombres nuevos identifican clases que realizan una función similar en la estructuración de codificaciones (por ejemplo, la concatenación), pero a las que se van a aplicar objetos de codificación diferentes. Un nombre de clase nuevo asignado a una clase antigua retiene determinadas características de esa clase antigua. Así pues, una asignación tal como **`#My-Sequence::= #SEQUENCE`** crea el nombre de clase nuevo **#My-Sequence** que sigue siendo una clase de codificación relacionada con la concatenación de componentes. Se dice que esas clases de codificación están en la misma categoría.

9.6.2 Si se crea una clase de codificación nueva a partir de una clase de codificación existente, los objetos de codificación de ambas clases de codificación, la antigua y la nueva, pueden aparecer en un conjunto de objetos de codificación.

9.6.3 Todas las clases de codificación incorporadas proceden de una de las clases de un pequeño número de clases de codificación primitivas. Así, por ejemplo, **`#SEQUENCE`** y **`#SET`** proceden ambas de la clase **`#CONCATENATION`**, las clases **`#INTEGER`** y **`#ENUMERATED`** se derivan de la clase **`#INT`**, y las clases de los diferentes tipos de cadena de caracteres ASN.1 proceden todas ellas de la clase **`#CHARS`**. Una estructura de codificación (por ejemplo, una estructura generada implícitamente a partir de un tipo ASN.1) puede contener una combinación de clases diferentes derivadas de la misma clase primitiva, lo que permite aplicar codificaciones diferentes a **`#SEQUENCE`** y a **`#SET`** (por ejemplo).

9.6.4 A menudo conviene agrupar las clases de codificación en categorías, sobre la base de la clase primitiva de la que proceden. De este modo, se dice que **#INTEGER**, **#ENUMERATED** e **#INT** (y cualquier clase derivada de ellas en una declaración de asignación de clase tal como "**#My-int::= #INT**") están en la categoría entero. Hay además grupos de categorías que contienen clases muy diferentes que comparten alguna característica. Cualquier clase que pueda tener valores abstractos asociados directamente a ella, y que produce por tanto bits en una codificación, se dice que está en el grupo de categorías campo de bits. Todas las clases que se hallan en la categoría entero o la categoría booleano o la categoría cadena de caracteres están en el grupo de categorías campo de bits. Las clases responsables de la agrupación o repetición de codificaciones (por ejemplo, las clases en la categoría alternativas o la categoría repetición) están en el grupo de categorías constructor de codificación. Existen, por otra parte, dos clases cuyos objetos de codificación definen procedimientos no relacionados directamente con la construcción de una codificación (**#TRANSFORM** y **#OUTER**): se describen diciendo que ambas están en el grupo de categorías procedimiento de codificación. Las estructuras de codificación se definen utilizando clases en el grupo de categorías campo de bits que se combinan utilizando clases en el grupo de categorías constructor de codificación, junto con clases en la categoría opcionalidad (que representa procedimientos de codificación para la resolución de la opcionalidad) y en la categoría rótulo (que representa la codificación de rótulos). Todas esas clases están en la categoría estructura de codificación (y también en el grupo de categorías campo de bits).

9.6.5 Para las clases primitivas, la categoría se asigna directamente. Para las clases creadas en una declaración de asignación de clase de codificación, la categoría viene determinada por la notación situada a la derecha del símbolo "**::=**". Si esa notación es una definición de estructura de codificación, la clase está tanto en la categoría estructura de codificación como en el grupo de categorías campo de bits. Si la notación es un nombre de referencia de clase simple, la categoría de la clase nueva es la misma que la categoría de la clase que se asigna.

9.6.6 Las categorías de las clases de codificación (véase 16.1.3) son:

- La categoría alternativas (clases derivadas por asignación de clase a partir de **#ALTERNATIVES**).
- La categoría concatenación (clases derivadas por asignación de clase a partir de **#CONCATENATION**).
- La categoría repetición (clases derivadas por asignación de clase a partir de **#REPETITION**).
- La categoría opcionalidad (clases derivadas por asignación de clase a partir de **#OPTIONAL**).
- La categoría rótulo (clases derivadas por asignación de clase a partir de **#TAG**).
- Las categorías booleano, cadena de bits, cadena de caracteres, entero, nulo, identificador de objeto, cadena de octetos, tipo abierto, relleno y real (categorías de clases derivadas de las clases primitivas correspondientes).
- La categoría estructura de codificación (clases generadas a partir de definiciones de tipo ASN.1, o por definición explícita de una estructura de codificación).

9.6.7 Se definen los grupos de categorías siguientes:

- El grupo de categorías campo de bits (clases que corresponden a campos reales en una codificación, por ejemplo, las que están en las categorías entero o booleano, junto con cualquier clase en la categoría estructura de codificación). Las clases de este grupo de categorías se denominan también clases de campo de bits.
- El grupo de categorías constructor de codificación (clases que están en las categorías alternativas, concatenación o repetición). Las clases de este grupo de categorías se denominan también clases de constructor de codificación.
- El grupo de categorías procedimiento de codificación (clases no relacionadas directamente con constructivos ASN.1, y a las que no se les pueden asignar nombres nuevos: **#OUTER**, **#TRANSFORM**, **#CONDITIONAL-INT**, **#CONDITIONAL-REPETITION**). Las clases de este grupo de categorías se denominan también clases de procedimiento de codificación.

9.7 Definición de objetos de codificación

Se dispone de ocho mecanismos para la definición de un objeto de codificación de una clase de codificación dada. No todos ellos se pueden aplicar a todas las clases de codificación.

9.7.1 El primero consiste en especificar el objeto de codificación como igual a algún otro objeto de codificación definido de la clase requerida. Con esto no se hace sino proporcionar un sinónimo de los objetos de codificación.

9.7.2 El segundo, disponible para un conjunto restringido de clases de codificación, consiste en utilizar una sintaxis definida (véase 17.2) para especificar la información que requiere la definición de un objeto de codificación de esa clase. Gran parte de la información necesaria es común a todas las clases de codificación, pero una parte de esa información depende siempre de la clase de codificación específica. (Véase en D.1.1.2 un ejemplo de definición de objeto de codificación de la clase **#BOOLEAN** que contiene codificaciones del tipo ASN.1 booleano).

9.7.3 El tercero, disponible para todas las clases de codificación, consiste en definir un objeto de codificación como la codificación de la clase requerida que está contenida en algún conjunto de objetos de codificación existente. Esto se aplica sobre todo en la denominación de un objeto de codificación de una clase particular que llevarán a cabo las codificaciones BER o PER para esa clase.

NOTA – Lo anterior puede ser útil a menudo, pero requiere el conocimiento de las codificaciones producidas por reglas de codificación normalizadas.

9.7.4 El cuarto consiste en establecer la correspondencia entre valores abstractos asociados a una clase de codificación (digamos, "**#A**") y valores abstractos asociados a otra (normalmente más compleja) clase de codificación (digamos, "**#B**"), y definir un objeto de codificación de "**#B**" (utilizando cualquiera de los mecanismos disponibles). A continuación se puede definir un objeto de codificación de los valores abstractos asociados a "**#A**" como la aplicación del objeto de codificación de "**#B**" a los valores abstractos correspondientes asociados a "**#B**". (Véase en D.2.8.3 un ejemplo). Existen numerosas variantes de todo esto (véase 9.17).

NOTA – Este es el modelo que subyace en la definición de un objeto para la codificación de un tipo entero en BER. Se establece la correspondencia entre el entero y una estructura de codificación que contiene un campo de clase de rótulo (**UNIVERSAL**, **APPLICATION**, **PRIVATE**, o específica del contexto), un booleano primitivo/constructor, un campo de número de rótulo y una parte valor que codifica los valores abstractos del entero original.

9.7.5 El quinto mecanismo consiste en definir un objeto de codificación de una clase (por ejemplo, una correspondiente a un tipo ASN.1 definido por el usuario), definiendo por separado objetos de codificación de los componentes y del constructor de codificación utilizado en la definición de la clase de codificación.

9.7.6 El sexto consiste en definir un objeto de codificación para la codificación-decodificación diferencial (véase 9.8), utilizando dos objetos de codificación separados, uno de los cuales define el comportamiento del codificador y el otro indica a un decodificador la codificación que deberá suponerse.

NOTA – Un ejemplo consistiría en codificar un campo que fuese "reservado para uso futuro" con todo ceros, pero aceptar cualquier valor cuando se decodifique.

9.7.7 El séptimo consiste en definir un objeto de codificación de opciones de codificación, que contiene una lista ordenada de objetos de codificación de la misma clase. El codificador elige el objeto de codificación de la lista que se va a aplicar a reserva de la restricción de que si sólo una opción de codificación puede codificar un valor abstracto dado, ésa será la utilizada, y con la recomendación de que se utilice la primera codificación disponible en la lista.

NOTA – Se podría utilizar un objeto de codificación de opciones de codificación, por ejemplo, en la especificación de codificaciones de longitud de formato acortado cuando puedan codificar una determinada longitud de cadena, utilizando codificaciones de longitud de formato alargado si no se puede utilizar el formato acortado. No hay actualmente ningún mecanismo mediante el cual el especificador de la ECN pueda requerir la utilización del primer objeto de codificación disponible (si es que el valor abstracto lo puede codificar más de uno), aparte de la formulación de comentarios.

9.7.8 Finalmente, un objeto de codificación puede ser definido utilizando una notación distinta de la ECN. Se trata de una facilidad que permite utilizar la notación que se desee (incluido el lenguaje natural) para definir el objeto de codificación (véase D.2.7.3).

NOTA – La notación no ECN deberá utilizarse con cautela, ya que en este caso no es posible por lo general el soporte de herramientas de implementación.

9.8 Codificación-decodificación diferencial

9.8.1 Codificación-decodificación diferencial es la expresión aplicada a una especificación que exige que una implementación acepte (cuando decodifica) otros esquemas de bits además de los que ya se le permite generar cuando codifica.

9.8.2 La codificación-decodificación diferencial se halla en la base del soporte de la "extensibilidad" (la posibilidad de que la implementación de una versión previa de una norma interfuncione convenientemente con la implementación de una versión posterior de esa norma).

9.8.3 La naturaleza precisa de la codificación-decodificación diferencial puede ser muy compleja. Normalmente, incluye el requisito de que un decodificador acepte (e ignore en silencio) campos de relleno (por lo general de longitud variable) que las versiones posteriores de una norma utilizarán para la transferencia de información que se agrega a la transferida en comunicaciones de la versión previa.

9.8.4 El soporte de la codificación-decodificación diferencial en la ECN corre a cargo de la sintaxis que permite la definición de un objeto de codificación (de cualquier clase) que encapsula dos objetos de codificación. Uno y otro definen reglas de codificación. El primer de ellos define las reglas que utiliza un codificador. El decodificador utiliza el segundo objeto de codificación como una especificación del modo en que se efectuó la codificación.

NOTA – En la ECN, las reglas que utiliza un decodificador (según una versión previa de la norma) se expresan siempre dando las reglas de codificación que debe suponer que utiliza su socio comunicante. Las reglas de decodificación no se dan como reglas de decodificación explícitas. El especificador de la ECN garantizará que esas reglas de decodificación permiten cualquier "extensibilidad" necesaria.

9.9 Opciones de los codificadores en las codificaciones

9.9.1 En la actualidad, se considera que deben evitarse por lo general las opciones de los codificadores en los protocolos, pero la ECN ha de sustentar esas opciones si el diseñador de un protocolo decide (o decidió en el pasado) incluirlas.

9.9.2 Cuando se codifican valores en un espacio de codificación, es posible especificar que el tamaño del espacio de codificación (véase 9.21.5) sea una opción del codificador, siempre que haya alguna forma de determinante de longitud asociado a la codificación. (El margen de opcionalidad del codificador puede estar limitado por el valor máximo que se puede codificar en el determinante de longitud). Esto proporciona un nivel detallado del soporte de las opciones del codificador.

9.9.3 Un mecanismo más global es similar al soporte de la codificación-decodificación diferencial (véase 9.8), pero en este caso se puede definir un objeto de codificación de una clase como una elección del codificador de cualquier objeto de codificación de una lista ordenada de objetos de codificación definidos para esa clase. Además de especificar la lista de codificaciones posibles, es necesario proporcionar la especificación de un objeto de codificación de una clase en la categoría alternativas (véase 9.6). Este objeto de codificación especifica las codificaciones y procedimientos que necesita un decodificador para determinar qué objeto de codificación fue utilizado por el codificador.

9.10 Propiedades de objetos de codificación

9.10.1 Los objetos de codificación tienen algunas propiedades de carácter general. En la mayoría de los casos definen por completo una codificación, pero algunas veces son **constructores de codificación**, es decir, sólo definen aspectos estructurales de la codificación, por lo que hace falta que los objetos de codificación de los componentes de la estructura de codificación completen la definición de una codificación.

9.10.2 Otra característica clave de un objeto de codificación es que puede requerir información del entorno en el que sus reglas son aplicadas eventualmente. Un aspecto del entorno sustentado plenamente es la presencia de límites en la definición del tipo ASN.1, siempre que sean visibles a los efectos de las PER (véase 9.3 de la Rec. UIT-T X.691 | ISO/CEI 8825-2).

NOTA – Una dependencia externa algo diferente (y no normalizada) sería la definición de un objeto de codificación no ECN para una clase de codificación **`#ALTERNATIVES`** que determine la alternativa seleccionada de acuerdo con datos externos, tales como el canal por el que se envía el mensaje.

9.10.3 Una tercera característica clave consiste en que el objeto de codificación puede exhibir un **asa de identificación** en sus codificaciones. Esto forma parte de todas la codificaciones que produce y las distingue de las codificaciones de otros objetos de codificación (de cualquier clase) que presenten la misma asa de identificación. Los decodificadores han de poder ver las asas de identificación sin tener conocimiento de la clase de codificación o el valor abstracto que fue codificado (pero sabiendo cuál es el nombre del asa de identificación que se utiliza). Este concepto modela (y generaliza) la utilización de rótulos en las codificaciones BER: el valor del rótulo en BER se puede determinar sin saber cuál es la clase de codificación, para todas las codificaciones BER, y sirve para identificar la codificación a efectos de resolución de la opcionalidad, ordenación de los conjuntos y alternativas de elección.

9.11 Parametrización

9.11.1 Al igual que los tipos y valores ASN.1, se pueden parametrizar los objetos de codificación, los conjuntos de objetos de codificación y las clases de codificación. Se trata simplemente de una ampliación del mecanismo ASN.1 normal.

9.11.2 Se hace una utilización primaria de la parametrización en la definición de un objeto de codificación que necesita la identificación de un determinante para completar la definición de la codificación (véase 9.13.2). (Véase en D.1.11.3 un ejemplo de definición ECN parametrizada).

9.11.3 Otra utilización importante de la parametrización es la definición de una estructura de codificación que será utilizada para reemplazar gran número de clases diferentes en una codificación (véase también 9.16.5). Por ejemplo, el mecanismo utilizado para tratar la opcionalidad es a menudo un "bit de presencia" que precede inmediatamente (de forma obligatoria) a cada componente opcional. Para describir una estructura parametrizada, se puede decir que está

constituida por la concatenación de un componente **#BOOLEAN** (utilizado como determinante de presencia) seguido por un componente opcional definido como un parámetro ficticio (que será instanciado con el componente al que reemplazará la estructura), y cuya presencia está determinada por el componente **#BOOLEAN**. El procedimiento de codificación **#OPTIONAL** original se define ahora como la sustitución del componente original por su estructura obligatoria, utilizando el componente opcional original como el parámetro real. (D.3.2 es un ejemplo más completo de este proceso).

9.11.4 Parámetros ficticios pueden ser los objetos de codificación, conjuntos de objetos de codificación, clases de codificación, referencias a campos de la estructura de codificación y cualesquiera valores de los tipos ASN.1 utilizados en las clases de codificación incorporadas definidas en la cláusula 23, según lo especificado en la Rec. UIT-T X.683 | ISO/CEI 8824-4 modificada por B.10 de la presente Recomendación | Norma Internacional.

9.11.5 La modificación de la sintaxis de parametrización que se especifica en el anexo C requiere la utilización del símbolo "**{<**" (sin espacios) en vez de "**{**" para empezar una lista de parámetros ficticios o reales, y del símbolo "**>}**" para terminarla.

NOTA – Esto se ha hecho para facilitar el análisis de la sintaxis ECN por los ordenadores, y evitar ambigüedades cuando se emplean clases definidas por el usuario en las definiciones de estructuras en lugar de **#SEQUENCE**, **#CHOICE**, **#REPETITION**, **#SEQUENCE-OF** o **#SET-OF**.

9.12 Gobernadores

9.12.1 Los conceptos de gobernador y de notación gobernada serán habituales a partir de la notación de valores ASN.1, en donde siempre hay un tipo de definición que "gobierna" la notación de valores y determina su sintaxis y significado.

9.12.2 El mismo concepto se amplia a la definición de objetos de codificación de una clase de codificación dada. La sintaxis para definir un objeto de codificación de la clase **#BOOLEAN** (por ejemplo) es muy diferente de la sintaxis para definir un objeto de clase **#INTEGER** (por ejemplo). En todos los casos en que se requiere la definición de un objeto de codificación, hay alguna notación asociada que define la clase de ese objeto de codificación y "gobierna" la sintaxis que se ha de utilizar en su especificación.

9.12.3 La sintaxis ECN requiere que los gobernadores que son clases de codificación, sean nombres de referencia o nombres de referencia de clase parametrizada.

9.12.4 Si la notación gobernada es un nombre de referencia de un objeto de codificación, es preciso que el objeto de codificación sea de la misma clase que el gobernador (véase 17.1.7).

9.13 Aspectos generales de las codificaciones

9.13.1 La ECN permite el soporte de un cierto número de técnicas utilizadas normalmente en la definición de reglas de codificación (no sólo las utilizadas en BER o PER). Por ejemplo, reconoce que la opcionalidad se puede resolver de alguna de las tres maneras siguientes: utilizando un determinante de presencia, utilizando un asa de identificación (véase 9.13.3) o alcanzando el final de un contenedor de longitud delimitada (o el final de la PDU) antes de que aparezca el componente opcional.

9.13.2 De manera similar, reconoce que la delimitación de las repeticiones se puede hacer (por ejemplo) mediante:

- Alguna forma de cómputo de la longitud,
- Detectando el final de un contenedor (o PDU) del que es el último elemento.
- La utilización de un asa de identificación en cada una de las repeticiones y en las codificaciones siguientes (véase 9.13.3).
- Algún esquema de terminación que no se puede producir nunca en una codificación de la serie repetida. (Un ejemplo sencillo sería el de una cadena de caracteres terminada en nulo).
- La utilización de un bit "more" (bit "más") con cada elemento, fijado en uno para indicar que sigue otra repetición y fijado en cero para indicar el final de la repetición.

La ECN admite todos estos mecanismos de delimitación de las repeticiones, y admite mecanismos similares para la identificación de alternativas y la resolución de la opcionalidad.

9.13.3 Además de para terminar repeticiones, la técnica del asa de identificación se puede utilizar para determinar la presencia de componentes opcionales o de alternativas. El mecanismo es similar en todos los casos. Las codificaciones de todos los valores de una codificación de "clase siguiente posible" dada tendrán el mismo esquema de bits (su identificación) en algún lugar de su codificación (el asa), pero la identificación de codificaciones de "clase siguiente posible" diferentes será diferente para cada una de ellas. Todas esas codificaciones pueden ser interpretadas por un decodificador como una codificación de cualquier "clase siguiente posible", y la identificación del asa determinará qué codificación de "clase siguiente posible" está presente. El concepto es similar al de la utilización de rótulos para esos mismos fines en las BER. Las asas de identificación tienen nombres que han de ser únicos dentro de una especificación ECN.

9.13.4 Es importante señalar aquí que la ECN permite la definición de codificaciones de manera muy flexible, pero no puede garantizar que una especificación de codificación sea correcta, es decir, que un decodificador pueda recuperar de manera satisfactoria los valores abstractos originales a partir de una codificación. Por ejemplo, un especificador de la ECN podría asignar el mismo esquema de bits para valores booleanos verdadero y falso. Lo cual sería erróneo, y en este caso el error podría ser detectado con bastante facilidad por medio de una herramienta. Otro error sería alegar que una codificación era autodelimitante (y no requería determinante de longitud), cuando de hecho no lo era. Este error también podría ser detectado por medio de una herramienta. En casos más sutiles y complejos, no obstante, es posible que resulte muy difícil diagnosticar con una herramienta una especificación errónea (una especificación que no siempre puede ser decodificada de manera satisfactoria).

9.14 Identificación de elementos de información

9.14.1 Muchos protocolos tienen una codificación (normalmente con un número de bits fijo) para identificar los que a menudo se denominan "elementos de información" o "elementos de datos" de un protocolo. Estas identificaciones corresponden aproximadamente a los rótulos ASN.1, pero en general son menos complejas. Con frecuencia se emplean como asas de identificación, aunque no siempre son utilizadas.

9.14.2 La ECN contiene una clase **`#TAG`** que soporta la definición de la codificación de identificadores de elementos de información utilizando la notación de rótulo ASN.1. (También admite la inclusión de esos elementos dentro de una estructura de codificación sin referencia a los rótulos ASN.1).

9.14.3 Cuando una estructura de codificación es generada implícitamente a partir de una definición de tipo ASN.1 (véase la cláusula 11), la primera notación de rótulo ASN.1 **presente textualmente** en esa definición genera una instancia de la clase **`#TAG`**, con el número del rótulo ASN.1 asociado a esa instancia de la clase **`#TAG`**. Las instancias presentes textualmente de notación de rótulo ASN.1 subsiguientes no se hacen corresponder con clases #TAG en la estructura generada implícitamente, pero esos rótulos y sus valores pasan a ser propiedades del elemento. Una codificación de esta clase de codificación se puede definir de manera similar a una codificación de la clase **`#INTEGER`**, y codificará el número de la notación de rótulo.

9.14.4 La lista completa de rótulos ASN.1 (múltiples rótulos, cada uno con una clase y un número) se asocia conceptualmente a todos los valores abstractos de un tipo rotulado, de acuerdo con el modelo ASN.1. Sin embargo, esa información sólo es accesible en la versión actual de la ECN mediante la definición no ECN de un objeto de codificación (véase 9.7.8). La generación de una clase **`#TAG`** es un mecanismo aparte, más sencillo y específico, y tiene pleno soporte en la ECN.

9.14.5 Es importante señalar no obstante que, a efectos de la generación de una clase **`#TAG`**, sólo es visible la notación de rótulo presente textualmente. Los rótulos de clase universal y los rótulos generados mediante rotulación automática no son visibles. De manera similar, la clase de cualquier notación de rótulo presente textualmente es ignorada. Sólo el número de rótulo está disponible para codificar objetos de la clase #TAG.

9.15 Campos de referencia y determinante

9.15.1 Una manera muy frecuente (pero no la única) de determinar la presencia de un campo opcional, la duración de una repetición o la selección de una alternativa consiste en incluir (en algún lugar del mensaje) un campo de determinante. Si este mecanismo se utiliza a efectos de determinación, los campos de determinante han de ser identificados, lo cual requiere frecuentemente que se suministre un parámetro ficticio de una definición de objeto de codificación con el parámetro real, que proporciona el nombre del campo de la estructura de codificación del determinante, cuando el objeto de codificación se aplica a una estructura de codificación.

9.15.2 Se introduce un concepto nuevo, el de **campo de referencia**, para satisfacer la necesidad de un parámetro ficticio que hace referencia a un campo de estructura de codificación. El gobernador es la palabra reservada "**`REFERENCE`**", y la notación permitida para un parámetro real con este gobernador es cualquier nombre de campo de estructura de codificación dentro de la estructura de codificación a la que se aplica un objeto de codificación o un conjunto de objetos de codificación con ese parámetro (véase 17.5.15). (Véase en D.1.11.3 un ejemplo de referencias a nombres de campos de la estructura de codificación).

9.16 Clases y estructuras de sustitución

9.16.1 Cuando se escriben especificaciones ASN.1 para protocolos de legado (o para generar codificaciones especializadas de nuevos protocolos) es normal pasar por alto asuntos relativos a la codificación, en particular, los campos de determinante que sólo están presentes como soporte de la decodificación. Sólo los campos que tienen importancia en el código de aplicación (que llevan la semántica de la aplicación) se incluyen en la especificación ASN.1.

9.16.2 Cuando esos protocolos utilizan más de un mecanismo de codificación para soportar (por ejemplo) construcciones **SEQUENCE-OF** en diferentes lugares del protocolo, no es posible (ni sería apropiado) especificar esto formalmente dentro de la propia ASN.1.

9.16.3 Lo anterior significa que la estructura de codificación generada implícitamente no distinguirá entre esas construcciones ni contendrá campos relacionados con la codificación de determinantes, y que para disponer de una estructura que satisfaga los requisitos de la codificación es necesario modificarla a fin de "corregir" ambos problemas.

9.16.4 La primera modificación, y la más sencilla, consiste en sustituir algunas instancias de una clase (dentro de la estructura generada implícitamente) por nombres de clase nuevos asignados a clases antiguas en una declaración de asignación de clase. Esto se hace mediante la creación de una **estructura generada explícitamente** utilizando una cláusula de redenominaciones en un EDM. Dicha cláusula importa una estructura generada implícitamente desde un módulo ASN.1 y lleva a cabo las sustituciones especificadas de ocurrencias (textuales) de clases denominadas. La sustitución puede ser de todas las ocurrencias textualmente dentro de una lista de clases generadas implícitamente (correspondientes a las definiciones de tipo ASN.1 en un módulo), o dentro de componentes de una de esas clases, o de "todas las ocurrencias excepto" las de una definición determinada o un componente determinado (véase 15.3). Conviene señalar aquí que esas sustituciones se limitan a la utilización de clases que han sido definidas con una declaración de asignación de clase de codificación en la que se asigna el nombre de una clase de sustitución a una clase antigua (por ejemplo: "**#Replacement-class ::= #Old-class**"), por lo que a veces se dice, de manera coloquial, que este mecanismo es un "colorante". El "colorante" identifica aquellas partes de la especificación que requieren codificaciones diferentes de los de otras partes. (En D.3.7 se da un ejemplo de "colorante").

9.16.5 Incluso con "colorante", la estructura de codificación generada explícitamente, al igual que la estructura de codificación generada implícitamente, sólo contiene campos correspondientes a los campos de la especificación ASN.1, y por lo general es necesario modificar las estructuras generadas para añadir campos de determinante, etc. Se necesita una **estructura de sustitución** nueva (para la totalidad o una parte de la estructura original) con campos añadidos. También es importante identificar (para cada campo de la estructura original) qué campos de la estructura de sustitución (y qué valores abstractos de esos campos) se utilizan para llevar la semántica de los valores abstractos originales. Se habla de establecimiento de la correspondencia entre los valores abstractos de la estructura original y la estructura de sustitución.

9.16.6 Hay muchos mecanismos de definición de un objeto de codificación de una estructura existente como un objeto de codificación de una estructura de sustitución totalmente diferente, con **correspondencias de valores** definidas entre la estructura antigua y la estructura de sustitución. Estos mecanismos se describen en 9.17.

9.16.7 Con frecuencia se produce, no obstante, una situación más sencilla en la que el diseñador exige a la estructura antigua que constituya (en su totalidad) un componente único de la estructura de sustitución, estableciéndose la correspondencia entre todos los valores abstractos de la estructura antigua y los valores correspondientes de ese componente de la estructura de sustitución. Para que este mecanismo sea de uso general, la estructura de sustitución ha de tener un parámetro ficticio para este componente único, que sea instanciado con el conjunto de parámetros reales de la estructura antigua. Esto se describió en 9.11.3.

9.16.8 Cuando se definen objetos de codificación de una clase (cualquiera que sea), siempre es posible especificar que la primera acción del objeto de codificación ha de consistir en sustituir la clase que está codificando por una estructura de sustitución parametrizada, instanciada como se describe en 9.16.7, y con valores abstractos cuya correspondencia se ha establecido entre la clase antigua y el componente.

9.16.9 También es posible definir objetos de codificación de la clase **#OPTIONAL** (o de cualquier clase en la categoría opcionalidad) que sustituyan el componente opcional por una estructura de sustitución parametrizada (con frecuencia una estructura que contiene un campo **#BOOLEAN** como determinante de presencia). En D.3.2.3 se da un ejemplo de esto.

9.16.10 Para clases de constructor tales como **#CONCATENATION**, **#REPETITION**, etc., es posible también definir objetos de codificación que reemplazan no ya la estructura entera sino cada uno de los componentes por separado (o bien los componentes obligatorios o los componentes opcionales).

9.16.11 Un mecanismo más avanzado, y potente, consiste en exigir que la acción de sustitución incluya además la inserción de un campo especificado en el encabezamiento de una **#CONCATENATION** (o estructura similar). En D.3.1.5 se da un ejemplo al respecto.

9.17 Establecimiento de la correspondencia entre valores abstractos y campos de estructura de codificación

Se dispone de seis mecanismos a tal fin.

9.17.1 El primero consiste en establecer la correspondencia entre valores abstractos especificados asociados a una clase de codificación sencilla y valores abstractos especificados asociados a otra clase de codificación sencilla. Esto se puede hacer de muchas maneras. Por ejemplo, estableciendo la correspondencia entre los valores de una cadena de caracteres (de dígitos) y los valores enteros (y, por tanto, codificados como valores enteros). Se puede establecer la correspondencia entre valores de un tipo enumerado y valores enteros, y así sucesivamente (véase 19.2). (Véase un ejemplo en D.1.10.2).

9.17.2 El segundo mecanismo consiste en establecer la correspondencia entre un campo completo de una estructura de codificación y un campo de una estructura de codificación compatible, que pueda contener campos adicionales utilizados normalmente como determinantes de longitud o elección (véase 19.3). (Véase un ejemplo D.2.8.3).

9.17.3 El tercero consiste en establecer la correspondencia transformando todos los valores abstractos asociados a una clase de codificación en valores abstractos asociados a una clase de codificación diferente (por lo general, pero no necesariamente) utilizando un objeto de codificación transformada (véase 9.18). Con este mecanismo es posible, por ejemplo, establecer la correspondencia entre una **#INTEGER** y una **#CHARS** para obtener caracteres que pueden ser codificados a continuación de la manera que se desee (por ejemplo, decimal codificado en binario o ASCII). (Véase un ejemplo en D.1.6.3).

9.17.4 El cuarto mecanismo consiste en utilizar un orden definido de los valores abstractos de determinados tipos y construcciones, estableciendo la correspondencia de acuerdo con ese orden. Se dispone así de un medio muy poderoso de codificación de valores abstractos asociados a una clase de codificación como si fuesen valores asociados a una clase de codificación totalmente desvinculada (véase 19.5). (Véase un ejemplo en D.1.4.2).

9.17.5 El quinto mecanismo consiste en distribuir los valores abstractos (utilizando la notación de la gama de valores) asociados a una clase de codificación (normalmente **#INTEGER**) en los campos de otra clase de codificación. (Véanse ejemplos al respecto en 19.6 y D.2.1.3).

9.17.6 El mecanismo final permite al especificador de la ECN proporcionar una correspondencia explícita entre valores enteros (que pueden haber sido producidos mediante establecimientos de la correspondencia previos a partir, por ejemplo, de una clase **#ENUMERATED**) y los bits que se han de utilizar para codificar esos valores. Se pretende con esto sustentar las codificaciones Huffman, cuando se sabe cuál es (al menos aproximadamente) la frecuencia de ocurrencia de cada valor y se requiere la codificación óptima. El anexo E describe las codificaciones Huffman con más detalle, y da ejemplos de este mecanismo, junto con una referencia al soporte lógico (software) que generará la sintaxis ECN de estos establecimientos de la correspondencias, conociendo tan sólo la frecuencia relativa con que se prevé que se utilice cada valor del entero (véase 19.7.).

9.18 Transformadas y compuestos de transformadas

9.18.1 Las transformadas son objetos de codificación de la clase **#TRANSFORM**. Se pueden utilizar para transformar valores abstractos entre clases de codificación diferentes, y también para definir funciones aritméticas sencillas tales como la multiplicación por un valor fijo, la sustracción de un valor fijo, y así sucesivamente. Cuando se aplican de manera sucesiva, permiten especificar operaciones aritméticas de carácter general (véase 19.4). (Véase un ejemplo en D.2.4.2).

9.18.2 Una transformada puede tomar un valor único como su origen y producir a continuación un valor único como su resultado. Lo que sigue es una clasificación de los valores que pueden ser orígenes y resultados de las transformadas:

- un entero
- un booleano
- una cadena de caracteres
- una cadena de bits
- un carácter único
- un bit único (origen solamente, soportando la codificación de una cadena de bits; véase 23.2).

9.18.3 Los compuestos de transformadas son una lista ordenada de elementos, cada uno de los cuales es un valor único y tiene la misma clasificación (indicada en 9.18.2). (Por ejemplo, una lista ordenada de caracteres únicos, o de octetos únicos, o de enteros). Se producen solamente como resultado de las transformadas y sólo se pueden utilizar como origen de una transformada siguiente.

9.18.4 Si la clasificación es cadena de bits, el tamaño de cada valor cadena de bits del compuesto es el mismo, y está determinado estáticamente por la transformada que produce el compuesto. (Por ejemplo, una lista ordenada de bits únicos, o unidades de seis bits).

9.18.5 Hay transformadas de los siguientes valores abstractos a compuestos:

- cadena de caracteres a un compuesto de caracteres únicos;
- cadena de bits a un compuesto de cadenas de bits (todos los valores cadena de bits del compuesto tienen el mismo tamaño);
- cadena de octetos a un compuesto de cadenas de bits (todos los valores cadena de bits del compuesto tienen un tamaño de 8 bits).

9.18.6 Hay transformadas de los siguientes compuestos a valores abstractos:

- compuestos de caracteres únicos a valores cadena de caracteres;
- compuestos de cadenas de bits a valores cadena de bits;
- compuestos de cadenas de bits (con valores cadena de bits de 8 bits de tamaño) a valores cadena de octetos.

9.18.7 Todas las demás transformadas pueden tomar un valor como su origen y producir un valor nuevo (de la misma clasificación o de una clasificación diferente). También pueden tomar un compuesto de transformadas como su origen y producir un compuesto como su resultado, transformando cada elemento del compuesto origen en un elemento del compuesto resultado.

9.19 Contenido de los módulos de definición de codificación

9.19.1 Los módulos de definición de codificación (los EDM) contienen declaraciones de exportación e importación exactamente igual que la ASN.1 (sólo pueden importar objetos de codificación, conjuntos de objetos de codificación y clases de codificación de otros módulos EDM, o de módulos ASN.1 en el caso de estructuras de codificación generadas implícitamente).

9.19.2 Un EDM puede contener también una cláusula de redenominaciones (véase la cláusula 15) que hace referencia a estructuras de codificación generadas implícitamente a partir de uno o más módulos ASN.1 y genera, "coloreándolos" (véase 9.16.4), una estructura de codificación generada explícitamente para cada uno de ellos. Estas estructuras de codificación generadas explícitamente están disponibles para su utilización dentro del EDM, pero también son exportadas automáticamente para su posible importación en el módulo de enlace de codificación.

9.19.3 El cuerpo de un módulo EDM contiene:

- declaraciones "EncodingObjectAssignment" que definen y denominan un objeto de codificación de alguna clase de codificación (hay siete formas de esta declaración, analizadas en 9.7 y definidas en la cláusula 17);
- declaraciones "EncodingObjectSetAssignment" que definen conjuntos de objetos de codificación (véase la cláusula 17);
- declaraciones "EncodingClassAssignment" que definen y denominan clases de codificación nuevas (véase la cláusula 15).

9.19.4 El EDM puede contener también versiones parametrizadas de estas declaraciones, según se especifica en las cláusulas 14 y C.1.

9.19.5 Se pueden definir objetos de codificación de clases de codificación incorporadas dentro de cualquier módulo EDM. Se pueden definir objetos de codificación de una estructura de codificación generada solamente en módulos EDM que importan la estructura de codificación generada implícitamente desde el módulo ASN.1 que define el tipo correspondiente (utilizando una cláusula de importaciones o de redenominaciones), o que importan la estructura de codificación generada desde el módulo EDM que la ha exportado.

NOTA – Si ocurre que una estructura de codificación generada implícitamente tiene un nombre que es el mismo que el de una clase de codificación reservada (véase 8.5), todavía puede ser importada en un EDM, pero ha de ser referenciada en el cuerpo del EDM utilizando un nombre totalmente cualificado (véase "ExternalEncodingClassReference" en 10.6).

9.20 Contenido del módulo de enlace de codificación

9.20.1 Todas las aplicaciones de la notación de control de codificación requieren la identificación de un solo módulo de enlace de codificación (o ELM).

9.20.2 El ELM aplica conjuntos de objetos de codificación a tipos ASN.1 (formalmente, a una estructura de codificación generada correspondiente al tipo ASN.1). Estos conjuntos de objetos de codificación (o sus objetos de codificación constituyentes) son importados en el módulo ELM desde uno o más módulos EDM.

9.20.3 Hay limitaciones impuestas a la aplicación de los conjuntos de objetos de codificación para asegurar que no existe ambigüedad alguna a propósito de las reglas de codificación que se aplican realmente (véase 12.2.5). Por ejemplo, no se permite que un ELM aplique más de un conjunto de objetos de codificación a una estructura específica generada implícitamente.

9.20.4 En casos sencillos es posible que un módulo ELM contenga tan sólo una declaración (tras una cláusula de importaciones) que aplica un conjunto de objetos de codificación a la estructura de codificación generada implícitamente correspondiente al único tipo de nivel máximo de una aplicación. (Véase un ejemplo en D.1.17).

9.21 Definición de codificaciones de clases de codificación primitivas

9.21.1 Las reglas de codificación de algunas clases de codificación primitivas se pueden definir utilizando una sintaxis fácil para el usuario que se especifica en las declaraciones **`WITH SYNTAX`** de las definiciones de clase de codificación (véanse las cláusulas 23 y 25). Esta sintaxis se puede utilizar también para definir las reglas de codificación de las clases de codificación derivadas de esas clases de codificación primitivas (codificando declaraciones de asignación de clase).

9.21.2 La notación utilizada para las definiciones de clases de codificación en las cláusulas 23 y 25 se basa en la notación utilizada para la definición de clases de objetos de información. Esta sintaxis (y la semántica asociada a la misma) se define por referencia a la Rec. UIT-T X.681 | ISO/CEI 8824-2 modificada por el anexo B a esta Recomendación | Norma Internacional.

9.21.3 La definición de clases de codificación especifica la información que se ha de suministrar para definir las reglas de codificación de clases de codificación particulares. El conjunto de reglas de codificación que se puede definir de esta manera no representa, naturalmente, la totalidad de las reglas posibles, pero se considera que abarca las especificaciones de codificación que probablemente requieran los usuarios de la ECN.

9.21.4 Estas definiciones de clases de codificación especifican una serie de campos (con los tipos y las semánticas ASN.1 correspondientes). Las reglas de codificación se especifican dando valores a esos campos. Los valores de esos campos proporcionan de manera efectiva los valores de una serie de propiedades de codificación que definen colectivamente una codificación.

9.21.5 El significado de las propiedades de codificación se especifica utilizando un modelo de codificación (véase la figura 1) en el que el valor de cada clase de campo de bits produce una **codificación de valor** que se coloca (justificada a la izquierda o a la derecha) en un **espacio de codificación**.

9.21.6 Se puede hacer que el espacio de codificación tenga su borde delantero alineado con alguna frontera (por ejemplo, una frontera de octeto) mediante el relleno previo del espacio de codificación, y su tamaño puede ser fijo o variable. La codificación del valor se adapta dentro de él, quizá justificado a la izquierda o a la derecha, y con el relleno a su alrededor. Si el tamaño del espacio de codificación es variable, la codificación del valor ha de ser autodelimitante o ha de haber algún mecanismo externo que permita a un decodificador determinar el tamaño del espacio de codificación. Se dispone de varios mecanismos para esa determinación.

9.21.7 Finalmente, el espacio de codificación completo con la codificación del valor y cualquier relleno previo al valor o posterior al mismo, se hace corresponder con los bits en la línea mediante una especificación opcional de **inversión de bits**. De esta manera se tratan las codificaciones que exigen que "el byte más significativo sea el primero" o que "el byte más significativo sea el último" o que los bits de un octeto estén en orden inverso al normal.

9.21.8 Así pues, hay tres amplias categorías de información necesaria:

- la primera se refiere al espacio de codificación en el que se sitúa la codificación;
- la segunda se refiere a la manera en que se establece la correspondencia entre un valor abstracto y los bits (codificación del valor), y el posicionamiento de esos bits dentro del espacio de codificación; y
- la tercera se refiera a cualesquiera inversiones de bits que se requieran.

9.21.9 La figura 1 muestra el espacio de codificación (con relleno antes del valor) y la codificación del valor (con relleno después del valor). Dicha figura ilustra además la especificación de una unidad de espacio de codificación. El espacio de codificación es siempre un múltiplo entero del número de bits especificado

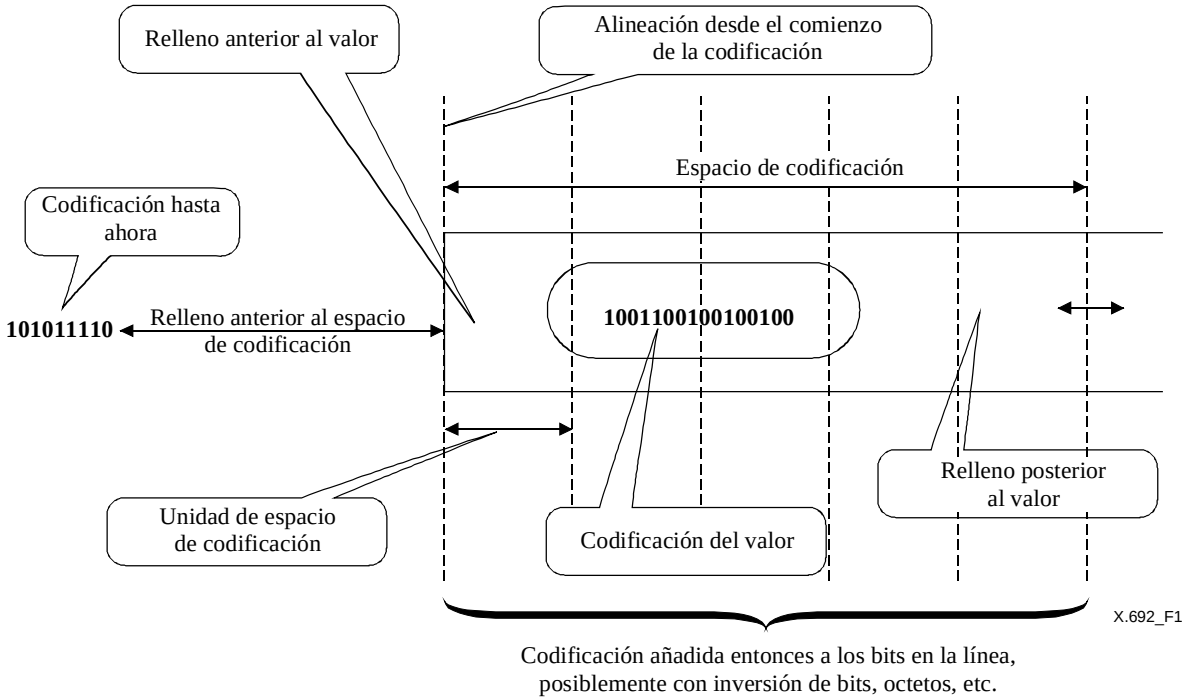

Figura 1 – Conceptos relativos a espacio de codificación, codificación de valor y relleno

9.21.10 Si el espacio de codificación no es del mismo tamaño para todos los valores codificados por un objeto de codificación, se necesita algún mecanismo adicional de determinación del espacio de codificación real utilizado en una instancia de una codificación.

9.21.11 También es posible especificar una cantidad cualquiera de relleno previo del codificador (superior a la que se necesita para la alineación) que termina cuando el valor de un **puntero de comienzo** anterior identifica el comienzo de un campo.

9.21.12 Los pasos para definir la codificación de una clase de codificación de campo de bits primitiva son como sigue:

- Especificación de la alineación requerida (si se necesita) del borde delantero del espacio de codificación (con respecto al **punto de alineación**, normalmente el comienzo de la codificación del tipo de nivel máximo, es decir, el tipo al que se aplica un objeto de codificación en el ELM). (Véase 22.2).
- Especificación de la forma de cualquier relleno necesario hasta ese puntero (relleno previo del espacio de codificación). (Véase 22.2).
- Especificación (si se necesita) de un campo que proporciona un puntero al punto de comienzo del espacio de codificación. (Véase 22.3).
- Especificación de la codificación de valores abstractos en bits (codificación de valor).
- Especificación de las unidades del espacio de codificación (el espacio de codificación será siempre un múltiplo entero de esas unidades). (Véase 22.4).
- Especificación del tamaño del espacio de codificación en esas unidades. Puede ser fijo (utilizando el conocimiento de los límites de enteros o de tamaño asociados a los valores abstractos que se han de codificar) o variable (diferente para cada valor abstracto). La especificación puede también (en todos los casos) especificar el empleo de un determinante de longitud que ha de ser codificado con la longitud del campo, y permite decodificar o proporciona información redundante (en el caso de un espacio de codificación de tamaño fijo) que un decodificador puede comprobar. (Véase 22.4).
- Especificación de la alineación de la codificación del valor dentro del espacio de codificación. (Véase 22.8).
- Especificación de la forma de cualquier relleno necesario desde el comienzo del espacio de codificación hasta el comienzo de la codificación del valor (relleno anterior al valor). (Véase 22.8).
- Especificación de la forma de cualquier relleno necesario entre el final de la codificación del valor y el final del espacio de codificación (relleno posterior al valor). (Véase 22.8).
- Especificación de cualesquiera inversiones necesarias de bits del contenido del espacio de codificación antes de añadir bits a la codificación efectuada hasta ahora. (Véase 22.12).

9.21.13 Se dispone de propiedades de codificación con las que sustentar la especificación de las reglas de codificación de todos esos pasos.

9.21.14 En los casos reales, sólo algunas (¡o ninguna!) de esas propiedades de codificación tendrán valores inusuales, y la adopción de valores por defecto actúa como si no estuvieran especificados. (Véase en D.13 un ejemplo de definición de la codificación de un entero que está alineado a la derecha en un campo fijo de dos octetos, empezando en una frontera de octeto).

9.22 Aplicación de codificaciones

9.22.1 La aplicación de codificaciones (reglas de codificación) a las estructuras de codificación es una parte fundamental del trabajo ECN, pero difiere mucho de la definición de las reglas de codificación. La aplicación final de codificaciones (a una estructura de codificación generada a partir de una definición de tipo ASN.1) sólo se produce dentro de un módulo de enlace de codificación, pero se puede utilizar la aplicación de codificaciones a campos de una estructura de codificación en la definición de codificaciones de una estructura de codificación mayor.

9.22.2 Las codificaciones se aplican por referencia a un conjunto de objetos de codificación (o a un solo objeto de codificación). Esa aplicación puede ocurrir en un EDM al definir objetos de codificación de cualquier clase (incluidos los objetos de codificación de una estructura de codificación generada y de una estructura de codificación definida por el usuario). La aplicación en un EDM consiste simplemente en la definición de más objetos de codificación de esa clase de codificación. La aplicación definitiva a un tipo real sólo se produce en el ELM.

9.22.3 Cuando se aplica un conjunto de objetos de codificación, el resultado es siempre una especificación de codificación completa de las clases de codificación a las que se aplican los objetos. Si en cualquier aplicación dada se necesitan codificaciones de clases de codificación (presentes dentro de una estructura de codificación que se codifica) para las que no hay objetos de codificación en el conjunto que se aplica, se trata de un error (véase 13.2.11).

NOTA – Aunque se complete la especificación de las reglas de codificación, la forma precisa de la codificación real (por ejemplo, la presencia o ausencia de relleno antes del espacio de codificación, o el efecto de los valores de límites referenciados en las reglas de codificación) sólo se puede determinar cuando la definición de la codificación se aplica al tipo ASN.1 de nivel máximo.

9.22.4 Hay dos excepciones a 9.22.3. La primera se produce cuando se utiliza el mecanismo de parametrización (como el de la ASN.1) para definir un objeto de codificación parametrizado. En tal caso, la codificación completa sólo se define tras la instanciación con parámetros reales. La segunda excepción ocurre cuando se define un objeto de codificación de un constructor de codificación (**`#CONCATENATION`**, **`#ALTERNATIVES`**, **`#REPETITION`**, **`#SEQUENCE`**, etc.). En este último caso, las reglas de codificación asociadas a la clase de codificación definen simplemente las reglas asociadas a los aspectos estructuradores. Una especificación de codificación completa de una estructura de codificación que utilice esas clases de codificación requerirá además reglas para codificar los componentes de esa estructura de codificación.

NOTA – Hay una diferencia entre objetos de codificación de la clase **`#SEQUENCE`** (un constructor de codificación) y objetos de codificación de una estructura de codificación generada implícitamente "**`#My-Type`**" (que se ha de definir utilizando el tipo ASN.1 "**`SEQUENCE`**"). Esto último no es un constructor de codificación y los objetos de codificación de esta clase proporcionarán reglas de codificación completa de valores del tipo "**`My-Type`**".

9.23 Conjunto de objetos de codificación combinados

9.23.1 Para proporcionar una codificación completa, el usuario de la ECN tiene que suministrar un primer conjunto de objetos de codificación y un segundo conjunto de objetos de codificación introducido por las palabras reservadas **`COMPLETED BY`**.

9.23.2 El conjunto de objetos de codificación aplicado se define como el **conjunto de objetos de codificación combinados** formado añadiendo al primer conjunto objetos de codificación de cualquier clase de codificación para la que el primer conjunto carezca de un objeto de codificación y el segundo conjunto contenga uno (véase 13.2). Un conjunto que se utiliza frecuentemente con **`COMPLETED BY`** es el conjunto incorporado **`PER-BASIC-UNALIGNED`**. (Véase en D.1.17 un ejemplo de la aplicación de un conjunto de objetos de codificación combinados).

9.23.3 Aunque un conjunto de objetos de codificación sólo puede contener un objeto de codificación para una clase **`#SEQUENCE-OF`** (por ejemplo), también puede contener un conjunto de objetos de codificación para una clase **`#Special-sequence-of`** (por ejemplo), que se define como "**`#Special-sequence-of ::= #SEQUENCE-OF`**". Una estructura de codificación generada explícitamente puede tener tanto la clase **`#SEQUENCE-OF`** como la clase **`#Special-sequence-of`** en su definición. De este modo, se puede aplicar un conjunto único de objetos de codificación combinados para producir codificaciones normalizadas para algunos de los constructivos **`SEQUENCE OF`** originales y codificaciones especializadas para otros.

9.24 Punto de aplicación

9.24.1 En cualquier aplicación de codificaciones dada hay un punto de comienzo definido (para el ELM, se trata de la estructura o las estructuras de codificación generadas de nivel máximo a las que se aplican codificaciones). En el caso de la estructura codificada por el ELM se denomina "punto de aplicación inicial".

9.24.2 El conjunto de objetos de codificación combinados se aplica a una estructura de codificación generada, y las codificaciones definidas para los valores abstractos de esa estructura de codificación son las que codifican los valores abstractos del tipo ASN.1.

9.24.3 Si hay un objeto de codificación en el conjunto de objetos de codificación combinados que concuerda con una clase de codificación de campo de bits (inicialmente, una estructura de codificación generada) en el punto de aplicación, se aplica y termina el proceso. De no ser así, la clase en el punto de aplicación es "expandida" por desreferenciación. La expansión continuará hasta que se encuentre un objeto de codificación o se alcance una clase primitiva. Si la clase en el punto de aplicación es un constructor de codificación, y hay un objeto de codificación de ese constructor de codificación (**`#CHOICE, #SEQUENCE, #SEQUENCE-OF`**, etc.), se aplica y el punto de aplicación pasa entonces a cada componente (como una actividad paralela).

9.24.4 En un caso más complejo puede haber una clase **`#OPTIONAL`** tras una clase componente (y una clase **`#TAG`** precediéndola). El punto de aplicación pasa primero a la **`#OPTIONAL`**, y el objeto de codificación de esa clase puede reubicar el componente (véase 9.16.9). El punto de aplicación pasa entonces al rótulo, y por último al propio componente.

9.25 Codificaciones condicionales

9.25.1 Ya se ha hecho mención de la clase de codificación **`#TRANSFORM`** como una manera de efectuar operaciones aritméticas sencillas con valores enteros (véase 9.17.3). Esta clase de codificación desempeña, sin embargo, un papel fundamental en la especificación de codificaciones de algunas clases primitivas. Por lo general, la especificación de codificaciones de muchos de los tipos ASN.1 incorporados es un proceso de dos o tres etapas, en el que se utilizan objetos de codificación de la clase **`#TRANSFORM`** y (por ejemplo) de clase **`#CONDITIONAL-INT`** o **`#CONDITIONAL-REPETITION`**.

9.25.2 Las clases de codificación **`#TRANSFORM`**, **`#CONDITIONAL-INT`** y **`#CONDITIONAL-REPETITION`** son de uso limitado. Para estas clases sólo se pueden definir objetos de codificación utilizando la sintaxis de la cláusula 24, 23.7 y 23.13, respectivamente, o mediante la definición no ECN de un objeto de codificación, y sólo se pueden utilizar a continuación en la definición de otros objetos de codificación. No pueden aparecer en conjuntos de objetos de codificación ni se pueden aplicar directamente a campos de codificación de estructuras de codificación (véase 18.1.7).

9.25.3 La especificación de la codificación de clases de codificación en la categoría entero procede como sigue: se definen las codificaciones (de la clase de codificación **`#CONDITIONAL-INT`**) para una determinada **condición de límites**, especificando el tamaño del contenedor (y cómo está delimitado), la transformada del entero en bits (utilizando codificaciones de complemento de dos o de entero positivo), y la manera en que estos bits se acomodan en el contenedor. (Un ejemplo de condición de límites es la existencia de un límite superior y de un límite inferior no negativo). Esto se denomina **codificación condicional**. La codificación de la clase en la categoría entero se define como una lista de esas codificaciones condicionales, siendo la codificación realmente aplicada en cualquier circunstancia dada la primera de la lista cuya condición de límites se cumple. (Véase un ejemplo en D.1.5.4).

9.25.4 La especificación de la codificación de clases de codificación en la categoría repetición utiliza la clase de codificación **`#CONDITIONAL-REPETITION`**, que define la manera de delimitar el espacio de codificación de los elementos repetidos y cómo se han de colocar las codificaciones repetidas dentro de él, para una **condición de gama** dada, produciendo de nuevo una codificación condicional. Al igual que la codificación de clases en la categoría entero, la codificación final se define como una lista ordenada de codificaciones condicionales.

9.25.5 La especificación de la codificación de clases de codificación en la categoría cadena de octetos procede como sigue: primero se definen los objetos de codificación **`#TRANSFORM`** para establecer la correspondencia entre un octeto único y una cadena de bits autodelimitante. A continuación se definen uno o más objetos de codificación **`#CONDITIONAL-REPETITION`** (con condiciones de gama de tamaños específicas) para tomar cada una de las cadenas de bits (transformadas a partir de un octeto de la cadena de octetos) y concatenarlas en un contenedor delimitado (la definición de esos objetos de codificación no es específica de la clase **`#OCTETS`** de codificación). La codificación final de la clase en la categoría cadena de octetos se define como una lista ordenada de objetos de codificación **`#CONDITIONAL-REPETITION`**. (Véase un ejemplo en D.1.8.2).

9.25.6 La especificación de la codificación de clases de codificación en la categoría cadena de bits procede como sigue: primero se definen los objetos de codificación **`#TRANSFORM`** para establecer la correspondencia entre un bit único y una cadena de bits, de manera similar a la codificación de un entero en bits, pero en este caso la correspondencia del bit debe establecerse con una cadena autodelimitante. A continuación se definen uno o más objetos de codificación

#CONDITIONAL-REPETITION para la repetición de los bits. (Estos podrían ser los mismos objetos de codificación que se definieron para utilizarlos con una clase de codificación en las categorías repetición o cadena de octetos). Por último, se define la codificación de la clase en la categoría cadena de bits como una lista ordenada de objetos de codificación **#CONDITIONAL-REPETITION**. (Véase un ejemplo en D.1.7.3).

9.25.7 La especificación de la codificación de clases de codificación en la categoría cadena de caracteres procede como sigue: primero se definen los objetos de codificación **#TRANSFORM** para establecer la correspondencia entre un carácter único y una cadena de bits autodelimitante, utilizando varios mecanismos posibles para definir la codificación del carácter, y aplicando las constricciones de alfabeto permitido efectivo donde estén disponibles. A continuación se definen uno o más objetos de codificación **#CONDITIONAL-REPETITION** y por último se define la codificación de la clase en la categoría cadena de caracteres como una lista ordenada de aquellos. (Véase un ejemplo en D.1.9.2).

9.26 Cambios en las Recomendaciones | Normas Internacionales relativas a la ASN.1

9.26.1 Esta Recomendación | Norma Internacional hace referencia a otras Recomendaciones | Normas Internacionales relativas a la ASN.1 para definir su notación sin repetición. Para que esas referencias sean correctas, la semántica de la notación (por ejemplo, la cláusula importaciones, la parametrización y la definición de objetos de información) ha de ser ampliada de modo que se reconozcan los nombres de referencia de las clases de codificación, los objetos de codificación, etc., que forman parte de la ECN.

9.26.2 También es preciso ampliar la notación de las clases de objetos de información para hacer posibles campos que sean listas ordenadas de valores u objetos, no simplemente conjuntos de objetos desordenados, a fin de poder utilizar esa notación en la definición de la sintaxis ECN para definir objetos de codificación de determinadas clases.

9.26.3 Por último, se hacen menos estrictas las reglas de parametrización para posibilitar la utilización de un parámetro ficticio de una referencia de objeto de codificación (asignado en una declaración de asignación) como un parámetro real de referencia de la clase de codificación que gobierna la notación definidora del nombre de referencia del objeto de codificación. Se puede utilizar, en particular, una clase de codificación parametrizada como gobernador en una declaración de asignación de objeto de codificación (véase C.2/8.4), siendo el parámetro real un parámetro ficticio del objeto de codificación que se define.

9.26.4 Estas modificaciones de otras Recomendaciones | Normas Internacionales relativas a la ASN.1 se especifican en los anexos A a C, y se señalan únicamente a efectos de la presente Recomendación | Norma Internacional.

10 Identificación de clases de codificación, objetos de codificación y conjuntos de objetos de codificación

10.1 Muchas de las producciones de esta Recomendación | Norma Internacional requieren que se identifique una clase de codificación, un objeto de codificación o un conjunto de objetos de codificación.

10.2 Para cada uno de los anteriores hay cinco maneras de efectuar la identificación:

a) Utilizando un nombre de referencia simple.

b) Utilizando un nombre de referencia incorporado (no aplicable en el caso de objetos de codificación, ya que no hay objetos de codificación incorporados).

c) Utilizando una referencia externa (llamada también nombre totalmente cualificado).

d) Utilizando una referencia parametrizada.

e) Mediante una definición en línea.

NOTA – La referencia parametrizada se puede utilizar con un nombre de referencia simple o con una referencia externa (véase C.3).

10.3 Hay producciones (o elementos de léxico) para todas estas formas de identificación. También hay producciones que permiten varias alternativas. Estos elementos de léxico o nombres de producción se utilizan, cuando así procede, en otras producciones y se definen en el resto de la presente cláusula.

10.4 Los elementos de léxico para utilizar un nombre de referencia simple son:

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

10.4.1 Una "encodingclassreference" es un nombre:

a) al que se ha asignado una clase de codificación en una "EncodingClassAssignment" (véase la cláusula 16); o

b) que ha sido importado en un EDM desde algún otro EDM que lo ha exportado; o

c) que ha sido importado como el nombre de una estructura de codificación generada implícitamente desde un módulo ASN.1 (véase 14.11); o

d) que ha sido generado por una cláusula de redenominaciones en el EDM (véase la cláusula 15).

NOTA – En un ELM sólo se pueden importar clases que son estructuras de codificación generadas (véase 12.1.8).

10.4.2 Un nombre "encodingclassreference" no será importado desde un módulo EDM (como se especifica en 10.4.1) a menos que:

a) esté definido o haya sido importado en el módulo referenciado, y ese módulo no tenga cláusula de exportaciones; o

NOTA 1 – Si el módulo referenciado no tiene cláusula de exportaciones, ello equivale a exportar todo.

b) esté definido o haya sido importado en el módulo referenciado, y aparezca como un símbolo en la cláusula de exportaciones de ese módulo; o

c) sea uno de los nombres de referencia generados explícitamente por una cláusula de redenominaciones en el módulo desde el que está siendo importado.

NOTA 2 – Las estructuras de codificación generadas implícitamente sólo pueden ser importadas desde el módulo ASN.1 que las genera.

10.4.3 Una referencia de estructura de codificación generada implícitamente nunca aparece en la cláusula de exportaciones de un módulo ASN.1, pero siempre puede ser importada desde cualquier módulo ASN.1 en el que se define y desde el que se exporta el tipo correspondiente.

10.4.4 Una referencia de estructura de codificación generada explícitamente (que es exportada automáticamente por la cláusula de redenominaciones que la genera) no aparecerá en la cláusula de exportaciones del módulo EDM en el que es generada, pero cualquier utilización de la misma en otro EDM o el ELM requiere su importación desde ese módulo EDM.

10.4.5 Una "encodingobjectreference" es un nombre:

a) al que se ha asignado un objeto de codificación con una "EncodingObjectAssignment" (véase la cláusula 17) de un EDM; o

b) que ha sido importado en un EDM o ELM desde algún otro EDM en el que se le ha asignado un objeto de codificación o ha sido importado.

10.4.6 Un nombre "encodingobjectreference" no será importado desde un EDM si el módulo referenciado tiene una cláusula de exportaciones y la "encodingobjectreference" no aparece como un símbolo en esa cláusula de exportaciones.

NOTA – Si el módulo referenciado no tiene cláusula de exportaciones, ello equivale a exportar todo.

10.4.7 Una "encodingobjectsetreference" es un nombre:

a) al que se ha asignado un conjunto de objetos de codificación con una "EncodingObjectSetAssignment" (véase la cláusula 18) en un EDM; o

b) que ha sido importado en un EDM o ELM desde algún otro EDM en el que se le ha asignado un conjunto de objetos de codificación o ha sido importado.

10.4.8 Un nombre "encodingobjectsetreference" no será importado desde un EDM si el módulo referenciado tiene una cláusula de exportaciones y la "encodingobjectsetreference" no aparece como un símbolo en esa cláusula de exportaciones.

NOTA – Si el módulo referenciado no tiene cláusula de exportaciones, ello equivale a exportar todo.

10.5 Las producciones para utilizar un nombre de referencia incorporado son:

encoding class	**"BuiltinEncodingClassReference"** (véase 16.1.6)
encoding object set	**"BuiltinEncodingObjectSetReference"** (véase18.2.1.)

10.6 Las producciones para utilizar un nombre de referencia externo son:

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

ExternalEncodingObjectReference ::=
modulereference "." encodingobjectreference

ExternalEncodingObjectSetReference ::=
modulereference "." encodingobjectsetreference

10.6.1 La producción "modulereference" se define en 11.5 de la Rec. UIT-T X.680 | ISO/CEI 8824-1 e identifica un módulo al que se hace referencia en la lista de importaciones del EDM o el ELM.

10.6.2 La producción "ExternalEncodingClassReference" alternativa que incluye una "BuiltinEncodingClassReference" será utilizada en el cuerpo de un EDM si, y solamente si, hay una estructura de codificación generada (cuyo nombre es el mismo que el de una "BuiltinEncodingClassReference") que es:

a) definida implícitamente en el módulo ASN.1 referenciado por la "modulereference" (véase 11.4.1); o

b) importada en otro EDM referenciado por la "modulereference" y exportada desde ese módulo; o

c) generada en una cláusula de redenominaciones de otro EDM referenciado por la "modulereference"; o

d) generada en este EDM en una cláusula de redenominaciones, en cuyo caso la "modulereference" hará referencia a este EDM.

NOTA – El nombre "BuiltinEncodingClassReference" puede aparecer como un "Symbol" en la cláusula de importaciones (véase A.1).

10.6.3 Las producciones definidas en 10.6 (salvo lo especificado en 10.6.2) serán utilizadas si, y solamente si, el nombre de referencia simple correspondiente ha sido importado desde el módulo identificado por la "modulereference" y:

a) nombres de referencia idénticos han sido importados desde módulos diferentes, o han sido generados en una cláusula de redenominaciones en este EDM, o han sido importados y generados; o

b) el nombre de referencia simple es "BuiltinEncodingClassReference" (véase 10.5); o

c) se cumplen ambas condiciones.

10.7 Una referencia parametrizada es un nombre de referencia definido en una "ParameterizedAssignment" (véase C.1) y suministrado con un parámetro real de acuerdo con la sintaxis de C.3. Las producciones involucradas son:

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

10.8 Las producciones que permiten todas las formas de identificación son:

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

10.9 Las producciones que permiten todas las formas de definición excepto la definición en línea son:

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

con la salvedad de que las clases de codificación incorporadas y los conjuntos de objetos de codificación incorporados no están permitidos por "DefinedEncodingClass" ni por "DefinedEncodingObjectSet".

NOTA – Se utiliza también otra producción, la "SimpleDefinedEncodingClass". Se define en C.3 y sólo permite "encodingclassreference" y "ExternalEncodingClassReference".

10.9.1 The "DefinedEncodingClass" and "DefinedOrBuiltinEncodingClass are:

DefinedEncodingClass ::=
encodingclassreference
| **ExternalEncodingClassReference**
| **ParameterizedEncodingClass**

DefinedOrBuiltinEncodingClass ::=
DefinedEncodingClass
| **BuiltinEncodingClassReference**

10.9.2 The "DefinedEncodingObject" is:

DefinedEncodingObject ::=
encodingobjectreference
| **ExternalEncodingObjectReference**
| **ParameterizedEncodingObject**

10.9.3 The "DefinedEncodingObjectSet" and "DefinedOrBuiltinEncodingObjectSet" are:

DefinedEncodingObjectSet ::=
encodingobjectsetreference
| **ExternalEncodingObjectSetReference**
| **ParameterizedEncodingObjectSet**

DefinedOrBuiltinEncodingObjectSet ::=
DefinedEncodingObjectSet
| **BuiltinEncodingObjectSetReference**

11 Codificación de tipos ASN.1

11.1 Generalidades

11.1.1 Para todo tipo ASN.1 existe la correspondiente estructura de codificación generada implícitamente. La estructura de codificación se genera implícitamente para cada asignación de tipo ASN.1 y se exporta de forma automática desde el módulo ASN.1 que contiene esa asignación de tipo. (No obstante, si se va a utilizar ha de ser importada en un módulo EDM). El nombre de la estructura de codificación correspondiente es el nombre del tipo precedido por un carácter **#**. Esta estructura de codificación define una clase de codificación y se denomina **estructura de codificación generada implícitamente**.

11.1.2 Puede haber una o más **estructuras de codificación generadas explícitamente**. Se generan en un EDM utilizando una cláusula de redenominaciones.

11.1.3 La codificación de un tipo ASN.1 se define formalmente como el resultado de las codificaciones aplicadas de manera precisa a una de las estructuras de codificación generadas (implícita o explícitamente) a partir del tipo ASN.1. Las codificaciones se aplican mediante declaraciones en el ELM (véase la cláusula 12), utilizando objetos de codificación de un conjunto de objetos de codificación combinados. Un ELM aplicará codificaciones como máximo a una de las estructuras de codificación generadas correspondientes a cualquier tipo ASN.1 dado.

11.1.4 La estructura de codificación generada implícitamente se define simplificando primero y ampliando la notación ASN.1 (especificada en 11.3), y estableciendo a continuación la correspondencia entre tipos ASN.1, constructores de tipos y nombres de componentes por un lado y clases de codificación incorporadas, constructores de codificación y nombres de campos de estructura de codificación correspondientes por otro.

11.1.5 Una estructura de codificación generada explícitamente se define introduciendo los cambios especificados en la estructura de codificación generada implícitamente mediante una cláusula de redenominaciones.

11.1.6 Cada campo de una estructura de codificación generada tiene asociados los valores abstractos del tipo correspondiente e información relativa a la construcción derivada de la definición del tipo ASN.1 (véase 11.4.2). Por definición, las codificaciones de los valores abstractos de la estructura de codificación generada son las codificaciones de los valores abstractos correspondientes del tipo ASN.1 original.

11.1.7 La presente cláusula 11 especifica:

a) Las clases de codificación incorporadas que se utilizan para definir las estructuras de codificación generadas implícitamente correspondientes a tipos ASN.1 (véase 11.2).

NOTA – En 16.1.14 se especifican clases adicionales utilizadas en la definición de estructuras de codificación definidas por el usuario.

b) Las transformaciones de la sintaxis ASN.1 (simplificación y expansión) antes de que se produzca la estructura generada implícitamente (véase 11.3).

c) La estructura de codificación generada implícitamente para cualquier tipo ASN.1 (véase 11.4).

11.2 Clases de codificación incorporadas utilizadas para estructuras de codificación generadas implícitamente

11.2.1 Las clases de codificación utilizadas para estructuras de codificación generadas implícitamente, y los tipos o constructores ASN.1 a los que corresponden, se indican en el cuadro 2 que sigue.

11.2.2 La columna 1 da la notación ASN.1 que es reemplazada por una clase de codificación en la estructura de codificación generada implícitamente. La columna 2 da la clase de codificación que reemplaza a la notación de la columna 1. La columna 3 da la clase primitiva de la que se deriva la clase de la columna 2.

Cuadro 2 – Clases de codificación de la notación ASN.1

ASN.1 notation	Encoding Class	Primitive Class
BIT STRING	**#BIT-STRING**	**#BITS**
BOOLEAN	**#BOOLEAN**	**#BOOL**
CHARACTER STRING	**#CHARACTER-STRING**	*Defined using **#SEQUENCE***
CHOICE	**#CHOICE**	**#ALTERNATIVES**
EMBEDDED PDV	**#EMBEDDED-PDV**	*Defined using **#SEQUENCE***
ENUMERATED	**#ENUMERATED**	**#INT**
EXTERNAL	**#EXTERNAL**	*Defined using **#SEQUENCE***
INTEGER	**#INTEGER**	**#INT**
NULL	**#NULL**	**#NUL**
OBJECT IDENTIFIER	**#OBJECT-IDENTIFIER**	**#OBJECT-IDENTIFIER**
OCTET STRING	**#OCTET-STRING**	**#OCTETS**
open type notation	**#OPEN-TYPE**	**#OPEN-TYPE**
OPTIONAL	**#OPTIONAL**	**#OPTIONAL**
REAL	**#REAL**	**#REAL**
RELATIVE-OID	**#RELATIVE-OID**	**#OBJECT-IDENTIFIER**
SEQUENCE	**#SEQUENCE**	**#CONCATENATION**
SEQUENCE OF	**#SEQUENCE-OF**	**#REPETITION**
SET	**#SET**	**#CONCATENATION**
SET OF	**#SET-OF**	**#REPETITION**
GeneralizedTime	**#GeneralizedTime**	**#CHARS**
UTCTime	**#UTCTime**	**#CHARS**
ObjectDescriptor	**#ObjectDescriptor**	**#CHARS**
BMPString	**#BMPString**	**#CHARS**
GeneralString	**#GeneralString**	**#CHARS**
GraphicString	**#GraphicString**	**#CHARS**
IA5String	**#IA5String**	**#CHARS**
NumericString	**#NumericString**	**#CHARS**
PrintableString	**#PrintableString**	**#CHARS**
TeletexString	**#TeletexString**	**#CHARS**
UniversalString	**#UniversalString**	**#CHARS**
UTF8String	**#UTF8String**	**#CHARS**
VideotexString	**#VideotexString**	**#CHARS**
VisibleString	**#VisibleString**	**#CHARS**
Textually present tag notation	**#TAG**	**#TAG**

11.3 Simplificación y expansión de la notación ASN.1 a efectos de codificación

11.3.1 En la ECN se supone que determinados constructivos sintácticos ASN.1 han sido expandidos (o reducidos) convirtiéndolos en construcciones equivalentes o más sencillas.

NOTA – Los tipos definidos por las construcciones más sencillas pueden llevar el mismo conjunto de valores abstractos que las estructuras sintácticas ASN.1 originales, y esos valores abstractos se hacen corresponder con las construcciones más sencillas.

11.3.2 La expansión o simplificación de construcciones sintácticas ASN.1 es:

a) definida por completo en 11.3.4; o

b) referenciada en llamadas del tipo "véase 11.3.2 b" y definida por completo en la Rec. UIT-T X.680 | ISO/CEI 8824-1 (incluido el anexo F) con todas las enmiendas y corrigendos técnicos publicados; o

c) referenciada en llamadas del tipo "véase 11.3.2 c" y definida por completo en la Rec. UIT-T X.681 | ISO/CEI 8824-2 con todas las enmiendas y corrigendos técnicos publicados; o

d) referenciada en llamadas del tipo "véase 11.3.2 d" y definida por completo en la Rec. UIT-T X.683 | ISO/CEI 8824-4 con todas las enmiendas y corrigendos técnicos publicados.

11.3.3 En el resto de la presente Recomendación | Norma Internacional no se hacen más referencias a los constructivos sintácticos eliminados por las expansiones y simplificaciones que se indican más adelante.

11.3.4 Las expansiones o simplificaciones siguientes se aplicarán a todos los módulos ASN.1.

11.3.4.1 Las transformaciones que se indican a continuación no son recursivas y por tanto sólo se aplican una vez:

a) Todas las "ValueSetTypeAssignment" serán sustituidos por sus "TypeAssignment" equivalentes con constricciones de subtipo. (Véase 11.3.2 b).

b) La construcción ASN.1 **INSTANCE OF** será expandida hasta su tipo secuencia equivalente. (Véase 11.3.2 c).

c) El "TypeFromObject" será sustituido por el tipo que es referenciado. (Véase 11.3.2 c).

d) El "ValueSetFromObjects" será sustituido por el tipo que es referenciado. (Véase 11.3.2 c).

e) Cuando una instancia de notación de rótulo ASN.1 vaya seguida textualmente por una o más instancias adicionales de notación de rótulo ASN.1, la segunda instancia e instancias subsiguientes de la notación de rótulo serán descartadas.

NOTA – Esto es similar a las reglas de rotulación implícitas de ASN.1, pero se aplica a todos los entornos de rotulación. Todavía es posible la rotulación múltiple del mismo tipo utilizando nombres de referencia de tipo.

11.3.4.2 Las transformaciones siguientes se aplicarán de manera recursiva en el orden especificado, hasta llegar a una expresión de coma fija:

a) Todas las parametrizaciones ASN.1 se resolverán por completo sustituyendo los parámetros reales por parámetros ficticios. (Véase 11.3.2 d).

NOTA – Esto significa que cuando una notación de tipo ASN.1 contiene la instanciación de un tipo ASN.1 parametrizado, esa instanciación pasa a ser una definición en línea.

b) Todos los "ComponentsOf" serán expandidos hasta su forma completa. (Véase 11.3.2 b).

c) Todos los usos de "SelectionType" serán resueltos. (Véase 11.3.2 b).

11.3.4.3 A continuación se aplicarán las transformaciones siguientes:

a) Las listas de números denominados de las definiciones de tipos enteros serán eliminadas. Los números denominados no pueden ser vistos por la ECN. La ECN ve una sola clase **#INTEGER** (posiblemente con los límites especificados en 11.3.4.3 c).

b) Las listas de bits denominados de las definiciones de cadenas de bits serán eliminadas. Los bits denominados no pueden ser vistos por la ECN.

c) Toda notación de constricción no visible a los efectos de PER, excepto la constricción de contenido, será descartada. Las constricciones no visibles a los efectos de PER serán resueltas para proporcionar los valores siguientes que pueden ser referenciados en la definición de reglas de codificación:

i) Un límite superior aplicado a enteros y enumeraciones;

ii) Un límite inferior aplicado a enteros y enumeraciones;

iii) El alfabeto PER permitido efectivo y las constricciones de tamaño efectivo (véase 9.3 de la Rec. UIT-T X.691 | ISO/CEI 8825-2).

d) Si hay una constricción de contenido con una construcción **CONTAINING**, la existencia de la constricción de contenido, su tipo contenido y la presencia o ausencia de una cláusula **ENCODED BY** pasan a ser propiedades asociadas a los valores abstractos de ese tipo constreñido de cadena de octetos o cadena de bits, y la constricción es entonces descartada. Si hay una constricción de contenido sin construcción **CONTAINING**, la ECN no puede verla y es descartada.

NOTA – Cuando se especifican codificaciones de valores con una constricción de contenido asociada, se puede suministrar un objeto de codificación combinado aparte para codificar el tipo contenido. Se puede especificar la invalidación o no de cualquier **ENCODED BY** que esté presente, a criterio del diseñador. (Véase 11.3 y 13.2).

e) Toda rotulación que no esté presente textualmente en la notación ASN.1 será ignorada en el establecimiento de la correspondencia con estructuras de codificación, pero (para modelar las codificaciones BER y los procedimientos PER) la lista completa de rótulos de un tipo pasa a ser una propiedad del campo de la estructura de codificación con la que se establece la correspondencia de los valores correspondientes.

f) La notación de rótulo presente textualmente tiene la clase del rótulo eliminada. (Véase también 11.3.4.1 e).

g) **DEFAULT** Value será sustituido por **OPTIONAL-ENCODING #OPTIONAL** y el valor por defecto se asociará al campo de la estructura con la que se establece la correspondencia del componente ASN.1.

h) **OPTIONAL** será sustituido por **OPTIONAL-ENCODING #OPTIONAL**.

i) **T61String** será sustituido por **#TeletexString**.

j) **ISO64String** será sustituido por **#VisibleString**.

11.3.4.4 Por último, se aplicarán las transformaciones siguientes:

a) Se efectuará la atribución automática de valores a las enumeraciones (si procede). La sintaxis **ENUMERATED** será sustituida por la clase de codificación **#ENUMERATED** con un límite superior y un límite inferior fijados. (Véase 11.3.4.3 c).

NOTA 1 – La clase **#ENUMERATED** desreferencia a la clase **#INT** (véase 11.2.2), y se establece la correspondencia entre las enumeraciones y los valores enteros limitados de la clase. La ECN no puede ver los nombres reales de las enumeraciones.

b) Todas las ocurrencias de "ObjectClassFieldType" (véase la cláusula 14 de la Rec. UIT-T X.681 | ISO/CEI 8824-2) que se refieren a un campo de tipo, un campo de valor de tipo variable o un campo de conjunto de valores de tipo variable serán sustituidas por la clase de codificación **#OPEN-TYPE**. (Véase 11.3.2 c).

c) Los marcadores de extensibilidad y los corchetes de versión de las construcciones de secuencia, conjunto y elección son eliminados, pero (para modelar las codificaciones BER y los procedimientos PER) la identificación de un componente como parte de la raíz, o de la versión 1, o de la versión 2, etc., pasa a ser una propiedad del componente, y la existencia del marcador de extensibilidad se convierte en una propiedad de la clase con la que se corresponde la construcción.

d) El marcador de extensibilidad de las constricciones es eliminado, pero la existencia del marcador de extensibilidad pasa a ser una propiedad de la clase y la presencia de un valor abstracto en la raíz o en una extensión se convierte en una propiedad del valor abstracto.

NOTA 2 – Las propiedades referenciadas en los incisos c) y d) anteriores sólo pueden ser interrogadas mediante una definición no ECN de los objetos codificados en esta versión de la presente Recomendación | Norma Internacional. Está previsto dar, en una versión posterior de esta Recomendación | Norma Internacional, pleno soporte a la extensibilidad.

11.3.5 Con estas transformaciones, todos los constructivos relacionados con tipos ASN.1 tienen clases de codificación correspondientes, indicadas en el cuadro 2. La estructura de codificación generada implícitamente se construirá estableciendo la correspondencia entre constructivos relacionados con tipos ASN.1 de la columna 1 y clases de la columna 2 del cuadro 2 (como se especifica en 11.4).

11.4 Estructura de codificación generada implícitamente

11.4.1 Hay una estructura generada implícitamente por cada definición de tipo ASN.1 con un nombre construido a partir del nombre de referencia del tipo ASN.1, poniendo como prefijo el carácter "**#**". Cuando se necesite un nombre totalmente cualificado para una estructura de codificación generada implícitamente, ese nombre totalmente cualificado incluirá el "ModuleIdentifier" del módulo ASN.1 que contiene la definición del tipo. (En D.1.9.2 se da un ejemplo de estructura generada implícitamente).

NOTA – Se genera implícitamente y se exporta una estructura por cada tipo ASN.1 de un módulo ASN.1 con independencia de si ese tipo está o no indicado en la cláusula **EXPORTS**.

11.4.2 La estructura de codificación generada implícitamente es similar a la de la definición del tipo ASN.1, con:

a) Identificadores de componentes del tipo ASN.1 cuya correspondencia se establece con nombres de campos de la estructura de codificación.

b) Notaciones ASN.1 de la columna 1 del cuadro 2 cuya correspondencia se establece con las clases de codificación incorporadas de la columna 2 del cuadro 2.

NOTA 1 – Se establece la correspondencia entre el primer rótulo presente textualmente y una construcción "**[#TAG]**" en la estructura generada implícitamente. La estructura generada implícitamente no contiene ninguna construcción "**[#TAG]**" para rótulos presentes textualmente subsiguientes.

c) Se establece la correspondencia entre los "DefinedType" de ASN.1 y un nombre de clase de codificación derivado de la referencia del tipo mediante la adición de un carácter "**#**". Si se importa un tipo en el módulo ASN.1, cualquier referencia "ExternalEncodingClassReference" que se haga a la clase correspondiente en una estructura generada implícitamente hará referencia al módulo ASN.1 que contiene la definición del tipo referenciado.

NOTA 2 – Si la clase resultante es el nombre de una clase de codificación incorporada, todas las referencias a la misma en la cláusula de redenominaciones, o en el ELM, utilizarán la notación "ExternalEncodingClassReference".

d) Se establece la correspondencia entre valores abstractos de un campo de la definición de tipo y el campo correspondiente de la estructura de codificación.

e) Se establece la correspondencia entre límites superiores e inferiores impuestos a tipos enteros y enumerados y todas las constricciones de tamaño efectivo y constricciones de alfabeto permitido efectivo (véase 9.3 de la Rec. UIT-T X.691 | ISO/CEI 8825-2) de la definición de tipo por un lado y el campo correspondiente de la estructura de codificación por otro.

f) Se establece la correspondencia entre el número de rótulo del primer rótulo presente textualmente y la clase **#TAG**.

11.4.3 Se producen y exportan desde todos los módulos ASN.1 otras tres estructuras generadas implícitamente. Sus nombres son **#CHARACTER-STRING**, **#EMBEDDED-PDV** y **#EXTERNAL**, y las estructuras a las que desreferencian son estructuras generadas implícitamente correspondientes a los tipos asociados para **CHARACTER STRING**, **EMBEDDED PDV** y **EXTERNAL**, que se especifican en 40.5, 33.5 y 34.5, respectivamente, de la Rec. UIT-T X.680 | ISO/CEI 8824-1.

11.4.4 Todas las estructuras de codificación generadas implícitamente pueden ser codificadas por los conjuntos de objetos de codificación incorporados (véase 18.2), y producirán las mismas codificaciones especificadas por la Recomendación | Norma Internacional correspondiente para esas codificaciones cuando se apliquen a tipos ASN.1.

12 Módulo de enlace de codificación (ELM)

NOTA – En la ECN hay dos producciones de nivel máximo, la "ELMDefinition" especificada en esta cláusula y la "EDMDefinition" especificada en la cláusula 14. En ellas se especifica la sintaxis de la definición del ELM y los EDM, respectivamente.

12.1 Estructura del ELM

12.1.1 La producción "ELMDefinition" es:

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

12.1.2 En cualquier aplicación dada de la ECN, habrá precisamente un ELM que determine la codificación de todos los mensajes utilizados en esa aplicación.

NOTA – El tipo o los tipos ASN.1 que definen "mensajes" se denominan a menudo "tipos de nivel máximo".

12.1.3 La producción "ModuleIdentifier" (y su semántica) se define en 12.1 de la Rec. UIT-T X.680 | ISO/CEI 8824-1.

12.1.4 El "ModuleIdentifier" proporciona una identificación inequívoca de cualquier módulo del conjunto de todos los módulos ASN.1, ELM y EDM.

12.1.5 La producción "ELMModuleBody" es:

```
ELMModuleBody ::=
        Imports ?
        EncodingApplicationList

EncodingApplicationList ::=
        EncodingApplication
        EncodingApplicationList ?
```

12.1.6 La producción "Imports" (y su semántica) se define en 12.1, 12.15 y 12.16 de la Rec. UIT-T X.680 | ISO/CEI 8824-1, modificada por A.1 de la presente Recomendación | Norma Internacional.

12.1.7 Todos los nombres de referencia utilizados en el "ELMModuleBody" serán importados en el ELM.

NOTA – Este requisito es más estricto que el impuesto para los módulos ASN.1. En los módulos ASN.1 se pueden utilizar referencias externas para tipos y valores que no han sido importados. En un módulo ELM (y en un módulo EDM) sólo se pueden utilizar referencias externas para clases de codificación que han sido referenciadas en una cláusula de importaciones. Las referencias externas tienen por objeto únicamente resolver ambigüedades entre nombres importados y nombres incorporados, o entre dos nombres idénticos importados desde módulos diferentes.

12.1.8 La producción "Imports" pone a disposición del ELM:

a) estructuras de codificación generadas implícitamente procedentes de un módulo ASN.1;

b) estructuras de codificación generadas explícitamente procedentes de un módulo EDM;

NOTA – Cuando un ELM importa desde un EDM una estructura de codificación generada explícitamente, las cláusulas de redenominaciones de otros EDM no tienen ningún efecto en la codificación de esa estructura (véase 15.2.4).

c) objetos y conjuntos de objetos de codificación procedentes de un módulo EDM.

12.1.9 La producción "EncodingApplicationList" ha de contener al menos una "EncodingApplication", ya que la única función de un ELM es aplicar codificaciones.

12.2 Tipos de codificación

12.2.1 Una "EncodingApplication" es:

12.2.1 An "EncodingApplication" is:

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

12.2.2 Una "EncodingApplication" define la codificación de los tipos ASN.1 correspondientes a las "SimpleDefinedEncodingClass" que serán estructuras de codificación generadas. La codificación de los tipos es especificada por las "CombinedEncodings" aplicadas a las estructuras de codificación generadas que se especifican en 13.2.

NOTA – Para un ELM, lo normal será codificar un solo tipo de un solo módulo, pero cuando se codifican múltiples tipos, los fabricantes de herramientas ECN pueden suponer (aunque no necesariamente) que así se identifican implícitamente tipos de nivel máximo que requieren soporte en estructuras de datos generadas.

12.2.3 Las codificaciones aplicadas a una estructura de codificación generada correspondiente a un tipo ASN.1 definido en algún módulo ASN.1 están vinculadas únicamente a la utilización de ese tipo como mensajes de aplicación. No tienen implicación alguna en la codificación del tipo cuando es referenciado por otros tipos o cuando es exportado desde ese módulo ASN.1 e importado en un módulo ASN.1 diferente.

12.2.4 La codificación del tipo en una constricción de contenido es la especificada por el objeto de codificación aplicado a la clase contenedora en la categoría cadena de octetos o cadena de bits, y puede ser un conjunto cualquiera de objetos de codificación combinados o el conjunto de objetos de codificación combinados que se aplicó a la clase contenedora en la categoría cadena de octetos o cadena de bits.

12.2.5 Un ELM no aplicará codificaciones al mismo tipo ASN.1 más de una vez.

NOTA – Las reglas de aplicación de las codificaciones (especificadas en la cláusula 13) significan que una "EncodingApplication" define por completo la codificación de un tipo a menos que contenga una instancia de una constricción de contenido.

13 Aplicación de codificaciones

13.1 Generalidades

13.1.1 Las codificaciones son aplicadas por el ELM a una estructura generada (o independientemente a múltiples estructuras generadas) utilizando la definición de "CombinedEncodings" especificada en 13.1.3. Esta cláusula, junto con 13.2, especifica la aplicación de "CombinedEncodings" a una estructura de codificación generada.

13.1.2 En el ELM, la aplicación se hace a las estructuras de codificación generadas identificadas en la "EncodingApplication". En cláusulas posteriores se especifica además la aplicación de codificaciones a la totalidad o a parte de una definición de estructura de codificación cualquiera. La presente cláusula es aplicable en ambos casos.

13.1.3 La producción "CombinedEncodings" es:

```
CombinedEncodings ::=
    WITH
    PrimaryEncodings
    CompletionClause ?

CompletionClause ::=
    COMPLETED BY
    SecondaryEncodings

PrimaryEncodings ::= EncodingObjectSet

SecondaryEncodings ::= EncodingObjectSet
```

13.1.4 "EncodingObjectSet" se define en 18.1.1.

13.1.5 La utilización de "CombinedEncodings" se especifica en 13.2.

13.2 Conjunto de objetos de codificación combinados y su aplicación

13.2.1 Un **conjunto de objetos de codificación combinados** se forma a partir de la producción "CombinedEncodings" (véase 13.1.3) como sigue:

13.2.2 Si no hay "CompletionClause", las "PrimaryEncodigns" forman el conjunto de objetos de codificación combinados.

13.2.3 De otro modo,

a) se colocan todos los objetos de codificación de las "PrimaryEncodings" en el conjunto de objetos de codificación combinados, y a continuación

b) se añaden todos los objetos de codificación de las "SecondaryEncodings" al conjunto de objetos de codificación combinados si, y solamente si, no hay ya ningún objeto de codificación en el conjunto de objetos de codificación combinados que tenga la misma clase de codificación (véanse 17.1.7 y 9.23.2).

13.2.4 Tras esta construcción conceptual del conjunto de objetos de codificación combinados, comienza la codificación con el nombre "encodingclassreference" de las estructuras de codificación identificadas en la aplicación de la codificación (véanse 13.1.2 y 17.5).

13.2.5 Cuando hay varias aplicaciones de codificación en el ELM, las reglas de 12.2 garantizan el que las aplicaciones no se solapen. Proceden independientemente. De manera similar, las codificaciones que se aplican a las estructuras de codificación de los EDM (que se especifican en 13.2.10) no se solapan nunca. Las subcláusulas que siguen contienen las reglas de aplicación a una estructura de codificación única.

13.2.6 Los objetos de codificación de un conjunto de objetos de codificación combinados se aplican en un **punto de aplicación**. El punto de aplicación es inicialmente la "encodingclassreference" de una estructura de codificación generada (cuando la aplicación se hace en el ELM, como se especifica en 13.1.2) o es un componente de una estructura de codificación (cuando la aplicación se hace en un EDM, como se especifica en 17.5).

13.2.7 Cualquier clase de codificación en las categorías alternativas, concatenación y repetición (véanse 16.1.8, 16.1.9 y 16.1.10) es un constructor de codificación.

13.2.8 El término "componente" en el texto que viene a continuación se refiere a cualquiera de los elementos siguientes:

a) Las alternativas de un constructor que está en la categoría alternativas.

b) El campo que sigue a un constructor que está en la categoría repetición.

c) Los componentes de un constructor que está en la categoría concatenación.

d) Un tipo contenido (un tipo especificado en una constricción de contenido).

e) El tipo elegido (en una instancia de comunicación) para utilizarlo con una clase en la categoría tipo abierto.

13.2.9 En etapas posteriores de estos procedimientos, el punto de aplicación puede ser cualquiera de los elementos siguientes:

a) Un nombre de clase de codificación. Puede ser codificado por completo utilizando la especificación de un nombre de codificación de la misma clase (véase 17.1.7).

b) Un constructor de codificación (véase 16.2.12). Los procedimientos de construcción pueden ser determinados por la especificación contenida en un objeto de codificación de la clase constructor de codificación, pero ese objeto de codificación no determina la codificación de los componentes. La especificación del objeto de codificación que se aplica quizá requiera que uno o más componentes del constructor sean sustituidos por otras estructuras (parametrizadas) antes de que el punto de aplicación pase a los componentes.

c) Una clase en la categoría cadena de bits o cadena de octetos que tiene un tipo contenido como una propiedad asociada a los valores (véase 11.3.4.3 d). La codificación del tipo contenido depende de si está o no presente un **ENCODED BY** y de la especificación del objeto de codificación que se aplica (véase 22.11).

d) Un componente que es una clase de codificación (posiblemente precedida por una o más clases en la categoría rótulo), seguido por una clase de codificación en la categoría opcionalidad. Los procedimientos y las codificaciones para determinar la presencia o la ausencia vienen determinados por la especificación contenida en un objeto de codificación de la clase en la categoría opcionalidad. Este objeto de codificación puede requerir también la sustitución de la clase de codificación (junto con todas sus clases precedentes en la categoría rótulo) por una estructura de sustitución (parametrizada) antes de que la clase sea codificada. El punto de aplicación pasa entonces a la primera clase en la categoría rótulo (si la hay), o al componente, o a su reemplazante.

e) Una clase de codificación precedida por una clase de codificación en la categoría rótulo. El número de rótulo asociado a la clase en la categoría rótulo se codifica utilizando la especificación de un objeto de codificación de la clase en la categoría rótulo, y el punto de aplicación pasa entonces a la clase rotulada.

f) Cualquier otra clase de codificación incorporada. Puede ser codificada por completo utilizando la especificación contenida en un objeto de codificación de esa clase.

13.2.10 La codificación continúa como sigue:

13.2.10.1 Si el conjunto de objetos de codificación combinados contiene un objeto de codificación de la misma clase (véase 17.1.7) que el punto de codificación actual, se aplica ese objeto de codificación. La aplicación puede provocar la sustitución de uno o más componentes de la clase a la que se aplica la codificación. Si el conjunto de objetos de codificación combinados no contiene ese objeto de codificación:

a) la clase de codificación en el punto de aplicación actual es una referencia a otra clase de codificación; en este caso se desreferencia y se aplican los procedimientos de 13.2.10 de manera recursiva; o

b) la clase de codificación en el punto de aplicación actual no es una referencia a otra clase de codificación; en este caso la especificación ECN es un error.

13.2.10.2 Si ha sido aplicada una codificación en el punto de aplicación a la clase de codificación, y no está en la categoría opcionalidad o rótulo y no tiene ningún componente (véase 13.2.7), esa aplicación determina por completo la codificación de la clase y finaliza estos procedimientos.

13.2.10.3 Si ha sido aplicada una codificación en el punto de aplicación a una clase de codificación que está en la categoría opcionalidad, el punto de aplicación pasa al componente opcional (posiblemente rotulado).

13.2.10.4 Si ha sido aplicada una codificación en el punto de aplicación a una clase de codificación que está en la categoría rótulo, el punto de aplicación pasa al elemento rotulado y se aplican los procedimientos de 13.2.10 de manera recursiva.

13.2.10.5 Si ha sido aplicada una codificación en el punto de aplicación a una clase de codificación que tiene componentes que no son un tipo contenido, se aplican los procedimientos de 13.2.10 de manera recursiva a cada componente.

NOTA – Esto implica la aplicación del conjunto actual de objetos de codificación combinados al tipo elegido (en una instancia de comunicación) para su utilización con una clase en la categoría tipo abierto (véase 13.2.8 e).

13.2.10.6 Si ha sido aplicada una codificación en el punto de aplicación a una clase de codificación que tiene un componente que es una clase en la categoría cadena de bits o cadena de octetos con un tipo contenido asociado a los valores, pueden darse cuatro casos:

a) La constricción de contenido contiene un parámetro **ENCODED BY** y el objeto de codificación de esa clase no contiene una especificación de la codificación del tipo contenido, o especifica que no debería invalidar **ENCODED BY** (véase 22.11). En este caso, la especificación **ENCODED BY** será utilizada para el tipo contenido, y el punto de aplicación pasa al tipo contenido que utiliza esta especificación de codificación.

b) La constricción de contenido contiene un parámetro **ENCODED BY** pero el objeto de codificación de esta clase contiene una especificación de la codificación del tipo contenido, y especifica que debería invalidar **ENCODED BY**. En este caso, la especificación del objeto de codificación será aplicada al tipo contenido, y el punto de aplicación pasa al tipo contenido que utiliza esta especificación de codificación.

c) La constricción de contenido no contiene un parámetro **ENCODED BY** y el objeto de codificación de esta clase contiene una especificación de la codificación del tipo contenido. En este caso, la especificación del objeto de codificación se aplica al tipo contenido, y el punto de aplicación pasa al tipo contenido que utiliza esta especificación de codificación.

d) La constricción de contenido no contiene un parámetro **ENCODED BY** y el objeto de codificación de esta clase no contiene una especificación de la codificación del tipo contenido. En este caso, el conjunto de objetos de codificación combinados que se aplica a la clase será aplicado también al tipo contenido, y el punto de aplicación pasa al tipo contenido que utiliza esta especificación de codificación.

13.2.10.7 Si en el conjunto de objetos de codificación combinados no hay ningún objeto de codificación de la misma clase (véase 17.1.7) que el punto de aplicación actual y el punto de aplicación actual es un nombre de referencia, es desreferenciado y se aplican estos procedimientos de manera recursiva a la estructura de codificación nueva.

13.2.10.8 De otro modo, la especificación ECN es un error.

13.2.11 El algoritmo anterior puede resumirse de la manera siguiente: el conjunto de objetos de codificación combinados se aplica de arriba abajo. Si en este proceso se encuentra un nombre de referencia de estructura de codificación y hay un objeto en el conjunto de objetos de codificación combinados que puede codificarlo, dicho objeto determina su codificación. De no ser así, el nombre de referencia se expande por desreferenciación. Si en cualquier etapa se requiere una codificación (que no existe) para una clase de codificación que no puede ser desreferenciada, la especificación ECN es incorrecta y se dice que la clase de codificación combinada está incompleta. Cuando se alcanza una clase de campo de bits primitiva, la codificación termina con la codificación de esa clase, excepto si tiene un tipo contenido en cuyo caso la codificación avanza hasta la estructura de codificación generada correspondiente al tipo contenido. Cuando se alcanza un tipo con componentes, el proceso continúa aplicando el conjunto de objetos de codificación combinados a cada componente de manera independiente. Cuando intervienen rótulos y opcionalidad, se codifica primero la clase opcionalidad, a continuación la clase de codificación en la categoría rótulo y por último el elemento. Cuando se aplican codificaciones a clases de constructor, se puede provocar la sustitución de uno o más componentes. Cuando se aplican a una clase de opcionalidad, se puede provocar la sustitución de todo el elemento (con la excepción de la clase de opcionalidad, pero incluyendo cualquier clase de codificación en la categoría rótulo).

13.2.12 En el proceso de codificación, los objetos de codificación aplicados a constructores de codificación (y a clases en la categoría opcionalidad) pueden requerir que los objetos de codificación aplicados a sus componentes exhiban asas de identificación (de un nombre determinado) para resolver alternativas, o la opcionalidad, o el orden en una concatenación similar a un conjunto. Si en este caso las codificaciones de los componentes no exhiben las asas de identificación requeridas, la especificación ECN es un error.

NOTA – Lo más probable es que este problema se plantee si se aplican objetos de codificación BER a constructores de codificación y no a sus componentes, ya que las BER dependen mucho de las asas de identificación. Los objetos de codificación de las PER no utilizan asas de identificación.

14 Módulo de definición de codificación (EDM)

NOTA – En la ECN hay dos producciones de nivel máximo, la "EDMDefinition" especificada en esta cláusula y la "ELMDefinition" especificada en la cláusula 12. En ellas se especifica la sintaxis para definir los EDM y el ELM, respectivamente.

14.1 La producción "EDMDefinition" es:

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

14.2 En cualquier aplicación dada de la ECN hay cero, uno o más EDM que definen objetos de codificación de aplicación en el ELM.

NOTA – Si hay cero EDM en el ELM, sólo se pueden utilizar conjuntos de objetos de codificación incorporados.

14.3 La producción "ModuleIdentifier" (y su semántica) se define en 12.1 de la Rec. UIT-T X.680 | ISO/CEI 8824-1.

14.4 El "ModuleIdentifier" proporciona una identificación inequívoca de cualquier módulo del conjunto de todos los módulos ASN.1, ELM y EDM.

14.5 La producción "ELMModuleBody" es:

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

EDMAssignmentList ::=
        EDMAssignment
        EDMAssignmentList ?


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

14.6 Las producciones "Exports" e "Imports" (y sus semánticas) se definen en 12.1 de la Rec. UIT-T X.680 | ISO/CEI 8824-1, modificada por A.1 de la presente Recomendación | Norma Internacional.

14.7 La producción "Exports" facilita la disponibilidad de cualquier nombre de referencia definido o importado en el EDM actual excepto el de una estructura generada explícitamente para su importación en otros EDM (y en el ELM). El "Symbol" de la "Exports" puede hacer referencia a cualquier clase de codificación (salvo una clase de codificación incorporada o una estructura generada implícitamente), un objeto de codificación o un conjunto de objetos de codificación. El "Symbol" habrá sido definido en este EDM o importado en él.

NOTA – Cuando el nombre de una estructura de codificación generada implícitamente e importada es una referencia de clase de codificación incorporada, se puede utilizar dentro del EDM con un nombre totalmente cualificado. Una estructura de codificación generada implícitamente nunca puede ser exportada desde un EDM (no obstante, estructuras de codificación que no han sido definidas utilizándolo sí pueden ser exportadas, naturalmente).

14.8 La producción "RenamesAndExports" se define en la cláusula 15.

14.9 La producción "RenamesAndExports" (llamada cláusula de redenominaciones) facilita la disponibilidad (dentro del EDM) de estructuras de codificación generadas explícitamente derivadas de las estructuras de codificación generadas implícitamente en módulos ASN.1 especificados. También facilita esas estructuras de codificación generadas explícitamente para su importación en otros EDM (y en el ELM). (Véase la cláusula 15).

14.10 La producción "Imports" facilita la disponibilidad (dentro del EDM) de clases de codificación, objetos de codificación y conjuntos de objetos de codificación exportados desde otro EDM o exportados automáticamente desde módulos ASN.1.

14.11 Todos los módulos ASN.1 que definen nombres de referencia de tipos no parametrizados producen automáticamente y exportan una estructura de codificación generada implícitamente del mismo nombre precedida por el carácter "**#**". Esas clases de codificación pueden ser importadas en un EDM desde ese módulo ASN.1.

NOTA – Cuando esos nombres son los mismos que los de clases de codificación incorporadas, se ha de utilizar la forma externa de referencia, especificada en A.1, en el cuerpo del módulo importador y en cualquier cláusula de redenominaciones.

14.12 Cada "EDMAssignment" define un nombre de referencia, y puede hacer uso de otros nombres de referencia. Cada nombre de referencia utilizado en un módulo será importado en ese módulo o será definido con precisión una vez dentro de ese modo.

NOTA – Este requisito es más estricto que el impuesto para los módulos ASN.1. En los módulos ASN.1 se pueden utilizar referencias externas para tipos y valores que no han sido importados. En un módulo EDM (y en un módulo ELM) sólo se pueden utilizar referencias externas para clases de codificación que han sido referenciadas en una cláusula de importaciones. La finalidad de las referencias externas es únicamente resolver ambigüedades entre nombres importados y nombres incorporados, o entre dos nombres idénticos importados desde módulos diferentes.

14.13 No es preciso definir textualmente(en otra declaración de asignación) cualquier nombre de referencia utilizado en una asignación antes de utilizarlo.

14.14 Las producciones de "EDMAssignment" se definen en las cláusulas que se indican a continuación:

EncodingClassAssignment	Cláusula 16
EncodingObjectAssignment	Cláusula 17
EncodingObjectSetAssignment	Cláusula 18
ParameterizedAssignment	Subcláusula C.1

NOTA – La producción "ParameterizedAssignment" permite la parametrización de una "EncodingClassAssignment", una "EncodingObjectAssignment" y una "EncodingObjectSetAssignment", como se especifica en C.1.

15 Cláusula de redenominaciones

15.1 Estructuras generadas explícitamente y exportadas

15.1.1 La producción "RenamesAndExports" es:

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

ExplicitGenerationList ::=
        ExplicitGeneration
        ExplicitGenerationList ?

ExplicitGeneration ::=
        OptionalNameChanges
        FROM GlobalModuleReference

OptionalNameChanges ::=
        NameChanges | GENERATES
```

NOTA – En D.3.7 se da un ejemplo de utilización de la cláusula de redenominaciones para producir estructuras de codificación generadas explícitamente.

15.1.2 La producción "GlobalModuleReference" se define en 12.1 de la Rec. UIT-T X.680 | ISO/CEI 8824-1 e identificará un módulo ASN.1.

15.1.3 La producción "RenamesAndExports" se denomina cláusula de redenominaciones.

15.1.4 Cada "ExplicitGeneration" genera explícitamente y exporta desde este módulo una estructura de codificación para cada una de las estructuras de codificación generadas implícitamente del módulo ASN.1 referenciado por la "GlobalModuleReference". Cada campo de la estructura de codificación generada explícitamente tiene asociados los mismos valores abstractos que el campo correspondiente de la estructura de codificación generada implícitamente (que son los valores asociados al campo correspondiente del tipo ASN.1 a partir del cual fue generada).

15.1.5 Si una cláusula de redenominaciones hace referencia a más de un módulo ASN.1, y como resultado de ello dos estructuras generadas explícitamente tienen el mismo nombre simple, ninguna de las dos puede ser importada explícitamente en un módulo ELM o EDM.

NOTA – Estas estructuras generadas explícitamente existen, no obstante, y probablemente sean referenciadas implícitamente por otras estructuras generadas explícitamente que son exportadas sin restricción alguna.

15.1.6 El objetivo primordial de la cláusula de redenominaciones es facilitar la disponibilidad de estructuras generadas explícitamente para importarlas en otros módulos, en particular el ELM. Sin embargo, esta cláusula también facilita la disponibilidad de esas estructuras a efectos de referencia dentro del módulo EDM que contiene la cláusula de redenominaciones, con la salvedad de lo especificado en 15.1.7. Si el nombre simple es ambiguo, se utilizará un nombre totalmente cualificado dentro del módulo EDM que contiene la cláusula de redenominaciones, como se especifica en 15.1.9.

NOTA – Puede haber un cierto grado de ambigüedad debido a que los nombre entran en colisión con nombres de clase incorporada, o por colisiones de nombres simples entre estructuras generadas a partir de más de un módulo ASN.1, o por ambas cosas.

15.1.7 Cuando una cláusula de redenominaciones produce una estructura generada explícitamente a partir de una estructura generada implícitamente, esa estructura generada implícitamente no puede ser importada en el módulo EDM utilizando una cláusula de importaciones, y la estructura generada implícitamente nunca está disponible en este módulo EDM.

15.1.8 Estas estructuras de codificación generadas explícitamente tienen el mismo nombre de referencia simple que la estructura de codificación generada implícitamente a partir de la cual se formaron (pero son clases distintas). Cuando se requiera un nombre totalmente cualificado para una estructura de codificación generada explícitamente, ese nombre totalmente cualificado incluirá el "ModuleIdentifier" del módulo EDM que contiene la cláusula de redenominaciones, como se especifica en 15.1.9.

NOTA – Las estructuras de codificación generadas implícitamente utilizadas en su generación tienen el mismo nombre de referencia simple, pero su nombre totalmente cualificado incluye el "ModuleIdentifier" del módulo ASN.1 en el que se definió el tipo correspondiente.

15.1.9 Si un EDM produce estructuras de codificación generadas explícitamente a partir de más de un módulo ASN.1, es posible que algunas de esas estructuras tengan los mismos nombres de clase de codificación simple. Si alguna de esas estructuras son referenciadas en el cuerpo del EDM, la referencia será una "ExternalEncodingClassReference" que contiene la "modulereference" utilizada como referencia de módulo ASN.1 en la cláusula de sustituciones de este módulo EDM.

15.1.10 La notación "ExternalEncodingClassReference" no se utilizará en una cláusula de importaciones salvo cuando así lo requiera 15.1.9.

15.1.11 Si un nombre que ha sido importado utilizando una "ExternalEncodingClassReference" se utiliza en el cuerpo de un módulo, se puede utilizar la "encodingclassreference" simple a menos que se requiera una "ExternalEncodingClassReference" según lo especificado en 15.1.9.

15.1.12 Si "OptionalNameChanges" es "**GENERATES**", todas las estructuras de codificación generadas explícitamente son las mismas estructuras de codificación generadas implícitamente utilizadas en su generación, salvo lo especificado en 15.1.14.

NOTA – (Preceptiva). Si en un módulo EDM hay múltiples estructuras con el mismo nombre de referencia simple (con independencia de si esos nombres proceden de una cláusula de importaciones o de una cláusula de redenominaciones, o de colisiones con clases incorporadas, o de cualquier combinación de todo esto), se utiliza un nombre totalmente cualificado salvo para hacer referencia a una clase incorporada. En el caso de estructuras de codificación generadas implícitamente, el nombre totalmente cualificado utiliza siempre el nombre del módulo ASN.1. En el caso de estructuras generadas por la cláusula de redenominaciones en un módulo EDM, se utiliza el nombre totalmente cualificado. Este nombre totalmente cualificado del cuerpo del EDM utiliza siempre el nombre del módulo ASN.1 referenciado por la cláusula de redenominaciones. En el caso de estructuras importadas desde otro módulo EDM, el nombre totalmente cualificado utiliza el nombre de ese módulo EDM. No cabe aquí ambigüedad alguna, ya que no se permite la importación si un módulo EDM produce múltiples estructuras generadas explícitamente con el mismo nombre de referencia simple.

15.1.13 Si "OptionalNameChanges" es "NameChanges" se sigue aplicando 15.1.14, pero las estructuras de codificación generadas explícitamente se modifican todavía más según lo especificado en 15.2.

15.1.14 Considérese una estructura de codificación generada implícitamente (digamos, A) que contiene una referencia de clase de codificación a alguna otra estructura de codificación generada implícitamente (digamos, B). Entonces:

a) Si esta cláusula de redenominaciones (en cualquiera de sus "ExplicitGeneration") produce una estructura de codificación generada explícitamente correspondiente a B (digamos, B1), la referencia correspondiente en la estructura de codificación generada explícitamente correspondiente a A es una referencia a B1.

b) Si no hay ninguna estructura de codificación generada explícitamente correspondiente a B, la referencia en la estructura de codificación generada correspondiente a A es una referencia a B.

15.2 Cambios de nombre

15.2.1 La producción "NameChanges" es:

NameChanges ::=
NameChange
NameChanges ?

NameChange ::=
OriginalClassName
AS
NewClassName
IN
NameChangeDomain

OriginalClassName ::= SimpleDefinedEncodingClass | BuiltinEncodingClassReference

NewClassName ::= encodingclassreference

15.2.2 Cada "NameChanges" especifica que, en la generación explícita de estructuras de codificación, todas las ocurrencias del "OriginalClassName" dentro del "NameChangeDomain" en las estructuras de codificación generadas implícitamente han de ser redenominadas como clase "NewClassName". El "NameChangeDomain" se especifica en 15.3 e identifica una o más estructuras de codificación generadas implícitamente (o componentes de esas estructuras) de un módulo ASN.1 referenciado por la "GlobalModuleReference" en la "ExplicitGeneration".

NOTA 1 – Así es posible aplicar a algunas ocurrencias de una clase codificaciones diferentes de las aplicadas a otras ocurrencias.

NOTA 2 – Esto implica que el "OriginalClassName" sólo puede ser un nombre generado implícitamente a partir de un tipo ASN.1, es decir, el nombre del tipo ASN.1 definido por el usuario (precedido por un carácter "#"), o uno de los nombres de clase cuya relación figura en la columna 2 del cuadro 2.

15.2.3 Las referencias de "OriginalClassName" a campos de la estructura de codificación generada implícitamente que corresponden a la utilización de "ExternalTypeReference" en la definición del tipo ASN.1 utilizarán la notación "SimpleDefinedEncodingClass" con la misma "modulereference" que la "ExternalTypeReference". De otro modo, si el "DefinedType" (precedido por un carácter "#") no es una "BuiltinEncodingClassReference", se utilizará una "encodingclassreference" simple. Si una "typereference" (precedida por un carácter "#") es una "BuiltinEncodingClassReference", se utilizará la notación "SimpleDefinedEncodingClass" con la misma "modulereference" que el módulo ASN.1 que generó la estructura de codificación generada implícitamente.

15.2.4 Cuando un ELM importa una estructura de codificación generada explícitamente desde un EDM, las cláusulas de redenominaciones de los otros EDM no tienen ningún efecto en la codificación de esa estructura.

NOTA – Esto significa en la práctica que todo el "colorante" (véase 9.16.4) necesario para un determinado mensaje particular se ha de aplicar en un solo EDM.

15.2.5 El "NewClassName" será definido en una declaración de asignación de clase de codificación (véase la cláusula 16) de la forma:

<*NewClassName*> **::=** <*OriginalClassName*>

donde "<*NewClassName*>" y "<*OriginalClassName*>" son los nombres de las clases nueva y original que aparecen en la producción "NameChanges". La asignación se hará en el módulo EDM con la cláusula de redenominaciones.

NOTA – Es preciso que el "<*OriginalClassName*>" haga referencia a una clase de codificación incorporada o una estructura de codificación generada externamente producida por la cláusula de redenominaciones en este módulo. En caso de ambigüedad, habrá que utilizar una referencia externa en "<*OriginalClassName*>".

15.3 Especificación de la región para cambios de nombre

15.3.1 La producción "NameChangeDomain" es:

NameChangeDomain ::=
IncludedRegions
Exception ?

Exception ::=
EXCEPT
ExcludedRegions

IncludedRegions ::=
ALL | RegionList

ExcludedRegions ::= RegionList

RegionList ::=
Region "," +

```
Region ::=
        SimpleDefinedEncodingClass |
        ComponentReference

ComponentReference ::=
        SimpleDefinedEncodingClass
        "."
        ComponentIdList

ComponentIdList ::=
        identifier "." +
```

15.3.2 Cada "SimpleDefinedEncodingClass" tendrá el nombre de una estructura de codificación generada implícitamente del módulo ASN.1 referenciado por la "GlobeModuleReference" en la "ExplicitGeneration". Cuando se utiliza en "Region", identifica la totalidad de una definición de estructura de codificación.

NOTA – La forma "ExternalEncodingClassReference" de "SimpleDefinedEncodingClass" se utiliza si la clase referenciada se ha derivado de un nombre "typereference" que (cuando va precedido por el carácter "#") es una "BuiltinEncodingClassReference" (véase 15.2.3).

15.3.3 Cada "identifier" será el "identifier" en un "NamedField" de la estructura de codificación generada implícitamente identificada por la "encodingclassreference" en la "ComponentReference". La "ComponentReference" identifica la definición completa del componente identificado de esa estructura de codificación.

15.3.4 El primer "identifier" de la "ComponentIdList" será el "identifier" en un "NamedField" de la estructura de codificación generada implícitamente identificada por la "encodingclassreference" de la "ComponentReference", e identificará la definición completa de ese componente de la estructura de codificación. Cada "identifier" subsiguiente de la "ComponentIdList" será el "identifier" en un "NamedField" de la estructura de codificación generada implícitamente identificada por la parte previa de la "ComponentIdList", e identificará la definición completa de ese componente.

15.3.5 Las definiciones identificadas por las diferentes "Region" de una "RegionList" serán separadas. Una definición es identificada por una "RegionList" si, y solamente si, es identificada por una "Region" de la "RegionList".

15.3.6 Si "IncludedRegions" es "**ALL**", identifica todas las partes de todas las estructuras de codificación generadas implícitamente del módulo ASN.1 referenciado por la "GlobalModuleReference" en la "ExplictGeneration".

15.3.7 Las definiciones identificadas por "ExcludedRegions" serán un subconjunto apropiado de las definiciones identificadas por "IncludedRegions".

15.3.8 La especificación "NameChangeDomain" identifica las definiciones en las que se han de introducir los cambios de nombre. Las definiciones del "NameChangeDomain" son las definiciones identificadas por "IncludedRegions" que no son identificadas también por "ExcludedRegions".

16 Asignaciones de clases de codificación

16.1 Generalidades

16.1.1 La producción "EncodingClassAssignment" es:

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

16.1.2 La "EncodingClassAssignment" asigna la "EncodingClass" a la "encodingclassreference".

NOTA – Cualquier notación "EncodingObject" que sea válida con "EncodingClass" como gobernador, es válida con "encodingclassreference" como gobernador.

16.1.3 Una clase de codificación está en una de las categorías siguientes:

a) Una categoría en el grupo de categorías campo de bits (véase 16.1.7).

b) La categoría alternativas (véase 16.1.8).

c) La categoría concatenación (véase 16.1.9).

d) La categoría repetición (véase 16.1.10).

e) La categoría opcionalidad (véase 16.1.11).

f) La categoría rótulo (véase 16.1.12).

g) Una categoría en el grupo de categorías procedimiento de codificación (véase 16.1.13).

NOTA – La expresión constructor de codificación se utiliza para cualquier clase en las categorías alternativas, concatenación y repetición. A éstas categorías se les denomina también grupo de categorías constructor de codificación.

16.1.4 La categoría de cada clase de codificación incorporada se especifica en 16.1.14.

NOTA – Si una clase de codificación es una clase rotulada (véase 16.2.1), o tiene límites (véase 16.2.6), la categoría de la clase es la categoría de la clase con el rótulo y los límites eliminados.

16.1.5 La producción "EncodingClass" es:

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

16.1.6 La producción "BuiltinEncodingClassReference" es:

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

16.1.7 La producción "BitfieldClassReference" es:

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

Las categorías de las clases a las que hacen referencia estos nombres incorporados (véase 16.1.14) se han definido de modo que estén todas ellas en el grupo de categorías campo de bits.

16.1.8 La producción "AlternativesClassReference" es:

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

16.1.9 La producción "ConcatenationClassReference" es:

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

16.1.10 La producción "RepetitionClassReference" es:

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

16.1.11 La producción "OptionalityClassReference" es:

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

16.1.12 La producción "TagClassReference" es:

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

16.1.13 La producción "EncodingProcedureClassReference" es:

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

16.1.14 Algunas de estas clases se definen como primitivas y sólo pueden ser codificadas por objetos de codificación de su propia clase. Otras se derivan de una clase primitiva mediante declaraciones de asignación de clase, y pueden ser desreferenciadas a estas clases. Su categoría es la de la clase de la que se han derivado. Las clases que siguen son las clases primitivas de las que se deriva cada clase incorporada mediante declaraciones de asignación de clase. Cuando se definen objetos de codificación de clases derivadas, se puede utilizar para la clase derivada cualquier sintaxis permitida para la clase primitiva correspondiente. La tercera columna del cuadro da la categoría de cada una de las clases incorporadas no derivadas de otras clases

Built-in class	**Derived from**	**Category**
#ALTERNATIVES	(primitive)	**alternatives**
#BITS	(primitive)	**bitstring**
#BIT-STRING	**#BITS**	
#BOOL	(primitive)	**boolean**
#BOOLEAN	**#BOOL**	
#CHARACTER-STRING	(defined using **#SEQUENCE)**	
#CHARS	(primitive)	**characterstring**
#CHOICE	**#ALTERNATIVES**	
#CONCATENATION	(primitive)	**concatenation**
#CONDITIONAL-INT	(primitive)	**encoding procedure**
#CONDITIONAL-REPETITION	(primitive)	**encoding procedure**
#EMBEDDED-PDV	(defined using **#SEQUENCE)**	
#ENUMERATED	**#INT**	
#EXTERNAL	(defined using **#SEQUENCE)**	
#INT	(primitive)	**integer**
#INTEGER	**#INT**	
#NUL	(primitive)	**null**
#NULL	**#NUL**	
#OBJECT-IDENTIFIER	(primitive)	**objectidentifier**
#OCTETS	(primitive)	**octetstring**
#OCTET-STRING	**#OCTETS**	
#OPEN-TYPE	(primitive)	**opentype**
#OPTIONAL	(primitive)	**optionality**
#OUTER	(primitive)	**encoding procedure**
#PAD	(primitive)	**pad**
#REAL	(primitive)	**real**

#RELATIVE-OID	**#OBJECT-IDENTIFIER**	
#REPETITION	(primitive)	**repetition**
#SEQUENCE	**#CONCATENATION**	
#SEQUENCE-OF	**#REPETITION**	
#SET	**#CONCATENATION**	
#SET-OF	**#REPETITION**	
#TAG	(primitive)	**tag**
#TRANSFORM	(primitive)	**encoding procedure**
#GeneralizedTime	**#CHARS**	
#UTCTime	**#CHARS**	
#ObjectDescriptor	**#CHARS**	
#BMPString	**#CHARS**	
#GeneralString	**#CHARS**	
#GraphicString	**#CHARS**	
#IA5String	**#CHARS**	
#NumericString	**#CHARS**	
#PrintableString	**#CHARS**	
#TeletexString	**#CHARS**	
#UniversalString	**#CHARS**	
#UTF8String	**#CHARS**	
#VideotexString	**#CHARS**	
#VisibleString	**#CHARS**	

16.2 Definición de estructura de codificación

16.2.1 La producción "EncodingStructure" es:

```
EncodingStructure ::=
        TaggedStructure
    |   UntaggedStructure

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

UntaggedStructure ::=
        DefinedEncodingClass
    |   EncodingStructureField
    |   EncodingStructureDefn

TagClass ::=
        DefinedEncodingClass |
        TagClassReference

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

16.2.2 Una "EncodingStructure" define una clase de codificación basada en la estructura que utiliza la notación especificada más adelante. Esta notación permite la definición de cualquier clase de codificación utilizando clases de codificación incorporadas y clases de codificación definidas (que pueden ser estructuras de codificación generadas) para campos de bits, constructores de codificación y clases de procedimiento de codificación en la categoría opcionalidad. Todas las clases definidas por "EncodingStructure" están en la categoría estructura de codificación. (En D.2.8.4 se dan ejemplos de asignación de estructura de codificación que ilustran muchas de las estructuras sintácticas, y D.2.2.3 es un ejemplo de la utilización de **`#TAG`**).

NOTA – La sintaxis prohíbe la especificación de una clase de rótulo inmediatamente después de otra clase de rótulo en la definición de una estructura de codificación; esas estructuras tampoco pueden ser producidas por múltiples rótulos textuales en una definición de tipo ASN.1 (véase 11.3.4.1 e).

16.2.3 La producción "DefinedEncodingClass" se especifica en 10.9.1 y será una clase en el grupo de categorías campo de bits.

16.2.4 La producción "DefinedEncodingClass" de la "TagClass" será una clase en la categoría rótulo (véase 16.1.3).

16.2.5 El "**`number`**" en "TagValue" especifica un número de rótulo que está asociado a la clase en la categoría rótulo.

16.2.6 La producción "EncodingStructureField" es:

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

16.2.7 Los "EncodingStructureField" representan todas las codificaciones de cadena de bits posibles de los tipos ASN.1 correspondientes y se les pueden asignar valores de esos tipos en un establecimiento de la correspondencia de valores (véase la cláusula 19).

16.2.8 Los valores ASN.1 que pueden ser asociados a cada campo primitivo son como sigue:

#NUL	The null value
#BOOL	The boolean values
#INT	The integer values
#BITS	Bitstring values
#OCTETS	Octetstring values
#CHARS	Character string values
#PAD	None
#OBJECT-IDENTIFIER	Object identifier values
#OPEN-TYPE	Open type values
#REAL	Real values
#TAG	Tag numbers

NOTA – El campo **#PAD** no puede tener valores ASN.1 asociados, y nunca está visible fuera de los procedimientos de codificación y decodificación.

16.2.9 "Bounds" y "Size" especifican los límites o la constricción de tamaño efectivo, respectivamente, impuestos a los valores abstractos cuya correspondencia se puede establecer con el campo (véase la cláusula 19).

NOTA – En una definición de estructura de codificación no se pueden asignar constricciones de alfabeto permitido efectivo. Sólo se pueden asignar mediante los establecimientos de la correspondencia de valores de la cláusula 19.

16.2.10 Los campos "Bounds" y "Size" son:

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

EffectiveRange ::=
        MinMax
    |   Fixed

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

SizeEffectiveRange ::=
        "(" EffectiveRange ")"

MinMax ::=
        ValueOrMin
        ".."
        ValueOrMax

ValueOrMin      ::=
        SignedNumber |
        MIN

ValueOrMax      ::=
        SignedNumber |
        MAX

Fixed ::= SignedNumber
```

16.2.11 **`MIN`** y **`MAX`** especifican que no hay límite superior ni inferior, respectivamente. **`MIN`** no se utilizará en "Size". "Fixed" significa que se trata de un solo valor o de un solo tamaño. "SignedNumber" se especifica en 18.1 de la Rec. UIT-T X.680 | ISO/CEI 8824-1. No será negativo cuando se utilice en "Size". "ValueOrMin" y "ValueOrMax" especifican límites superiores e inferiores, respectivamente.

16.2.12 La producción "EncodingStructureDefn" es:

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

16.2.13 Estas estructuras de codificación se definen en las cláusulas siguientes:

AlternativesStructure	16.3
RepetitionStructure	16.4
ConcatenationStructure	16.5

16.3 Estructura de codificación de alternativas

16.3.1 La producción "AlternativesStructure" es:

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

AlternativesClass ::=
        DefinedEncodingClass                |
        AlternativesClassReference

NamedFields ::= NamedField "," +

NamedField ::=
        identifier
        EncodingStructure
```

16.3.2 La "AlternativesStructure" identifica la presencia en una codificación de precisamente una de las "EncodingStructure" en sus "NamedFields". La "DefinedEncodingClass" será una clase en la categoría alternativas (véase 16.1.8). Los mecanismos utilizados para identificar cuál de las "EncodingStructure" está presente en una codificación son especificados por un objeto de codificación de la "AlternativesClass".

16.3.3 La "AlternativesStructure" es un constructor de codificación: cuando se aplica un conjunto de objetos de codificación a esta estructura como se especifica en 13.2, la codificación de la "AlternativesClass" determina la selección de alternativas, y el punto de aplicación pasa entonces a cada una de las "EncodingStructure" en sus "NamedFields".

16.4 Estructura de codificación de repetición

16.4.1 La producción "RepetitionStructure" es:

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

RepetitionClass ::=
        DefinedEncodingClass        |
        RepetitionClassReference
```

16.4.2 La "RepetitionStructure" identifica la presencia en una codificación de ocurrencias repetidas de la "EncodingStructure" de la producción. La construcción "Size" opcional (véase 16.2.9) especifica los límites impuestos al número de repeticiones. Los mecanismos utilizados para identificar cuántas repeticiones de la "EncodingStructure" están presentes en una codificación son especificados por un objeto de codificación de la clase "RepetitionClass". La "DefinedEncodingClass" será una clase en la categoría repetición (véase 16.1.10).

16.4.3 La "RepetitionStructure" es un constructor de codificación: cuando se aplica un objeto de codificación a esta estructura como se especifica en 13.2, la codificación de la "RepetitionClass" determina los mecanismos de determinación del número de repeticiones, y el punto de aplicación pasa entonces a la "EncodingStructure" de la producción.

NOTA – En esta construcción se utilizan los caracteres "**{**" y "**}**", que no están en cambio en la construcción "**SEQUENCE OF**" ASN.1 conexa. Se ha hecho así para evitar ambigüedades sintácticas en la definición de estructuras.

16.5 Estructura de codificación de concatenación

16.5.1 La producción "ConcatenationStructure" es:

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

ConcatenationClass ::=
        DefinedEncodingClass        |
        ConcatenationClassReference

ConcatComponents ::=
        ConcatComponent "," *

ConcatComponent ::=
        NamedField
        ConcatComponentPresence ?

ConcatComponentPresence ::=
        OPTIONAL-ENCODING
        OptionalClass

OptionalClass ::=
        DefinedEncodingClass        |
        OptionalityClassReference
```

16.5.2 La "ConcatenationStructure" identifica la presencia en una codificación de cero o una codificaciones de cada una de las "EncodingStructure" en sus "NamedField". La "DefinedEncodingClass" de la "ConcatenationClass" será una clase en la categoría concatenación (véase 16.1.9), y la "DefinedEncodingClass" de la "OptionalClass" será una clase en la categoría opcionalidad (véase 16.1.3).

16.5.3 Si "ConcatComponentPresence" está ausente en un "Component", la "EncodingStructure" en el campo denominado aparecerá exactamente una vez en la codificación.

16.5.4 Si "ConcatComponentPresence" está presente, el mecanismo utilizado para determinar si hay una codificación de la "EncodingStructure" correspondiente lo especifica el objeto de codificación que codifica la "OptionalClass".

16.5.5 El orden en que aparecen las codificaciones en cada "NamedField" de una codificación de la concatenación (y la manera de identificar qué "NamedField" representa una codificación) lo determina un objeto de codificación de la clase "ConcatenationClass".

16.5.6 La "ConcatenationStructure"es un constructor de codificación: cuando se aplica un objeto de codificación a esta estructura como se especifica en 13.2, la codificación de la "ConcatenationClass" determina los procedimientos de concatenación, y el punto de aplicación pasa entonces a cada una de las "EncodingStructure" en sus campos denominados.

17 Asignaciones de objeto de codificación

17.1 Generalidades

17.1.1 La producción "EncodingObjectAssignment" es:

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

17.1.2 La"EncodingObjectAssignment" define la "encodingobjectreference" como una referencia de objeto de codificación al "EncodingObject", que ha de ser una producción que genera un objeto de la clase de codificación "DefinedOrBuiltinEncodingClass". (En D.1.2.2, D.1.7.3 y D.1.8.2 se dan ejemplos de asignación de objeto de codificación para las diferentes construcciones sintácticas de "EncodingObject" que se especifican más adelante).

17.1.3 Se dice que la "DefinedOrBuiltinEncodingClass" es el gobernador de la notación "EncodingObject" de esta producción.

NOTA 1 – Siempre que aparece la producción "EncodingObject" en la ECN hay un gobernador, y la sintaxis de la notación gobernada depende de la clase de codificación del gobernador.

NOTA 2 – La sintaxis de la notación gobernada se ha diseñado de tal manera que un analizador pueda encontrar su final sin conocimiento del gobernador.

17.1.4 No habrá definición recursiva (véase 3.2.38) de una "encodingobjectreference", y no habrá instanciación recursiva (véase 3.2.39) de una "encodingobjectreference".

17.1.5 La producción "EncodingObject" es:

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

17.1.6 "DefinedEncodingObject" define un objeto de codificación y se especifica en 10.9.2. El "DefinedEncodingObject" será de la misma clase de codificación que el gobernador, o de una clase que pueda obtenerse a partir del gobernador por desreferenciación. La "encodingobjectreference" que se define exhibe un asa de identificación si, y solamente si, el "DefinedEncodingObject" exhibe ese asa de identificación.

17.1.7 En la presente Recomendación | Norma Internacional, deberá entenderse que "la misma clase de codificación" y "la misma clase" significan que la notación utilizada para definir las dos clases será el mismo nombre de referencia de clase de codificación, o serán nombres de referencia que desreferencian al mismo nombre de clase de codificación.

17.1.8 Las producciones restantes de "EncodingObject" se definen en las cláusulas que siguen y dan maneras alternativas de definir objetos de codificación de la clase gobernante:

DefinedSyntax	17.2 con las cláusulas 20 a 25
EncodeWith	17.3
EncodeByValueMapping	17.4
EncodeStructure	17.5
DifferentialEncodeDecodeObject	17.6
EncodingOptionsEncodingObject	17.7
NonECNEncodingObject	17.8

17.2 Codificación con una sintaxis definida

17.2.1 La producción "DefinedSyntax" se especifica en 11.5 y 11.6 de la Rec. UIT-T X.681 | ISO/CEI 8824-2, modificada por B.16 de la presente Recomendación | Norma Internacional, y se utiliza para definir objetos de codificación de una clase de codificación gobernante. La sintaxis detallada que se requiere se especifica en las cláusulas 23 a 25, y la semántica de los constructivos se especifica en la cláusula 22.

17.2.2 La notación de la definición de objetos de codificación sólo está disponible para las clases de codificación gobernantes en las categorías (o de las clases) indicadas en el cuadro 3 que sigue. La sintaxis que se ha de utilizar con cada objeto de codificación es la "DefinedSyntax" de la categoría o clase de codificación correspondiente (especificada en las cláusulas 23 a 25).

NOTA 1 – La utilización de esta sintaxis hace necesaria con frecuencia la inclusión de un parámetro para un determinante. Los objetos de codificación parametrizados con esos parámetros (incluidos posiblemente como parte de un conjunto de objetos de codificación parametrizados) sólo son de utilidad cuando se aplican a una estructura de codificación en un EDM, o se incluyen como objetos de codificación que se han de aplicar formando parte de una acción de sustitución. No se pueden aplicar en el ELM.

NOTA 2 – Esta notación permite a los usuarios especificar objetos de codificación que codifican **`#SET`** de la misma manera que las PER codifican normalmente **`#SEQUENCE`**, y viceversa. Es de esperar que los usuarios sean responsables cuando utilicen esta notación.

Cuadro 3 – Categorías y clases soportadas por una sintaxis definida

null category
boolean category
integer category
bitstring category
octetstring category
characterstring category
pad category
alternatives category
repetition category
concatenation category
optionality category
`#CONDITIONAL-INT` class
`#CONDITIONAL-REPETITION` class
tag category
`#TRANSFORM` class
`#OUTER` class

17.2.3 La información requerida (y la sintaxis que se ha de utilizar) para especificar un objeto de codificación de una de estas categorías o clases utilizando "DefinedSyntax" es especificada por las definiciones de las cláusulas 23 a 25.

17.2.4 Si es preciso utilizar un gobernador para un valor de uno de los campos que aparecen en la "DefinedSyntax" de una lista de parámetros ficticios, se empleará la notación "EncodingClassFieldType" (especificada en B.17). No se hará ningún otro uso de la notación "EncodingClassFieldType".

17.2.5 Cuando la sintaxis definida en la cláusula 23 requiere la provisión de una "**`REFERENCE`**", ésta sólo puede ser suministrada en la construcción "**`DefinedSyntax`**" utilizando un parámetro ficticio del objeto de codificación que se define o, en el caso de "**`flag-to-be-used`**" o "**`flag-to-be-set`**", empleando un nombre de referencia que esté presente en la definición de una estructura de sustitución.

17.2.6 La notación "DefinedSyntax" especifica si la "encodingobjectreference" que se define exhibe un asa de identificación.

17.3 Codificación con conjuntos de objetos de codificación

17.3.1 La producción "EncodeWith" es:

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

17.3.2 "CombinedEncodings" y su aplicación a una clase de codificación se especifican en la cláusula 13.

17.3.3 El objeto de codificación definido por la producción "EncodeWith" es la aplicación de las "CombinedEncodings" a la clase de codificación gobernante (véase 17.1.3) de la notación "EncodeWith".

17.3.4 Si no se produce una especificación de codificación completa de la clase gobernante, se trata de un error de la especificación.

17.3.5 Si un conjunto de objetos de codificación de "CombinedEncodings" se parametriza con un parámetro que es una **REFERENCE**, el parámetro real suministrado en esta construcción sólo puede ser un parámetro ficticio del objeto de codificación que se define.

17.3.6 En la aplicación de codificaciones especificada en la cláusula 13 hay un objeto de codificación (digamos, A) que produce el primer campo de bits de la codificación resultante. La "encodingobjectreference" que se define exhibe un asa de identificación si, y solamente si, el objeto de codificación A exhibe ese asa de codificación.

17.4 Codificación utilizando establecimientos de la correspondencia de valores

17.4.1 La producción "EncodeByValueMapping" es:

```
EncodeByValueMapping ::=
    "{"
    USE
    DefinedOrBuiltinEncodingClass
    MAPPING
    ValueMapping
    WITH
    ValueMappingEncodingObjects
    "}"

ValueMappingEncodingObjects ::=
     EncodingObject                          |
     DefinedOrBuiltinEncodingObjectSet
```

17.4.2 La producción "DefinedOrBuiltinEncodingClass" y su semántica se definen en 10.9.1. Será una estructura de codificación definida por el usuario o una clase incorporada en el grupo de categorías campo de bits (véase 16.1.7).

17.4.3 La producción "ValueMapping" se especifica en 19.1.7, y será un establecimiento de la correspondencia entre los valores asociados a la clase de codificación gobernante y la clase identificada por la producción "DefinedOrBuiltinEncodingClass". La clase de codificación gobernante será una clase en el grupo de categorías campo de bits.

17.4.4 La producción "ValueMappingEncodingObjects" especifica la codificación de la "DefinedOrBuiltinEncodingClass". "EncodingObject" definirá un objeto de codificación utilizando la notación gobernada por esa clase, o por una clase a la que pueda ser desreferenciada (véase 17.1.3). Se puede utilizar de manera alternativa "DefinedOrBuiltinEncodingObjectSet" para especificar la codificación de la "DefinedOrBuiltinEncodingClass" y contendrá objetos de codificación suficientes para especificar por completo la codificación de esa clase mediante la aplicación de codificaciones especificada en la cláusula 13.

17.4.5 La sintaxis de "EncodingObject" permite tanto la definición en línea de objetos de codificación (aplicación recursiva de esta cláusula) como la utilización de nombres de referencia. (En D.2.9.3 se da un ejemplo de definición en línea para efectuar los establecimientos de la correspondencia de dos valores en una sola asignación).

17.4.6 Cuando "EncodingObject" requiere la provisión de una **REFERENCE**, ésta sólo puede ser suministrada en esa construcción utilizando un parámetro ficticio del objeto de codificación que se define.

17.4.7 Cuando hay límites o constricciones de tamaño efectivo impuestos a los campos de "DefinedOrBuiltinEncodingClass" y las especificaciones de la cláusula 19 exigen que se hagan corresponder valores con esos campos de un modo tal que no se respetan los límites especificados o las constricciones de tamaño efectivo, no se establece la correspondencia de dichos valores y no es posible la codificación de los mismos. Si esos valores son presentados para su codificación, se trata de un error de la especificación ECN o de la aplicación.

17.4.8 Si se utiliza la alternativa "EncodingObject" de "ValueMappingEncodingObjects", la "encodingobjectreference" que se define exhibe un asa de identificación si, y solamente si, "EncodingObject" exhibe ese asa de identificación. Si se utiliza la alternativa "DefinedOrBuiltinEncodingObjectSet" de "ValueMappingEncodingObjects" para definir la codificación de "DefinedOrBuiltinEncodingClass", la determinación de si "encodingobjectreference" exhibe o no un asa de identificación se hace de acuerdo con 17.3.6.

17.5 Codificación de una estructura de codificación

17.5.1 La producción "EncodeStructure" es:

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

StructureEncoding ::=
                STRUCTURED WITH
               TagEncoding ?
                    EncodingOrUseSet

TagEncoding ::= "[" EncodingOrUseSet "]"

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

17.5.2 La "EncodeStructure" sólo puede ser utilizada para definir una codificación si la clase de codificación gobernante desreferencia a una construcción definida utilizando un constructor de codificación en las categorías alternativas, concatenación o repetición, o a una construcción definida utilizando una de esas categorías precedida por una clase en la categoría rótulo. A este constructor de codificación se le denomina constructor de codificación gobernante.

17.5.3 Si la producción "StructureEncoding" está presente, definirá una codificación del constructor de codificación gobernante y de cualquier clase precedente en la categoría rótulo anterior al constructor de codificación gobernante. Si la producción está ausente, estará presente "CombinedEncodings" y contendrá objetos de codificación que pueden codificar el constructor de codificación gobernante y cualquier clase precedente en la categoría rótulo; de otro modo, se trata de un error de la especificación ECN.

NOTA – "CombinedEncodings" tiene que estar presente si la "StructureEncoding" está ausente, porque se ha de producir una codificación completa. Si se desea diferir la especificación de parte de una codificación, deberá utilizarse un parámetro ficticio.

17.5.4 El objeto de codificación aplicado al constructor de codificación gobernante (ya sea desde **`STRUCTURED WITH`** o desde "CombinedEncodings") no especificará ninguna acción de sustitución.

17.5.5 Si el "EncodingOrUseSet" de la "StructureEncoding" es un "EncodingObject", será gobernado por el constructor de codificación gobernante.

17.5.6 Si **`USE-SET`** está especificado en cualquier "EncodingOrUseSet", la codificación de la clase correspondiente se obtiene aplicando las "CombinedEncodings", que estarán presentes, y bastará codificar la clase correspondiente; de otro modo, se trata de un error de la especificación ECN.

17.5.7 La producción "ComponentEncodingList" es:

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

ComponentEncoding ::=
    NonOptionalComponentEncodingSpec    |
    OptionalComponentEncodingSpec
```

17.5.8 Habrá como máximo una "ComponentEncoding" por cada componente del constructor de codificación gobernante. Las "ComponentEncoding" estarán en el mismo orden textual.

NOTA – La ausencia de las "ComponentEncoding" puede ser detectada siguiendo campos denominados, o por el final de la "ComponentEncodingList".

17.5.9 La producción "OptionalComponetEncodingSpec" será utilizada si, y solamente si, el componente es opcional (es decir, contiene una clase de codificación en la categoría opcionalidad).

17.5.10 Si la "ComponentEncoding" de cualquier componente no está presente en la "ComponentEncodingList", si lo estarán en cambio las "CombinedEncodings" (véase, no obstante, 17.5.6), y es preciso, en la aplicación al componente (véase 13.2), proporcionar una codificación completa de ese componente (incluyendo posiblemente la utilización de parámetros ficticios); de otro modo, se trata de un error de la especificación ECN.

NonOptionalComponentEncodingSpec ::=
identifier ?
TagAndElementEncoding

OptionalComponentEncodingSpec ::=
identifier
TagAndElementEncoding
OPTIONAL-ENCODING
OptionalEncoding

TagAndElementEncoding ::=
TagEncoding ?
EncodingOrUseSet

OptionalEncoding ::= EncodingOrUseSet

17.5.11 El "identifier" será el "identifier" del componente del constructor de codificación gobernante. El "identifier" de "NonOptionalComponentEncodingSpec" será omitido si, y solamente si, el constructor de codificación gobernante es una clase en la categoría repetición para la que no hay identificador en el elemento repetido.

17.5.12 La producción "TagAndElementEncoding" de la "ComponentEncoding" proporcionará una codificación completa del componente (incluyendo cualquier clase en la categoría rótulo que sea un prefijo del elemento, pero excluyendo cualquier clase en la categoría opcionalidad que siga al elemento).

17.5.13 Los "EncodingObject" del "EncodingOrUseSet" en la "TagAndElementEncoding" serán gobernados por las clases de codificación correspondientes del componente. Si un "EncodingOrUseSet" es "USE-SET", la codificación se obtiene aplicando las "CombinedEncodings" (que estarán presentes).

17.5.14 El "EncodingOrUseSet" de la "OptionalEncoding" codificará por completo la clase en la categoría opcionalidad del componente. Si un "EncodingOrUseSet" es **`USE-SET`**, la codificación de la clase en la categoría opcionalidad se obtiene aplicando las "CombinedEncodings" (que estarán presentes).

17.5.15 Si se necesita una **`REFERENCE`** como el parámetro real de cualquiera de los objetos de codificación o conjuntos de objetos de codificación utilizados en esta producción, puede ser suministrada como un parámetro de referencia del objeto de codificación que se define o bien como cualquiera de los "identifier" que están presentes textualmente en la construcción obtenida desreferenciando el gobernador. Si el constructor de codificación gobernante es una clase en la categoría repetición, el parámetro real para la **`REFERENCE`** puede ser un identificador que está presente textualmente en la definición de la "EncodingStructure" en la "RepetitionStructure" de la repetición. Si se requiere que la **`REFERENCE`** identifique un contenedor, puede ser suministrada también como:

a) **`STRUCTURE`** (siempre que el constructor de la estructura que se codifica no sea una categoría alternativas) cuando se refiere a esa estructura;

b) **`OUTER`** cuando se refiere al contenedor de la codificación completa.

NOTA – La "EncodeStructure" es la única producción en la que se pueden suministrar las **`REFERENCE`**, salvo si se utilizan parámetros ficticios o se utiliza **`OUTER`** o cuando las referencias admiten **`flag-to-be-used`** o **`flag-to-be-set`** en la definición de un objeto de codificación de una clase en la categoría repetición que utiliza sustitución.

17.5.16 La determinación de si la "encodingobjectreference" que se define exhibe o no un asa de identificación se hace de acuerdo con 17.3.6.

17.6 Codificación-decodificación diferencial

17.6.1 La producción "DifferentialEncodeDecodeObject" es:

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

SpecForEncoding ::= EncodingObject

SpecForDecoders ::= EncodingObject
```

17.6.2 "DifferentialEncodingObject" especifica reglas de codificación de valores abstractos asociados a la clase del gobernador de esta notación y (por separado) reglas que han de ser utilizadas por los decodificadores para recuperar valores abstractos de codificaciones que se supone han sido producidas por objetos de codificación de la clase del gobernador.

17.6.3 La "SpecForEncoding" será aplicada por los codificadores. Los decodificadores decodificarán como si el codificador hubiera aplicado la "SpecForDecoders".

NOTA 1 – La "SpecForDecoders" sigue siendo una especificación de codificación. Indica a los decodificadores que tienen que suponer que los codificadores han utilizado esta especificación.

NOTA 2 – El comportamiento de los decodificadores que decodifican en el supuesto de que un codificador haya utilizado la "SpecForDecoders", pero detectan errores de codificación, no está normalizado.

17.6.4 Los objetos de codificación "SpecForEncoding" y "SpecForDecoders" no deberán ser definidos utilizando **`ENCODE-DECODE`**, y ningún objeto de codificación empleado en su definición deberá ser definido utilizando **`ENCODE-DECODE`**.

NOTA – Esta limitación está presente porque, de otro modo, la especificación del significado de la construcción codificación/decodificación resultaría más compleja sin ninguna funcionalidad añadida.

17.6.5 La "encodingobjectreference" que se define exhibe un asa de identificación si, y solamente si, la misma asa de identificación es exhibida por la "SpecForEncoding" y la "SpecForDecoders".

17.7 Opciones de codificación

17.7.1 La producción "EncodingOptionsEncodingObject" es:

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

EncodingOptionsList ::= OrderedEncodingObjectList

AlternativesEncodingObject ::= EncodingObject
```

17.7.2 La producción "EncodingOptionsEncodingObject" especifica que el codificador puede codificar (a reserva de 17.7.5) utilizando cualquiera de los "EncodingObject" de la "EncodingOptionsList". Estos "EncodingObject" serán, todos ellos, objetos de codificación de la clase gobernante.

NOTA – Se recomienda encarecidamente que las nuevas implementaciones codifiquen utilizando el primero de los "EncodingObject" de la lista ordenada que sea capaz de codificar el valor abstracto que ha de ser codificado (véase 17.7.5). La especificación de opciones de codificación se da solamente porque se necesita para reflejar las opciones proporcionadas en los protocolos de legado y para sustentar diferentes formas de codificación de la longitud de las cadenas. Cuando se decodifica pueden estar presentes, naturalmente, todas las opciones de codificación.

17.7.3 La producción "AlternativesEncodingObject" será un objeto de codificación de cualquier clase en la categoría alternativas, y los codificadores y decodificadores utilizarán las especificaciones y los procedimientos empleados por ese objeto de codificación como si las opciones de codificación fueran codificaciones de componentes de una instancia de esa clase. La "AlternativesEncodingObject" no contendrá una especificación **`REPLACE`** (véase 23.1.1). El parámetro **`DETERMINED BY`** se fijará en **`handle`**.

NOTA – Si la "AlternativesEncodingObject" se parametriza con un parámetro de campo de referencia, la "encodingobjectreference" que se define ha de ser parametrizada con un parámetro de campo de referencia ficticio utilizado como el parámetro real de la "AlternativesEncodingObject".

17.7.4 Todos los "EncodingObject" de la "EncodingOptionList" exhibirán ese asa de identificación.

17.7.5 El codificador limitará su elección de los "EncodingObject" de la "EncodingOptionList" a los que proporcionan las codificaciones del valor abstracto real que se codifica. Si no hay al menos uno de esos "EncodingObject" para cualquier valor abstracto que se tenga que codificar, se trata de un error de la especificación ECN o de la aplicación.

NOTA 1 – Es posible que los conjuntos de valores abstractos codificados por los "EncodingObject" de la "EncodingOptionsList" estén separados. No se trata de un error, y puede ser una manera conveniente de especificar estructuras diferentes para la codificación de gamas distintas de valores abstractos de la clase gobernante, por ejemplo, codificaciones de formato acortado y formato alargado cuando el formato acortado es obligatorio para valores pequeños.

NOTA 2 – Es posible utilizar un objeto de codificación de opciones de codificación tal como "SpecForDecoders" (véase 17.6), cuando "SpecForEncoding" es un objeto de codificación de opciones de codificación que contiene exactamente una de las opciones de "SpecForDecoders". Esta es otra manera de plantear la extensibilidad.

17.8 Definición no ECN de objetos de codificación

17.8.1 La producción "NonECNEncodingObject" es:

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

17.8.2 La "NonECNEncodingObject" especificará un objeto de codificación de la clase gobernante (véase 17.1.3). La notación utilizada a tal fin está contenida en "anystringexceptnonecnend" y no está normalizada.

17.8.3 La producción "AssignedIdentifier" y su semántica se definen en 12.1 de la Rec. UIT-T X.680 | ISO/CEI 8824-1, modificada por A.1 de la presente Recomendación | Norma Internacional. Identifica la notación utilizada en "anystringexceptuserdefinedend" para especificar la codificación.

17.8.4 Si se utiliza la alternativa "empty" de "AssignedIdentifier", la notación se determina por medios que quedan fuera del alcance de esta Recomendación | Norma Internacional.

17.8.5 La asignación de identificadores de objeto a cualquier notación para utilizarlos en "anystringexceptnonecnend" sigue las reglas normales de la asignación de identificadores de objeto que se especifican en la serie de Recomendaciones UIT-T X.660 | ISO/CEI 9834.

17.8.6 La "encodingobjectreference" que se define exhibe un asa de identificación si, y solamente si, "anystringexceptnonecnend" especifica que se haga así. La manera de aplicar esa especificación no se define en la presente Recomendación | Norma Internacional.

18 Asignaciones de conjunto de objetos de codificación

18.1 Generalidades

18.1.1 La producción "EncodingObjectSetAssignment" es:

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

EncodingObjectSet ::=
     DefinedOrBuiltinEncodingObjectSet |
     EncodingObjectSetSpec
```

18.1.2 La notación "EncodingObjectSet" es gobernada por la palabra reservada **#ENCODINGS** y cumplirá las condiciones que se indican a continuación.

18.1.3 No habrá definición recursiva (véase 3.2.38) de una "encodingclassreference" y no habrá instanciación recursiva (véase 3.2.39) de una "encodingclassreference".

18.1.4 "DefinedOrBuiltinEncodingObjectSet" se define en 10.9.3.

18.1.5 La producción "EncodingObjectSetSpec" es:

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

EncodingObjects ::=
    DefinedEncodingObject |
    DefinedEncodingObjectSet

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

18.1.6 "EncodingObjectSetSpec" define un conjunto de objetos de codificación utilizando uno o más objetos de codificación o conjuntos de objetos de codificación.

18.1.7 Los objetos de codificación que forman un conjunto de objetos de codificación serán de clases de codificación distintas, que no estarán en el grupo de categorías procedimiento de codificación a menos que se trate de la clase **#OUTER** (véase 16.1.13).

NOTA –Se utiliza un conjunto de objetos de codificación para definir otros conjuntos de objetos de codificación, para definir objetos de codificación en el EDM y para importar en el ELM a efectos de la aplicación de codificaciones.

18.1.8 Si la "CompletionClause" está presente, se considera que el conjunto de objetos de codificación definido por "EncodingObjectSetSpec" es "PrimaryEncodings" (véase 13.2), y el conjunto de objetos de codificación asignado a la "encodingobjectsetreference" es el conjunto de objetos de codificación formado como se especifica en 13.2.

18.2 Conjuntos de objetos de codificación incorporados

18.2.1 La producción "BuiltinEncodingObjectSetReference" es:

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

18.2.2 Estos nombres de conjuntos de objetos de codificación hacen referencia a los conjuntos de objetos de codificación definidos por la Rec. UIT-T X.690 | ISO/CEI 8825-1 y la Rec. UIT-T X.691 | ISO/CEI 8825-2. Los identificadores de objetos para las reglas de codificación que proporcionan estos conjuntos de objetos de codificación se dan en el cuadro 4.

NOTA – Esas Recomendaciones | Normas Internacionales se elaboraron antes que la presente Recomendación | Norma Internacional relativa a la ECN, y no utilizan la terminología de los objetos de codificación. Definen, por ejemplo, la manera en que se ha de codificar un tipo ASN.1 **INTEGER** o **BOOLEAN**, lo que deberá interpretarse como la definición de un objeto de codificación de la clase **#INTEGER** o **#BOOLEAN**.

Cuadro 4 – Nombres de conjuntos de objetos de codificación incorporados e identificadores de objetos asociados

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

18.2.3 Cada uno de estos conjuntos de objetos de codificación es un conjunto completo de objetos de codificación que se puede aplicar a cualquier estructura de codificación (generada implícitamente a partir de un tipo ASN.1 o definida por el usuario) para especificar las codificaciones BER o PER correspondientes.

NOTA – Un objeto de codificación definida por el usuario o generada implícitamente podrá añadirse a dicho conjunto, y tendrá prioridad sobre cualquier codificación obtenida por desreferenciación.

18.2.4 Todos los conjuntos anteriores contienen objetos de codificación de las clases utilizadas en estructuras de codificación generadas implícitamente (véase 11.2) que son diferentes para cada conjunto de reglas de codificación. Cada uno de ellos contiene además objetos de codificación idénticos de las clases **#INT**, **#BOOL**, **#NUL**, **#CHARS**, **#OCTET**, **#BITS** y **#CONCATENATION**. Esos conjuntos **no** contienen objetos de codificación de **#ALTERNATIVES**, **#REPETITION** y **#PAD**.

18.2.5 Estas clases de codificación representan bloques básicos de construcción de codificaciones, y son codificadas simplemente por todos los conjuntos de objetos de codificación incorporados anteriores. Los objetos de codificación de estas clases especifican las codificaciones como sigue:

18.2.5.1 La clase **#INT** se codifica como una codificación **PER-BASIC-UNALIGNED #INTEGER**, siempre que esté limitada. Si la **#INT** no tiene un límite superior y un límite inferior cuando se le aplica este objeto de codificación, se trata de un error de la especificación ECN.

18.2.5.2 Las clases **#BOOL** y **#NUL** se codifican como **PER-BASIC-UNALIGNED #BOOLEAN** y **#NULL**, respectivamente.

18.2.5.3 Las clases **#CHARS**, **#OCTETS** y **#BITS** se codifican como **PER-BASIC-UNALIGNED UTF8String**, **#OCTET-STRING** y **#BIT-STRING**, respectivamente, siempre que sean de un tamaño único. Si **#CHARS**, **#OCTETS** o **#BITS** no tienen una constricción de tamaño efectivo que las limite a un solo tamaño, se trata de un error de la especificación ECN.

18.2.5.4 La clase **#CONCATENATION** se codifica como una codificación **PER-BASIC-UNALIGNED** de una **#SEQUENCE** sin componentes opcionales. Si estos objetos de codificación se aplican a una **#CONCATENATION** con componentes opcionales, se trata de un error de la especificación ECN.

18.2.6 Los objetos de codificación **#OPEN-TYPE** de los conjuntos de objetos de codificación incorporados BER, CER y DER no producen ninguna codificación adicional de la clase **#OPEN-TYPE**. La aplicación de estos objetos de codificación a una clase en la categoría tipo abierto es un error de la especificación ECN si las especificaciones de los valores del tipo elegido (en una instancia de comunicación) para utilizarlo con la clase **#OPEN-TYPE** no son autodelimitantes.

NOTA – El conjunto de objetos de codificación combinados aplicado al tipo elegido para utilizarlo con la clase **#OPEN-TYPE** es siempre el mismo que el conjunto de objetos de codificación combinados aplicado a la clase **#OPEN-TYPE** (véase 13.2.10.5).

19 Establecimiento de la correspondencia de valores

19.1 Generalidades

19.1.1 Esta cláusula especifica la sintaxis del establecimiento de la correspondencia entre los valores (y números de rótulos) que han de ser codificados por los campos de una estructura de codificación (que puede ser una estructura de codificación generada o cualquier otra estructura de codificación) y los campos de otra estructura de codificación.

NOTA – Se ha limitado la potencia proporcionada en una sola utilización de esta notación (para evitar complejidades). Se pueden lograr establecimientos de la correspondencia más complejos utilizando múltiples instancias de "EncodeByValueMapping" (véase 17.4 y el ejemplo de D.1.10.2). Estos mecanismos de establecimiento de la correspondencia se pueden ampliar y generalizar, pero no se hará eso a menos que se identifiquen otros requisitos de usuario.

19.1.2 Al especificar la notación "EncodeByValueMapping" (véase 17.4.1), la estructura a la que desreferencia la "DefinedOrBuiltinEncodingClass" de la "EncodingObjectAssignment" (véase 17.1.1) de la que forma parte se denomina gobernador origen o clase de codificación origen (dependiendo del contexto). La clase a la que desreferencia la propia "DefinedOrBuiltinEncodingClass" de "EncodeByValueMapping" se denomina gobernador destino o clase de codificación destino (dependiendo del contexto).

19.1.3 Si el gobernador origen tiene una clase inicial en la categoría rótulo, el gobernador destino tendrá una clase inicial en la categoría rótulo y el número de rótulo de la clase del gobernador origen se hace corresponder con el número de rótulo de la clase en la categoría rótulo del gobernador destino. Si la clase en la categoría rótulo del gobernador destino tiene un número de rótulo asociado y dicho número de rótulo difiere de aquel cuya correspondencia se está estableciendo desde el gobernador origen, se trata de un error de la especificación ECN.

19.1.4 Si el gobernador origen no tiene una clase inicial en la categoría rótulo, no es preciso que el gobernador destino tenga una clase inicial en la categoría rótulo, pero si la tiene, habrá un número de rótulo asociado a ese rótulo en la definición del gobernador destino.

19.1.5 El efecto de la presencia de una clase inicial en la categoría rótulo de los gobernadores origen o destino lo determinan por completo 19.1.3 y 19.1.4, y el texto que sigue ignora la posible presencia de esas clases.

19.1.6 Las codificaciones especificadas para valores cuya correspondencia se ha establecido con la clase de codificación destino pasan a ser las codificaciones de esos valores en la clase de codificación origen.

NOTA 1 – Si la especificación ECN total sólo establece la correspondencia entre algunos de los valores de un tipo ASN.1 y las codificaciones, no se trata de un error. Es una constricción impuesta por la ECN a los valores que pueden ser utilizados por la aplicación. Esas constricciones deberán ser señaladas normalmente mediante un comentario en la especificación ASN.1 o en la especificación ECN (véase 17.4.7).

NOTA 2 – Si la especificación ECN total establece la correspondencia entre dos valores y la misma codificación producida por un solo objeto de codificación, se trata de un error de la codificación ECN. Esos errores pueden ser detectados por herramientas ECN, pero en la presente Recomendación UIT-T | Norma Internacional todavía no se han establecido reglas para evitarlos, y la responsabilidad al respecto es del usuario de la ECN.

19.1.7 La producción "ValueMapping" es:

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

NOTA – Todas las ocurrencias de esta sintaxis van precedidas por la palabra reservada **`MAPPING`**. (En D.1.2.2, D.1.4.2, D.1.10.2 y D.2.1.3 y en el anexo E se dan ejemplos de codificaciones utilizando cada uno de esos establecimientos de la correspondencia de valores).

19.1.8 Las producciones "ValueMapping" se especifican en las cláusulas que se indican a continuación:

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

NOTA – Con frecuencia ocurre que se pueden utilizar varios establecimientos de la correspondencia de valores para definir la misma codificación, pero algunos producirán una especificación más evidente o con menos verbosidad que otros. Los diseñadores de la ECN deberán seleccionar con cuidado la manera de establecer la correspondencia de valores que se va a utilizar.

19.2 Establecimiento de la correspondencia mediante valores explícitos

19.2.1 Esta subcláusula contiene la notación para la especificación del establecimiento de la correspondencia de valores entre diferentes clases de codificación de campo de bits primitivas. (En D.1.10.2 se da un ejemplo al respecto).

19.2.2 Esta subcláusula utiliza la notación de los valores de la ASN.1 (notación de valor ASN.1) especificada en la Rec. UIT-T X.680 | ISO/CEI 8824-1 para el tipo que corresponde a una clase de codificación.

19.2.3 El cuadro 5 especifica la notación del valor ASN.1 que se ha de utilizar con cada clase de codificación gobernante. En cada caso, la clase puede tener o no asociada una constricción de tamaño o de gama de valores.

19.2.4 La ECN admite el establecimiento de la correspondencia mediante valores explícitos (hacia o desde la clase de codificación) para todas las clases de codificación en las categorías indicadas en la columna 1 del cuadro 5. La columna 2 del cuadro especifica la notación de valor (como una producción ASN.1 o por referencia a una cláusula de la Rec. UIT-T X.680 | ISO/CEI 8824-1 o ambas cosas) que se utilizará cuando se especifique como gobernante de la notación una clase de codificación en la categoría indicada en la columna 1. También especifica la cláusula de la Rec. UIT-T X.680 | ISO/CEI 8824-1 en que se define la notación del valor.

NOTA – Ninguna de las notaciones de valor ASN.1 que siguen puede utilizar los "DefinedValue" (definidos en 13.1 de la Rec. UIT-T X.680 | ISO/CEI 8824-1) porque las "valuereference" no pueden ser importadas ni definidas en un módulo EDM o ELM.

Cuadro 5 – Categorías de clase de codificación y notaciones de valor utilizadas en el establecimiento de la correspondencia mediante valores explícitos

Category of governing encoding class	ASN.1 value notation
bitstring	**"bstring" or "hstring"** (véanse 11.10 y 11.12 de la Rec. UIT-T X.680 | ISO/CEI 8824-1)
boolean	**"BooleanValue"** (véase 17.3 de la Rec. UIT-T X.680 | ISO/CEI 8824-1)
characterstring	**"RestrictedCharacterStringValue"** (véase 37.8 de la Rec. UIT-T X.680 | ISO/CEI 8824-1)
integer	**"SignedNumber"** (véase 18.1 de la Rec. UIT-T X.680 | ISO/CEI 8824-1)
null	**"NullValue"** (véase 23.3 de la Rec. UIT-T X.680 | ISO/CEI 8824-1)
objectidentifier	**"DefinitiveIdentifier"** (véase A.1)
octetstring	**"bstring" or "hstring"** (véanse 11.10 y 11.12 de la Rec. UIT-T X.680 | ISO/CEI 8824-1)
real	**"RealValue"** (véase 20.6 de la Rec. UIT-T X.680 | ISO/CEI 8824-1)

19.2.5 La producción "MappingByExplictValues" es:

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

MappedValues ::=
        MappedValue1
        TO
        MappedValue2

MappedValue1 ::= Value

MappedValue2 ::= Value
```

19.2.6 "MappedValue1" será la notación de valor gobernada por el gobernador origen y "MappedValue2" será la notación de valor gobernada por el gobernador destino (véase 19.1.2). El valor en el origen especificado por "MappedValue1" se hace corresponder con el valor en el destino especificado por "MappedValue2".

19.2.7 Si "MappedValue2" es un valor que no respeta un límite o una constricción de tamaño en el destino, se trata de un error de la especificación ECN.

19.3 Establecimiento de la correspondencia mediante campos concordantes

19.3.1 Este establecimiento de la correspondencia permite sobre todo definir la codificación de un tipo ASN.1 como la codificación de una estructura de codificación que tiene campos correspondientes a los componentes del tipo, pero que también tiene campos añadidos para determinantes.

19.3.2 La producción "MappingByMatchingFields" es:

```
MappingByMatchingFields ::=
    FIELDS
```

19.3.3 Si las clases de codificación origen o destino son estructuras de codificación definidas por el usuario (véase 9.2.2.3) o estructuras de codificación generadas, estas referencias se resuelven hasta el comienzo del origen y el destino con un constructor de codificación. Si este constructor de codificación en el destino está en la categoría repetición, la desreferenciación del componente de este constructor de codificación de repetición se efectúa hasta que el componente comienza con un constructor de codificación. Las referencias dentro de las estructuras resultantes no se resuelven.

19.3.4 El efecto de la posible presencia de clases en la categoría rótulo en la desreferenciación inicial de nombres "DefinedOrBuiltinEncodingClass" en el origen y el destino se especificó por completo en 19.1.3 a 19.1.5. Si se introducen más clases iniciales en la categoría rótulo por aplicación de 19.3.3, se trata de un error de la especificación ECN.

19.3.5 Tras la aplicación de 19.3.3, las clases de codificación origen y destino comenzarán con el mismo constructor de codificación. Será un constructor de codificación en la categoría concatenación o un constructor de codificación en la categoría repetición. Si este constructor de codificación está en la categoría repetición, su componente en destino será una clase en la categoría concatenación. A los efectos de la presente subcláusula 19.3, las estructuras de codificación resultantes se llaman estructuras de codificación origen y destino, respectivamente.

19.3.6 Los campos de nombres de los componentes (de nivel máximo) del constructor de codificación producidos por la aplicación de 19.3.3 al origen se llaman campos origen.

NOTA – Los campos origen se limitan a los campos de nivel máximo de una concatenación o al componente de una repetición. Esta restricción se impone para facilitar la implementación de la ECN, y podría hacerse menos estricta en el futuro.

19.3.7 Los campos de los componentes del constructor de codificación en la categoría concatenación producidos por la aplicación de 19.3.3 al destino se denominan campos destino potenciales.

NOTA – Los campos destino potenciales pueden ser los componentes de una concatenación de nivel máximo o los componentes de una concatenación que es el componente de una repetición.

19.3.8 Para todo campo origen habrá un campo destino potencial con el mismo nombre de campo (el campo destino concordante).

NOTA – El establecimiento de la correspondencia de un componente de una clase repetición sólo es posible si el componente contiene un identificador (que concuerda con un identificador en el destino). La utilización del establecimiento de la correspondencia mediante campos concordantes no se permitiría si el identificador estuviese ausente.

19.3.9 Un campo destino concordante será un elemento opcional en una concatenación si, y solamente si, su campo origen es un elemento opcional en una concatenación, y la presencia o ausencia del campo origen en un valor abstracto asociado a la estructura de codificación origen determina la presencia o la ausencia del campo destino en la estructura de codificación destino.

19.3.10 Si el campo origen tiene una clase inicial en la categoría rótulo, el campo destino concordante tendrá una clase inicial en la categoría rótulo y el número de rótulo de la clase del campo origen se hace corresponder con el número de rótulo de la clase en la categoría rótulo del campo destino concordante. Si la clase en la categoría rótulo del campo destino concordante tiene un número de rótulo asociado y dicho número de rótulo difiere de aquel cuya correspondencia se está estableciendo desde el campo origen, se trata de un error de la especificación ECN.

19.3.11 Si el campo origen no tiene una clase inicial en la categoría rótulo, no es preciso que el campo destino concordante tenga una clase inicial en la categoría rótulo, pero si la tiene, habrá un número de rótulo asociado a ese rótulo en la definición del campo destino concordante.

19.3.12 Aparte de la presencia o ausencia de clases en la categoría rótulo y la categoría opcionalidad (que se especifican en 19.3.9 a 19.3.11), el campo destino concordante y el campo origen tendrán la misma clase concordante (véase 17.1.7) o se definirán utilizando la misma secuencia de elementos de léxico, sin tener en cuenta los comentarios ni los espacios en blanco ni las especificaciones de límites.

19.3.13 Se establece la correspondencia entre todos los valores abstractos de cada uno de los campos origen y los campos destino concordantes. Los campos adicionales de la estructura de codificación destino no adquieren valores abstractos. En una especificación ECN correcta, el valor de esos campos ha de ser especificado por referencia como un determinante.

19.3.14 Si los constructores de codificación origen y destino son clases en la categoría repetición, el número de repeticiones en el valor abstracto asociado a la estructura de codificación origen se hace corresponder con el número de repeticiones en la estructura de codificación destino.

19.3.15 Si un campo origen tiene asociada una constricción de contenido, se establece la correspondencia de la misma con una constricción de contenido asociada al campo destino concordante.

19.3.16 Si, debido a la presencia de constricciones de límites o tamaño, hay valores en algunos campos origen que no están presentes en el campo destino concordante, se aplicará 17.4.7.

19.4 Establecimiento de la correspondencia mediante objetos de codificación `#TRANSFORM`

19.4.1 Este establecimiento de la correspondencia permite aplicar a tal fin uno o más objetos de codificación **`#TRANSFORM`**.

19.4.2 La clase de codificación **`#TRANSFORM`** se define en la cláusula 24. Con ella es posible especificar objetos de codificación que transformarán valores abstractos origen en valores abstractos resultado. Las reglas para formar una lista ordenada de transformadas (para "OrderedTransformList") se especifican en cláusula 24. La lista completa se define para transformar desde un origen en un resultado.

NOTA – En D.1.2.2 y D.2.4.2 se dan ejemplos de establecimiento de la correspondencia definidos con estas transformadas. El ejemplo de D.1.6.3 muestra la utilización de esta producción para definir codificaciones BCD de un entero ASN.1

19.4.3 La producción "MappingByTransformEncodingObjects" es:

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

OrderedTransformList ::= Transform "," +

Transform ::= EncodingObject
```

19.4.4 Todos los "EncodingObject" de la "OrderedTransformList" serán gobernados por la clase de codificación **#TRANSFORM**.

19.4.5 Las clases origen y destino para este establecimiento de la correspondencia (véase 19.1.2) serán de la categoría cadena de bits, booleano, cadena de caracteres, entero o cadena de octetos. El origen de la primera transformada de la lista y el resultado de la última transformada de la lista concordarán con la categoría de las clases origen y destino como se especifica en 24.2.7.

19.4.6 Si cualquier "Transform" de la "OrderedTransformList" no es reversible para el valor abstracto cuya correspondencia se establece, se trata de un error de la especificación ECN o de la aplicación.

NOTA – La cláusula 24 especifica para cada transformada los valores abstractos para los que, por definición, ha de ser reversible.

19.4.7 Si hay límites o constricciones de tamaño efectivo impuestos a la clase de codificación destino, se aplicará 17.4.7.

19.5 Establecimiento de la correspondencia mediante el ordenamiento de valores abstractos

19.5.1 Este establecimiento de la correspondencia permite que valores abstractos asociados a clases de codificación simples sean distribuidos en los campos de estructuras de codificación complejas, y que se hagan corresponder valores abstractos asociados a estructuras de codificación complejas con clases de codificación simples, tales como **#INT**. También permite la compactación de valores enteros o enumeraciones en un conjunto contiguo de valores enteros (véase D.1.4).

NOTA – Los números de rótulo asociados a clases en la categoría rótulo no son valores abstractos.

19.5.2 La producción "MappingByAbstractValueOrdering" es:

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

19.5.3 Para este establecimiento de la correspondencia, todos los nombres de clase de codificación son desreferenciados (recursivamente), y el resultado será una clase en la categoría nulo, booleano, entero o real, o será una constricción definida utilizando una clase en la categoría alternativas, o una clase en la categoría concatenación con un solo componente no opcional.

19.5.4 El conjunto ordenado de valores puede ser finito o infinito.

19.5.4.1 Se define un conjunto finito de valores abstractos ordenados para clases de codificación en las categorías siguientes:

a) nulo;

b) booleano;

c) entero limitado;

d) real limitado a un número finito de valores

e) una estructura de codificación definida utilizando la categoría alternativas, siempre que todas las alternativas tengan un ordenamiento finito definido;

f) una estructura de codificación definida utilizando la categoría concatenación que tiene un solo componente no opcional, siempre que el componente tenga un ordenamiento finito definido.

19.5.4.2 Se define un conjunto infinito de valores abstractos ordenados para clases de codificación en las categorías siguientes:

a) entero, con la restricción de tener un límite inferior finito;

b) una estructura de codificación definida utilizando la categoría alternativas, siempre que todas las alternativas excepto la última se definan de modo que tengan un conjunto finito de valores ordenados, y la última alternativa se defina de modo que tenga un conjunto infinito de valores ordenados;

c) una estructura de codificación definida utilizando la categoría concatenación que tiene un solo componente no opcional, siempre que el componente se defina de modo que tenga un conjunto infinito de valores abstractos ordenados.

19.5.5 Las clases en la categoría nulo tienen un solo valor abstracto. Las clases en la categoría booleano se definen de modo que tengan **TRUE** antes que **FALSE**. Las clases en la categoría entero se definen de modo que tengan los valores enteros superiores después de los valores enteros inferiores. Las clases en la categoría real se definen de modo que tengan los valores superiores después de los valores inferiores.

NOTA – El número de valores abstractos asociados a una clase en la categoría entero no necesariamente es un número finito.

19.5.6 Cualesquiera límites presentes en el origen o en el destino serán tenidos totalmente en cuenta al determinar el conjunto ordenado de valores abstractos.

19.5.7 El ordenamiento de los valores abstractos asociados a una clase en la categoría alternativas (cuyas alternativas tienen, todas ellas, un ordenamiento definido de valores abstractos) se define de modo que sean los valores abstractos (ordenados) de la alternativa primera textualmente, seguidos por los de la alternativa segunda textualmente, y así sucesivamente hasta la alternativa última textualmente.

19.5.8 El ordenamiento de los valores abstractos asociados a una clase en la categoría concatenación que tiene un solo componente no opcional será el determinado por el ordenamiento de los valores abstractos de su único componente.

19.5.9 El establecimiento de la correspondencia se define entre los valores abstractos de la primera clase de codificación y los valores abstractos de la segunda clase de codificación de acuerdo con su posición en el ordenamiento anterior.

19.5.10 Se señala que con las reglas anteriores se asegura la existencia de un primer valor definido en cada ordenamiento, y un valor definido siguiente. No es necesario que haya un valor definido último (cualquiera de los conjuntos o ambos pueden ser infinitos).

19.5.11 Si el número de valores abstractos del ordenamiento del destino es inferior al número de valores abstractos del ordenamiento del origen, no se trata de un error. Sin embargo, la especificación ECN no podrá codificar alguno de los valores abstractos de la especificación ASN.1 y esto debería ser señalado por algún comentario en la especificación ASN.1 o la especificación ECN.

19.5.12 Si el número de valores abstractos del ordenamiento del destino es superior al número de valores abstractos del ordenamiento del origen, puede haber algunas codificaciones definidas por la ECN que no tengan ningún valor abstracto ASN.1, y que nunca serán generadas.

19.5.13 Este establecimiento de la correspondencia se puede aplicar también en todos los casos en que los únicos valores abstractos de la estructura destino son los asociados a una sola instancia de la misma clase que la estructura origen.

NOTA – Esto es lo que ocurriría si la estructura destino fuese la misma que la estructura origen precedida por una o más instancias de clases en la categoría rótulo.

19.5.14 Clases en la categoría rótulo pueden estar presentes en la estructura destino, pero han de tener un número de rótulo asociado especificado en la definición de la estructura. Su presencia no tiene ningún efecto en el establecimiento de la correspondencia de valores abstractos.

19.6 Establecimiento de la correspondencia mediante la distribución de valores

19.6.1 Este procedimiento toma gamas de valores de una clase de codificación en la categoría entero y establece la correspondencia entre cada gama y un campo de entero diferente en una estructura de codificación más compleja. Los valores de los campos que reciben valores no abstractos se determinarán mediante la aplicación de determinantes.

19.6.2 Todos los nombres de estructuras de codificación son desreferenciados (recursivamente) antes de la aplicación de este establecimiento de la correspondencia.

19.6.3 La clase de codificación origen será entonces una clase en la categoría entero, posiblemente con una clase precedente en la categoría rótulo cuya correspondencia se establece de acuerdo con 19.1.3 a 19.1.5.

19.6.4 La clase de codificación destino puede ser cualquier estructura de codificación, y puede contener clases en la categoría rótulo, pero todos los nombres de campos de la estructura de codificación total serán distintos, y todas las clases en la categoría rótulo del destino (excepto aquéllas cuya correspondencia se ha establecido según 19.6.3) tendrán un número de rótulo en su definición y, por otra parte, son ignoradas en el establecimiento de la correspondencia.

19.6.5 Los valores sólo se harán corresponder con campos de la estructura destino que sean clases en la categoría entero, precedidas posiblemente por clases en la categoría rótulo (véase 19.6.4) y posiblemente con límites.

19.6.6 La producción "MappingByValueDistribution" es:

```
MappingByValueDistribution ::=
    DISTRIBUTION
    "{"
    Distribution "," +
    "}"

Distribution ::=
    SelectedValues
    TO
    identifier

SelectedValues ::=
   SelectedValue
   |   DistributionRange
   |   REMAINDER

DistributionRange ::=
     DistributionRangeValue1
    ".."
    DistributionRangeValue2

SelectedValue ::= SignedNumber

DistributionRangeValue1 ::= SignedNumber
DistributionRangeValue2 ::= SignedNumber
```

19.6.7 "SignedNumber" se especifica en 18.1 de la Rec. UIT-T X.680 | ISO/CEI 8824-1.

19.6.8 "DistributionRangeValue1" será inferior a "DistributionRangeValue2".

19.6.9 El valor especificado por "SelectedValue" en "SelectedValues", o el conjunto de valores superiores o iguales a "DistributionRangeValue1" e inferiores o iguales a "DistributionRangeValue2", se hace corresponder con el campo especificado por "Identifier".

19.6.10 La palabra reservada **REMINDER** sólo se utilizará una vez para el último "SelectedValues" y especifica todos los valores abstractos de la clase de codificación origen que no han sido distribuidos por un "SelectedValues" anterior.

19.6.11 No se establecerá la correspondencia entre un valor y más de un campo destino, pero varios "SelectedValues" pueden tener el mismo destino.

19.6.12 Si hay límites impuestos al campo destino, se aplicará 17.4.7.

19.6.13 Si se establece la correspondencia entre un valor del origen y un campo destino cuya presencia depende de la opcionalidad o de la elección de alternativas o de ambas cosas, no se trata de un error, pero la opcionalidad y la elección de alternativas en el destino (cuando se codifican esos valores) serán tales que la codificación del destino incluya el campo destino.

19.7 Establecimiento de la correspondencia entre valores enteros y bits

19.7.1 Este procedimiento toma valores únicos o gamas de valores de una clase de codificación en la categoría entero (posiblemente precedida por clases en la categoría rótulo como se especifica en 19.1.3 a 19.1.5) y establece la correspondencia entre cada valor entero y un valor cadena de bits (precedido posiblemente por clases en la categoría rótulo).

NOTA – Este establecimiento de la correspondencia tiene por objeto el soporte de codificaciones de enteros autodelimitantes, por ejemplo, las codificaciones Huffman. (Véase en el anexo E un análisis más detallado y ejemplos de las codificaciones Huffman).

19.7.2 La clase de codificación origen será una clase en la categoría entero, precedida posiblemente por clases en la categoría rótulo.

19.7.3 La clase de codificación destino será una clase en la categoría cadena de bits, precedida posiblemente por clases en la categoría rótulo.

19.7.4 El establecimiento de la correspondencia de las clases en la categoría rótulo se hace tal como se especifica en 19.1.3 a 19.1.5.

19.7.5 La producción "MappingIntToBits" es:

```
MappingIntToBits ::=
    TO BITS
    "{"
    MappedIntToBits "," +
    "}"

MappedIntToBits ::=
    SingleIntValMap   |
    IntValRangeMap
```

19.7.6 Cada "SingleIntValMap" hace corresponder un solo valor entero con un solo valor cadena de bits.

19.7.7 Cada "IntValRangeMap" hace corresponder una gama de valores enteros contiguos y crecientes con una gama de valores cadena de bits contiguos y crecientes.

19.7.8 Se dice que los valores cadena de bits son contiguos si:

a) Todos ellos tienen la misma longitud en bits.

b) Cuando se interpretan como un valor entero positivo, los valores enteros correspondientes son valores enteros contiguos y crecientes.

19.7.9 Sólo los valores especificados en el establecimiento de la correspondencia son codificables. No es posible establecer la correspondencia de otros valores abstractos del origen, que no pueden ser codificados por el objeto de codificación definido por la asignación de objeto de codificación utilizando este constructivo. Si esos valores son presentados para su codificación, se trata de un error de la especificación ECN o de la aplicación.

NOTA – Esta limitación de la codificación debería estar reflejada por constricciones impuestas al tipo ASN.1 al que se aplica, o mediante comentarios en la especificación ASN.1.

19.7.10 La producción "SingleIntValMap" es:

```
SingleIntValMap ::=
    IntValue
    TO
    BitValue

IntValue ::= SignedNumber
BitValue ::=
     bstring   |
     hstring
```

19.7.11 El "SignedNumber", la "bstring" y la "hstring" se especifican en 18.1, 11.10 y 11.12 respectivamente, de la Rec. UIT-T X.680 | ISO/CEI 8824-1.

19.7.12 El "SingleIntValMap" hace corresponder el valor entero especificado con el valor cadena de bits especificado.

19.7.13 La producción "IntValRangeMap" es:

```
IntValRangeMap ::=
        IntRange
        TO
        BitRange

IntRange ::=
        IntRangeValue1
        ".."
        IntRangeValue2

BitRange ::=
        BitRangeValue1
        ".."
        BitRangeValue2
```

IntRangeValue1 ::= SignedNumber

IntRangeValue2 ::= SignedNumber

BitRangeValue1 ::=
bstring |
hstring

BitRangeValue2 ::=
bstring |
hstring

19.7.14 Las cadenas de bits "BitRangeValue1" y "BitRangeValue2" tendrán el mismo número de bits.

19.7.15 El valor "IntRangeValue2" será superior al valor "IntRangeValue1".

19.7.16 Cuando se interprete como la codificación de un entero positivo (véase 8.3.3 de la Rec. UIT-T X.690 | ISO/CEI 8825-1), "BitRangeValue2" representará un valor entero (digamos, "B") superior al representado por "BitRangeValue1" (digamos, "A") y la diferencia entre los valores enteros correspondientes a "BitRangeValue2" y "BitRangeValue1" ("B"-"A") será igual a la diferencia entre los valores de "IntRangeValue2" e "IntRangeValue1".

19.7.17 La "BitRange" representa el conjunto ordenado de cadenas de bits correspondientes a los valores enteros entre "A" y "B".

19.7.18 El "IntValRangeMap" hace corresponder cada uno de los enteros de la gama especificada con el valor cadena de bits correspondiente de la "BitRange". (En el anexo E se dan ejemplos de un "IntValRangeMap").

19.7.19 Si una "BitRange" incluye un valor que no respeta una constricción de tamaño impuesta al destino, se trata de un error de la especificación ECN.

20 Definición de objetos de codificación utilizando sintaxis definida

20.1 Las cláusulas 21 a 25 especifican la información que se necesita para definir objetos de codificación de cada categoría de clases de codificación y la sintaxis que se ha de utilizar. Se dice que dicha sintaxis es la sintaxis definida y se especifica utilizando la notación de clases de objetos de información de la Rec. UIT-T X.681 | ISO/CEI 8824-2 modificada por el anexo B a la presente Recomendación | Norma Internacional.

20.2 La sintaxis definida de cada categoría se puede utilizar también para definir objetos de codificación de estructuras que son clases en esa categoría, precedidas por una o más instancias de una clase en la categoría rótulo. Cuando en el texto que sigue se requiere que una clase esté en una categoría especificada, se incluye en ese requerimiento el caso de la clase precedida por una clase en la categoría rótulo.

20.3 La notación de clases de objetos de información modificada sólo se utilizará dentro de esta Recomendación | Norma Internacional.

20.4 La utilización de la notación de sintaxis definida para definir objetos de información se especifica en 17.2. La sintaxis definida para definir objetos de codificación será la sintaxis especificada por las declaraciones **WITH SYNTAX** de las cláusulas 23 a 25.

20.5 Las declaraciones **WITH SYNTAX** imponen constricciones a los valores de algunas propiedades de la codificación, junto con los valores de otras propiedades de la codificación, para aplicar algunas (pero no todas las) constricciones semánticas. En el texto se especifican otras constricciones sobre la utilización de las declaraciones **WITH SYNTAX**.

20.6 La sintaxis definida para cada clase de codificación especifica un cierto número de propiedades de codificación a las que se pueden dar valores de los tipos ASN.1 definidos en la cláusula 21 (o en algunos casos con otras clases de codificación y otros objetos de codificación) para proporcionar la información que se necesita en la especificación de un objeto de codificación de esa clase. La información requerida para la definición de un objeto de codificación consiste, por lo general, en una combinación de valores de propiedades de codificación, junto con la instancia particular de la sintaxis definida utilizada para especificar esos valores.

NOTA – Lo anterior difiere de la utilización de una declaración **WITH SYNTAX** en la definición de objetos de información normal, en donde la semántica asociada al objeto de información depende únicamente del conjunto de valores de los campos de la clase de objeto de información y no de la forma de la declaración **WITH SYNTAX** utilizada para fijar esos valores (véase B.15).

20.7 Las propiedades de codificación utilizadas en las cláusulas 23 a 25 actúan conjuntamente en grupos de propiedades de codificación y utilizan valores de tipos ASN.1 para su definición. La cláusula 21 especifica el significado de los valores de los tipos utilizados habitualmente en la especificación de esas propiedades de codificación.

20.8 Parte del texto definitivo de las cláusulas 21 y 22 se ha copiado en las cláusulas 22 a 25. Cuando tal es el caso, el texto copiado **aparece en gris** y se hace una referencia al texto definitivo.

20.9 La cláusula 25 especifica un cierto número de transformadas que se pueden aplicar a valores abstractos. Varios grupos de propiedades de codificación requieren una lista ordenada de las transformadas que han de ser aplicadas por un codificador. Para que la decodificación sea posible, las transformadas aplicadas por un codificador han de poder ser invertidas por un decodificador a fin de recuperar los valores abstractos originales. Las cláusulas 23 y 24 especifican cuándo han de ser reversibles las transformadas, y la cláusula 25 especifica los valores abstractos para los que cualquier transformada es reversible.

21 Tipos utilizados en la especificación de la sintaxis definida

NOTA – En todas las definiciones de tipos ASN.1 que aquí se dan se supone la presencia de rótulos automáticos y la no extensibilidad.

21.1 Tipo `Unit`

21.1.1 El tipo "**`Unit`**" es:

```
Unit ::= INTEGER
       {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
        dword32(32)} (0..256)
```

21.1.2 El valor por defecto de este tipo es siempre **`bit`**.

21.1.3 Una propiedad de codificación de este tipo especifica la unidad en la que se cuentan otras propiedades de codificación o campos de determinante.

21.1.4 El valor de una propiedad de codificación de este tipo está restringido en todos los casos, excepto en uno, a valores distintos de cero. En estos casos, la propiedad de codificación especifica un número de bits. Ese número de bits determina la unidad en la que se cuentan otras propiedades de codificación o campos de determinante.

21.1.5 Cuando se utiliza en la definición de un objeto de codificación de una clase en la categoría repetición, se permite también el valor **`repetitions`**, que especifica que la cuenta asociada da el número de repeticiones en la codificación.

21.2 Tipo `EncodingSpaceSize`

21.2.1 El tipo "**`EncodingSpaceSize`**" es :

```
EncodingSpaceSize ::= INTEGER
      { encoder-option-with-determinant(-3),
        variable-with-determinant(-2),
        self-delimiting-values(-1),
        fixed-to-max(0)} (-3..MAX)
```

21.2.2 El valor por defecto de este tipo es siempre **`self-delimiting-values`**.

21.2.3 Una propiedad de codificación de este tipo especifica el tamaño del espacio de codificación (véase 9.21.5).

21.2.4 Valores positivos (distintos de cero) especifican un tamaño fijo para el espacio de codificación, tal como el valor del tipo "**`Unit`**" multiplicado por el valor del tipo "**`EncodingSpaceSize`**", en bits. Si el valor del tipo "**`Unit`**" es "**`repetitions`**", el tamaño del espacio de codificación puede ser variable (porque el espacio de codificación necesario para cada componente puede ser diferente), pero es siempre ese número fijo de repeticiones, y si se va a de codificar un valor abstracto que no tiene ese número de repeticiones, se trata de un error de la especificación ECN o de la aplicación.

21.2.5 El valor "**`encoder-option-with-determinant`**" especifica que el tamaño del espacio de codificación puede variar de acuerdo con el valor abstracto que se codifica y que el codificador elegirá el tamaño del espacio de codificación, registrando el tamaño elegido en el determinante asociado. En este caso se requiere un valor del tipo "**`EncodingSpaceDeterminant`**" (véase 21.3) o "**`RepetitionSpaceDeterminant`**" (véase 21.7).

NOTA – En este caso (y en el caso de 21.2.6) se requiere un valor del tipo "**`EncodingSpaceDeterminant`**" o "RepetitionSpaceDeterminant" (para determinar el tamaño del espacio de codificación), pero se permite la provisión de un determinante en todos los demás casos para soportar codificaciones (similares a las de BER) que utilizan determinantes de longitud incluso cuando son redundantes. Cualquier diferencia entre los dos determinantes es un error. Es posible, sin embargo, que no siempre se pueda determinar si se trata de un error de la especificación ECN o de un error de la aplicación, pero los codificadores conformes no han de transmitir esas codificaciones.

21.2.6 El valor "**`variable-with-determinant`**" especifica que el tamaño del espacio de codificación puede variar de acuerdo con el valor abstracto que se codifica. En este caso se requiere un valor del tipo "**`EncodingSpaceDeterminant`**" (véase 21.3) o "**`RepetitionSpaceDeterminant`**" (véase 21.7) (para proporcionar una manera precisa de determinar el tamaño del espacio de codificación).

21.2.7 El valor "**self-delimiting-values**" especifica que la codificación del valor es autodelimitante, es decir, que cada valor se codifica en un múltiplo del valor especificado del tipo "**Unit**". No habrá ningún par de valores abstractos para los que la codificación de un valor abstracto sea la primera parte de la codificación del otro valor abstracto.

NOTA – Un decodificador puede determinar (tras la posible determinación de los bits no utilizados y la justificación) el final del espacio de codificación contrastando la codificación de cada uno de los valores abstractos posibles con la codificación que se examina. Precisamente uno de ellos concordará en las codificaciones producidas por un codificador conforme. Los decodificadores pueden desarrollar procedimientos más eficaces, pero son equivalentes.

21.2.8 El valor "**fixed-to-max**" especifica que el espacio de codificación ha de ser el mismo para la codificación de todos los valores abstractos. Especifica además que el tamaño del espacio de codificación ha de ser el múltiplo más pequeño de "**Unit**" que puede contener la codificación especificada de uno cualquiera de los valores abstractos (de todos). Este valor no será utilizado si el valor abstracto que se ha de codificar en el espacio de codificación está asociado a una clase en la categoría concatenación (véase 23.5.2.3) o repetición (véase 23.13.2.5).

NOTA 1 – Un caso especial es aquel en el que hay un solo valor abstracto cuya codificación es de cero bits, lo que da como resultado un espacio de codificación vacío (cero bits).

NOTA 2 – Si esa especificación se aplica cuando no se puede determinar un tamaño máximo (por ejemplo, para codificar un entero sin límites), se trata de un error de la especificación ECN, pero es preciso que los codificadores conformes rehúsen generar codificaciones en tales casos.

21.3 Tipo EncodingSpaceDetermination

21.3.1 El tipo "**EncodingSpaceDetermination**" es:

```
EncodingSpaceDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, container}
```

21.3.2 El valor por defecto de este tipo es siempre "**field-to-be-set**".

21.3.3 Una propiedad de codificación de este tipo especifica la manera en que se determina el espacio de codificación cuando una propiedad de codificación del tipo "**EncodingSpaceSize**" (véase 21.2) se fija en "**variable-with-determinant**" o "**encoder-option-with-determinant**".

21.3.4 El valor "**field-to-be-set**" requiere la especificación de una **REFERENCE** en un campo cuyo valor será fijado por el codificador para llevar información de longitud, y será utilizado por un decodificador. La especificación de la codificación determina cómo tiene que fijar un codificador el valor de este campo a partir del tamaño (en unidades de codificación) del espacio de codificación. Si un campo se fija más de una vez utilizando "**field-to-be-set**" o "**flag-to-be-set**" (véase 21.7) y los diferentes procedimientos de codificación producen valores diferentes, se trata de un error de la especificación ECN o de la aplicación, y en este caso los codificadores no generarán codificaciones.

21.3.5 El valor "**field-to-be-used**" requiere la especificación de una **REFERENCE** en un campo cuyo valor puede ser fijado desde la sintaxis abstracta (es decir, dentro de la especificación ASN.1 aparece un campo correspondiente) o puede ser fijado por algunas otras acciones de codificador invocadas por "**field-to-be-set**" o "**flag-to-be-set**". La especificación de la codificación determina cómo ha de obtener un decodificador el tamaño del espacio de codificación a partir del valor de este campo. Un codificador conforme no producirá codificaciones en las que las transformadas del decodificador de este campo no identifiquen correctamente el espacio de codificación.

21.3.6 El valor "**container**" requiere la especificación de una **REFERENCE** en otro campo cuya clase de codificación (el contenedor) tiene un determinante de longitud y cuyo contenido incluye este espacio de codificación, o una especificación de que el final de la PDU determina el final del espacio de codificación (utilizando "**OUTER**"). El espacio de codificación termina cuando termina el contenedor especificado o cuando se encuentra el final de la PDU. Esta especificación sólo se puede utilizar si el espacio de codificación del elemento que se codifica es la última codificación que se ha de poner en el contenedor.

NOTA – Si se ponen más codificaciones en el contenedor, se trata de un error del codificador de la ECN (resultante posiblemente de un error de la especificación ECN o de la aplicación).

21.4 Tipo UnusedBitsDetermination

21.4.1 El tipo "**UnusedBitsDetermination**" es:

```
UnusedBitsDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, not-needed}
```

21.4.2 El valor por defecto de este tipo es siempre "**field-to-be-set**".

21.4.3 Una propiedad de codificación de este tipo especifica la manera en que un decodificador puede determinar los bits no utilizados cuando una codificación de valor está justificada a la izquierda o a la derecha en un espacio de codificación.

21.4.4 El valor "**field-to-be-set**" requiere la especificación de una **REFERENCE** en un campo cuyo valor será fijado por el codificador para llevar información de bits no utilizados, y será utilizado por un decodificador. La especificación de la codificación establece cómo ha de determinar un codificador el número de bits no utilizados, y cómo se fija el valor de este campo a partir del número de bits no utilizados. Si un campo se fija más de una vez utilizando "**field-to-be-set**" o "**flag-to-be-set**" (véase 21.7) y los diferentes procedimientos de codificación producen valores diferentes, se trata de un error de la especificación ECN o de la aplicación, y en este caso los codificadores no generarán codificaciones.

21.4.5 El valor "**field-to-be-used**" requiere la especificación de una **REFERENCE** en un campo cuyo valor puede ser fijado desde la sintaxis abstracta (es decir, dentro de la especificación ASN.1 aparece un campo correspondiente) o puede ser fijado por algunas otras acciones de codificador invocadas por "**field-to-be-set**" o "**flag-to-be-set**". La especificación de la codificación establece cómo ha de determinar un decodificador el número de bits no utilizados a partir del valor de este campo. Un codificador conforme no producirá codificaciones en las que las transformadas del decodificador de este campo no identifiquen correctamente el número de bits no utilizados.

21.4.6 El valor "**not-needed**" indica que un decodificador no requiere un determinante explícito para descubrir el número de bits no utilizados. El número de bits no utilizados se podrá deducir de la especificación de la codificación sin conocer el valor abstracto efectivo que ha sido codificado. Esta determinación se describe para cada codificación de valor.

21.5 Tipo `OptionalityDetermination`

21.5.1 El tipo "**OptionalityDetermination**" es:

```
OptionalityDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, container, handle, pointer}
```

21.5.2 El valor por defecto de este tipo es siempre "**field-to-be-set**".

21.5.3 Una propiedad de codificación de este tipo especifica la manera en que se determina la presencia o la ausencia de un componente opcional.

21.5.4 El valor "**field-to-be-set**" requiere la especificación de una **REFERENCE** en un campo cuyo valor será fijado por el codificador para llevar información de opcionalidad, y será utilizado por un decodificador. La especificación ECN incluirá también una propiedad de codificación que especifique cómo tiene que fijar un codificador el valor de este campo a partir de un valor booleano conceptual, que es verdadero si está presente el componente opcional y falso si está ausente el componente opcional. Si un campo se fija más de una vez utilizando "**field-to-be-set**" o "**flag-to-be-set**" (véase 21.7) y los diferentes procedimientos de codificación producen valores diferentes, se trata de un error de la especificación ECN o de la aplicación, y en este caso los codificadores no generarán codificaciones.

21.5.5 El valor "**field-to-be-used**" requiere la especificación de una **REFERENCE** en un campo cuyo valor puede ser fijado desde la sintaxis abstracta (es decir, dentro de la especificación ASN.1 aparece un campo correspondiente) o puede ser fijado por algunas otras acciones de codificador invocadas por "**field-to-be-set**" o "**flag-to-be-set**". La especificación incluirá también una propiedad de codificación que especifique cómo ha de determinar un decodificador la presencia o ausencia del componente opcional a partir del valor de este campo. Un codificador conforme asegurará que el valor de este campo determina correctamente la presencia o ausencia del campo opcional.

21.5.6 El valor "**container**" requiere la especificación de una **REFERENCE** en otro campo cuya clase de codificación (el contenedor) tiene un determinante de longitud y cuyo contenido incluye este componente opcional, o de una especificación de que el contenedor es el final de la PDU (utilizando **OUTER**). Si el final del contenedor está presente cuando un decodificador está buscando el comienzo de este componente opcional, el decodificador determinará que este componente opcional está ausente.

NOTA – Esta especificación sólo puede ser utilizada si los valores abstractos que se codifican son tales que no se van a poner más codificaciones en el contenedor. Esto quizá requiera la imposición de restricciones a los valores abstractos del tipo ASN.1, por ejemplo, para prohibir la inclusión de un componente opcional posterior a menos que estén presentes todos los componentes opcionales anteriores. Si se tienen que poner más codificaciones en el contenedor que sigue a un componente cuya opcionalidad se determina de esta manera, se trata de un error de la especificación ECN o de la aplicación, pero un codificador conforme no generará esas codificaciones.

21.5.7 El valor "**handle**" requiere la especificación de un asa de identificación. El asa de identificación será exhibida por el objeto de codificación del componente opcional y por cualquier codificación alternativa que pueda seguir si este componente opcional está ausente, y el valor del asa será diferente para la codificación del componente opcional y todas las codificaciones alternativas que puedan seguir. Si se alcanza el final de cualquier contenedor abierto (o el final de la PDU) en el momento en que un decodificador está tratando de detectar la presencia o la ausencia del componente opcional, quiere decir que ese componente está ausente. De otro modo, un decodificador determinará que el componente está presente si, y solamente si, la decodificación de las partes restantes de la codificación produce un valor del asa de identificación especificada que concuerda con el del componente opcional. Si esto no da como resultado la identificación correcta de la presencia o la ausencia de una codificación del componente opcional, se trata de un error de la especificación ECN, pero los codificadores conformes no generarán esas codificaciones.

21.5.8 El valor "**pointer**" requiere la especificación de una **REFERENCE** de comienzo de codificación a otro campo. Si ese campo es cero, este componente está ausente. Si es distinto de cero, se aplican las reglas del puntero de comienzo de codificación (véase 22.3).

21.6 Tipo `AlternativeDetermination`

21.6.1 El tipo "**AlternativeDetermination**" es:

```
AlternativeDetermination ::=
      ENUMERATED {field-to-be-set, field-to-be-used, handle}
```

21.6.2 El valor por defecto de este tipo es siempre "**field-to-be-set**".

21.6.3 Una propiedad de codificación de este tipo especifica la manera en que un decodificador determina qué alternativa está presente en una codificación de una clase en la categoría alternativas.

21.6.4 El valor "**field-to-be-set**" requiere la especificación de una **REFERENCE** en un campo cuyo valor será fijado por el codificador para llevar información que identifique una alternativa, y será utilizado por un decodificador. La especificación incluirá también una propiedad de codificación que especifique cómo ha de fijar un codificador el valor de este campo a partir de un valor entero conceptual que identifique cada alternativa (utilizando un orden especificado en otras propiedades de codificación). Si un campo se fija más de una vez utilizando "**field-to-be-set**" o "**flag-to-be-set**" (véase 21.7) y los diferentes procedimientos de codificación producen valores diferentes, se trata de un error de la especificación ECN o de la aplicación, y en este caso los codificadores no generarán codificaciones.

21.6.5 El valor "**field-to-be-used**" requiere la especificación de una **REFERENCE** en un campo cuyo valor puede ser fijado desde la sintaxis abstracta (es decir, dentro de la especificación ASN.1 aparece un campo correspondiente) o puede ser fijado por algunas otras acciones de codificador invocadas por "**field-to-be-set**" o "**flag-to-be-set**". La especificación incluirá también una propiedad de codificación que especifique cómo ha de determinar un decodificador (a partir del valor del campo referenciado) un valor entero conceptual que identifique la alternativa (utilizando un orden especificado en otras propiedades de codificación).

21.6.6 El valor "**handle**" requiere la especificación de un asa de identificación. Este asa de identificación será exhibida por (las codificaciones de) todas las alternativas de la clase, y la codificación de cada alternativa tendrá un valor diferente para el asa de identificación. (El no cumplimiento de esta regla es un error de la especificación ECN; es preciso que los codificadores conformes no generen codificaciones cuando se produzca ese incumplimiento). Este valor especifica que un decodificador tendrá que determinar cuál es la alternativa presente decodificando las partes restantes de la codificación para producir un valor del asa de identificación especificada. La alternativa cuyo valor de asa de identificación concuerda con este valor es la alternativa que está presente. Si se alcanza el final de cualquier contenedor abierto (o el final de la PDU) antes de que el asa de identificación pueda ser decodificada, o si el valor del asa de identificación no concuerda con el de ninguna alternativa, se trata de un error de codificación.

21.7 Tipo `RepetitionSpaceDetermination`

21.7.1 El tipo "RepetitionSpaceDetermination" es:

```
RepetitionSpaceDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, flag-to-be-set, flag-to-be-used,
       container, pattern, handle, not-needed}
```

21.7.2 El valor por defecto de este tipo es siempre "**field-to-be-set**".

21.7.3 Una propiedad de codificación de este tipo especifica la manera en que un decodificador determina el final del espacio de codificación en la codificación de una clase en la categoría repetición. Sustituye a la utilización de una propiedad de codificación del tipo "**EncodingSpaceDetermination**" en la codificación de repetición.

21.7.4 El valor "**field-to-be-set**" requiere la especificación de una **REFERENCE** en un campo cuyo valor será fijado por el codificador para llevar información que identifique el tamaño del espacio de repetición. La especificación de la codificación determina cómo ha de fijar un codificador el valor de este campo a partir del tamaño (en unidades de espacio de repetición) del espacio de repetición. Si un campo se fija más de una vez utilizando "**field-to-be-set**" o "**flag-to-be-set**" y los diferentes procedimientos de codificación producen valores diferentes, se trata de un error de la especificación ECN o de la aplicación, y en este caso los codificadores no generarán codificaciones.

21.7.5 El valor "**field-to-be-used**" requiere la especificación de una **REFERENCE** en un campo cuyo valor puede ser fijado desde la sintaxis abstracta (es decir, dentro de la especificación ASN.1 aparece un campo correspondiente) o puede ser fijado por algunas otras acciones de codificador invocadas por "**field-to-be-set**" o "**flag-to-be-set**". La especificación de la codificación determina cómo ha de obtener un decodificador el tamaño (en unidades de espacio de repetición) del espacio de codificación a partir del valor de este campo. Un codificador conforme no producirá codificaciones en las que las transformadas del decodificador de este campo no identifiquen correctamente el final del espacio de codificación.

21.7.6 El valor "**flag-to-be-set**" requiere la especificación de una **REFERENCE** en un campo que es parte del elemento repetido, y cuyo valor será fijado por el codificador para identificar el último elemento de la repetición. La especificación de la codificación determina cómo ha de fijar un codificador el valor de este campo a partir de un booleano que es falso si el elemento es el último de la repetición, y es verdadero en otro caso. Si un campo se fija más de una vez utilizando "**flag-to-be-set**" o "**field-to-be-set**" y los diferentes procedimientos de codificación producen valores diferentes, se trata de un error de la especificación ECN o de la aplicación, y en este caso los codificadores no generarán codificaciones.

21.7.7 El valor "**flag-to-be-used**" requiere la especificación de una **REFERENCE** en un campo que es parte del elemento repetido y cuyo valor puede ser fijado desde la sintaxis abstracta (es decir, dentro de la especificación ASN.1 aparece un campo correspondiente) o puede ser fijado por algunas otras acciones de codificador invocadas por "**flag-to-be-set**" o "**field-to-be-set**". La especificación de la codificación determina cómo ha de obtener un decodificador un valor booleano a partir del valor de este campo. El valor booleano será falso si el elemento es el último elemento de la repetición, y será verdadero en otro caso. Un codificador conforme no producirá codificaciones en las que las transformadas del decodificador de este campo no identifiquen correctamente el último elemento de la repetición.

21.7.8 El valor "**container**" requiere la especificación de una **REFERENCE** en otro campo cuya clase de codificación (el contenedor) tiene un determinante de longitud y cuyo contenido incluye la clase de codificación en la categoría repetición, o de una especificación (utilizando **OUTER**) de que el final de la PDU determina el final de las repeticiones. Las repeticiones terminan cuando el contenedor especificado termina o cuando, tras la codificación completa de una repetición, se encuentra el final de la PDU.

NOTA – Esta especificación sólo puede ser utilizada si la codificación de la clase (categoría repetición) es la última codificación que hay que poner en el contenedor. Si se han de poner más codificaciones en el contenedor, se trata de un error de la especificación ECN, pero los codificadores conformes no generarán esas codificaciones.

21.7.9 El valor "**pattern**" especifica que algún esquema de bits especificado (véase 21.10) terminará las repeticiones. En este caso, propiedades de codificación adicionales requerirán la inserción por un codificador de un esquema especificado, y la detección de este esquema por un decodificador. Si la codificación del esquema puede ser la parte inicial de la codificación de un valor abstracto de una definición, se trata de un error de la especificación ECN. Un codificador conforme detectará esos errores y no generará codificaciones que incumplan esta regla.

NOTA – Un ejemplo al respecto es una cadena de caracteres terminada en nulo en cuyo contenido no se permite que figure un carácter nulo.

21.7.10 El valor "**handle**" requiere que se especifique un asa de identificación. El asa de identificación será exhibida por el elemento que está siendo repetido, y por todos los elementos siguientes posibles (teniendo en cuenta la opcionalidad). El valor del asa de identificación del elemento que se repite será diferente del de todos los elementos siguientes posibles.

21.7.11 El valor "**not-needed**" indica que el número de repeticiones en la sintaxis abstracta es fijo.

NOTA – Si se especifica esta codificación y el número de repeticiones no se limita de esa manera, o si la aplicación no respeta esta limitación, se trata de un error de la especificación ECN (que será detectado y bloqueado por los codificadores).

21.8 Tipo Justification

21.8.1 El tipo "**Justification**" es:

```
Justification ::= CHOICE
                { left              INTEGER (0..MAX),
                  right             INTEGER (0..MAX)}
```

21.8.2 El valor por defecto de este tipo es siempre "**right:0**".

21.8.3 Una propiedad de codificación de este tipo especifica justificación a la derecha o a la izquierda de la codificación de un valor dentro del espacio de codificación, con un desplazamiento en bits con respecto a los extremos del espacio de codificación.

21.8.4 La alternativa "**left**" especifica que el bit inicial de la codificación del valor está posicionado con respecto al borde delantero del espacio de codificación. El valor entero especifica el número de bits entre el borde delantero del espacio de codificación y el bit inicial de la codificación del valor.

NOTA – Si la codificación del valor no tiene una longitud fija o no es autodelimitante, la utilización de relleno del valor en un contenedor de tamaño fijo puede imposibilitar, en algunas circunstancias, el que un decodificador recupere los valores abstractos originales. Esto sería un error de la especificación ECN.

21.8.5 La alternativa "**right**" especifica que el bit final de la codificación del valor está posicionado con respecto al borde trasero del espacio de codificación. El valor entero especifica el número de bits entre el bit final de la codificación del valor y el borde trasero del espacio de codificación.

21.8.6 La fijación de los bits (si los hay) antes o después de la codificación del valor está determinada por propiedades de codificación del tipo "**Padding**" y "**Pattern**" (véanse 21.9 y 21.10).

21.9 Tipo Padding

21.9.1 El tipo "**Padding**" es:

Padding ::= ENUMERATED {zero, one, pattern, encoder-option}

21.9.2 El valor por defecto de una propiedad de codificación de este tipo es siempre "**zero**".

21.9.3 Una propiedad de codificación de este tipo especifica los detalles del relleno previo, para clases en la categoría relleno, y del relleno posterior de una PDU especificado en la clase de codificación **#OUTER**.

21.9.4 Si el valor es "**zero**", el relleno se hace con cero bits.

21.9.5 Si el valor es "**one**", el relleno se hace con un bit.

21.9.6 Si el valor es "**pattern**", los bits se fijan de acuerdo con la propiedad de codificación del tipo "**Pattern**" (véase 21.10).

21.9.7 Si el valor es "**encoder-option**", el codificador elige libremente los valores de los bits.

21.10 Tipos Pattern y Non-Null-Pattern

21.10.1 El tipo "**Pattern**" es:

```
Pattern ::= CHOICE
      {bits                       BIT STRING,
       octets                     OCTET STRING,
       char8                      IA5String,
       char16                     BMPString,
       char32                     UniversalString,
       any-of-length              INTEGER (1..MAX),
       different                  ENUMERATED {any} }
```

21.10.2 El tipo "**Non-Null-Pattern**" es:

```
Non-Null-Pattern ::= Pattern
      (ALL EXCEPT (bits:''B | octets:''H | char8:"" | char16:"" |
                   char32:""))
```

21.10.3 El valor por defecto de una propiedad de codificación de este tipo es siempre "**bits:'0'B**".

21.10.4 La alternativa "**bits**" u "**octects**" especifica un esquema de longitud y valor iguales a los de la cadena de bits o cadena de octetos dada, respectivamente.

21.10.5 La alternativa "**char8**" especifica un esquema (múltiplo de 8 bits) en el que cada carácter de la cadena dada se convierte en su valor ISO/CEI 10646-1 como un valor de 8 bits.

21.10.6 La alternativa "**char16**" especifica un esquema (múltiplo de 16 bits) en el que cada carácter de la cadena dada se convierte en su valor ISO/CEI 10646-1 como un valor de 16 bits.

21.10.7 La alternativa "**char32**" especifica un esquema (múltiplo de 32 bits) en el que cada carácter de la cadena dada se convierte en su valor ISO/CEI 10646-1 como un valor de 32 bits.

21.10.8 La alternativa "**any-of-length**" especifica un tamaño para el esquema. El valor efectivo del esquema es una opción del codificador.

21.10.9 El valor "**different:any**" sólo se permite cuando hay otra propiedad de codificación del tipo "**Pattern**" en el mismo grupo de propiedades de codificación. En este caso, cualquiera de las propiedades (pero no ambas) de codificación del tipo "**Pattern**" se puede fijar en "different:any". El valor "**different:any**" especifica que la longitud del esquema será la misma que la del esquema especificado para la otra propiedad de codificación. También especifica que su valor es una opción del codificador, siempre que el valor sea diferente del valor del esquema especificado para la otra propiedad de codificación.

21.10.10 Cuando se emplea para relleno previo y justificación (pero no para otros usos), se utiliza el "**Non-Null-Pattern**", y el esquema es truncado y/o reproducido según se necesite a fin de proporcionar bits suficientes para el relleno previo, el relleno previo al valor o el relleno posterior al valor.

21.10.11 El valor "**different:any**" del tipo "**Pattern**" está excluido en la mayoría de las utilizaciones de este tipo. Cuando se emplea un parámetro del tipo "Pattern" para especificar el esquema de un valor booleano (digamos, "**TRUE**"), se puede utilizar el valor "**different:any**" para especificar el esquema de otro valor booleano ("**FALSE**" en este caso). Cuando se emplea de esta manera, "different:any" representa la opción de un codificador para el esquema. El codificador puede utilizar cualquier esquema que elija, pero deberá ser de la misma longitud que el otro esquema y deberá diferir de él en al menos la posición de un bit.

21.11 Tipo RangeCondition

21.11.1 El tipo "**RangeCondition**" es

```
RangeCondition ::= ENUMERATED
      { unbounded-or-no-lower-bound,
        semi-bounded-with-negatives,
        bounded-with-negatives,
        semi-bounded-without-negatives,
        bounded-without-negatives}
```

21.11.2 El valor por defecto de una propiedad de este tipo es siempre "**unbounded-or-no-lower-bound**".

21.11.3 Se utiliza una propiedad de codificación del tipo "**RangeCondition**" en la especificación de un predicado que comprueba la existencia y la naturaleza de límites en los valores enteros asociados a una clase de codificación en la categoría entero.

21.11.4 El predicado se satisface para cada valor enumeración si, y solamente si, los límites impuestos a la clase de codificación en la categoría entero cumplen las condiciones siguientes:

a) **unbounded-or-no-lower-bound**: o bien no hay límites, o bien hay un solo límite superior pero no hay límite inferior.

b) **semi-bounded-with-negatives**: hay un límite inferior que es menor que cero, pero no hay límite superior.

c) **bounded-with-negatives**: hay un límite inferior que es menor que cero, y un límite superior.

d) **semi-bounded-without-negatives**: hay un límite inferior que es mayor o igual que cero, pero no hay límite superior.

e) **bounded-without-negatives**: hay un límite inferior que es mayor o igual que cero, y un límite superior.

NOTA – Para cualquier conjunto de límites dado, se satisfará exactamente un predicado.

21.12 Tipo SizeRangeCondition

21.12.1 El tipo "**SizeRangeCondition**" es:

```
SizeRangeCondition ::= ENUMERATED
      { no-ub-with-zero-lb,
        ub-with-zero-lb,
        no-ub-with-non-zero-lb,
        ub-with-non-zero-lb,
        fixed-size}
```

21.12.2 El valor por defecto de una propiedad de codificación de este tipo es siempre "**no-ub-with-zero-lb**".

21.12.3 Se utiliza una propiedad de codificación del tipo "**SizeRangeCondition**" para probar propiedades de los límites de una constricción de tamaño efectivo asociada a una clase en la categoría repetición o cadena de caracteres.

21.12.4 El predicado se satisface para cada valor enumeración si, y solamente si, la constricción de tamaño efectivo cumple las condiciones siguientes:

a) **no-ub-with-zero-lb**: no hay un límite superior impuesto al tamaño y el límite inferior es cero.

b) **ub-with-zero-lb**: hay un límite superior impuesto al tamaño y el límite inferior es cero.

c) **no-ub-with-non-zero-lb**: no hay un límite superior impuesto al tamaño y el límite inferior es distinto de cero.

d) **ub-with-non-zero-lb**: hay un límite superior impuesto al tamaño y el límite inferior es distinto de cero.

e) **fixed-size**: el límite superior y el límite inferior impuestos al tamaño tienen el mismo valor.

NOTA – Sólo el caso "**fixed-size**" se solapa con otros predicados.

21.13 Tipo ReversalSpecification

21.13.1 El tipo "**ReversalSpecification**" es:

```
ReversalSpecification ::= ENUMERATED
      {no-reversal,
      reverse-bits-in-units,
      reverse-half-units,
      reverse-bits-in-half-units}
```

21.13.2 El valor por defecto de una propiedad de codificación de este tipo es siempre "**no-reversal**".

21.13.3 Se utiliza una propiedad de codificación del tipo "**ReversalSpecification**" en la transformada final de bits de un espacio de codificación en una memoria intermedia de salida para transmisión (aplicándose la transformada inversa para la decodificación).

NOTA – Los bits insertados como resultado del relleno previo especificado por un objeto de codificación no forman parte de la codificación a la que se aplica la inversión de bits especificada por ese objeto de codificación, pero pueden ser sometidos a la inversión de bits especificada por un objeto de codificación de un contenedor en el que está incorporada la codificación completa.

21.13.4 Valores de este tipo se utilizan siempre junto con una propiedad de codificación del tipo "**Unit**" que especifica el tamaño de una unidad en bits (véase 21.1).

21.13.5 Si los valores "**reverse-half-units**" y " **reverse-bits-in-half-units**" son utilizados cuando la propiedad de codificación del tipo "**Unit**" no es un número par de bits, se trata de un error de la especificación ECN.

21.13.6 Las enumeraciones especifican (en el orden de enumeración indicado más arriba) lo siguiente:

a) sin inversión de bits, o

b) inversión del orden de las semiunidades (sin cambiar el orden de los bits en cada semiunidad), o

c) inversión del orden de los bits en cada semiunidad sin invertir el orden de las semiunidades, o

d) inversión del orden de los bits en cada unidad.

21.13.7 Si el número de bits de una codificación a la que se aplica la inversión de bits no es un múltiplo entero de "**Unit**", se trata de un error de la especificación ECN.

21.13.8 La inversión de bits puede ser especificada para la codificación de todas las clases que puedan aparecer como campos de estructuras de codificación, excepto una clase de codificación de la categoría alternativas que no utiliza el concepto de espacio de codificación.

21.14 Tipo ResultSize

21.14.1 El tipo "**ResultSize**" es:

```
ResultSize ::= INTEGER {variable(-1), fixed-to-max(0)} (-1..MAX)
```

21.14.2 El valor por defecto de una propiedad de codificación de este tipo es siempre "**variable**".

21.14.3 Una propiedad de codificación de este tipo especifica el tamaño del resultado en una clase **#TRANSFORM**.

21.14.4 El valor "**variable**" especifica que el tamaño del resultado **#TRANSFORM** variará para diferentes valores abstractos, y lo determina la especificación detallada de la transformada.

21.14.5 El valor "**fixed-to-max**" especifica que el tamaño del resultado **#TRANSFORM** ha de ser el mismo para la transformada de todos los valores abstractos. Especifica además que el tamaño del destino ha de ser el tamaño más pequeño que puede contener la codificación especificada de uno cualquiera de los valores abstractos (de todos). Los detalles precisos de esta especificación se definen para cada transformada en la que se utilizan valores de este tipo.

21.14.6 Un valor positivo del tipo "**ResultSize**" especifica que el tamaño del resultado **#TRANSFORM** es fijo. Este valor se utiliza en la especificación de la transformada real.

21.15 Tipo HandleValue

21.15.1 El tipo "**HandleValue**" es:

```
HandleValue ::= CHOICE {
      bits        BIT  STRING,
      octets      OCTET STRING,
      number      INTEGER (0..MAX),
      tag         ENUMERATED {any}}
```

21.15.2 El tipo "**HandleValue**" se utiliza para especificar el valor de un asa de identificación exhibida por objetos de codificación particulares.

21.15.3 Los valores de cualquier asa de identificación exhibida por un objeto de codificación han de ser los mismos para todos lo valores abstractos que codifica ese objeto de codificación (véase 22.9.2.2). El valor de un asa de identificación puede ser utilizado para indicar la presencia o la ausencia de componentes opcionales, la elección de alternativas o el final de una repetición. En tales circunstancias se requiere que los valores del asa exhibida por la codificación de diferentes alternativas o componentes sean distintos (véanse 21.5.7, 21.6.6 y 21.7.10).

NOTA – Los valores de las asas de identificación exhibidas por un objeto de codificación dado pueden ser determinados, en teoría, mediante la codificación de un valor de prueba. Sin embargo, para facilitar el trabajo de implementación se pide al especificador de la ECN que especifique el valor del asa en todos los casos excepto cuando (para codificaciones de la clase rótulo) el valor del asa de identificación depende del número de rótulo asociado a esa clase de rótulo, ya sea directamente por la generación implícita a partir de un rótulo ASN.1, o por el establecimiento de la correspondencia desde una estructura generada implícitamente.

21.15.4 Las alternativas "**bits**", "**octets**" y "**number**" especifican el valor del asa como una cadena de bits, cadena de octetos y valor entero, respectivamente. Si este valor no puede ser codificado dentro del número de bits especificado para el asa de identificación, se trata de un error de la especificación ECN (véase 22.9).

21.15.5 La alternativa "**tag:any**" especifica que el valor del asa es determinado por el número especificado en una estructura de codificación ECN para una clase en la categoría rótulo, o por el número de rótulo cuya correspondencia se ha establecido desde una construcción de rótulo ASN.1. Sólo se utilizará cuando se especifique la identificación del asa para codificar una clase en la categoría rótulo.

22 Grupos de propiedades de codificación utilizados habitualmente

En esta cláusula se especifican grupos de propiedades de codificación utilizados habitualmente en la sintaxis definida (véase la cláusula 20).También se especifican el objetivo de cada grupo, las restricciones impuestas a los valores de las propiedades de codificación y la sintaxis que se puede utilizar, así como las acciones de codificador y decodificador para cada grupo.

22.1 Especificación de sustitución

Hay tres variantes de la especificación de sustitución:

a) Especificación de sustitución completa: Se utiliza para clases en la categoría concatenación, en donde se puede sustituir toda la estructura, o bien se pueden sustituir de manera selectiva componentes opcionales y no opcionales.

b) Especificación de sustitución de estructura o componente: Se utiliza para clases en la categoría alternativas y para la clase de codificación **#CONDITIONAL-REPETITION**, en donde la sustitución puede ser de toda la estructura o del componente.

   NOTA – Cuando se utiliza un objeto de codificación de la clase **#CONDITIONAL-REPETITION** para definir las codificaciones de una clase en la categoría cadena de bits, cadena de caracteres o cadena de octetos, sólo se puede efectuar sustitución de estructura únicamente.

c) Especificación de sustitución de estructura únicamente: Se utiliza con clases que no tienen componentes.

22.1.1 Propiedades de codificación, sintaxis y objetivo

22.1.1.1 La especificación de sustitución completa utiliza las propiedades de codificación siguientes:

```
&#Replacement-structure
      OPTIONAL,
&#Replacement-structure2
      OPTIONAL,
&replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,
&replacement-structure-encoding-object2 &#Replacement-structure2   OPTIONAL,
&#Head-end-structure                                                 OPTIONAL,
&#Head-end-structure2                                                OPTIONAL
```

22.1.1.2 La sintaxis que se ha de utilizar para la especificación de sustitución completa será:

```
[REPLACE
      [STRUCTURE]
      [COMPONENT]
      [ALL COMPONENTS]
      [OPTIONALS]
      [NON-OPTIONALS]
      WITH &#Replacement-structure
            [ENCODED BY &replacement-structure-encoding-object
                  [INSERT AT HEAD &#Head-end-structure]]
            [AND OPTIONALS WITH &#Replacement-structure2
                  [ENCODED BY &replacement-structure-encoding-object2
                        [INSERT AT HEAD &#Head-end-structure2]]] ]
```

22.1.1.3 La especificación de sustitución de estructura o componente utiliza las propiedades de codificación siguientes:

```
&#Replacement-structure
      OPTIONAL,
&replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,
&#Head-end-structure                                                 OPTIONAL
```

22.1.1.4 La sintaxis que se ha de utilizar para la especificación de sustitución de estructura o componente será:

```
[REPLACE
      [STRUCTURE]
      [COMPONENT]
      [ALL COMPONENTS]
      WITH &Replacement-structure
      [ENCODED BY &replacement-structure-encoding-object
            [INSERT AT HEAD &#Head-end-structure]]]
```

22.1.1.5 La especificación de sustitución de estructura únicamente utiliza las propiedades de codificación siguientes:

```
&#Replacement-structure
      OPTIONAL,
&replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL
```

22.1.1.6 La sintaxis que se ha de utilizar para la especificación de sustitución de estructura únicamente será:

```
[REPLACE
      [STRUCTURE]
      WITH &#Replacement-structure
            [ENCODED BY &replacement-structure-encoding-object]]
```

22.1.1.7 La utilización de "**WITH SYNTAX**" para estos grupos de propiedades de codificación especifica que:

a) la clase de codificación a la que se aplica este objeto de codificación ha de ser sustituida por completo (**REPLACE STRUCTURE**); en el caso de una clase de codificación en la categoría opcionalidad, se sustituye todo el componente; en el caso de un objeto de codificación **#CONDITIONAL-REPETITION** utilizado en la definición de un objeto de codificación de una clase en la categoría cadena de bits, cadena de caracteres, cadena de octetos o repetición, se sustituye en su totalidad la estructura de la cadena de bits, cadena de caracteres, cadena de octetos o repetición (si se cumple la condición de gama); o

b) todos sus componentes (excepto en el caso de especificación de estructura únicamente) han de ser sustituidos (con la misma acción de sustitución para todos los componentes) ("**REPLACE COMPONENT**" o "**REPLACE ALL COMPONENTS**"); o

c) todos sus componentes opcionales (sólo en el caso de especificación de sustitución completa) han de ser sustituidos ("**REPLACE OPTIONALS**"); o

d) todos sus componentes no opcionales (sólo en el caso de especificación de sustitución completa) han de ser sustituidos ("**REPLACE NON-OPTIONALS**"); o

e) todos sus componentes (sólo en el caso de especificación de sustitución completa) han de ser sustituidos, con acciones de sustitución diferentes para opcionales y no opcionales ("**REPLACE NON-OPTIONALS AND OPTIONALS**").

22.1.1.8 "**REPLACE COMPONENT**" es sinónimo de "**REPLACE ALL COMPONENTS**". Sería normal, aunque no se requiere, utilizar esto si sólo hay un componente único.

22.1.1.9 Los "**ENCODED BY**" opcionales especifican un objeto de codificación de la estructura de sustitución.

22.1.1.10 Los "**INSERT AT HEAD**" opcionales especifican una estructura de codificación (la inserción en el extremo de cabeza) que se ha de insertar antes que todos los componentes de la clase (constructor) que efectúa la sustitución. Hay una inserción en el extremo de cabeza por cada componente que es sustituido, y las inserciones se efectúan en el orden de los componentes originales.

22.1.2 Restricciones de la especificación

22.1.2.1 Se utilizará exactamente una de las sintaxis permitidas entre "**REPLACE**" y "**WITH**".

22.1.2.2 Las estructuras de sustitución "**WITH**" serán estructuras de codificación parametrizadas con un solo parámetro de clase de codificación. Cuando se especifiquen en la sintaxis definida anteriormente, sólo se dará el nombre de referencia de clase de la estructura. No tendrán ninguna lista de parámetros en esta utilización de los nombres.

22.1.2.3 Estas estructuras parametrizadas son instanciadas durante la acción de sustitución con un parámetro real como se especifica en 22.1.3. La utilización del parámetro ficticio en las estructuras parametrizadas de sustitución será coherente con la clase del parámetro real que será suministrado en la acción de sustitución.

NOTA – En particular, si se utiliza "**REPLACE STRUCTURE**" para una clase de codificación en la categoría rótulo, el parámetro ficticio sólo puede estar presente en la estructura de sustitución cuando se permite una clase de codificación en la categoría rótulo.

22.1.2.4 Los objetos de codificación "**ENCODED BY**" serán objetos de codificación parametrizados para las estructuras de codificación "**WITH**". Tendrán un parámetro ficticio (digamos, **#D**) que es una clase de codificación, y se definirán en una asignación de objeto de codificación parametrizado en la que el gobernador es la estructura de codificación parametrizada "**WITH**", instanciada con **#D**. Cuando se especifiquen en la sintaxis definida anteriormente, sólo se dará el nombre de referencia de objeto de codificación. No tendrán ninguna lista de parámetros en esta utilización de los nombres.

22.1.2.5 Estos objetos de codificación parametrizados son instanciados durante la acción de sustitución por un parámetro real que es el mismo parámetro real utilizado para instanciar las estructuras de codificación de sustitución " **WITH** " correspondientes. También pueden tener:

– (opcionalmente) otro parámetro ficticio (pero sólo uno) que es un conjunto de objetos de codificación; cuando son instanciados durante la acción de sustitución, el parámetro real de este parámetro ficticio es el conjunto de objetos de codificación combinados actual;

– (condicionalmente) otro parámetro ficticio (pero sólo uno) que es un parámetro **REFERENCE**. Este parámetro estará presente si, y solamente si, se especifica "**INSERT AT HEAD**". Cuando los objetos de codificación son instanciados durante la acción de sustitución, el parámetro real de este parámetro ficticio es una referencia a la estructura "**INSERT AT HEAD**"correspondiente.

22.1.2.6 Todos los campos de la estructura de sustitución que no son parte del parámetro de la clase de codificación son parámetros auxiliares, y serán fijados por la codificación de la estructura de sustitución.

22.1.2.7 Las estructuras de codificación "**INSERT AT HEAD**" no tendrán parámetros ficticios. Todos sus campos son campos auxiliares y serán fijados por el objeto de codificación "**ENCODED BY**" mediante su parámetro **REFERENCE**.

22.1.2.8 Si un objeto de codificación tiene una cláusula "**REPLACE STRUCTURE**", no tendrá cláusula "**INSERT AT HEAD**" y sí tendrá una cláusula "**ENCODED BY**".

22.1.3 Acciones de codificador

22.1.3.1 Si un objeto de codificación de una clase en el grupo de categorías campo de bits o en la categoría rótulo especifica "**REPLACE STRUCTURE**", un codificador sustituirá la estructura por una instanciación de la estructura de sustitución, utilizando el nombre de la estructura original como el parámetro real.

22.1.3.2 Si un objeto de codificación de una clase en la categoría constructor de codificación especifica "**REPLACE STRUCTURE**", un codificador sustituirá la construcción en su totalidad por una instanciación de la estructura de sustitución utilizando toda la construcción original como el parámetro real.

22.1.3.3 Si un objeto de codificación de una clase en la categoría opcionalidad especifica "**REPLACE STRUCTURE**", un codificador sustituirá la construcción en su totalidad por una instanciación no opcional de la estructura de sustitución. El parámetro real será un nombre de estructura oculta (que no concuerda con ninguna otra estructura, que nunca puede tener objetos de codificación). Este nombre de estructura oculta se desreferenciará al componente opcional original total (incluyendo cualesquiera clases en la categoría rótulo) excepto para la clase en la categoría opcionalidad.

22.1.3.4 Si un objeto de codificación de cualquier clase especifica "**REPLACE COMPONENT**", "**REPLACE ALL COMPONENTS**", "**REPLACE OPTIONAL COMPONENTS**" o "**REPLACE NON-OPTIONAL COMPONENTS**", un codificador sustituirá el componente o los componentes especificados en su totalidad por una instanciación no opcional de la estructura de sustitución. El parámetro real será un nombre de estructura oculta (que no concuerda con ninguna otra estructura, y que nunca puede tener objetos de codificación). Este nombre de estructura oculta se desreferenciará al componente opcional original total (incluyendo cualesquiera clases en la categoría rótulo) excepto para cualquier clase en la categoría opcionalidad.

22.1.3.5 Todos los valores abstractos y números de rótulo de la estructura o del componente original se harán corresponder con los valores abstractos y números de rótulo correspondientes del parámetro real de la estructura de sustitución. Los valores de otros campos de la estructura de sustitución se fijarán de acuerdo con la especificación del objeto de codificación de la estructura de sustitución.

22.1.3.6 Si se especifica una inserción en el extremo de cabeza, el codificador insertará la estructura del extremo de cabeza antes que todos los componentes de la estructura cuyo objeto de codificación está efectuando la sustitución. Las inserciones en el extremo de cabeza se llevarán a cabo en el mismo orden textual que los componentes que se sustituyen. Los valores de los campos de esta estructura se fijarán de acuerdo con la especificación del objeto de codificación de la estructura de sustitución.

NOTA – Estas estructuras serán normalmente un campo de entero simple que proporciona un determinante de ubicación para el campo que se sustituye.

22.1.3.7 El codificador instanciará el objeto o los objetos de codificación de la estructura de sustitución con parámetros reales como sigue:

a) Al parámetro ficticio que es una clase de codificación se le dará un parámetro real que sea el mismo que el parámetro real de la instanciación de la estructura de sustitución.

b) Al parámetro ficticio (si hay alguno) que es un parámetro **REFERENCE** se le dará un parámetro real que sea una referencia a la estructura de extremo de cabeza insertada.

c) Al parámetro ficticio (si hay alguno) que es un conjunto de objetos de codificación (cuyo gobernador es **#ENCODINGS**) se le dará un parámetro real que sea el conjunto de objetos de codificación combinados actual.

22.1.3.8 El codificador utilizará entonces este objeto de codificación instanciado para codificar la estructura de sustitución correspondiente en vez del conjunto de objetos de codificación combinados.

NOTA – La codificación de las inserciones en el extremo de cabeza es determinada por la aplicación del conjunto de objetos de codificación combinados actual.

22.1.4 Acciones de decodificador

Un decodificador generará (para una aplicación) los valores abstractos de la estructura original que estaba siendo codificada, ocultando cualquier actividad de sustitución (incluso si se lleva a cabo mediante la aplicación repetida de sustituciones).

22.2 Especificación de prealineación y relleno

22.2.1 Propiedades de codificación, sintaxis y objetivo

22.2.1.1 La especificación de prealineación y relleno utiliza las propiedades de codificación siguientes:

```
&encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
&encoding-space-pre-padding        Padding DEFAULT zero,
&encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                   DEFAULT bits:'0'B
```

22.2.1.2 La sintaxis que se ha de utilizar para la especificación de prealineación y relleno será:

```
[ALIGNED TO
```

```
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
```

22.2.1.3 Las definiciones de los tipos utilizados en la especificación de prealineación y relleno son como sigue:

```
Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
       dword32(32)} (0..256) --  (véase 21.1)

Padding ::= ENUMERATED {zero, one, pattern, encoder-option} -- (véase 21.9)

Pattern ::= CHOICE
      {bits                       BIT STRING,
       octets                     OCTET STRING,
       char8                      IA5String,
       char16                     BMPString,
       char32                     UniversalString,
       any-of-length              INTEGER (1..MAX),
       different                  ENUMERATED {any} }

Non-Null-Pattern ::= Pattern
      (ALL EXCEPT (bits:''B | octets:''H | char8:"" | char16:"" |
                   char32:"")) -- (véase 21.10)
```

22.2.1.4 Las propiedades de codificación de prealineación utilizan un valor del tipo "**Unit**" para especificar que un contenedor tiene que empezar en un múltiplo de "**Unit**" bits a partir del punto de alineación. El punto de alineación es el comienzo de la codificación del tipo al que un ELM aplicó una codificación, excepto cuando es reposicionado para la codificación de un tipo contenido por la utilización de un objeto de codificación **#OUTER** (véase la cláusula 25). Las propiedades de codificación del tipo "**Padding**" y "**Pattern**" se utilizan para controlar los bits que proporcionan el relleno de la alineación requerida. La especificación de "**ALIGNED TO NEXT**" produce el número mínimo de bits insertados. La especificación de "**ALIGNED TO ANY**" deja el número efectivo de bits insertados (a reserva de la restricción anterior de un múltiplo de "**Unit**") a criterio de los codificadores, y requiere la especificación de un puntero de comienzo.

22.2.2 Constricciones de la especificación

22.2.2.1 Se especificará, como máximo, un solo "**NEXT**" y un solo "**ANY**". Si no se especifica ninguno, se supone "**NEXT**".

22.2.2.2 Si se especifica "**ALIGNED TO ANY**", la especificación del objeto de codificación incluirá la cláusula "**START-POINTER**".

22.2.3 Acciones de codificador

22.2.3.1 Si se especifica "**NEXT**" (o se toma su valor por defecto), el codificador insertará el número mínimo de bits necesario para asegurar que el número total de bits de la codificación (desde el punto de alineación hasta el punto de comienzo del contenedor, véase 22.2.1.4) es un múltiplo de la propiedad de codificación del tipo "**Unit**".

22.2.3.2 Si se especifica "**ANY**", el codificador insertará un número de bits dependiente del codificador, siempre que el número total de bits de la codificación (desde el punto de alineación) sea un múltiplo de la propiedad de codificación del tipo "**Unit**".

22.2.3.3 Los bits insertados se fijarán de tal manera que el primer bit insertado sea el bit inicial de "**Pattern**", y así sucesivamente. Si se necesitan más bits que los que están presentes en la propiedad de codificación del tipo "**Pattern**", el esquema será reutilizado empezando con el bit más significativo.

22.2.4 Acciones de decodificador

22.2.4.1 Si se especifica "**NEXT**", el decodificador determinará el número de bits insertados a partir de las acciones del decodificador.

22.2.4.2 Si se especifica "**ANY**", el decodificador determinará el número de bits insertados a partir de la especificación del puntero de comienzo.

22.2.4.3 En todos los casos, el decodificador descartará de la aplicación los bits insertados transparentemente. No diagnosticará un error del codificador o de la especificación si los bits no están de acuerdo con las acciones de codificador especificadas.

22.3 Especificación de puntero de comienzo

22.3.1 Propiedades de codificación, sintaxis y objetivo

22.3.1.1 La especificación de puntero de comienzo utiliza las propiedades de codificación siguientes:

```
&start-pointer                    REFERENCE    OPTIONAL,
&start-pointer-unit               Unit (ALL EXCEPT repetitions) DEFAULT bit,
&Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL
```

22.3.1.2 La sintaxis que se ha de utilizar para la especificación de puntero de comienzo será:

```
[START-POINTER   &start-pointer
     [MULTIPLE OF     &start-pointer-unit]
     [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
```

22.3.1.3 La definición del tipo utilizado en la especificación de puntero de comienzo es como sigue:

```
Unit ::= INTEGER
     {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
      dword32(32)} (0..256) --  (véase 21.1)
```

22.3.1.4 Esta especificación identifica el comienzo del espacio de codificación de un elemento. Si el comienzo del espacio de codificación del elemento es "n" "**MULTIPLE OF**" unidades, el valor situado en el campo referenciado por la propiedad de codificación "**START-POINTER**" es el valor obtenido aplicando "**ENCODER-TRANSFORMS**" a "n".

NOTA 1 – Si "**MULTIPLE OF**" no es "bits", esto significa que el desplazamiento entre el comienzo del campo referenciado por la propiedad de codificación "**START-POINTER**" y el comienzo del espacio de codificación ha de ser un múltiplo entero de "**MULTIPLE OF**" unidades.

NOTA 2 – Por lo general habrá codificaciones de otros elementos, y quizá de otros punteros de comienzo entre el campo referenciado por la propiedad de codificación "**START-POINTER**" y el comienzo de la codificación de este elemento.

22.3.2 Constricciones de la especificación

22.3.2.1 Si "**ENCODER-TRANSFORMS**" no está presente, "**START-POINTER**" será una clase en la categoría entero.

22.3.2.2 Si "**ENCODER-TRANSFORMS**" está presente, "**START-POINTER**" será una clase con una categoría que pueda codificar un valor del resultado de la transformada final de las "**ENCODER-TRANSFORMS**".

22.3.2.3 Si cualquier transformada de las "**ENCODER-TRANSFORMS**" no es reversible para el valor abstracto al que se aplica, se trata de un error de la especificación ECN o de la aplicación. El origen de la primera transformada será un entero.

22.3.3 Acciones de codificador

22.3.3.1 El codificador determinará el número "n" de "**MULTIPLE OF**" unidades entre el comienzo de la codificación del campo "**START-POINTER**" (después de cualquier prealineación de ese campo) y el comienzo de la codificación de los elementos con la especificación de puntero de comienzo (después de cualquier prealineación de ese elemento). Si "n" no es un número entero, se trata de un error de la especificación ECN. Si el elemento que se codifica es opcional, y está ausente, "n" se fijará en cero.

22.3.3.2 El valor "n" será transformado utilizando las "**ENCODER-TRANSFORMS**" (si están presentes) para producir un valor conceptual "m". Si el valor "m" resultante no es un valor abstracto que pueda asociarse a la clase de codificación del "**START-POINTER**", se trata un error de la especificación ECN y la codificación no avanzará. De otro modo, el valor "m" será el valor codificado en el campo referenciado por "**START-POINTER**".

NOTA – El objeto de codificación aplicado al campo referenciado por "**START-POINTER**" determinará la codificación del valor "m".

22.3.4 Acciones de decodificador

22.3.4.1 El decodificador determinará el valor conceptual "m" del campo referenciado por "**START-POINTER**" y utilizará el conocimiento de las acciones del codificador a fin de invertir las transformadas (si las hay) para producir el valor entero "n".

22.3.4.2 Si "n" es cero, el decodificador diagnosticará un error de codificador cuando el elemento que se decodifica no sea un elemento opcional con una especificación de opcionalidad que determina opcionalidad por el puntero de comienzo. Si "n" es cero, y el elemento que se decodifica es un elemento opcional con una especificación de opcionalidad que determina opcionalidad por el puntero de comienzo, el decodificador determinará que el elemento está ausente.

22.3.4.3 El valor "n" se multiplica por "**MULTIPLE OF**" y se añade el comienzo de la codificación del campo "**START-POINTER**" para producir una posición "p". Si "p" es una posición en la codificación anterior al punto de decodificación actual, el decodificador diagnosticará un error de codificación.

22.3.4.4 Si "p" es una posición en la codificación igual o posterior al punto de decodificación actual, el decodificador ignorará en silencio todos los bits hasta la posición "p" y continuará decodificando este elemento desde la posición "p".

22.4 Especificación de espacio de codificación

22.4.1 Propiedades de codificación, sintaxis y objetivo

22.4.1.1 La especificación de espacio de codificación utiliza las propiedades de codificación siguientes:

```
&encoding-space-size              EncodingSpaceSize
                                  DEFAULT self-delimiting-values,
&encoding-space-unit              Unit (ALL EXCEPT repetitions)
                                  DEFAULT bit,
&encoding-space-determination     EncodingSpaceDetermination
                                  DEFAULT field-to-be-set,
&encoding-space-reference         REFERENCE OPTIONAL,
&Encoder-transforms               #TRANSFORM ORDERED OPTIONAL,
&Decoder-transforms               #TRANSFORM ORDERED OPTIONAL
```

22.4.1.2 La sintaxis que se ha de utilizar para la especificación de espacio de codificación será:

```
ENCODING-SPACE
      [SIZE &encoding-space-size
            [MULTIPLE OF &encoding-space-unit]]
      [DETERMINED BY &encoding-space-determination]
      [USING &encoding-space-reference
            [ENCODER-TRANSFORMS &Encoder-transforms]
            [DECODER-TRANSFORMS &Decoder-transforms]]
```

22.4.1.3 Las definiciones de los tipos utilizados en esta especificación son como sigue:

```
EncodingSpaceSize ::= INTEGER
      { encoder-option-with-determinant(-3),
        variable-with-determinant(-2),
        self-delimiting-values(-1),
        fixed-to-max(0)} (-3..MAX) -- (véase 21.2)

Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
       dword32(32)} (0..256) --  (véase 21.1)

EncodingSpaceDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, container}  -- (véase 21.3)
```

22.4.1.4 El objetivo de esta especificación es determinar acciones de codificador y decodificador para asegurar que un decodificador puede determinar correctamente el final de un espacio de codificación.

NOTA – Una codificación de valor real no necesariamente tiene que llenar todo el espacio de codificación, y la recuperación de la codificación del valor por un decodificador requerirá además, por lo general, acciones especificadas para el relleno del valor y la justificación (véase 22.8).

22.4.1.5 El significado de las propiedades de codificación de los tipos "**Unit**", "**EncodingSpaceSize**" y "**EncodingSpaceDetermination**" se dio en 21.1, 21.2 y 21.3. En conjunto especifican la manera en que se determina el final del espacio de codificación de este elemento.

NOTA – Se puede especificar "**variable-with-determinant**" incluso si el espacio de codificación es de tamaño fijo, caso de que el especificador de la ECN requiera que se incluya un determinante de longitud, aunque no se necesite.

22.4.1.6 La especificación "**USING**" es una referencia que permite a un decodificador determinar el final del espacio de codificación. Se trata de una referencia a un campo auxiliar o a un campo que lleva valores abstractos, o a un contenedor, dependiendo del valor de "**DETERMINED BY**".

22.4.2 Restricciones de la especificación

22.4.2.1 Si "**SIZE**" es "**variable with determinant**" y "**DETERMINED BY**" no está presente, se supone el valor por defecto ("**field-to-be-set**").

22.4.2.2 "**USING**" será especificado si, y solamente si, "**SIZE**" es "**variable with determinant**" o "**encoder-option-with-determinant**".

22.4.2.3 "**ENCODER-TRANSFORMS**" sólo estará presente si "**DETERMINED-BY**" se ha fijado en (o toma el valor por defecto de) "**field-to-be-set**". En este caso, la referencia "**USING**" será un campo auxiliar de la categoría cadena de bits, cadena de caracteres o entero.

22.4.2.4 Si cualquier transformada de las "**ENCODER-TRANSFORMS**" no es reversible para el valor abstracto al que se aplica, se trata de un error de la especificación ECN o de la aplicación. La primera transformada tendrá un origen que será un entero y la última transformada deberá tener un resultado que pueda ser codificado por la clase del campo referenciado por "**USING**".

22.4.2.5 "**DECODER-TRANSFORMS**" sólo estará presente si "**DETERMINED-BY**" se ha fijado en **field-to-be-used**. La primera transformada tendrá un origen que será el mismo que el de la categoría del campo referenciado por "**USING**", que no será un campo auxiliar. El resultado de la última transformada será un entero.

22.4.2.6 La propiedad de codificación "**USING**", si está presente, será una referencia a un campo que está presente en la codificación antes que el campo que se codifica. Si, en una instancia de la codificación, el campo que se codifica está presente pero el campo referenciado por la propiedad de codificación "**USING**" está ausente (por el ejercicio de la opcionalidad), se trata de un error de la especificación ECN o de la aplicación.

22.4.2.7 Si "**DETERMINED-BY**" es "**container**", "**USING**" hará referencia a una concatenación o a una repetición (o a una cadena de bits o una cadena de octetos con un tipo contenido) en la que el elemento que se codifica es un componente (o un componente de un componente, hasta cualquier profundidad). Si, en una instancia de codificación, han de ser codificados elementos dentro de la misma concatenación o repetición, se trata de un error de la especificación ECN.

22.4.2.8 Se considera que esta especificación está fijada si se utiliza la palabra clave "**ENCODING-SPACE**", y es obligatorio fijarla en todos los sitios de la sintaxis definida en donde esté permitida. La utilización de valores por defecto para todas las propiedades de codificación de este grupo (por ejemplo, la utilización de "**ENCODING-SPACE**" solo) no satisfaría las constricciones anteriores.

22.4.3 Acciones de codificador

22.4.3.1 Los codificadores no generarán codificaciones si no se cumplen las condiciones de 22.4.2.

22.4.3.2 Si "**SIZE**" es un valor positivo, el espacio de codificación es ese múltiplo de "**MULTIPLE OF**" unidades y no hay más acciones de codificador.

22.4.3.3 Si "**SIZE**" no está fijado en un valor positivo, el codificador determinará el tamaño del espacio de codificación (digamos, "s") en "**MULTIPLE OF**" unidades a partir del valor de la especificación de la codificación. Esta determinación se especifica en las cláusulas relativas a la especificación de la codificación de valores.

22.4.3.4 Si "**SIZE**" es "**encoder-option-with-determinant**", el codificador (como una opción de codificador) puede incrementar el tamaño "s" (determinado en 22.4.3.3) en "**MULTIPLE OF**" unidades desde el determinado a partir de la especificación de la codificación del valor hasta cualquier valor que pueda ser codificado en el determinante asociado.

22.4.3.5 Si "**SIZE**" es "**fixed-to-max**" o "**self-delimiting-values**", no hay más acciones de codificador.

22.4.3.6 Si "**SIZE**" es "**variable-with-determinant**" y "**DETERMINED BY**" es "**container**", no hay más acciones de codificador.

22.4.3.7 Si "**DETERMINED BY**" es "**field-to-be-set**", el codificador aplicará las transformadas especificadas por "**ENCODER-TRANSFORMS**" (si las hay) al valor "s" para producir un valor que será codificado en la referencia "**USING**".

NOTA – La codificación de la referencia "**USING**" (digamos, campo de bits "A") aparece en este caso en la codificación antes que la codificación de este campo (digamos, campo de bits "B"), y un codificador necesitará diferir la codificación del campo de bits "A" hasta que el valor que se ha de codificar haya sido determinado por la codificación del campo de bits "B".

22.4.3.8 Si "**DETERMINED BY**" es "**field-to-be-set**", el codificador comprobará que el valor de la referencia "**USING**" cuando es transformado por las "**DECODER-TRANSFORMS**" (si las hay), es igual a "s". Si esta condición no se cumple, se trata de un error de la aplicación y la codificación no avanzará.

22.4.4 Acciones de decodificador

22.4.4.1 Si "**SIZE**" es un valor positivo, el decodificador determina que el espacio de codificación es ese múltiplo de "**MULTIPLE OF**" unidades.

22.4.4.2 Si "**SIZE**" es "**fixed-to-max**" o "**self-delimiting-values**", el codificador determinará el final del espacio de codificación de acuerdo con la especificación de la codificación del valor. Esta determinación se especifica en las cláusulas relativas a la especificación de la codificación de valores.

22.4.4.3 Si "**SIZE**" es "**variable-with-determinant**" y "**DETERMINED BY**" está fijado en "**container**", el codificador utilizará el final del contenedor especificado por "**USING**" como el final del espacio de codificación.

22.4.4.4 Si "**SIZE**" es "**variable-with-determinant**" y "**DETERMINED BY**" está fijado en (o toma el valor por defecto de) "**field-to-be-set**", el codificador recuperará el valor "s" aplicando la inversión de las "**ENCODER-TRANSFORMS**" (si las hay) al valor de la referencia "**USING**".

22.4.4.5 Si "**DETERMINED BY**" es "**fiel-to-be-used**", el decodificador recuperará el valor "s" aplicando las "**DECODER-TRANSFORMS**" (si las hay) al valor de ese campo.

22.5 Determinación de opcionalidad

22.5.1 Propiedades de codificación, sintaxis y objetivo

22.5.1.1 La determinación de opcionalidad utiliza las propiedades de codificación siguientes:

```
&optionality-determination      OptionalityDetermination
                                DEFAULT field-to-be-set,
&optionality-reference          REFERENCE OPTIONAL,
&Encoder-transforms             #TRANSFORM ORDERED OPTIONAL,
&Decoder-transforms             #TRANSFORM ORDERED OPTIONAL,
&handle-id                      PrintableString
                                DEFAULT "default-handle"
```

22.5.1.2 La sintaxis que se ha de utilizar en la determinación de opcionalidad será:

```
PRESENCE
      [DETERMINED BY &optionality-determination
            [HANDLE &handle-id]]
      [USING &optionality-reference
            [ENCODER-TRANSFORMS &Encoder-transforms]
            [DECODER-TRANSFORMS &Decoder-transforms]]
```

22.5.1.3 La determinación de los tipos utilizados en la determinación de opcionalidad es:

```
OptionalityDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, container, handle, pointer} --
      (véase 21.5)
```

22.5.1.4 El objetivo de esta especificación es especificar las reglas con las que se asegura que un decodificador puede determinar correctamente si un codificador ha codificado un valor de un componente. Cuando se utiliza un puntero para determinar opcionalidad, se requiere además la especificación de alineación y puntero de comienzo.

22.5.1.5 Un codificador codificará el valor de un componente opcional si así lo requiere la aplicación, a menos que esa codificación incumpla las reglas que rigen la presencia de componentes opcionales.

NOTA – Un ejemplo de incumplimiento de esas reglas sería el caso en que la presencia de un componente opcional (ausente) tuviera que ser determinada por el final de un contenedor, y la aplicación pidiera que se codificasen componentes opcionales posteriores en el mismo contenedor.

22.5.1.6 Se considera que esta especificación está fijada si se utiliza la palabra clave "**PRESENCE**", y es obligatorio fijarla en todos los sitios de la sintaxis definida en donde esté permitida. La utilización de valores por defecto para todas las demás partes de esta sintaxis definida (por ejemplo, la utilización de "**PRESENCE**" solo) no satisfaría las constricciones anteriores.

22.5.2 Restricciones de la especificación

22.5.2.1 Si "**DETERMINED BY**" no está presente, se supone el valor por defecto ("**field-to-be-set**").

22.5.2.2 No se especificará "**HANDLE**" a menos que "**DETERMINED BY**" sea "**handle**".

22.5.2.3 No se especificará "**USING**" si "**DETERMINED BY**" es "**handle**" o "**pointer**".

22.5.2.4 Si "**DETERMINED BY**" es "**pointer**", habrá una especificación "**START-POINTER**" en el mismo objeto de codificación (véase 22.3).

NOTA – Una especificación de comienzo de puntero también necesita normalmente una especificación de prealineación con "**ALIGNED TO ANY**" (véase 22.2).

22.5.2.5 Si se especifica "**HANDLE**", el componente cuya presencia se determina, junto con todas las codificaciones opcionales siguientes y la próxima codificación obligatoria (si la hay) serán producidos por objetos de codificación todas cuyas especificaciones exhiben un asa de identificación con el mismo nombre que "**HANDLE**". La próxima codificación obligatoria puede ser un componente de la concatenación que contiene el componente opcional, o puede ser una codificación que sigue a la concatenación. El valor del asa de identificación será diferente para todos estos componentes.

NOTA – Es preciso que los bits que forman un asa de identificación tengan el mismo valor para todos los valores abstractos codificados por un objeto de codificación que exhibe ese asa de identificación (véase 22.9.2.2).

22.5.2.6 "**ENCODER-TRANSFORMS**" sólo estará presente si "**DETERMINED BY**" se ha fijado en (o toma el valor por defecto de) "**fixed-to-be-set**". En este caso, la referencia "**USING**" será un campo auxiliar de la categoría cadena de bits, booleano, cadena de octetos o entero.

22.5.2.7 Si una transformada de las "**ENCODER-TRANSFORMS**" no es reversible para el valor abstracto al que se aplica, se trata de un error de la especificación ECN o de la aplicación. La primera transformada tendrá un origen booleano y la última transformada deberá tener un resultado que pueda ser codificado por la clase del campo referenciado por "**USING**".

22.5.2.8 "**DECODER-TRANSFORMS"** sólo estará presente si "**DETERMINED BY**" se ha fijado en "**field-to-be-used**". La primera transformada tendrá un origen que será el mismo que el de la categoría del campo referenciado por "**USING**", que no será un campo auxiliar. El resultado de última transformada será booleano.

22.5.2.9 La propiedad de codificación "**USING**", si está presente, será una referencia a un campo que está presente en la codificación antes que el campo cuya presencia se determina. Si, en una instancia de la codificación, el campo referenciado por la propiedad de codificación "**USING**" es requerido por un codificador pero está ausente (por el ejercicio de la opcionalidad), se trata de un error de la especificación ECN o de la aplicación.

22.5.2.10 Si "**DETERMINED BY**" es "**container**", "**USING**" hará referencia a una concatenación o una repetición (o a una cadena de bits o cadena de octetos con un tipo contenido) en la que el elemento que se codifica es un componente (o un componente de un componente, hasta cualquier profundidad). Si, en una instancia de codificación, han de ser codificados elementos posteriores dentro de la misma concatenación o repetición cuando el elemento cuya opcionalidad se determina está ausente, se trata de un error de la especificación ECN o de la aplicación.

22.5.2.11 Si "**DETERMINED BY**" es "**container**" y cualquiera de los valores abstractos del componente opcional tiene una codificación que es cero bits, se trata de un error de la especificación ECN.

22.5.3 Acciones de codificador

22.5.3.1 Los codificadores no generarán codificaciones si no se cumplen las condiciones de 22.5.2.

22.5.3.2 Un codificador determinará si la aplicación desea que se codifique el componente opcional y creará un valor booleano conceptual "**element-is-present**" fijado en "**TRUE**" si se ha de codificar un valor del componente y, de no ser así, en "**FALSE**".

22.5.3.3 Si "**DETERMINED BY**" es "**field-to-be-set**", el codificador aplicará las transformadas especificadas por las "**ENCODER-TRANSFORMS**" (si las hay) al valor booleano conceptual "**element-is-present**" para producir un valor que será codificado en la referencia "**USING**".

NOTA – En este caso, la codificación de la referencia "**USING**" aparece en la codificación antes que la codificación de este campo, y un codificador tendrá que suspender la codificación de ese campo hasta que el valor que se ha de codificar haya sido determinado por la codificación de este campo.

22.5.3.4 Si "**DETERMINED BY**" es "**field-to-be-set**", el codificador comprobará que el valor de la referencia "**USING**" cuando es transformado por las "**DECODER-TRANSFORMS**" (si las hay) es un valor booleano igual al valor conceptual "**element-is-present**". Si esta condición no se cumple, se trata de un error de la aplicación y la codificación no avanzará.

22.5.3.5 Si "**DETERMINED BY**" es "**container**", no es preciso que el codificador realice más acciones salvo detectar un error y terminar la codificación si la aplicación pide la codificación de más componentes del contenedor "**USING**" cuando el valor conceptual "**element-is-present**" es falso para este componente opcional.

22.5.3.6 Si "**DETERMINED BY**" es "**handle**", no es preciso que el codificador realice más acciones.

22.5.3.7 Si "**DETERMINED BY**" es "**pointer**", no es preciso que el codificador realice más acciones salvo las de las especificaciones de prealineación acompañante (si las hay) y puntero de comienzo.

22.5.4 Acciones de decodificador

22.5.4.1 Si "**DETERMINED BY**" se ha fijado en (o toma el valor por defecto de) "**field-to-be-set**", el decodificador recuperará el valor "**element-is-present**" aplicando la inversa de las "**ENCODER-TRANSFORMS**" (si las hay) al valor de la referencia "**USING**".

22.5.4.2 Si "**DETERMINED BY**" es "**field-to-be-used**", el decodificador recuperará el valor conceptual "**element-is-present**" aplicando las "**DECODER-TRANSFORMS**" (si las hay) al valor de ese campo.

22.5.4.3 Si "**DETERMINED BY**" es "**container**", el decodificador fijará el valor conceptual "**element-is-present**" en "**TRUE**" si, y solamente si, hay por lo menos un bit restante en el contenedor "**USING**".

22.5.4.4 Si "**DETERMINED BY**" es "**handle**", el decodificador determinará el valor del asa de identificación especificada. Si el valor concuerda con el valor del asa de identificación del componente opcional, el decodificador fijará el valor conceptual "**element-is-present**" en "**TRUE**"; de otro modo, lo fijará en "**FALSE**".

22.5.4.5 Si "**DETERMINED BY**" es "**pointer**", el decodificador procederá como se especifica en 22.3 para determinar el valor conceptual "**element-is-present**".

22.5.4.6 Si el decodificador determina (por cualquiera de los procedimientos anteriores) que el valor conceptual "**element-is-present**" es "**FALSE**", la decodificación avanza al componente siguiente; de otro modo, el decodificador espera la decodificación de un valor del componente opcional y diagnosticará un error de codificación si no hay ninguno presente.

22.6 Determinación de alternativa

22.6.1 Propiedades de codificación, sintaxis y objetivo

22.6.1.1 La determinación de alternativa utiliza las propiedades siguientes:

```
&alternative-determination      AlternativeDetermination
                                DEFAULT field-to-be-set,
&alternative-reference          REFERENCE OPTIONAL,
&Encoder-transforms             #TRANSFORM ORDERED OPTIONAL,
&Decoder-transforms             #TRANSFORM ORDERED OPTIONAL,
&handle-id                      PrintableString
                                DEFAULT "default-handle",
&alternative-ordering           ENUMERATED {textual, tag}
                                DEFAULT textual
```

22.6.1.2 La sintaxis que se ha de utilizar para la determinación de alternativa será:

```
ALTERNATIVE
      [DETERMINED BY &alternative-determination
            [HANDLE &handle-id]]
      [USING &alternative-reference
            [ORDER &alternative-ordering]
            [ENCODER-TRANSFORMS &Encoder-transforms]
            [DECODER-TRANSFORMS &Decoder-transforms]]
```

22.6.1.3 La definición del tipo utilizado para la determinación de alternativa es como sigue:

```
AlternativeDetermination ::=
      ENUMERATED {field-to-be-set, field-to-be-used, handle} -- (véase 21.6)
```

22.6.1.4 El objetivo de esta especificación es determinar las reglas con las que se asegura que un decodificador puede identificar correctamente qué componente de una clase de codificación en la categoría alternativas ha sido codificado.

22.6.2 Restricciones de la especificación

22.6.2.1 Si "**DETERMINED BY**" no está presente, se supone el valor por defecto ("**field-to-be-set**").

22.6.2.2 No se especificará "**HANDLE**" a menos que "**DETERMINED BY**" sea "**handle**".

22.6.2.3 No se especificará "**USING**" si "**DETERMINED BY**" es "**handle**".

22.6.2.4 Si se especifica "**HANDLE**", todas las alternativas de la clase de codificación en la categoría alternativas serán codificadas por objetos de codificación cuya especificación exhibe y define un asa de identificación con el mismo nombre que "**HANDLE**", y con el mismo valor del asa de identificación. El valor del asa de identificación será diferente para todas estas alternativas.

NOTA – Es preciso que un asa de identificación tenga el mismo valor para todos los valores abstractos codificados por un objeto de codificación que exhibe ese asa de identificación (véase 22.9.2.2).

22.6.2.5 "**ENCODER TRANSFORMS**" sólo estará presente si "**DETERMINED BY**" se ha fijado en (o toma el valor por defecto de) "**field-to-be-set**". La primera transformada tendrá un origen que será un entero y la última transformada deberá tener un resultado que pueda ser codificado por la clase del campo referenciado por "**USING**".

22.6.2.6 Si cualquier transformada de las "**ENCODER-TRANSFORMS**" no es reversible para el valor abstracto al que se aplica, se trata de un error de la especificación ECN o de la aplicación.

22.6.2.7 "**DECODER-TRANSFORMS**" sólo estará presente si "**DETERMINED BY**" se ha fijado en "**field-to-be-used**". La primera transformada tendrá un origen que será el mismo que el de la categoría del campo referenciado por "**USING**", que no será un campo auxiliar. La última transformada tendrá un resultado que será un entero.

22.6.2.8 La propiedad de codificación "**USING**", si está presente, será una referencia a un campo que está presente en la codificación antes que la codificación de la alternativa. Si, en una instancia de codificación, el campo referenciado por la propiedad de codificación "**USING**" es requerido por un decodificador pero está ausente (por el ejercicio de la opcionalidad), se trata de un error de la especificación ECN o de la aplicación.

22.6.2.9 Se considera que esta especificación está fijada si se utiliza la palabra clave "**ALTERNATIVE**", y es obligatorio fijarla en todos los sitios de la sintaxis definida en donde esté permitida. La utilización de valores por defecto para todas las demás partes de esta sintaxis definida (por ejemplo, la utilización de "**ALTERNATIVE**" solo) no satisfaría las constricciones anteriores.

22.6.2.10 Si "**ORDER**" es "**tag**", cada alternativa empezará con una clase de codificación en la categoría rótulo. Al número de rótulo asociado a esta clase se le denomina rótulo de componente.

22.6.2.11 Los rótulos de componente de cada alternativa serán distintos.

22.6.3 Acciones de codificador

22.6.3.1 Los codificadores no generarán codificaciones si no se cumplen las condiciones de 22.6.2.

22.6.3.2 Un codificador determinará qué alternativa desea la aplicación que se codifique, y creará un valor entero conceptual "**alternative-index**" para identificar esa alternativa.

22.6.3.3 El valor "**alternative-index**" será cero para la primera alternativa, uno para la siguiente, y así sucesivamente, en donde el orden de las alternativas es determinado por "**ORDER**".

22.6.3.4 Si "**ORDER**" es "textual", se utilizará el orden textual de la especificación de tipo ASN.1 o la definición de estructura ECN. Si "**ORDER**" es "**tag**", el orden será el de los números de rótulo de los rótulos de componente (primero el número de rótulo más bajo).

22.6.3.5 Si "**DETERMINED BY**" es "**field-to-by-set**", el codificador aplicará las transformadas especificadas por las "**ENCODER-TRANSFORMS**" (si las hay) al valor conceptual "**alternative-index**" para producir un valor que será codificado en la referencia "**USING**".

NOTA – En este caso, la codificación de la referencia "**USING**" aparece en la codificación antes que la codificación de la alternativa, y un codificador tendrá que suspender la codificación de ese campo hasta que la alternativa que se ha de codificar haya sido determinada.

22.6.3.6 Si "**DETERMINED BY**" es "**field-to-be-used**", el codificador comprobará que el valor de la referencia "**USING**" cuando es transformado por las "**DECODER TRANSFORMS**" (si las hay) es un valor entero igual al valor conceptual "**alternative-index**". Si esta condición no se cumple, se trata de un error de la aplicación y la codificación no avanzará.

22.6.3.7 Si "**DETERMINED BY**" es "**handle**", no es preciso que el codificador realice más acciones.

22.6.4 Acciones de decodificador

22.6.4.1 El decodificador utilizará "**ORDER**", tal como se especifica para las acciones de codificador, para determinar el valor del **índice de alternativa** que está asociado a cada alternativa y supondrá la presencia de una codificación de la alernativa asociada una vez que se haya determinado un valor conceptual "**alternative-index**".

22.6.4.2 Si "**DETERMINED BY**" se ha fijado en (o toma el valor por defecto de) "**field-to-be-set**", el decodificador recuperará el valor "**alternative-index**" aplicando la inversa de las "**ENCODER-TRANSFORMS**" (si las hay) al valor de la referencia "**USING**".

22.6.4.3 Si "**DETERMINED BY**" es "**field-to-be-used**", el decodificador recuperará el valor conceptual "**alternative-index**" aplicando las "**DECODER TRANSFORMS**" (si las hay) al valor de ese campo.

22.6.4.4 Si "**DETERMINED BY**" es "**handle**", el decodificador determinará el valor del asa de identificación. Este valor será comparado con el valor del asa de identificación de cada una de las alternativas. Si no concuerda con ninguno, el decodificador diagnosticará un error de codificador. De otro modo, se fijará el valor conceptual "**alternative-index**" en la alternativa concordante.

22.7 Especificación de espacio de repetición

22.7.1 Propiedades de codificación, sintaxis y objetivo

22.7.1.1 La especificación de espacio de repetición utiliza las propiedades de codificación siguientes:

```
&repetition-space-size            EncodingSpaceSize
                                  DEFAULT self-delimiting-values,
&repetition-space-unit            Unit
                                  DEFAULT bit,
&repetition-space-determination   RepetitionSpaceDetermination
                                  DEFAULT field-to-be-set,
&main-reference                   REFERENCE OPTIONAL,
&Encoder-transforms               #TRANSFORM ORDERED OPTIONAL,
&Decoder-transforms               #TRANSFORM ORDERED OPTIONAL,
&handle-id                        PrintableString
                                  DEFAULT "default-handle",
&termination-pattern              Non-Null-Pattern (ALL EXCEPT
                                  different:any) DEFAULT bits '0'B
```

22.7.1.2 La sintaxis que se ha de utilizar para la especificación de espacio de repetición será:

```
REPETITION-SPACE
      [SIZE &repetition-space-size
            [MULTIPLE OF &repetition-space-unit]]
      [DETERMINED BY &repetition-space-determination
            [HANDLE &handle-id]]
      [USING &main-reference
            [ENCODER-TRANSFORMS &Encoder-transforms]
            [DECODER-TRANSFORMS &Decoder-transforms]]
      [PATTERN &termination-pattern]
```

22.7.1.3 Las definiciones de los tipos utilizados en esta especificación son como sigue:

```
EncodingSpaceSize ::= INTEGER
      { encoder-option-with-determinant(-3),
        variable-with-determinant(-2),
        self-delimiting-values(-1),
        fixed-to-max(0)} (-3..MAX) -- (véase 21.2)

Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
       dword32(32)} (0..256) --  (véase 21.1)

RepetitionSpaceDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, flag-to-be-set, flag-to-be-used,
       container, pattern, handle, not-needed} -- (véase 21.7)

Non-Null-Pattern ::= Pattern
      (ALL EXCEPT (bits:''B | octets:''H | char8:"" | char16:"" |
                   char32:"")) -- (véase 21.10.2)
```

22.7.1.4 El objetivo de esta especificación es determinar las acciones de codificador y decodificador para asegurar que un decodificador puede determinar correctamente el final del espacio de codificación ocupado por una repetición.

NOTA – Una codificación de repetición real no necesariamente tiene que llenar todo el espacio de codificación, y la recuperación de la codificación de la repetición por un decodificador requerirá además, por lo general, acciones especificadas para el relleno del valor y la justificación (véase 22.8).

22.7.1.5 El significado de las propiedades de codificación de los tipos "**Unit**", "**EncodingSpaceSize**" y "**RepetitionSpaceDetermination**" se dio en 21.1, 21.2 y 21.7. En conjunto especifican la manera en que se determina el final del espacio de codificación de la repetición.

NOTA – Si el especificador de la ECN requiere que se incluya un determinante de longitud, se puede especificar el valor "**variable-with-determinant**" de "**SIZE**" incluso si el espacio de repetición tiene un tamaño fijo.

22.7.1.6 La especificación "**USING**" es una referencia a un campo auxiliar o a un campo que lleva valores abstractos, o a un contenedor, dependiendo del valor de "**DETERMINED BY**".

22.7.2 Constricciones de la especificación

22.7.2.1 Si "**SIZE**" es "**variable-with-determinant**" y "**DETERMINED BY**" no está presente, se supone el valor por defecto ("**field-to-by-set**").

22.7.2.2 "**USING**" será especificado si, y solamente si, "**SIZE**" es "**variable-with-determinant**" y "**DETERMINED BY**" es "**field-to-be-set**" o "**field-to-be-used**" o "**flag-to-be-used**" o "**container**".

22.7.2.3 "**ENCODER TRANSFORMS**" sólo estará presente si "**DETERMINED BY**" se ha fijado en (o toma el valor por defecto de) "**field-to-be-set**" o"**flag-to-be-set**". La primera transformada tendrá un origen que será un entero si "**DETERMINED BY**" es "**field-to-be-set**" y será booleano si "**DETERMINED BY**" es "**flag-to-be-set**". La última transformada deberá tener un resultado que pueda ser codificado por la clase del campo referenciado por "**USING**".

22.7.2.4 Si cualquier transformada de las "**ENCODER TRANSFORMS**" no es reversible para el valor abstracto al que se aplica, se trata de un error de la especificación ECN o de la aplicación.

22.7.2.5 "**DECODER TRANSFORMS**" sólo estará presente si "**DETERMINED BY**" se ha fijado en "**field-to-be-used**" o "**flag-to-be-used**". La primera transformada tendrá un origen que será el mismo que el de la categoría del campo referenciado por "**USING**". La última transformada tendrá un resultado que será un entero si "**DETERMINED BY**" es "**field-to-be-used**" y será booleano si "**DETERMINED BY**" es "**flag-to-be-used**".

22.7.2.6 La propiedad de codificación "**USING**", si está presente, será una referencia para "**field-to-be-set**" o "**field-to-be-used**" a un campo que está presente en la codificación antes que el campo que se codifica. Si, en una instancia de la codificación, la repetición que se codifica está presente pero el campo referenciado por la propiedad de codificación "**USING**" está ausente (por el ejercicio de la opcionalidad), se trata de un error de la especificación ECN o de la aplicación.

22.7.2.7 La propiedad de codificación "**USING**", si está presente, será una referencia para "**flag-to-be-set**" o "**flag-to-be-used**" a un campo presente en el elemento repetido de una repetición. Si, en una instancia de la codificación, el campo referenciado por la propiedad de codificación "**USING**" está ausente (por el ejercicio de la opcionalidad) de cualquiera de los elementos repetidos, se trata de un error de la especificación ECN o de la aplicación.

NOTA – El requisito de que el campo referenciado esté presente en un elemento de la repetición se satisface si lo que está visible es un identificador de acuerdo con 17.5 (codificación de una estructura de codificación), 19.3 (establecimiento de la correspondencia mediante campos concordantes), 19.6 (establecimiento de la correspondencia mediante la distribución de valores), o si está textualmente presente en la definición de una estructura de sustitución cuando "**REPLACE COMPONENT**" es utilizado por un objeto de codificación de una clase en la categoría repetición.

22.7.2.8 Si "**DETERMINED BY**" es "**container**", "**USING**" se referirá a una concatenación o a una repetición (o a una cadena de bits o cadena de octetos con un tipo contenido) en la que la repetición que se codifica es un componente (o un componente de un componente, hasta cualquier profundidad). Si, en una instancia de codificación, van a ser codificados elementos posteriores dentro de la misma concatenación o repetición, se trata de un error de la especificación ECN o de la aplicación.

22.7.2.9 "**HANDLE**" sólo se especificará si "**SIZE**" es "**variable-with-determinant**" y "**DETERMINED BY**" es "**handle**".

22.7.2.10 Si se especifica "**HANDLE**", el elemento repetido, junto con cualquier elemento que (por el ejercicio de la opcionalidad) pueda seguir al elemento repetido, serán codificados por objetos de codificación cuya especificación exhibe un asa de identificación con el mismo nombre que"**HANDLE**". El valor del asa de identificación en el elemento que se repite será diferente del de cualquier posible elemento siguiente.

NOTA – Es preciso que un asa de identificación tenga el mismo valor para todos los valores abstractos codificados por un objeto de codificación que exhibe ese asa de identificación (véase 22.9.2.2).

22.7.2.11 "**PATTERN**" sólo se especificará si "**SIZE**" es "**variable-with-determinant**" y "**DETERMINED BY**" es "**PATTERN**".

22.7.2.12 "**PATTERN**" no será la subcadena inicial de la codificación de ningún valor del elemento repetido.

NOTA – No se prohíbe la ocurrencia de "**PATTERN**" dentro de una codificación del elemento repetido distinta de la de su comienzo.

22.7.2.13 Se considera que esta especificación está fijada si se utiliza la palabra clave "**REPETITION-SPACE**", y es obligatorio fijarla en todos los sitios de la sintaxis definida en donde esté permitida. La utilización de valores por defecto para todas las demás partes de esta sintaxis definida (por ejemplo, la utilización de "**REPETITION-SPACE**" solo) no satisfaría las constricciones anteriores.

22.7.3 Acciones de codificador

22.7.3.1 Los codificadores no generarán codificaciones si no se cumplen las condiciones de 22.7.2.

22.7.3.2 Si "**SIZE**" es un valor positivo, el espacio de codificación es ese múltiplo de "**MULTIPLE OF**" unidades. Si "**MULTIPLE OF**" es repeticiones, el codificador dejará de codificar si el valor abstracto que va a ser codificado no es "**SIZE**" repeticiones diagnosticando un error de especificación o aplicación.

22.7.3.3 Si "**SIZE**" no está fijado en un valor positivo, el codificador determinará el tamaño "s" del espacio de repetición en "**MULTIPLE OF**" unidades a partir de la especificación de codificación del valor. Esta determinación se especifica en las subcláusulas relativas a la especificación de la codificación de valores.

22.7.3.4 Si "**SIZE**" es "**encoder-option-with-determinant**", el codificador (como una opción del codificador) puede incrementar el tamaño el tamaño "s" (determinado en 22.7.3.3) en "**MULTIPLE OF**" unidades desde el determinado a partir de la especificación de la codificación del valor hasta cualquier valor que pueda ser codificado en el determinante asociado.

22.7.3.5 Si "**SIZE**" es "**fixed-to-max**" o "**self-delimiting-values**", no hay más acciones de codificador.

22.7.3.6 Si "**SIZE**" es "**variable-with-determinant**" y "**DETERMINED BY**" es "**container**", no hay más acciones de codificador.

22.7.3.7 Si "**DETERMINED BY**" es "**field-to-be-set**", el codificador aplicará las transformadas especificadas por las "**ENCODER-TRANSFORMS**" (si las hay) al valor "s" para producir un valor que será codificado en la referencia "**USING**".

NOTA – En este caso, la codificación de la referencia "**USING**" aparece en la codificación antes que la codificación de la repetición, y un codificador tendrá que suspender la codificación de ese campo hasta que la repetición que se ha de codificar haya sido determinada.

22.7.3.8 Si "**DETERMINED BY**" es "**field-to-be-used**", el codificador comprobará que el valor de la referencia "**USING**" cuando es transformado por las "**DECODER-TRANSFORMS**" (si las hay) es igual a "s". Si esta condición no se cumple, se trata de un error de la aplicación y la codificación no avanzará.

22.7.3.9 Si "**DETERMINED BY**" es "**flag-to-be-set**", el codificador aplicará (para cada elemento repetido) las transformadas especificadas por las "**ENCODER-TRANSFORMS**" (si las hay) a un valor booleano que es verdadero para todos los elementos excepto el último y es falso para el último elemento. El resultado de las "**ENCODER-TRANSFORMS**" será codificado en la referencia "**USING**".

22.7.3.10 Si "**DETERMINED BY**" es "**flag-to-be-used**", el codificador comprobará (para cada elemento repetido) que el valor de la referencia "**USING**" cuando es transformado por las "**DECODER-TRANSFORMS**" (si las hay) es un valor booleano que es verdadero para todos los elementos excepto el último y es falso para el último elemento. Si esta condición no se cumple, se trata de un error de la aplicación y la codificación no avanzará.

22.7.3.11 Si "**DETERMINED BY**" es "**handle**", no hay más acciones de codificador.

22.7.3.12 Si "**DETERMINED BY**" es "**pattern**", el codificador comprobará que el esquema especificado no es una subcadena inicial de ninguna de las codificaciones del elemento repetido y dejará de codificar si esta codificación falla, diagnosticando un error de la especificación o la aplicación. El codificador añadirá el esquema "**PATTERN**" al final de la codificación de la repetición.

22.7.4 Acciones de decodificador

22.7.4.1 Si "**SIZE**" es un valor positivo, el decodificador determina que el espacio de codificación es ese múltiplo de "**MULTIPLE OF**" unidades. Si "**MULTIPLE OF**" es repeticiones, el final real del espacio de repetición se determina decodificando y contando las repeticiones.

22.7.4.2 Si "**SIZE**" no está fijado en un valor positivo, el codificador determinará el tamaño "s" del espacio de repetición en "**MULTIPLE OF**" unidades a partir de la especificación de la codificación del valor. Esta determinación se especifica en las subcláusulas relativas a la especificación de la codificación de los valores.

22.7.4.3 Si "**SIZE**" es "**variable-with-determinant**" y "**DETERMINED BY**" está fijado en "**container**", el decodificador utilizará el final del contenedor especificado por "**USING**" como final del espacio de codificación.

22.7.4.4 Si "**SIZE**" es "**variable-with-determinant**" y "**DETERMINED BY**" ha sido fijado en (o toma el valor por defecto de) "**field-to-be-set**", el decodificador recuperará el valor "s" aplicando la inversa de las "**ENCODER-TRANSFORMS**" (si las hay) al valor de la referencia "**USING**".

22.7.4.5 Si "**DETERMINED BY**" es "**field-to-be-used**", el decodificador recuperará el valor "s" aplicando las "**DECODER-TRANSFORMS**" (si las hay) al valor de la referencia "**USING**".

22.7.4.6 Si "**DETERMINED BY**" es "**flag-to-be-set**", el decodificador recuperará un valor booleano aplicando la inversa de las "**ENCODER-TRANSFORMS**" (si las hay) al valor de la referencia "**USING**". El elemento es el último de la repetición si, y solamente si, el valor booleano es falso.

22.7.4.7 Si "**DETERMINED BY**" es "**flag-to-be-used**", el decodificador recuperará un valor booleano aplicando las "**DECODER TRANSFORMS**" (si las hay) al valor de la referencia "**USING**". El elemento es el último de la repetición si, y solamente si, el valor booleano es falso.

22.7.4.8 Si "**DETERMINED BY**" es "**handle**", el decodificador determinará el valor del asa de identificación y tratará de decodificar el próximo elemento (en paralelo) como una repetición nueva o como un elemento siguiente, utilizando el asa de identificación para distinguir esas alternativas. Si la decodificación tiene éxito con más de una de ellas o con ninguna, se trata de un error de la codificación o de la especificación.

22.7.4.9 Si "**DETERMINED BY**" es "pattern", el decodificador comprobará, al comienzo de la decodificación de cada repetición, si está presente "**PATTERN**". Si "**PATTERN**" está presente, los bits del esquema serán descartados y se terminará la repetición.

22.8 Relleno y justificación de valor

22.8.1 Propiedades de codificación, sintaxis y objetivo

22.8.1.1 Relleno y justificación de valor utiliza las propiedades de codificación siguientes:

```
&value-justification              Justification DEFAULT right:0,
&value-pre-padding                Padding DEFAULT zero,
&value-pre-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
&value-post-padding               Padding DEFAULT zero,
&value-post-pattern               Non-Null-Pattern DEFAULT bits:'0'B,
&unused-bits-determination        UnusedBitsDetermination
                                  DEFAULT field-to-be-set,
&unused-bits-reference            REFERENCE OPTIONAL,
&Unused-bits-encoder-transforms   #TRANSFORM ORDERED OPTIONAL,
&Unused-bits-decoder-transforms   #TRANSFORM ORDERED OPTIONAL
```

22.8.1.2 La sintaxis que se ha de utilizar para relleno y justificación de valor será:

```
[VALUE-PADDING
      [JUSTIFIED &value-justification]
      [PRE-PADDING &value-pre-padding
            [PATTERN &value-pre-pattern]]
      [POST-PADDING &value-post-padding
            [PATTERN &value-post-pattern]]
      [UNUSED BITS
            [DETERMINED BY &unused-bits-determination]
            [USING &unused-bits-reference
                  [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                  [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
```

22.8.1.3 Las definiciones de los tipos utilizados en justificación son como sigue:

```
Justification ::= CHOICE
               { left             INTEGER (0..MAX),
                 right            INTEGER (0..MAX)} -- (véase 21.8)

Padding ::= ENUMERATED {zero, one, pattern, encoder-option} -- (véase 21.9)

Pattern ::= CHOICE
      {bits                       BIT STRING,
       octets                     OCTET STRING,
       char8                      IA5String,
       char16                     BMPString,
       char32                     UniversalString,
       any-of-length              INTEGER (1..MAX),
       different                  ENUMERATED {any} }
```

```
Non-Null-Pattern ::= Pattern
      (ALL EXCEPT (bits:''B | octets:''H | char8:"" | char16:"" |
                   char32:"")) -- (véase 21.10)

UnusedBitsDetermination ::= ENUMERATED
      {field-to-be-set, field-to-be-used, not-needed} -- (véase 21.4)
```

22.8.1.4 El objetivo de esta especificación es determinar la manera en que un codificador pone la codificación de un valor en un espacio de codificación, y hace posible que un decodificador determine la codificación del valor.

22.8.1.5 El número exacto de bits que ha de añadir un codificador depende de la especificación del espacio de codificación y de la especificación de la codificación del valor, y se especifica para cada instancia de codificación de valor.

22.8.1.6 "**USING**" es una referencia que permite a un codificador determinar el número de bits de relleno insertados. Es una referencia a un campo auxiliar o a un campo que lleva valores abstractos, dependiendo de "**DETERMINED BY**".

22.8.2 Restricciones de la especificación

22.8.2.1 El número de bits especificado en la justificación será menor o igual que el número total de bits de relleno "b" (véase más adelante).

22.8.2.2 "**USING**" se especificará si, y solamente si, "**DETERMINED BY**" no es "not-needed".

22.8.2.3 "**ENCODER-TRANSFORMS**" sólo estará presente si "**DETERMINED BY**" se ha fijado en (o toma el valor por defecto de) "**field-to-be-set**". La primera transformada tendrá un origen que será un entero y la última transformada deberá tener un resultado que pueda ser codificado por la clase del campo referenciado por "**USING**".

22.8.2.4 Si cualquier transformada de las "**ENCODER-TRANSFORMS**" no es reversible para el valor abstracto al que se aplica, se trata de un error de la especificación ECN o de la aplicación.

22.8.2.5 "**DECODER-TRANSFORMS**" sólo estará presente si "**DETERMINED BY**" se ha fijado en "**field-to-be-set**". La primera transformada tendrá un origen que será el mismo que el de la categoría del campo referenciado por "**USING**", que no será un campo auxiliar. La última transformada tendrá un resultado que será un entero.

22.8.2.6 La propiedad de codificación "**USING**", si está presente, será una referencia a un campo que está presente en la codificación antes que el campo que se codifica. Si, en una instancia de codificación, el campo que se codifica está presente pero el campo referenciado por la propiedad de codificación "**USING**" está ausente (por el ejercicio de la opcionalidad), se trata de un error de la especificación ECN o de la aplicación.

22.8.2.7 Esta codificación se considera fijada si se utiliza la palabra clave "**VALUE-PADDING**". Si no está fijada, se especifican acciones en todos los sitios en que esa sintaxis está permitida.

22.8.3 Acciones de codificador

22.8.3.1 Los codificadores no generarán codificaciones si no se cumplen las condiciones de 22.8.2.

22.8.3.2 Esta especificación se aplica si, y solamente si, el espacio de codificación o la especificación de la codificación del espacio de codificación, junto con la especificación de la codificación del valor, determinan que puede haber bits de relleno añadidos en torno a la codificación del valor o la repetición dentro del espacio de codificación o repetición. Sea "b" el número determinado de bits de relleno añadidos en una instancia de codificación (donde "b" es mayor o igual que cero).

22.8.3.3 Si "**JUSTIFIED**" es "**right:n**", se añadirán "b"-"n" bits de relleno previo antes de la codificación del valor o la repetición, y se añadirán "n" bits de relleno posterior.

22.8.3.4 Si "**JUSTIFIED**" es "**left:n**", se añadirán "n" bits de relleno previo antes de la codificación del valor o la repetición, y se añadirán "b"-"n" bits de relleno posterior.

22.8.3.5 Los bits de relleno se fijarán de acuerdo con las especificaciones "**PRE-PADDING**" y "**POST-PADDING**", siendo el bit inicial del esquema el primer bit insertado en cada caso.

22.8.3.6 Si "**DETERMINED BY**" es "not-needed", esto completa las acciones de los codificadores.

22.8.3.7 Si "**DETERMINED BY**" es "**field-to-be-set**", el codificador aplicará las transformadas especificadas por las "**ENCODER-TRANSFORMS**" (si las hay) al valor "b" para producir un valor que será codificado en la referencia "**USING**".

NOTA – En este caso, la codificación de la referencia "**USING**" aparece en la codificación antes que la codificación de este campo, y un codificador tendrá que suspender la codificación de ese campo hasta que el valor que se ha de codificar haya sido determinado por la codificación de este campo.

22.8.3.8 Si "**DETERMINED BY**" es "**field-to-be-used**", el codificador comprobará que el valor de la referencia "**USING**" cuando es transformado por las "**DECODER-TRANSFORMS**" (si las hay) es igual a "b". Si esta condición no se cumple, se trata de un error de la aplicación y la codificación no avanzará.

22.8.4 Acciones de decodificador

22.8.4.1 Si "**DETERMINED BY"** es "**not-needed**", el decodificador establecerá el valor "b" determinado por la especificación de la codificación del valor y la determinación del espacio de codificación o la repetición.

22.8.4.2 Si "**DETERMINED BY**" se ha fijado en (o toma el valor por defecto de) "**field-to-be-set**", el decodificador recuperará el valor "b" aplicando la inversa de las "**ENCODER-TRANSFORMS**" (si las hay) al valor de la referencia "**USING**".

22.8.4.3 Si "**DETERMINED BY**" es "**field-to-be-used**", el decodificador recuperará el valor "b" aplicando las "**DECODER-TRANSFORMS**" (si las hay) al valor de ese campo.

22.8.4.4 El decodificador utilizará "**JUSTIFIED**" y el valor "b" para determinar la posición de la codificación del valor dentro del espacio de codificación, e ignorará el valor de todos los bits de relleno.

22.9 Especificación de asa de identificación

22.9.1 Propiedades de codificación, sintaxis y objetivo

22.9.1.1 La especificación de asa de identificación utiliza las propiedades de codificación siguientes:

```
&exhibited-handle                PrintableString OPTIONAL,
&Handle-positions                INTEGER (0..MAX) OPTIONAL,
&handle-value                    HandleValue DEFAULT tag:any
```

22.9.1.2 La sintaxis que se ha de utilizar para la especificación del asa de identificación será:

```
[EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
      [AS &handle-value]]
```

22.9.1.3 La definición del tipo utilizado en la especificación de asa de identificación es como sigue:

```
HandleValue ::= CHOICE {
      bits        BIT  STRING,
      octets      OCTET STRING,
      number      INTEGER (0..MAX),
      tag         ENUMERATED {any}}  -- (véase 21.15)
```

22.9.1.4 Esta especificación se utiliza para señalar que un objeto de codificación exhibe un asa de identificación dentro de todas sus codificaciones (es decir, para todos los valores abstractos posibles que codifica). Se especifican el nombre del asa de identificación y los bits asociados a ese asa de identificación. El valor del asa de identificación es especificado por "**HandleValue**".

22.9.1.5 La lista de posiciones en "**AT**" constará de las posiciones de los bits que forman el asa de identificación en la codificación final, una vez que se haya aplicado cualquier prealineación, y después de que hayan tenido lugar cualesquiera acciones de inversión de bits del codificador, excepto las inversiones de bits resultantes de la especificación de un objeto de codificación en la clase **#OUTER**.

NOTA – Esto significa que un decodificador necesita efectuar cualesquiera inversiones de bits especificadas en **#OUTER** para la PDU completa, pero por otra parte examina las posiciones de los bits y sus valores sin tener en cuenta las inversiones de bits que puedan ser especificadas para objetos de codificación particulares

22.9.1.6 La lista de posiciones en "**AT**" es un conjunto de valores enteros (no necesariamente contiguos, y no necesariamente en orden ascendente en la especificación ECN). Estas posiciones serán ordenadas por los codificadores y decodificadores desde la posición cero (el primer bit en la parte de la codificación que exhibe el asa) hacia arriba, y los bits de esas posiciones forman un campo de asa conceptual.

22.9.1.7 Para un valor "**number**" de "**HandleValue**" o la codificación de un número de rótulo, el bit del campo de asa conceptual más cercano a la posición cero es el bit de orden superior, y el "**number**" o número de rótulo que especifica el "**HandleValue**" está justificado a la derecha dentro de ese campo. Si el "**number**" o número de rótulo es demasiado grande para el campo, se trata de un error de la especificación ECN.

22.9.1.8 Si se utilizan las alternativas "**bitstring**" u "**octectstring**" de "**HandleValue**", sus valores tendrán el mismo número de bits que los especificados para el asa de identificación por "**AT**". El bit del campo de asa conceptual más cercano a la posición cero es el bit inicial de la "**bitstring**" u "**octectstring**" que especifica el "**HandleValue**".

22.9.1.9 El "**HandleValue**" no será especificado como "**tag:any**" a menos que la especificación se refiera a un objeto de codificación de la clase **#TAG**. En este caso, el valor del asa de identificación es determinado por el número de rótulo de la especificación ECN o por el número de rótulo cuya correspondencia se ha establecido desde un rótulo ASN.1 (como se especifica en la cláusula 19), y no es preciso especificarlo utilizando "**HandleValue**". Si, no obstante, "**HandleValue**" especifica un valor que difiere del asignado en una especificación ECN de una clase de rótulo o en un rótulo ASN.1 que se corresponde con un rótulo ECN, se trata de un error de la especificación ECN.

22.9.2 Constricciones de la especificación

22.9.2.1 En cualquier aplicación de la ECN, todas las asas de identificación con el mismo nombre especificarán el mismo conjunto de bits para la ubicación del asa de identificación.

NOTA – No hay un requisito general de que el valor del asa de identificación (exhibida por diferentes objetos de codificación) deba ser distinto, pero se requieren valores distintos cuando el asa de identificación se utiliza para resolver opcionalidad, selección de alternativa o terminación de repetición (véanse 21.5.7, 21.6.6 y 21.7.10).

22.9.2.2 El especificador de la ECN asegurará que cualquier objeto de codificación que exhiba un asa de identificación produce el mismo valor del asa de identificación para todo valor abstracto que se codifique.

22.9.2.3 Todos los objetos de codificación que exhiban la misma asa de identificación tendrán especificación de no prealineación o alinearán la misma unidad de prealineación.

NOTA – Esta restricción se impone para que los decodificadores puedan pasar a la posición de alineación antes de buscar el asa cuando la decodificación depende de un valor de asa.

22.9.2.4 Si un objeto de codificación de una clase en la categoría repetición exhibe un asa de identificación, ese asa de identificación será exhibida también (con el mismo valor) por la codificación del elemento repetido.

22.9.2.5 Si un objeto de codificación de una clase en la categoría alternativas exhibe un asa de identificación, ese asa de identificación será exhibida también por (la codificación de) todas las alternativas, y el valor del asa de identificación será el mismo para todas las alternativas.

NOTA – En este caso, el asa de identificación no puede ser utilizada para la determinación de alternativa en esta alternativa, y la determinación de alternativa se tiene que hacer utilizando un asa de identificación diferente o por algún otro medio.

22.9.2.6 Si un objeto de codificación de una clase en la categoría concatenación exhibe un asa de identificación, el primer componente codificado (si lo hay o, si está rotulado, el rótulo), teniendo en cuenta la opcionalidad, exhibirá ese asa de identificación con el mismo valor.

22.9.2.7 Se considera que esta especificación está fijada si se utiliza la palabra clave "**EXHIBITS-HANDLE**". Si no está fijada no se exhibe ningún asa de identificación.

22.9.3 Acciones de codificador

22.9.3.1 Si un objeto de codificación exhibe un asa de identificación, el codificador comprobará que la codificación tiene el valor del asa de identificación, y si no es así diagnosticará un error de la especificación o de la aplicación.

22.9.4 Acciones de decodificador

22.9.4.1 No hay acciones de decodificador que sean consecuencia directa de la exhibición de un asa de identificación. Las acciones de los decodificadores sólo resultan de la utilización del asa de identificación para determinar opcionalidad, final de repeticiones o elección de alternativas.

22.10 Especificación de concatenación

22.10.1 Propiedades de codificación, sintaxis y objetivo

22.10.1.1 La especificación de concatenación especifica las propiedades de codificación siguientes:

```
&concatenation-order            ENUMERATED {textual, tag, random}
                                DEFAULT textual,
&concatenation-alignment        ENUMERATED {none, aligned}
                                DEFAULT aligned,
&concatenation-handle           PrintableString
                                DEFAULT "default-handle"
```

22.10.1.2 La sintaxis que se ha de utilizar para la especificación de concatenación es :

```
[CONCATENATION
      [ORDER &concatenation-order]
      [ALIGNMENT &concatenation-alignment]
      [HANDLE &concatenation-handle]]
```

22.10.1.3 Esta especificación determina el orden en que se codifican los componentes de una clase de codificación en la categoría concatenación, los medios que utiliza un codificador para identificar cada componente y el relleno de prealineación que se ha de proporcionar entre componentes.

22.10.2 Constricciones de la especificación

22.10.2.1 Si "**ORDER**" es "**random**", "**HANDLE**" toma el valor por defecto de "**default-handle**" en caso de que el valor no esté fijado, y todos los componentes exhibirán "**HANDLE**" con valores distintos del asa de identificación.

22.10.2.2 Si "**ALIGNMENT**" es "**aligned**", la especificación de prealineación toma el valor por defecto a menos que el valor esté fijado.

22.10.2.3 Si un componente tiene su propia prealineación explícita, se aplica ésta después de cualquier prealineación del componente resultante de la fijación de "**ALIGNMENT**" en la clase de codificación de la categoría concatenación.

NOTA – La función equivalente no se proporciona para repeticiones, ya que se puede conseguir de manera más sencilla mediante la prealineación del componente único.

22.10.2.4 Si "**ORDER**" es "**tag**", cada componente empezará con una clase de codificación en la categoría rótulo. Al número de rótulo asociado a esta clase se le denomina rótulo de componente.

22.10.2.5 Los rótulos de componente de cada alternativa serán distintos.

22.10.2.6 Se considera que esta especificación está fijada si se utiliza la palabra clave "**CONCATENATION**". Si no está fijada, los codificadores y decodificadores actúan como si estuviera fijada tomando cada propiedad de codificación su valor por defecto.

22.10.2.7 Si (por el ejercicio de la opcionalidad) hay al menos un valor abstracto de una concatenación que no tiene bits en su codificación, la concatenación no tendrá prealineación.

NOTA – Esta subcláusula se aplicará si una concatenación no tiene componentes obligatorios, o si es posible que (por el ejercicio de la opcionalidad) ninguno de sus componentes obligatorios tenga bits en su codificación.

22.10.3 Acciones de codificador

22.10.3.1 Si "**ORDER**" es "**textual**", se utilizará el orden textual en la especificación del tipo ASN.1 o la definición de la estructura ECN.

22.10.3.2 Si "**ORDER**" es "**tag**", el orden será el de los números de rótulo de los rótulos de componente (primero el número de rótulo más bajo).

22.10.3.3 Si "**ORDER**" es "**random**", el codificador determinará el orden de la concatenación sin constricciones.

22.10.3.4 Si "**ALIGNMENT**" es "**none**", el codificador yuxtapondrá las codificaciones de componentes con bits no insertados.

22.10.3.5 Si "**ALIGNMENT**" es "**aligned**", el codificador aplicará la especificación de prealineación de la clase en la categoría concatenación antes de codificar cada componente, con la salvedad de que una especificación de prealineación de "**ALIGNED TO ANY**" se interpretará como una especificación de "**ALIGNED TO NEXT**" (véase 22.2).

NOTA 1 – Esto se debe a que sólo puede haber un puntero de comienzo único para "**ALIGNED TO ANY**".

NOTA 2 – Cualquier prealineación especificada para un componente (incluyendo "**ALIGNED TO ANY**") se aplica después de las acciones anteriores.

22.10.4 Acciones de decodificador

22.10.4.1 Cuando se decodifique un componente, el decodificador efectuará primero las acciones de decodificador asociadas a la especificación de prealineación para "**ALIGNMENT**" si está fijado en "**aligned**", tratando "**ALIGNED TO ANY**" como "**ALIGNED TO NEXT**" (véase 22.2). Si "**ALIGNMENT**" está fijado en "**none**", el decodificador pasará directamente a decodificar el componente.

22.10.4.2 El decodificador determinará el orden de los componentes a partir del orden definido para el codificador si "**ORDER**" es "**textual**" o "**tag**".

22.10.4.3 Si "**ORDER**" es "**random**", el decodificador determinará el orden de los componentes examinando el valor de los bits asociados a "**HANDLE**".

22.10.4.4 Cada componente tiene un valor distinto para los bits asociados a "**HANDLE**" que permite que el componente sea identificado. La decodificación avanzará hasta que haya sido obtenido un valor abstracto para cada componente, y un decodificador diagnosticará un error de codificador si se identifica más de una codificación de un componente, o si aparecen valores no esperados de asas de identificación durante la decodificación.

NOTA – Valores no esperados pueden ocurrir como parte de la provisión de extensibilidad, pero esta versión de la presente Recomendación | Norma Internacional no contempla tal posibilidad, y esas ocurrencias se tratarán como errores de codificador.

22.11 Especificación de codificación de tipo contenido

22.11.1 Propiedades de codificación, sintaxis y objetivo

22.11.1.1 La especificación de codificación de tipo contenido utiliza las propiedades de codificación siguientes:

```
&Primary-encoding-object-set      #ENCODINGS OPTIONAL,
&Secondary-encoding-object-set    #ENCODINGS OPTIONAL,
&over-ride-encoded-by             BOOLEAN DEFAULT FALSE
```

22.11.1.2 La sintaxis que se ha de utilizar para la especificación de codificación de tipo contenido será:

```
[CONTENTS-ENCODING &Primary-encoding-object-set
                  [COMPLETED BY &Secondary-encoding-object-set]
                  [OVERRIDE &over-ride-encoded-by]]
```

22.11.1.3 El objetivo de esta especificación es determinar la especificación de un tipo contenido, y si será invalidada o no una constricción de contenido "**ENCODED BY**" ASN.1 asociada a ese tipo contenido.

22.11.1.4 Esta especificación proporciona uno o dos conjuntos de objetos de codificación. Si se proporcionan dos, se combinan de acuerdo con 13.2 para producir un conjunto de objetos de codificación combinados.

22.11.1.5 Se considera que esta especificación está fijada si se utiliza la palabra clave "**CONTENTS-ENCODING**".

22.11.2 Acciones de codificador

22.11.2.1 Si "**CONTENTS-ENCODING**" no está fijado, se codificará un tipo contenido con el conjunto de objetos de codificación combinados aplicado al contenedor en el caso de que "**ENCODED BY**" no esté presente en la constricción de contenido ASN.1, o de otro modo, con las reglas de codificación especificadas por la declaración "**ENCODED BY**".

22.11.2.2 Si "**CONTENTS-ENCODING**" está fijado, se aplicará el conjunto de objetos de codificación combinados formado a partir de "**COMPLETED BY**" al tipo contenido en el caso de que "**ENCODED BY**" no esté presente en la constricción de contenido ASN.1, o bien si "**ENCODED BY**" está presente y "**OVERRIDE**" es "**TRUE**". De otro modo, se aplicará al tipo contenido el conjunto de objetos de codificación combinados aplicado al tipo conteniendo.

22.11.3 Acciones de decodificador

22.11.3.1 Un decodificador decodificará el tipo contenido de acuerdo con la codificación aplicada por el codificador, como se especificó anteriormente.

22.12 Especificación de inversión de bits

22.12.1 Propiedades de codificación, sintaxis y objetivo

22.12.1.1 La especificación de inversión de bits utiliza la propiedad de codificación siguiente:

```
&bit-reversal                     ReversalSpecification
                                  DEFAULT no-reversal
```

22.12.1.2 La sintaxis que se ha de utilizar para la especificación de inversión de bits será:

```
[BIT-REVERSAL &bit-reversal]
```

22.12.1.3 La definición de tipos utilizada en este grupo es:

```
ReversalSpecification ::= ENUMERATED
      {no-reversal,
      reverse-bits-in-units,
      reverse-half-units,
      reverse-bits-in-half-units} -- (véase 21.13)
```

22.12.1.4 El objetivo de esta especificación es hacer posible que el orden de los bits en la codificación final sea diferente del de los bits generados como parte de un espacio de codificación o espacio de repetición, o en la codificación completa de una PDU (véase la cláusula 25).

NOTA 1 – La inversión de bits se puede especificar para codificaciones de campos de bits individuales y también para los resultados de la concatenación o la repetición. Se ha de tener la precaución de evitar que una inversión anule a la otra.

NOTA 2 – La inversión de bits se aplica al contenido de un espacio de codificación o de un espacio de repetición (incluyendo cualquier relleno previo o relleno posterior al valor), pero no se aplica a ningún relleno de prealineación.

22.12.2 Constricciones de la especificación

22.12.2.1 Esta especificación sólo está disponible cuando se requiere la codificación de un espacio de codificación o de un espacio de repetición, y dentro de **`#OUTER`**.

22.12.2.2 "**`BIT-REVERSAL`**" no será "**`reverse-half-units`**" o "**`reverse-bits-in-half-units`**" a menos que "**`MULTIPLE OF`**" esté fijado en un número par de bits para el espacio de codificación o el espacio de repetición o la inversión de **`#OUTER`**. (Lo anterior significa que en este caso no se permite un valor de "repetitions" para "**`MULTIPLE OF`**").

22.12.2.3 "**`BIT-REVERSAL`**" no se fijará a menos que "**`MULTIPLE OF`**" sea **`repeticiones`** o superior a un bit.

22.12.2.4 Se considera que esta especificación está fijada si se utiliza la palabra clave "**`BIT-REVERSAL`**". Si no está fijada, los codificadores y decodificadores actúan como si estuviera fijada tomando cada propiedad de codificación su valor por defecto.

22.12.3 Acciones de codificador

22.12.3.1 Excepto cuando efectúe acciones de **`#OUTER`**, un codificador dividirá el contenido del espacio de codificación o el espacio de repetición en "**`MULTIPLE OF`**" unidades, a menos que "**`MULTIPLE OF`**" sea "**`repetitions`**". Si "**`MULTIPLE OF`**" es "**`repetitions`**", se tratará todo el espacio de codificación como una sola unidad. Cuando se efectúe la inversión de bits de **`#OUTER`**, se dividirá toda la codificación (después de haber aplicado cualquier "**`PADDING`**") en "**`MULTIPLE OF`**" unidades. Si la codificación completa no es un múltiplo entero de "**`MULTIPLE OF`**" unidades, se trata de un error de la especificación ECN.

22.12.3.2 El codificador no invertirá el valor por defecto, o invertirá los bits de cada unidad, o invertirá las semiunidades (sin cambiar el orden de los bits de cada semiunidad), o invertirá los bits dentro de cada semiunidad, según lo especificado por el valor de "**`BIT-REVERSAL`**".

22.12.4 Acciones de decodificador

22.12.4.1 El decodificador comenzará por determinar (véanse las especificaciones de espacio de codificación y espacio de repetición) el final del espacio de codificación y el espacio de repetición o (en el caso de especificación de inversión de bits dentro de **`#OUTER`**) el final de toda la codificación, y efectuará entonces las acciones de inversión especificadas para el codificador antes de continuar con la decodificación.

NOTA – Realizando las mismas inversiones se recuperará el orden de los bits original

23 Especificación de sintaxis definida de clases de campo de bits y constructor

Esta cláusula contiene la sintaxis completa para la definición de objetos de codificación de cada clase de codificación en las diferentes categorías.

NOTA – Las acciones de codificador y decodificador se especifican en las subcláusulas que siguen como condicionadas a un grupo de propiedades de codificación que se fija. Un grupo se fija si, y solamente si, la palabra clave inicial del grupo está presente en la especificación del objeto de codificación.

23.1 Definición de objetos de codificación de clases en la categoría alternativas

23.1.1 Sintaxis definida

La sintaxis de la definición de objetos de codificación de clases en la categoría alternativas se define como sigue:

```
#ALTERNATIVES ::= ENCODING-CLASS {

    -- Structure or component replacement specification (véase 22.1)
&#Replacement-structure
      OPTIONAL,
&replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,
&#Head-end-structure                                                OPTIONAL,
```

```
        -- Pre-alignment and padding specification (véase 22.2)
        &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &encoding-space-pre-padding        Padding DEFAULT zero,
        &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                           DEFAULT bits:'0'B,

        -- Start pointer specification (véase 22.3)
        &start-pointer                     REFERENCE    OPTIONAL,
        &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

        -- Alternative determination (véase 22.6)
        &alternative-determination         AlternativeDetermination
                                           DEFAULT field-to-be-set,
        &alternative-reference             REFERENCE OPTIONAL,
        &Encoder-transforms                #TRANSFORM ORDERED OPTIONAL,
        &Decoder-transforms                #TRANSFORM ORDERED OPTIONAL,
        &handle-id                         PrintableString
                                           DEFAULT "default-handle",
        &alternative-ordering              ENUMERATED {textual, tag}
                                           DEFAULT textual,

        -- Identification handle specification (véase 22.9)
        &exhibited-handle                  PrintableString OPTIONAL,
        &Handle-positions                  INTEGER (0..MAX) OPTIONAL,
        &handle-value                      HandleValue DEFAULT tag:any

    } WITH SYNTAX {
        [REPLACE
            [STRUCTURE]
            [COMPONENT]
            [ALL COMPONENTS]
            WITH &Replacement-structure
            [ENCODED BY &replacement-structure-encoding-object
                [INSERT AT HEAD &#Head-end-structure]]]
        [ALIGNED TO
            [NEXT]
            [ANY]
                &encoding-space-pre-alignment-unit
                [PADDING &encoding-space-pre-padding
                    [PATTERN &encoding-space-pre-pattern]]]
        [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
        ALTERNATIVE
            [DETERMINED BY &alternative-determination
                [HANDLE &handle-id]]
            [USING &alternative-reference
                [ORDER &alternative-ordering]
                [ENCODER-TRANSFORMS &Encoder-transforms]
                [DECODER-TRANSFORMS &Decoder-transforms]]
        [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]

    }
```

23.1.2 Objetivo y restricciones

23.1.2.1 Esta sintaxis se utiliza para definir el comienzo del espacio de codificación de una clase de codificación en la categoría alternativas, la determinación de la alternativa que ha sido codificada y una declaración opcional de que todas las codificaciones exhiben un asa de identificación especificada (con valores de asa de identificación distintos).

23.1.2.2 Si "**REPLACE STRUCTURE**" está fijado, no se fijarán otros grupos de propiedades de codificación.

23.1.2.3 Las codificaciones de esta clase no exhiben un asa de identificación a menos que "**EXHIBITS HANDLE**" esté fijado (incluso si todos los componentes exhiben un asa de identificación, puede ser o no ser la misma).

23.1.2.4 Si "**EXHIBITS HANDLE**" está fijado, es preciso que las codificaciones de todas las alternativas de esta clase exhiban el asa de identificación definida y que tengan valores distintos para ese asa de identificación.

NOTA – Esto requeriría normalmente que cada componente tuviera un "**EXHIBITS HANDLE**" fijado en el mismo valor, a menos que una inserción en el extremo de cabeza exhibiera el asa de identificación (véase 9.10.3).

23.1.3 Acciones de codificador

23.1.3.1 Para cualquier grupo de propiedades de codificación que se fije, el codificador efectuará las acciones de codificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Sustitución.

b) Prealineación y relleno.

c) Puntero de comienzo.

d) Determinación de alternativa.

e) Asa de identificación.

23.1.4 Acciones de decodificador

23.1.4.1 Para cualquier grupo de propiedades de codificación que se fije, el decodificador efectuará las acciones de decodificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Prealineación y relleno.

b) Puntero de comienzo.

c) Determinación de alternativa.

23.2 Definición de objetos de codificación de clases en la categoría cadena de bits

23.2.1 Sintaxis definida

La sintaxis de la definición de objetos de codificación de clases en la categoría cadena de bits se define como sigue:

```
#BITS ::= ENCODING-CLASS {

      -- Pre-alignment and padding specification (véase 22.2)
      &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &encoding-space-pre-padding        Padding DEFAULT zero,
      &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                         DEFAULT bits:'0'B,

      -- Start pointer specification (véase 22.3)
      &start-pointer                     REFERENCE    OPTIONAL,
      &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

      -- Bits value encoding
      &value-reversal                    BOOLEAN DEFAULT FALSE,
      &Transforms                        #TRANSFORM ORDERED OPTIONAL,
      &Bits-repetition-encodings         #CONDITIONAL-REPETITION ORDERED OPTIONAL,
      &bits-repetition-encoding          #CONDITIONAL-REPETITION OPTIONAL,

      -- Identification handle specification (véase 22.9)
      &exhibited-handle                  PrintableString OPTIONAL,
      &Handle-positions                  INTEGER (0..MAX) OPTIONAL,
      &handle-value                      HandleValue DEFAULT tag:any,

      -- Contained type encoding specification (véase 22.11)
      &Primary-encoding-object-set       #ENCODINGS OPTIONAL,
      &Secondary-encoding-object-set     #ENCODINGS OPTIONAL,
      &over-ride-encoded-by              BOOLEAN DEFAULT FALSE


} WITH SYNTAX {

      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
```

```
        [VALUE-REVERSAL             &value-reversal]
        [TRANSFORMS                 &Transforms]
        [REPETITION-ENCODINGS       &Bits-repetition-encodings]
        [REPETITION-ENCODING        &bits-repetition-encoding]
        [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
             [AS &handle-value]]
        [CONTENTS-ENCODING &Primary-encoding-object-set
                       [COMPLETED BY &Secondary-encoding-object-set]
                       [OVERRIDE &over-ride-encoded-by]]
}
```

23.2.2 Modelo para la codificación de clases en la categoría cadena de bits

23.2.2.1 El modelo de codificación de una cadena de bits es como sigue:

a) El orden de los bits en la cadena de bits puede ser invertido.

b) Se consideran entonces que los bits son la repetición de un bit.

c) Hay una transformada opcional (especificada por "**TRANSFORMS**") en la que cada bit se transforma en una cadena de bits autodelimitante.

d) "**REPETITION-ENCODING**" o "**REPETITION-ENCODINGS**" especifican cómo se ha de codificar la repetición de las secuencias de bits (o los bits originales, si "**TRANSFORMS**" no está fijado).

NOTA – El único objetivo de prever "**REPETITION-ENCODING**" así como "**REPETITION-ENCODINGS**" es proporcionar una sintaxis que no contenga dobles llaves ("**{{**") en el caso corriente de una codificación condicional única. Se desaconseja, aunque se permite, la utilización de "**REPETITION-ENCODINGS**" cuando hay una codificación condicional única.

23.2.2.2 Los límites (si están presentes) impuestos a la clase que se codifica (una clase en la categoría cadena de bits) son límites al número de bits de la cadena de bits que forma cada valor abstracto.

23.2.2.3 Cuando se considere que son la repetición de un bit, estos límites se interpretarán como límites impuestos al número de repeticiones y se podrán utilizar en la especificación de los objetos de codificación de la clase **#CONDITIONAL-REPETITION** que se utilizan en la especificación de este objeto de codificación.

23.2.3 Objetivo y restricciones

23.2.3.1 Esta sintaxis se utiliza para definir el comienzo del espacio de codificación de una clase en la categoría cadena de bits, la codificación de los valores abstractos de esa clase, una declaración opcional de que todas las codificaciones de bits exhiben un asa de identificación especificada y una especificación de cómo se codifica un tipo contenido.

23.2.3.2 La **#CONDITIONAL-REPETITION** que es aplicada por este objeto de codificación no especificará "**REPLACE**" a menos que sea "**REPLACE STRUCTURE**".

23.2.3.3 Si cualquiera de los objetos de codificación **#CONDITIONAL-REPETITION** contiene una cláusula "**REPLACE STRUCTURE**", todos los objetos de codificación **#CONDITIONAL-REPETITION** contendrán una cláusula "**REPLACE STRUCTURE**".

23.2.3.4 Si hay una cláusula "**REPLACE STRUCTURE**" en los objetos de codificación **#CONDITIONAL-REPETITION**, no se fijarán otros parámetros.

23.2.3.5 La primera transformada de las "**TRANSFORMS**" (si las hay) tendrá un origen que será un bit único y la última transformada tendrá un resultado que será una cadena de bits. Las cadenas de bits producidas como resultado de un origen de un bit y de cero bits formarán un conjunto autodelimitante (véase 3.2.41).

NOTA – Esto significa que la transformada final ha de ser autodelimitante.

23.2.3.6 Si cualquier transformada de las "**TRANSFORMS**" no es reversible para el valor abstracto al que se aplica, se trata de un error de la especificación ECN o de la aplicación.

23.2.3.7 De las especificaciones "**REPETITION-ENCODING**" y "**REPETITION-ENCODINGS**" se fijará exactamente una.

23.2.3.8 Si un objeto de codificación de la lista ordenada "**REPETITION-ENCODINGS**" se define utilizando "**IF**", todos los objetos de codificación precedentes de esa lista se definirán utilizando "**IF**".

23.2.3.9 Si "**DETERMINED BY**" es "**not-needed**" en una o más de las especificaciones "**REPETITION-ENCODING(S)**", los valores abstractos de la cadena de bits original a la que se aplica ese objeto de codificación estarán limitados a un conjunto autodelimitante finito que puede ser identificado a partir de la especificación ECN.

NOTA – Éste sería el caso si los valores cadena de bits resultaran de una codificación estilo Huffman (véase el anexo E) especificada estableciendo la correspondencia entre valores enteros y bits (véase 19.7), o los valores cadena de bits tuvieran un límite visible a los efectos de la ECN que los limitara a un número fijo de bits.

23.2.3.10 Si "**EXHIBITS HANDLE**" está fijado, todas las codificaciones de valores asociados a esta clase exhibirán el asa de identificación especificada.

NOTA – Esto requerirá por lo general restricciones de los valores abstractos del tipo asociado o la adición de bits redundantes en la transformación en bits, o ambas cosas.

23.2.3.11 Si "**EXHIBITS HANDLE**" está fijado, no se fijará "**ALIGNED TO**" en ninguna de las especificaciones "**REPETITION ENCODING(S)**".

23.2.4 Acciones de codificador

23.2.4.1 Para cualquier grupo de propiedades de codificación que se fije, el codificador efectuará las acciones de codificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Prealineación y relleno.

b) Puntero de comienzo.

c) Codificación del valor de los bits (véase 23.2.4.2).

d) Asa de identificación.

e) Codificación del tipo contenido.

23.2.4.2 Para la codificación del valor de los bits, el codificador hará lo siguiente:

a) Invertirá el orden de los bits en el valor abstracto cadena de bits total si "**VALUE REVERSAL**" está fijado en "**TRUE**";

b) Tratará el valor cadena de bits como la repetición de un bit;

c) Aplicará las "**TRANSFORMS**" (si las hay) a cada bit para producir una repetición de bits;

d) Codificará la repetición aplicando la primera "**REPETITION-ENCODING(S)**" cuya condición se cumple.

23.2.4.3 Si no hay "**REPETITION-ENCODING(S)**" cuya condición se cumpla, se trata de un error de la especificación ECN.

23.2.5 Acciones de decodificador

23.2.5.1 Para cualquier grupo de propiedades de codificación que se fije, el decodificador efectuará las acciones de decodificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Prealineación y relleno.

b) Puntero de comienzo.

c) Decodificación del valor de los bits (véase 23.2.5.2).

d) Decodificación del tipo contenido.

23.2.5.2 Para la decodificación del valor de los bits, el decodificador utilizará la "**REPETITION-ENCODING(S)**" a fin de determinar el espacio de repetición y recuperar el orden de bits original utilizando la especificación "**BIT-REVERSAL**".

23.2.5.3 Si "**TRANSFORMS**" está fijado, el decodificador utilizará la propiedad autodelimitante de la codificación de cada bit para determinar el final de cada repetición, e invertirá las transformadas para recuperar el valor cadena de bits original.

23.2.5.4 Si "**VALUE-RESERVAL**" está fijado en "**TRUE**", se invertirá el orden final de los bits del valor abstracto cadena de bits.

23.3 Definición de objetos de codificación de clases en la categoría booleano

23.3.1 Sintaxis definida

La sintaxis de la definición de objetos de codificación de clases en la categoría booleano se define como sigue:

```
#BOOL ::= ENCODING-CLASS {

      -- Structure-only replacement specification (véase 22.1)
      &#Replacement-structure
            OPTIONAL,
      &replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,

      -- Pre-alignment and padding specification (véase 22.2)
      &encoding-space-pre-alignment-unitUnit (ALL EXCEPT repetitions) DEFAULT bit,
      &encoding-space-pre-padding       Padding DEFAULT zero,
      &encoding-space-pre-pattern       Non-Null-Pattern (ALL EXCEPT different:any)
                                        DEFAULT bits:'0'B,

      -- Start pointer specification (véase 22.3)
      &start-pointer                    REFERENCE     OPTIONAL,
      &start-pointer-unit               Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL,

      -- Encoding space specification (véase 22.4)
      &encoding-space-size              EncodingSpaceSize
                                        DEFAULT self-delimiting-values,
      &encoding-space-unit              Unit (ALL EXCEPT repetitions)
                                        DEFAULT bit,
      &encoding-space-determination     EncodingSpaceDetermination
                                        DEFAULT field-to-be-set,
      &encoding-space-reference         REFERENCE OPTIONAL,
      &Encoder-transforms               #TRANSFORM ORDERED OPTIONAL,
      &Decoder-transforms               #TRANSFORM ORDERED OPTIONAL,

      -- Boolean value encoding
      &value-true-pattern               Pattern DEFAULT bits:'1'B,
      &value-false-pattern              Pattern DEFAULT bits:'0'B,

      -- Value padding and justification (véase 22.8)
      &value-justification              Justification DEFAULT right:0,
      &value-pre-padding                Padding DEFAULT zero,
      &value-pre-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
      &value-post-padding               Padding DEFAULT zero,
      &value-post-pattern               Non-Null-Pattern DEFAULT bits:'0'B,
      &unused-bits-determination        UnusedBitsDetermination
                                        DEFAULT field-to-be-set,
      &unused-bits-reference            REFERENCE OPTIONAL,
      &Unused-bits-encoder-transforms   #TRANSFORM ORDERED OPTIONAL,
      &Unused-bits-decoder-transforms   #TRANSFORM ORDERED OPTIONAL,

      -- Identification handle specification (véase 22.9)
      &exhibited-handle                 PrintableString OPTIONAL,
      &Handle-positions                 INTEGER (0..MAX) OPTIONAL,
      &handle-value                     HandleValue DEFAULT tag:any,

      -- Bit reversal specification (véase 22.12)
      &bit-reversal                     ReversalSpecification
                                        DEFAULT no-reversal

} WITH SYNTAX {
      [REPLACE
            [STRUCTURE]
            WITH &#Replacement-structure
                  [ENCODED BY &replacement-structure-encoding-object]]
      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
```

```
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
      ENCODING-SPACE
            [SIZE &encoding-space-size
                  [MULTIPLE OF &encoding-space-unit]]
            [DETERMINED BY &encoding-space-determination]
            [USING &encoding-space-reference
                  [ENCODER-TRANSFORMS &Encoder-transforms]
                  [DECODER-TRANSFORMS &Decoder-transforms]]
      [TRUE-PATTERN &value-true-pattern]
      [FALSE-PATTERN &value-false-pattern]
      [VALUE-PADDING
            [JUSTIFIED &value-justification]
            [PRE-PADDING &value-pre-padding
                  [PATTERN &value-pre-pattern]]
            [POST-PADDING &value-post-padding
                  [PATTERN &value-post-pattern]]
            [UNUSED BITS
                  [DETERMINED BY &unused-bits-determination]
                  [USING &unused-bits-reference
                        [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                        [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
      [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]
      [BIT-REVERSAL &bit-reversal]
}
```

23.3.2 Objetivo y restricciones

23.3.2.1 Esta sintaxis se utiliza para definir el comienzo del espacio de codificación de una clase en la categoría booleano, la codificación de los valores abstractos de esa clase, su posicionamiento dentro del espacio de codificación, una declaración opcional de que todas las codificaciones de bits exhiben un asa de identificación especificada y la posible inversión de los bits del espacio de codificación de la variable booleana.

23.3.2.2 Si "**REPLACE**" está fijado, no se fijarán otros grupos de propiedades de codificación.

23.3.2.3 De los esquemas "**TRUE-PATTERN**" y "**FALSE-PATTERN**" se fijará al menos uno en "**different:any**".

23.3.2.4 Si se selecciona la alternativa "**any-of-length**" para cualquier esquema (o para ambos), la longitud en bits de los dos esquemas será diferente.

23.3.2.5 Si "**ENCODING-SPACE SIZE**" es "**self-delimiting**", "**TRUE-PATTERN**" y "**FALSE-PATTERN**" formarán un conjunto autodelimitante (véase 3.2.41).

23.3.2.6 "**UNUSED BITS DETERMINED BY**" no será "**not-needed**" a menos que:

a) Ambos esquemas sean múltiplos enteros de "**ENCODING-SPACE MULTIPLE OF**" unidades y "**ENCODING-SPACE SIZE**" sea "**variable-with-determinant**"; o

b) Ambos esquemas tengan la misma longitud; o

c) "**JUSTIFIED**" sea "**left**" y los parámetros formen un conjunto autodelimitante; o

d) "**JUSTIFIED**" sea "**right**" y las inversas de los esquemas formen un conjunto autodelimitante (véase 3.2.41).

23.3.2.7 Si hay algunos bits no utilizados en el espacio de codificación se fijará "**VALUE-PADDING**".

23.3.3 Acciones de codificador

23.3.3.1 Para cualquier grupo de propiedades de codificación que se fije, el codificador efectuará las acciones de codificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Sustitución.

b) Prealineación y relleno.

c) Puntero de comienzo.

d) Espacio de codificación (véase 23.3.3.2).

e) Codificación de valor (véase 23.3.3.3).

f) Relleno y justificación de valor.

g) Asa de identificación.

h) Inversión de bits.

23.3.3.2 Si "**ENCODING-SPACE SIZE**" no está fijado en un valor positivo, el tamaño del espacio de codificación "s" es el número más pequeño de "**MULTIPLE OF**" unidades (a reserva de 23.3.3.3) que puede acomodar el esquema del valor que se ha de codificar.

23.3.3.3 El codificador (como una opción de codificador) puede incrementar el tamaño del espacio de codificación "s" (determinado en 23.3.3.2) en "**MULTIPLE OF**" unidades (a reserva de cualesquiera restricciones que imponga la gama de valores de "**field-to-be-set**" o "**field-to-be-used**") si el "**ENCODING-SPACE SIZE**" se fija en "**encoder-option-with-determinant**".

23.3.3.4 El número de bits no utilizados se puede determinar a partir del valor "s" y del esquema del valor que se ha de codificar.

23.3.3.5 Si el número de bits no utilizados es distinto de cero, se aplicará "**VALUE-PADDING**".

23.3.4 Acciones de decodificador

23.3.4.1 Para cualquier grupo de propiedades de codificación que se fije, el decodificador efectuará las acciones de decodificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Prealineación y relleno.

b) Puntero de comienzo

c) Espacio de codificación.

d) Inversión de bits.

e) Relleno y justificación de valor.

f) Decodificación de valor (véase 23.3.4.2)

23.3.4.2 La decodificación del valor se llevará a cabo identificando el "**TRUE-PATTERN**" o el "**FALSE-PATTERN**" de la manera siguiente:

a) Utilizando una determinación "**UNUSED BIT**", si la hay; o

b) Utilizando la propiedad autodelimitante de los esquemas o sus inversos.

23.4 Definición de objetos de codificación de clases en la categoría cadena de bits

23.4.1 Sintaxis definida

La sintaxis de la definición de objetos de codificación de clases en la categoría cadena de bits se define como sigue:

```
#CHARS ::= ENCODING-CLASS {

    -- Pre-alignment and padding specification (véase 22.2)
    &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding        Padding DEFAULT zero,
    &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                       DEFAULT bits:'0'B,

    -- Start pointer specification (véase 22.3)
    &start-pointer                     REFERENCE    OPTIONAL,
    &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

    -- Chars value encoding
    &value-reversal                    BOOLEAN DEFAULT FALSE,
    &Transforms                        #TRANSFORM ORDERED OPTIONAL,
    &Chars-repetition-encodings        #CONDITIONAL-REPETITION ORDERED OPTIONAL,
    &chars-repetition-encoding         #CONDITIONAL-REPETITION OPTIONAL,

    -- Identification handle specification (véase 22.9)
    &exhibited-handle                  PrintableString OPTIONAL,
    &Handle-positions                  INTEGER (0..MAX) OPTIONAL,
    &handle-value                      HandleValue DEFAULT tag:any
```

```
} WITH SYNTAX {
      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
      [VALUE-REVERSAL          &value-reversal]
      [TRANSFORMS              &Transforms]
      [REPETITION-ENCODINGS    &Chars-repetition-encodings]
      [REPETITION-ENCODING     &chars-repetition-encoding]
      [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]

}
```

23.4.2 Modelo para la codificación de clases en la categoría cadena de caracteres

23.4.2.1 El modelo de codificación de una cadena de caracteres es como sigue:

a) El orden de los caracteres en la cadena de caracteres puede ser invertido.

b) Se considera entonces que los caracteres son la repetición de un carácter.

c) Hay una transformada (especificada por "**TRANSFORMS**") en la que cada carácter se transforma en una cadena de bits autodelimitante.

d) "**REPETITION-ENCODING**" o "**REPETITION-ENCODINGS**" especifican cómo se ha de codificar la repetición de la cadena de bits.

NOTA – El único objetivo de prever "**REPETITION-ENCODING**" así como "**REPETITION-ENCODINGS**" es proporcionar una sintaxis que no contenga dobles llaves ("**{{**") en el caso corriente de una codificación condicional única. Se desaconseja, aunque se permite, la utilización de "**REPETITION-ENCODINGS**" cuando hay una codificación condicional única.

23.4.2.2 Los límites (si están presentes) impuestos a la clase que se codifica (una clase en la categoría cadena de caracteres) son límites al número de caracteres de la cadena de caracteres que forma cada valor abstracto.

23.4.2.3 Cuando se considere que son la repetición de un carácter, estos límites se interpretarán como límites impuestos al número de repeticiones y se podrán utilizar en la especificación de los objetos de codificación de la clase **#REPETITION-ENCODING** que se utilizan en la especificación de este objeto de codificación.

23.4.3 Objetivo y restricciones

23.4.3.1 Esta sintaxis se utiliza para definir el comienzo del espacio de codificación de una clase en la categoría cadena de caracteres, la codificación de los valores abstractos asociados a esa clase y una declaración opcional de que todas las codificaciones de caracteres exhiben un asa de identificación especificada.

23.4.3.2 La **#CONDITIONAL-REPETITION** que es aplicada por este objeto de codificación no especificará "**REPLACE**" a menos que sea "**REPLACE STRUCTURE**".

23.4.3.3 Si cualquiera de los objetos de codificación **#CONDITIONAL-REPETITION** contiene una cláusula "**REPLACE STRUCTURE**", todos los objetos de codificación **#CONDITIONAL-REPETITION** contendrán una cláusula "**REPLACE STRUCTURE**".

23.4.3.4 Si no hay una cláusula "**REPLACE STRUCTURE**" en los objetos de codificación **#CONDITIONAL-REPETITION**, se fijará "**TRANSFORMS**". Si hay una cláusula "**REPLACE STRUCTURE**" en los objetos de codificación **#CONDITIONAL-REPETITION**, no se fijarán otros parámetros.

23.4.3.5 La primera transformada de las "**TRANSFORMS**" tendrá un origen que será un carácter único y la última transformada tendrá un resultado que será una cadena de bits. Las cadenas de bits producidas para el conjunto de todos los caracteres que se han de codificar formarán un conjunto autodelimitante (véase 3.2.41).

NOTA – Esto significa que la transformada final ha de ser autodelimitante.

23.4.3.6 Si cualquier transformada de las "**TRANSFORMS**" no es reversible para el valor abstracto al que se aplica, se trata de un error de la especificación ECN o de la aplicación.

23.4.3.7 De las especificaciones "**REPETITION-ENCODING**" y "**REPETITION-ENCODINGS**" se fijará exactamente una.

23.4.3.8 Si un objeto de codificación de la lista ordenada "**REPETITION-ENCODINGS**" se define utilizando "**IF**", todos los objetos de codificación precedentes de esa lista se definirán utilizando "**IF**".

23.4.3.9 Si "**EXHIBITS HANDLE**" está fijado, todas las codificaciones de valores asociados a esta clase exhibirán el asa de identificación.

NOTA – Esto requerirá por lo general restricciones de los valores abstractos del tipo asociado o la inclusión de bits redundantes en la codificación de cada carácter, o ambas cosas.

23.4.3.10 Si "**EXHIBITS HANDLE**" está fijado, no se fijará "**ALIGNED TO**" en ninguna de las especificaciones "**REPETITION ENCODING(S)**".

23.4.4 Acciones de codificador

23.4.4.1 Para cualquier grupo de propiedades de codificación que se fije, el codificador efectuará las acciones de codificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Prealineación y relleno.

b) Puntero de comienzo.

c) Codificación del valor de los caracteres (véase 23.4.4.3).

d) Codificación de la repetición especificada por la primera "**REPETITION-ENCODING(S)**" cuya condición se cumple.

e) Especificación del asa de identificación.

23.4.4.2 Si no hay ninguna "**REPETITION-ENCODING(S)**" cuya condición se cumpla, se trata de un error de la especificación ECN.

23.4.4.3 Para la codificación del valor cadena de caracteres, el codificador hará lo siguiente:

a) Invertirá el orden de los caracteres en el valor abstracto cadena de caracteres total si "**VALUE REVERSAL**" está fijado en "**TRUE**";

b) Tratará el valor cadena de caracteres como la repetición de un carácter;

c) Aplicará las "**TRANSFORMS**" (si las hay) a cada carácter para producir una repetición de bits ;

d) Codificará la repetición aplicando la "**REPETITION-ENCODING(S)**".

23.4.5 Acciones de decodificador

23.4.5.1 Para cualquier grupo de propiedades de codificación que se fije, el decodificador efectuará las acciones de decodificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Prealineación y relleno.

b) Puntero de comienzo.

c) Decodificación de la repetición especificada por la primera "**REPETITION-ENCODING(S)**" cuya condición se cumple.

d) Decodificación del valor cadena de caracteres (véase 23.4.5.2).

23.4.5.2 Para la decodificación del valor cadena de caracteres, el decodificador utilizará la "**REPETITION-ENCODING(S)**" a fin de determinar el espacio de repetición y recuperar los caracteres originales. Si "**TRANSFORMS**" está fijado, el decodificador utilizará la propiedad autodelimitante (que incluye una posible longitud fija) de la codificación de cada carácter para determinar el final de cada repetición, e invertirá las transformadas para recuperar un valor cadena de caracteres.

23.4.5.3 Si "**VALUE-RESERVAL**" está fijado en "**TRUE**", se invertirá el orden final de los caracteres del valor abstracto cadena de caracteres.

23.5 Definición de objetos de codificación de clases en la categoría concatenación

23.5.1 Sintaxis definida

La sintaxis de la definición de objetos de codificación de clases en la categoría concatenación se define como sigue:

```
#CONCATENATION ::= ENCODING-CLASS {

    -- Full replacement specification (véase 22.1)
    &#Replacement-structure
         OPTIONAL,
    &#Replacement-structure2
         OPTIONAL,
    &replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,
    &replacement-structure-encoding-object2 &#Replacement-structure2   OPTIONAL,
    &#Head-end-structure                                                 OPTIONAL,
    &#Head-end-structure2                                                OPTIONAL,

    -- Pre-alignment and padding specification (véase 22.2)
    &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding        Padding DEFAULT zero,
    &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                       DEFAULT bits:'0'B,

    -- Start pointer specification (véase 22.3)
    &start-pointer                     REFERENCE    OPTIONAL,
    &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

    -- Encoding space specification (véase 22.4)
    &encoding-space-size               EncodingSpaceSize
                                       DEFAULT self-delimiting-values,
    &encoding-space-unit               Unit (ALL EXCEPT repetitions)
                                       DEFAULT bit,
    &encoding-space-determination      EncodingSpaceDetermination
                                       DEFAULT field-to-be-set,
    &encoding-space-reference          REFERENCE OPTIONAL,
    &Encoder-transforms                #TRANSFORM ORDERED OPTIONAL,
    &Decoder-transforms                #TRANSFORM ORDERED OPTIONAL,

    -- Concatenation specification (véase 22.10)
    &concatenation-order               ENUMERATED {textual, tag, random}
                                       DEFAULT textual,
    &concatenation-alignment           ENUMERATED {none, aligned}
                                       DEFAULT aligned,
    &concatenation-handle              PrintableString
                                       DEFAULT "default-handle",

    -- Value padding and justification (véase 22.8)
    &value-justification               Justification DEFAULT right:0,
    &value-pre-padding                 Padding DEFAULT zero,
    &value-pre-pattern                 Non-Null-Pattern DEFAULT bits:'0'B,
    &value-post-padding                Padding DEFAULT zero,
    &value-post-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
    &unused-bits-determination         UnusedBitsDetermination
                                       DEFAULT field-to-be-set,
    &unused-bits-reference             REFERENCE OPTIONAL,
    &Unused-bits-encoder-transforms    #TRANSFORM ORDERED OPTIONAL,
    &Unused-bits-decoder-transforms    #TRANSFORM ORDERED OPTIONAL,

    -- Identification handle specification (véase 22.9)
    &exhibited-handle                  PrintableString OPTIONAL,
    &Handle-positions                  INTEGER (0..MAX) OPTIONAL,
    &handle-value                      HandleValue DEFAULT tag:any,

    -- Bit reversal specification (véase 22.12)
    &bit-reversal                      ReversalSpecification
                                       DEFAULT no-reversal
```

```
} WITH SYNTAX {
      [REPLACE
            [STRUCTURE]
            [COMPONENT]
            [ALL COMPONENTS]
            [OPTIONALS]
            [NON-OPTIONALS]
            WITH &#Replacement-structure
                  [ENCODED BY &replacement-structure-encoding-object
                        [INSERT AT HEAD &#Head-end-structure]]
                  [AND OPTIONALS WITH &#Replacement-structure2
                        [ENCODED BY &replacement-structure-encoding-object2
                              [INSERT AT HEAD &#Head-end-structure2]]] ]
      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
      ENCODING-SPACE
            [SIZE &encoding-space-size
                  [MULTIPLE OF &encoding-space-unit]]
            [DETERMINED BY &encoding-space-determination]
            [USING &encoding-space-reference
                  [ENCODER-TRANSFORMS &Encoder-transforms]
                  [DECODER-TRANSFORMS &Decoder-transforms]]
      [CONCATENATION
            [ORDER &concatenation-order]
            [ALIGNMENT &concatenation-alignment]
            [HANDLE &concatenation-handle]]
      [VALUE-PADDING
            [JUSTIFIED &value-justification]
            [PRE-PADDING &value-pre-padding
                  [PATTERN &value-pre-pattern]]
            [POST-PADDING &value-post-padding
                  [PATTERN &value-post-pattern]]
            [UNUSED BITS
                  [DETERMINED BY &unused-bits-determination]
                  [USING &unused-bits-reference
                        [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                        [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
      [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]
      [BIT-REVERSAL &bit-reversal]

}
```

23.5.2 Objetivo y restricciones

23.5.2.1 Esta sintaxis se utiliza para definir el comienzo del espacio de codificación de una clase en la categoría concatenación, la manera en que se combinan las codificaciones de los componentes, su posicionamiento dentro del espacio de codificación, una declaración opcional de que todas las codificaciones exhiben un asa de identificación especificada y la posible inversión de los bits del espacio de codificación.

23.5.2.2 Si "**`REPLACE STRUCTURE`**" está fijado, no se fijarán otros grupos de parámetros de codificación.

23.5.2.3 "**`ENCODING-SPACE SIZE`**" será "**`variable-with-determinant`**" o "**`self-delimiting-values`**".

23.5.2.4 Si "**`EXHIBTIS HANDLE`**" está fijado, la codificación de todos los valores abstractos posibles asociados a esta clase exhibirá el asa de identificación definida.

NOTA – Esto se conseguirá a menudo asegurando que el primer componente de la concatenación, o una inserción en el extremo de cabeza, exhibe el asa de identificación.

23.5.3 Acciones de codificador

23.5.3.1 Para cualquier grupo de propiedades de codificación que se fije, el codificador efectuará las acciones de codificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Sustitución.

b) Prealineación y relleno.

c) Puntero de comienzo.

d) Espacio de codificación (véase 23.5.3.2).

e) Concatenación.

f) Relleno y justificación de valor.

g) Especificación del asa de identificación.

h) Inversión de bits.

23.5.3.2 Si "**ENCODING SPACE**" es "**variable-with-determinant**", será el número mínimo de "**MULTIPLE OF**" unidades necesario para contener la concatenación.

23.5.4 Acciones de decodificador

23.5.4.1 Para cualquier grupo de propiedades de codificación que se fije, el decodificador efectuará las acciones de decodificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Prealineación y relleno.

b) Puntero de comienzo.

c) Espacio de codificación.

d) Inversión de bits.

e) Relleno y justificación de valor.

f) Concatenación.

23.6 Definición de objetos de codificación de clases en la categoría entero

23.6.1 Sintaxis definida

La sintaxis de la definición de objetos de codificación de clases en la categoría entero se define como sigue:

```
#INT ::= ENCODING-CLASS {

      -- Integer encoding
      &Integer-encodings          #CONDITIONAL-INT ORDERED OPTIONAL,
      &integer-encoding           #CONDITIONAL-INT OPTIONAL

} WITH SYNTAX {
      [ENCODINGS &Integer-encodings]
      [ENCODING &integer-encoding]

}
```

23.6.2 Objetivo y restricciones

23.6.2.1 Esta sintaxis se utiliza para definir la codificación de una clase en la categoría entero especificando una o más codificaciones de la clase **#CONDITIONAL-INT**.

23.6.2.2 De los parámetros ordenados "**ENCODING**" y "**ENCODINGS**" se fijará exactamente uno.

NOTA – El único objetivo de prever "**ENCODING**" así como "**ENCODINGS**" es proporcionar una sintaxis que no contenga dobles llaves ("**{{**") en el caso corriente de un objeto codificación único. Se desaconseja, aunque se permite, la utilización de "**ENCODINGS**" cuando hay un objeto codificación único.

23.6.2.3 Si un objeto de codificación de la lista ordenada "**ENCODINGS**" se define utilizando "**IF**", todos los objetos de codificación precedentes de esa lista se definirán utilizando "**IF**".

23.6.3 Acciones de codificador

23.6.3.1 El codificador seleccionará y aplicará el primer objeto de codificación **#CONDITIONAL-INT** de "**ENCODING(S)**" cuyas condiciones se cumplan. Si ninguna de las codificaciones condicionales tiene condiciones que se cumplen, se trata de un error de la especificación ECN.

NOTA – Sería inusual, aunque no ilegal, el que estuvieran presentes objetos de codificación **#CONDITIONAL-INT** que podrían no ser utilizados nunca porque se cumplirán siempre las condiciones sobre la utilización de objetos de codificación anteriores.

23.6.4 Acciones de decodificador

23.6.4.1 El decodificador seleccionará y utilizará el primer objeto de codificación **#CONDITIONAL-INT** de "**ENCODING(S)**" cuyas condiciones se cumplan.

23.7 Definición de objetos de codificación de la clase #CONDITIONAL-INT

23.7.1 Sintaxis definida

La sintaxis de la definición de objetos de codificación de la clase **#CONDITIONAL-INT** se define como sigue:

```
#CONDITIONAL-INT ::= ENCODING-CLASS {

    -- Condition (véase 21.11)
    &range-condition                        RangeCondition  OPTIONAL,

    -- Structure-only replacement specification (véase 22.1)
    &#Replacement-structure
         OPTIONAL,
    &replacement-structure-encoding-object  &#Replacement-structure   OPTIONAL,

    -- Pre-alignment and padding specification (véase 22.2)
    &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding        Padding DEFAULT zero,
    &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                       DEFAULT bits:'0'B,

    -- Start pointer specification (véase 22.3)
    &start-pointer                     REFERENCE    OPTIONAL,
    &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

    -- Encoding space specification (véase 22.4)
    &encoding-space-size               EncodingSpaceSize
                                       DEFAULT self-delimiting-values,
    &encoding-space-unit               Unit (ALL EXCEPT repetitions)
                                       DEFAULT bit,
    &encoding-space-determination      EncodingSpaceDetermination
                                       DEFAULT field-to-be-set,
    &encoding-space-reference          REFERENCE OPTIONAL,
    &Encoder-transforms                #TRANSFORM ORDERED OPTIONAL,
    &Decoder-transforms                #TRANSFORM ORDERED OPTIONAL,

    -- Value encoding
    &Transform                  #TRANSFORM ORDERED OPTIONAL,
    &encoding                   ENUMERATED
                                {positive-int, twos-complement,
                                reverse-positive-int, reverse-twos-complement}
                                DEFAULT twos-complement,

    -- Value padding and justification (véase 22.8)
    &value-justification               Justification DEFAULT right:0,
    &value-pre-padding                 Padding DEFAULT zero,
    &value-pre-pattern                 Non-Null-Pattern DEFAULT bits:'0'B,
    &value-post-padding                Padding DEFAULT zero,
    &value-post-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
    &unused-bits-determination         UnusedBitsDetermination
                                       DEFAULT field-to-be-set,
    &unused-bits-reference             REFERENCE OPTIONAL,
    &Unused-bits-encoder-transforms    #TRANSFORM ORDERED OPTIONAL,
    &Unused-bits-decoder-transforms    #TRANSFORM ORDERED OPTIONAL,
```

```
        -- Identification handle specification (véase 22.9)
        &exhibited-handle                PrintableString OPTIONAL,
        &Handle-positions                INTEGER (0..MAX) OPTIONAL,
        &handle-value                    HandleValue DEFAULT tag:any,

        -- Bit reversal specification (véase 22.12)
        &bit-reversal                    ReversalSpecification
                                         DEFAULT no-reversal

    } WITH SYNTAX {
        [IF &range-condition] [ELSE]
        [REPLACE
            [STRUCTURE]
            WITH &#Replacement-structure
                [ENCODED BY &replacement-structure-encoding-object]]
        [ALIGNED TO
            [NEXT]
            [ANY]
                &encoding-space-pre-alignment-unit
                [PADDING &encoding-space-pre-padding
                    [PATTERN &encoding-space-pre-pattern]]]
        [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
        ENCODING-SPACE
            [SIZE &encoding-space-size
                [MULTIPLE OF &encoding-space-unit]]
            [DETERMINED BY &encoding-space-determination]
            [USING &encoding-space-reference
                [ENCODER-TRANSFORMS &Encoder-transforms]
                [DECODER-TRANSFORMS &Decoder-transforms]]
        [TRANSFORMS     &Transforms]
        [ENCODING       &encoding]
        [VALUE-PADDING
            [JUSTIFIED &value-justification]
            [PRE-PADDING &value-pre-padding
                [PATTERN &value-pre-pattern]]
            [POST-PADDING &value-post-padding
                [PATTERN &value-post-pattern]]
            [UNUSED BITS
                [DETERMINED BY &unused-bits-determination]
                [USING &unused-bits-reference
                    [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                    [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
        [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]
        [BIT-REVERSAL &bit-reversal]

    }
```

23.7.2 Objetivo y restricciones

23.7.2.1 Esta sintaxis se utiliza para definir un objeto de codificación **#CONDITIONAL-INT**. La única utilización de ese objeto de codificación está en la especificación de un objeto de codificación de una clase en la categoría entero.

23.7.2.2 La sintaxis permite la especificación de una sola condición impuesta a los límites del entero para que se aplique esta codificación (utilización de "**IF**"). Permite además la especificación de la no existencia de condición alguna. La utilización de "**ELSE**", o la omisión de "**IF**" y "**ELSE**", especifica que no hay ninguna condición.

23.7.2.3 Utilizando esta sintaxis, el especificador de la ECN puede definir el comienzo del espacio de codificación de la codificación de una clase en la categoría entero, la codificación de los valores abstractos asociados a esa clase, su posicionamiento dentro del espacio de codificación y la posible inversión de los bits del espacio de codificación.

23.7.2.4 De las condiciones "**IF**" y "**ELSE**", como máximo estará presente una.

23.7.2.5 Si "**REPLACE**" está fijado, no se fijarán otros grupos de propiedades de codificación.

23.7.2.6 Si cualquier transformada de las "**TRANSFORMS**" no es reversible para el valor abstracto al que se aplica, se trata de un error de la especificación ECN o de la aplicación. La primera transformada de las "**TRANSFORMS**", si las hay, tendrá un origen que será un entero y la última transformada tendrá un resultado que será un entero.

NOTA – La prueba de la condición "**IF**" tiene lugar en los límites del valor original, y no se ve afectada por estas transformadas.

23.7.2.7 La transformada "**INT-TO-INT**" con el valor "**subtract:lower-bound**" se incluirá si, y solamente si, la condición "**IF**" restringe la aplicación de esta codificación a clases en la categoría entero con un límite más bajo, y (si está presente) será la primera transformada de la lista.

23.7.2.8 "**ENCODING-SPACE SIZE**" no se fijará en "**fixed-to-max**" a menos que la condición "**IF**" restrinja la codificación a una clase con un límite más alto y un límite más bajo.

23.7.2.9 "**ENCODING-SPACE SIZE**" no se fijará en "**self-delimiting-values**".

23.7.2.10 Si "**EXHIBITS HANDLE**" está fijado, el especificador confirma que la codificación de todos los valores exhibe el asa de identificación.

NOTA – Esto requerirá normalmente la utilización de "**VALUE-PADDING**" con justificación a la izquierda para que el relleno muestre el asa de identificación.

23.7.3 Acciones de codificador

23.7.3.1 El codificador detectará un error de la especificación ECN o de la aplicación si no se respeta cualquiera de las restricciones de 23.7.2.

23.7.3.2 Para cualquier grupo de propiedades de codificación que se fije, el codificador efectuará las acciones de codificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Sustitución.

b) Prealineación y relleno.

c) Puntero de comienzo.

d) Espacio de codificación.

e) Codificación de valor (véase más adelante).

f) Relleno y justificación de valor.

g) Asa de identificación.

h) Inversión de bits.

23.7.3.3 El codificadora aplicará las "**TRANSFORMS**" (si las hay) al valor que se codifica.

23.7.3.4 El codificador utilizará el siguiente cuadro que da la gama de valores enteros que pueden ser codificados en "n" bits.

"**ENCODING**"	Valor mín.	Valor máx.
"**positive-int**"	0	$2^n - 1$
"**reverse-positive-int**"	0	$2^n - 1$
"**twos-complement**"	-2^{n-1}	$2^{n-1} - 1$
"**reverse-twos-complement**"	-2^{n-1}	$2^{n-1} - 1$

23.7.3.5 El parámetro "**ENCODING**" selecciona como codificación la codificación en complemento de 2 o la codificación de un entero positivo, o bien la inversa de una de éstas. En 8.3.2 y 8.3.3 de la Rec. UIT-T X.690 | ISO/CEI 8825-1 se da la especificación de la codificación en complemento de 2 y la codificación de un entero positivo. La inversa de estas codificaciones es una codificación en la que, tras producir de "n" bits, se invierte el orden de los "n" bits.

23.7.3.6 El codificador detectará un error de la especificación ECN o de la aplicación si se va a codificar un valor en un número de bits insuficiente según lo especificado en 23.7.3.4.

23.7.3.7 Si el "**ENCODING-SPACE SIZE**" es un entero positivo, su tamaño en bits se calcula multiplicando "**SIZE**" por "**MULTIPLE OF**" unidades. Si "**VALUE-PADDING**" no está fijado, este será el número de bits "n" en que se codificará el entero y no habrá bits no utilizados. Si "**VALUE-PADDING**" está fijado, el número de bits en que se codificará el entero se reduce en el valor entero "m" especificado para "**JUSTIFIED**" y habrá "m" bits no utilizados.

23.7.3.8 Si el "**ENCODING-SPACE SIZE**" es "**fixed-to-max**", el codificador determinará el número mínimo de "**MULTIPLE OF**" unidades que basta para codificar cualquiera de los valores de la clase y procederá (según lo especificado anteriormente) como si "**SIZE**" fuese un entero positivo fijado en ese valor.

23.7.3.9 Si el "**ENCODING-SPACE SIZE**" es "**variable-with-determinant**", el codificador determinará el número mínimo de "**MULTIPLE OF**" unidades (digamos, "s") que tienen bits suficientes para codificar el valor abstracto real que se codifica y procederá (según lo especificado anteriormente) como si "**SIZE**" fuese un conjunto de enteros positivos fijado en ese valor.

23.7.3.10 El codificador (como una opción de codificador) puede incrementar "s" (determinado en 23.7.3.9) en "**MULTIPLE OF**" unidades (a reserva de cualesquiera restricciones que imponga la gama de valores de "**field-to-be-set**" o "**field-to-be-used**") si "**ENCODING-SPACE SIZE**" se fija en "**encoder-option-with-determinant**".

23.7.3.11 El codificador procederá entonces (según lo especificado anteriormente) como si "**SIZE**" fuese un entero positivo fijado en "s".

23.7.4 Acciones de decodificador

23.7.4.1 Para cualquier grupo de propiedades de codificación que se fije, el decodificador efectuará las acciones de decodificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Prealineación y relleno.

b) Puntero de comienzo.

c) Espacio de codificación.

d) Inversión de bits.

e) Relleno y justificación de valor.

f) Decodificación de valor (véase 23.7.4.2).

23.7.4.2 El decodificador recuperará el valor entero de los bits utilizados para codificarlo, decodificando de acuerdo con la codificación especificada, e invertirá entonces las "**TRANSFORMS**" (si se especifican) para recuperar el valor abstracto original.

23.8 Definición de objetos de codificación de clases en la categoría nulo

23.8.1 Sintaxis definida

La sintaxis de la definición de objetos de codificación de clases en la categoría nulo se define como sigue:

```
#NUL ::= ENCODING-CLASS {

      -- Structure-only replacement specification (véase 22.1)
      &#Replacement-structure
            OPTIONAL,
      &replacement-structure-encoding-object  &#Replacement-structure   OPTIONAL,

      -- Pre-alignment and padding specification (véase 22.2)
      &encoding-space-pre-alignment-unitUnit (ALL EXCEPT repetitions) DEFAULT bit,
      &encoding-space-pre-padding       Padding DEFAULT zero,
      &encoding-space-pre-pattern       Non-Null-Pattern (ALL EXCEPT different:any)
                                        DEFAULT bits:'0'B,

      -- Start pointer specification (véase 22.3)
      &start-pointer                    REFERENCE    OPTIONAL,
      &start-pointer-unit               Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL,

      -- Encoding space specification (véase 22.4)
      &encoding-space-size              EncodingSpaceSize
                                        DEFAULT self-delimiting-values,
      &encoding-space-unit              Unit (ALL EXCEPT repetitions)
                                        DEFAULT bit,
      &encoding-space-determination     EncodingSpaceDetermination
                                        DEFAULT field-to-be-set,
      &encoding-space-reference         REFERENCE OPTIONAL,
      &Encoder-transforms               #TRANSFORM ORDERED OPTIONAL,
      &Decoder-transforms               #TRANSFORM ORDERED OPTIONAL,

      -- Value pattern
      &value-pattern                    Pattern (ALL EXCEPT different:any)
                                        DEFAULT bits:''B,
```

```
        -- Value padding and justification (véase 22.8)
        &value-justification              Justification DEFAULT right:0,
        &value-pre-padding                Padding DEFAULT zero,
        &value-pre-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
        &value-post-padding               Padding DEFAULT zero,
        &value-post-pattern               Non-Null-Pattern DEFAULT bits:'0'B,
        &unused-bits-determination        UnusedBitsDetermination
                                          DEFAULT field-to-be-set,
        &unused-bits-reference            REFERENCE OPTIONAL,
        &Unused-bits-encoder-transforms   #TRANSFORM ORDERED OPTIONAL,
        &Unused-bits-decoder-transforms   #TRANSFORM ORDERED OPTIONAL,

        -- Identification handle specification (véase 22.9)
        &exhibited-handle                 PrintableString OPTIONAL,
        &Handle-positions                 INTEGER (0..MAX) OPTIONAL,
        &handle-value                     HandleValue DEFAULT tag:any,

        -- Bit reversal specification (véase 22.12)
        &bit-reversal                     ReversalSpecification
                                          DEFAULT no-reversal

    } WITH SYNTAX {
        [REPLACE
            [STRUCTURE]
            WITH &#Replacement-structure
                [ENCODED BY &replacement-structure-encoding-object]]
        [ALIGNED TO
            [NEXT]
            [ANY]
                &encoding-space-pre-alignment-unit
                [PADDING &encoding-space-pre-padding
                    [PATTERN &encoding-space-pre-pattern]]]
        [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
        ENCODING-SPACE
            [SIZE &encoding-space-size
                [MULTIPLE OF &encoding-space-unit]]
            [DETERMINED BY &encoding-space-determination]
            [USING &encoding-space-reference
                [ENCODER-TRANSFORMS &Encoder-transforms]
                [DECODER-TRANSFORMS &Decoder-transforms]]
        [NULL-PATTERN &value-pattern]
        [VALUE-PADDING
            [JUSTIFIED &value-justification]
            [PRE-PADDING &value-pre-padding
                [PATTERN &value-pre-pattern]]
            [POST-PADDING &value-post-padding
                [PATTERN &value-post-pattern]]
            [UNUSED BITS
                [DETERMINED BY &unused-bits-determination]
                [USING &unused-bits-reference
                    [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                    [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
        [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]
        [BIT-REVERSAL &bit-reversal]

    }
```

23.8.2 Objetivo y restricciones

23.8.2.1 Esta sintaxis se utiliza para definir la codificación de una clase en la categoría nulo.

23.8.2.2 Si se fija "**REPLACE STRUCTURE**" no se fijarán otros grupos de propiedades de codificación.

23.8.2.3 Si el "**ENCODING-SPACE SIZE**" es positivo, será suficiente retener el tamaño del "**NULL-PATTERN**" junto con los bits añadidos como resultado de una especificación "**VALUE-PADDING**".

23.8.2.4 Si hay bits no utilizados en el espacio de codificación, se fijará "**VALUE-PADDING**".

23.8.3 Acciones de codificador

23.8.3.1 Para cualquier grupo de propiedades de codificación que se fije, el codificador efectuará las acciones de codificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Sustitución.

b) Prealineación y relleno.

c) Puntero de comienzo.

d) Espacio de codificación.

e) Codificación de valor (véase 23.8.3.2).

f) Relleno y justificación de valor.

g) Asa de identificación.

h) Inversión de bits.

23.8.3.2 La codificación del valor será los bits del "**NULL-PATTERN**".

23.8.3.3 Si el "**ENCODING-SPACE SIZE**" es "**variable-with-determinat**" o "**encoder-option-with-determinant**", será el número mínimo de "**MULTIPLE OF**" unidades necesario para contener el esquema (digamos, "s"), a reserva de 23.8.3.4.

23.8.3.4 El codificador (como una opción de codificador) puede incrementar "s" (determinado en 23.8.3.3) en "**MULTIPLE OF**" unidades (a reserva de cualesquiera restricciones que imponga la gama de valores de "**field-to-be-set**" o "**field-to-be-used**") si "**ENCODING-SPACE SIZE**" se fija en "**encoder-option-with-determinant**".

23.8.3.5 Si hay bits no utilizados en el espacio de codificación, se aplicará "**VALUE-PADDING**".

23.8.4 Acciones de decodificador

23.8.4.1 Para cualquier grupo de propiedades de codificación que se fije, el decodificador efectuará las acciones de decodificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Prealineación y relleno.

b) Puntero de comienzo.

c) Espacio de codificación.

d) Inversión de bits.

e) Relleno y justificación de valor.

23.8.4.2 El decodificador determinará el tamaño del esquema nulo e identificará los bits de la codificación, pero aceptará en silencio cualquier valor para esos bits.

23.9 Definición de objetos de codificación de clases en la categoría cadena de octetos

23.9.1 Sintaxis definida

La sintaxis de la definición de objetos de codificación de clases en la categoría cadena de octetos se define como sigue:

```
#OCTETS ::= ENCODING-CLASS {

    -- Pre-alignment and padding specification (véase 22.2)
    &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding        Padding DEFAULT zero,
    &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                       DEFAULT bits:'0'B,

    -- Start pointer specification (véase 22.3)
    &start-pointer                     REFERENCE    OPTIONAL,
    &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

    -- Octets value encoding
    &value-reversal                    BOOLEAN DEFAULT FALSE,
    &Transforms                        #TRANSFORM ORDERED OPTIONAL,
    &Octets-repetition-encodings       #CONDITIONAL-REPETITION ORDERED OPTIONAL,
    &octets-repetition-encoding        #CONDITIONAL-REPETITION OPTIONAL,
```

```
        -- Identification handle specification (véase 22.9)
        &exhibited-handle                PrintableString OPTIONAL,
        &Handle-positions                INTEGER (0..MAX) OPTIONAL,
        &handle-value                    HandleValue DEFAULT tag:any,

        -- Contained type encoding specification (véase 22.11)

        &Primary-encoding-object-set     #ENCODINGS OPTIONAL,
        &Secondary-encoding-object-set   #ENCODINGS OPTIONAL,
        &over-ride-encoded-by            BOOLEAN DEFAULT FALSE


    } WITH SYNTAX {
        [ALIGNED TO
            [NEXT]
            [ANY]
                &encoding-space-pre-alignment-unit
                [PADDING &encoding-space-pre-padding
                    [PATTERN &encoding-space-pre-pattern]]]
        [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
        [VALUE-REVERSAL          &value-reversal]
        [TRANSFORMS              &Transforms]
        [REPETITION-ENCODINGS    &Octets-repetition-encodings]
        [REPETITION-ENCODING     &octets-repetition-encoding]
        [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]
        [CONTENTS-ENCODING &Primary-encoding-object-set
                    [COMPLETED BY &Secondary-encoding-object-set]
                    [OVERRIDE &over-ride-encoded-by]]

    }
```

23.9.2 Modelo para la codificación de clases en la categoría cadena de octetos

23.9.2.1 El modelo de codificación de una cadena de octetos es como sigue:

a) El orden de los octetos en la cadena de octetos puede ser invertido.

b) Se consideran entonces que los octetos son la repetición de un octeto.

c) Hay una transformada opcional (especificada por "**TRANSFORMS**") en la que cada octeto se transforma en una cadena de bits autodelimitante.

d) "**REPETITION-ENCODING**" o "**REPETITION-ENCODINGS**" especifican cómo se ha de codificar la repetición del octeto.

NOTA – El único objetivo de prever "**REPETITION-ENCODING**" así como "**REPETITION-ENCODINGS**" es proporcionar una sintaxis que no contenga dobles llaves ("**{{**") en el caso corriente de una codificación condicional única. Se desaconseja, aunque se permite, la utilización de "**REPETITION-ENCODINGS**" cuando hay una codificación condicional única.

23.9.2.2 Los límites (si están presentes) impuestos a la clase que se codifica (una clase en la categoría cadena de octetos) son límites al número de octetos de la cadena de octetos que forma cada valor abstracto.

23.9.2.3 Cuando se considere que son la repetición de un octeto, estos límites se interpretarán como límites impuestos al número de repeticiones y se podrán utilizar en la especificación de los objetos de codificación de la clase **#CONDITIONAL-REPETITION** que se utilizan en la especificación de este objeto de codificación.

23.9.3 Objetivo y restricciones

23.9.3.1 Esta sintaxis se utiliza para definir el comienzo del espacio de codificación de una clase en la categoría cadena de octetos, la codificación de los valores abstractos asociados a esa clase, una declaración opcional de que todas las codificaciones de cadenas de octetos exhiben un asa de identificación especificada y una especificación de cómo se codifica un tipo contenido.

23.9.3.2 La **#CONDITIONAL-REPETITION** que es aplicada por este objeto de codificación no especificará "**REPLACE**" a menos que sea "**REPLACE STRUCTURE**".

23.9.3.3 Si cualquiera de los objetos de codificación **#CONDITIONAL-REPETITION** contiene una cláusula "**REPLACE STRUCTURE**", todos los objetos de codificación **#CONDITIONAL-REPETITION** contendrán una cláusula "**REPLACE STRUCTURE**".

23.9.3.4 Si hay una cláusula "**REPLACE STRUCTURE**" en los objetos de codificación **#CONDITIONAL-REPETITION**, no se fijarán otros parámetros.

23.9.3.5 La primera transformada de las "**TRANSFORMS**" (si las hay) tendrá un origen que será una cadena de bits y la última transformada tendrá un resultado que será una cadena de bits autodelimitante (véase 3.2.41).

23.9.3.6 Si cualquier transformada de las "**TRANSFORMS**" no es reversible para el valor abstracto al que se aplica, se trata de un error de la especificación ECN o de la aplicación.

23.9.3.7 De las especificaciones "**REPETITION-ENCODING**" y "**REPETITION-ENCODINGS**" se fijará exactamente una.

23.9.3.8 Si un objeto de codificación de la lista ordenada "**REPETITION-ENCODINGS**" se define utilizando "**IF**", todos los objetos de codificación precedentes de esa lista se definirán utilizando "**IF**".

23.9.3.9 Si "**EXHIBITS HANDLE**" está fijado, todas las codificaciones de valores de esta clase exhibirán el asa de identificación especificada.

NOTA – Esto requerirá por lo general restricciones de los valores abstractos del tipo asociado.

23.9.3.10 Si "**EXHIBITS HANDLE**" está fijado, no se fijará "**ALIGNED TO**" en ninguna de las especificaciones "**REPETITION ENCODING(S)**".

23.9.4 Acciones de codificador

23.9.4.1 Para cualquier grupo de propiedades de codificación que se fije, el codificador efectuará las acciones de codificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Prealineación y relleno.

b) Punto de comienzo.

c) Codificación de valor como se especifica más adelante.

d) Codificación de repetición según lo especificado por la(s) primera(s) "**REPETITIONS-ENCODING(S)**" cuya condición se cumple.

e) Asa de identificación.

f) Codificación de tipo contenido.

23.9.4.2 Para la codificación del valor, el codificador hará lo siguiente:

a) Invertirá el orden de los octetos en el valor abstracto cadena de octetos total si "**VALUE-REVERSAL**" está fijado en "**TRUE**";

b) Tratará el valor cadena de octetos como la repetición de un octeto;

c) Aplicará las "**TRANSFORMS**" (si las hay) a cada octeto para producir una repetición de cadena de bits.

NOTA – Si hay transformadas, cada octeto forma una cadena de bits.

d) Codificará la repetición aplicando la primera "**REPETITION-ENCODING(S)**" cuya condición se cumple.

23.9.4.3 Si no hay ninguna "**REPETITION-ENCODING(S)**" cuya condición se cumpla, se trata de un error de la especificación ECN.

23.9.5 Acciones de decodificador

23.9.5.1 Para cualquier grupo de propiedades de codificación que se fije, el decodificador efectuará las acciones de decodificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Prealineación y relleno.

b) Puntero de comienzo.

c) Decodificación de valor (véase 23.9.5.2).

d) Decodificación de tipo contenido.

23.9.5.2 El decodificador invertirá las "**TRANSFORMS**" (si las hay) para recuperar los octetos originales.

23.9.5.3 Si "**VALUE-REVERSALS**" está fijado en "**TRUE**", se invertirá el orden final de los octetos del valor abstracto cadena de octetos.

23.10 Definición de objetos de codificación de clases en la categoría opcionalidad

23.10.1 Sintaxis definida

La sintaxis de la definición de objetos de codificación de clases en la categoría opcionalidad se define como sigue:

```
#OPTIONAL ::= ENCODING-CLASS {

      -- Structure-only replacement specification (véase 22.1)
      &#Replacement-structure
            OPTIONAL,
      &replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,

      -- Pre-alignment and padding specification (véase 22.2)
      &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &encoding-space-pre-padding        Padding DEFAULT zero,
      &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                         DEFAULT bits:'0'B,

      -- Start pointer specification (véase 22.3)
      &start-pointer                     REFERENCE     OPTIONAL,
      &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
      &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

      -- Optionality determination (véase 22.5)
      &optionality-determination         OptionalityDetermination
                                         DEFAULT field-to-be-set,
      &optionality-reference             REFERENCE OPTIONAL,
      &Encoder-transforms                #TRANSFORM ORDERED OPTIONAL,
      &Decoder-transforms                #TRANSFORM ORDERED OPTIONAL,
      &handle-id                         PrintableString
                                         DEFAULT "default-handle"

} WITH SYNTAX {
      [REPLACE
            [STRUCTURE]
            WITH &#Replacement-structure
                  [ENCODED BY &replacement-structure-encoding-object]]
      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
      PRESENCE
            [DETERMINED BY &optionality-determination
                  [HANDLE &handle-id]]
            [USING &optionality-reference
                  [ENCODER-TRANSFORMS &Encoder-transforms]
                  [DECODER-TRANSFORMS &Decoder-transforms]]

}
```

23.10.2 Objetivo y restricciones

23.10.2.1 Esta sintaxis se utiliza para definir la codificación de una clase en la categoría opcionalidad.

23.10.2.2 Si "**REPLACE STRUCTURE**" está fijado, no se fijarán otros grupos de propiedades de codificación.

23.10.3 Acciones de codificador

23.10.3.1 Para cualquier grupo de propiedades de codificación que se fije, el decodificador efectuará las acciones de decodificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Sustitución (véase 23.10.3.2).

b) Prealineación y relleno.

c) Puntero de comienzo.

d) Determinación de opcionalidad.

23.10.3.2 Si "**REPLACE STRUCTURE**" está fijado, el componente entero (incluyendo cualquier clase en la categoría rótulo, pero excluyendo las clases en la categoría opcionalidad) se proporciona como el parámetro real para la estructura de sustitución, que pasa a ser un componente obligatorio.

23.10.4 Acciones de decodificador

23.10.4.1 Para cualquier grupo de propiedades de codificación que se fije, el decodificador efectuará las acciones de decodificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Prealineación y relleno.

b) Puntero de comienzo.

c) Determinación de opcionalidad.

23.11 Definición de objetos de codificación de clases en la categoría relleno

23.11.1 Sintaxis definida

La sintaxis de la definición de objetos de codificación de clases en la categoría relleno se define como sigue:

```
#PAD ::= ENCODING-CLASS {

    -- Structure-only replacement specification (véase 22.1)
    &#Replacement-structure
          OPTIONAL,
    &replacement-structure-encoding-object  &#Replacement-structure   OPTIONAL,

    -- Pre-alignment and padding specification (véase 22.2)
    &encoding-space-pre-alignment-unitUnit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding       Padding DEFAULT zero,
    &encoding-space-pre-pattern       Non-Null-Pattern (ALL EXCEPT different:any)
                                      DEFAULT bits:'0'B,

    -- Start pointer specification (véase 22.3)
    &start-pointer                    REFERENCE    OPTIONAL,
    &start-pointer-unit               Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL,

    -- Encoding space specification (véase 22.4)
    &encoding-space-size              EncodingSpaceSize
                                      DEFAULT self-delimiting-values,
    &encoding-space-unit              Unit (ALL EXCEPT repetitions)
                                      DEFAULT bit,
    &encoding-space-determination     EncodingSpaceDetermination
                                      DEFAULT field-to-be-set,
    &encoding-space-reference         REFERENCE OPTIONAL,
    &Encoder-transforms               #TRANSFORM ORDERED OPTIONAL,
    &Decoder-transforms               #TRANSFORM ORDERED OPTIONAL,

    -- Value encoding
    &pad-pattern                      Pattern (ALL EXCEPT different:any)
                                      DEFAULT bits:''B,

    -- Identification handle specification (véase 22.9)
    &exhibited-handle                 PrintableString OPTIONAL,
    &Handle-positions                 INTEGER (0..MAX) OPTIONAL,
    &handle-value                     HandleValue DEFAULT tag:any,
```

```
        -- Bit reversal specification (véase 22.12)
        &bit-reversal                    ReversalSpecification
                                         DEFAULT no-reversal

} WITH SYNTAX {
      [REPLACE
            [STRUCTURE]
            WITH &#Replacement-structure
                  [ENCODED BY &replacement-structure-encoding-object]]
      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
      ENCODING-SPACE
            [SIZE &encoding-space-size
                  [MULTIPLE OF &encoding-space-unit]]
            [DETERMINED BY &encoding-space-determination]
            [USING &encoding-space-reference
                  [ENCODER-TRANSFORMS &Encoder-transforms]
                  [DECODER-TRANSFORMS &Decoder-transforms]]
      [PAD-PATTERN &pad-pattern]
      [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]
      [BIT-REVERSAL &bit-reversal]

}
```

23.11.2 Objetivo y restricciones

23.11.2.1 Esta sintaxis se utiliza para definir la codificación de una clase en la categoría relleno.

23.11.2.2 Si el "**ENCODING-SPACE SIZE**" es positivo, "**PAD-PATTERN**" no será de longitud cero, y es reproducido y truncado para llenar el espacio de codificación.

23.11.2.3 Si "**REPLACE STRUCTURE**" está fijado, no se fijarán otros grupos de propiedades de codificación.

23.11.3 Acciones de codificador

23.11.3.1 Para cualquier grupo de propiedades de codificación que se fije, el codificador efectuará las acciones de codificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Sustitución.

b) Prealineación y relleno.

c) Puntero de comienzo.

d) Espacio de codificación.

e) Codificación de valor (véase más adelante).

f) Asa de identificación.

g) Inversión de bits.

23.11.3.2 Si el "**ENCODING-SPACE SIZE**" es positivo, el valor será el "**PAD-PATTERN**", reproducido y truncado para llenar el espacio de codificación.

23.11.3.3 Si el "**ENCODING-SPACE SIZE**" es "**fixed-to-max**" o "**variable-with-determinant**" o "**encoder-option-with-determinant**", el espacio de codificación será el número más pequeño de "**MULTIPLE OF**" unidades que sea mayor que el tamaño de "**PAD–PATTERN**" (digamos, "s"), y el "**PAD-PATTERN**" será reproducido y truncado entonces para llenar ese espacio (véase, no obstante, 23.11.3.4).

NOTA – Dicho espacio será un espacio de codificación vacío si el "**PAD-PATTERN**" es nulo.

23.11.3.4 El codificador (como una opción de codificador) puede incrementar "s" (determinado en 23.11.3.3) en "**MULTIPLE OF**" unidades (a reserva de cualesquiera restricciones que imponga la gama de valores de "**field-to-be-set**" o "**field-to-be-used**") si "**ENCODING-SPACE SIZE**" se fija en "**encoder-option-with-determinant**".

23.11.4 Acciones de decodificador

23.11.4.1 Para cualquier grupo de propiedades de codificación que se fije, el decodificador efectuará las acciones de decodificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Prealineación y relleno.

b) Puntero de comienzo.

c) Inversión de bits.

d) Codificación de espacio.

23.11.4.2 El decodificador determinará el tamaño de la codificación del valor de relleno e identificará los bits de la codificación, pero aceptará en silencio cualquier valor para esos bits.

23.12 Definición de objetos de codificación de clases en la categoría repetición

23.12.1 Sintaxis definida

La sintaxis de la definición de objetos de codificación de clases en la categoría repetición se define como sigue:

```
#REPETITION ::= ENCODING-CLASS {

      -- Repetition encoding
      &Repetition-encodings      #CONDITIONAL-REPETITION ORDERED OPTIONAL,
      &repetition-encoding       #CONDITIONAL-REPETITION OPTIONAL

} WITH SYNTAX {
      [REPETITION-ENCODINGS &Repetition-encodings]
      [REPETITION-ENCODING &repetition-encoding]

}
```

23.12.2 Objetivo y restricciones

23.12.2.1 Esta sintaxis se utiliza para definir la codificación de una clase en la categoría repetición especificando una o más codificaciones de la clase **#CONDITIONAL-REPETITION**.

23.12.2.2 De las especificaciones "**REPETITION-ENCODING**" y "**REPETITION-ENCODINGS**" se fijará exactamente una.

NOTA – El único objetivo de prever "**REPETITION-ENCODING**" así como "**REPETITION-ENCODINGS**" es proporcionar una sintaxis que no contenga dobles llaves ("**{{**") en el caso corriente de un objeto de codificación único. Se desaconseja, aunque se permite, la utilización de "**REPETITION-ENCODINGS**" cuando hay un objeto de codificación único.

23.12.2.3 Si un objeto de codificación de la lista ordenada "**REPETITION-ENCODINGS**" se define utilizando "**IF**", todos los objetos de codificación precedentes de esa lista se definirán utilizando "**IF**".

23.12.3 Acciones de codificador

23.12.3.1 El codificador seleccionará y aplicará el primer objeto de codificación **#CONDITIONAL-REPETITION** de "**ENCODING(S)**" cuyas condiciones se cumplan. Si ninguna de las codificaciones condicionales tiene condiciones que se cumplen, se trata de un error de la especificación ECN.

NOTA – Sería inusual, aunque no ilegal, el que estuvieran presentes objetos de codificación **#CONDITIONAL-REPETITION** que podrían no ser utilizados nunca porque se cumplirían siempre las condiciones sobre la utilización de objetos de codificación anteriores.

23.12.4 Acciones de decodificador

23.12.4.1 El decodificador seleccionará y utilizará el primer objeto de codificación **#CONDITIONAL-REPETITION** de "**ENCODING(S)**" cuyas condiciones se cumplan.

23.13 Definición de objetos de codificación de la clase #CONDITIONAL-REPETITION

23.13.1 Sintaxis definida

La sintaxis de la definición de objetos de codificación de la clase **#CONDITIONAL-REPETITIONS** se define como sigue:

```
#CONDITIONAL-REPETITION ::= ENCODING-CLASS {

    -- Condition (véase 21.12)
    &size-range-condition              SizeRangeCondition  OPTIONAL,

    -- Structure or component replacement specification (véase 22.1)
    &#Replacement-structure
         OPTIONAL,
    &replacement-structure-encoding-object  &#Replacement-structure    OPTIONAL,
    &#Head-end-structure                                                OPTIONAL,

    -- Pre-alignment and padding specification (véase 22.2)
    &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding        Padding DEFAULT zero,
    &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                       DEFAULT bits:'0'B,

    -- Start pointer specification (véase 22.3)
    &start-pointer                     REFERENCE    OPTIONAL,
    &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

    -- Repetition space specification (véase 22.7)
    &repetition-space-size             EncodingSpaceSize
                                       DEFAULT self-delimiting-values,
    &repetition-space-unit             Unit
                                       DEFAULT bit,
    &repetition-space-determination    RepetitionSpaceDetermination
                                       DEFAULT field-to-be-set,
    &main-reference                    REFERENCE OPTIONAL,
    &Encoder-transforms                #TRANSFORM ORDERED OPTIONAL,
    &Decoder-transforms                #TRANSFORM ORDERED OPTIONAL,
    &handle-id                         PrintableString
                                       DEFAULT "default-handle",
    &termination-pattern               Non-Null-Pattern (ALL EXCEPT
                                       different:any) DEFAULT bits '0'B,

    -- Repetition alignment
    &repetition-alignment              ENUMERATED {none, aligned}
                                       DEFAULT none,

    -- Value padding and justification (véase 22.8)
    &value-justification               Justification DEFAULT right:0,
    &value-pre-padding                 Padding DEFAULT zero,
    &value-pre-pattern                 Non-Null-Pattern DEFAULT bits:'0'B,
    &value-post-padding                Padding DEFAULT zero,
    &value-post-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
    &unused-bits-determination         UnusedBitsDetermination
                                       DEFAULT field-to-be-set,
    &unused-bits-reference             REFERENCE OPTIONAL,
    &Unused-bits-encoder-transforms    #TRANSFORM ORDERED OPTIONAL,
    &Unused-bits-decoder-transforms    #TRANSFORM ORDERED OPTIONAL,

    -- Identification handle specification (véase 22.9)
    &exhibited-handle                  PrintableString OPTIONAL,
    &Handle-positions                  INTEGER (0..MAX) OPTIONAL,
    &handle-value                      HandleValue DEFAULT tag:any,

    -- Bit reversal specification (véase 22.12)
    &bit-reversal                      ReversalSpecification
                                       DEFAULT no-reversal
```

```
} WITH SYNTAX {
      [IF &size-range-condition] [ELSE]
      [REPLACE
            [STRUCTURE]
            [COMPONENT]
            [ALL COMPONENTS]
            WITH &Replacement-structure
            [ENCODED BY &replacement-structure-encoding-object
                  [INSERT AT HEAD &#Head-end-structure]]]
      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
      REPETITION-SPACE
            [SIZE &repetition-space-size
                  [MULTIPLE OF &repetition-space-unit]]
            [DETERMINED BY &repetition-space-determination
                  [HANDLE &handle-id]]
            [USING &main-reference
                  [ENCODER-TRANSFORMS &Encoder-transforms]
                  [DECODER-TRANSFORMS &Decoder-transforms]]
            [PATTERN &termination-pattern]
      [ALIGNMENT  &repetition-alignment]
      [VALUE-PADDING
            [JUSTIFIED &value-justification]
            [PRE-PADDING &value-pre-padding
                  [PATTERN &value-pre-pattern]]
            [POST-PADDING &value-post-padding
                  [PATTERN &value-post-pattern]]
            [UNUSED BITS
                  [DETERMINED BY &unused-bits-determination]
                  [USING &unused-bits-reference
                        [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                        [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
      [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]
      [BIT-REVERSAL &bit-reversal]

}
```

23.13.2 Objetivo y restricciones

23.13.2.1 Esta sintaxis se utiliza para definir la codificación de una clase en la categoría repetición a reserva del cumplimiento de una condición basada en los límites de la repetición (utilización de "**IF**"). Permite además la especificación de la no existencia de condición alguna. La utilización de "**ELSE**", o la omisión de "**IF**" y "**ELSE**", especifica que no hay ninguna condición.

23.13.2.2 De las condiciones "**IF**" y "**ELSE**", como máximo estará presente una.

23.13.2.3 Si "**REPLACE STRUCTURE**" está fijado, no se fijarán otros grupos de propiedades de codificación.

23.13.2.4 Si "**EXHIBITS HANDLE**" está fijado, esto significa que todas las codificaciones de esta clase exhiben el asa de identificación especificada (véase también 22.9.2.4).

23.13.2.5 El "**REPETITION-SPACE SIZE**" no será "**fixed-to-max**".

23.13.2.6 Si el "**REPETITION-SPACE SIZE**" es "**self-delimiting-values**" y "**MULTIPLE OF**" es "repetitions", el número de repeticiones estará limitado por los límites impuestos a un valor único.

23.13.2.7 Si hay algunos bits no utilizados en el espacio de codificación, se fijará "**VALUE-PADDING**".

23.13.3 Acciones de codificador

23.13.3.1 Para cualquier grupo de propiedades de codificación que se fije, el codificador efectuará las acciones de codificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Sustitución.

b) Prealineación y relleno.

c) Puntero de comienzo.

d) Espacio de repetición.

e) Codificación de repetición (véase 23.13.3.4.)

f) Relleno y justificación de valor

g) Asa de identificación.

h) Inversión de bits.

23.13.3.2 Si "**ALIGNMENT**" está fijado en "**aligned**", las fijaciones de prealineación y relleno se utilizarán para prealinear cada codificación del componente.

NOTA – Esto se lleva a cabo antes de cualquier prealineación especificada por el componente.

23.13.3.3 Las codificaciones completas de los componentes (con cualquier prealineación especificada) se encadenarán para formar los bits del valor de la repetición.

23.13.3.4 Si el "**REPETITION-SPACE SIZE**" es "**variable-with-determinant**" o "**encoder-option-with-determinant**", el tamaño será el múltiplo más pequeño de "**MULTIPLE OF**" unidades (digamos, "s") que contenga el valor de la repetición (véase, no obstante, 23.13.3.5).

23.13.3.5 El codificador (como una opción de codificador) puede incrementar "s" (determinado en 23.13.3.4) en "**MULTIPLE OF**" unidades (a reserva de cualesquiera restricciones que imponga la gama de valores de "**field-to-be-set**" o "**field-to-be-used**") si el "**ENCODING-SPACE SIZE**" se fija en "**encoder-option-with-determinant**".

23.13.3.6 El valor de repetición se coloca en el espacio de codificación, utilizando "**VALUE-PADDING**" si no hay ningún bit utilizado.

23.13.4 Acciones de decodificador

23.13.4.1 Para cualquier grupo de propiedades de codificación que se fije, el decodificador efectuará las acciones de decodificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Prealineación y relleno.

b) Puntero de comienzo.

c) Espacio de repetición.

d) Inversión de bits.

e) Relleno y justificación de valor

f) Decodificación de repetición (véase 23.13.4.2).

23.13.4.2 Cada repetición será extraída, y decodificada de acuerdo con la especificación de codificación del componente de la clase de repetición.

23.14 Definición de objetos de codificación de clases en la categoría rótulo

23.14.1 Sintaxis definida

La sintaxis de la definición de objetos de codificación de clases en la categoría rótulo se define como sigue:

```
#TAG ::= ENCODING-CLASS {

      -- Structure-only replacement specification (véase 22.1)
      &#Replacement-structure
            OPTIONAL,
      &replacement-structure-encoding-object  &#Replacement-structure   OPTIONAL,
```

```
        -- Pre-alignment and padding specification (véase 22.2)
        &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &encoding-space-pre-padding        Padding DEFAULT zero,
        &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                           DEFAULT bits:'0'B,

        -- Start pointer specification (véase 22.3)
        &start-pointer                     REFERENCE     OPTIONAL,
        &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

        -- Encoding space specification (véase 22.4)
        &encoding-space-size               EncodingSpaceSize
                                           DEFAULT self-delimiting-values,
        &encoding-space-unit               Unit (ALL EXCEPT repetitions)
                                           DEFAULT bit,
        &encoding-space-determination      EncodingSpaceDetermination
                                           DEFAULT field-to-be-set,
        &encoding-space-reference          REFERENCE OPTIONAL,
        &Encoder-transforms                #TRANSFORM ORDERED OPTIONAL,
        &Decoder-transforms                #TRANSFORM ORDERED OPTIONAL,

        -- Value padding and justification (véase 22.8)
        &value-justification               Justification DEFAULT right:0,
        &value-pre-padding                 Padding DEFAULT zero,
        &value-pre-pattern                 Non-Null-Pattern DEFAULT bits:'0'B,
        &value-post-padding                Padding DEFAULT zero,
        &value-post-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
        &unused-bits-determination         UnusedBitsDetermination
                                           DEFAULT field-to-be-set,
        &unused-bits-reference             REFERENCE OPTIONAL,
        &Unused-bits-encoder-transforms    #TRANSFORM ORDERED OPTIONAL,
        &Unused-bits-decoder-transforms    #TRANSFORM ORDERED OPTIONAL,

        -- Identification handle specification (véase 22.9)
        &exhibited-handle                  PrintableString OPTIONAL,
        &Handle-positions                  INTEGER (0..MAX) OPTIONAL,
        &handle-value                      HandleValue DEFAULT tag:any,

        -- Bit reversal specification (véase 22.12)
        &bit-reversal                      ReversalSpecification
                                           DEFAULT no-reversal

    } WITH SYNTAX {
        [REPLACE
            [STRUCTURE]
            WITH &#Replacement-structure
                [ENCODED BY &replacement-structure-encoding-object]]
        [ALIGNED TO
            [NEXT]
            [ANY]
                &encoding-space-pre-alignment-unit
                [PADDING &encoding-space-pre-padding
                    [PATTERN &encoding-space-pre-pattern]]]
        [START-POINTER   &start-pointer
            [MULTIPLE OF     &start-pointer-unit]
            [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
        ENCODING-SPACE
            [SIZE &encoding-space-size
                [MULTIPLE OF &encoding-space-unit]]
            [DETERMINED BY &encoding-space-determination]
            [USING &encoding-space-reference
                [ENCODER-TRANSFORMS &Encoder-transforms]
                [DECODER-TRANSFORMS &Decoder-transforms]]
        [VALUE-PADDING
            [JUSTIFIED &value-justification]
            [PRE-PADDING &value-pre-padding
                [PATTERN &value-pre-pattern]]
            [POST-PADDING &value-post-padding
                [PATTERN &value-post-pattern]]
```

```
            [UNUSED BITS
                  [DETERMINED BY &unused-bits-determination]
                  [USING &unused-bits-reference
                        [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                        [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
      [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]
      [BIT-REVERSAL &bit-reversal]
```

}

23.14.2 Objetivo y restricciones

23.14.2.1 Esta sintaxis se utiliza para definir la codificación de una clase en la categoría rótulo.

23.14.2.2 Si "**REPLACE STRUCTURE**" está fijado, no se fijarán otras especificaciones.

23.14.2.3 El "**ENCODING-SPACE SIZE**" no será "**fixed-to-max**"ni "**self-delimiting-values**".

23.14.3 Acciones de codificador

23.14.3.1 Para cualquier grupo de propiedades de codificación que se fije, el codificador efectuará las acciones de codificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Sustitución.

b) Prealineación y relleno.

c) Puntero de comienzo.

d) Espacio de codificación.

e) Codificación de valor (véase 23.14.3.3).

f) Relleno y justificación de valor.

g) Asa de identificación.

h) Inversión de bits.

23.14.3.2 El codificador determinará el número mínimo de bits "n" necesario para codificar el número de rótulo como el valor más pequeño de "n" que haga que $2^n–1$ sea mayor o igual que el número de rótulo. Si "n" es cero, se aumentará a 1.

23.14.3.3 La codificación será una codificación de entero positivo. La especificación de una codificación de entero positivo figura en 8.3.2 y 8.3.3 de la Rec. UIT-T X.690 | ISO/CEI 8825-1.

23.14.3.4 El codificador detectará un error de la especificación ECN si se va a codificar un número de rótulo en un número de bits insuficiente, según lo especificado anteriormente.

23.14.3.5 Si "**ENCODING-SPACE SIZE**" es un entero positivo, su tamaño en bits se calcula multiplicando "**SIZE**" por "**MULTIPLE OF**" unidades. Si "**VALUE-PADDING**" no está fijado, este será el número de bits "n" en que se codificará el número de rótulo y no habrá bits no utilizados. Si "**VALUE-PADDING**" está fijado, el número de bits en que se codificará el número de rótulo se reduce en el valor entero "m"especificado para "**JUSTIFIED**", y habrá "m" bits no utilizados.

23.14.3.6 Si "**ENCODING-SPACE SIZE**" es "**variable-with-determinant**" o "**encoder-option-with-determinant**", el codificador determinará el número mínimo de "**MULTIPLE OF**" unidades que basta para codificar el número de rótulo (digamos, "s") y procederá (según lo especificado anteriormente) como si "**SIZE**" fuese un entero positivo fijado en ese valor (no obstante véase 23.14.3.7).

23.14.3.7 El codificador (como una opción de codificador) puede incrementar "s" (determinado en 23.14.3.6) en "**MULTIPLE OF**" unidades (a reserva de cualesquiera restricciones que imponga la gama de valores de "**field-to-be-set**" o "**field-to-be-used**") si "**ENCODING-SPACE SIZE**" se fija en "**encoder-option-with-determinant**".

23.14.4 Acciones de decodificador

23.14.4.1 Para cualquier grupo de propiedades de codificación que se fije, el decodificador efectuará las acciones de decodificador especificadas en la cláusula 22, en el orden siguiente y de acuerdo con la definición de objeto de codificación:

a) Prealineación y relleno.

b) Puntero de comienzo.

c) Espació de codificación.

d) Inversión de bits.

e) Relleno y justificación de valor.

f) Decodificación de valor.

23.14.4.2 El decodificador recuperará el número de rótulo a partir de los bits utilizados para codificarlo, decodificando desde una codificación de entero positivo.

23.15 Definición de objetos de codificación de clases en las demás categorías

En esta versión de la presente Recomendación | Norma Internacional no hay sintaxis definida para clases en las categorías siguientes:

objectidentifier (identificador de objeto)
opentype (tipo abierto)
real (real)

24 Especificación de la sintaxis definida para la clase de codificación #TRANSFORM

24.1 Resumen de propiedades de codificación y sintaxis definida

24.1.1 La sintaxis de la definición de objetos de codificación de la clase #TRANSFORM será:

```
#TRANSFORM ::= ENCODING-CLASS {

      -- int-to-int (véase 24.3)
      &int-to-int            CHOICE
                             {increment       INTEGER (1..MAX),
                             decrement        INTEGER (1..MAX),
                             multiply         INTEGER (2..MAX),
                             divide           INTEGER (2..MAX),
                             negate           ENUMERATED{value},
                             modulo           INTEGER (2..MAX),
                             subtract         ENUMERATED{lower-bound}
                             } OPTIONAL,

      -- bool-to-bool (véase 24.4)
      &bool-to-bool          CHOICE
                             {logical        ENUMERATED{not}}
                             DEFAULT logical:not,

      -- bool-to-int (véase 24.5)
      &bool-to-int           ENUMERATED {true-zero, true-one}
                             DEFAULT true-one,

      -- int-to-bool (véase 24.6)
      &int-to-bool                ENUMERATED {zero-true, zero-false}
                                  DEFAULT zero-false,
      &Int-to-bool-true-is        INTEGER OPTIONAL,
      &Int-to-bool-false-is       INTEGER OPTIONAL,

      -- int-to-chars (véase 24.7)
      &int-to-chars-size          ResultSize DEFAULT variable,
      &int-to-chars-plus          BOOLEAN DEFAULT FALSE,
      &int-to-chars-pad           ENUMERATED
                                  {spaces, zeros} DEFAULT zeros,
```

```
      -- int-to-bits (véase 24.8)
      &int-to-bits-encoded-as      ENUMERATED
                                   {positive-int, twos-complement}
                                   DEFAULT twos-complement,
      &int-to-bits-unit            Unit (1..MAX) DEFAULT bit,
      &int-to-bits-size            ResultSize DEFAULT variable,

      -- bits-to-int (véase 24.9)
      &bits-to-int-decoded-assuming     ENUMERATED
                                        {positive-int, twos-complement}
                                        DEFAULT twos-complement,

      -- char-to-bits (véase 24.10)
      &char-to-bits-encoded-as     ENUMERATED
                                   {iso10646, compact, mapped}
                                   DEFAULT compact,
      &Char-to-bits-chars          UniversalString (SIZE(1))
                                   ORDERED OPTIONAL,
      &Char-to-bits-values         BIT STRING ORDERED OPTIONAL,
      &char-to-bits-unit           Unit (1..MAX) DEFAULT bit,
      &char-to-bits-size           ResultSize DEFAULT variable,

      -- bits-to-char (véase 24.11)
      &bits-to-char-decoded-assuming    ENUMERATED
                                        {iso10646, mapped}
                                        DEFAULT iso10646,
      &Bits-to-char-values              BIT STRING ORDERED OPTIONAL,
      &Bits-to-char-chars               UniversalString (SIZE(1))
                                        ORDERED OPTIONAL,

      -- bit-to-bits (véase 24.12)
      &bit-to-bits-one             Non-Null-Pattern DEFAULT bits:'1'B,
      &bit-to-bits-zero            Non-Null-Pattern DEFAULT bits:'0'B,

      -- bits-to-bits (véase 24.13)
      &Source-values               BIT STRING ORDERED,
      &Result-values               BIT STRING ORDERED,

      -- chars-to-composite-char (véase 24.14)
      -- There are no encoding properties for this transformation

      -- bits-to-composite-bits (véase 24.15)
      &bits-to-composite-bits-unit      Unit (1..MAX) DEFAULT bit

      -- octets-to-composite-bits (véase 24.16)
      -- There are no encoding properties for this transformation

      -- composite-char-to-chars (véase 24.17)
      -- There are no encoding properties for this transformation

      -- composite-bits-to-bits (véase 24.18)
      -- There are no encoding properties for this transformation

      -- composite-bits-to-octets (véase 24.19)
      -- There are no encoding properties for this transformation
} WITH SYNTAX {

      -- Only one of the following clauses can be used.

      [INT-TO-INT &int-to-int]

      [BOOL-TO-BOOL [AS &bool-to-bool]]

      [BOOL-TO-INT AS &bool-to-int]

      [INT-TO-BOOL
            [AS &int-to-bool]
            [TRUE-IS &Int-to-bool-true-is]
            [FALSE-IS &Int-to-bool-false-is]]
```

```
        [INT-TO-CHARS
              [SIZE &int-to-chars-size]
              [PLUS-SIGN &int-to-chars-plus]
              [PADDING &int-to-chars-pad]]

        [INT-TO-BITS
              [AS &int-to-bits-encoded-as]
              [SIZE &int-to-bits-size]
              [MULTIPLE OF &int-to-bits-unit]]

        [BITS-TO-INT
              [AS &bits-to-int-decoded-assuming]]

        [CHAR-TO-BITS
              [AS &char-to-bits-encoded-as]
              [CHAR-LIST &Char-to-bits-chars]
              [BITS-LIST &Char-to-bits-values]
              [SIZE &char-to-bits-size]
              [MULTIPLE OF &char-to-bits-unit]]

        [BITS-TO-CHAR
              [AS &bits-to-char-decoded-assuming]
              [BITS-LIST &Bits-to-char-values]
              [CHAR-LIST &Bits-to-char-chars]]

        [BIT-TO-BITS
              [ZERO-PATTERN &bit-to-bits-zero]
              [ONE-PATTERN &bit-to-bits-one]]

        [BITS-TO-BITS
              SOURCE-LIST &Source-values
              RESULT-LIST &Result-values]

        [CHARS-TO-COMPOSITE-CHAR]

        [BITS-TO-COMPOSITE-BITS
              [UNIT &bits-to-composite-bits-unit]]

        [OCTETS-TO-COMPOSITE-BITS]

        [COMPOSITE-CHAR-TO-CHARS]

        [COMPOSITE-BITS-TO-BITS]

        [COMPOSITE-BITS-TO-OCTETS]

    }
```

24.2 Origen y destino de transformadas

24.2.1 La clase de codificación **#TRANSFORM** permite la especificación de procedimientos que transforman valores abstractos de entrada (el origen) en valores abstractos de salida del mismo tipo o de un tipo diferente (el resultado). Permite además la especificación de procedimientos que establecen la correspondencia entre un origen cadena de caracteres, cadena de octetos o cadena de bits en un compuesto de transformadas, y un compuesto de transformadas (cuyos valores son un carácter único, un octeto único o cadenas de bits con un tamaño unitario fijado) en un valor abstracto (cadena de caracteres, cadena de octetos o cadena de bits). El origen es el resultado de una transformada previa o se obtiene a partir de una clase origen (véase 19.4). El resultado es el origen de la transformada siguiente o se convierte en asociado a una clase destino (véase 19.4).

NOTA – La cláusula 23 utiliza también transformadas cuyo origen es un bit único y un carácter único.

24.2.2 Estas transformadas se utilizan en la definición del establecimiento de la correspondencia de valores y en la definición de objetos de codificación de clases de codificación en el grupo de categorías campo de bits (véanse las cláusulas 20 a 23).

24.2.3 El origen y el resultado se indican mediante palabras ("**INT-TO-INT**", "**BOOL-TO-BOOL**", etc.) en la especificación de un objeto de codificación **#TRANSFORM**, y se definen en el texto asociado.

24.2.4 Las subcláusulas 24.2.4.1 a 24.2.4.3 especifican las reglas de utilización de las transformadas de manera sucesiva, y de las clases origen y destino de una lista de transformadas.

24.2.4.1 Cuando se especifican objetos de codificación de la clase **#TRANSFORM** en una lista ordenada, el origen de un objeto de codificación **#TRANSFORM** siguiente será el resultado del objeto de codificación **#TRANSFORM** precedente.

24.2.4.2 Para la primera y la última transformadas de una lista ordenada utilizadas en la definición de objetos de codificación en las cláusulas 22 y 23, el texto de dichas cláusulas especifica el origen de la primera transformada y el resultado requerido de la última transformada.

24.2.4.3 Para la primera y la última transformadas de una lista ordenada utilizadas en la especificación del establecimiento de la correspondencia de valores mediante transformadas en 19.4, el texto de esa subcláusula especifica una clase origen y una clase destino, y ambas serán de la categoría cadena de bits, valor booleano, cadena de caracteres, entero o cadena de octetos (véase 19.4.2). El origen requerido de la primera transformada y el resultado requerido de la última transformada (para cada una de esas categorías) se especifican en 24.2.7.

24.2.5 El texto de esta subcláusula especifica el origen de una transformada y el resultado de una transformada como un valor entero, un valor booleano, una cadena de caracteres, una cadena de bits, un carácter único o un bit único (sólo origen). El origen y el resultado de una transformada pueden ser también un compuesto de estos valores. Los compuestos de transformadas sólo pueden ser producidos por transformadas, y deben ser procesados por otra transformada (el texto) de la lista de transformadas. Hay dos grupos de transformadas: el de las diseñadas para crear compuestos a partir de valores abstractos o producir un valor abstracto a partir de un compuesto y el de las diseñadas para transformar valores únicos. Estas últimas pueden transformar también compuestos de esos valores, produciendo como resultado un compuesto que es la transformada de cada elemento del compuesto origen.

24.2.6 Un origen o un destino que sean un bit único o un carácter único sólo ocurren cuando transformadas sucesivas los tienen como salida y entrada, o según se especifica en las cláusulas 22 y 23. La primera transformada de la lista ordenada referenciada en 19.4 no tendrá un origen que sea un bit único o un carácter único. La última transformada de la lista ordenada referenciada en 19.4 no tendrá un destino que sea un bit único o un carácter único.

24.2.7 Cuando se utilicen en 19.4, el origen de la primera transformada y el destino de la última transformada serán lo mismo que la categoría de la clase de codificación origen y la clase de codificación destino (respectivamente), con las excepciones que se indican a continuación. Cuando la categoría de la clase de codificación origen sea cadena de octetos, el origen de la primera transformada será cadena de bits (tratando cada valor cadena de octetos como un valor cadena de bits). Cuando la última transformada sea "**BITS-TO-BITS**" con "**MULTIPLE OF**" fijado en 8, la clase destino podrá ser cadena de octetos.

24.2.8 Las subcláusulas que siguen especifican las condiciones impuestas a los valores abstractos del origen que permiten definir una transformada como reversible. Si esos valores se dan a una transformada que ha de ser reversible, se trata de un error de la especificación ECN o de la aplicación, y los codificadores no generarán codificaciones para los mismos.

24.3 Transformada int-to-int

NOTA – En D.1.2.2 se dan ejemplos de esta transformada.

24.3.1 La transformada int-to-int utiliza la propiedad de codificación siguiente:

```
&int-to-int             CHOICE
                        {increment      INTEGER (1..MAX),
                        decrement       INTEGER (1..MAX),
                        multiply        INTEGER (2..MAX),
                        divide          INTEGER (2..MAX),
                        negate          ENUMERATED{value},
                        modulo          INTEGER (2..MAX),
                        subtract        ENUMERATED{lower-bound}
                        } OPTIONAL
```

24.3.2 La sintaxis de la transformada int-to-int será:

```
[INT-TO-INT &int-to-int]
```

24.3.3 Tanto el origen como el resultado de esta transformada son un entero o un compuesto de enteros. No hay límites asociados al resultado a menos que se trate de la última transformada de un establecimiento de la correspondencia mediante transformadas (véase 19.4) (lo que significa que ni el origen ni el destino pueden ser un compuesto) y la clase destino del establecimiento de la correspondencia mediante transformadas tenga límites. Si en este caso la transformada se aplica a valores enteros origen que no tienen correspondencia en los límites de la clase destino, se trata de un error de la especificación ECN o de la aplicación.

24.3.4 Una transformada int-to-int se define dando un valor a "**INT-TO-INT**" y permitiendo que cualquier objeto de codificación dado especifique de manera precisa una operación aritmética. No obstante, la aritmética de carácter general se puede definir utilizando una lista ordenada de transformadas (esto se autoriza siempre que intervengan transformadas que impliquen enteros).

24.3.5 Los valores "**increment:n**", "**decrement:n**", "**multiply:n**" y "**negate:n**" tienen su significado matemático normal.

24.3.6 El valor "**divide:n**" produce, por definición, un resultado entero que es el valor entero más cercano al resultado matemático, pero no va más allá de cero que ese resultado. En términos de programación, "**divide:n**" trunca hacia cero, por lo que un valor de –1 con "**divide:2**" dará cero.

24.3.7 La transformada para el valor "**modulo:n**" se define como sigue: sean "i" el valor entero original y "**modulo:n**" la transformada. Sea "j" el resultado de aplicar "**divide:n**" seguido por "**multiply:n**" a "i". La transformada "**modulo:n**" aplicada a "i" equivale entonces, por definición, a la aplicación de "**decrement:j**" a "i".

24.3.8 La transformada para el valor "**subtract:lower-bound**" sólo se utilizará como la primera de una lista ordenada de transformadas (y por tanto nunca puede ser utilizada si el oigen es un compuesto). El origen tendrá un límite inferior.

24.3.9 Cada una de estas transformadas se define como reversible si el origen es un valor único, no un compuesto, y si se cumple la condición impuesta al valor abstracto (al que se aplica) que se indica en el cuadro 6. Se dice también que es reversible si el origen es un compuesto y en el cuadro 6 se especifica como condición *Siempre reversible*.

Cuadro 6 – Reversión de transformadas "int-to-int"

Transformada	Condición
increment:n	*Siempre reversible*
decrement:n	*Siempre reversible*
multiply:n	*Siempre reversible*
divide:n	*El valor es un múltiplo de n*
negate:value	*Siempre reversible*
modulo:n	*Nunca reversible*
subtract:lower-bound	*Siempre reversible*

24.4 Transformada bool-to-bool

24.4.1 La transformada bool-to-bool utiliza la propiedad de codificación siguiente:

```
&bool-to-bool          CHOICE
                       {logical        ENUMERATED{not}}
                       DEFAULT logical:not
```

24.4.2 La sintaxis de la transformada bool-to-bool será:

```
[BOOL-TO-BOOL [AS &bool-to-bool]]
```

24.4.3 Tanto el origen como el resultado de esta transformada son un booleano o un compuesto de booleanos.

24.4.4 Si el origen es un booleano, el resultado es un booleano. Si el origen es un compuesto de booleanos, el resultado es un compuesto de booleanos en el que cada elemento del origen ha sido transformado como se especifica en 24.4.5.

24.4.5 Sólo hay un valor para "**BOOL-TO-BOOL**", "**AS logical:not**", que puede ser omitido. Esta transformada convierte el booleano "**TRUE**" en "**FALSE**", y viceversa.

24.4.6 Esta transformada se define como reversible para todos los valores abstractos.

24.5 Transformada bool-to-int

24.5.1 La transformada bool-to-int utiliza la propiedad de codificación siguiente:

```
&bool-to-int            ENUMERATED {true-zero, true-one}
                        DEFAULT true-one
```

24.5.2 La sintaxis de la transformada bool-to-int será:

```
[BOOL-TO-INT AS &bool-to-int]
```

24.5.3 El origen de esta transformada es un booleano o un compuesto de booleanos y el resultado es un entero o un compuesto de enteros. El resultado entero (y cada elemento del compuesto de enteros) tiene el valor cero o uno. El resultado no tiene límites asociados.

24.5.4 Si el origen es un booleano, el resultado es un entero. Si el origen es un compuesto de booleanos, el resultado es un compuesto de enteros en el que cada elemento del origen ha sido transformado como se especifica en 24.5.5.

24.5.5 El valor "**true-zero**" de "**BOOL-TO-INT**" produce el entero 0 para "**TRUE**" y el entero 1 para "**FALSE**". El valor "**true-one**" produce el entero 1 para "**TRUE**" y el entero 0 para "**FALSE**".

24.5.6 Esta transformada se define como reversible para todos los valores abstractos.

24.6 Transformada int-to-bool

24.6.1 La transformada int-to-bool utiliza las propiedades de codificación siguientes:

```
&int-to-bool                ENUMERATED {zero-true, zero-false}
                            DEFAULT zero-false,
&Int-to-bool-true-is        INTEGER OPTIONAL,
&Int-to-bool-false-is       INTEGER OPTIONAL
```

24.6.2 La sintaxis de la transformada int-to-bool será:

```
[INT-TO-BOOL
      [AS &int-to-bool]
      [TRUE-IS &Int-to-bool-true-is]
      [FALSE-IS &Int-to-bool-false-is]]
```

24.6.3 El origen de esta transformada es un entero o un compuesto de enteros y el resultado es un booleano o un compuesto de booleanos.

24.6.4 O bien se fija "**AS**", "**TRUE-IS**" o "**FALSE-IS**", o bien se fijan "**TRUE-IS**" y "**FALSE-IS**" (y no se fija "**AS**"), o no se fija ninguno. Si no se fija ninguno, se supone el valor por defecto para "**AS**".

24.6.5 Si se fija "**AS**" (o se toma su valor por defecto), el valor "**zero-true**" produce "**TRUE**" para el valor cero y "**FALSE**" para todos los valores distintos de cero, y el valor "**zero-false**" produce "**FALSE**" para el valor cero y "**TRUE**" para todos los valores distintos de cero.

24.6.6 Si sólo se fija "**TRUE-IS**", todos los valores enteros para "**TRUE-IS**" producen "**TRUE**" y todos los demás valores enteros producen "**FALSE**".

24.6.7 Si sólo se fija "**TRUE-IS**", todos los valores enteros para "**TRUE-IS**" producen "**FALSE**" y todos los demás valores enteros producen "**TRUE**".

24.6.8 Si se fijan "**TRUE-IS**" y "**FALSE-IS**", se separarán los valores enteros de "**TRUE-IS**" y "**FALSE-IS**". En este caso, si los valores abstractos no incluidos en "**TRUE-IS**" ni en "**FALSE-IS**" están incluidos en el origen, se trata de un error de la especificación ECN o de la aplicación, y los codificadores no generarán codificaciones para esos valores.

24.6.9 La transformada se define como reversible si, y solamente si, se fijan tanto "**TRUE-IS**" como "**FALSE-IS**" y cada uno de ellos especifica un valor entero único.

24.7 Transformada int-to-chars

24.7.1 La transformada int-to-chars utiliza las propiedades de codificación siguientes:

```
&int-to-chars-size          ResultSize DEFAULT variable,
&int-to-chars-plus          BOOLEAN DEFAULT FALSE,
&int-to-chars-pad           ENUMERATED
                            {spaces, zeros} DEFAULT zeros
```

24.7.2 La sintaxis de la transformada int-to-chars será:

```
[INT-TO-CHARS
      [SIZE &int-to-chars-size]
      [PLUS-SIGN &int-to-chars-plus]
      [PADDING &int-to-chars-pad]]
```

24.7.3 La definición del tipo utilizado en la transformada int-to-chars es como sigue:

```
ResultSize ::= INTEGER {variable(-1), fixed-to-max(0)} (-1..MAX) --  (véase 21.14)
```

24.7.4 El origen de esta transformada es un entero, y el resultado es una cadena de caracteres o un compuesto de cadenas de caracteres.

24.7.5 Si el origen es un entero, el resultado es una cadena de caracteres. Si el origen es un compuesto de enteros, el resultado es un compuesto de cadenas de caracteres en el que cada elemento del origen ha sido transformado como se especifica en 24.7.6 a 24.7.13.

24.7.6 "**SIZE**", "**PLUS-SIGN**" y "**PADDING**" tienen valores por defecto y pueden ser omitidos.

24.7.7 "**SIZE**" especifica:

a) un tamaño fijo en caracteres para el tamaño resultante (un valor positivo de "**SIZE**"); o

b) que se ha de producir una cadena de caracteres de longitud variable (el valor "**variable**" de "**SIZE**"); o

c) un tamaño fijo y justo para contener la transformada de todos los valores abstractos de la clase origen (el valor "**fixed-to-max**" de "**SIZE**").

24.7.8 "**SIZE**" no se fijará en "**fixed-to-max**" a menos que ésta sea la primera transformada de un conjunto ordenado, y la clase origen tenga un límite superior y un límite inferior. Esto equivale a la especificación de un valor positivo igual al valor más pequeño necesario para contener la transformada de todos los valores abstractos dentro de los límites.

24.7.9 El valor entero es el primero que se convierte en una representación decimal sin ceros iniciales y con un prefijo "-" (HYPHEN-MINUS) si es negativo. Si, y solamente si, "**PLUS-SIGN**" se fija en verdadero, los valores positivos tienen un prefijo "+" (PLUS-SIGN) delante de los dígitos.

24.7.10 El dígito más significativo estará en el extremo inicial de la cadena de caracteres.

24.7.11 Si "**SIZE**" es "**variable**", es la cadena de caracteres resultante. En este caso, especificar un valor para "**PADDING**" no es un error, pero el valor es ignorado.

24.7.12 Si "**SIZE**" es un valor positivo o "**fixed-to-max**", y la cadena resultante (en una instancia de aplicación de esta transformada durante la codificación) es demasiado grande para el tamaño fijo, se trata de un error de la especificación ECN o de la aplicación, y los codificadores no generarán codificaciones para esos valores abstractos.

24.7.13 Si "**SIZE**" es un valor positivo o "**fixed-to-max**", y la cadena es más pequeña que el tamaño fijo, se rellena con " " (espacios) o "0" (dígitos cero), según determine el valor de "**PADDING**", como prefijo para producir el tamaño especificado.

24.7.14 Esta transformada se define como reversible para todos los valores abstractos.

24.8 Transformada int-to-bits

NOTA – En D.1.5.5 se da un ejemplo de esta transformada.

24.8.1 La transformada int-to-bits utiliza las propiedades de codificación siguientes:

```
&int-to-bits-encoded-as      ENUMERATED
                             {positive-int, twos-complement}
                             DEFAULT twos-complement,
&int-to-bits-unit            Unit (1..MAX) DEFAULT bit,
&int-to-bits-size            ResultSize DEFAULT variable
```

24.8.2 La sintaxis de la transformada int-to-bits será:

```
[INT-TO-BITS
      [AS &int-to-bits-encoded-as]
      [SIZE &int-to-bits-size]
      [MULTIPLE OF &int-to-bits-unit]]
```

24.8.3 Las definiciones de los tipos utilizados en la transformada int-to-bits son como sigue:

```
Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
       dword32(32)} (0..256) --  (véase 21.1)

ResultSize ::= INTEGER {variable(-1), fixed-to-max(0)} (-1..MAX) --  (véase 21.14)
```

24.8.4 El origen de esta transformada es un entero o un compuesto de enteros y el resultado es una cadena de bits o un compuesto de cadenas de bits. No hay límites asociados al resultado. En las cláusulas que siguen se emplea la expresión cadena de bits resultante.

24.8.5 Si el origen es un entero, el resultado es una cadena de bits resultante. Si el origen es un compuesto de enteros, el resultado es un compuesto de cadenas de bits en el que cada elemento del origen ha sido transformado en la cadena de bits resultante especificada en 24.5.5.

24.8.6 "**AS**" y "**MULTIPLE OF**" tienen valores por defecto y no es preciso fijarlos.

24.8.7 "**SIZE**" tiene un valor por defecto y no es preciso fijarlo si el origen no es un compuesto. Si el origen es un compuesto, se fijará en un valor positivo.

24.8.8 "**SIZE**" no se fijará en "**fixed-to-max**" a menos que ésta sea la primera transformada de un conjunto ordenado de la sintaxis definida en 19.4, y la clase origen tenga un límite superior y un límite inferior. Esto equivale a la especificación de un valor positivo igual al valor más pequeño necesario para contener la transformada de todos los valores abstractos dentro de los límites.

NOTA – "**SIZE**" no se puede fijar en "**fixed-to-max**" si el origen es un compuesto de transformadas.

24.8.9 "**AS**" selecciona la codificación del entero como una codificación en complemento de 2 o una codificación de entero positivo. La definición de estas codificaciones se da en 8.3.2 y 8.3.3 de la Rec. UIT-T X.690 | ISO/CEI 8825-1.

24.8.10 El bit más significativo estará en el extremo inicial de la cadena de bits.

24.8.11 El entero será codificado primero en el número mínimo de bits necesario para producir una cadena de bits inicial. Esto significa que una codificación de entero positivo no tendrá cero como bit inicial (a menos que haya un bit cero único en la codificación), y una codificación en complemento de 2 no tendrá dos bits cero iniciales sucesivos o dos bits uno iniciales sucesivos.

24.8.12 Si "**AS**" se fija en "**positive-int**", y el valor que se ha de transformar es negativo, se trata de un error de la especificación ECN o de la aplicación y los codificadores no codificarán esos valores.

24.8.13 Si "**SIZE**" es "**variable**", la cadena de bits inicial pasa a ser la cadena de bits resultante. En este caso, no es un error especificar un valor para "**MULTIPLE OF**", pero el valor es ignorado.

NOTA – Esta cláusula no se puede aplicar si el origen es compuesto.

24.8.14 Si "**SIZE**" es un valor positivo, el tamaño de la cadena de bits resultante será "**MULTIPLE OF**" multiplicado por "**SIZE**".

24.8.15 Si "**SIZE**" es "**fixed-to-max**", el tamaño de la cadena de bits resultante será el múltiplo más pequeño de "**MULTIPLE OF**" que sea lo bastante grande como para recibir la codificación de cualquier valor abstracto de la clase a la que se aplica la transformada.

NOTA – Esta cláusula no se puede aplicar si el origen es un compuesto.

24.8.16 Si la cadena de bits inicial (en una instancia de la aplicación de esta transformada durante la codificación) es demasiado grande para el tamaño fijo, se trata de un error de la especificación ECN o de la aplicación y los codificadores no codificarán esos valores.

24.8.17 Si la cadena de bits inicial tiene un tamaño inferior al especificado, se aplicará un prefijo de cero bits a la codificación de un entero positivo para producir la cadena de bits resultante. Si la codificación es en complemento de 2, se aplicará un prefijo de bits de un valor igual al del bit inicial original para producir la cadena de bits resultante.

24.8.18 Esta transformada se define como reversible para todos los valores abstractos. Esta transformada produce una cadena de bits autodelimitante si, y solamente si, "**SIZE**" no es "**variable**" y el origen no es un compuesto. Un resultado compuesto nunca es autodelimitante.

24.9 Transformada bits-to-int

24.9.1 La transformada bits-to-int utiliza la propiedad de codificación siguiente:

```
&bits-to-int-decoded-assuming      ENUMERATED
                                   {positive-int, twos-complement}
                                   DEFAULT twos-complement
```

24.9.2 La sintaxis de la transformada bits-to-int será:

```
[BITS-TO-INT
      [AS &bits-to-int-decoded-assuming]]
```

24.9.3 El origen de esta transformada es una cadena de bits o un compuesto de cadenas de bits y el resultado es un entero o un compuesto de enteros. No hay límites asociados al resultado.

24.9.4 Si el origen es una cadena de bits, el resultado es un entero. Si el origen es un compuesto de cadenas de bits, el resultado es un compuesto de enteros en el que cada entero es el resultado de la especificación de 24.9.5.

24.9.5 El valor entero se producirá interpretando los bits como una codificación en complemento de 2 o como una codificación de entero positivo, según se especifica en 8.3.2 y 8.3.3 de la Rec. UIT-T X.690 | ISO/CEI 8825-1. El valor de "**AS**" (o su valor por defecto si no está fijado) determina la codificación que se ha de suponer.

24.9.6 Esta transformada no se utilizará cuando se requieran transformadas reversibles.

24.10 Transformada char-to-bits

24.10.1 La transformada char-to-bits utiliza las propiedades de codificación siguientes:

```
&char-to-bits-encoded-as     ENUMERATED
                             {iso10646, compact, mapped}
                             DEFAULT compact,
&Char-to-bits-chars          UniversalString (SIZE(1))
                             ORDERED OPTIONAL,
&Char-to-bits-values         BIT STRING ORDERED OPTIONAL,
&char-to-bits-unit           Unit (1..MAX) DEFAULT bit,
&char-to-bits-size           ResultSize DEFAULT variable
```

24.10.2 La sintaxis de la transformada char-to-bits será:

```
[CHAR-TO-BITS
      [AS &char-to-bits-encoded-as]
      [CHAR-LIST &Char-to-bits-chars]
      [BITS-LIST &Char-to-bits-values]
      [SIZE &char-to-bits-size]
      [MULTIPLE OF &char-to-bits-unit]]
```

24.10.3 Las definiciones de los tipos utilizados en la transformada char-to-bits son como sigue:

```
Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
       dword32(32)} (0..256) --  (véase 21.1)

ResultSize ::= INTEGER {variable(-1), fixed-to-max(0)} (-1..MAX) --  (véase 21.14)
```

24.10.4 El origen de esta transformada es un carácter único procedente de:

a) la especificación de una codificación de la categoría cadena de caracteres (véase 23.4.2.1); o

b) un compuesto de caracteres únicos

y el resultado es una cadena de bits en el caso a) y un compuesto de cadenas de bits en el caso b).

24.10.5 El compuesto de cadenas de bits del caso b) será la secuencia ordenada de las cadenas de bits producidas por las transformaciones siguientes aplicadas a cada elemento del compuesto de cadenas de bits origen. Si el "**ZERO-PATTERN**" y el "**ONE-PATTERN**" tienen tamaños diferentes, se trata de un error de la especificación ECN.

24.10.6 El origen de esta transformada es un carácter único o un compuesto de caracteres únicos. Si el origen es un carácter único, el resultado es una cadena de bits. Si el origen es un compuesto de caracteres únicos, el resultado es un compuesto de cadenas de bits.

24.10.7 Cuando el origen es un compuesto, el compuesto resultante se determina aplicando la especificación siguiente a todos los elementos del compuesto origen para formar el compuesto resultado. Si esta transformada se aplica a un compuesto con "**AS**" fijado en "**mapped**" y el tamaño de todas las cadenas de bits de la "**BITS-LIST**" no es el mismo, se trata de un error de la especificación ECN.

24.10.8 Cuando el texto siguiente se refiere a una posible "constricción de alfabeto permitido efectivo", esa constricción existe si, y solamente si, la transformada es la primera de una lista ordenada utilizada en 23.4 y la clase a la que se aplica el objeto de codificación tiene una constricción de alfabeto permitido efectivo.

NOTA – Esto sólo puede ocurrir si la clase a la que se aplica la transformada forma parte de una estructura generada implícita o explícitamente. Esta cláusula no se puede aplicar a un compuesto cuyos elementos nunca tienen constricciones de alfabeto permitido efectivo.

24.10.9 "**AS**", "**SIZE**" y "**MULTIPLE OF**" tienen valores por defecto y no es preciso fijarlos. "**CHAR-LIST**" y "**BITS-LIST**" sólo se utilizan si "**AS**" se fija en "**mapped**", en cuyo caso su presencia es obligatoria, y contendrán por lo menos un elemento de la lista ordenada.

24.10.10 La ECN sólo admite caracteres del conjunto de caracteres ISO/CEI 10646-1. Cuando se utilizan tipos ASN.1 tales como "GeneralString", en teoría pueden aparecer caracteres ajenos a ese conjunto de caracteres. Esos caracteres no son admitidos por esta transformada.

24.10.11 Si "**AS**" es "**mapped**", la transformada es especificada por los valores de "**CHAR-LIST**" y "**BITS-LIST**", que a su vez serán especificados, y los valores de "**MULTIPLE OF**" y "**SIZE**" son ignorados. La transformada se especifica en 24.10.11.1 a 24.10.11.5.

24.10.11.1 "**CHAR-LIST**" y "**BITS-LIST**" son, respectivamente, una lista ordenada de caracteres únicos y valores cadenas de bits. (Estos parámetros son ignorados si "**AS**" no está fijado en "**mapped**").

24.10.11.2 En cada lista habrá un número igual de valores, y todos los valores de caracteres de la "**CHAR-LIST**" serán distintos.

24.10.11.3 La transformada de un carácter de "**CHAR-LIST**" es la cadena de bits especificada en la posición correspondiente de "**BITS-LIST**".

24.10.11.4 Si en una instancia de aplicación de esta transformada se ha de transformar un carácter que no está en la "**CHAR-LIST**", se trata de un error de la especificación ECN o de la aplicación.

NOTA – Por lo general, sólo se podrá comprobar este error con una herramienta en el momento de la codificación, ya que las restricciones impuestas a los valores abstractos posibles quizá no estén presentes formalmente en la especificación ASN.1.

24.10.11.5 En este caso ("**AS**" fijado en "**mapped**"), la transformada se define como reversible (para todos los valores abstractos) si, y solamente si, todos los valores cadena de bits de "**BITS-LIST**" son distintos; de no ser así, no se utilizará ese conjunto de valores cuando se requiera una transformada reversible. El resultado es autodelimitante si los valores cadena de bits de "**BITS-LIST**" son autodelimitantes (véase 3.2.41). Un resultado compuesto nunca es autodelimitante.

24.10.12 Si "**AS**" es "**iso10646**", la transformada se especifica en 24.10.12.1 a 24.10.12.5.

24.10.12.1 El carácter se convierte primero en un entero con el valor numérico especificado en ISO/CEI 10646-1.

NOTA – ISO/CEI 10646-1 incluye los llamados caracteres de control ASCII, que tienen posiciones en la fila 1.

24.10.12.2 Si el carácter procede de una cadena de caracteres que tiene asociada una constricción de alfabeto permitido efectivo (véase 24.10.8), el entero tiene constricciones de tamaño efectivo que permiten contener justamente los valores numéricos de todos los caracteres del alfabeto permitido efectivo.

24.10.12.3 Si no hay ninguna constricción de alfabeto permitido efectivo, el entero tiene asociada una constricción de tamaño efectivo de 0...32767.

24.10.12.4 Este valor entero se convierte a continuación en bits utilizando la transformada:

```
INT-TO-BITS -- (véase 24.8)
      AS positive-int
      SIZE <size>
      MULTIPLE OF <multiple-of>
```

donde "<size>" es el valor de "**SIZE**" y "<multiple-of>" es el valor de "**MULTIPLE OF**" para la transformada char-to-bits. ("**SIZE**" y "**MULTIPLE OF**" toman sus valores por defecto si no están fijados).

24.10.12.5 En este caso ("**AS**" fijado en "**iso10646**"), la transformada se define como reversible para todos los valores abstractos. Produce una cadena de bits autodelimitante si, y solamente si, si "**SIZE**" no es "**variable**". Un resultado compuesto nunca es autodelimitante.

24.10.13 Si "**AS**" es "**compact**", se trata un error de la especificación ECN caso de que no haya ninguna constricción de alfabeto permitido efectivo; de otro modo, la transformada se especifica en 24.10.13.1 a 24.10.13.4.

24.10.13.1 Todos los caracteres del alfabeto permitido efectivo se ponen en orden canónico utilizando su valor ISO/CEI 10646-1, siendo el valor más bajo el primero. Al primero de la lista se le asigna entonces el valor entero cero, al siguiente el uno, y así sucesivamente.

24.10.13.2 Si el alfabeto permitido efectivo contiene "n" caracteres, el entero tiene una constricción de tamaño efectivo de 0...n-1.

24.10.13.3 Este valor entero se convierte a continuación en bits utilizando la transformada:

```
INT-TO-BITS -- (véase 24.8)
      AS positive-int
      SIZE <size>
      MULTIPLE OF <multiple-of>
```

donde "<size>" es el valor de "**SIZE**" y "<multiple-of>" es el valor de "**MULTIPLE OF**" para la transformada char-to-bits. ("**SIZE**" y "**MULTIPLE OF**" toman sus valores por defecto si no están fijados).

NOTA – La codificación PER de tipos cadena de caracteres utiliza el equivalente de "**compact**" solamente si la aplicación de este algoritmo reduce el número de bits requerido para codificar caracteres (utilizando "**fixed-to-max**"). Ese grado de control no es posible en la presente versión de esta Recomendación | Norma Internacional.

24.10.13.4 En este caso ("**AS**" fijado en "**compact**"), la transformada se define como reversible para todos los valores abstractos. Produce una cadena de bits autodelimitante si, y solamente si, "**SIZE**" no es "**variable**". Un resultado compuesto nunca es autodelimitante.

24.11 Transformada bits-to-char

24.11.1 La transformada bits-to-char utiliza las propiedades de codificación siguientes:

```
&bits-to-char-decoded-assuming    ENUMERATED
                                  {iso10646, mapped}
                                  DEFAULT iso10646,
&Bits-to-char-values              BIT STRING ORDERED OPTIONAL,
&Bits-to-char-chars               UniversalString (SIZE(1))
                                  ORDERED OPTIONAL
```

24.11.2 La sintaxis de la transformada bits-to-char será:

```
[BITS-TO-CHAR
      [AS &bits-to-char-decoded-assuming]
      [BITS-LIST &Bits-to-char-values]
      [CHAR-LIST &Bits-to-char-chars]]
```

24.11.3 El origen de esta transformada es una cadena de bits o un compuesto de cadenas de bits. Si el origen es una cadena de bits, el resultado es un carácter único. Si el origen es un compuesto de cadenas de bits, el resultado es un compuesto de caracteres únicos.

24.11.4 Si el origen es un compuesto de cadenas de bits, el compuesto de caracteres únicos resultante es una lista ordenada de caracteres únicos obtenida a partir de la transformación de cada uno de los elementos del compuesto de cadenas de bits.

24.11.5 Si "**AS**" es "**iso10646**", la cadena de bits será interpretada como una codificación de entero positivo que contiene el valor numérico ISO/CEI 10646-1 de un carácter. Si el valor entero excede de 32767, se trata de un error de la especificación ECN.

24.11.6 Si "**AS**" es "**mapped**", la transformada es especificada por los valores de "**CHAR-LIST**" y "**BITS-LIST**". La transformada se define en 24.11.6.1 a 24.11.6.5.

24.11.6.1 "**CHAR-LIST**" y "**BITS-LIST**" son, respectivamente, una lista ordenada de caracteres únicos y de valores cadenas de bits. (Estos parámetros son ignorados si **"AS"** no está fijado en "**mapped**").

24.11.6.2 No habrá un número igual de valores en cada lista, y todos los valores de caracteres y valores cadenas de bits de la lista serán distintos.

24.11.6.3 La transformada de una cadena de bits de la "**BITS-LIST**" es el carácter especificado en la posición correspondiente de la "**CHAR-LIST**".

24.11.6.4 Si en una instancia de aplicación de esta transformada se ha de transformar una cadena de bits que no está en la "**BITS-LIST**", se trata de un error de la especificación ECN o de la aplicación.

NOTA – Por lo general, sólo se podrá comprobar este error con una herramienta en el momento de la codificación, ya que las restricciones impuestas a los valores abstractos posibles quizá no estén presentes formalmente en la especificación ASN.1.

24.11.6.5 La transformada se define como reversible para todos los valores abstractos.

24.12 Transformada bit-to-bits

24.12.1 La transformada bit-to-bits utiliza las propiedades de codificación siguientes:

```
&bit-to-bits-one            Non-Null-Pattern DEFAULT bits:'1'B,
&bit-to-bits-zero           Non-Null-Pattern DEFAULT bits:'0'B
```

24.12.2 La sintaxis de la transformada bit-to-bits será:

```
[BIT-TO-BITS
     [ZERO-PATTERN &bit-to-bits-zero]
     [ONE-PATTERN &bit-to-bits-one]]
```

24.12.3 La definición del tipo utilizado en la transformada bit-to-bits es como sigue:

```
Non-Null-Pattern ::= Pattern
     (ALL EXCEPT (bits:''B | octets:''H | char8:"" | char16:"" |
                  char32:"")) -- (véase 21.10.2)
```

24.12.4 El origen de esta transformada es un bit único procedente de:

a) la especificación de una codificación para la categoría cadena de bits (véase 23.2); o

b) un compuesto de cadenas de bits con una unidad de 1 bit.

y el resultado es una cadena de bits en el caso a) y un compuesto de cadenas de bits en el caso b).

24.12.5 El compuesto de cadenas de bits del caso b) será la secuencia ordenada de las cadenas de bits producidas por las transformaciones siguientes aplicadas a cada elemento del compuesto de cadenas de bits origen. Si el "**ZERO-PATTERN**" y el "**ONE-PATTERN**" tienen tamaños diferentes, se trata de un error de la especificación ECN.

24.12.6 De los esquemas "**ZERO-PATTERN**" y "**ONE-PATTERN**", uno como máximo será "**different:any**".

NOTA – Un valor de "**different:any**" significa aquí un esquema que es diferente de los otros esquemas, pero tiene la misma longitud.

24.12.7 La alternativa "**any-of-length**" no será utilizada para "**ZERO-PATTERN**" o para "**ONE-PATTERN**".

24.12.8 Si el bit está fijado en cero, el resultado es el "**ZERO-PATTERN**". Si el bit está fijado en uno, el resultado es el "**ONE-PATTERN**".

24.12.9 Si "**ZERO-PATTERN**" y "**ONE-PATTERN**" son el mismo, o si uno de ellos es una subcadena inicial del otro, se trata de un error de la especificación ECN.

24.12.10 Esta transformada se define como reversible para todos los valores abstractos y el resultado es autodelimitante a menos que la transformada se aplique a un compuesto. Un resultado compuesto nunca es autodelimitante.

24.13 Transformada bits-to-bits

24.13.1 La transformada bits-to-bits utiliza las propiedades de codificación siguientes:

```
&Source-values              BIT STRING ORDERED,
&Result-values              BIT STRING ORDERED
```

24.13.2 La sintaxis de la transformada bits-to-bits será:

```
[BITS-TO-BITS
     SOURCE-LIST &Source-values
     RESULT-LIST &Result-values]
```

24.13.3 El origen de esta transformada es una cadena de bits o un compuesto de cadenas de bits. Si el origen es una cadena de bits, el resultado es un carácter único. Si el origen es un compuesto de cadenas de bits, el resultado es un compuesto de caracteres únicos.

24.13.4 Si el origen es un compuesto de cadenas de bits, el compuesto de cadenas de bits resultante es la lista ordenada de cadenas de bits obtenida aplicando la especificación siguiente a cada una de las cadenas de bits del origen.

24.13.5 "**SIZE**" y "**MULTIPLE OF**" tienen valores por defecto y no es preciso fijarlos. Se requieren "**SOURCE-LIST**" y "**RESULT-LIST**", que contendrán al menos un elemento de la lista ordenada.

24.13.6 La transformada es especificada por los valores de "**SOURCE-LIST**" y "**RESULT-LIST**".

24.13.7 Habrá un número igual de valores cadena de bits en cada lista y todos los valores cadena de bits de "**SOURCE-LIST**" serán distintos.

24.13.8 La transformada de una cadena de bits de "**SOURCE-LIST**" es la cadena de bits especificada en la posición correspondiente de "**RESULT-LIST**".

24.13.9 Si esta transformada se aplica a un compuesto, todas las cadenas de bits de la "**RESULT-LIST**" tendrán el mismo tamaño.

24.13.10 Si en una instancia de aplicación de esta transformada, una cadena de bits origen no está en la "**SOURCE-LIST**", se trata de un error de la especificación ECN o de la aplicación.

NOTA – Por lo general, sólo se podrá comprobar este error con una herramienta en el momento de la codificación, ya que las restricciones impuestas a los valores abstractos posibles quizá no estén presentes formalmente en la especificación ASN.1

24.13.11 La transformada se define como reversible (para todos los valores abstractos) si, y solamente si, todos los valores cadena de bits de "**RESULT-LIST**" son distintos; de no ser así, no se utilizará ese conjunto de valores cuando se requiera una transformada reversible. El resultado es autodelimitante si los valores cadena de bits de "**RESULT-LIST**" son distintos y autodelimitantes (véase 3.2.41) y la transformada se aplica a una cadena de bits. Un resultado compuesto nunca es autodelimitante.

24.14 Transformada chars-to-composite-char

24.14.1 La transformada chars-to-composite-char convierte una cadena de caracteres en un compuesto de caracteres únicos.

24.14.2 La sintaxis de la transformada chars-to-composite-char será:

```
[CHARS-TO-COMPOSITE-CHAR]
```

24.14.3 El origen de esta transformada es una cadena de caracteres y el resultado es un compuesto de caracteres únicos.

24.14.4 El compuesto de caracteres únicos es una lista ordenada de los caracteres de la cadena de caracteres origen.

24.14.5 Esta transformada se define como reversible para todos los valores abstractos.

24.15 Transformada bits-to-composite-bits

24.15.1 La transformada bits-to-composite-bits convierte una cadena de bits en un compuesto de cadenas de bits, teniendo cada elemento cadena de bits el mismo tamaño (conocido).

24.15.2 La transformada bits-to-composite-bits utiliza las propiedades de codificación siguientes:

```
&bits-to-composite-bits-unit      Unit (1..MAX) DEFAULT bit
```

24.15.3 La sintaxis de la transformada bits-to-composite-bits será:

```
[BITS-TO-COMPOSITE-BITS
     [UNIT &bits-to-composite-bits-unit]]
```

24.15.4 La definición del tipo utilizado en la transformada bits-to-composite-bits es como sigue:

```
Unit ::= INTEGER
     {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
      dword32(32)} (0..256) --  (véase 21.1)
```

24.15.5 El origen de esta transformada es una cadena de bits y el resultado es un compuesto de cadenas de bits de tamaño "**UNIT**".

24.15.6 El compuesto de cadenas de bits de tamaño "**UNIT**" es una lista ordenada de cadenas de bits cada una de ellas de tamaño "**UNIT**". La primera cadena de bits del compuesto está formada por los primeros "**UNIT**" bits de la cadena de bits origen. La segunda está formada por los "**UNIT**" bits siguientes, y así sucesivamente. Si la cadena de bits origen no es un múltiplo de "**UNIT**" bits, se trata de un error de la especificación ECN o de la aplicación.

24.15.7 Esta transformada se define como reversible para todos los valores abstractos.

24.16 Transformada octects-to-composite-bits

24.16.1 La transformada octects-to-composite-bits convierte una cadena de octetos en un compuesto de cadenas de bits de 8 bits de tamaño.

24.16.2 La sintaxis de la transformada octects-to-composite-bits será:

```
[OCTETS-TO-COMPOSITE-BITS]
```

24.16.3 El origen de esta transformada es una cadena de octetos y el resultado es un compuesto de cadenas de bits de 8 bits de tamaño.

24.16.4 El compuesto de cadenas de bits de 8 bits de tamaño es una lista ordenada de las cadenas de bits correspondientes a los octetos de la cadena de octetos origen.

24.16.5 Esta transformada se define como reversible para todos los valores abstractos.

24.17 Transformada composite-char-to-chars

24.17.1 La transformada composite-char-to-chars convierte un compuesto de caracteres únicos en una cadena de caracteres.

24.17.2 La sintaxis de la transformada composite-char-to-chars será:

```
[COMPOSITE-CHAR-TO-CHARS]
```

24.17.3 El origen de esta transformada es un compuesto de caracteres únicos y el resultado es una cadena de caracteres.

24.17.4 La cadena de caracteres se forma a partir de la lista ordenada de caracteres presentes en el compuesto de caracteres únicos (origen).

24.17.5 Esta transformada se define como reversible para todos los valores abstractos.

24.18 Transformada composite-bits-to-bits

24.18.1 La transformada composite-bits-to-bits convierte un compuesto de cadenas de bits de tamaño unitario conocido en una cadena de bits.

24.18.2 La sintaxis de la transformada composite-bits-to-bits será:

```
[COMPOSITE-BITS-TO-BITS]
```

24.18.3 El origen de esta transformada es un compuesto de cadenas de bits y el resultado es una cadena de bits.

24.18.4 La cadena de bits se forma a partir de la lista ordenada de cadenas de bits presentes en el compuesto de cadenas de bits (origen).

24.18.5 Esta transformada se define como reversible para todos los valores abstractos. La cadena de bits resultante no es autodelimitante.

NOTA – Esta transformada es reversible porque las unidades utilizadas en su generación están especificadas en la transformada que produjo el compuesto de cadenas de bits y están asociadas a ese compuesto.

24.19 Transformada composite-bits-to-octets

24.19.1 La transformada composite-bits-to-octets convierte un compuesto de cadenas de bits de tamaño unitario 8 en una cadena de octetos. Si esto se aplica a un compuesto de cadenas de bits cuyo tamaño unitario no es 8, se trata de un error de la especificación ECN.

24.19.2 La sintaxis de la transformada composite-bits-to-octets será:

```
[COMPOSITE-BITS-TO-OCTETS]
```

24.19.3 El origen de esta transformada es un compuesto de cadenas de bits y el resultado es una cadena de octetos.

24.19.4 La cadena de octetos se forma a partir de la lista ordenada de cadenas de bits presentes en el compuesto de cadenas de bits (origen).

24.19.5 Esta transformada se define como reversible para todos los valores abstractos.

25 Codificaciones completas y la clase #OUTER

Si no hay ningún objeto de codificación de la clase **#OUTER** en el conjunto de objetos de codificación combinados que se aplica a un tipo en el ELM, el codificador y el decodificador supondrán la presencia de un objeto de codificación de esta clase en el que todas las propiedades de codificación tienen su valor por defecto.

25.1 Propiedades de codificación, sintaxis y objetivo para la clase #OUTER

25.1.1 La sintaxis de la definición de objetos de codificación de la clase **#OUTER** se define como sigue:

```
#OUTER ::= ENCODING-CLASS {

      -- Alignment point
      &alignment-point                  ENUMERATED
                                        {unchanged, reset } DEFAULT reset,
      -- Padding
      &post-padding-unit                Unit (1..MAX) DEFAULT octet,
      &post-padding                     Padding DEFAULT zero,
      &post-padding-pattern             Non-Null-Pattern (ALL EXCEPT different:any)
                                        DEFAULT bits:'0'B,

      -- Bit reversal specification (véase 22.12)
      &bit-reversal                     ReversalSpecification
                                        DEFAULT no-reversal,

      -- Added bits action
      &added-bits                       ENUMERATED
                                        {hard-error, signal-application,
                                        silently-ignore, next-value}
                                        DEFAULT hard-error

} WITH SYNTAX {

      [ALIGNMENT &alignment-point]
      [PADDING
            [MULTIPLE OF &post-padding-unit]
            [POST-PADDING &post-padding
                  [PATTERN &post-padding-pattern]]]
      [BIT-REVERSAL &bit-reversal]
      [ADDED BITS DECODING         &added-bits]

}
```

25.1.2 Las definiciones de los tipos utilizados en la especificación **#OUTER** son como sigue:

```
Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
       dword32(32)} (0..256) --  (véase 21.1)

Padding ::= ENUMERATED {zero, one, pattern, encoder-option} -- (véase 21.9)

Non-Null-Pattern ::= Pattern
      (ALL EXCEPT (bits:''B | octets:''H | char8:"" | char16:"" |
                   char32:"")) -- (véase 21.10.2)
```

25.1.3 Los objetos de codificación de la clase **#OUTER** especifican acciones de codificador y decodificador en relación con la codificación total de un tipo que es codificado por:

a) la aplicación de una codificación en el ELM; o

b) la aplicación de una codificación al tipo contenido.

25.1.4 Se pueden hacer tres especificaciones independientes (véase 25.1.5 a 25.1.7).

25.1.5 La especificación "**ALIGNMENT**", que sólo es aplicable para un tipo contenido y determina si el punto de alineación se ha de reponer en la cabeza del contenedor o si es el mismo que se utiliza para la codificación del contenedor.

25.1.6 La especificación "**PADDING**", que determina que se ha de rellenar toda la codificación con bits finales para hacer que el número desde el punto de alineación sea un múltiplo entero de alguna unidad.

25.1.7 La especificación "**ADDED BITS DECODING**", que sólo es aplicable a decodificadores y determina la acción que se ha de efectuar si hay más bits en la PDU después de haber completado la decodificación de acuerdo con las especificaciones de la codificación.

NOTA – Esta disposición tiene por objeto principalmente proporcionar un mecanismo de extensibilidad sencillo sin utilizar el marcador de extensibilidad ASN.1. Se prevé que en una versión posterior de la presente Recomendación | Norma Internacional se insista en el soporte de la extensibilidad.

25.1.8 "**ALIGNMENT**", "**PADDING**" y "**ADDED BITS DECODING**" toman sus valores por defecto si no están fijados o si no hay ningún objeto de codificación de la clase **#OUTER** en el conjunto de objetos de codificación combinados.

NOTA – Los valores por defecto son los utilizados por el objeto de codificación de la clase **#OUTER** para las reglas de codificación no alineada básica PER.

25.2 Acciones de codificador para #OUTER

25.2.1 Si "**ALIGNMENT**" es "**unchanged**", el punto de alineación utilizado al codificar un tipo contenido será el punto de alineación utilizado al codificar el contenedor.

25.2.2 Si "**ALIGNMENT**" es "**reset**", el punto de alineación utilizado al codificar un tipo contenido será el comienzo de la codificación de ese tipo.

25.2.3 Si se fija "**PADDING**", el codificador añadirá bits de acuerdo con el valor de "**PADDING**" y "**PATTERN**" para hacer que el número de bits desde el punto de alineación sea un múltiplo de "**MULTIPLE OF**" unidades. "**PATTERN**" será reproducido y truncado según se necesite.

25.2.4 El codificador diagnosticará un error de la especificación ECN o de la aplicación si se trata de la codificación de un tipo en una constricción, relativa al contenido, impuesta a una cadena de octetos y la codificación (después de todas las acciones "**PADDING**" especificadas) no es un múltiplo entero de ocho bits.

25.2.5 Si se fija inversión de bits, se aplicarán las acciones de codificador especificadas en 22.12 utilizando el valor de "**MULTIPLE OF**" especificado para (o tomado por defecto en) "**PADDING**".

25.2.6 El codificador ignorará "**ADDED BITS DECODING**".

25.3 Acciones de decodificador para #OUTER

25.3.1 Si se fija inversión de bits, se aplicarán las acciones de decodificador especificadas en 22.12 utilizando el valor de "**MULTIPLE OF**" especificado para (o tomado por defecto en) "**PADDING**".

25.3.2 Si "**ALIGNMENT**" es "**unchanged**", el punto de alineación utilizado al codificar un tipo contenido será el punto de alineación utilizado al codificar el contenedor.

25.3.3 Si "**ALIGNMENT**" es "**reset**", el punto de alineación utilizado al codificar un tipo contenido será el comienzo de la codificación de ese tipo.

25.3.4 El decodificador determinará los bits añadidos por "**PADDING**" (en su caso), e ignorará en silencio los bits añadidos, cualquiera que sea su valor.

25.3.5 Si la PDU (o el contenedor de un tipo contenido) contiene más bits después del final de la codificación, el decodificador efectuará las acciones siguientes:

a) si "**ADDED BITS DECODING**" es "**hard-error**": diagnosticará un error de codificador;

b) si "**ADDED BITS DECODING**" es "**signal-application**": ignorará todos los demás bits y señalará a la aplicación que puede haber ampliaciones críticas del protocolo;

c) si "**ADDED BITS DECODING**" es "**silently-ignore**": ignorará todos los demás bits;

d) si "**ADDED BITS DECODING**" es "**next-value**": dejará de codificar y esperará a que la aplicación inicie la decodificación de un valor nuevo a partir de los bits restantes.

Anexo A

Addendum a la Rec. UIT-T X.680 | ISO/CEI 8824-1

(Este anexo es parte integrante de la presente Recomendación | Norma Internacional)

Este anexo especifica las modificaciones que se han de aplicar cuando en la presente Recomendación | Norma Internacional se haga referencia a producciones y/o cláusulas de la Rec. UIT-T X.680 | ISO/CEI 8824-1.

A.1 Cláusulas de exportaciones e importaciones

Las producciones "AssignedIdentifier", "Symbol" y "Reference" de 12.1, así como las subcláusulas 12.12 y 12.15 de la Recomendación UIT-T X.680 | ISO/CEI 8824-1 se modifican como sigue:

12.1 **AssignedIdentifier ::= DefinitiveIdentifier |**
empty

Symbol ::=
Reference
| **BuiltinEncodingClassReference**
| **ParameterizedReference**

Reference ::=
encodingclassreference
| **ExternalEncodingClassReference**
| **encodingobjectreference**
| **encodingobjectsetreference**

NOTA 1 – La producción "AssignedIdentifier" se cambia porque las "valuereference" no pueden ser definidas ni importadas en el ELM o los módulos EDM.

NOTA 2 – "BuiltinEncodingClassReference" sólo se puede utilizar como un "Symbol" en la cláusula de importaciones. La utilización de la producción "ExternalEncodingClassReference" en "Reference" se explica en 14.11.

12.12 Cuando se seleccione la alternativa "SymbolsExported" de "Exports", cada "Symbol" de "SymbolsExported" satisfará una, y solamente una, de las condiciones siguientes:

a) se define en el módulo del que se exporta; o

b) aparece exactamente una vez en la alternativa "SymbolsImported" de "Imports" del módulo del que se exporta

12.15 Cuando se seleccione la alternativa "SymbolsImported" de "Imports":

a) Cada "Symbol" de "SymbolsFromModule":

1) se definirá en el cuerpo del módulo designado por la "GlobalModuleReference" en "SymbolsFromModule"; o

2) estará presente exactamente una vez en la cláusula de importaciones del módulo designado por la "GlobalModuleReference" en "SymbolsFromModule".

NOTA – Esto no impide que el mismo nombre de símbolo definido en dos módulos diferentes sea importado en otro módulo. Sin embargo, si el mismo nombre de "Symbol" aparece más de una vez en la cláusula de importaciones del módulo "A", ese nombre de "Symbol" no puede ser exportado desde "A" para ser importado en otro módulo "B".

b) Todos los "SymbolsFromModule" de la "SymbolsFromModuleList" incluirán ocurrencias de la "GlobalModuleReference" de tal manera que:

i) todas sus "modulereference" sean diferentes entre si (tanto si se trata de módulos ASN.1 como EDM) y diferentes de la "modulereference" asociada al módulo referenciador; y

ii) el "AssignedIdentifier", cuando no esté vacío, denote valores de identificador de objeto que son todos diferentes entre si y diferentes del valor del identificador de objeto (si lo hay) asociado al módulo referenciador.

A.2 Adición de "REFERENCE"

NOTA – Esta modificación se introduce únicamente a efectos de la cláusula 23.

La producción "Type" de 16.1 de la Rec. UIT-T X.680 | ISO/CEI 8824-1 se modifica como sigue:

```
Type ::=
        BuiltinType
      | ReferencedType
      | ConstrainedType
      | REFERENCE
```

A.3 Notación de valores cadena de caracteres

La producción "CharsDefn" de 37.8 de la Rec. UIT-T X.680 | ISO/CEI 8824-1 se modifica como sigue:

```
CharsDefn ::=
        cstring
      | Quadruple
      | Tuple
      | AbsoluteCharReference

AbsoluteCharReference ::=
        ModuleIdentifier
        "."
        valuereference
```

La "AbsoluteCharReference" es un nombre totalmente cualificado que hace referencia a un valor cadena de caracteres (del tipo "**IA5String**" o "**BMPString**") definido en el "**ASN1-CHARACTER-MODULE**" (véase 38.1 de la Rec. UIT-T X.680 | ISO/CEI 8824-1).

Anexo B

Addendum a la Rec. UIT-T X.681 | ISO/CEI 8824-2

(Este anexo es parte integrante de la presente Recomendación | Norma Internacional)

Este anexo especifica las modificaciones que se han de aplicar cuando en la presente Recomendación | Norma Internacional se haga referencia a producciones y/o cláusulas de la Rec. UIT-T X.681 | ISO/CEI 8824-2.

B.1 Definiciones

Se añaden las definiciones siguientes a 3.4 de la Rec. UIT-T X.681 | ISO/CEI 8824-2:

campo de clase de codificación: Campo que contiene una clase de codificación cualquiera.

tipo de campo de clase de codificación: Tipo especificado por referencia a algún campo de un tipo de una clase de objetos de codificación.

campo de objeto de codificación: Campo que contiene un objeto de codificación de alguna clase de codificación especificada. Ese campo es de clase fija o de clase variable. En el primer caso, la clase del objeto de codificación está fijada por la especificación del campo. En el segundo caso, la clase del objeto de codificación está contenida en algún campo de clase de codificación (específico) del mismo objeto de codificación.

campo de conjunto de objetos de codificación: Campo que contiene un conjunto de objetos de codificación de alguna clase de codificación especificada.

campo de lista de valores de tipo fijo ordenada: Campo que contiene una lista ordenada (posiblemente vacía) de valores de algún tipo especificado.

campo de lista de objetos de codificación ordenada: Campo que contiene una lista no vacía ordenada de objetos de codificación de alguna clase de codificación especificada.

campo de referencia: Campo que contiene una referencia a un campo de estructura de codificación (véase también 17.5.15).

B.2 Elementos de léxico adicionales

NOTA – Esta modificación se introduce únicamente a efectos de la cláusula 23.

Se añaden las definiciones siguientes a la cláusula 7 de la Rec. UIT-T X.681 | ISO/CEI 8824-2:

B.2.1 Referencias de campo de lista de valores ordenada

Nombre del elemento: orderedvaluelistfieldreference

Un "orderedvaluelistfieldreference" constará de un signo de "**y**" comercial ("**&**") seguido inmediatamente por una secuencia caracteres según lo especificado para una "typereference" en 11.2 de la Rec. UIT-T X.680 | ISO/CEI 8824-1.

B.2.2 Referencias de campo de lista de objetos de codificación ordenada

Nombre del elemento: orderedencodingobjectlistfieldreference

Una "orderedencodingobjectlistfieldreference" constará de un signo de "**y**" comercial ("**&**") seguido inmediatamente por una secuencia de caracteres según lo especificado para una "objectsetference" en 7.3 de la Rec. UIT-T X.681 | ISO/CEI 8824-2.

B.2.3 Referencias de campo de clase de codificación

Nombre del elemento: encodingclassfieldreference

Una "encodingclassfieldreference" constará de un signo de "**y**" comercial ("**&**") seguido inmediatamente por una secuencia de caracteres según lo especificado para una "encodingclassreference" en 8.3.

B.3 Adición de "ENCODING-CLASS"

NOTA – Esta modificación se introduce únicamente a efectos de la cláusula 23.

Sustituir la palabra reservada "CLASS" por "ENCODING-CLASS" en 9.3 de la Rec. UIT-T X.681 | ISO/CEI 8824-2.

B.4 Adiciones de FieldSpec

NOTA – Esta modificación se introduce únicamente a efectos de la cláusula 23.

La cláusula 9.4 de la Rec. UIT-T X.681 | ISO/CEI 8824-2 se modifica como sigue:

```
FieldSpec ::=
            FixedTypeValueFieldSpec
          | FixedTypeValueSetFieldSpec
          | FixedTypeOrderedValueListFieldSpec
          | FixedClassEncodingObjectFieldSpec
          | VariableClassEncodingObjectFieldSpec
          | FixedClassEncodingObjectSetFieldSpec
          | FixedClassOrderedEncodingObjectListFieldSpec
          | EncodingClassFieldSpec
```

B.5 Especificación de campo de lista de valores de tipo fijo ordenada

NOTA – Esta modificación se introduce únicamente a efectos de la cláusula 23.

Una "FixedTypeOrderedValueListFieldSpec" especifica que el campo es un campo de lista de valores ordenados de tipo fijo (véase B.1 de la presente Recomendación | Norma Internacional):

```
FixedTypeOrderedValueListFieldSpec ::=
        orderedvaluelistfieldreference
        DefinedType
        ORDERED
        FixedTypeOrderedValueListOptionalitySpec ?

FixedTypeOrderedValueListOptionalitySpec ::= OPTIONAL | DEFAULT OrderedValueList
```

El nombre del campo es "orderedvaluelistfieldreference". El "DefinedType" hace referencia al tipo de valores contenidos en este campo. La "FixedTypeOrderedValueListOptionalitySpec", si está presente, especifica que el campo puede no estar especificado en una definición de objeto de codificación o, en el caso "**DEFAULT**", la omisión produce la "OrderedValueList" siguiente (véase 25.3 de la Rec. UIT-T X.680 | ISO/CEI 8824-1), todos cuyos valores serán "DefinedType".

B.6 Especificación de campo de objetos de codificación de clase fija

NOTA – Esta modificación se introduce únicamente a efectos de la cláusula 23.

Una "FixedClassEncodingObjectFieldSpec" especifica que el campo es un campo de objetos de codificación de clase fija (véase B.1 de la presente Recomendación | Norma Internacional):

```
FixedClassEncodingObjectFieldSpec ::=
        objectfieldreference
        DefinedOrBuiltinEncodingClass
        EncodingObjectOptionalitySpec?

EncodingObjectOptionalitySpec ::= OPTIONAL | DEFAULT EncodingObject
```

El nombre del campo es "objectfieldreference". La "DefinedOrBuiltinEncodingClass" hace referencia a la clase de objetos de codificación contenida en el campo (que puede ser la "EncodingClass" que se define). La "EncodingObjectOptionalitySpec", si está presente, especifica que el campo puede no estar especificado en una definición de objeto de codificación o, en el caso "**DEFAULT**", la omisión produce el "EncodingObject" siguiente (véase 17.1.5 de la presente Recomendación | Norma Internacional) que será de la "DefinedOrBuiltinEncodingClass".

B.7 Especificación de campo de objetos de codificación de clase variable

Una "VariableClassEncodingObjectFieldSpec" especifica que el campo es un campo de objetos de codificación de clase variable (véase B.1 de la presente Recomendación | Norma Internacional):

```
VariableClassEncodingObjectFieldSpec ::=
        objectfieldreference
        encodingclassfieldreference
        EncodingObjectOptionalitySpec?
```

El nombre del campo es "objectfieldreference". La "encodingclassfieldreference" hace referencia a un campo de clase de codificación de la clase de codificación que se especifica. La "EncodingObjectOptionalitySpec", si está presente, especifica que el objeto de codificación puede ser omitido en una definición objeto de codificación o, en el caso "**DEFAULT**", la omisión produce el "EncodingObject" siguiente. La "EncodingObjectOptionalitySpec" será tal que:

a) si el campo de tipo designado por la "encodingclassfieldreference" tiene una "EncodingClassOptionalitySpec" de "**OPTIONAL**", la "EncodingClassOptionalitySpec" será también "**OPTIONAL**"; y

b) si la "EncodingObjectOptionalitySpec" es "**DEFAULT** EncodingObject", el campo de clase de codificación designado por la "encodingclassfieldreference" tendrá una EncodingClassOptionalitySpec" de "**DEFAULT** DefinedOrBuiltinEncodingClass", y el "EncodingObject" será un objeto de codificación de esa clase.

B.8 Especificación de campo de conjunto de objetos de codificación de clase fija

NOTA – Esta modificación se introduce únicamente a efectos de la cláusula 23.

Una "FixedClassEncodingObjectSetFieldSpec" especifica que el campo es un campo de conjunto de objetos de codificación de clase fija (véase B.1 de la presente Recomendación | Norma Internacional):

```
FixedClassEncodingObjectSetFieldSpec ::=
        objectsetfieldreference
        DefinedOrBuiltinEncodingClass
        EncodingObjectSetOptionalitySpec?

EncodingObjectSetOptionalitySpec ::= OPTIONAL | DEFAULT EncodingObjectSet
```

El nombre del campo es "objectsetfieldreference". La "DefinedOrBuiltinEncodingClass" hace referencia a la clase de los objetos de codificación contenidos en el campo. La "EncodingObjectSetOptionalitySpec", si está presente, especifica que el campo puede no estar especificado en una definición de objeto de codificación o, en el caso "**DEFAULT**", la omisión produce el "EncodingObjectSet" siguiente (véase la cláusula 18), todos cuyos objetos serán de la "DefinedOrBuiltinEncodingClass".

B.9 Especificación de campo de lista de objetos de codificación de clase fija ordenada

NOTA – Esta modificación se introduce únicamente a efectos de la cláusula 23.

Una "FixedClassOrderedEncodingObjectListFieldSpec" especifica que el campo es un campo de lista de objetos de codificación de clase fija ordenada (véase B.1 de la presente Recomendación | Norma Internacional):

```
FixedClassOrderedEncodingObjectListFieldSpec ::=
        orderedencodingobjectlistfieldreference
        DefinedOrBuiltinEncodingClass
        ORDERED
        OrderedEncodingObjectListOptionalitySpec?

OrderedEncodingObjectListOptionalitySpec ::= OPTIONAL | DEFAULT OrderedEncodingObjectList
```

El nombre del campo es "orderedencodingobjectlistfieldreference". La "DefinedOrBuiltinEncodingClass" hace referencia a la clase de los objetos de codificación contenidos en el campo. La "OrderedEncodingObjectListOptionalitySpec", si está presente, especifica que el campo puede no estar especificado en una definición de objeto de codificación, o , en el caso "**DEFAULT**", la omisión produce la "OrderedEncodingObjectList" siguiente (véase B.11 de la presente Recomendación | Norma Internacional), todos cuyos objetos serán de la "DefinedOrBuiltinEncodingClass".

B.10 Especificación de campo de clase de codificación

NOTA – Esta modificación se introduce únicamente a efectos de la cláusula 23.

Una "EncodingClassFieldSpec" especifica que el campo es un campo de clase de codificación (véase B.1 de la presente Recomendación | Norma Internacional):

```
EncodingClassFieldSpec ::=
        encodingclassfieldreference
        EncodingClassOptionalitySpec?

EncodingClassOptionalitySpec ::= OPTIONAL | DEFAULT DefinedOrBuiltinEncodingClass
```

El nombre del campo es "encodingclassfieldreference". Si la "EncodingClassOptionalitySpec" está ausente, es preciso que todas las definiciones de objetos de codificación de esa clase incluyan la especificación de una clase de codificación de ese campo. Si "**OPTIONAL**" está presente, se puede dejar el campo sin definir. Si "**DEFAULT**" está presente, la "DefinedOrBuiltinEncodingClass" proporciona la fijación por defecto del campo en el caso de que se omita en una definición.

B.11 Notación de lista de valores ordenada

OrderedValueList ::= "{" Value "," + "}"

La "OrderedValueList" es una lista ordenada de uno o más valores del tipo gobernante. Se utiliza cuando la aplicación aplica la semántica al orden de los valores de la lista.

NOTA – Una lista de valores sólo puede ser especificada mediante notación en línea (que es gobernada por un campo de tipo, un campo de conjunto de valores de tipo fijo o un campo de lista de valores de tipo fijo ordenada).

B.12 Notación de lista de objetos de codificación ordenada

OrderedEncodingObjectList ::= "{" EncodingObject "," + "}"

La "OrderedEncodingObjectList" es una lista ordenada de uno o más objetos de codificación de la clase gobernante. Se utiliza cuando la aplicación aplica la semántica al orden de los objetos de codificación de la lista.

Ejemplo: Una lista de objetos de codificación **#TRANSFORM** se aplica en el orden establecido.

NOTA – Las restricciones siguientes provienen de textos normativos y producciones BNF: una lista de objetos de codificación ordenada sólo puede ser especificada por notación en línea (que es gobernada por un campo de lista de objetos de codificación ordenada); los objetos de codificación de esa lista pueden ser especificados utilizando un nombre de referencia o una notación en línea; el gobernador no puede ser **#ENCODINGS**.

B.13 Nombres de campo primitivos

La cláusula 9.13 de la Rec. UIT-T X.681 | ISO/CEI 8824-2 se modifica como sigue:

9.13 El constructivo "PrimitiveFieldName" se utiliza para identificar un campo relativo a la clase de codificación que contiene su especificación:

PrimitiveFieldName ::=
valuefieldreference
| **valuesetfieldreference**
| **orderedvaluelistfieldreference**

B.14 Palabras reservadas adicionales

Las cláusulas 10.6 y 10.7 de la Rec. UIT-T X.681 | ISO/CEI 8824-2 se modifican como sigue:

10.6 Un elemento de léxico "word" utilizado como "Literal" no puede ser uno de los siguientes:

BEGIN	**MINUS-INFINITY**	**PER-CANONICAL-UNALIGNED**
BER	**NON-ECN-BEGIN**	**PLUS-INFINITY**
CER	**NULL**	**TRUE**
DER	**OPTIONS**	**UNION**
ENCODE	**OUTER**	**USE**
ENCODE-DECODE	**PER-BASIC-ALIGNED**	**USE-SET**
END	**PER-BASIC-UNALIGNED**	
FALSE	**PER-CANONICAL-UNALIGNED**	

NOTA – Esta lista comprende solamente aquellas palabras reservadas ASN.1 que pueden aparecer como primer elemento de un "Value", "EncodingObject" o "EncodingObjectSet", y también la palabra reservada "**END**". La utilización de otras palabras reservadas ECN no provoca ninguna ambigüedad y está permitida. Cuando la sintaxis definida se utiliza en un entorno en el que una "word" es también una "encodingobjectsetreference", tiene precedencia la utilización como "word".

10.7 Un "Literal" especifica la inclusión efectiva de ese "Literal", que ha de ser una "word", en esa posición de la sintaxis definida.

B.15 Definición de objetos de codificación

Se elimina la restricción impuesta por 10.12 d) de la Rec. UIT-T X.681 | ISO/CEI 8824-2.

NOTA – Esto afecta a la sintaxis definida de la definición de objetos de codificación de algunas clases (véanse las cláusulas 23 y 24). Significa, por ejemplo, que para una sintaxis definida tal como:

```
[BOOL-TO-INT [AS &bool-to-int]]
```

el usuario está autorizado a escribir:

BOOL-TO-INT

cuando define un objeto de codificación de esta clase. En tal caso, se utiliza el valor "**DEFAULT**" con el parámetro "**&bool-to-int**" (es decir, "**false-zero**") en la definición de la transformada "**BOOL-TO-INT**".

B.16 Adiciones a "Setting"

Se modifica la cláusula 11.7 de la Rec. UIT-T X.681 | ISO/CEI 8824-2 como sigue:

11.7 Un "Setting" especifica la fijación de algún campo dentro de un objeto de codificación que se define:

Setting ::=
Value
| **ValueSet**
| **OrderedValueList**
| **EncodingObject**
| **EncodingObjectSet**
| **OrderedEncodingObjectList**
| **DefinedOrBuiltinEncodingClass**
| **OUTER**

Si el campo es:

a) un campo de valor, se seleccionará la alternativa "Value";

b) un campo de conjunto de valores de tipo fijo, se seleccionará la alternativa "ValueSet";

c) un campo de lista de valores de tipo fijo ordenada, se seleccionará la alternativa "OrderedValueList";

d) un campo de objeto de codificación, se seleccionará la alternativa "EncodingObject";

e) un campo de conjunto de objetos de codificación, se seleccionará la alternativa "EncodingObjectSet";

f) un campo de lista de objetos de codificación ordenada, se seleccionará la alternativa "OrderedEncodingObjectList";

g) un campo de clase de codificación, se seleccionará la alternativa "DefinedOrBuiltinEncodingClass";

h) un campo de referencia, se seleccionará la alternativa "Value" o la alternativa "**OUTER**".

Para un campo de referencia especificado utilizando la sintaxis de las cláusulas 20 a 25, el "Value" será un parámetro ficticio. Se puede utilizar "**OUTER**" siempre que se requiera una referencia e identifique a un contenedor que es la codificación completa.

NOTA – La fijación se restringe aún más según se describe en 9.5 a 9.12, 11.8 y 11.9 de la Rec. UIT-T X.681 | ISO/CEI 8824-2.

B.17 Tipo de campo de clase de codificación

El tipo que es referenciado por esta notación depende de la categoría del nombre de campo. Las cláusulas B.17.2 a B.17.4 especifican el tipo al que se hace referencia para las diferentes categorías de nombre de campo.

B.17.1 La notación de un tipo de campo de clase de codificación será "EncodingClassFieldType":

EncodingClassFieldType ::=
DefinedOrBuiltinEncodingClass
"."
FieldName

donde el "FieldName" es tal como se especifica en 9.14 de la Rec. UIT-T X.681 | ISO/CEI 8824-2, relativa a las clases de codificación identificadas por la "DefinedOrBuiltinEncodingClass".

B.17.2 Para un valor de tipo fijo, un campo de conjunto de valores de tipo fijo o un campo de lista de valores de tipo fijo ordenada, la notación denota el "Type" que aparece en la especificación de ese campo de la definición de la clase de objetos de codificación.

B.17.3 Esta notación no se permite si el campo es un objeto de codificación, un conjunto de objetos de codificación o una lista de objetos de codificación ordenada.

B.17.4 La notación para definir un valor de este tipo será "FixedTypeFieldVal", definida en 14.6 de la Rec. UIT-T X.681 | ISO/CEI 8824-2.

Anexo C

Addendum a la Rec. UIT-T X.683 | ISO/CEI 8824-4

(Este anexo es parte integrante de la presente Recomendación | Norma Internacional)

Este anexo especifica las modificaciones que se han de aplicar cuando en la presente Recomendación | Norma Internacional se haga referencia a producciones y/o cláusulas de la Rec. UIT-T X.683 | ISO/CEI 8824-4.

C.1 Asignaciones parametrizadas

*Las cláusulas 8.1 y 8.3 de la Rec. UIT-T X.683 | ISO/CEI 8824-4 se modifican como sigu*e:

8.1 Hay declaraciones de asignación parametrizada correspondientes a las declaraciones de asignación especificadas en esta Recomendación | Norma Internacional. El constructivo "ParameterizedAssignment" es:

ParameterizedAssignment ::=
ParameterizedEncodingObjectAssignment
| **ParameterizedEncodingClassAssignment**
| **ParameterizedEncodingObjectSetAssignment**

8.3 **ParameterList ::= "{<" Parameter "," + ">}"**

Governor ::=
EncodingClassFieldType
| **REFERENCE**
| **DefinedOrBuiltinEncodingClass**
| **#ENCODINGS**

Una "DummyReference" en "Parameter" puede significar:

a) una clase de codificación, en cuyo caso no habrá "ParamGovernor";

b) un valor ASN.1, conjunto de valores o lista de valores de tipo fijo ordenada, en cuyo caso estará presente el "ParamGovernor" como un "Governor" que es un tipo extraído de una clase de codificación ("EncodingClassFieldType");

c) un "identifier", en cuyo caso estará presente el "ParamGovernor" como un "Governor" que es "**REFERENCE**";

d) un objeto de codificación, o una lista de objetos de codificación ordenada, en cuyo caso estará presente el "ParamGovernor" como un "Governor" que es una clase de codificación ("DefinedOrBuiltinEncodingClass");

e) un conjunto de objetos de codificación, en cuyo caso estará presente el "ParamGovernor" como un "Governor" que es "**#ENCODINGS**".

NOTA – En la ECN no se permiten los "DumyGovernor".

C.2 Asignaciones de codificaciones parametrizadas

Se añaden a 8.2 de la Rec. UIT-T X.683 | ISO/CEI 8824-4 las producciones siguientes:

ParameterizedEncodingClassAssignment ::=
encodingclassreference
ParameterList
"::="
EncodingClass

ParameterizedEncodingObjectAssignment ::=
encodingobjectreference
ParameterList
DefinedOrBuiltinEncodingClass
"::="
EncodingObject

ParameterizedEncodingObjectSetAssignment ::=
encodingobjectsetreference
ParameterList
#ENCODINGS
"::="
EncodingObjectSet

La cláusula 8.4 de la Rec. UIT-T X.683 | ISO/CEI 8824-4 se modifica como sigue:

8.4 El alcance de una "DummyReference" que aparece en una "ParameterList" es la propia "ParameterList", junto con la parte de la "ParameterizedList que sigue al símbolo **"::="**. En el caso de una "ParameterizedEncodingObjectAssignment", el alcance se amplía a la "DefinedOrBuiltinEncodingClass" que precede al símbolo **"::="**. La "DummyReference" oculta cualquier otra "Reference" con el mismo nombre en ese mismo ámbito de aplicación.

NOTA – Lo previsto es que el caso especial de "ParameterizedEncodingObjectClass" se utilice en común con las cláusulas de redenominaciones (véase D.3.3.3). Permite escribir una asignación como la siguiente, en la que el parámetro ficticio "**`#Any-Class`**" del objeto de codificación "**`new-component-encoding`**" se utiliza como el parámetro real de la clase de codificación "**`#New-component`**":

```
new-component-encoding {< #Any-class >} #New-component {< #Any-class >} ::=
        { -- encoding object definition -- }
```

C.3 Definiciones de referencias parametrizadas

La producción "ParameterizedReference" de 9.1 de la Rec. UIT-T X.683 | ISO/CEI 8824-4 se modifica como sigue:

ParameterizedReference ::=
    Reference
| **Reference "{<" ">}"**

Las producciones siguientes se añaden a 9.2 de la Rec. UIT-T X.683 | ISO/CEI 8824-4:

ParameterizedEncodingObject ::=
    SimpleDefinedEncodingObject
    ActualParameterList

SimpleDefinedEncodingObject ::=
    ExternalEncodingObjectReference
| **encodingobjectreference**

ParameterizedEncodingObjectSet ::=
    SimpleDefinedEncodingObjectSet
    ActualParameterList

SimpleDefinedEncodingObjectSet ::=
    ExternalEncodingObjectSetReference
| **encodingobjectsetreference**

ParameterizedEncodingClass ::=
    SimpleDefinedEncodingClass
    ActualParameterList

SimpleDefinedEncodingClass ::=
    ExternalEncodingClassReference
| **encodingclassreference**

C.4 Lista de parámetros reales

La cláusula 9.5 de la Rec. UIT-T X.683 | ISO/CEI 8824-4 se modifica como sigue:

9.5 La "ActualParameterList" es:

ActualParameterList ::=
    "{<" ActualParameter "," + ">}"

ActualParameter ::=
        Value
    | **ValueSet**
    | **OrderedValueList**
    | **DefinedOrBuiltinEncodingClass**
    | **EncodingObject**
    | **EncodingObjectSet**
    | **OrderedEncodingObjectList**
    | **identifier**
    | **STRUCTURE**
    | **OUTER**

Si el parámetro ficticio correspondiente es:

a) un valor, se seleccionará la alternativa "Value";

b) un conjunto de valores, se seleccionará la alternativa "ValueSet";

c) una lista de valores de tipo fijo ordenada, se seleccionará la alternativa "OrderedValueList";

d) una clase de codificación, se seleccionará la alternativa "DefinedOrBultinEncodingClass";

e) un objeto de codificación, se seleccionará la alternativa "EncodingObject";

f) un conjunto de objetos de codificación, se seleccionará la alternativa "EncodingObjectSet";

g) una lista de objetos de codificación ordenada, se seleccionará la alternativa "OrderedEncodingObjectList";

h) una referencia, se seleccionará la alternativa "**STRUCTURE**" o la alternativa "**OUTER**".

"**STRUCTURE**" sólo se seleccionará cuando el parámetro real se utilice como se especifica en 17.5.15. "**OUTER**" se puede utilizar siempre que se requiera una referencia para identificar a un contenedor, e identifique al contenedor de la codificación completa.

Anexo D

Ejemplos

(Este anexo no es parte integrante de la presente Recomendación | Norma Internacional)

Este anexo contiene ejemplos de la utilización de la ECN. Los ejemplos se dividen en cinco grupos:

- Ejemplos generales, que muestran el aspecto y la conveniencia de las definiciones ECN (D.1).
- Ejemplos de especialización, que muestran cómo se modifican algunas partes de la codificación normalizada. Cada ejemplo tiene una descripción de los requisitos de la codificación y una descripción de la solución seleccionada y posibles soluciones alternativas (D.2).
- Ejemplos de estructuras generadas explícitamente, que muestran la utilización de esas estructuras cuando se emplea la misma codificación especializada varias veces (D.2.15).
- Un ejemplo de protocolo de legado que muestra tres maneras de tratar el problema de la terminación de una "sequence-of" (secuencia de) siguiendo el planteamiento tradicional a base de un bit "more" (un bit "más") (D.3.8).
- Un segundo ejemplo de protocolo de legado, que muestra cómo se construyen definiciones ECN para un protocolo cuyas codificaciones de mensajes han sido especificadas utilizando una notación tabular (D.5).

D.1 Ejemplos generales

Los ejemplos descritos en D.1.1 a D.1.14 forman parte de una especificación ECN completa cuyos módulos ASN.1, EDM y ELM se presentan a grandes rasgos en D.1.15, D.1.16 y D.1.17, y se dan "in extenso" en una copia de este anexo que está disponible en el sitio web citado en el anexo F.

D.1.1 Objeto de codificación de un tipo booleano

D.1.1.1 La asignación ASN.1 es:

```
Married ::= BOOLEAN
```

D.1.1.2 La asignación de objeto de codificación (véase 23.3.1) es:

```
booleanEncoding #BOOLEAN ::= {
        ENCODING-SPACE
                SIZE 1
                MULTIPLE OF bit
         TRUE-PATTERN bits:'1'B
         FALSE-PATTERN bits:'0'B}

marriedEncoding-1 #Married ::= booleanEncoding
```

D.1.1.3 No hay prealineación y el espacio de codificación es de un bit, por lo que "**`Married`**" se codifica como un campo de bits de longitud 1. Los esquemas para valores "**`TRUE`**" y "**`FALSE`**" (en este caso, un solo bit) son '1B' y '0B', respectivamente.

D.1.1.4 Los valores especificados anteriormente son los valores que se fijarían por defecto (véase 23.3.1) si se omitieran las propiedades de codificación correspondientes, por lo que se puede conseguir la misma codificación con menos verbosidad de la manera siguiente:

```
marriedEncoding-2 #Married ::= {
        ENCODING-SPACE
                SIZE 1}
```

D.1.1.5 Esta codificación de un tipo booleano es, por supuesto, la que justamente proporcionan las PER. Otra alternativa consiste en especificar la codificación de un tipo booleano con la sintaxis indicada en 17.3.1 utilizando el objeto de codificación PER.

```
marriedEncoding-3 #Married ::= {
        ENCODE WITH PER-BASIC-UNALIGNED}
```

D.1.1.6 Como muestran estos ejemplos, a menudo hay casos en los que la ECN permite múltiples maneras de definir una codificación. El usuario tiene que decidir qué alternativa utiliza, eligiendo entre la verbosidad que conlleva la declaración explicita de los valores que se pueden tomar por defecto por un lado y la legibilidad y claridad por otro.

D.1.2 Objeto de codificación de un tipo entero

D.1.2.1 Las asignaciones ASN.1 son:

```
EvenPositiveInteger ::= INTEGER (1..MAX) (CONSTRAINED  BY {-- Must be even --})

EvenNegativeInteger ::= INTEGER (MIN..-1) (CONSTRAINED BY {-- Must be even --})
```

D.1.2.2 Las asignaciones de objeto de codificación son:

```
evenPositiveIntegerEncoding #EvenPositiveInteger ::= {
        USE #NonNegativeInt
MAPPING TRANSFORMS {{INT-TO-INT divide:2}}
        WITH PER-BASIC-UNALIGNED}

#NonNegativeInt ::= #INT(0..MAX)

evenNegativeIntegerEncoding #EvenNegativeInteger ::= {
        USE #NonPositiveInt
MAPPING TRANSFORMS {{INT-TO-INT divide:2
                                        -- Nota: -1 / 2 = 0 - véase 24.3.6 -- }}
        WITH PER-BASIC-UNALIGNED}

#NonPositiveInt ::= #INT(MIN..0)
```

D.1.2.3 Un valor par se divide por dos y a continuación se codifica utilizando reglas de codificación PER normalizadas para tipos enteros positivos y negativos.

D.1.3 Otro objeto de codificación de un tipo entero

D.1.3.1 Aquí se supone que existe el requisito de definir un objeto de codificación que codifica un entero en un campo de dos octetos situado en una frontera de octeto.

D.1.3.2 La asignación ASN.1 es:

```
Altitude ::= INTEGER (0..65535)
```

D.1.3.3 La asignación de objeto de codificación (véanse 23.6.1 y 23.7.1) es:

```
integerRightAlignedEncoding #Altitude ::= {
        ENCODING {
            ALIGNED TO NEXT octet
            ENCODING-SPACE
                    SIZE 16}}
```

D.1.4 Objeto de codificación de un tipo entero con huecos

D.1.4.1 La asignación ASN.1 es:

```
IntegerWithHole ::= INTEGER (-256..-1 | 32..1056)
```

D.1.4.2 La asignación de objeto de codificación (véase 19.5.2) es:

```
integerWithHoleEncoding #IntegerWithHole ::= {
        USE #IntFrom0To1280
        MAPPING ORDERED VALUES
        WITH PER-BASIC-UNALIGNED}

#IntFrom0To1280 ::= #INT (0..1280)
```

D.1.4.3 "`IntegerWithHole`" se codifica como un entero positivo. Se establece la correspondencia entre valores de la gama –256...-1 y valores de la gama 0...255 y también entre valores de la gama 32...1056 y valores de la gama 256...1280.

D.1.5 Objeto de codificación más complejo de un tipo entero

D.1.5.1 Las asignaciones ASN.1 son:

```
PositiveInteger ::= INTEGER (1..MAX)

NegativeInteger ::= INTEGER (MIN..-1)
```

D.1.5.2 Las asignaciones de objeto de codificación son:

```
positiveIntegerEncoding #PositiveInteger ::=
        integerEncoding

negativeIntegerEncoding #NegativeInteger ::=
        integerEncoding
```

D.1.5.3 Los valores de los tipos "**`PositiveInteger`**" y "**`NegativeInteger`**" son codificados por el objeto de codificación "**`integerEncoding`**" como un entero positivo o como un entero en complemento de 2, respectivamente. A continuación se define esto, y se dan diferentes codificaciones dependiendo de los límites del tipo al que se aplican.

D.1.5.4 El objeto de codificación "**`integerEncoding`**" aquí definido es muy poderoso, pero muy complejo. Contiene cinco objetos de codificación de la clase **`#CONDITIONAL-INT`**; todos ellos definen una codificación con **`alineación de octetos`**. Cuando los valores enteros que se codifican están limitados, el número de bits es fijo; cuando los valores no están limitados, es preciso que el tipo sea el último en una PDU, y el valor está justificado a la derecha en los octetos restantes de la PDU.

D.1.5.5 La definición del objeto de codificación (véanse 23.6.1 y 23.7.1) es:

```
integerEncoding #INT ::= {ENCODINGS {
        { IF unbounded-or-no-lower-bound
            ENCODING-SPACE
                    SIZE variable-with-determinant
                    DETERMINED BY container
                    USING OUTER
            ENCODING twos-complement} ,
        { IF bounded-with-negatives
            ENCODING-SPACE
                    SIZE fixed-to-max
            ENCODING twos-complement} ,
        { IF semi-bounded-with-negatives
            ENCODING-SPACE
                    SIZE variable-with-determinant
                    DETERMINED BY container
                    USING OUTER
            ENCODING twos-complement} ,
        { IF semi-bounded-without-negatives
            ENCODING-SPACE
                    SIZE variable-with-determinant
                    DETERMINED BY container
                    USING OUTER
            ENCODING positive-int} ,
        { IF bounded-without-negatives
            ENCODING-SPACE
                    SIZE fixed-to-max
            ENCODING positive-int}}}
```

D.1.6 Enteros positivos codificados en BCD

D.1.6.1 Este ejemplo muestra cómo se codifica un entero positivo en BCD (decimal codificado en binario) aplicando transformadas sucesivas: de entero en cadena de caracteres y a continuación de cadena de caracteres en cadena de bits.

D.1.6.2 La asignación ASN.1 es:

```
PositiveIntegerBCD ::= INTEGER(0..MAX)
```

D.1.6.3 La asignación de objeto de codificación (véanse 19.4, 24.1 y 23.4.1) es:

```
positiveIntegerBCDEncoding #PositiveIntegerBCD ::= {
        USE #CHARS
        MAPPING TRANSFORMS{{
                INT-TO-CHARS
                    -- We convert to characters (e.g., integer 42
                    -- becomes character string "42") and encode the characters
                    -- with the encoding object "numeric-chars-to-bcdEncoding"
                        SIZE variable
                        PLUS-SIGN FALSE}}
        WITH numeric-chars-to-bcdEncoding }
```

```
numeric-chars-to-bcdEncoding #CHARS ::= {
        ALIGNED TO NEXT nibble
                TRANSFORMS {{
                        CHAR-TO-BITS
                            -- We convert each character to a bitstring
                            --(e.g., character "4" becomes '0100'B and "2" becomes '0010'B)
                                AS mapped
                                CHAR-LIST { "0","1","2","3",
                                            "4","5","6","7",
                                            "8","9"}
                                BITS-LIST { '0000'B, '0001'B, '0010'B, '0011'B,
                                            '0100'B, '0101'B, '0110'B, '0111'B,
                                            '1000'B, '1001'B }}}
                REPETITION-ENCODING {
                        REPETITION-SPACE
                        -- We determine the concatenation of the bitstrings for the
                        -- characters and add a terminator (e.g.,
                        -- '0100'B + '0010'B becomes '0100 0010 1111'B)
                                SIZE variable-with-determinant
                                DETERMINED BY pattern
                 PATTERN bits:'1111'B}}
```

D.1.6.4 El número positivo es transformado primero en una cadena de caracteres por la transformada int-to-chars utilizando las opciones longitud variable y no signo más, y además la opción por defecto no relleno, con lo que se obtiene una cadena que contiene los caracteres "**0**" a "**9**". La cadena de caracteres se codifica a continuación de tal manera que cada carácter se transforma en un esquema de bits, **'0000'B** para "**0**", **'0001'B** para "**1**"**…, '1001'B** para "**9**". La cadena de bits se alinea en la frontera de un cuarteto y termina con el esquema específico **'1111'B**.

D.1.6.5 Una alternativa más compleja, no mostrada aquí pero utilizada corrientemente, consistiría en insertar la codificación BCD en una cadena de octetos, con un booleano externo que identifique si al final hay o no un cuarteto no utilizado.

D.1.7 Objeto de codificación de la clase #BITS

D.1.7.1 Este ejemplo define un objeto de codificación de la clase **#BITS** (véase 23.2.1) para una cadena de bits con alineación de octetos, rellenada con 0 y terminada con un campo de 8 bits que contiene **'00000000'B** (se supone que un valor abstracto nunca contiene ocho ceros sucesivos).

D.1.7.2 La asignación ASN.1 es:

```
Fax ::= BIT STRING (CONSTRAINED BY {-- must not contain eight successive zero bits --})
```

D.1.7.3 La asignación de objeto de codificación (véanse 23.2.1, 23,12.1 y 23.13.1) es:

```
faxEncoding #Fax ::= {
        ALIGNED TO NEXT octet
        REPETITION-ENCODING {
           REPETITION-SPACE
                        SIZE variable-with-determinant
                        DETERMINED BY pattern
                 PATTERN bits:'00000000'B}}
```

D.1.7.4 Este objeto de codificación (de la clase **#BITS**) contiene un objeto de codificación incorporado de la clase **#CONDITIONAL-REPETITION** que especifica el mecanismo y el esquema de terminación.

D.1.7.5 Al igual que en muchos de los ejemplos de este anexo, se da aquí una fuerte dependencia con respecto a los valores por defecto indicados en la cláusula 23, y se aprovecha la posibilidad de definir objetos de codificación en línea en vez de asignarlos por separado a nombres de referencia que son utilizados a continuación en otras asignaciones.

D.1.8 Objeto de codificación de un tipo cadena de octetos

D.1.8.1 La asignación ASN.1 es:

```
BinaryFile ::= OCTET STRING
```

D.1.8.2 La asignación de objeto de codificación (véase 23.9.1) es:

```
binaryFileEncoding #BinaryFile ::= {
        ALIGNED TO NEXT octet
                PADDING one
        REPETITION-ENCODING {
            REPETITION-SPACE
                    SIZE variable-with-determinant
                DETERMINED BY container
                        USING OUTER}}
```

D.1.8.3 El valor se alinea en octetos utilizando relleno de unos y termina con el final de la PDU.

D.1.9 Objeto de codificación de un tipo cadena de caracteres

D.1.9.1 La asignación ASN.1 es:

```
Password ::= PrintableString
```

D.1.9.2 La asignación de objeto de codificación (véanse 23.4.1 y 23.13.1) es:

```
passwordEncoding #Password ::= {
        ALIGNED TO NEXT octet
                TRANSFORMS {{CHAR-TO-BITS
                                        AS compact
                                        SIZE fixed-to-max
                                        MULTIPLE OF bit }}
        REPETITION-ENCODING {
            REPETITION-SPACE
                    SIZE variable-with-determinant
                    DETERMINED BY container
                            USING OUTER}}
```

D.1.9.3 La cadena se alinea en octetos utilizando relleno de ceros y termina con el final de la PDU; la codificación de caracteres se especifica como "**`compact`**", de manera que cada carácter se codifica en 7 bits utilizando **`'0000000'B`** para el primer carácter ASCII de tipo **`PrintableString`**, **`'0000001'B`** para el siguiente, y así sucesivamente.

D.1.10 Establecimiento de la correspondencia entre valores de caracteres y valores de bits

D.1.10.1 La asignación ASN.1 es:

```
CharacterStringToBit ::= IA5String ("FIRST" | "SECOND" | "THIRD")
```

D.1.10.2 La asignación de objeto de codificación (véase 19.2) es:

```
characterStringToBitEncoding #CharacterStringToBit ::= {
        USE #IntFrom0To2
        MAPPING VALUES {
            "FIRST"     TO 0,
            "SECOND"    TO 1,
            "THIRD"     TO 2}
        WITH integerEncoding}

#IntFrom0To2 ::= #INT (0..2)
```

donde "integerEncoding" se define en D.1.5.5.

D.1.10.3 Los tres valores abstractos posibles se hacen corresponder con tres números enteros y a continuación esos números se codifican en un campo de dos bits.

D.1.11 Objeto de codificación de un tipo secuencia

D.1.11.1 Aquí se codifica un tipo secuencia que tiene un campo "**a**" que lleva semántica de aplicación (es decir, es visible desde la aplicación), pero se desea también utilizarlo como un determinante de presencia de un segundo campo de entero (opcional) "b". A continuación hay una cadena de octetos que tiene alineación de octetos y está delimitada por el final de la PDU. Se necesita dar codificaciones especializadas para la opcionalidad de "**b**", y se utiliza la codificación especializada definida en D.1.8 (por referencia al objeto de codificación "binaryFileEncoding") para la cadena de octetos "**c**". Se desea codificar todo lo demás aplicando las reglas de codificación no alineada básica PER.

D.1.11.2 La asignación ASN.1 es:

```
Sequence1       ::= SEQUENCE {
        a       BOOLEAN,
        b       INTEGER OPTIONAL,
        c       BinaryFile
                -- "BinaryFile" is defined in D.1.8.1 --}
```

D.1.11.3 Las asignaciones ECN (véanse 17.5 y 23.10.1) son:

```
sequence1Encoding #Sequence1 ::= {
        ENCODE STRUCTURE {
            b USE-SET OPTIONAL-ENCODING parameterizedPresenceEncoding {< a >},
            c binaryFileEncoding
            -- "binaryFileEncoding" is defined in D.1.8.2 -- }
WITH PER-BASIC-UNALIGNED}

parameterizedPresenceEncoding {< REFERENCE:reference >}  #OPTIONAL ::= {
        PRESENCE
            DETERMINED BY field-to-be-used
            USING reference}
```

D.1.11.4 Obsérvese que no ha sido necesario proporcionar la propiedad de codificación "**DECODERS-TRANSFORMS**" en el objeto de codificación "**parameterizedPresenceEncoding**", porque el componente "**a**" era un booleano, y se ha supuesto que "**TRUE**" significa que "**b**" estaba presente. Si, no obstante, "**a**" hubiera sido un campo de entero o si el valor de la aplicación de "**TRUE**" para "**a**" significara de hecho que "**b**" estaba ausente, se habría incluido una propiedad de codificación "**DECODER-TRANSFORMS**" como en D.2.6.

D.1.12 Objeto de codificación de un tipo elección

D.1.12.1 Un tipo elección con tres alternativas se codifica utilizando el número de rótulo del contexto de la clase, codificado en un campo de tres bits, como selector. El objeto de codificación de la clase **#ALTERNATIVES** especifica que se utilice el asa de identificación "**Tag**" como determinante; el objeto de codificación de la clase **#TAG** define la posición del asa de identificación (tres bits). Para cada alternativa, el valor se codifica con las reglas de codificación no alineada básica PER.

D.1.12.2 La asignación ASN.1 es:

```
Choice ::= CHOICE {
        boolean  [1]        BOOLEAN,
integer [3] INTEGER,
string   [5] IA5String}
```

D.1.12.3 Las asignaciones ECN (véanse 23.1.1 y 23.14.1) son:

```
choiceEncoding #Choice ::= {
        ENCODE STRUCTURE {
            boolean [tagEncoding] USE-SET,
            integer [tagEncoding] USE-SET,
            string [tagEncoding] USE-SET
            STRUCTURED WITH {
                ALTERNATIVE
                    DETERMINED BY handle
                        HANDLE "Tag"}}
WITH PER-BASIC-UNALIGNED}

tagEncoding #TAG ::= {
        ENCODING-SPACE
            SIZE 3
            MULTIPLE OF bit
        EXHIBITS HANDLE "Tag" AT {0 | 1 | 2}}
```

D.1.12.4 Una manera más sencilla de proporcionar la primera asignación de D.1.12.3 consistiría quizá en definir un nuevo conjunto de objetos de codificación y aplicarlo como sigue:

```
MyEncodings #ENCODINGS ::= { tagEncoding } COMPLETED BY PER-BASIC-UNALIGNED

choiceEncoding #Choice ::= {
        ENCODE STRUCTURE {
            STRUCTURED WITH {
                ALTERNATIVE
                    DETERMINED BY handle
                        HANDLE "Tag"}}
WITH MyEncodings}
```

D.1.13 Codificación de una cadena de bits que contiene otra codificación

D.1.13.1 Un valor cadena de bits codificado con las reglas de codificación no alineada básica PER contiene la codificación de una secuencia como un número entero de octetos (rellenado con ceros) pero no necesariamente alineada en una frontera de octeto.

D.1.13.2 Las asignaciones ASN.1 son:

```
Sequence2 ::= SEQUENCE {
        a   BOOLEAN,
        b   BIT STRING (CONTAINING Sequence3) }

Sequence3 ::= SEQUENCE {
        a   INTEGER(0..10),
        b   BOOLEAN }
```

D.1.13.3 Las asignaciones ECN (véase 25.1) son:

```
sequence2Encoding #Sequence2 ::= {
        ENCODE STRUCTURE {
            b   { REPETITION-ENCODING
                REPETITION-SPACE
                    SIZE 8
                    MULTIPLE OF bit
                            CONTENTS-ENCODING {containerEncoding}
                                    COMPLETED BY PER-BASIC-UNALIGNED}}
        WITH PER-BASIC-UNALIGNED}

containerEncoding #OUTER ::= {
        PADDING
                MULTIPLE OF octet}
```

D.1.14 Conjunto de objetos de codificación

Este conjunto de objetos de codificación contiene definiciones de codificación de algunos tipos especificados en el módulo ASN.1 de D.1.15.

```
Example1Encodings #ENCODINGS ::= {
                marriedEncoding-1
            |   integerRightAlignedEncoding
            |   evenPositiveIntegerEncoding
            |   evenNegativeIntegerEncoding
            |   integerRightAlignedEncoding
            |   integerWithHoleEncoding
            |   positiveIntegerEncoding
            |   negativeIntegerEncoding
            |   positiveIntegerBCDEncoding
            |   faxEncoding
            |   binaryFileEncoding
            |   passwordEncoding
            |   characterStringToBitEncoding
            |   sequence1Encoding
            |   choiceEncoding
            |   sequence2Encoding}
```

D.1.15 Definiciones ASN.1

D.1.15.1 Este módulo ASN.1 agrupa todas las definiciones ASN.1 de D.1.1 a D.1.12.4. Se codificarán de acuerdo con los objetos de codificación definidos en el EDM de D.1.16, junto con las reglas de codificación no alineada básica PER.

```
Example1-ASN1-Module {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module1(2)}
        DEFINITIONS AUTOMATIC TAGS : :=
        BEGIN

        MyPDU ::= CHOICE {
                marriedMessage          Married,
                altitudeMessage         Altitude
                -- etc.
        }

        Married ::= BOOLEAN

        Altitude ::= INTEGER (0..65535)

         -- etc.

        END
```

D.1.16 Definiciones EDM

D.1.16.1 Este módulo EDM agrupa todas las definiciones ECN de D.1.1 a D.1.12.4.

```
Example1-EDM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module1(3)}
        ENCODING-DEFINITIONS ::=
        BEGIN

        EXPORTS Example1Encodings;
        IMPORTS #Married, #Altitude, #EvenPositiveInteger, #EvenNegativeInteger, #IntegerRightAligned,
                #IntegerWithHole, #PositiveInteger, #NegativeInteger, #PositiveIntegerBCD, #Fax,
                #BinaryFile, #Password, #CharacterStringToBit, #Sequence1, #Choice, #Sequence2
                 FROM Example1-ASN1-Module { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module1(2) };

        Example1Encodings #ENCODINGS ::= {
                    marriedEncoding-1       |
                        -- etc
                    sequence2Encoding}

        -- etc

        END
```

D.1.17 Definiciones ELM

El ELM siguiente codifica el módulo ASN.1 definido en D.1.15, utilizando objetos especificados en el EDM definido en D.1.16.

```
Example1-ELM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) elm-module1(1)}
        LINK-DEFINITIONS ::=
        BEGIN

        IMPORTS
                Example1Encodings FROM Example-EDM
                        {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module1(3)}
                #MyPDU FROM Example1-ASN1-Module
                        {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module1(2)};

        ENCODE #MyPDU WITH Example1Encodings
                COMPLETED BY PER-BASIC-UNALIGNED

        END
```

D.2 Ejemplos de especialización

Los ejemplos de esta cláusula muestran cómo se modifican partes seleccionadas de una codificación de tipos dados a fin de minimizar el tamaño de los mensajes codificados. Las codificaciones no alineadas básicas PER producen tantas codificaciones compactas como es posible. En determinados casos, sin embargo, podrían ser convenientes codificaciones especializadas:

- Hay semánticas especiales asociadas a componentes de mensajes que permiten eliminar algunos de los campos auxiliares generados por las PER.
- El usuario desea codificaciones diferentes para los campos auxiliares PER generados por defecto, tales como los campos de determinante de anchura variable.

D.2.1 Codificación distribuyendo valores en una estructura de codificación alternativa

D.2.1.1 La asignación ASN.1 es:

```
NormallySmallValues ::= INTEGER (0..1000)
-- Usually values are in the range 0..63, but sometimes the whole value range is used.
```

D.2.1.2 Las PER codificarían el tipo utilizando 10 bits. Se desea reducir al mínimo el tamaño de la codificación de tal manera que el caso normal se codifique empleando el menor número posible de bits.

NOTA – En este ejemplo se sigue un planteamiento directo sencillo. En E.1 se da un procedimiento más complejo en el que se utilizan codificaciones Huffman.

D.2.1.3 La asignación de objeto de codificación (véase 19.6) es:

```
normallySmallValuesEncoding-1 #NormallySmallValues ::= {
        USE     #NormallySmallValuesStruct
        MAPPING DISTRIBUTION {
                0..63           TO small,
                REMAINDER       TO large }
        WITH PER-BASIC-UNALIGNED}
```

D.2.1.4 La asignación de estructura de codificación es:

```
#NormallySmallValuesStruct ::= #CHOICE {
        small   #INT (0..63),
        large   #INT (64..1000)}
```

D.2.1.5 Los valores utilizados normalmente se codifican empleando el campo "`small`" y los utilizados sólo ocasionalmente se codifican empleando el campo "`large`". La selección entre los dos se lleva a cabo mediante un campo de selector de un bit generado por las PER. La longitud del campo "`small`" es de 6 bits y la longitud del campo "`large`" es de 10 bits, con lo que el caso normal se codifica utilizando 7 bits y el caso infrecuente utilizando 11 bits.

D.2.2 Codificación estableciendo la correspondencia entre valores abstractos ordenados y una estructura de codificación alternativa

D.2.2.1 El ejemplo de D.2.1 utiliza la definición explícita de la manera de establecer la correspondencia entre gamas de valores y campos de la estructura de codificación. El mismo efecto se puede conseguir de manera más sencilla utilizando el "establecimiento de la correspondencia de valores abstractos ordenados". No obstante, y a título ilustrativo, aquí se modifica también el requisito: valores arbitrariamente grandes pueden ocurrir de manera ocasional, y se supone que se elimina la constricción impuesta a la asignación ASN.1.

D.2.2.2 Las asignaciones de objeto de codificación (véase 19.5) son:

```
normallySmallValuesEncoding-2 #NormallySmallValues ::= {
        USE     #NormallySmallValuesStruct2
        MAPPING ORDERED VALUES
        WITH    NormallySmallValuesTag-encoding-plus-PER}

normallySmallValuesTag-encoding   #TAG ::= {
        ENCODING-SPACE
                SIZE 1}

NormallySmallValuesTag-encoding-plus-PER #ENCODINGS ::= {normallySmallValuesTag-encoding}
        COMPLETED BY PER-BASIC-UNALIGNED
```

D.2.2.3 La asignación de estructura de codificación es:

```
#NormallySmallValuesStruct2 ::= #CHOICE {
        small     [#TAG(0)]         #INT (0..63),
        large     [#TAG(1)]         #INT (0..MAX) }
```

D.2.2.4 El resultado es muy similar al de D.2.1, pero ahora los valores superiores a 64 que se hacen corresponder con el campo "**large**" se codifican de cero en adelante. Las dos alternativas se distinguen por un índice de un bit. Otra diferencia consiste en que el campo "**large**" se deja sin límites, con lo que el objeto de codificación puede codificar enteros tan grandes como se desee, pero a costa de un campo de longitud en el caso "**large**". El presente ejemplo se puede utilizar también si no existe un límite superior impuesto a los valores que pudieran ocurrir ocasionalmente (en la estructura de sustitución, "**large**" no está limitado). Esto ilustra de nuevo la flexibilidad de que disponen los especificadores de la ECN para diseñar codificaciones adaptadas a sus requisitos particulares.

D.2.3 Compresión de gamas de valores no continuos

D.2.3.1 Este ejemplo utiliza además un establecimiento de la correspondencia de valores abstractos ordenados. En este caso, el establecimiento de la correspondencia se utiliza para comprimir valores dispersos de una especificación ASN.1 básica. La compresión podría lograrse también definiendo el valor abstracto ASN.1 "x" de modo que tenga la semántica de aplicación de "2x", y utilizando a continuación una constricción más sencilla aplicada al tipo ASN.1 entero. La hipótesis establecida en el presente ejemplo, no obstante, es que el diseñador de la ASN.1 prefirió no hacer eso, y es preciso aplicar la compresión durante el establecimiento de la correspondencia entre valores abstractos y codificaciones.

D.2.3.2 La asignación ASN.1 es:

```
SparseEvenlyDistributedValueSet ::= INTEGER (2 | 4 | 6 | 8 | 10 | 12 | 14 | 16)
```

D.2.3.3 La codificación no alineada básica PER sólo tiene en cuenta límites inferiores y límites superiores cuando se determina el número de bits necesario para codificar un entero. Esto da lugar a esquemas de bits no utilizados en la codificación. La codificación puede ser comprimida de tal manera que se omitan los esquemas de bits no utilizados, y cada valor se codifica utilizando el número de bits mínimo.

D.2.3.4 La asignación de objeto de codificación (véase 19.5) es:

```
sparseEvenlyDistributedValueSetEncoding #SparseEvenlyDistributedValueSet ::= {
        USE     #IntFrom0To7
        MAPPING ORDERED VALUES
        WITH PER-BASIC-UNALIGNED}

#IntFrom0To7 ::= #INT (0..7)
```

D.2.3.5 Los ocho valores abstractos posibles se han hecho corresponder con la gama **0...7** y se codificarán en un campo de tres bits.

D.2.4 Compresión de gamas de valores no continuos utilizando una transformada

D.2.4.1 En el ejemplo de D.2.3 se utiliza el establecimiento de la correspondencia de valores abstractos ordenados. Se puede conseguir el mismo efecto utilizando la clase **#TRANSFORM**.

D.2.4.2 La asignación de objeto de codificación (véase 19.4) es:

```
sparseEvenlyDistributedValueSetEncoding-2 #SparseEvenlyDistributedValueSet ::= {
        USE     #IntFrom0To7
        MAPPING TRANSFORMS {{INT-TO-INT divide: 2}, {INT-TO-INT decrement:1}}
        WITH PER-BASIC-UNALIGNED}
```

D.2.4.3 De nuevo, se hacen corresponder los ocho valores abstractos posibles con la gama **0..7** y se codifican en un campo de tres bits.

D.2.5 Compresión de un conjunto de valores distribuidos de manera no uniforme estableciendo la correspondencia de valores abstractos ordenados

D.2.5.1 La asignación ASN.1 es:

```
SparseUnevenlyDistributedValueSet ::= INTEGER (0|3|5|6|11|8)
-- Out of order to illustrate that order does not matter in the constraint
```

D.2.5.2 La codificación deberá ser tal que no haya huecos en los esquemas de codificación utilizados.

D.2.5.3 La asignación de objeto de codificación es:

```
sparseUnevenlyDistributedValueSetEncoding #SparseUnevenlyDistributedValueSet ::= {
        USE  #IntFrom0To5
        MAPPING ORDERED VALUES
        WITH PER-BASIC-UNALIGNED}

#IntFrom0To5 ::= #INT (0..5)
```

D.2.5.4 Los seis valores abstractos posibles se hacen corresponder con la gama **`0..5`** y se codifican en un campo de tres bits. El establecimiento de la correspondencia es como sigue: 0→0 , 3→1 , 5→2 , 6→3 , 8→4 y 11→5.

D.2.6 Presencia de un componente opcional dependiente del valor de otro componente opcional

D.2.6.1 La asignación ASN.1 es:

```
ConditionalPresenceOnValue ::= SEQUENCE {
        a   INTEGER (0..4),
        b   INTEGER (1..10),
        c   BOOLEAN OPTIONAL
                -- Condition: "c" is present  if "a" is 0, otherwise "c" is absent --,
        d   BOOLEAN OPTIONAL
                -- Condition: "d" is absent  if "a" is 1, otherwise "d" is present -- }
        -- Note the implied presence constraints in comments.
        -- Note also that the integer field "a" carries application semantics and has values
        -- other than zero and one.  If  "a" has value 0, both "c" and "d" are present.  If "a"
        -- has value 1, both "c" and "d" are missing.  If "a" has values 3 or 4, "c" is absent
        -- and "d" is present  These conditions are very hard to express formally using ASN.1 alone.
```

D.2.6.2 El componente "**a**" actúa a modo de determinante de presencia de los componentes "**c**" y "**d**", pero una codificación PER produciría dos bits auxiliares para los componentes opcionales. Se requiere una codificación en la que esos dos bits auxiliares estén ausentes.

D.2.6.3 La asignación de objeto de codificación es:

```
conditionalPresenceOnValueEncoding #ConditionalPresenceOnValue ::= {
        ENCODE STRUCTURE {
            c   USE-SET OPTIONAL-ENCODING is-c-present{< a >},
            d   USE-SET OPTIONAL-ENCODING is-d-present{< a >}}
        WITH PER-BASIC-UNALIGNED}

is-c-present {< REFERENCE : a >} #OPTIONAL ::= {
        PRESENCE
            DETERMINED BY field-to-be-used
            USING a
                DECODER-TRANSFORMS {{INT-TO-BOOL  TRUE-IS {0}}}}

is-d-present {< REFERENCE : a >} #OPTIONAL ::= {
        PRESENCE
            DETERMINED BY field-to-be-used
            USING a
                DECODER-TRANSFORMS {{INT-TO-BOOL TRUE-IS {0 | 2 | 3 | 4}}}}
```

D.2.6.4 Se tiene aquí una especificación sencilla, formal y clara de las condiciones de presencia de "**c**" y "**d**" que pueden ser comprendidas por las herramientas de codificador-decodificador. Los comentarios ASN.1 no pueden ser manejados mediante herramientas. La provisión de codificación de opcionalidad para "**c**" y "**d**" significa que, en este caso, no se utiliza la codificación PER de "**`OPTIONAL`**", y no hay bits auxiliares.

D.2.6.5 Los objetos de codificación parametrizados "**`is-c-present`**" e "**`is-d-present`**" especifican la manera de determinar la presencia de los componentes durante la decodificación. Se señala que no se necesita (ni se permite) ninguna transformación para codificar porque el determinante tiene semántica de aplicación (es decir, está visible en la definición del tipo ASN.1). Sin embargo, una buena herramienta de codificación controlará la fijación de "**a**" por la aplicación, para tener la seguridad de que su valor es coherente con la presencia o ausencia de "**c**" y "**d**" que el código de aplicación ha determinado.

D.2.7 La presencia de un componente opcional depende de algunas condiciones externas

D.2.7.1 La asignación ASN.1 es:

```
ConditionalPresenceOnExternalCondition ::= SEQUENCE {
        a   BOOLEAN OPTIONAL
            -- Condition: "a" is present  if the external  condition "C" holds,
            -- otherwise "a" absent -- }
        -- Note that the presence constraint can only be supplied in comment.
```

D.2.7.2 El código de aplicación de un emisor y un receptor puede evaluar la condición "C" a partir de alguna información externa al mensaje. El especificador de la ECN desea disponer de herramientas con las que invocar ese código para determinar la presencia de "**a**", en vez de utilizar un bit en la codificación.

D.2.7.3 La asignación de objeto de codificación es:

```
conditionalPresenceOnExternalConditionEncoding #ConditionalPresenceOnExternalCondition ::= {
        ENCODE STRUCTURE {
            a    USE-SET OPTIONAL-ENCODING is-a-present}
        WITH PER-BASIC-UNALIGNED}

is-a-present #OPTIONAL ::=
        NON-ECN-BEGIN {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) user-notation(7)}
        extern C;
        extern channel;
        /* a is present only if channel is equal to some value "C" */
        int is_a_present() {
            if(channel == C) return 1;
            else return 0; }
        NON-ECN-END
```

D.2.7.4 Puesto que la condición es externa al mensaje, el objeto de codificación para determinar la presencia del componente "**a**" sólo puede ser especificado por una definición no ECN de un objeto de codificación. Aunque de esta manera se ahorran bits en la línea, es posible que muchos diseñadores piensen que es mejor, no obstante, incluir el bit en el mensaje para reducir la posibilidad de error, y facilitar la realización de pruebas y la supervisión técnica. Corresponde al especificador de la ECN tomar las decisiones al respecto.

D.2.8 Lista de longitud variable

D.2.8.1 La asignación ASN.1 es:

```
EnclosingStructureForList ::= SEQUENCE {
        list VariableLengthList}

VariableLengthList ::= SEQUENCE (SIZE (0..1023) ) OF INTEGER (1..2)
-- Normally the list contains only a few elements (0..31), but it might contain many.
```

D.2.8.2 La codificación no alineada básica PER codifica la longitud de la lista utilizando 10 bits incluso si normalmente la longitud está en la gama **`0..31`**. Se desea reducir al mínimo el tamaño de la codificación del determinante de longitud del caso normal, previendo al mismo tiempo valores que raramente ocurren.

D.2.8.3 La asignación de objeto de codificación es:

```
enclosingStructureForListEncoding #EnclosingStructureForList ::= {
        USE    #EnclosingStructureForListStruct
        MAPPING FIELDS
        WITH {
            ENCODE STRUCTURE {
                aux-length  list-lengthEncoding,
                list  {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH {
                                                    REPETITION-ENCODING {
                                                        REPETITION-SPACE
                                                            DETERMINED BY field-to-be-set
                                                        USING aux-length}}}
                                        WITH PER-BASIC-UNALIGNED }}
                    WITH PER-BASIC-UNALIGNED}}
                        -- First mapping: use of an encoding structure with an explicit length determinant.

list-lengthEncoding #AuxVariableListLength ::= {
        USE       #AuxVariableListLengthStruct          -- See D.2.8.4.
        MAPPING ORDERED VALUES
        WITH PER-BASIC-UNALIGNED}
        -- Second mapping: list length is encoded as a choice between a short form "normally" and
        -- a long form "sometimes".
```

D.2.8.4 Las asignaciones de estructura de codificación son:

```
#EnclosingStructureForListStruct ::= #CONCATENATION {
        aux-length   #AuxVariableListLength,
        list      #VariableLengthList}

#AuxVariableListLength ::= #INT (0..1023)

#AuxVariableListLengthStruct ::= #ALTERNATIVES {
        normally       #INT (0..31),
        sometimes   #INT (32..1023)}
```

D.2.8.5 El determinante de longitud del componente "**`list`**" es variable. El determinante de longitud de los valores de lista corta se codifican utilizando 1 bit para el determinante de selección y 5 bits para el determinante de longitud. El determinante de longitud de los valores de lista larga se codifica utilizando 1 bit para el determinante de selección y 10 bits para el determinante de longitud.

D.2.9 Listas de igual longitud

D.2.9.1 La asignación ASN.1 es:

```
EqualLengthLists ::= SEQUENCE {
        list1    List1,
        list2    List2}
        (CONSTRAINED BY {
          -- "list1" and "list2" always have the same number of elements. --
         })

List1 ::= SEQUENCE (SIZE (0..1023)) OF BOOLEAN

List2 ::= SEQUENCE (SIZE (0..1023)) OF INTEGER (1..2)
```

D.2.9.2 La "**`list1`**" y la "**`list2`**" tienen el mismo número de elementos, y el especificador de la ECN desea utilizar un determinante de longitud único para ambas listas. (Las PER codificarían campos de longitud para ambos componentes).

D.2.9.3 Las asignaciones de objeto de codificación son:

```
equalLengthListsEncoding #EqualLengthLists ::= {
        USE    #EqualLengthListsStruct
        MAPPING FIELDS
        WITH {
            ENCODE STRUCTURE {
                list1    list1Encoding{< aux-length >},
                list2    list2Encoding{< aux-length >}}
                  WITH PER-BASIC-UNALIGNED}}
```

El primer objeto de codificación se define con dos objetos de codificación parametrizados de la clase **`#List1`** y la clase **`#List2`**, respectivamente, utilizando el campo de longitud como un parámetro real. Estos dos objetos de codificación utilizan un objeto de codificación parametrizado común de la clase **`#REPETITION`**.

```
list1Encoding {< REFERENCE : length >} #List1 ::= {
        ENCODE STRUCTURE { USE-SET
                STRUCTURED WITH list-with-determinantEncoding {< length >}}
        WITH PER-BASIC-UNALIGNED}

list2Encoding {< REFERENCE : length >} #List2 ::= {
        ENCODE STRUCTURE { USE-SET
                STRUCTURED WITH list-with-determinantEncoding {< length >}}
        WITH PER-BASIC-UNALIGNED}

list-with-determinantEncoding {< REFERENCE : length-determinant >} #REPETITION ::= {
        REPETITION-ENCODING {
                REPETITION-SPACE
                  SIZE variable-with-determinant
                      MULTIPLE OF repetitions
                DETERMINED BY field-to-be-set
                USING length-determinant}}
```

D.2.9.4 Las asignaciones de estructura de codificación son:

```
#EqualLengthListsStruct ::= #CONCATENATION {
        aux-length    #AuxListLength,
        list1         #List1,
        list2         #List2}

#AuxListLength ::= #INT (0..1023)
```

D.2.10 Probabilidades de alternativas de elección no uniformes

D.2.10.1 La asignación ASN.1 es:

```
EnclosingStructureForChoice ::= SEQUENCE {
        choice     UnevenChoiceProbability }

UnevenChoiceProbability ::= CHOICE {
        frequent1   INTEGER (1..2),
        frequent2   BOOLEAN,
        common1   INTEGER (1..2),
        common2   BOOLEAN,
        common3   BOOLEAN,
        rare1       BOOLEAN,
        rare2       INTEGER (1..2),
        rare3       INTEGER (1..2)}
```

D.2.10.2 Las alternativas del tipo de elección tienen probabilidades de selección diferentes. Hay alternativas que aparecen con mucha frecuencia ("`frequent1`" y " `frequent2`"), o son bastante comunes ("`common1`", " `common2`" y " `common3`"), o aparecen sólo raramente ("`rare1`", "`rare2`" y "`rare3`"). La codificación del determinante de alternativa deberá ser tal que las alternativas que aparezcan frecuentemente tengan campos de determinante más cortos que las que aparecen raramente.

D.2.10.3 Las asignaciones de estructura de codificación son:

```
#EnclosingStructureForChoiceStruct ::= #CONCATENATION {
        aux-selector        #AuxSelector,
        choice              #UnevenChoiceProbability }
        -- Explicit auxiliary alternative determinant for "choice".

#AuxSelector ::= #INT (0..7)
```

D.2.10.4 Las asignaciones de objeto de codificación son:

```
enclosingStructureForChoiceEncoding #EnclosingStructureForChoice ::= {
        USE      #EnclosingStructureForChoiceStruct
        MAPPING FIELDS
        WITH {
            ENCODE STRUCTURE {
                aux-selector    auxSelectorEncoding,
                choice {
                   ENCODE STRUCTURE  {
                       STRUCTURED WITH  {
                           ALTERNATIVE
                               DETERMINED BY field-to-be-set
                               USING aux-selector}}
                   WITH PER-BASIC-UNALIGNED }}
                  WITH PER-BASIC-UNALIGNED} }
                  -- First mapping: inserts an explicit auxiliary alternative determinant.
                  -- This encoding object specifies that an auxiliary determinant is used
                  -- as an alternative determinant.

auxSelectorEncoding #AuxSelector ::= {
        USE      #BITS
        -- ECN Huffman
        -- RANGE (0..7)
        -- (0..1) IS 60%
        -- (2..4) IS 30%
        -- (5..7) IS 10%
        -- End Definition
        -- Mappings produced by "ECN Public Domain Software for Huffman encodings, version 1"
        -- (see E.8)
        MAPPING TO BITS {
                0 ..  1 TO '10'B   .. '11'B,
                2 ..  4 TO '001'B .. '011'B,
                5  TO '0001'B,
                6 ..  7 TO '00000'B .. '00001'B}
        WITH PER-BASIC-UNALIGNED }
        -- Second mapping: Map determinant indexes to bitstrings
```

D.2.10.5 En esas asignaciones se han cuantificado "frequent", "common" y "rare" como el 60%, el 30% y el 10%, respectivamente, y se ha utilizado el generador Huffman ECN de dominio público (véase E.8) para determinar los esquemas de bits óptimos que se han de emplear con cada gama de enteros.

D.2.10.6 Lo anterior es un óptimo en sentido matemático, pero su importancia como porcentaje del tráfico total depende del contenido de las otras partes del protocolo. Si bien producir y utilizar codificaciones óptimas no cuesta nada en términos de esfuerzo de implementación (porque se pueden aplicar herramientas), es posible que las ganancias finales sean poco significativas.

D.2.11 Mensaje de versión 1

D.2.11.1 La asignación ASN.1 es:

```
Version1Message ::= SEQUENCE {
        ie-1        BOOLEAN,
        ie-2        INTEGER (0..20)}
```

Se desea utilizar la codificación no alineada básica PER, pero se pretende añadir más campos en la versión 2, y se desea especificar que los sistemas de la versión 1 deben aceptar e ignorar cualquier material adicional de la PDU.

D.2.11.2 Se utilizan dos estructuras de codificación para codificar el mensaje: la primera es la estructura de codificación generada implícitamente que sólo contiene los campos de la versión 1, y la segunda es una estructura que se define como contenedora de los campos de la versión 1 más un campo de relleno de longitud variable que prolonga el final de la PDU. El sistema de la versión 1 utiliza la primera estructura para codificar, y la segunda para decodificar. Aparte de esta aproximación a la extensibilidad, todas las codificaciones son no alineadas básicas PER. La estructura de decodificación de la versión 1 es:

```
#Version1DecodingStructure ::= #CONCATENATION {
        ie-1        #BOOL,
        ie-2        #INT (0..20),
        future-additions    #PAD}
```

D.2.11.3 Las asignaciones de objeto de codificación son:

```
version1MessageEncoding  #Version1Message ::= {
        ENCODE-DECODE
            {ENCODE WITH PER-BASIC-UNALIGNED }
        DECODE AS IF decodingSpecification}
decodingSpecification #Version1Message ::= {
        USE #Version1DecodingStructure
        MAPPING FIELDS
                WITH {
                ENCODE STRUCTURE {
                future-additions    additionsEncoding{< OUTER >} }
                WITH PER-BASIC-UNALIGNED}}

additionsEncoding {< REFERENCE:determinant >} #PAD ::= {
        ENCODING-SPACE
            SIZE encoder-option-with-determinant
            DETERMINED BY container
            USING determinant}
```

D.2.12 Conjunto de objetos de codificación

Este conjunto de objetos de codificación contiene las definiciones de la codificación de algunos de los tipos especificados en el módulo ASN.1 denominado "**`Example2-ASN1-Module`**" (el resto se codifica utilizando la codificación no alineada básica PER).

```
Example2Encodings #ENCODINGS ::= {
        normallySmallValuesEncoding-1                           |
        sparseEvenlyDistributedValueSetEncoding             |
        sparseUnevenlyDistributedValueSetEncoding           |
        conditionalPresenceOnValueEncoding                  |
        conditionalPresenceOnExternalConditionEncoding    |
        enclosingStructureForListEncoding                       |
        equalLenghListsEncoding                                         |
        enclosingStructureForChoiceEncoding                 |
        version1MessageEncoding }
```

D.2.13 Definiciones ASN.1

Este módulo agrupa todas las definiciones ASN.1 de D.2.1 a D.2.11 que se codificarán de acuerdo con los objetos de codificación definidos en el EDM, y hace una lista de las demás definiciones ASN.1 que serán codificadas aplicando las reglas de codificación no alineada básica PER.

```
Example2-ASN1-Module {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module2(5)}
        DEFINITIONS AUTOMATIC TAGS : :=
        BEGIN

        ExampleMessages ::= CHOICE {
                firstExample        NormallySmallValues,
                secondExample       SparseEvenlyDistributedValueSet
                -- etc.
    }

        NormallySmallValues ::= INTEGER (0..1024)

        SparseEvenlyDistributedValueSet ::= INTEGER (2 | 4 | 6 | 8 | 10 | 12 | 14 | 16)

        -- etc.

        END
```

D.2.14 Definiciones EDM

```
Example2-EDM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module2(6)}
        ENCODING-DEFINITIONS ::=
        BEGIN

        EXPORTS Example2Encodings;

        IMPORTS #NormallySmallValues, #SparseEvenlyDistributedValue,
                #SparseUnevenlyDistributedValueSet, #ConditionalPresenceOnValueSet,
                #ConditionalPresenceOnExternalCondition,
                #EnclosingStructureForList, #EqualLengthLists, #EnclosingStructureForChoice,
                #Version1Message
                FROM Example2-ASN1-Module
                {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module2(5)};

        Example2Encodings #ENCODINGS ::= {
                        normallySmallValuesEncoding   |
                                -- etc.
                        extensibleMessageEncoding}
        -- etc.

        END
```

D.2.15 Definiciones ELM

El ELM siguiente está asociado al módulo ASN.1 definido en D.2.13 y al EDM definido en D.2.14.

```
Example2-ELM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) elm-module2(4)}
        LINK-DEFINITIONS ::=
        BEGIN

        IMPORTS
                Example2Encodings FROM Example2-EDM
                        {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module2(6)}
                #ExampleMessages FROM Example2-ASN1-Module
                        {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module2(5)};

        ENCODE #ExampleMessages WITH Example2Encodings
                COMPLETED BY PER-BASIC-UNALIGNED

        END
```

D.3 Ejemplos de estructura generada explícitamente

Los ejemplos descritos en D.3.1 a D.3.4 muestran la utilización de estructuras generadas explícitamente para sustituir una clase de codificación de una estructura de codificación generada implícitamente por una clase sinónima. A continuación se producen codificaciones especializadas incluyendo en el conjunto de objetos de codificación un objeto de la clase sinónima.

Los ejemplos se presentan utilizando el formato siguiente:

- La "asignación de tipo ASN.1", que da la definición del tipo ASN.1 original.
- El requisito, con lo que se tiene la relación de cambios que es preciso introducir con respecto a las codificaciones obtenidas aplicando la codificación no alineada básica PER.
- La modificación de la estructura de codificación generada implícitamente para producir una estructura de codificación nueva.
- La clase de codificación y las asignaciones de objeto de codificación.

D.3.1 Secuencia con componentes opcionales definidos por un puntero

D.3.1.1 La asignación ASN.1 es:

```
Sequence1 ::= SEQUENCE {
  component1  INTEGER          OPTIONAL,
  component2  INTEGER          OPTIONAL,
  component3  VisibleString }
```

D.3.1.2 En vez de utilizar el mapa de bits PER para los dos componentes de tipo entero marcados "**`OPTIONAL`**", se determina la presencia y la posición de esos componentes mediante punteros al comienzo de la codificación de la secuencia. Cada puntero contiene 0 (componente ausente) o bien un desplazamiento relativo con respecto a la codificación del componente que empieza en una frontera de octeto.

D.3.1.3 La clase de codificación **`#INTEGER`** es sustituida por **`"#Integer-with-pointer-concat`**" en el objeto de codificación de "**`sequence1-encoding`**". La clase **`"#Integer-with-pointer-concat`**" se define como una estructura de concatenación que contiene un elemento que es el elemento sustituido combinado con una clase en la categoría opcionalidad "**`#Integer-optionality`**".

D.3.1.4 A continuación se definen dos objetos de codificación. El primero, "**`integer-with-pointer-concat-encoding`**" de la clase **`#Integer-with-pointer-concat`**, recibe tres parámetros: el elemento sustituido, el puntero y el conjunto de objetos de codificación combinados actual (véase 22.1.3.7). El segundo, "**`integer-optionality-encoding`**" de la clase "**`#Integer-optionality`**", recibe un parámetro, el puntero, que se utiliza para determinar la presencia del componente. Puesto que **`PER-BASIC-UNALIGNED`** no contiene un objeto de codificación de la clase **`#CONCATENATION`** con componentes opcionales, es preciso definir un tercer objeto de codificación de la clase **`#CONCATENATION`**. Este objeto, "concat", utiliza fijaciones de valores por defecto.

D.3.1.5 La clase de codificación y las asignaciones de objeto de codificación son:

```
sequence1-encoding #SEQUENCE ::= {
        REPLACE OPTIONALS
                WITH #Integer-with-pointer-concat
                ENCODED BY integer-with-pointer-concat-encoding
                        INSERT AT HEAD #Pointer
        ENCODING-SPACE
                SIZE variable-with-determinant
                DETERMINED BY container
                USING OUTER }

#Pointer ::= #INTEGER

#Integer-with-pointer-concat {< #Element >} ::= #CONCATENATION {
        element  #Element OPTIONAL-ENCODING #Integer-optionality }

#Integer-optionality ::= #OPTIONAL

integer-optionality-encoding{< REFERENCE: start-pointer>} #Integer-optionality ::= {
        ALIGNED TO ANY octet
        START-POINTER start-pointer
        PRESENCE DETERMINED BY pointer}
```

```
integer-with-pointer-concat-encoding {< #Element, REFERENCE:pointer, #ENCODINGS:EncodingObjectSet >}
        #Integer-with-pointer-concat{< #Element >} ::= {
        ENCODE STRUCTURE {
            element USE-SET OPTIONAL-ENCODING integer-optionality-encoding{< pointer >}}
        WITH EncodingObjectSet}

concat #CONCATENATION ::= {
        ENCODING-SPACE }
```

D.3.2 Adición de un tipo booleano como determinante de presencia

D.3.2.1 La asignación ASN.1 es:

```
Sequence2 ::= SEQUENCE {
  component1   BOOLEAN        OPTIONAL,
  component2   INTEGER,
  component3   VisibleString   OPTIONAL }
```

D.3.2.2 En vez de utilizar el mapa de bits PER para componentes marcados "**`OPTIONAL`**", se relaciona la presencia de un componente opcional con el valor de un bit de presencia único que es igual a 1 (componente ausente) o a 0 (componente presente). En ese caso, se invierte el bit de presencia.

D.3.2.3 Las estructuras de codificación y los objetos de codificación se definen como sigue:

La clase de codificación **`#OPTIONAL`** se redenomina como **`#Sequence2-optional`** en la cláusula "**`RENAMES`**" (véase D.3.7). La clase "**`#Sequence2`**" es sustituida, por tanto, implícitamente por:

```
#Sequence2 ::= #SEQUENCE  {
  component1    #BOOL         OPTIONAL-ENCODING #Sequence2-optional,
  component2    #INTEGER,
component3    #VisibleString OPTIONAL-ENCODING #Sequence2-optional}
```

donde:

```
#Sequence2-optional ::= #OPTIONAL
```

A continuación se define un objeto de codificación de la clase "**`#Sequence2-optional`**"; ese objeto sustituye la definición de codificación de componente (véase 23.10.3.2) por la clase "**`Optional-with-determinant`**", utilizando el grupo de sustitución.

```
sequence2-optional-encoding #Sequence2-optional ::= {
        REPLACE STRUCTURE
        WITH #Optional-with-determinant
                ENCODED BY optional-with-determinant-encoding}
```

Esta clase, que es parametrizada por el componente original, pertenece a la categoría concatenación y tiene dos componentes: el determinante (booleano) y el componente original.

```
#Optional-with-determinant{< #Element >} ::= #CONCATENATION {
  determinant   #BOOLEAN,
  component     #Element OPTIONAL-ENCODING #Presence-determinant}
```

donde:

```
#Presence-determinant ::= #OPTIONAL
```

A continuación se define un objeto de codificación de la clase "**`#Optional-with-determinant`**"; ese objeto tiene dos parámetros ficticios: la clase del componente y un conjunto de objetos de codificación utilizado para codificar todo excepto el determinante y la opcionalidad del componente:

```
optional-with-determinant-encoding {< #Element, #ENCODINGS: Sequence2-combined-encoding-object-set >}
        #Optional-with-determinant {< #Element >}    ::= {
  ENCODE STRUCTURE {
      determinant   determinant-encoding,
  component USE-SET
                        OPTIONAL-ENCODING if-component-present-encoding{< determinant >} }
      WITH Sequence2-combined-encoding-object-set }
```

La codificación es especificada por completo por la definición de los objetos de codificación "**if-component-present-encoding**" y "**determinant-encoding**":

```
if-component-present-encoding {<REFERENCE:presence-bit>} #Presence-determinant ::= {
        PRESENCE
                DETERMINED BY field-to-be-set
                USING presence-bit}

determinant-encoding #BOOLEAN ::= {
        ENCODING-SPACE
            SIZE 1
            MULTIPLE OF bit
        TRUE-PATTERN bits:'0'B
        FALSE-PATTERN bits:'1'B}
```

D.3.3 Secuencia con componentes opcionales identificados por un rótulo único y delimitados por un campo de longitud

D.3.3.1 Las asignaciones ASN.1 son:

```
Octet3 ::= OCTET STRING (CONTAINING Sequence3)

Sequence3 ::=SEQUENCE  {
  component1    [0] BIT STRING (SIZE(0..2047))  OPTIONAL,
  component2    [1] OCTET STRING (SIZE(0..2047))     OPTIONAL,
  component3    [2] VisibleString (SIZE(0..2047))   OPTIONAL }
```

D.3.3.2 Cada componente es identificado por un rótulo de cuatro bits y la longitud total de la secuencia se especifica con un campo de once bits que precede a la codificación del primer componente.

D.3.3.3 Las clases de codificación **#OCTECTS**, #**OPTIONAL** y **#TAG** se redenominan respectivamente como **#Octects3**, **#Sequence3-optional** y **#TAG-4-bits** en la cláusula "**RENAMES**" (véase D.3.7). A continuación se definen los objetos de codificación de las clases de codificación nuevas.

D.3.3.4 La clase de codificación y las asignaciones de objeto de codificación de la cadena de octetos son:

```
#Octets3 ::= #OCTET-STRING

octets3-encoding #Octets3 ::= {
        REPETITION-ENCODING {
                REPLACE STRUCTURE
                        WITH #Octets-with-length
                        ENCODED BY octets-with-length-encoding}}

#Octets-with-length{< #Element >} ::= #CONCATENATION {
        length    #INT(0..2047),
        octets    #Element}

octets-with-length-encoding{< #Element >} #Octets-with-length{< #Element >} ::= {
        ENCODE STRUCTURE {
                octets octets-encoding{< length >}}
        WITH PER-BASIC-UNALIGNED}

octets-encoding{< REFERENCE:length >} #OCTETS ::= {
        REPETITION-ENCODING {
                REPETITION-SPACE
                  SIZE variable-with-determinant
               DETERMINED BY field-to-be-set
               USING length} }
```

D.3.3.5 La clase de codificación y las asignaciones de objeto de codificación de la secuencia son:

```
#Sequence3-optional ::= #OPTIONAL

sequence3-optional-encoding  #Sequence3-optional ::= {
  PRESENCE
            DETERMINED BY container
                USING OUTER}

#TAG-4-bits ::= #TAG

tag-4-bits-encoding  #TAG-4-bits      ::= {
  ENCODING-SPACE
    SIZE 4}
```

D.3.4 Tipo secuencia de (sequence-of) con una cuenta

D.3.4.1 La asignación ASN.1 es:

```
SequenceOfIntegers ::= SEQUENCE(SIZE(0..63)) OF INTEGER(0..1023)
```

D.3.4.2 El número de elementos se codifica en un campo de seis bits que precede a la codificación del primer elemento.

D.3.4.3 La clase de codificación **#SEQUENCE-OF** se redenomina como **#SequenceOf** en la cláusula "**RENAMES**" (véase D.3.7). Se define un objeto de codificación de la clase de codificación nueva. La clase de codificación y las asignaciones de objeto de codificación son:

```
#SequenceOf ::= #REPETITION

sequenceOf-encoding #SequenceOf ::= {
   REPETITION-ENCODING {
                REPLACE STRUCTURE
                WITH #SequenceOf-with-count
                ENCODED BY sequenceOf-with-count-encoding}}


#SequenceOf-with-count{< #Element >} ::= #CONCATENATION {
          count     #INT(0..63),
          elements #Element }


sequenceOf-with-count-encoding{< #Element >} #SequenceOf-with-count{< #Element >} ::= {
          ENCODE STRUCTURE {
              elements {
                          ENCODE STRUCTURE {
                                  STRUCTURED WITH elements-encoding{< count >}}
                          WITH PER-BASIC-UNALIGNED}}
          WITH PER-BASIC-UNALIGNED}


elements-encoding{< REFERENCE:count >} #REPETITION ::= {
          REPETITION-ENCODING {
              REPETITION-SPACE
                 SIZE variable-with-determinant
          MULTIPLE OF repetitions
                 DETERMINED BY field-to-be-set
                 USING count}}
```

D.3.4.4 El campo cuenta se codifica utilizando las reglas de codificación PER para un tipo entero con la constricción de gama de valores (0...63), lo que da un campo de seis bits.

D.3.5 Conjunto de objetos de codificación

El conjunto de objetos de codificación contiene objetos de codificación de clases definidas en el módulo EDM.

```
Example3Encodings          #ENCODINGS ::= {
                sequence1-encoding                  |
                concat                              |
                sequence2-optional-encoding         |
                octets3-encoding                    |
                sequence3-optional-encoding         |
                tag-4-bits-encoding                 |
                sequenceOf-encoding }
```

D.3.6 Definiciones ASN.1

Este módulo agrupa las definiciones ASN.1 de D.3.1 a D.3.4 que se codificarán de acuerdo con los objetos de codificación definidos en el EDM de D.3.7.

```
Example3-ASN1-Module {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module3(9)}
        DEFINITIONS
        AUTOMATIC TAGS ::=
        BEGIN

        Sequence1 ::=       SEQUENCE    {
            component1      BOOLEAN             OPTIONAL,
            component2      INTEGER             OPTIONAL,
            component3      VisibleString       OPTIONAL }

 -- etc.

END
```

D.3.7 Definiciones EDM

```
Example3-EDM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module3(10)}
ENCODING-DEFINITIONS ::=
BEGIN

EXPORTS Example3Encodings;
RENAMES
 #OPTIONAL AS #Sequence2-optional
    IN #Sequence2
 #OCTET-STRING AS #Octets3
    IN ALL
 #OPTIONAL AS #Sequence3-optional
    IN #Sequence3
 #TAG AS #TAG-4-bits
    IN #Sequence3
FROM Example3-ASN1-Module { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module3(9)};

Example3Encodings #ENCODINGS  ::= {
   sequence1-optional-encoding |
              -- etc.
   sequenceOf-encoding }
 -- etc.
END
```

D.3.8 Definiciones ELM

El ELM siguiente está asociado al módulo ASN.1 definido en D.3.6 y al EDM definido en D.3.7.

```
Example3-ELM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) elm-module3(8)}
LINK-DEFINITIONS ::=
BEGIN

IMPORTS Example3Encodings, #Sequence1, #Sequence2, #Octet3, #Sequence3, #SequenceOfIntegers
FROM Example3-EDM { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module3(10) };

ENCODE #Sequence1, #Sequence2, #Octet3, #Sequence3, #SequenceOfIntegers
WITH Example3Encodings
COMPLETED BY PER-BASIC-UNALIGNED

END
```

D.4 Ejemplo de codificación del bit "more"

D.4.1 Descripción del problema

D.4.1.1 Este ejemplo se ha tomado de la Rec. UIT-T Q.763 (*Sistema de señalización N.º7 – Formatos y códigos de la parte usuario de la RDSI*).

D.4.1.2 Existe el requisito de producir la codificación siguiente como una serie de octetos:

8	7 6	5 4 3 2 1
indicador de extensión	reserva	perfil de protocolo

D.4.1.3 El bit 8 es un "indicador de extensión". Si es 0, hay un octeto que sigue con el mismo formato. Si es 1, este octeto es el último octeto de la serie.

NOTA – La codificación PER de variables booleanas es 1 para "**TRUE**" y 0 para "**FALSE**", y la ECN requiere que el último elemento retorne "**FALSE**", y los elementos anteriores "**TRUE**". Así pues, si se utiliza una variable booleana codificada según PER para el bit "more" (bit "más"), es preciso aplicar la transformada "**not**".

D.4.1.4 Este es el empleo tradicional de un bit "more", si bien con un cero para "más" y un uno para "último", lo que resulta un tanto inusual.

D.4.1.5 El ejemplo se simplificaría si la utilización del "indicador de extensión" tuviera cero y uno intercambiados, y si no hubiera bits de "reserva", pero aquí se ha preferido la utilización del ejemplo real.

D.4.1.6 Hay cuatro procedimientos para resolver este problema.

D.4.1.7 El primer procedimiento consiste en incluir un componente en la especificación ASN.1 para proporcionar el determinante de bit "more" (véase D.4.2). Este procedimiento se desaconseja por dos motivos. El primero, porque la definición de tipo ASN.1 contiene un componente que no lleva semántica de aplicación. El segundo, porque exige que la aplicación fije (de manera redundante) este campo correctamente en cada elemento de la repetición de un bit "more".

D.4.1.8 El segundo procedimiento consiste en utilizar establecimientos de la correspondencia entre los valores de una estructura generada implícitamente y una estructura definida por el usuario que incluye el determinante de bit "more" (véase D.4.3).

D.4.1.9 El tercer procedimiento consiste en utilizar el mecanismo de sustitución para incluir el determinante de bit "more" (véase D.4.4).

D.4.1.10 El cuarto procedimiento consiste en utilizar la inserción en el extremo de cabeza del determinante de bit "more" (lo cual no se ilustra aquí).

D.4.1.11 Cada uno de los tres últimos procedimientos tiene sus propias ventajas, y la elección de uno u otro es en buena medida una cuestión de estilo.

D.4.2 Utilización de la ASN.1 para proporcionar el determinante de bit "more"

D.4.2.1 En este procedimiento, la ASN.1 refleja todos los campos en la codificación. Algo que se considera "sucio" por lo general, ya que los campos que sólo deberían ser visibles en la codificación son visibles en la aplicación, reduciendo así la "ocultación de la información" que es en lo que radica la fortaleza de la ASN.1. En este caso, la ASN.1 es:

```
ProfileIndication ::= SEQUENCE OF
        SEQUENCE {
                more-bit            BOOLEAN,
                reserved            BIT STRING (SIZE (2)),
                protocol-Profile-ID    INTEGER (0..32) }
```

D.4.2.2 La estructura de codificación generada implícitamente es:

```
#ProfileIndication ::= #SEQUENCE-OF {
        #SEQUENCE {
                more-bit            #BOOLEAN,
                reserved            #BIT-STRING (SIZE (2)),
                protocol-Profile-ID    #INTEGER (0..32) } }
```

D.4.2.3 Primero se produce un objeto de codificación genérico para **#SEQUENCE-OF** que utiliza un bit "more" en un campo identificado como un parámetro del objeto de codificación, y con **BOOLEAN TRUE** (codificado como un bit "1" único por las PER) para el último elemento:

```
more-bit-encoding {< REFERENCE:more-bit >} #SEQUENCE-OF ::= {
        REPETITION-ENCODING {
                REPETITION-SPACE
                        SIZE variable-with-determinant
                        DETERMINED BY flag-to-be-set
                                USING more-bit
                                        ENCODER-TRANSFORMS
                                                { { BOOL-TO-BOOL AS logical:not } } } }
```

D.4.2.4 Este objeto de codificación se utiliza también en D.4.3 y D.4.4 ya que proporciona la descripción fundamental de la codificación necesaria para la repetición.

D.4.2.5 Con el primer procedimiento (¡sencillo pero sucio!) se puede definir ahora el objeto de codificación de **#ProfileIndication** utilizando **ENCODE STRUCTURE**, y aplicar ese objeto de codificación en el ELM, con lo que se completa el ejemplo. El objeto de codificación se define de la manera siguiente:

```
profileIndicationEncoding #ProfileIndication ::= {
        ENCODE STRUCTURE {
                STRUCTURED WITH
                        more-bit-encoding {< more-bit >} }
        WITH PER-BASIC-UNALIGNED }
```

D.4.3 Utilización de establecimientos de la correspondencia de valores para proporcionar el determinante de bit "more"

D.4.3.1 En este procedimiento se oculta la estructura de codificación en la definición ECN de una estructura de codificación definida por el usuario, y se utilizan los establecimientos de la correspondencia de valores emparejando campos para que una codificación de la estructura de codificación definida por el usuario codifique una definición de tipo ASN.1 simplificada.

D.4.3.2 La definición de tipo ASN.1 es ahora:

```
ProfileIndication2 ::= SEQUENCE OF
        protocol-Profile-ID  INTEGER (0..32)
```

D.4.3.3 Esto tiene una estructura de codificación generada implícitamente (a la que se aplican las codificaciones en el ELM) de:

```
#ProfileIndication2 ::=  #SEQUENCE-OF {
        protocol-Profile-ID  #INTEGER (0..32) }
```

D.4.3.4 Se define una estructura de codificación de la codificación requerida, similar a la ASN.1 que se escribió en el primer procedimiento (véase D.4.2.1), con la salvedad de que se utiliza **#PAD** para los bits reservados:

```
#ProfileIndicationStruct ::= #SEQUENCE-OF {
        #SEQUENCE {
                more-bit-field          #BOOLEAN,
                reserved            #PAD,
                protocol-Profile-ID     #INTEGER (0..32) } }
```

D.4.3.5 Ahora se necesita un objeto de codificación del **#PAD** de dos bits, antes de poder completar la codificación:

```
pad-encoding #PAD ::= {
        ENCODING-SPACE SIZE 2
        PATTERN bits:'00'B }
```

NOTA – En 23.11.4.2 se especifica que los decodificadores deberán aceptar cualquier valor para los bits de **#PAD**, que es lo que aquí se requiere, por lo que no hace falta una codificación/decodificación diferencial

D.4.3.6 Se define un objeto de codificación de la propia estructura, lo cual se parece en buena medida al primer procedimiento (véase D.4.2.5):

```
profileIndicationStructEncoding #ProfileIndicationStruct ::= {
        ENCODE STRUCTURE {
                STRUCTURED WITH
                        more-bit-encoding {< more-bit-field >} }
        WITH {pad-encoding} COMPLETED BY PER-BASIC-UNALIGNED }
```

D.4.3.7 Por último, se utiliza el establecimiento de la correspondencia de valores entre la estructura generada implícitamente y la propia estructura generada explícitamente para definir la codificación final:

```
profileIndication2Encoding #ProfileIndication2 ::= {
        USE #ProfileIndicationStruct
                MAPPING FIELDS
        WITH profileIndicationStructEncoding }
```

D.4.4 Utilización del mecanismo de sustitución para proporcionar el determinante de bit "more"

D.4.4.1 En el procedimiento final, se define una codificación de secuencia de (sequence-of) genérica que se puede aplicar a cualquier secuencia. Para ello se necesita una estructura de codificación parametrizada:

```
#SequenceOfStruct {< #Component >} ::=
        #SEQUENCE {
                more-bit-field          #BOOLEAN,
                reserved            #PAD,
                sequence-of-component #Component }
```

D.4.4.2 Se define la codificación de secuencia de (sequence-of) propia para efectuar una sustitución del componente por esta estructura, especificando la codificación del bit "more" y utilizando la codificación de relleno definida:

```
sequence-of-encoding #SEQUENCE-OF ::= {
    REPETITION-ENCODING {
        REPLACE COMPONENT WITH #SequenceOfStruct
            REPETITION-SPACE
                SIZE variable-with-determinant
                DETERMINED BY flag-to-be-set
                USING more-bit-field
                    ENCODER-TRANSFORMS
                        { { BOOL-TO-BOOL AS logical:not } } } }
```

D.4.4.3 Cuando se aplica esto en el ELM, se utiliza "**`COMPLETED BY PER-BASIC-UNALIGNED`**" como conjunto de objetos de codificación combinados con el que se completa la codificación, dando el efecto deseado.

D.5 Protocolo de legado especificado con notación tabular

D.5.1 Introducción

D.5.1.1 El objetivo del ejemplo de esta cláusula es mostrar cómo se construyen definiciones ECN para un protocolo cuyas codificaciones de mensajes han sido especificadas utilizando imágenes de "bits y bytes" y notación tabular. En los cuadros que siguen figura el contenido de los mensajes (sólo el "**`Message 1`**" (mensaje 1) se muestra completo):

Message 1:

	8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Octet 1	Message id							
Octet 2	A			b-flag	c-len			reserved
Octet 3	b1		b2	reserved	b3		reserved	
…								
Octet Y	c1				c2			
Octet Y+1	c3						reserved	
…								
Octet Z	d1	d2			d3			reserved

Message 2:

	8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Octet 1	Message id							
Octet 2…	Something – 1							

Message 3:

	8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Octet 1	Message id							
Octet 2…	Something – 2							

D.5.1.2 Todos los mensajes tienen una parte de encabezamiento común (mostrada con fondo gris en los cuadros). En este ejemplo se utiliza sólo para la identificación del mensaje.

D.5.1.3 Message 1 tiene tres clases de campos:

- campos obligatorios ("a")
- campos obligatorios que son determinantes de otros campos ("b-flag", "c-len")
- campos opcionales ("b", "c" y "d")

D.5.1.4 Es preciso que los tres campos "b", "c" y "d" empiecen en una frontera de octeto.

D.5.1.5 Los campos "b", "c" y"d" están compuestos por subcampos ("b1", "b2", "b3", "c1", etc) de longitud fija. Además, los campos "c" y "d" pueden aparecer múltiples veces (si bien antes sólo se ha mostrado una ocurrencia). El campo "b2" ha de empezar en una frontera de cuarteto.

D.5.1.6 La presencia de un componente opcional se indica utilizando diferentes métodos:

- El campo "b" está presente si el valor del campo "b-flag" es 1.
- El campo "d" está presente si se han dejado octetos en el mensaje.

D.5.1.7 La longitud de un campo que puede aparecer múltiples veces se determina utilizando diferentes métodos:

- El número de repeticiones del campo "c" es gobernado por el campo de determinante "c-len".
- El número de repeticiones del campo "d" es determinado por el final del mensaje.

D.5.1.8 El método ASN.1 siguiente contiene definiciones de las estructuras de mensajes presentadas anteriormente. Se han tomado las decisiones de diseño siguientes:

- Hay un tipo encapsulador que contiene las definiciones comunes de todos los mensajes.
- Los campos de determinante auxiliares de los mensajes son visibles a nivel de la ASN.1. Se señala que esto se hace así para simplificar la exposición del presente ejemplo, pero debería ser práctica normal mantener esos campos fuera de la definición ASN.1 a menos que lleven semántica de aplicación real.
- La extensibilidad se expresa en forma de comentarios.
- El relleno no es visible.

D.5.1.9 El módulo ASN.1 es:

```
LegacyProtocol-ASN1-Module {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module4(11)}

    DEFINITIONS AUTOMATIC TAGS ::=
    BEGIN

    LegacyProtocolMessages ::= SEQUENCE {
            message-id          ENUMERATED {message1, message2, message3},
            messages CHOICE {
                message1      Message1,
                message2      Message2,
                message3      Message3}}
            -- The CHOICE is constrained by the value of message-id.

    Message1 ::= SEQUENCE {
            a     A,
            b-flag  BOOLEAN,
            c-len   INTEGER (0..max-c-len),
            b     B OPTIONAL,     -- determined by "b-flag"
            c     C,          -- determined by "c-len"
            d     D OPTIONAL}     -- determined by end of PDU


    A ::= INTEGER (0..7)      -- Values 5..7 are reserved for future use.  Version 1 systems should treat 5 to 7 as 4.

    B ::= SEQUENCE {
            b1    ENUMERATED { e0, e1, e2, e3 },
            b2    BOOLEAN,
            b3    INTEGER (0..3) }

    C ::= SEQUENCE (SIZE (0..max-c-len)) OF C-elem

    C-elem ::= SEQUENCE {
            c1    BIT STRING (SIZE (4)),
            c2    INTEGER (0..1024) }

    D ::= SEQUENCE (SIZE (0..max-d-len)) OF D-elem

    D-elem ::= SEQUENCE {
            d1    BOOLEAN,
            d2    ENUMERATED { f0, f1, f2, f3, f4, f5, f6, f7 },
            d3    INTEGER (0..7) }

    max-c-len INTEGER ::= 7

    max-d-len INTEGER ::= 20

    Message2 ::= SEQUENCE {
            -- something 1 -- }


    Message3 ::= SEQUENCE {
            -- something 2 -- }


    END
```

D.5.1.10 El módulo EDM de D.5.7 contiene definiciones de codificación de los mensajes especificados en el módulo ASN.1 "**`LegacyProtocol-ASN1-Module`**". Se han tomado las decisiones de diseño siguientes:

- El relleno dentro de octetos se especifica de manera explícita como campos de relleno.
- El relleno de alineación no se especifica como campos de relleno explícito.

D.5.2 Definición de codificación de la estructura de mensaje de nivel máximo

D.5.2.1 El objeto de codificación "**`legacyProtocolMessagesEncoding`**" especifica la manera de codificar partes comunes del protocolo de legado. El identificador del mensaje se especifica en ASN.1 como un tipo enumerado. La codificación no alineada básica PER codifica "**`message-id`**" utilizando el número mínimo de bits (es decir, 2) pero aquí sería preferible que se codificara utilizando 8 bits. Además, hay que especificar que "**`message-id`**" se ha de utilizar como un determinante de "**`messages`**"

D.5.2.2 El objeto de codificación "**`legacyProtocolMessagesEncoding`**" es:

```
legacyProtocolMessagesEncoding #LegacyProtocolMessages ::= {
        ENCODE STRUCTURE {
                message-id {
                        ENCODING {
                                ENCODING-SPACE
                                        SIZE 8}},
            messages {
                        ENCODE STRUCTURE {
                                STRUCTURED WITH {
                        ALTERNATIVE
                            DETERMINED BY field-to-be-used
                            USING message-id}}
                            WITH PER-BASIC-UNALIGNED}}
        WITH PER-BASIC-UNALIGNED}
```

D.5.3 Definición de codificación de una estructura de mensaje

D.5.3.1 El objeto de codificación "**`message1Encoding`**" especifica cómo se han de codificar los valores de "**`Message1`**":

- El campo "**`b`**" está presente si el campo "**`b-flag`**" contiene el valor "**`TRUE`**".
- El campo "**`c`**" está presente si el campo "**`c-len`**" no contiene el valor 0. El "**`c-len`**" gobierna también el número de elementos de "**`c`**".
- El campo "**`d`**" está presente si todavía hay octetos en una codificación del mensaje.

D.5.3.2 El objeto de codificación de "Message1" es:

```
message1Encoding #Message1 ::= {
        ENCODE STRUCTURE {
            b   b-encoding
                OPTIONAL-ENCODING {
                    PRESENCE
                        DETERMINED BY field-to-be-used
                        USING b-flag},
            c   octet-aligned-seq-of-with-ext-determinant{< c-len >},
            d   octet-aligned-seq-of-until-end-of-container
                                OPTIONAL-ENCODING USE-SET}
        WITH PER-BASIC-UNALIGNED}
```

D.5.4 Codificación del tipo secuencia "B"

D.5.4.1 Se inserta relleno de un bit entre los campos "**`b2`**" y "**`b3`**" ("**`aux-reserved`**"). La codificación de "**`B`**" se hace con alineación de octetos.

D.5.4.2 La codificación de "**B**" es:

```
b-encoding #B ::= {
        ENCODE STRUCTURE {
                -- Components
                b3 {
                        ALIGNED TO NEXT nibble
                        ENCODING {
                                ENCODING-SPACE
                                        SIZE 2
                                                MULTIPLE OF bit }}
                -- Structure
                STRUCTURED WITH {
                        ALIGNED TO NEXT octet
                        ENCODING-SPACE
                                SIZE self-delimiting-values
                                        MULTIPLE OF bit }}
        -- The rest
        WITH PER-BASIC-UNALIGNED}
```

D.5.5 Codificación de un tipo secuencia de (sequence-of) con alineación de octetos con un determinante de longitud

D.5.5.1 Uno de los tipos secuencia de (sequence-of) utilizados en el protocolo de legado tiene un determinante de longitud explícito.

D.5.5.2 La codificación se hace con alineación de octetos. La cuenta del número de elementos la determina el campo "**len**".

```
octet-aligned-seq-of-with-ext-determinant{< REFERENCE : len >} #REPETITION ::= {
        REPETITION-ENCODING {
            ALIGNED TO NEXT octet
            REPETITION-SPACE
                SIZE variable-with-determinant
                DETERMINED BY field-to-be-used
                USING len}}
```

D.5.6 Codificación de un tipo secuencia de (sequence-of) con alineación de octetos que continúa al final de la PDU

D.5.6.1 La codificación se hace con alineación de octetos. El número de elementos lo determina el final de la PDU.

D.5.6.2 El objeto de codificación es:

```
octet-aligned-seq-of-until-end-of-container #REPETITION ::= {
        REPETITION-ENCODING {
            ALIGNED TO NEXT octet
            REPETITION-SPACE
               SIZE variable-with-determinant
               DETERMINED BY container
               USING OUTER}}
```

D.5.7 Definiciones EDM

Las definiciones EDM son:

```
LegacyProtocol-EDM-Module {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module4(13)}
        ENCODING-DEFINITIONS ::=
        BEGIN

        EXPORTS LegacyProtocolEncodings;

        IMPORTS #LegacyProtocolMessages
        FROM    LegacyProtocol-ASN1-Module
        { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module4(11) };

        LegacyProtocolEncodings #ENCODINGS ::= {
                legacyProtocolMessagesEncoding      |
                message1Encoding }


        -- etc.

        END
```

D.5.8 Definiciones ELM

El ELM del protocolo de legado es:

```
LegacyProtocol-ELM-Module { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) elm-module4(12) }
        LINK-DEFINITIONS ::=
        BEGIN

        IMPORTS
                LegacyProtocolEncodings FROM LegacyProtocol-EDM-Module
                        { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module4(13) }
                #LegacyProtocolMessages FROM LegacyProtocol-ASN1-Module
                        { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module4(11) };

        ENCODE #LegacyProtocolMessages WITH LegacyProtocolEncodings
                COMPLETED BY PER-BASIC-UNALIGNED

        END
```

Anexo E

Soporte de las codificaciones Huffman

(Este anexo no es parte integrante de la presente Recomendación | Norma Internacional)

E.1 Las codificaciones Huffman son las codificaciones óptimas de un conjunto finito de valores enteros cuando se conoce la frecuencia con la que se transmitirá cada valor.

E.2 Las codificaciones son autodelimitantes (no se necesita determinante de longitud) y utilizan un pequeño número de bits para valores frecuentes y un número de bits mayor para valores menos frecuentes.

E.3 Hay muchas codificaciones Huffman posibles. Si, por ejemplo, se cambian en una de esas codificaciones todos los "1" (unos) a "0" (ceros) y viceversa se tiene una codificación Huffman diferente (pero igual de eficaz). Se pueden efectuar cambios más sutiles para producir otras codificaciones Huffman que son igual de convenientes.

E.4 Para que las codificaciones Huffman sean eficaces al aplicar los decodificadores es conveniente que, cuando se codifiquen valores enteros sucesivos en el mismo número de bits, esos bits definan valores enteros sucesivos al ser interpretados como una codificación de enteros positivos.

E.5 Se ha definido una codificación Huffman ECN que tiene esta propiedad, y se ha creado un macro Microsoft Word 97 que generará la sintaxis de un establecimiento de la correspondencia "MappingIntToBits" (véase 19.7) óptima y fácil de codificar.

E.6 Está disponible una versión del presente anexo con un botón de macro que capta la especificación de los valores enteros que se han de codificar y su frecuencia y genera en línea la especificación formal del establecimiento de la correspondencia conforme a la notación ECN. (La versión de este anexo con el macro asociado se puede obtener en el sitio web UIT http://www.itu.int/ITU-T/publications/recs.html Recomendación X.692 y en el sitio web ISO http://www.iso.ch/iso/en/ittf/PubliclyAvailableStandards/c034390_ISO_8825-3_2003(E)_Annex_E.html).

E.7 El texto que sigue contiene tres ejemplos de especificación Huffman ECN.

E.8 En la versión con el macro, haciendo un doble clic en el botón siguiente:

ECN Huffman

se añaden al texto las especificaciones del establecimiento de la correspondencia Huffman ECN.

E.9 El usuario de la versión con el macro quizá desee modificar la especificación de los valores cuya correspondencia se ha de establecer y sus frecuencias para ver las codificaciones producidas en los diferentes casos.

NOTA – En la versión con macros, una vez producidas las especificaciones de codificación, se pueden suprimir, se puede cambiar la especificación Huffman ECN y se puede hacer de nuevo clic en el botón de macro.

E.10 La sintaxis informal de una especificación Huffman ECN debe quedar clara con los ejemplos que vienen a continuación. Todas las líneas comienzan con un marcador de comentario ASN.1 ("--").

E.11 La primera línea (si se va a utilizar el macro) debe contener exactamente "ECN Huffman" precedido por dos guiones y un espacio, pero las líneas siguientes son insensibles a la escritura en mayúsculas/minúsculas y pueden contener más o menos espacios.

E.12 La segunda línea es necesaria y especifica los valores mayor y menor cuya correspondencia se va a establecer. La gama (límite superior menos límite inferior) se reduce a 1000, pero puede incluir valores negativos. No es preciso establecer la correspondencia de todos los valores de la gama.

E.13 Se dan porcentajes de valores únicos o de gamas de valores. No es necesario que los porcentajes se acumulen hasta el 100%, pero se da un aviso si no lo hacen.

E.14 La línea "REST" es opcional y contiene las frecuencias de cualesquiera valores de la gama no indicados explícitamente. Si falta, los valores cuya correspondencia se establece serán sólo los especificados de manera explícita.

E.15 La línea final es obligatoria y debe contener "End Definition" (en mayúsculas o minúsculas). La especificación de la codificación ECN formal es insertada (por el macro) después de esta línea.

E.15.1 El primer ejemplo es:

```
my-int-encoding1 #My-Special-1 ::=
{ USE  #BITS
      -- ECN Huffman
      -- RANGE (-1..10)
      -- –1 IS 20%
      -- 1 IS 25%
      -- 0 IS 15%
      -- (3..6) IS 10%
      -- Rest IS 2%
      -- End Definition
      -- Mappings produced by "ECN Public Domain Software for Huffman encodings, version 1"
        MAPPING TO BITS {
                –1 TO '11'B,
                0 ..  1 TO '01'B .. '10'B,
                2 TO '0000001'B ,
                3 ..  5 TO '0001'B .. '0011'B,
                6 TO '00001'B,
                7 ..  8 TO '0000010'B .. '0000011'B,
                9 ..  10 TO '00000000'B .. '00000001'B
        }
WITH  my-self-delim-bits-encoding }
```

E.15.2 El segundo ejemplo es :

```
my-int-encoding2 #My-Special-2 ::=
{ USE  #BITS
      -- ECN Huffman
      -- RANGE (-10..10)
      -- –10 IS 20%
      -- 1 IS 25%
      -- 5 IS 15%
      -- (7..10) is 10%
      -- End Definition
      -- Mappings produced by "ECN Public Domain Software for Huffman encodings, version 1"
        MAPPING TO BITS {
                –10 TO '11'B ,
                1 TO '10'B ,
                5 TO '01'B ,
                7 ..  10 TO '0000'B .. '0011'B
        }
WITH my-self-delim-bits-encoding }
```

E.15.3 El tercer ejemplo es:

```
my-int-encoding3 #My-Special-3 ::=
{ USE  #BITS
      -- ECN Huffman
      -- RANGE  (0..1000)
      -- (0..63) IS 100%
      -- REST IS 0%
      -- End Definition
      -- Mappings produced by "ECN Public Domain Software for Huffman encodings"
        MAPPING TO BITS {
                0 ..  62 TO '000001'B .. '111111'B,
                63 TO '0000001'B ,
                64 ..  150 TO '0000000110101001'B .. '0000000111111111'B,
                151 ..  1000 TO '0000000000000000'B .. '0000000101010001'B
        }
WITH my-self-delim-bits-encoding }
```

Anexo F

Información adicional sobre la notación de control de codificación (ECN)

(Este anexo no es parte integrante de la presente Recomendación | Norma Internacional)

Se puede encontrar información adicional sobre la notación de control de codificación y enlaces propios de la misma en el sitio web siguiente:

- http://asn1.elibel.tm.fr/ecn

Anexo G

Resumen de la notación ECN

(Este anexo no es parte integrante de la presente Recomendación | Norma Internacional)

G.1 Símbolos terminales

En esta Recomendación | Norma Internacional se utilizan los símbolos terminales que se indican a continuación.

G.1.1 Los elementos siguientes se definen en la cláusula 8:

anystringexceptnonecend
encodingobjectreference
encodingobjectsetreference
encodingclassreference
"::="
".."
"{"
"}"
"("
")"
","
"."
"|"
ALL
AS
BEGIN
BER
BITS
BY
CER
COMPLETED
DECODE
DER
DISTRIBUTION
ENCODE
ENCODE-DECODE
ENCODING-CLASS
ENCODING-DEFINITIONS
END
EXCEPT
EXPORTS
FALSE
FIELDS
FROM
GENERATES
IF
IMPORTS
IN
LINK-DEFINITIONS
MAPPING
MAX
MIN
MINUS-INFINITY
NON-ECN-BEGIN
NON-ECN-END
NULL
OPTIONAL-ENCODING
OPTIONS
ORDERED
OUTER
PER-BASIC-ALIGNED
PER-BASIC-UNALIGNED
PER-CANONICAL-ALIGNED
PER-CANONICAL-UNALIGNED
PLUS-INFINITY
REFERENCE
REMAINDER
RENAMES
SIZE
STRUCTURE
STRUCTURED
TO
TRANSFORMS
TRUE
UNION
USE
USE-SET
VALUES
WITH

G.1.2 El elemento siguiente se define en el anexo A:

REFERENCE

G.1.3 Los elementos siguientes se definen en la Rec. UIT-T X.680 | ISO/CEI 8824-1:

bstring
cstring
hstring
identifier
modulereference
number
realnumber
typereference
"-"
";"
":"
ALL
EXCEPT
EXPORTS
FALSE
FROM
IMPORTS
MINUS-INFINITY
NULL
PLUS-INFINITY
TRUE

G.1.4 Los elementos siguientes se definen en la Rec. UIT-T X.681 | ISO/CEI 8824-2:

word
valuefieldreference
valuesetfieldreference

G.1.5 Los elementos siguientes se definen en la Rec. UIT-T X.683 | ISO/CEI 8824-4:

"{<"
">}"

G.2 Producciones

G.2.1 Las producciones siguientes se utilizan en esta Rec. | Norma Internacional, con los elementos definidos en G.1 como símbolos terminales:

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

ELMModuleBody ::=
        Imports ?
        EncodingApplicationList

EncodingApplicationList ::=
        EncodingApplication
        EncodingApplicationList ?

EncodingApplication ::=
        ENCODE
        SimpleDefinedEncodingClass "," +
        CombinedEncodings

CombinedEncodings ::=
        WITH
        PrimaryEncodings
        CompletionClause ?

CompletionClause ::=
        COMPLETED BY
        SecondaryEncodings

PrimaryEncodings ::= EncodingObjectSet

SecondaryEncodings ::= EncodingObjectSet

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

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

EDMAssignmentList ::=
        EDMAssignment
        EDMAssignmentList ?

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

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

ExplicitGenerationList ::=
        ExplicitGeneration
        ExplicitGenerationList ?

ExplicitGeneration ::=
        OptionalNameChanges
        FROM GlobalModuleReference

OptionalNameChanges ::=
        NameChanges |  GENERATES

NameChanges ::= NameChange NameChanges ?

NameChange ::=
        OriginalClassName
        AS
        NewClassName
        IN
        NameChangeDomain

OriginalClassName ::= SimpleDefinedEncodingClass | BuiltinEncodingClassReference

NewClassName ::= encodingclassreference

NameChangeDomain ::=
        IncludedRegions
        Exception ?

Exception ::=
        EXCEPT
        ExcludedRegions

IncludedRegions ::=
        ALL | RegionList

ExcludedRegions ::= RegionList

RegionList ::=
        Region "," +

Region ::=
        SimpleDefinedEncodingClass |
        ComponentReference

ComponentReference ::=
        SimpleDefinedEncodingClass
        "."
        ComponentIdList

ComponentIdList ::=
        identifier "." +

EncodingClassAssignment ::=
        encodingclassreference
        "::="
        EncodingClass

EncodingClass ::=
        BuiltinEncodingClassReference        |
        EncodingStructure

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

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

```
EncodingObjectSet ::=
        DefinedOrBuiltinEncodingObjectSet |
        EncodingObjectSetSpec

EncodingStructure ::=
        TaggedStructure             |
        UntaggedStructure

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

UntaggedStructure ::=
            DefinedEncodingClass
        |   EncodingStructureField
        |   EncodingStructureDefn

TagClass ::=
            DefinedEncodingClass
        |   TagClassReference

TagValue ::=
        "(" number ")"

EncodingStructureDefn ::=
            AlternativesStructure
        |   RepetitionStructure
        |   ConcatenationStructure

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

AlternativesClass ::=
            DefinedEncodingClass
        |   AlternativesClassReference

NamedFields ::= NamedField "," +

NamedField ::=
        identifier
        EncodingStructure

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

RepetitionClass ::=
            DefinedEncodingClass
        |   RepetitionClassReference

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

ConcatenationClass ::=
            DefinedEncodingClass
        |   ConcatenationClassReference

ConcatComponents ::=
        ConcatComponent "," *

ConcatComponent ::=
        NamedField
        ConcatComponentPresence ?
```

```
ConcatComponentPresence ::=
        OPTIONAL-ENCODING
        OptionalClass

OptionalClass ::=
            DefinedEncodingClass
        |   OptionalityClassReference

DefinedEncodingClass ::=
            encodingclassreference
        |   ExternalEncodingClassReference
        |   ParameterizedEncodingClass

DefinedOrBuiltinEncodingClass ::=
            DefinedEncodingClass
        |   BuiltinEncodingClassReference

DefinedEncodingObject ::=
            encodingobjectreference
        |   ExternalEncodingObjectReference
        |   ParameterizedEncodingObject

DefinedEncodingObjectSet ::=
            encodingobjectsetreference
        |   ExternalEncodingObjectSetReference
        |   ParameterizedEncodingObjectSet

DefinedOrBuiltinEncodingObjectSet ::=
            DefinedEncodingObjectSet
        |   BuiltinEncodingObjectSetReference

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

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

ExternalEncodingObjectReference ::=
        modulereference "." encodingobjectreference

ExternalEncodingObjectSetReference ::=
        modulereference "." encodingobjectsetreference

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

EncodingObjects ::=
            DefinedEncodingObject
        |   DefinedEncodingObjectSet

UnionMark ::=
        "|" |
        UNION

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

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

EncodeByValueMapping ::=
        "{"
        USE
        DefinedOrBuiltinEncodingClass
        MAPPING
        ValueMapping
        WITH
        ValueMappingEncodingObjects
        "}"

ValueMappingEncodingObjects ::=
            EncodingObject
        |   DefinedOrBuiltinEncodingObjectSet

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

SpecForEncoding ::= EncodingObject

SpecForDecoders ::= EncodingObject

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

EncodingOptionsList ::= OrderedEncodingObjectList

AlternativesEncodingObject ::= EncodingObject

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

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

StructureEncoding ::=
        STRUCTURED WITH
        TagEncoding ?
        EncodingOrUseSet

ComponentEncodingList ::=
        ComponentEncoding "," *

ComponentEncoding ::=
            NonOptionalComponentEncodingSpec
        |   OptionalComponentEncodingSpec

NonOptionalComponentEncodingSpec ::=
        identifier ?
        TagAndElementEncoding
```

```
OptionalComponentEncodingSpec ::=
        identifier
        TagAndElementEncoding
        OPTIONAL-ENCODING
        OptionalEncoding

TagAndElementEncoding ::=
        TagEncoding ?
        EncodingOrUseSet

TagEncoding ::= "[" EncodingOrUseSet "]"

OptionalEncoding ::= EncodingOrUseSet

EncodingOrUseSet ::=
            EncodingObject
        |   USE-SET

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

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

AlternativesClassReference ::=
            #ALTERNATIVES
        |   #CHOICE

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

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

OptionalityClassReference ::=
            #OPTIONAL

TagClassReference ::=
            #TAG

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

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

Bounds ::= "(" EffectiveRange ")"

EffectiveRange ::=
            MinMax
        |   Fixed

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

SizeEffectiveRange ::=
        "(" EffectiveRange ")"

MinMax ::=
        ValueOrMin
        ".."
        ValueOrMax

ValueOrMin      ::=
            SignedNumber
        |   MIN
```

```
ValueOrMax    ::=
              SignedNumber
          |   MAX

Fixed ::=  SignedNumber

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

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

MappedValues ::=
          MappedValue1
          TO
          MappedValue2

MappedValue1 ::= Value

MappedValue2 ::= Value

MappingByMatchingFields ::=
          FIELDS

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

OrderedTransformList ::= Transform "," +

Transform ::= EncodingObject

MappingByAbstractValueOrdering ::=
          ORDERED VALUES

MappingByValueDistribution ::=
          DISTRIBUTION
          "{"
          Distribution "," +
          "}"

Distribution ::=
          SelectedValues
          TO
          identifier

SelectedValues ::=
              SelectedValue
          |   DistributionRange
          |   REMAINDER

DistributionRange ::=
          DistributionRangeValue1
          ".."
          DistributionRangeValue2

SelectedValue ::= SignedNumber

DistributionRangeValue1 ::= SignedNumber

DistributionRangeValue2 ::= SignedNumber

MappingIntToBits ::=
          TO BITS
          "{"
          MappedIntToBits "," +
          "}"
```

```
MappedIntToBits ::=
        SingleIntValMap
    |   IntValRangeMap

SingleIntValMap ::=
        IntValue
        TO
        BitValue

IntValue ::= SignedNumber

BitValue ::=
        bstring   |
        hstring

IntValRangeMap ::=
        IntRange
        TO
        BitRange

IntRange ::=
        IntRangeValue1
        ".."
        IntRangeValue2

BitRange ::=
        BitRangeValue1
        ".."
        BitRangeValue2

IntRangeValue1 ::= SignedNumber

IntRangeValue2 ::= SignedNumber

BitRangeValue1 ::=
        bstring |
        hstring

BitRangeValue2 ::=
        bstring |
        hstring
```

G.2.2 Las producciones siguientes se definen en la Rec. UIT-T X.680 | ISO/CEI 8824-1, modificada por el anexo A, con los elementos definidos en G.1 como símbolos terminales:

NOTA – En la ECN no se permiten producciones especiales.

```
ModuleIdentifier ::=
    modulereference
    DefinitiveIdentifier ?

DefinitiveIdentifier ::=
        "{" DefinitiveObjIdComponentList "}"

DefinitiveObjIdComponentList ::=
        DefinitiveObjIdComponent
    |   DefinitiveObjIdComponent DefinitiveObjIdComponentList

DefinitiveObjIdComponent ::=
        NameForm
    |   DefinitiveNumberForm
    |   DefinitiveNameAndNumberForm

NameForm ::= identifier

DefinitiveNumberForm ::= number

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

Exports ::=
        EXPORTS SymbolsExported? ";" |
        EXPORTS ALL ";"
```

```
SymbolsExported ::= SymbolList

Imports ::= IMPORTS SymbolsImported? ";"

SymbolsImported ::= SymbolsFromModuleList

SymbolsFromModuleList ::=
    SymbolsFromModule |
    SymbolsFromModuleList  SymbolsFromModule

SymbolsFromModule ::=
    SymbolList
            FROM
            GlobalModuleReference

GlobalModuleReference ::=
    modulereference AssignedIdentifier

AssignedIdentifier ::=
            DefinitiveIdentifier
        |   empty

SymbolList ::=
    Symbol                  |
    SymbolList "," Symbol

Symbol ::=
            Reference
        |   ParameterizedReference
        |   BuiltinEncodingClassReference

Reference ::=
            encodingclassreference
        |   ExternalEncodingClassReference
        |   encodingobjectreference
        |   encodingobjectsetreference

Value ::=
            BuiltinValue
        |   ReferencedValue
        |   ObjectClassFieldValue

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

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

```
BooleanValue ::=
        TRUE
    |   FALSE

CharacterStringValue ::=
        RestrictedCharacterStringValue
    |   UnrestrictedCharacterStringValue

RestrictedCharacterStringValue ::=
        cstring
    |   CharacterStringList
    |   Quadruple
    |   Tuple

CharacterStringList  ::=  "{" CharSyms "}"

CharSyms  ::=
        CharsDefn
    |   CharSyms "," CharsDefn

CharsDefn  ::=
        cstring
    |   Quadruple
    |   Tuple
    |   AbsoluteCharReference

Quadruple    ::= "{"  Group  ","  Plane  ","  Row  ","  Cell "}"
Group      ::= number
Plane      ::= number
Row        ::= number
Cell       ::= number

Tuple ::= "{" TableColumn "," TableRow "}"
TableColumn  ::= number
TableRow       ::= number

AbsoluteCharReference ::=
        ModuleIdentifier
        "."
        valuereference

UnrestrictedCharacterStringValue ::= SequenceValue

ChoiceValue ::= identifier ":" Value

EmbeddedPDVValue ::= SequenceValue

EnumeratedValue ::= identifier

ExternalValue ::= SequenceValue

IntegerValue ::=
        SignedNumber
    |   identifier

SignedNumber ::=
        number  |
        "-" number

NullValue ::= NULL

ObjectIdentifierValue ::=
        "{" ObjIdComponentsList "}"
    |   "{" DefinedValue ObjIdComponentsList "}"

ObjIdComponentsList ::=
        ObjIdComponents                 |
        ObjIdComponents ObjIdComponentsList
```

```
ObjIdComponents ::=
        NameForm        |
        NumberForm      |
        NameAndNumberForm

NameForm ::= identifier

NumberForm ::=
          number
      |   DefinedValue

NameAndNumberForm ::= identifier "(" NumberForm ")"

OctetStringValue ::=
      bstring     |
      hstring

RealValue ::=
          NumericRealValue
      |   SpecialRealValue

NumericRealValue ::=
          0
      |   realnumber
      |   "-" realnumber
      |   SequenceValue

SpecialRealValue ::=
          PLUS-INFINITY
      |   MINUS-INFINITY

RelativeOIDValue ::= "{" RelativeOidComponentsList "}"

RelativeOidComponentsList ::=
    RelativeOidComponents
|   RelativeOidComponents RelativeOidComponentsList

RelativeOidComponents ::=
    NumberForm
|   NameAndNumberForm
|   DefinedValue

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

ComponentValueList ::=
            NamedValue
        |   ComponentValueList "," NamedValue

NamedValue ::=
          identifier Value

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

ValueList ::=
        Value
    |   ValueList "," Value

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

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

```
ValueSet ::= "{" ElementSetSpecs "}"

ElementSetSpecs ::=
        RootElementSetSpec
    | RootElementSetSpec "," "..."
    | "..." "," AdditionalElementSetSpec
    | RootElementSetSpec "," "..." "," AdditionalElementSetSpec

RootElementSetSpec ::= ElementSetSpec

ElementSetSpec ::=
        Unions
    | ALL Exclusions


Exclusions ::= EXCEPT Elements

Unions ::=
        Intersections
    | UElems UnionMark Intersections

UElems ::= Unions

Intersections ::=
        IntersectionElements
    | IElems IntersectionMark IntersectionElements

IntersectionElements ::= Elements | Elems Exclusions

UnionMark ::=
        "|"   |
        UNION

Elements ::=
        SubtypeElements
    | ObjectSetElements
    |-"(" ElementSetSpec ")"

SubtypeElements ::=
        SingleValue
    | ContainedSubtype
    |-ValueRange
    | PermittedAlphabet
    | SizeConstraint
    | TypeConstraint
    | InnerTypeConstraints

SingleValue ::= Value
```

G.2.3 Las producciones siguientes se definen en la Rec. UIT-T X.681 | ISO/CEI 8824-2, modificada por el anexo B, con los elementos definidos en G.1 como símbolos terminales:

```
DefinedSyntax ::= "{" DefinedSyntaxList ? "}"

DefinedSyntaxList ::= DefinedSyntaxToken DefinedSyntaxList ?

DefinedSyntaxToken ::=
      Literal
  |   Setting
Literal ::=
      word
  |   ","
```

```
Setting ::=
        Value
    |   ValueSet
    |   OrderedValueList
    |   EncodingObject
    |   EncodingObjectSet
    |   OrderedEncodingObjectList
    |   DefinedOrBuiltinEncodingClass
    |   OUTER

OrderedValueList ::= "{" Value "," + "}"

OrderedEncodingObjectList ::= "{" EncodingObject "," + "}"

InstanceOfValue ::= Value

EncodingClassFieldType ::=
    DefinedEncodingClass
    "."
    FieldName

FieldName ::= PrimitiveFieldName "." +

PrimitiveFieldName ::=
        valuefieldreference
    |   valuesetfieldreference
    |   orderedvaluelistfieldreference
```

G.2.4 Las producciones siguientes se definen en la Rec. UIT-T X.683 | ISO/CEI 8824-4, modificada por el anexo C, con los elementos definidos en G.1 como símbolos terminales:

```
ParameterizedAssignment ::=
    ParameterizedEncodingObjectAssignment
|   ParameterizedEncodingClassAssignment
|   ParameterizedEncodingObjectSetAssignment

ParameterizedEncodingObjectAssignment ::=
    encodingobjectreference
    ParameterList
    DefinedOrBuiltinEncodingClass
    "::="
    EncodingObject

ParameterizedEncodingClassAssignment ::=
    encodingclassreference
    ParameterList
    "::="
    EncodingClass

ParameterizedEncodingObjectSetAssignment ::=
    encodingobjectsetreference
    ParameterList
    #ENCODINGS
    "::="
    EncodingObjectSet

ParameterList ::= "{<" Parameter "," + ">}"

Parameter ::=
            ParamGovernor ":" DummyReference
        |   DummyReference

ParamGovernor ::=
            Governor
        |   DummyGovernor

Governor ::=
            EncodingClassFieldType
        |   REFERENCE
        |   DefinedOrBuiltinEncodingClass
        |   #ENCODINGS
```

```
~~DummyGovernor ::= DummyReference~~

DummyReference ::=
          encodingclassreference
        | valuereference
        | typereference
        | identifier
        | encodingobjectreference
        | encodingobjectsetreference

ParameterizedReference ::=
          Reference
        | Reference "{<"  ">}"

ParameterizedEncodingObject ::=
        SimpleDefinedEncodingObject
        ActualParameterList

SimpleDefinedEncodingObject ::=
          ExternalEncodingObjectReference
        | encodingobjectreference

ParameterizedEncodingObjectSet ::=
        SimpleDefinedEncodingObjectSet
        ActualParameterList

SimpleDefinedEncodingObjectSet ::=
          ExternalEncodingObjectSetReference
        | encodingobjectsetreference

ParameterizedEncodingClass ::=
        SimpleDefinedEncodingClass
        ActualParameterList

SimpleDefinedEncodingClass ::=
          ExternalEncodingClassReference
        | encodingclassreference

ActualParameterList ::= "{<" ActualParameter "," + ">}"

ActualParameter ::=
          Value
        | ValueSet
        | OrderedValueList
        | DefinedOrBuiltinEncodingClass
        | EncodingObject
        | EncodingObjectSet
        | OrderedEncodingObjectList
        | identifier
        | STRUCTURE
        | OUTER
```

SERIES DE RECOMENDACIONES DEL UIT-T

Serie A	Organización del trabajo del UIT-T
Serie B	Medios de expresión: definiciones, símbolos, clasificación
Serie C	Estadísticas generales de telecomunicaciones
Serie D	Principios generales de tarificación
Serie E	Explotación general de la red, servicio telefónico, explotación del servicio y factores humanos
Serie F	Servicios de telecomunicación no telefónicos
Serie G	Sistemas y medios de transmisión, sistemas y redes digitales
Serie H	Sistemas audiovisuales y multimedios
Serie I	Red digital de servicios integrados
Serie J	Redes de cable y transmisión de programas radiofónicos y televisivos, y de otras señales multimedios
Serie K	Protección contra las interferencias
Serie L	Construcción, instalación y protección de los cables y otros elementos de planta exterior
Serie M	RGT y mantenimiento de redes: sistemas de transmisión, circuitos telefónicos, telegrafía, facsímil y circuitos arrendados internacionales
Serie N	Mantenimiento: circuitos internacionales para transmisiones radiofónicas y de televisión
Serie O	Especificaciones de los aparatos de medida
Serie P	Calidad de transmisión telefónica, instalaciones telefónicas y redes locales
Serie Q	Conmutación y señalización
Serie R	Transmisión telegráfica
Serie S	Equipos terminales para servicios de telegrafía
Serie T	Terminales para servicios de telemática
Serie U	Conmutación telegráfica
Serie V	Comunicación de datos por la red telefónica
Serie X	**Redes de datos y comunicación entre sistemas abiertos**
Serie Y	Infraestructura mundial de la información y aspectos del protocolo Internet
Serie Z	Lenguajes y aspectos generales de soporte lógico para sistemas de telecomunicación

Ginebra, 2003