UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

UIT-T X.680

(12/97)

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

SÉRIE X: RÉSEAUX POUR DONNÉES ET COMMUNICATION ENTRE SYSTÈMES OUVERTS

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

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

Recommandation UIT-T X.680

(Antérieurement Recommandation du CCITT)

RECOMMANDATIONS UIT-T DE LA SÉRIE X

RÉSEAUX POUR DONNÉES ET COMMUNICATION ENTRE SYSTÈMES OUVERTS

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

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

NORME INTERNATIONALE 8824-1

RECOMMANDATION UIT-T X.680

TECHNOLOGIES DE L'INFORMATION – NOTATION DE SYNTAXE ABSTRAITE NUMÉRO UN: SPÉCIFICATION DE LA NOTATION DE BASE

Résumé

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

La notation ASN.1 peut être utilisée chaque fois que cela est nécessaire pour définir la syntaxe abstraite d'informations, sans que cela impose une contrainte quelconque sur la manière de coder cette information en transmission. Cette notation s'applique notamment mais pas exclusivement aux protocoles de la couche application.

Source

La Recommandation X.680 de l'UIT-T a été approuvée le 12 décembre 1997. Un texte identique est publié comme Norme internationale ISO/CEI 8824-1.

AVANT-PROPOS

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

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

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

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

NOTE

Dans la présente Recommandation, le terme *exploitation reconnue (ER)* désigne tout particulier, toute entreprise, toute société ou tout organisme public qui exploite un service de correspondance publique. Les termes *Administration*, *ER* et *correspondance publique* sont définis dans la *Constitution de l'UIT (Genève, 1992)*.

DROITS DE PROPRIÉTÉ INTELLECTUELLE

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

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

TABLE DES MATIÈRES

Introduction

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

Bien que cette notation normalisée soit définie dans le cadre général de l'OSI, elle peut servir à de nombreuses autres fins. Dans les couches inférieures du Modèle de référence de base de l'OSI (Rec. UIT-T X.200 | ISO/CEI 7498-1) et dans de nombreuses autres architectures de protocoles, chaque message est spécifié comme la valeur binaire d'une séquence d'octets. Dans la couche présentation de l'OSI (voir la Rec. UIT-T X.216 | ISO/CEI 8822), la nature des paramètres données d'utilisateur est différente. Toutefois, les spécifications de la couche application imposent la définition de types de donnée relativement complexes pour véhiculer leurs messages, indépendamment de leur représentation binaire. Pour spécifier les types de donnée, ces Recommandations nécessitent une notation qui ne détermine pas nécessairement la représentation de chaque valeur. Une telle notation doit être complétée par la spécification d'un ou plusieurs algorithmes appelés règles de codage qui déterminent les valeurs des octets des couches inférieures véhiculant les données de la couche application (appelées syntaxes de transfert). Le protocole de la couche présentation de l'OSI (voir la Rec. UIT-T X.226 | ISO/CEI 8823-1) peut négocier les syntaxes de transfert (codages) à utiliser.

Hors du contexte de l'OSI, l'utilité de la notion de valeur abstraite d'une classe quelconque (par exemple une image donnée à 256 couleurs) détachée de toute considération de codage particulier est de plus en plus reconnue, ce qui, pour interpréter correctement la représentation binaire de la valeur, nécessiterait de connaître (en général d'après le contexte) le type (classe) de la valeur représentée, ainsi que le mécanisme de codage utilisé. L'identification d'un type constitue donc une partie importante de la présente Recommandation | Norme internationale.

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

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

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

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

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

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

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

Quatre classes d'étiquettes sont spécifiées dans la notation.

La première est la classe **universelle**. Les étiquettes de la classe universelle ne sont utilisées que selon les spécifications de la présente Recommandation | Norme internationale, chaque étiquette étant affectée:

a) soit à un type unique;

b) soit à un mécanisme de structuration.

Les utilisateurs de la présente notation ne sont pas autorisés à spécifier explicitement des étiquettes de la classe universelle dans leurs déclarations ASN.1, ces étiquettes étant prédéfinies et ne pouvant être explicitement spécifiées que dans la présente Recommandation | Norme internationale.

Les trois autres classes d'étiquette sont la classe **application**, la classe **privée** et la classe **propre au contexte**. Aucune différence d'utilisation formelle n'existe entre ces trois classes. Là où une étiquette de la classe application est utilisée, il est généralement possible d'utiliser à la place une étiquette de la classe privée ou de la classe propre au contexte, au choix de l'utilisateur. La présence des trois classes est en grande partie due à des raisons historiques, mais le paragraphe 2.12 de l'Annexe C fournit des indications sur la manière dont ces différentes classes sont généralement utilisées.

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

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

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

Les articles 10 à 31 inclus définissent les types simples pris en charge par la notation ASN.1 et spécifient la notation à utiliser pour faire référence à des types simples et pour définir de nouveaux types au moyen de ces types simples. Ils spécifient également la notation à utiliser pour spécifier les valeurs appartenant à des types définis en ASN.1.

Les articles 32 à 33 inclus définissent les types pris en charge par la notation ASN.1 pour véhiculer le codage complet des types ASN.1.

Les articles 34 à 39 inclus définissent les types de chaîne de caractères.

Les articles 40 à 43 inclus définissent certains types considérés comme étant d'utilité générale mais qui ne nécessitent aucune règle de codage supplémentaire.

Les articles 44 et 48 définissent une notation qui permet d'obtenir des sous-types à partir des valeurs d'un type parent.

L'Annexe A, qui fait partie intégrante de la présente Recommandation | Norme internationale, donne des indications sur la manière dont les utilisateurs de la présente Recommandation | Norme internationale peuvent faire référence aux types et valeurs ASN.1 définis au moyen de la Rec. X.208 du CCITT | ISO/CEI 8824.

L'Annexe B, qui fait partie intégrante de la présente Recommandation | Norme internationale, récapitule les valeurs d'identificateurs d'objets et de descripteurs d'objets affectées dans le cadre de la présente Recommandation | Norme internationale.

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

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

L'Annexe E, qui ne fait pas partie intégrante de la présente Recommandation | Norme internationale, décrit les caractéristiques de la version précédente de la notation ASN.1 aujourd'hui obsolètes.

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

L'Annexe G, qui ne fait pas partie intégrante de la présente Recommandation | Norme internationale, fournit un résumé de la notation ASN.1 en utilisant la notation de l'article 5.

NORME INTERNATIONALE

RECOMMANDATION UIT-T

TECHNOLOGIES DE L'INFORMATION – NOTATION DE SYNTAXE ABSTRAITE NUMÉRO UN: SPÉCIFICATION DE LA NOTATION DE BASE

1 Domaine d'application

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

La présente Recommandation | Norme internationale:

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

La notation ASN.1 peut être utilisée chaque fois qu'il est nécessaire de définir la syntaxe abstraite d'informations. Elle est en particulier applicable, mais non exclusivement, aux protocoles d'application.

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

2 Références normatives

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

2.1 Recommandations | Normes internationales identiques

- Recommandation UIT-T X.200 (1994) | ISO/CEI 7498-1:1994 *Technologies de l'information – Interconnexion des systèmes ouverts – Modèle de référence de base: le modèle de référence de base*.
- Recommandation UIT-T X.216 (1994) | ISO/CEI 8822:1994 *Technologies de l'information – Interconnexion des systèmes ouverts – Définition du service de présentation*.
- Recommandation UIT-T X.226 (1994) | ISO/CEI 8823-1:1994 *Technologies de l'information – Interconnexion des systèmes ouverts – Protocole de présentation en mode connexion: spécification du protocole*.
- Recommandation UIT-T X.660 (1992)/Amd.2 (1997) | ISO/CEI 9834-1:1993/Amd.2:1998: *Technologies de l'information – Interconnexion des systèmes ouverts – Procédures pour le fonctionnement des autorités d'enregistrement OSI: Procédures générales* (plus Amendements 1 et 2).

- Recommandation UIT-T X.681 (1997) | ISO/CEI 8824-2:1998 *Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification des objets informationnels*.
- Recommandation UIT-T X.682 (1997) | ISO/CEI 8824-3:1998 *Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification des contraintes*.
- Recommandation UIT-T X.683 (1997) | ISO/CEI 8824-4:1998 *Technologies de l'information – Notation de syntaxe abstraite numéro un: paramétrage des spécifications de la notation de syntaxe abstraite numéro un*.
- Recommandation UIT-T X.690 (1997) | ISO/CEI 8825-1:1998 *Technologies de l'information – Règles de codage ASN.1: spécification des règles de codage de base, des règles de codage canoniques et des règles de codage distinctives*.
- Recommandation UIT-T X.691 (1997) | ISO/CEI 8825-2:1998 *Technologies de l'information – Règles de codage ASN.1: spécification des règles de codage compact*.

2.2 Paires de Recommandations | Normes internationales équivalentes par leur contenu technique

- Recommandation X.208 du CCITT (1988), *Spécification de la syntaxe abstraite numéro 1 (ASN.1)*.

  ISO/CEI 8824:1990, *Technologies de l'information – Interconnexion de systèmes ouverts – Spécification de la notation de syntaxe abstraite numéro 1 (ASN.1)*.

2.3 Autres références

- Recommandation T.61 du CCITT (1988), *Répertoire de caractères et jeux de caractères codés pour le service international télex*.
- Recommandation T.100 du CCITT (1988), *Echange international d'informations pour le vidéotex interactif*.
- Recommandation UIT-T T.101 (1994), *Interfonctionnement international pour les services vidéotex*.
- ISO *Registre international des jeux de caractères codés à utiliser avec une séquence d'échappement.*
- ISO/CEI 646:1991, *Technologies de l'information – Jeux ISO de caractères codés à 7 éléments pour l'échange d'informations*.
- ISO/CEI 2022:1994, *Technologies de l'information – Structure de code de caractères et techniques d'extension*.
- ISO 6523:1984, *Echange de données – Structures pour l'identification des organisations*.
- ISO/CEI 7350:1991, *Technologies de l'information – Enregistrement des répertoires de caractères graphiques de l'ISO 10367*.
- ISO 8601:1988, *Eléments de données et formats d'échange – Echange d'information – Représentation de la date et de l'heure*.
- ISO/CEI 10646-1:1993, *Technologies de l'information – Jeu universel de caractères à plusieurs octets – Partie 1: Architecture et table multilingue*.
- ISO/CEI 10646-1:1993/Amd.2:1998*, Technologies de l'information – Jeu universel de caractères à plusieurs octets – Partie 1: Architecture et table multilingue – Amendement 2: Format de transformation UCS 8 (UTF-8).*

3 Définitions

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

3.1 Spécification des objets informationnels

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

a) objet informationnel;

b) classe d'objets informationnels;

c) ensemble d'objets informationnels;

d) type instance-de;

e) type champ de classe d'objets.

3.2 Spécification des contraintes

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

a) contrainte relationnelle entre composants;

b) contrainte tabulaire.

3.3 Paramétrage des spécifications ASN.1

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

a) type paramétré;

b) valeur paramétrée.

3.4 Définition du service de présentation

La présente Recommandation | Norme internationale utilise les termes suivants, définis dans la Rec. UIT-T X.216 | ISO/CEI 8822:

a) (une) syntaxe abstraite;

b) nom de syntaxe abstraite;

c) ensemble contextuel défini;

d) valeur de donnée de présentation;

e) (une) syntaxe de transfert;

f) nom de syntaxe de transfert.

3.5 Spécification du protocole de présentation

La présente Recommandation | Norme internationale utilise le terme suivant, défini dans la Rec. UIT-T X.226 | ISO/CEI 8823-1:

– identificateur de contexte de présentation.

3.6 Structure pour l'identification des organisations

La présente Recommandation | Norme internationale utilise les termes suivants, définis dans la norme ISO 6523:

a) organisme émetteur;

b) code d'organisation;

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

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

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

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

b) cellule;

c) caractère de combinaison;

d) symbole graphique;

e) groupe;

f) sous-ensemble limité;

g) plan;

h) rangée;

i) sous-ensemble sélectionné.

3.8 Définitions supplémentaires

3.8.1 caractère abstrait: ensemble de l'information associée à une cellule dans une table définissant un répertoire de caractères.

NOTE – Cette information comprend normalement tout ou partie des éléments suivants:

a) un symbole graphique;

b) le nom du caractère;

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

3.8.2 valeur abstraite: valeur dont la définition est basée uniquement sur le type, indépendamment de la manière dont elle est représentée par une règle de codage quelconque.

NOTE – L'utilisation de l'expression «valeur abstraite» suppose souvent que l'entité décrite varie probablement selon les règles de codage utilisées.

3.8.3 jeu de caractères ASN.1: jeu de caractères spécifié à l'article 10 et utilisé en notation ASN.1.

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

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

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

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

NOTE – Lorsqu'il est nécessaire de véhiculer des codages enchâssés d'une valeur abstraite, l'utilisation de valeurs de donnée de présentation enchâssées constituera généralement un mécanisme plus souple que le type chaîne binaire pour annoncer la nature des codages ou la négocier.

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

3.8.8 caractère: élément d'un ensemble utilisé pour l'organisation, la commande ou la représentation des données.

NOTE – Ceci implique par exemple que le caractère de combinaison «accent aigu» et la minuscule «e» constituent deux caractères de la grille française ISO 646, et non pas un caractère unique «é».

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

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

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

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

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

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

3.8.14 type composant: un des types indiqués en référence dans une déclaration de type «CHOICE», «SET», «SEQUENCE», «SET OF», ou «SEQUENCE OF».

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

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

NOTE – NEWLINE (nouvelle ligne) et TAB (tabulation) sont des exemples de caractères de contrôle qui se sont vus affecter des fonctions de formatage dans un environnement d'édition. DLE (échappement de transmission) est un exemple de caractère de contrôle qui s'est vu affecter une fonction dans un environnement de communication.

3.8.17 **temps universel coordonné** (**UTC**, ***Coordinated Universal Time***): échelle de temps conservée par le Bureau international de l'heure, et servant de base à la diffusion coordonnée des fréquences standards et des signaux horaires.

NOTE 1 – L'origine de cette définition est la Recommandation 460-2 du Comité consultatif international des radiocommunications (CCIR). Le CCIR a également défini le sigle UTC du temps universel coordonné.

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

3.8.18 **élément**: membre d'une classe d'éléments, distinguable de tous les autres éléments de cette classe.

3.8.19 **classe d'éléments**: type (dont les éléments sont ses valeurs) ou classe d'objets informationnels (dont les éléments sont tous les objets possibles de cette classe).

3.8.20 **ensemble d'éléments**: un ou plusieurs éléments d'une même classe.

3.8.21 **type valeur de donnée de présentation enchâssée (embedded-pdv type)**: type dont l'ensemble des valeurs est la réunion des ensembles de valeurs dans toutes les syntaxes abstraites possibles. Ce type fait partie d'une spécification ASN.1 qui véhicule une valeur dont le type peut être défini extérieurement à cette spécification ASN.1. Il comporte également un identificateur du type de la valeur véhiculée ainsi qu'un identificateur de la règle de codage utilisée pour coder la valeur.

3.8.22 **codage**: séquence binaire résultant de l'application d'un ensemble de règles de codage à une valeur d'une syntaxe abstraite donnée.

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

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

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

3.8.25 **addition d'extension**: une des notations ajoutées à une série d'extensions. Pour les types ensemble, séquence et choix, une addition d'extension est constituée d'un seul groupe d'additions d'extension ou d'un seul composant. Pour les types énumérés, il s'agit de l'addition d'une seule énumération supplémentaire. Pour une contrainte, il s'agit de l'addition d'un élément sous-type.

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

3.8.26 **groupe d'additions d'extension**: un ou plusieurs composants d'un type ensemble, séquence ou choix apparaissant entre des crochets de version. Un groupe d'additions d'extension est utilisé afin d'identifier clairement les composants de types ensemble, séquence ou choix qui ont été ajoutés dans une version particulière d'un module ASN.1.

3.8.27 **type d'addition d'extension**: type contenu au sein d'un groupe d'additions d'extension ou composant simple qui est lui-même une addition d'extension (dans ce cas, le type n'est pas contenu dans un groupe d'additions d'extension).

3.8.28 **contrainte extensible**: contrainte de sous-typage contenant un marqueur d'extension.

3.8.29 **point d'insertion d'extension**: position au niveau de laquelle des additions d'extension sont insérées au sein d'une définition de type. Cette position correspond à la fin de la notation du type immédiatement précédent dans la série d'extensions, s'il existe une seule occurrence de points de suspension dans la définition du type, ou immédiatement avant les deuxièmes points de suspension s'il existe une paire de marqueurs d'extension dans la définition du type.

3.8.30 marqueur d'extension: indicateur syntaxique (points de suspension) figurant dans tous les types constituant une partie d'une série d'extension.

3.8.31 paire de marqueurs d'extension: paire de marqueurs d'extension entre lesquels les additions d'extension sont insérées.

3.8.32 apparenté par extension: deux types possédant la même racine d'extension dont l'un a été créé par ajout de zéro, une ou plusieurs additions à l'autre.

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

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

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

NOTE – Il est possible d'étendre aussi bien des types imbriqués que des types non imbriqués.

3.8.35 type extensible: type possédant un marqueur d'extension.

3.8.36 référence externe: référence de type, référence de valeur, objet informationnel, etc. défini dans un module quelconque autre que celui dans lequel il y est fait référence, la référence à la définition s'effectuant en préfixant le nom du module de définition au nom de l'élément cité.

EXEMPLE – NomModule.RéférenceType

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

3.8.38 faux (false): une des deux valeurs distinctives du type booléen (voir «vrai»).

3.8.39 gouvernant; gouverneur: type ou classe d'objets informationnels qui commande l'interprétation d'un objet, d'un ensemble d'objets, d'une valeur, d'un ensemble de valeurs ou d'un sous-type, en imposant aux éléments intervenant dans leur notation d'être des notations de valeurs respectivement de ce type ou de cette classe.

3.8.40 type entier (integer type): type simple dont les valeurs distinctives sont les entiers relatifs (les positifs, les négatifs, et l'élément nul en tant que valeur unique).

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

3.8.41 unités lexicales: séquences nommées de caractères du jeu de caractères ASN.1, spécifiées à l'article 11, et utilisées pour former la notation ASN.1.

3.8.42 module: une ou plusieurs instances d'utilisation de la notation ASN.1 pour la définition de types, de valeurs, etc., qui sont regroupées au moyen de la notation de module ASN.1 (voir l'article 12).

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

3.8.44 objet: élément bien défini d'information, de définition ou de spécification, nécessitant un nom afin de l'identifier dans une instance de communication.

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

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

3.8.46 identificateur d'objet: valeur (distincte de toutes les autres), associée à un objet.

3.8.47 type identificateur d'objet (object identifier type): type simple dont les valeurs distinctives sont l'ensemble de tous les identificateurs d'objet affectés conformément aux règles de la Rec. UIT-T X.660 | ISO/CEI 9834-1.

NOTE – Les règles de la Rec. UIT-T X.660 | ISO/CEI 9834-1 permettent à des autorités très diverses d'associer indépendamment les unes des autres des identificateurs à des objets.

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

3.8.49 notation de type ouvert (open type notation): notation ASN.1 servant à désigner un ensemble de valeurs appartenant à plus d'un type ASN.1.

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

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

NOTE 3 – Dans la présente Recommandation | Norme internationale, la seule notation correspondant à un type ouvert est le type «ObjectClassFieldType» (type de champ de classe d'objets), spécifié dans la Rec. UIT-T X.681 | ISO/CEI 8824-2, et dans laquelle le nom de champ «FieldName» désigne soit un champ de type, soit un champ de valeur de type variable. La notation ANY, définie dans la Rec. X.208 du CCITT | ISO/CEI 8824, était une notation de type ouvert.

3.8.50 type parent (d'un sous-type): type dont dérive un sous-type par imposition de contraintes.

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

3.8.51 production: partie de la notation formelle utilisée pour spécifier la notation ASN.1.

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

3.8.53 définitions récursives: ensemble de définitions en notation ASN.1 qui ne peuvent pas être réordonnées de telle sorte que tous les types utilisés dans une structure soient définis avant la définition de cette structure.

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

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

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

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

NOTE – Une valeur du type séquence ne doit pas nécessairement contenir une valeur d'un type composant si celui-ci est déclaré optionnel.

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

3.8.58 types ensemble (set types): types définis par une liste fixe, non ordonnée, de types distincts (dont certains peuvent être déclarés optionnels); chaque valeur du type ensemble est une liste non ordonnée de valeurs, une par type composant.

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

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

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

3.8.61 caractère espacement: caractère d'un répertoire destiné à être inclus en impression avec une chaîne de caractères graphiques, mais qui est représenté matériellement par un vide; il n'est généralement pas considéré comme un caractère de contrôle (voir 3.8.16).

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

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

3.8.63 **étiquette**: dénomination de type associée à chaque type ASN.1.

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

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

3.8.66 **vrai (true)**: une des deux valeurs distinctives du type booléen (voir «faux»).

3.8.67 **type**: ensemble nommé de valeurs.

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

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

3.8.69 **type chaîne de caractères à alphabet non restreint (unrestricted character string type)**: type dont les valeurs sont celles d'une syntaxe abstraite de caractères identifiée séparément pour chaque instance d'utilisation de ce type.

3.8.70 **utilisateur (de la notation ASN.1)**: personne physique ou morale qui définit la syntaxe abstraite d'un élément d'information particulier en notation ASN.1.

3.8.71 **valeur**: élément distinctif d'un ensemble de valeurs.

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

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

3.8.74 **crochets de version**: paire de crochets gauches et droits adjacents ([[ou]]) utilisés pour délimiter le début et la fin d'un groupe d'additions d'extension.

3.8.75 **espace blanc**: toute action de formatage se traduisant par un espace blanc sur la page d'impression, par exemple un espacement, une tabulation, ou l'utilisation répétée de tels caractères.

4 Abréviations

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

ASN.1 Notation de syntaxe abstraite numéro un (*abstract syntax notation one*)

BER Règles de codage de base d'ASN.1 (*basic encoding rules of ASN.1*)

CEI Commission électrotechnique internationale

DCC Indicatif de pays pour transmission de données (*data country code*)

DNIC Code d'identification de réseau pour données (*data network identification code*)

ER Exploitation reconnue

ICD Descripteur de code international (*international code designator*)

ISO Organisation internationale de normalisation (*International organization for standardization*)

PDV Valeur de donnée de présentation (*presentation data value*)

PER Règles de codage compact d'ASN.1 (*packed encoding rules of ASN.1*)

UCS Jeu de caractères universels codés sur plusieurs octets *(universal multiple-octet coded character set*)

UIT-T Union internationale des télécommunications – Secteur de la normalisation des télécommunications

5 Notation

Une notation ASN.1 consiste en une suite de caractères pris dans le jeu de caractères ASN.1 spécifié à l'article 10.

Chaque instance de notation ASN.1 contient des caractères du jeu ASN.1 regroupés en unités lexicales. L'article 11 spécifie toutes les suites de caractères formant des unités lexicales ASN.1, ainsi que le nom de ces unités lexicales.

La notation ASN.1 est définie à l'article 12 (et les articles suivants), en spécifiant la collection des suites d'unités lexicales qui forment des instances valides de la notation ASN.1, avec la sémantique de chaque suite.

Pour spécifier ces ensembles, la présente Recommandation | Norme internationale utilise une notation formelle définie dans les points suivants.

5.1 Productions

Une collection nouvelle (plus complexe) de suites d'unités lexicales ASN.1 est définie au moyen d'une production. Une production utilise les noms des collections de suites de productions définies dans la présente Recommandation | Norme internationale, et forme une nouvelle collection de suites de productions en spécifiant:

a) soit que la nouvelle collection de suites de productions sera composée de n'importe quelle suite contenue dans l'une quelconque des collections d'origine;

b) soit que la nouvelle collection sera composée d'une suite qui peut être obtenue en prenant une suite et une seule dans chaque collection puis en les juxtaposant dans un ordre spécifié.

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

a) le nom de la nouvelle collection de suites de productions;

b) les caractères:

::=

c) une collection ou plusieurs collections au choix de suites de productions, telles qu'elles sont définies au 5.2, séparées par le caractère:

|

Une suite de productions figure dans la nouvelle collection si elle figure dans une ou plusieurs des collections offertes par le choix. La nouvelle collection est désignée dans la présente Recommandation | Norme internationale par le nom mentionné à l'alinéa a) ci-dessus.

NOTE – Si le même ensemble de productions apparaît dans plus d'une des collections offertes par le choix, toute ambiguïté sémantique de la notation résultante est résolue par d'autres parties de l'ensemble total des productions ASN.1.

5.2 Collections de suites de productions figurant dans un choix

Chacune des collections de suites de productions mentionnée dans «une collection ou plusieurs collections au choix de suites de productions» [voir 5.1 c)] est spécifiée par une liste de noms. Chaque nom est soit le nom d'une unité lexicale, soit le nom d'une collection de suites de productions définie par une production dans la présente Recommandation | Norme internationale.

La collection de suites de productions définie par l'une des formes offertes par le choix comprend toutes les suites de productions obtenues en prenant l'une quelconque des suites de productions (ou des unités lexicales) associées au premier nom, combinée avec (et suivie de) l'une quelconque des suites de productions (ou unités lexicales) associées au second nom, combinée avec (et suivie de) l'une quelconque des suites de productions (ou unités lexicales) associées au troisième nom, et ainsi de suite jusques et y compris le dernier nom (ou unité lexicale) de la forme retenue.

5.3 Exemple de production

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

est une production qui associe au nom «BitStringValue» une des suites de productions suivantes:

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

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

c) soit une suite de productions quelconque associée à la liste d'identificateurs «IdentifierList», précédée du caractère «{» et suivie du caractère «}».

NOTE – «{» et «}» sont les noms des unités lexicales contenant respectivement les seuls caractères { et } (voir 11.17).

Dans cet exemple, «IdentifierList» serait défini par une autre production, placée avant ou après la production définissant «BitStringValue» .

5.4 Mise en page

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

5.5 Récursivité

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

NOTE – Dans de nombreux cas, cette procédure récursive produit une collection non bornée de séquences autorisées, certaines pouvant elles-mêmes être non bornées. Ceci n'est pas une erreur.

5.6 Désignation d'une collection de suites de productions

La présente Recommandation | Norme internationale permet de désigner une collection de suites de productions (faisant partie de la notation ASN.1) en citant le nom du membre gauche (à gauche du signe ::=); le nom est mis entre guillemets « » pour le distinguer du texte en langage naturel, sauf s'il apparaît à l'intérieur d'une production.

5.7 Désignation d'une unité lexicale

La présente Recommandation | Norme internationale permet de désigner une unité lexicale en citant son nom; le nom est mis entre guillemets (« ») pour le distinguer du texte en langage naturel, sauf s'il apparaît à l'intérieur d'une production et qu'il ne s'agit ni d'un élément monocaractère ni de l'un des éléments suivants «::=», «..» ou «...».

5.8 Notations abrégées

Pour rendre les productions plus concises et en faciliter la lecture, les notations abrégées suivantes sont utilisées dans les définitions des collections de suites de production ASN.1 dans les Rec. UIT-T X.681 | ISO/CEI 8824-2, Rec. UIT-T X.682 | ISO/CEI 8824-3 et Rec. UIT-T X.683 | ISO/CEI 8824-4 (mais elles ne sont utilisées nulle part dans la présente Recommandation | Norme internationale):

a) un astérisque (*) placé après deux noms «A» et «B» indique soit l'unité lexicale vide (empty) (voir 11.7), soit une suite de productions associées à «A», soit une séquence alternée de suites de productions ASN.1 associées respectivement à «A» et à «B», commençant et se terminant par la suite de productions associées à «A». Ainsi:

C ::= A B *

équivaut à:

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

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

EXEMPLE – «C::= A B *» est une notation abrégée désignant l'une quelconque des productions suivantes:

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

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

E ::= A B +

équivaut à:

E ::= A | A B E

EXEMPLE – «E::= A B +» est une notation abrégée désignant l'une quelconque des productions suivantes:

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

c) un point d'interrogation (?) placé à la suite d'un nom indique soit l'unité lexicale vide (voir 11.7), soit une suite de productions associées à ce nom. Ainsi:

F ::= A ?

équivaut à:

F ::= empty | A

6 Modèle ASN.1 d'extension de type

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

a) absence d'additions d'extension attendues dans un type séquence ou ensemble;

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

D'un point de vue formel, une syntaxe abstraite définie par le type extensible «X» contient non seulement les valeurs du type «X», mais également les valeurs de tous les types qui sont des apparentées à «X» par extension. Il s'ensuit que le processus de décodage ne signale jamais d'erreur lorsque l'une des situations précédentes (a ou b) est détectée. L'action à effectuer dans chacune de ces situations doit faire l'objet d'une spécification par le concepteur de la couche application.

NOTE – L'action consistera souvent à ignorer la présence d'extensions supplémentaires non attendues et à utiliser une valeur par défaut ou un indicateur «absent» pour les additions d'extension attendues qui sont absentes.

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

7 Conditions imposées aux règles de codage par l'extensibilité

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

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

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

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

b) valeurs abstraites de chacune des additions d'extension éventuelles qui figurent à la fois dans les types «X» et «Y»;

c) codage délimité pour toute extension éventuelle figurant dans le type «X» et non dans le type «Y».

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

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

7.2 Toutes les règles de codage ASN.1 spécifieront le codage et le décodage d'un type énuméré et d'un type choix d'une manière telle que, si une valeur transmise se trouve dans l'ensemble d'additions d'extension communes au codeur et au décodeur, cette valeur soit décodée avec succès; en d'autres termes, le décodeur doit avoir la possibilité d'en délimiter le codage et de l'identifier comme étant une valeur correspondant à une addition d'extension (non connue).

7.3 Toutes les règles de codage ASN.1 spécifieront le codage et le décodage de types contenant des contraintes extensibles d'une manière telle que, si une valeur transmise appartient à l'ensemble d'additions d'extension communes au codeur et au décodeur, cette valeur soit décodée avec succès; en d'autres termes, le décodeur doit avoir la possibilité d'en délimiter le codage et de l'identifier comme étant une valeur correspondant à une addition d'extension (non connue).

Dans tous les cas, la présence d'additions d'extension n'affectera pas la capacité de reconnaissance ultérieure de l'information lorsqu'un type possédant un marqueur d'extension est imbriqué au sein d'un autre type.

NOTE – Toutes les variantes des règles de codage de base et des règles de codage compact d'ASN.1 satisfont à toutes ces prescriptions.

8 Etiquettes

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

- universelle;
- application;
- privée;
- propre au contexte.

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

L'article 30 spécifie les restrictions applicables aux étiquettes par l'utilisateur de la notation ASN.1.

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

8.3 Certaines règles de codage nécessitent de disposer d'un ordre canonique pour les étiquettes. Pour en assurer l'uniformité, un tel ordre canonique a été défini au 8.4.

8.4 L'ordre canonique des étiquettes est défini de la manière suivante:

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

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

Tableau 1 – Affectation des étiquettes de la classe universelle

UNIVERSAL 0	Utilisation réservée aux règles de codage
UNIVERSAL 1	Type booléen
UNIVERSAL 2	Type entier
UNIVERSAL 3	Type chaîne binaire
UNIVERSAL 4	Type chaîne d'octets
UNIVERSAL 5	Type néant
UNIVERSAL 6	Type identificateur d'objet
UNIVERSAL 7	Type descripteur d'objet
UNIVERSAL 8	Type externe et type instance-de
UNIVERSAL 9	Type réel
UNIVERSAL 10	Type énuméré
UNIVERSAL 11	Type valeur de donnée de présentation enchâssée
UNIVERSAL 12	Type UFT8String
UNIVERSAL 13-15	Réservées pour de futures éditions de la présente Recommandation | Norme internationale
UNIVERSAL 16	Types séquence et séquence-de
UNIVERSAL 170	Types ensemble et ensemble-de
UNIVERSAL 18-22, 25-30	Types chaînes de caractères
UNIVERSAL 23-24	Types temps
UNIVERSAL 31-...	Réservées aux additifs à la présente Recommandation | Norme internationale

9 Utilisation de la notation ASN.1

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

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

NOTE – Il n'est en général pas possible d'interpréter la notation d'une valeur sans en connaître le type.

9.3 Un type est affecté à un nom de référence de type en ASN.1 par la notation «TypeAssignment» (voir 15.1), «ValueSetTypeAssignment» (voir 15.4), «ParameterizedTypeAssignment» (voir 8.2 de la Rec. UIT-T X.683 | ISO/CEI 8824-4) ou «ParameterizedValueSetTypeAssignment» (voir 8.2 de la Rec. UIT-T X.683 | ISO/CEI 8824-4).

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

9.5 Les différentes productions de notation d'affectation «Assignment» ne seront utilisées que dans la notation «ModuleDefinition» (définition de module) (excepté ce qui est spécifié dans la Note 2 du 12.1).

10 Jeu de caractères ASN.1

10.1 Une unité lexicale ASN.1 consiste en une suite de caractères pris dans le Tableau 2, sous réserve des indications spécifiées aux 10.2 et 10.3.

Tableau 2 – Caractères ASN.1

A à Z
a à z
0 à 9
: = , { } < . @ ( ) [ ] – ' " | & ^ *; !

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

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

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

10.2 Lorsque la notation est utilisée pour spécifier la valeur d'un type chaîne de caractères, tous les symboles graphiques du jeu défini peuvent apparaître entre guillemets (") dans la notation ASN.1 (voir 11.11).

10.3 Des symboles graphiques supplémentaires quelconques peuvent apparaître dans l'unité lexicale commentaire (voir 11.6).

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

10.5 Les majuscules et les minuscules seront considérées comme distinctes.

11 Unités lexicales ASN.1

11.1 Règles générales

11.1.1 Les articles suivants spécifient les caractères utilisés pour les unités lexicales ASN.1. Dans chaque cas, le nom de l'unité lexicale est donné avec la définition des suites de caractères qui le forment.

11.1.2 Chaque unité lexicale spécifiée dans les points suivants (exception faite des unités lexicales «bstring», «hstring» et «cstring») apparaîtra sur une seule ligne et ne contiendra pas d'espace blanc (exception faite des unités lexicales «comment», «bstring», «hstring» et «cstring») (voir 11.9, 11.10 et 11.11).

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

11.1.4 Les unités lexicales des suites de productions spécifiées par la présente Recommandation | Norme internationale (la notation ASN.1) pourront apparaître sur une ou plusieurs lignes et être séparées par des espaces blancs, des interlignes ou des commentaires.

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

11.2 Référence de type

Nom de l'unité lexicale – typereference

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

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

11.2.2 Une référence «typereference» ne doit pas être l'une des suites de caractères réservées indiquées au 11.18.

11.3 Identificateur

Nom de l'unité lexicale – identifier

Un identificateur «identifier» sera constitué d'un nombre quelconque (un ou plusieurs) de caractères du type lettres, chiffres et traits d'union. Le premier caractère doit être une minuscule. Le dernier caractère ne peut pas être un trait d'union. Un trait d'union ne peut pas être immédiatement suivi par un autre trait d'union.

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

11.4 Référence de valeur

Nom de l'unité lexicale – valuereference

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

11.5 Référence de module

Nom de l'unité lexicale – modulereference

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

11.6 Commentaire

Nom de l'unité lexicale – comment

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

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

11.6.2 Un commentaire commence par un double trait d'union et se termine au premier double trait d'union rencontré ou en fin de la ligne. Un commentaire ne doit pas contenir de doubles traits d'union autres que ceux par lesquels il commence et éventuellement se termine. Il peut inclure des symboles graphiques qui n'appartiennent pas au jeu de caractères spécifié au 10.1 (voir 10.3).

11.7 Unité lexicale vide

Nom de l'unité lexicale – empty

L'unité lexicale vide «empty» ne contient aucun caractère. Elle est utilisée dans la notation de l'article 5 pour inclure l'ensemble vide dans les formes possibles d'un choix entre plusieurs ensembles de suites de productions.

11.8 Unité lexicale nombre

Nom de l'unité lexicale – number

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

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

11.9 Unité lexicale chaîne binaire

Nom de l'unité lexicale – bstring

Une chaîne binaire «bstring» se compose d'un nombre quelconque (éventuellement zéro) de 0, de 1, de blancs et d'interlignes, précédés du caractère «'» et suivis des deux caractères:

'B

Les blancs et les interlignes à l'intérieur d'une unité lexicale chaîne binaire n'ont pas de signification.

EXEMPLE – '01101100'B

11.10 Unité lexicale chaîne hexadécimale

Nom de l'unité lexicale – hstring

11.10.1 Une chaîne «hstring» est composée d'un nombre quelconque (éventuellement zéro) de blancs, d'interlignes et de caractères pris parmi:

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

précédés du caractère «'» et suivis des deux caractères:

'H

Les blancs et les interlignes à l'intérieur d'une unité lexicale chaîne hexadécimale n'ont pas de signification.

EXEMPLE – 'AB0196'H

11.10.2 Chaque caractère représente la valeur d'un demi octet exprimée en notation hexadécimale.

11.11 Unité lexicale chaîne de caractères

Nom de l'unité lexicale – cstring

11.11.1 Une chaîne de caractères «cstring» est composée d'un nombre quelconque (éventuellement zéro) de symboles graphiques et de caractères d'espacement du jeu désigné par le type de la chaîne de caractères, placés entre guillemets ("). Si le jeu de caractères comprend le guillemet, il sera représenté le cas échéant dans la chaîne de caractères par deux guillemets consécutifs ("") sur la même ligne et sans espace intermédiaire. La chaîne de caractères peut occuper plusieurs lignes de texte, auquel cas la valeur de chaîne de caractères représentée ne comportera pas de caractère d'espacement dans la position située juste avant ou après les fins de ligne de sa représentation. Un blanc situé immédiatement avant ou après une fin de ligne d'une chaîne «cstring» n'a pas de signification.

NOTE 1 – Une chaîne «cstring» ne peut être utilisée que pour représenter des chaînes dont chaque caractère est soit un symbole graphique soit un caractère d'espacement. Il existe d'autres syntaxes ASN.1 pour représenter des chaînes comportant des caractères de contrôle (voir l'article 34).

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

EXEMPLE 1 – «屎 屍 市 弒»

EXEMPLE 2 – La chaîne «cstring»

"ABCDE FGH

IJK""XYZ"

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

ABCDE FGHIJK"XYZ

où l'espacement voulu entre E et F peut être ambigu si une police de caractères à espacement proportionnel est utilisée dans la spécification (ce qui est le cas dans l'exemple).

11.11.2 Un caractère de combinaison sera représenté dans une chaîne «cstring» comme un caractère isolé et ne recevra pas en surimpression le caractère avec lequel il se combine. (Ceci garantit la détermination sans ambiguïté de l'ordre des caractères de combinaison dans la valeur de la chaîne.)

EXEMPLE – L'accent aigu de combinaison et la minuscule «e» sont deux caractères de la grille ISO 646 version française, et seront donc représentés dans la chaîne «cstring» correspondante par deux caractères et non par le simple caractère «é».

11.11.3 La chaîne «cstring» ne sera pas utilisée pour représenter des chaînes de caractères contenant des caractères de contrôle. Seuls les caractères graphiques et les caractères d'espacement sont permis.

11.12 Unité lexicale affectation

Nom de l'unité lexicale – «::=»

Cette unité lexicale est composée de la séquence de caractères:

::=

NOTE – Cette séquence ne doit contenir aucun caractère blanc (voir 11.1.2).

11.13 Séparateur de domaine de valeurs

Nom de l'unité lexicale – «..»

Cette unité lexicale se compose de la séquence de caractères:

..

NOTE – Cette séquence ne doit contenir aucun caractère blanc (voir 11.1.2).

11.14 Points de suspension

Nom de l'unité lexicale – «...»

Cette unité lexicale se compose de la séquence de caractères:

...

NOTE – Cette séquence ne doit contenir aucun caractère blanc (voir 11.1.2).

11.15 Crochets gauches de version

Nom de l'unité lexicale – «[[»

Cette unité lexicale se compose de la séquence de caractères:

[[

NOTE – Cette séquence ne doit contenir aucun caractère blanc (voir 11.1.2).

11.16 Crochets droits de version

Nom de l'unité lexicale – «]]»

Cette unité lexicale se compose de la séquence de caractères:

]]

NOTE – Cette séquence ne doit contenir aucun caractère blanc (voir 11.1.2).

11.17 Unités lexicales à caractère unique

Noms des unités lexicales:

«{»
«}»
«<»
«,»

«.»
«(»
«)»
«[»
«]»
«-» (trait d'union)
«:»
«,»
«@»
«|»
«!»
«^»

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

11.18 Unités lexicales mots réservés

Noms des mots réservés –

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

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

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

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

12 Définition de module

12.1 Une définition «ModuleDefinition» est spécifiée par les productions suivantes:

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

ModuleIdentifier ::=
        modulereference
        DefinitiveIdentifier

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

DefinitiveObjIdComponentList  ::=
        DefinitiveObjIdComponent     |
        DefinitiveObjIdComponent DefinitiveObjIdComponentList

DefinitiveObjIdComponent  ::=
        NameForm                     |
        DefinitiveNumberForm         |
        DefinitiveNameAndNumberForm

DefinitiveNumberForm    ::= number

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

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

ExtensionDefault ::=
        EXTENSIBILITY  IMPLIED          |
        empty

ModuleBody ::::=
        Exports Imports AssignmentList  |
        empty

Exports ::=
        EXPORTS SymbolsExported ";"     |
        empty

SymbolsExported ::=
        SymbolList |
        empty

Imports ::=
        IMPORTS SymbolsImported ";"     |
        empty

SymbolsImported ::=
        SymbolsFromModuleList           |
        empty

SymbolsFromModuleList ::=
        SymbolsFromModule               |
        SymbolsFromModuleList  SymbolsFromModule
```

```
SymbolsFromModule ::=
        SymbolList FROM GlobalModuleReference

GlobalModuleReference ::=
        modulereference AssignedIdentifier

AssignedIdentifier ::=
        ObjectIdentifierValue   |
        DefinedValue            |
        empty

SymbolList ::=
        Symbol          |
        SymbolList "," Symbol

Symbol ::=
        Reference       |
        ParameterizedReference

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

AssignmentList ::=
        Assignment                      |
        AssignmentList Assignment

Assignment ::=
        TypeAssignment                  |
        ValueAssignment                 |
        ValueSetTypeAssignment          |
        ObjectClassAssignment           |
        ObjectAssignment                |
        ObjectSetAssignment             |
        ParameterizedAssignment
```

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

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

NOTE 3 – Les productions «TypeAssignment» et «ValueAssignment» sont spécifiées à l'article 15.

NOTE 4 – Le regroupement de types de donnée ASN.1 en modules ne détermine pas nécessairement le regroupement des valeurs de donnée de présentation dans des syntaxes abstraites nommées pour la définition de contextes de présentation.

NOTE 5 – La valeur de l'étiquette par défaut «TagDefault» de la définition d'un module affecte uniquement les types définis explicitement dans ce module. Elle n'affecte pas l'interprétation des types importés.

NOTE 6 – Le point virgule «;» n'apparaît ni dans la spécification de la liste d'affectation «AssignmentList» ni dans l'une quelconque de ses productions subordonnées; son utilisation est réservée aux développeurs d'outils ASN.1.

12.2 Si l'étiquette par défaut «TagDefault» est vide, elle est prise par défaut comme étant «EXPLICIT TAGS».

NOTE – L'article 30 donne la signification des étiquetages explicites, implicites et automatiques: «EXPLICIT TAGS», «IMPLICIT TAGS» et «AUTOMATIC TAGS».

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

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

NOTE – L'option «EXTENSIBILITY IMPLIED» porte uniquement sur des types. Elle n'a aucun effet sur des ensembles d'objets.

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

NOTE – La possibilité de définir un même module ASN.1 comportant plusieurs corps de module «ModuleBody» partageant une même référence de module «modulereference» était autorisée (quoique déconseillée) dans les spécifications précédentes. Cette notation est interdite par la présente Recommandation | Norme internationale.

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

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

NOTE – Le problème de savoir quel volume de modifications impose l'adoption d'un nouvel identificateur «DefinitiveIdentifier» n'est pas traité dans la présente Recommandation | Norme internationale.

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

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

b) elle apparaît en tant que symbole «Symbol» dans un module «SymbolsFromModule», dont l'identificateur affecté «AssignedIdentifier» ne contient textuellement pas de référence de valeur «valuereference».

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

12.9 La référence globale de module «GlobalModuleReference» du module d'origine dans la notation des symboles importés «SymbolsFromModule» apparaîtra dans la définition «ModuleDefinition» d'un autre module, sauf si elle comporte un identificateur «DefinitiveIdentifier» non vide, dans ce cas la référence de module «modulereference» peut être différente.

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

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

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

12.12 Lorsque la notation d'exportation «Exports» contient des symboles dans «SymbolsExported»:

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

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

ii) soit apparaître une et une seule fois dans la liste «SymbolsImported» de la déclaration «Imports»;

b) tous les symboles qu'il serait utile de pouvoir désigner depuis l'extérieur du module seront inclus dans la liste «SymbolsExported», et seront les seuls à pouvoir être ainsi désignés;

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

12.13 Lorsque les exportations «Exports» se voient affecter la valeur vide, tous les symboles «Symbol» définis dans le module peuvent être désignés par d'autres modules.

NOTE – La valeur vide est incluse dans les possibilités d'affectation de la déclaration «Exports» pour des raisons de compatibilité amont.

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

12.15 Lorsque la notation d'importation «Imports» se voit affecter des symboles importés «SymbolsImported», toutes les conditions suivantes devront être remplies:

a) chacun des symboles apparaissant dans une déclaration «SymbolsFromModule» soit sera défini dans le corps du module désigné par la référence globale «GlobalModuleReference» de la déclaration «SymbolsFromModule», soit figurera dans une déclaration «IMPORTS» de ce module. L'importation d'un symbole «Symbol» figurant dans une déclaration «IMPORTS» du module désigné n'est autorisée que s'il existe une seule occurrence de ce symbole dans la déclaration d'importation du module désigné et que le symbole n'est pas défini dans ce module;

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

b) si, la déclaration «Exports» du module désigné par la référence «GlobalModuleReference» de la déclaration «SymbolsFromModule» est du type «EXPORTS SymbolsExported;», le symbole «Symbol» devra apparaître dans la liste «SymbolsExported»;

c) seuls les symboles figurant dans une liste «SymbolList» d'une production «SymbolsFromModule» peuvent apparaître dans une déclaration de référence externe «External<X>Reference» (<X> remplaçant ici «value», «type», «object», «objectclass» ou «objectset») pour laquelle la référence «modulereference» correspond à la déclaration «GlobalModuleReference» de la production «SymbolsFromModule»;

d) lorsqu'il n'existe pas de symbole à importer, la valeur vide sera affectée à la liste «SymbolsImported»;

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

e) les productions «SymbolsFromModule» de la liste «SymbolsFromModuleList» comporteront chacune une référence globale «GlobalModuleReference» telle que:

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

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

12.16 Lorsque les importations «Imports» se voient affecter la valeur vide, le module peut quand même faire référence à des symboles définis dans d'autres modules au moyen de déclarations «External<X>Reference».

NOTE – La valeur vide est incluse dans les possibilités d'affectation de la déclaration «Imports» pour des raisons de compatibilité amont.

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

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

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

12.20 Sauf mention contraire, les différentes possibilités d'affectation «Assignment» sont définies dans les articles suivants de la présente Recommandation | Norme internationale:

Forme d'affectation	*Référence de définition*
«TypeAssignment»	§ 15.1
«ValueAssignment»	§ 15.2
«ValueSetTypeAssignment»	§ 15.4
«ObjectClassAssignment»	Rec. UIT-T X.681 | ISO/CEI 8824-2, § 9.1
«ObjectAssignment»	Rec. UIT-T X.681 | ISO/CEI 8824-2, § 11.1
«ObjectSetAssignment»	Rec. UIT-T X.681 | ISO/CEI 8824-2, § 12.1
«ParameterizedAssignment»	Rec. UIT-T X.683 | ISO/CEI 8824-4, § 8.1

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

13 Définitions des références de types et de valeurs

13.1 Les productions suivantes relatives aux types et valeurs définis:

DefinedType ::=
Externaltypereference |
typereference |
ParameterizedType |
ParameterizedValueSetType

DefinedValue ::=
Externalvaluereference |
valuereference |
ParameterizedValue

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

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

13.3 Les références externes de type «Externaltypereference» (respectivement de valeur «Externalvaluereference») ne seront utilisées que si la référence de type «typereference» (respectivement de valeur «valuereference») correspondante:

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

b) ou est présente dans la section «IMPORTS»

dans le corps de module «ModuleBody» utilisé pour définir la référence «modulereference» correspondante. Il n'est permis de faire référence à une unité lexicale de la section «IMPORTS» d'un autre module que s'il n'existe pas plus d'une occurrence de ce symbole dans la section «IMPORTS».

NOTE – Ceci n'empêche pas d'importer dans un module deux symboles identiques définis dans deux autres modules différents. Mais si un même symbole «Symbol» apparaît plus d'une fois dans la section «IMPORTS» d'un module A, il n'est plus possible de faire référence à ce symbole dans le module A depuis un autre module extérieur.

13.4 Une référence externe ne sera utilisée dans un module que pour désigner une unité lexicale définie dans un autre module; cette référence est assurée par les productions suivantes:

```
Externaltypereference ::=
        modulereference
        "."
        typereference

Externalvaluereference ::=
        modulereference
        "."
        valuereference
```

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

13.5 Lorsque dans le module importateur, la déclaration d'importation «Imports» a la forme «IMPORTS SymbolsImported;», le champ «modulereference» de la référence externe apparaîtra dans le champ «GlobalModuleReference» d'une et une seule production «SymbolsFromModule» de la déclaration «SymbolsImported». Lorsque dans le module importateur, la déclaration d'importation «Imports» est vide, le champ «modulereference» de la référence externe apparaîtra dans la déclaration de définition «ModuleDefinition» du module extérieur dans lequel la «Reference» invoquée est définie.

14 Notation permettant de désigner des composants ASN.1

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

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

14.3 Toutes les parties d'une définition de type ASN.1 peuvent être désignées de l'extérieur en utilisant la structure syntaxique de référence absolue «AbsoluteReference»:

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

ItemSpec ::=
        typereference |
        ItemId "." ComponentId

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

NOTE – La production «AbsoluteReference» n'est pas utilisée dans la présente Recommandation | Norme internationale. Elle a été établie aux fins indiquées au 14.1.

14.4 La référence globale de module «GlobalModuleReference» identifie un module ASN.1 (voir 12.1).

14.5 La référence «typereference» est celle d'un type ASN.1 quelconque défini dans le module identifié par «GlobalModuleReference».

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

14.7 Le champ «identifier» de «ComponentId» peut être utilisé si l'élément «ItemId» antécédent est du type ensemble ou séquence; cet identificateur doit alors être l'un des identificateurs du type nommé «NamedType» de la liste des types de composants «ComponentTypeLists» de cet ensemble ou séquence. Il peut aussi être utilisé si «ItemId» désigne un type choix; il doit alors être l'un des identificateurs de type nommé «NamedType» de la liste des formes possibles «AlternativeTypeLists» de ce type choix. Il ne peut pas être utilisé dans d'autres cas.

14.8 Le champ «ComponentId» ne peut recevoir un numéro que si l'élément «ItemId» est un type séquence-de ou ensemble-de. La valeur du numéro identifie l'instance du type donné dans la structure séquence-de ou ensemble-de, le numéro «1» correspondant à la première instance. Le numéro «0» identifie un composant conceptuel de type entier (non présent explicitement dans le transfert et appelé **nombre d'itérations**) qui indique le nombre d'instances dans la structure séquence-de ou ensemble-de présentes dans la valeur du type contenant.

14.9 Le champ «ComponentId» ne peut recevoir un astérisque «*» que si l'élément «ItemId» est un type séquence-de ou ensemble-de. Toute sémantique associée à l'utilisation de la notation «*» s'applique à tous les composants de la structure séquence-de ou ensemble-de.

NOTE – Dans l'exemple suivant:

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

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

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

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

15 Affectation de types et de valeurs

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

```
TypeAssignment ::=
        typereference
        "::="
        Type
```

La référence «typereference» ne devra pas être l'un des mots ASN.1 réservés (voir 11.18).

15.2 Une valeur sera affectée à une référence de valeur «valuereference» par la notation spécifiée dans la production «ValueAssignment»:

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

La valeur «Value» affectée à la référence «valuereference» sera une notation de valeur correspondant au type défini par le champ «Type» (comme spécifié au 15.3).

15.3 «Value» est une notation de valeur d'un type donné si elle est:

a) soit une notation de valeur prédéfinie «BuiltinValue» de ce type (voir 16.8);

b) soit une notation de valeur définie «DefinedValue» compatible avec ce type.

15.4 La notation spécifiée par la production d'affectation de type ensemble de valeurs «ValueSetTypeAssignment» permet d'affecter un ensemble de valeurs à une référence de type «typereference»:

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

Cette notation affecte à «typereference» le type défini comme un sous-type du type «Type», et contenant exactement les valeurs spécifiées ou autorisées par «ValueSet». La référence «typereference» ne sera pas un mot ASN.1 réservé (voir 11.18), et il sera possible d'y faire référence comme à un type. L'ensemble de valeurs «ValueSet» est défini au 15.5.

15.5 Un ensemble de valeurs d'un type donné est spécifié par la notation «ValueSet»:

ValueSet ::= "{" ElementSetSpecs "}"

L'ensemble de valeurs comprend toutes les valeurs, au nombre d'une au moins, spécifiées par la production «ElementSetSpecs» (voir l'article 46).

16 Définition des types et des valeurs

16.1 Un type est spécifié par la notation «Type»:

Type ::= BuiltinType | ReferencedType | ConstrainedType

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

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

Les divers types prédéfinis «BuiltinType» possibles sont définis dans les articles suivants (de la présente Recommandation | Norme internationale sauf indication contraire):

BitStringType	21
BooleanType	17
CharacterStringType	35
ChoiceType	28
EmbeddedPDVType	32
EnumeratedType	19
ExternalType	33
InstanceOfType	Rec. UIT-T X.681 | ISO/CEI 8824-2, Annexe C
IntegerType	18
NullType	23
ObjectClassFieldType	Rec. UIT-T X.681 | ISO/CEI 8824-2, 14.1
ObjectIdentifierType	31
OctetStringType	22
RealType	20
SequenceType	24
SequenceOfType	25
SetType	26
SetOfType	27
TaggedType	30

16.3 Les types référencés de la notation ASN.1 sont spécifiés par la notation «ReferencedType»:

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

La notation «ReferencedType» offre une manière différente de désigner un autre type (et en dernier lieu un type prédéfini). Les différentes notations «ReferencedType» et la manière utilisée pour déterminer le type auquel elles renvoient sont spécifiées aux points suivants (de la présente Recommandation | Norme internationale sauf indication contraire):

DefinedType	13.1
UsefulType	40.1
SelectionType	29
TypeFromObject	Rec. UIT-T X.681 | ISO/CEI 8824-2, article 15
ValueSetFromObjects	Rec. UIT-T X.681 | ISO/CEI 8824-2, article 15

16.4 Le type contraint «ConstrainedType» est défini à l'article 44.

16.5 La présente Recommandation | Norme internationale spécifie l'utilisation de la notation «NamedType» suivante pour spécifier les composants des types ensemble, séquence et choix:

NamedType ::= identifier Type

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

16.7 Une valeur d'un type donné est spécifiée par la notation «Value»:

Value ::= BuiltinValue | ReferencedValue | ObjectClassFieldValue

NOTE – ObjectClassFieldValue est défini dans la Rec. UIT-T X.681 | ISO/CEI 8824-2, paragraphe 14.6.

16.8 Les valeurs des types prédéfinis de la notation ASN.1 peuvent être spécifiées par la notation de valeur prédéfinie «BuiltinValue» définie comme suit:

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

Chacune des formes possibles de «BuiltinValue» est définie dans le même article que le type correspondant de la notation «BuiltinType», selon la liste du 16.2 ci-dessus.

16.9 Les valeurs référencées de la notation ASN.1 sont spécifiées par la notation «ReferencedValue» suivante:

ReferencedValue ::=
DefinedValue |
ValueFromObject

La notation «ReferencedValue» offre une manière différente de désigner une autre valeur (et en dernier lieu une valeur prédéfinie). Les différentes notations «ReferencedValue» et la manière utilisée pour déterminer la valeur à laquelle elles renvoient sont spécifiées aux points suivants (de la présente Recommandation | Norme internationale sauf indication contraire):

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

16.10 Indépendamment du fait qu'un type soit prédéfini «BuiltinType», référencé «ReferencedType» ou contraint «ConstrainedType», ses valeurs peuvent être spécifiées par une valeur «BuiltinValue» ou «ReferencedValue» de ce type.

16.11 La valeur d'un type désigné à l'aide de la notation «NamedType» sera définie par la notation «NamedValue»:

NamedValue ::= identifier Value

où l'identificateur «identifier» est le même que celui de la notation du type nommé «NamedType».

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

16.12 La présence implicite ou explicite d'un marqueur d'extension n'a aucun effet sur la notation de la valeur. Ceci signifie que la notation de valeur pour un type avec un marqueur d'extension est exactement la même que si le marqueur d'extension était absent.

17 Notation du type booléen (boolean type)

17.1 Le type booléen (voir 3.8.7) est déclaré par la notation «BooleanType»:

BooleanType ::= BOOLEAN

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

17.3 La notation «BooleanValue» permet de définir la valeur d'un type booléen (voir 3.8.66 et 3.8.38):

BooleanValue ::= TRUE | FALSE

18 Notation du type entier (integer type)

18.1 Le type entier (voir 3.8.40) est déclaré par la notation «IntegerType»:

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

NamedNumberList ::=
        NamedNumber             |
        NamedNumberList "," NamedNumber

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

SignedNumber ::= number | "-" number
```

18.2 La seconde forme possible de «SignedNumber» ne sera pas utilisée si «number» est nul.

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

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

NOTE – Un identificateur ne pouvant servir à spécifier la valeur associée à «NamedNumber», la valeur définie «DefinedValue» ne peut jamais être prise pour un entier «IntegerValue». Dans les cas suivants:

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

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

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

18.6 Les identificateurs apparaissant dans la liste «NamedNumberList» seront tous différents.

18.7 L'ordre des séquences «NamedNumber» de la liste «NamedNumberList» n'est pas significatif.

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

18.9 La valeur d'un type entier est déclarée par la notation «IntegerValue»:

```
IntegerValue ::=
        SignedNumber |
        identifier
```

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

NOTE – Pour faire référence à un entier ayant reçu un identificateur, il est préférable d'utiliser l'identificateur «identifier» de la valeur «IntegerValue».

19 Notation du type énuméré (enumerated type)

19.1 Le type énuméré (voir 3.8.24) est déclaré par la notation «EnumeratedType»:

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

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

RootEnumeration ::= Enumeration

AdditionalEnumeration ::= Enumeration

Enumeration ::=
        EnumerationItem | EnumerationItem "," Enumeration

EnumerationItem ::=
        identifier | NamedNumber
```

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

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

19.2 L'identificateur et le nombre signé «SignedNumber» des différents nombres «NamedNumber» seront différents de tous les autres identificateurs et nombres signés de la production «Enumeration». Les dispositions des paragraphes 18.2 et 18.4 s'appliquent aussi à chaque nombre nommé «NamedNumber».

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

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

19.4 La valeur associée à chaque nouvelle unité lexicale de la production «AdditionalEnumeration» sera supérieure à celle de toute unité lexicale de la production «AdditionalEnumeration» définie précédemment dans le type.

19.5 Lorsqu'une unité lexicale définie comme nombre nommé «NamedNumber» est utilisée dans la production «AdditionalEnumeration», la valeur qui lui est associée sera différente de la valeur de toute unité lexicale «EnumerationItem» définie précédemment (dans ce type), indépendamment du fait que l'unité lexicale «EnumerationItem» définie précédemment figure ou non dans la racine de l'énumération. A titre d'exemple:

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

19.6 La valeur associée à la première unité lexicale de la production «AdditionalEnumeration» qui est définie par un identificateur «identifier» (et non par un nombre nommé «NamedNumber») sera la plus petite valeur à laquelle n'est associée aucune unité lexicale de la production «RootEnumeration» telle que la valeur associée à toute unité lexicale «EnumerationItem» précédente dans la production «AdditionalEnumeration» (s'il en existe) soit inférieure. Les exemples suivants sont tous valides:

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

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

19.8 La valeur d'un type énuméré est déclarée par la notation «EnumeratedValue»:

```
EnumeratedValue ::= identifier
```

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

20 Notation du type réel (real type)

20.1 Le type réel (voir 3.8.52) est déclaré par la notation «RealType»:

```
RealType ::= REAL
```

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

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

$$M \times B^{E}$$

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

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

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

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

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

NOTE 1 – Les valeurs représentées en «base» 2 et en «base» 10 sont considérées comme des valeurs abstraites distinctes même si elles correspondent à la même valeur réelle, et peuvent recouvrir des sémantiques d'application différentes.

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

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

20.6 La valeur d'un type réel sera déclarée au moyen de la notation «RealValue»:

```
RealValue ::=
        NumericRealValue | SpecialRealValue
```

```
NumericRealValue ::= 0          |
        SequenceValue                   -- valeur du type séquence associée

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

Si la valeur est nulle, elle sera représentée par la forme «0» et la seconde forme possible de «NumericRealValue» ne sera pas utilisée.

21 Notation du type chaîne binaire (bitstring type)

21.1 Le type chaîne binaire (voir 3.8.6) sera déclaré au moyen de la notation «BitStringType»:

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

NamedBitList ::=
        NamedBit  |
        NamedBitList "," NamedBit

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

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

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

21.3 «DefinedValue» désignera une valeur non négative de type entier.

21.4 Les valeurs «number» ou «DefinedValue» de la liste «NamedBitList» seront toutes différentes, et correspondront chacune au numéro d'un bit distinct dans une chaîne binaire.

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

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

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

```
a  INTEGER ::= 1

T1 ::= INTEGER { a(2) }

T2 ::= BIT STRING { a(3), b(a) }
```

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

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

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

21.8 Ce type porte l'étiquette numéro 3 de la classe universelle.

21.9 Une valeur d'un type chaîne binaire «bitstring» est déclarée au moyen de la notation «BitStringValue»:

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

IdentifierList ::=
        identifier   |
        IdentifierList "," identifier
```

21.10 Chaque identificateur «identifier» de la notation «BitStringValue» sera identique à l'identificateur de la production «BitStringType» auquel cette valeur est associée.

21.11 La notation «BitStringValue» représente une valeur de chaîne binaire dont les bits de numéros correspondant aux identificateurs contiennent des «1», tous les autres bits étant à «0».

NOTE – La suite de productions "{" "}" sert à représenter une chaîne binaire ne contenant pas de «1».

21.12 Lors de la spécification de règles de codage pour les chaînes binaires, on utilisera les expressions **premier bit** et **bit de fin** dans la désignation des bits, le premier bit portant le numéro zéro (voir 21.2).

21.13 Dans la notation «bstring», le **premier bit** est à gauche et le **bit de fin** à droite.

21.14 Dans la notation «hstring», le bit le plus significatif de chaque chiffre hexadécimal est celui de gauche.

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

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

EXEMPLE

'A98A'H

et

'1010100110001010'B

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

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

22.1 Le type chaîne d'octets «octetstring» (voir 3.8.48) est déclaré par la notation «OctetStringType»:

OctetStringType ::= OCTET STRING

22.2 Ce type porte l'étiquette numéro 4 de la classe universelle.

22.3 Une valeur d'un type chaîne d'octets est déclarée par la notation «OctetStringValue»:

OctetStringValue ::=
bstring |
hstring

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

22.5 Si la valeur est déclarée au moyen de la notation «bstring», le bit le plus à gauche sera le bit de plus fort poids du premier octet. Si le nombre de bits de la chaîne binaire n'est pas un multiple de huit, elle sera interprétée comme se terminant implicitement par des bits complémentaires tous nuls, la complétant au prochain multiple de huit bits.

22.6 Si la valeur est déclarée au moyen de la notation «hstring», le chiffre hexadécimal le plus à gauche sera le demi-octet le plus significatif du premier octet.

22.7 Si le nombre de chiffres de la chaîne hexadécimale n'est pas paire, elle sera interprétée comme se terminant implicitement par un chiffre hexadécimal complémentaire nul.

23 Notation du type néant (null type)

23.1 Le type néant «null» (voir 3.8.43) est déclaré au moyen de la notation «NullType»:

NullType ::= NULL

23.2 Ce type porte l'étiquette numéro 5 de la classe universelle.

23.3 Une valeur du type néant est déclarée par la notation «NullValue»:

NullValue ::= NULL

24 Notation des types séquence (sequence types)

24.1 Un type séquence (voir 3.8.56) est déclaré au moyen de la notation «SequenceType»:

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

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

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

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

RootComponentTypeList ::= ComponentTypeList

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

ExtensionAdditions ::= "," ExtensionAdditionList | empty

ExtensionAdditionList ::= ExtensionAddition |
        ExtensionAdditionList "," ExtensionAddition

ExtensionAddition ::= ComponentType | ExtensionAdditionGroup

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

ComponentTypeList ::=
        ComponentType                                   |
        ComponentTypeList "," ComponentType

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

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

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

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

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

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

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

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

24.4 Le type dans la notation «COMPONENTS OF Type» sera un type séquence. La notation «COMPONENTS OF Type» sera utilisée pour définir l'inclusion, à cet emplacement dans la liste des composants, de tous les composants contenus dans le type indiqué, à l'exception de tout marqueur d'extension et de toute addition d'extension pouvant figurer dans le «Type». (Seul l'élément «RootComponentTypeList» du type indiqué par la notation «COMPONENTS OF Type» est inclus; les marqueurs d'extension et les additions d'extension éventuels sont ignorés par la notation «COMPONENTS OF Type».)

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

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

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

24.5.2 Le paragraphe 24.5.1 s'appliquera aux séries de productions «ComponentType» dans la racine.

24.5.3 Le paragraphe 24.5.1 s'appliquera aux séries complètes de productions «ComponentType» dans la racine ou dans les additions d'extension, dans l'ordre textuel de leur apparition au sein de la définition du type (en ignorant toute notation de crochet de version et de points de suspension).

24.6 Tous les composants «ComponentType» dans les additions d'extension auront des étiquettes qui différeront des étiquettes de tous les composants «ComponentType» suivants dans l'ordre textuel figurant dans la racine jusques et y compris le premier composant «ComponentType» de cette sorte qui n'est pas marqué comme «OPTIONAL» ou «DEFAULT» (s'il en existe).

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

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

24.9 Le type étiqueté de remplacement «TaggedType» est spécifié comme suit:

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

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

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

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

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

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

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

```
T ::= SEQUENCE { a  Ta, b  Tb, c  Tc }

E ::= SEQUENCE { f1 E1, f2 T,  f3 E3 }
```

les étiquettes attachées à «a», «b» et «c» ne sont pas affectées par l'étiquetage automatique éventuellement appliqué aux composants de «E».

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

```
T ::= SEQUENCE { a  Ta,  b  SEQUENCE { b1  T1,  b2  T2,  b3  T3},  c Tc }

W ::= SEQUENCE { x  Wx,  COMPONENTS OF  T, y  Wy }
```

les étiquettes de «a», «b» et «c» dans l'expression de «T» ne sont pas nécessairement identiques aux étiquettes de «a», «b» et «c» dans l'expression de «W» si «W» a été défini dans un environnement d'étiquetage automatique, mais les étiquettes de «b1», «b2» et «b3» sont les mêmes dans «T» et «W». En d'autres termes, l'étiquetage automatique n'est appliqué qu'une seule fois à une liste «ComponentTypeList» donnée.

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

NOTE 5 – En étiquetage automatique, l'insertion de nouveaux composants peut induire des modifications sur les autres composants par suite des effets collatéraux de la modification des étiquettes.

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

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

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

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

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

NOTE 1 – Lorsque le type est passé de la racine d'extension (version 1) vers la version 2 puis vers la version 3 par l'ajout d'additions d'extension, la présence d'un codage de toute addition appartenant à la version 3 nécessite un codage de toutes les additions appartenant à la version 2 qui ne sont pas marquées «OPTIONAL» ou «DEFAULT».

NOTE 2 – Les types «ComponentType» qui sont des additions d'extension mais qui ne sont pas contenus dans un groupe «ExtensionAdditionGroup» doivent toujours être codés comme s'ils n'étaient pas marqués «OPTIONAL» ou «DEFAULT», à l'exception des données de présentation qui sont relayées à partir d'un émetteur utilisant une version antérieure de la syntaxe abstraite dans laquelle le type «ComponentType» n'est pas défini.

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

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

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

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

24.15 Tous les types séquence porteront l'étiquette numéro 16 de la classe universelle.

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

24.16 Une valeur de type séquence sera définie par la notation «SequenceValue»:

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

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

24.17 La suite de productions "{" "}" ne sera utilisée que si:

a) tous les composants «ComponentType» de la production «SequenceType» comportent la déclaration «DEFAULT» ou «OPTIONAL», et toutes les valeurs sont omises;

b) la notation du type était «SEQUENCE{ }».

24.18 Une valeur «NamedValue» figurera pour chaque type «NamedType» du type «SequenceType» ne comportant pas la déclaration «OPTIONAL» ou «DEFAULT», et ces valeurs apparaîtront dans le même ordre que les types «NamedType» de la séquence correspondante.

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

25.1 Un type séquence-de (voir 3.8.57) est défini à partir d'un «Type» donné par la notation «SequenceOfType»:

```
SequenceOfType  ::=  SEQUENCE OF Type
```

25.2 Les types séquence-de portent tous l'étiquette numéro 16 de la classe universelle.

NOTE – Les types séquence portent la même étiquette que les types séquence-de (voir 24.15).

25.3 Une valeur de type séquence-de est déclarée par la notation «SequenceOfValue»:

```
SequenceOfValue ::= "{" ValueList "}"    |    "{"  "}"

ValueList ::=
        Value      |
        ValueList "," Value
```

La notation "{" "}" est utilisée quand la valeur séquence-de est une liste vide.

25.4 Chaque valeur «Value» de la liste «ValueList» sera du type spécifié dans la déclaration «SequenceOfType».

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

26 Notation des types ensemble (set types)

26.1 Un type ensemble (voir 3.8.58) est défini à partir d'autres types par la notation «SetType»:

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

Les listes de types de composants «ComponentTypeLists», «ExtensionAndException» et «OptionalExtensionMarker» sont spécifiées au 24.1.

26.2 Le «Type» dans la notation «COMPONENTS OF Type» sera un type ensemble. La notation «COMPONENTS OF Type» sera utilisée pour définir l'inclusion, au niveau de ce point dans la liste de composants, de tous les composants contenus dans le type indiqué, à l'exception de tout marqueur d'extension et addition d'extension pouvant figurer dans le «Type». (Seul l'élément «RootComponentTypeList» du type indiqué par la notation «COMPONENTS OF Type» est inclus, les marqueurs d'extension et les additions d'extension éventuels sont ignorés par la notation «COMPONENTS OF Type».)

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

26.3 Les types «ComponentType» d'un type ensemble porteront tous des étiquettes différentes (voir l'article 30). Chaque nouveau type «ComponentType» ajouté à la liste «AdditionalComponentTypelist» portera une étiquette canoniquement plus grande (voir 8.4) que celles des autres composants de la liste.

NOTE – Lorsque l'étiquetage par défaut «TagDefault» du module dans lequel cette notation apparaît est «AUTOMATIC TAGS», il résulte des dispositions du 24.7 que l'étiquetage automatique est appliqué quel que soit le type «ComponentType» effectif.

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

26.5 Les types ensemble portent tous l'étiquette numéro 17 de la classe universelle.

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

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

26.7 Une valeur de type ensemble est déclarée par la notation «SetValue»:

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

La liste de valeurs de composants «ComponentValueList» est spécifiée au 24.16.

26.8 La liste «SetValue» ne se réduira à l'ensemble vide "{" "}" que si:

a) toutes les séquences «ComponentType» de la production «SetType» comportent la déclaration «DEFAULT» ou «OPTIONAL», et toutes les valeurs sont omises;

b) la notation du type était «SET{ }».

26.9 Une valeur «NamedValue» figurera pour chaque type «NamedType» du type «SetType» ne comportant pas la déclaration «OPTIONAL» ou «DEFAULT».

NOTE – Ces valeurs «NamedValue» peuvent figurer dans n'importe quel ordre.

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

27.1 Un type ensemble-de (voir 3.8.59) est défini à partir d'autres types par la notation «SetOfType»:

SetOfType ::=
SET OF Type

27.2 Les types ensemble-de portent tous l'étiquette de classe universelle numéro 17.

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

27.3 Une valeur de type ensemble-de est déclarée par la notation «SetOfValue»:

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

La liste de valeurs «ValueList» est spécifiée au 25.3.

La notation "{" "}" est utilisée quand la valeur ensemble-de est une liste vide.

27.4 Chaque valeur «Value» de la liste «ValueList» sera du type spécifié dans la déclaration «SetOfType».

NOTE 1 – L'ordre des valeurs n'est porteur d'aucun contenu sémantique.

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

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

28 Notation des types choix (choice types)

28.1 Un type choix (voir 3.8.13) est défini à partir d'autres types par la notation «ChoiceType»:

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

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

RootAlternativeTypeList ::= AlternativeTypeList

ExtensionAdditionAlternatives ::= "," ExtensionAdditionAlternativesList | empty

ExtensionAdditionAlternativesList ::= ExtensionAdditionAlternative |
ExtensionAdditionGroupAlternativesList "," ExtensionAdditionAlternative

ExtensionAdditionAlternative ::= ExtensionAdditionGroupAlternatives | NamedType

ExtensionAdditionGroupAlternatives ::= "[[" AlternativeTypeList "]]"

AlternativeTypeList ::=
NamedType |
AlternativeTypeList "," NamedType

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

28.2 Les types définis dans les productions «AlternativeTypeList» d'une production «AlternativeTypeLists» porteront des étiquettes distinctes (voir l'article 30). Si l'étiquetage automatique est en vigueur et que les types nommés «NamedType» dans la racine d'extension n'ont pas d'étiquettes, alors aucun type «NamedType» de la liste «ExtensionAdditionAlternativesList» ne pourra être étiqueté.

NOTE – Lorsque l'étiquetage par défaut «TagDefault» du module dans lequel cette notation apparaît est «AUTOMATIC TAGS», il résulte des dispositions du 24.7 que les étiquettes générées seront distinctes.

28.3 Lorsque la production «AlternativeTypeLists» apparaît dans la définition d'un module pour lequel l'étiquetage automatique a été choisi (voir 12.3), et si aucune des occurrences «NamedType» de cette liste ne contient un «Type» étiqueté par une production «TaggedType», alors l'étiquetage automatique est adopté pour l'ensemble de la liste «AlternativeTypeLists»; sinon, l'étiquetage automatique n'est pas appliqué. Lorsqu'il est adopté, l'étiquetage automatique d'une liste «AlternativeTypeLists» est appliqué à chaque type nommé «NamedType» de la liste en remplaçant le «Type» déclaré à l'origine dans la production «NamedType» par une occurrence «TaggedType» telle que celle-ci est spécifiée au 24.9.

28.4 L'étiquette de tout nouveau type «NamedType» ajouté à la liste «ExtensionAdditionAlternativesList» sera canoniquement supérieure (voir 8.4) à celle des autres types de la liste «ExtensionAdditionAlternativesList» et sera celle du dernier type «NamedType» de la liste «ExtensionAdditionAlternativesList».

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

28.6 Lorsque le type choix ne possède pas de marqueur d'extension et qu'il est utilisé dans un cas où la présente Recommandation | Norme internationale impose aux types de porter des étiquettes distinctes (voir 24.5 à 24.6, 26.3 et 28.2), la disposition s'appliquera à toutes les étiquettes possibles des valeurs du type choix. Les exemples suivants, dans lesquels on suppose que l'étiquetage par défaut «TagDefault» n'est pas l'étiquetage automatique «AUTOMATIC TAGS», illustrent cette prescription.

EXEMPLES

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

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

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

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

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

3 (INCORRECT)

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

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

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

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

28.7 Les identificateurs «identifier» de toutes les productions «NamedType» d'une même liste «AlternativeTypeLists» seront tous distincts.

28.8 Une valeur de type choix est déclarée par la notation «ChoiceValue»:

```
ChoiceValue ::= identifier ":" Value
```

28.9 «Value» sera la notation d'une valeur du type de la liste «AlternativeTypeLists» désigné par «identifier».

29 Notation des types sélection (selection types)

29.1 Un type sélection (voir 3.8.55) est défini par la notation «SelectionType»:

```
SelectionType ::= identifier "<" Type
```

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

29.2 Lorsque le type «SelectionType» est utilisé comme type nommé «NamedType», son identificateur «identifier» sera celui de ce type nommé.

29.3 Lorsque le type «SelectionType» est utilisé comme type «Type», son identificateur «identifier» n'est pas utilisé et le type désigné est celui de la forme sélectionnée.

29.4 La notation pour une valeur de type sélection sera la notation d'une valeur du type désigné par le type «SelectionType».

30 Notation des types étiquetés (tagged types)

Un type étiqueté «TaggedType» (voir 3.8.64) est un type nouveau, isomorphe d'un type existant, mais portant une étiquette différente. Le type étiqueté est principalement utilisé là où la présente Recommandation | Norme internationale spécifie l'emploi de types portant des étiquettes distinctes (voir 24.5 à 24.6, 26.3 et 28.2). L'adoption de l'étiquetage automatique «AUTOMATIC TAGS» comme étiquetage par défaut «TagDefault» dans un module permet d'accomplir un tel étiquetage sans que la notation de type étiqueté n'apparaisse explicitement dans le module.

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

30.1 Un type étiqueté est défini par la notation «TaggedType»:

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

Tag ::= "[" Class ClassNumber "]"

ClassNumber ::=
        number |
        DefinedValue
```

```
Class ::=
        UNIVERSAL                       |
        APPLICATION                     |
        PRIVATE                         |
        empty
```

30.2 La référence «valuereference» de la valeur «DefinedValue» sera du type entier et se verra affecter une valeur non négative.

30.3 Le type nouveau est isomorphe de l'ancien, mais porte une étiquette de la classe «Class» et de numéro «ClassNumber», à moins que la classe ne soit vide (option «empty»), ce qui correspond à une classe spécifique au contexte, auquel cas l'étiquette porte seulement un numéro «ClassNumber».

30.4 Seuls les types définis dans la présente Recommandation | Norme internationale peuvent porter une étiquette de la classe universelle «UNIVERSAL».

NOTE 1 – L'utilisation d'étiquettes de la classe universelle fait l'objet d'accords périodiques entre l'ISO et l'UIT-T.

NOTE 2 – Le paragraphe C.2.12 comporte des directives et des conseils stylistiques relatifs à l'utilisation des classes d'étiquettes.

30.5 L'étiquetage est toujours implicite ou explicite. L'étiquetage implicite indique, pour les règles de codage qui en offrent le choix, qu'il n'est pas nécessaire d'identifier explicitement en transfert l'étiquette d'origine du «Type» dans le type étiqueté.

NOTE – Il peut être utile de conserver l'ancienne étiquette si celle-ci est de la classe universelle et qu'elle identifie donc de façon non ambiguë l'ancien type sans connaître la définition en notation ASN.1 du nouveau type. Toutefois, l'utilisation de l'étiquetage implicite permet de minimiser le nombre d'octets à transférer. Un exemple de codage utilisant la déclaration d'étiquetage «IMPLICIT» est donné dans la Rec. UIT-T X.690 | ISO/CEI 8825-1.

30.6 La structure d'étiquetage spécifie un étiquetage explicite si l'une des conditions suivantes est vérifiée:

a) la forme «Tag EXPLICIT Type» est utilisée;

b) la forme «Tag Type» est utilisée et l'étiquetage par défaut «TagDefault» du module est explicite «EXPLICIT TAGS» ou vide «empty»;

c) la forme «Tag Type» est utilisée et l'étiquetage par défaut «TagDefault» du module est implicite «IMPLICIT TAGS» ou automatique «AUTOMATIC TAGS», mais le type défini par «Type» est un type choix, un type ouvert ou une référence muette «DummyReference» (voir le paragraphe 8.3 de la Rec. UIT-T X.683 | ISO/CEI 8824-4).

Dans tous les autres cas, la structure d'étiquetage spécifie un étiquetage implicite.

30.7 Si la classe indiquée «Class» est vide (option «empty»), l'utilisation de l'étiquette «Tag» n'est soumise à aucune autre restriction que l'obligation spécifiée par les paragraphes 24.5 à 24.6, 26.3 et 28.2 imposant d'avoir des étiquettes distinctes.

30.8 La déclaration d'étiquetage implicite «IMPLICIT» ne sera pas utilisée si le «Type» indiqué est un type choix, un type ouvert ou une référence muette «DummyReference» (voir le paragraphe 8.3 de la Rec. UIT-T X.683 | ISO/CEI 8824-4).

30.9 Une valeur de type «TaggedType» est déclarée par la notation «TaggedValue»:

```
TaggedValue ::= Value
```

où «Value» est la notation d'une valeur du type «Type» dans la notation «TaggedType».

NOTE – L'étiquette «Tag» n'apparaît pas dans cette notation.

31 Notation du type identificateur d'objet (object identifier type)

31.1 Le type identificateur d'objet (voir 3.8.47) est déclaré au moyen de la notation «ObjectIdentifierType»:

```
ObjectIdentifierType ::=
        OBJECT IDENTIFIER
```

31.2 Ce type porte l'étiquette numéro 6 de la classe universelle.

31.3 Une valeur d'identificateur d'objet est déclarée par la notation «ObjectIdentifierValue»:

```
ObjectIdentifierValue ::=
        "{" ObjIdComponentList "}"                      |
        "{" DefinedValue ObjIdComponentList "}"

ObjIdComponentList ::=
        ObjIdComponent                                  |
        ObjIdComponent ObjIdComponentList

ObjIdComponent ::=      NameForm        |
                        NumberForm      |
                        NameAndNumberForm

NameForm ::= identifier

NumberForm ::= number | DefinedValue

NameAndNumberForm ::=
        identifier "(" NumberForm ")"
```

31.4 La référence «valuereference» de la valeur «DefinedValue»de la notation «NumberForm» sera du type entier et se verra affecter une valeur non négative.

31.5 La référence «valuereference» de la valeur «DefinedValue» de la notation «ObjectIdentifierValue» sera du type identificateur d'objet.

31.6 La notation «NameForm» ne sera utilisée que pour les composants d'identificateur d'objet dont la valeur numérique et l'identificateur sont spécifiés dans les Annexes A à C de la Rec. UIT-T X.660 | ISO/CEI 9834-1 et correspondra à l'un des identificateurs spécifiés dans les Annexes A à C de la Rec. UIT-T X.660 | ISO/CEI 9834-1. Lorsque la Rec. UIT-T X.660 | ISO/CEI 9834-1 spécifie des identificateurs synonymes, tout synonyme peut être utilisé avec la même sémantique. Lorsqu'un même nom est à la fois un identificateur spécifié dans la Rec. UIT-T X.660 | ISO/CEI 9834-1 et dans une valeur de référence ASN.1 au sein du module contenant la forme de nom «NameForm», le nom au sein de l'objet identificateur sera traité comme un identificateur de la Rec. UIT-T X.660 | ISO/CEI 9834-1.

31.7 Le numéro «number» de la notation «NumberForm» sera la valeur numérique affectée au composant de l'identificateur d'objet.

31.8 Le champ «identifier» de la notation «NameAndNumberForm» sera spécifié quand une valeur numérique est affectée au composant de l'identificateur d'objet.

NOTE – Les autorités affectant des valeurs numériques aux composants d'identificateur d'objet sont identifiées dans les annexes de la Rec. UIT-T X.660 | ISO/CEI 9834-1.

31.9 La sémantique associée à une valeur d'identificateur d'objet est définie dans la Rec. UIT-T X.660 | ISO/CEI 9834-1.

31.10 La partie significative du composant identificateur d'objet est la forme «NameForm» ou la forme «NumberForm» à laquelle il se réduit et qui fournit la valeur numérique pour le composant identificateur d'objet. La valeur numérique du composant d'identificateur d'objet figure toujours dans une instance de la notation de valeur de l'identificateur d'objet, sauf pour les arcs spécifiés dans les Annexes A à C de la Rec. UIT-T X.660 | ISO/CEI 9834-1.

31.11 Quand la notation «ObjectIdentifierValue» comprend une valeur «DefinedValue» pour une valeur d'identificateur d'objet, la liste des composants de l'identificateur d'objet à laquelle elle se réfère s'ajoute par préfixation aux composants figurant explicitement dans la valeur.

NOTE – La Rec. UIT-T X.660 | ISO/CEI 9834-1 recommande d'attribuer également un descripteur d'objet chaque fois qu'une valeur d'identificateur d'objet est attribuée dans le but d'identifier un objet.

EXEMPLES

Avec les identificateurs affectés selon les dispositions de la Rec. UIT-T X.660 | ISO/CEI 9834-1, les valeurs:

```
{ iso standard 8571 pci (1) }
```

et

```
{ 1 0 8571 1 }
```

identifient tous deux un même objet, «pci», défini dans la norme ISO 8571.

Avec la définition additionnelle suivante:

ftam OBJECT IDENTIFIER ::= { iso standard 8571 }

la valeur suivante est équivalente aux deux premières valeurs:

{ ftam pci(1) }

32 Notation du type valeur de donnée de présentation enchâssé (embedded-pdv type)

32.1 Un type valeur de donnée de présentation enchâssée (voir 3.8.21) est défini par la notation «EmbeddedPDVType»:

EmbeddedPDVType ::= EMBEDDED PDV

32.2 Ce type porte l'étiquette numéro 11 de la classe universelle.

NOTE – Lorsque la négociation de la couche présentation est utilisée, le type «EMBEDDED PDV» assure la même fonction que le type «EXTERNAL» (plus des fonctions additionnelles), mais le train binaire transmis sera différent. Il est recommandé dans ce cas de remplacer dans les prochaines versions des protocoles applicatifs le type «EXTERNAL» par le type choix «CHOICE{external EXTERNAL, embedded-pdv EMBEDDED PDV}». L'Annexe C de la Rec. UIT-T X.681 | ISO/CEI 8824-2 propose de remplacer similairement l'utilisation du type «EXTERNAL» dans d'autres cas où la négociation de la couche de présentation n'est pas utilisée.

32.3 Ce type regroupe des valeurs constituées:

a) du codage d'une valeur de donnée unique, pouvant être une valeur d'un type ASN.1 mais ne l'étant pas nécessairement;

b) de l'identification (jointe ou séparée):

1) d'une classe de valeurs contenant cette valeur de donnée (une syntaxe abstraite);

2) du codage utilisé (la syntaxe de transfert) pour distinguer cette valeur de donnée des autres valeurs de la même classe.

NOTE 1 – La valeur de donnée peut être une valeur d'un type ASN.1, ou être par exemple le codage d'une image fixe ou animée. L'identification consiste en un ou deux identificateurs d'objet, ou désigne un contexte de présentation OSI permettant d'identifier la syntaxe abstraite et la syntaxe de transfert.

NOTE 2 – L'identification de la syntaxe abstraite et/ou du codage peut également être assurée par le concepteur de l'application sous la forme d'une valeur fixe, auquel cas elle ne sera pas codée dans une instance de communication.

32.4 Le type valeur de donnée de présentation enchâssée possède un type associé, qui sert à préciser la définition des valeurs abstraites de ce type, et qui sert également de base à la notation de ses valeurs et sous-types.

32.5 En supposant un environnement d'étiquetage automatique, le type associé servant à la définition des valeurs et au sous-typage est le suivant (les commentaires ont force de norme):

```
SEQUENCE {
    identification                  CHOICE {
        syntaxes                        SEQUENCE {
            abstract                        OBJECT IDENTIFIER,
            transfer                        OBJECT IDENTIFIER }
            -- Identificateurs d'objet de la syntaxe abstraite et de la syntaxe de transfert --,

        syntax                          OBJECT IDENTIFIER
            --- Identificateur d'objet unique identifiant la classe et le codage --,

        presentation-context-id         INTEGER
            -- (Ne s'applique que dans les environnements OSI)
            -- Le contexte de présentation négocié identifie la classe de la valeur de son codage --,

        context-negotiation             SEQUENCE {
            presentation-context-id         INTEGER,
            transfer-syntax                 OBJECT IDENTIFIER }
            -- (Ne s'applique que dans les environnements OSI)
            -- Négociation de contexte en cours pour identifier la classe de la valeur et son codage --,
```

```
        transfer-syntax                 OBJECT IDENTIFIER
            -- La classe de la valeur (une spécification indiquant par exemple qu'il s'agit d'une valeur
            -- d'un type ASN.1) est fixée par le concepteur de l'application (et est donc connue à la fois
            -- de l'expéditeur et du destinataire). Ce cas est prévu avant tout pour prendre en charge le
            -- chiffrement sélectif par champ (ou d'autres transformations de codage) d'un type ASN.1 --,

        fixed                           NULL
            -- La valeur de donnée est une valeur d'un type ASN.1 fixe
            -- (qui est donc connue à la fois de l'expéditeur et du destinataire) -- },

    data-value-descriptor               ObjectDescriptor  OPTIONAL
            -- Il s'agit de l'identification en langage naturel de la classe de la valeur --,
                                        data-value OCTET STRING }

( WITH COMPONENTS {
    ... ,
    data-value-descriptor  ABSENT } )
```

NOTE – Le type valeur de donnée de présentation enchâssée ne permet pas d'inclure une valeur de descripteur de valeur de donnée «data-value-descriptor». Toutefois, la définition du type associé donnée ici souligne les points communs existant entre le type valeur de donnée de présentation enchâssée, le type externe et le type chaîne de caractères à alphabet non restreint.

32.6 Lorsque l'identification est donnée par la forme «presentation-context-id», la valeur entière sera l'identificateur d'un contexte de présentation dans l'ensemble de contextes définis. Cette forme ne sera pas utilisée dans les primitives de demande «P-CONNECT» et de demande «P-ALTER-CONTEXT» lorsque ces primitives proposent l'addition ou la suppression du contexte de présentation identifié.

NOTE – La forme «presentation-context-id» ne peut pas être utilisée pour le contexte de présentation, même lorsqu'une seule syntaxe de transfert a été proposée pour le contexte de présentation dans la liste de définition des contextes de présentation.

32.7 L'identification par la forme «context-negotiation» ne sera utilisée que dans les primitives de demande «P-CONNECT» et de demande «P-ALTER-CONTEXT», et la valeur entière sera l'identificateur d'un contexte de présentation que ces primitives proposent d'ajouter à l'ensemble de contextes définis. L'identificateur d'objet «transfer-syntax» identifiera une syntaxe de transfert pour ce contexte de présentation, utilisée pour coder la valeur.

32.8 Une valeur de type valeur de donnée de présentation enchâssée est déclarée par la notation de valeur du type associé défini au 32.5 dans laquelle la valeur «data-value» du type chaîne d'octets «OCTET STRING» est codée conformément à la syntaxe de transfert spécifiée par le champ «identification».

```
EmbeddedPdvValue ::= SequenceValue   -- valeur du type associé défini au 32.5
```

32.9 EXEMPLE 1 – Lorsqu'un concepteur d'application souhaite que le codage soit indépendant de tout environnement de présentation (et qu'il soit donc possible de le retransmettre, de le stocker et de l'extraire sans modification), il doit interdire l'utilisation des formes d'identification «presentation-context-id» et «context-negotiation». Il peut y parvenir en écrivant:

```
EMBEDDED PDV (WITH COMPONENTS {
        ... ,
        identification  (WITH COMPONENTS {
                ... ,
                presentation-context-id         ABSENT,
                context-negotiation             ABSENT } ) } )
```

32.10 EXEMPLE 2 – S'il faut forcer le choix d'une seule option, par exemple celui de la forme «syntaxes», on peut y parvenir en écrivant:

```
EMBEDDED PDV (WITH COMPONENTS {
        ... ,
        identification  (WITH COMPONENTS {
            syntaxes  PRESENT } ) } )
```

33 Notation du type externe (external type)

33.1 Un type externe (voir 3.8.37) est défini par la notation «ExternalType»:

```
ExternalType ::= EXTERNAL
```

33.2 Ce type porte l'étiquette numéro 8 de la classe universelle.

33.3 Ce type regroupe des valeurs constituées:

a) du codage d'une valeur de donnée simple, pouvant être une valeur d'un type ASN.1 mais ne l'étant pas nécessairement;

b) de l'identification:

1) d'une classe de valeurs contenant cette valeur de donnée (une syntaxe abstraite); et

2) du codage utilisé (la syntaxe de transfert) pour distinguer cette valeur de donnée des autres valeurs de la même classe;

c) (optionnellement) d'un descripteur d'objet assurant une description en langage naturel de la classe de la valeur de donnée. Le descripteur d'objet optionnel ne sera présent que si le commentaire associé à l'utilisation de la notation «ExternalType» le permet explicitement.

NOTE – La Note 1 du 32.3 s'applique également au type externe.

33.4 Le type externe possède un type associé, qui sert à préciser la définition des valeurs abstraites de ce type, et qui sert également de base à la notation de ses valeurs et sous-types.

NOTE – Les règles de codage peuvent définir un type différent utilisé pour en dériver les codages, ou spécifier les codages sans faire référence à un quelconque type associé. En particulier, les règles de codage de base (BER) utilisent un type séquence équivalent identique à celui qui est présent dans la définition du type externe de la Rec. X.208 du CCITT | ISO/CEI 8824, et les codages des valeurs externes par les règles BER restent inchangés.

33.5 En supposant un environnement d'étiquetage automatique, le type associé servant à la définition des valeurs et au sous-typage est le suivant (les commentaires ont force de norme):

```
SEQUENCE {
    identification                  CHOICE {
        syntaxes                        SEQUENCE {
            abstract                        OBJECT IDENTIFIER,
            transfer                        OBJECT IDENTIFIER }
            -- Identificateurs d'objet de la syntaxe abstraite et de la syntaxe de transfert --,

        syntax                          OBJECT IDENTIFIER
            -- Identificateur d'objet unique identifiant la classe et le codage --,

        presentation-context-id         INTEGER
            -- (Ne s'applique que dans les environnements OSI)
            -- Le contexte de présentation négocié identifie la classe de la valeur et son codage --,

        context-negotiation             SEQUENCE {
            presentation-context-id         INTEGER,
            transfer-syntax                 OBJECT IDENTIFIER }
            -- (Ne s'applique que dans les environnements OSI)
            -- Négociation de contexte en cours pour identifier la classe de la valeur et son codage --,

        transfer-syntax                 OBJECT IDENTIFIER
            -- La classe de la valeur (une spécification indiquant par exemple qu'il s'agit d'une valeur
            -- d'un type ASN.1) est fixée par le concepteur de l'application (et est donc connue à la fois
            -- de l'expéditeur et du destinataire). Ce cas est prévu avant tout pour prendre en charge le
            -- chiffrement sélectif par champ (ou d'autres transformations de codage) d'un type ASN.1 --,

        fixed                           NULL
            -- La valeur de donnée est une valeur d'un type ASN.1 fixe
            -- (qui est donc connu à la fois de l'expéditeur et du destinataire) -- },

    data-value-descriptor           ObjectDescriptor  OPTIONAL
            -- Il s'agit de l'identification en langage naturel de la classe de la valeur --,
                                    data-value OCTET STRING }
```

```
( WITH COMPONENTS {
    ... ,
    identification  (WITH COMPONENTS {
        ... ,
        syntaxes                                ABSENT,
        transfer-syntax                         ABSENT,
        fixed                   ABSENT } ) } )
```

NOTE – Le type externe ne permet pas l'utilisation des formes «syntaxes», «transfer-syntax» et «fixed» de la notation «identification», afin de préserver la compatibilité amont avec le type externe défini dans la Rec. X.208 du CCITT | ISO/CEI 8824. Les concepteurs d'applications ayant besoin de faire appel à ces formes devront utiliser le type valeur de donnée de présentation enchâssée. La définition du type associé donnée ici souligne les points communs existant entre le type externe, le type chaîne de caractères à alphabet non restreint et le type valeur de donnée de présentation enchâssée.

33.6 Les dispositions des paragraphes 32.6 et 32.7 s'appliquent également au type externe.

33.7 Une valeur de type externe est déclarée par la notation de valeur du type associé défini au 33.5, où la valeur «data-value» de type chaîne d'octets «OCTET STRING» représente un codage utilisant la syntaxe de transfert spécifié dans «identification».

```
ExternalValue ::= SequenceValue    -- valeur du type associé défini au 33.5
```

NOTE – Pour des raisons historiques, les règles de codage permettent de transférer des valeurs enchâssées de type externe «EXTERNAL» codés sur un nombre de bits qui n'est pas multiple de huit. De telles valeurs ne peuvent pas être représentées par une notation de valeur au moyen du type associé ci-dessus.

34 Les types chaînes de caractères (character string types)

Ces types représentent les chaînes de caractères constituées à partir d'un répertoire de caractères spécifié donné. Il est normal de définir un répertoire de caractères et son codage en les représentant sous la forme de cellules disposées en une ou plusieurs grilles, chaque cellule correspondant à un caractère du répertoire. D'habitude, on affecte également à chaque cellule un symbole graphique et un nom de caractère, bien que dans certains répertoires, des cellules soient laissées vides ou qu'elles aient un nom mais pas de forme imprimable (on peut citer comme exemples de telles cellules ayant un nom mais pas de forme affectée les caractères de contrôle comme EOF de la norme ISO/CEI 646 et les caractères d'espacement comme THIN-SPACE and EN-SPACE de la norme ISO/CEI 10646-1).

L'expression **caractère abstrait** désigne la totalité de l'information associée à une cellule d'une grille de répertoire de caractères. Cette information décrit un caractère abstrait distinct même si cette information est vide (pas de symbole graphique ni de nom affecté à la cellule).

La notation de valeurs ASN.1 des types chaînes de caractères possède trois variantes (combinables) spécifiées comme suit:

a) une représentation imprimée des caractères de la chaîne utilisant les symboles graphiques attribués aux caractères, y compris éventuellement les caractères d'espacement; c'est la notation «cstring»;

   NOTE 1 – Une telle représentation peut être ambiguë lorsqu'un même symbole graphique est attribué à plus d'un caractère du répertoire.

   NOTE 2 – Une telle représentation peut être ambiguë lorsque la chaîne comprend des caractères d'espacement et que la notation est imprimée dans une police à espacement proportionnel.

b) une liste des caractères représentant la valeur de la chaîne sous la forme d'une suite des références de valeurs ASN.1 affectées aux caractères; un ensemble de telles références de valeurs est défini dans le module ASN.1 de l'article 37 pour les répertoires de caractères de la norme ISO/CEI 10646-1 et «IA5String»; cette représentation n'est utilisable pour les autres répertoires de caractères que si l'utilisateur leur affecte des références de valeurs au moyen de la notation décrite au point a) ci-dessus ou au point c) ci-dessous;

c) une liste des caractères représentant la valeur de la chaîne sous la forme d'une suite de positions de cellules dans la ou les grilles du répertoire; cette forme n'est utilisable que pour des chaînes «IA5String» (alphabet international n° 5), «UniversalString» (chaîne universelle), «UTF8String» (format de transformation UCS 8) et «BMPString» (table multilingue).

35 Notation des types chaînes de caractères

35.1 Un type chaîne de caractères (voir 3.8.11) est désigné au moyen de la notation:

CharacterStringType ::= RestrictedCharacterStringType | UnrestrictedCharacterStringType

«RestrictedCharacterStringType» est la notation du type de chaîne de caractères à alphabet restreint défini à l'article 36. «UnrestrictedCharacterStringType» est la notation du type de chaîne de caractères à alphabet non restreint défini au 39.1.

35.2 Les étiquettes portées par les différents types de chaînes de caractères à alphabet restreint sont spécifiées au 36.1. L'étiquette du type de chaîne de caractères à alphabet non restreint est spécifiée au 39.2.

35.3 Une valeur de chaîne de caractères est déclarée par la notation suivante:

CharacterStringValue ::= RestrictedCharacterStringValue | UnrestrictedCharacterStringValue

La valeur «RestrictedCharacterStringValue» est définie au 36.7. «UnrestrictedCharacterStringValue» est la notation des valeurs de chaîne de caractères à alphabet non restreint; elle est défini au 39.6.

36 Définition des types chaînes de caractères à alphabet restreint

Cet article définit les types dont les valeurs sont limitées aux séquences de zéro, un ou plusieurs caractères appartenant à une collection de caractères déterminée d'un jeu donné. Ce type de chaîne de caractères à alphabet restreint est désigné au moyen de la notation «RestrictedCharacterStringType»:

```
RestrictedCharacterStringType ::= BMPString       |
                                  GeneralString   |
                                  GraphicString   |
                                  IA5String       |
                                  ISO646String    |
                                  NumericString   |
                                  PrintableString |
                                  TeletexString   |
                                  T61String       |
                                  UniversalString |
                                  UTF8String      |
                                  VideotexString  |
                                  VisibleString
```

Chaque forme possible du type «RestrictedCharacterStringType» est définie en spécifiant:

a) l'étiquette qui lui est affectée;

b) un nom (par exemple «NumericString») par lequel le type est désigné;

c) les caractères du jeu utilisé pour définir le type, par référence à un tableau énumérant les différentes formes graphiques des caractères, ou par référence à un numéro d'enregistrement du registre international des jeux de caractères codés de l'ISO (voir le *Registre international de l'ISO des jeux de caractères codés à utiliser avec une séquence d'échappement*), ou par référence à la norme ISO/CEI 10646-1.

36.1 Le Tableau 3 donne la liste des noms désignant chacun des types de chaînes de caractères, le numéro de l'étiquette de la classe universelle affectée à ce type, le tableau, article ou numéro d'enregistrement de définition, et, le cas échéant, le numéro de la Note relative à cette entrée du tableau. Lorsqu'un nom de type possède un synonyme, celui-ci est indiqué entre parenthèses.

NOTE – L'étiquette affectée à chaque type de chaînes de caractères l'identifie de manière non ambiguë. Il est à noter toutefois que si la notation ASN.1 est utilisée pour définir de nouveaux types à partir de ce type (notamment en utilisant la déclaration IMPLICIT) il sera peut-être impossible de reconnaître ces types sans connaître la définition de type ASN.1.

Tableau 3 – Liste des types de chaînes de caractères à alphabet restreint

Nom utilisé pour référencer le type de chaîne	Numéro dans la classe universelle	Numéro d'enregistrement, numéro du tableau ou article de la Rec. UIT-T X.680 | ISO/CEI 8824-1 de définition[a)]	Notes
UTF8String (format de transformation UCS 8)	12	Voir 36.13	
NumericString (chaîne numérique)	18	Tableau 4	Note 1
PrintableString (chaîne imprimable)	19	Tableau 5	Note 1
TeletexString (T61String) [chaîne télétex (ou chaîne T61)]	20	6, 87, 102, 103, 106, 107, 126, 144, 150, 153, 156, 164, 165, 168 + SPACE + DELETE	Note 2
VideotexString (chaîne vidéotex)	21	1, 13, 72, 73, 87, 89, 102, 108, 126, 128, 129, 144, 150, 153, 164, 165, 168 + SPACE + DELETE	Note 3
IA5String (chaîne alphabet international nº 5)	22	1, 6 + SPACE + DELETE	
GraphicString (chaîne graphique)	25	Tous les jeux graphiques + SPACE	
VisibleString (ISO646String) [chaîne visible (chaîne ISO 646)]	26	6 + SPACE	
GeneralString (chaîne générale)	27	Tous les jeux graphiques et caractères + SPACE + DELETE	
UniversalString (chaîne universelle)	28	Voir 36.6	
BMPString (chaîne multilingue)	30	Voir 36.12	

a) Les numéros d'enregistrement de définition sont énumérés dans le Registre international de l'ISO des jeux de caractères codés à utiliser avec les séquences d'échappement.

NOTE 1 – Le style typographique, le corps, la couleur, la graisse et les autres caractéristiques d'impression ou d'affichage ne sont pas significatifs.

NOTE 2 – Les entrées correspondant à ces numéros d'enregistrement renvoient à la Rec. T.61 du CCITT, pour les règles concernant leur utilisation. Les entrées de registre 6 et 156 peuvent être utilisées à la place des entrées 102 et 103.

NOTE 3 – Les entrées correspondant à ces numéros d'enregistrement assurent les fonctionnalités définies dans les Rec. T.100 du CCITT et Rec. UIT-T T.101.

NOTE 4 – La référence au registre 6 du «Registre international de l'ISO des jeux de caractères codés à utiliser avec la séquence d'échappement» est une référence indirecte à la norme ISO/CEI 646:1991. Il s'agit là d'une modification par rapport à la Rec. X.208 du CCITT | ISO/CEI 8824, qui donnait en référence le registre 2 (référence indirecte à la norme ISO 646:1973). S'il est nécessaire de faire référence dans une application au registre n° 2, il faudra recourir à d'autres moyens [utiliser par exemple une chaîne de caractères à alphabet non restreint (voir 39)] pour véhiculer l'ancienne définition de la chaîne visible ou pour faire référence à la Rec. X.208 du CCITT | ISO/CEI 8824.

36.2 Le Tableau 4 donne la liste des caractères susceptibles d'apparaître dans les chaînes numériques «NumericString» et dans leur syntaxe abstraite de caractères.

Tableau 4 – Chaînes numériques «NumericString»

Nom	Caractère graphique
Chiffres	0, 1, ... 9
Espacement	(Espacement)

36.3 Les valeurs d'identificateur d'objet et de descripteur d'objet suivantes ont été affectées pour identifier et décrire la syntaxe abstraite de caractères chaîne numérique «NumericString»:

{ joint-iso-itu-t asn1(1) specification(0) characterStrings(1) numericString(0) }

et

"NumericString character abstract syntax"

NOTE 1 – Cette valeur d'identificateur d'objet peut être utilisée dans des valeurs de type chaîne de caractères «CHARACTER STRING» lorsqu'il est nécessaire de transmettre l'identificateur du type de chaîne de caractères indépendamment de la valeur.

NOTE 2 – Une valeur de syntaxe abstraite du type chaîne numérique «NumericString» peut être codée:

a) par une des règles données dans la norme ISO/CEI 10646-1 pour le codage des caractères abstraits. Dans ce cas, la syntaxe de transfert de caractères est identifiée par l'identificateur d'objet associé aux règles dans l'Annexe M de la norme ISO/CEI 10646-1;

b) par les règles de codage ASN.1 du type prédéfini «NumericString». Dans ce cas, la syntaxe de transfert de caractères est identifiée par la valeur d'identificateur d'objet {joint-iso-ccitt asn1(1) basic-encoding(1)}.

36.4 Le Tableau 5 donne la liste des caractères susceptibles d'apparaître dans les chaînes imprimables «PrintableString» et dans leur syntaxe abstraite de caractères.

Tableau 5 – Chaînes imprimables «PrintableString»

Nom	Symbole
Majuscules	A, B, ... Z
Minuscules	a, b, ... z
Chiffres	0, 1, ... 9
Espacement	(space)
Apostrophe	'
Parenthèse gauche	(
Parenthèse droite	)
Signe plus	+
Virgule	,
Trait d'union	-
Point	.
Barre oblique	/
Deux-points	:
Signe égal	=
Point d'interrogation	?

36.5 Les valeurs d'identificateur d'objet et de descripteur d'objet suivantes ont été affectées pour identifier et décrire la syntaxe abstraite de caractères chaîne imprimable «PrintableString»:

{ joint-iso-itu-t asn1(1) specification(0) characterStrings(1) printableString(1) }

et

"PrintableString character abstract syntax"

NOTE 1 – Cette valeur d'identificateur d'objet peut être utilisée dans des valeurs de type chaîne de caractères «CHARACTER STRING» lorsqu'il est nécessaire de transmettre l'identificateur du type de chaîne de caractères indépendamment de la valeur.

NOTE 2 – Une valeur de syntaxe abstraite du type chaîne imprimable «PrintableString» peut être codée:

a) par une des règles données dans la norme ISO/CEI 10646-1 pour le codage des caractères abstraits. Dans ce cas, la syntaxe de transfert de caractères est identifiée par l'identificateur d'objet associé aux règles dans l'Annexe M de la norme ISO/CEI 10646-1;

b) par les règles de codage ASN.1 du type prédéfini «PrintableString». Dans ce cas, la syntaxe de transfert de caractères est identifiée par la valeur d'identificateur d'objet {joint-iso-ccitt asn1(1) basic-encoding(1)}.

36.6 Les caractères susceptibles d'apparaître dans les chaînes de type chaîne universelle «UniversalString» sont tous les caractères autorisés par la norme ISO/CEI 10646-1. L'utilisation de ce type impose le respect des spécifications de conformité indiquées dans cette norme, notamment en ce qui concerne la zone d'utilisation restreinte.

NOTE 1 – L'utilisation de ce type sans contrainte est déconseillée, car la conformité sera généralement difficile à assurer.

NOTE 2 – L'article 37 contient un module ASN.1 définissant un certain nombre de sous-types de ce type pour les «collections de caractères graphiques des sous-jeux» définies à l'Annexe A de la norme ISO/CEI 10646-1.

36.7 Une valeur de type chaîne de caractères à alphabet restreint est déclarée par la notation «cstring» (voir 11.11), «CharacterStringList», «Quadruple» ou «Tuple». «Quadruple» ne peut définir qu'une chaîne de caractères de longueur 1, et ne peut être utilisée que pour noter une valeur de type chaîne universelle «UniversalString», format de transformation UCS 8 «UTF8String» ou multilingue «BMPString». «Tuple» ne peut définir qu'une chaîne de caractères de longueur 1, et ne peut être utilisée que pour noter une valeur de type chaîne alphabet international nº 5 «IA5String».

```
RestrictedCharacterStringValue ::= cstring | CharacterStringList | Quadruple | Tuple

CharacterStringList ::=  "{" CharSyms "}"
CharSyms ::=  CharsDefn | CharSyms "," CharsDefn
CharsDefn ::=  cstring | Quadruple | Tuple | DefinedValue

Quadruple ::= "{"  Group  ","  Plane  ","  Row  ","  Cell "}"
Group     ::= number
Plane     ::= number
Row       ::= number
Cell      ::= number

Tuple ::= "{" TableColumn "," TableRow "}"
TableColumn ::= number
TableRow ::= number
```

NOTE 1 – La notation «cstring» ne peut être utilisée que sur un dispositif capable d'afficher les symboles graphiques des caractères figurant dans la valeur. Réciproquement, si le dispositif utilisé n'offre pas une telle capacité, le seul moyen de spécifier une valeur de chaîne de caractères utilisant de tels symboles est la notation de liste «CharacterStringList», sous réserve que la chaîne soit de type «UniversalString», «UTF8String», «BMPString» ou «IA5String», et seulement si la forme «DefinedValue» de «CharsDefn» est utilisée (voir 37.1.2).

NOTE 2 – L'article 37 définit un certain nombre de références «valuereference» désignant des caractères simples (chaînes de longueur 1) du type «UniversalString», «UTF8String» ou «IA5String».

EXEMPLE – Supposons que l'on souhaite spécifier une valeur de chaîne universelle «UniversalString» «abcΣdef» dans laquelle le caractère «Σ» ne peut pas être représenté sur le dispositif utilisé. Cette valeur peut alors s'écrire comme suit:

```
IMPORTS BasicLatin, greekCapitalLetterSigma FROM ASN1-CHARACTER-MODULE

        { joint-iso-itu-t asn1(1) specification(0) modules(0) iso10646(0) };

MyAlphabet ::= UniversalString (FROM (BasicLatin | greekCapitalLetterSigma))

mystring MyAlphabet ::= { "abc" , greekCapitalLetterSigma , "def" }
```

NOTE 3 – Lorsqu'on spécifie une valeur du type «UniversalString», «UTF8String» ou «BMPString», la notation «cstring» ne sera pas utilisée à moins que n'aient été résolues les ambiguïtés résultant de la similitude de forme de caractères différents.

EXEMPLE – La notation suivante en chaîne «cstring» ne doit pas être utilisée car les symboles graphiques «P», «O», «I», «N» et «T» existent dans les alphabets BASIC LATIN, CYRILLIC et BASIC GREEK et sont donc ambigus.

```
IMPORTS BasicLatin, Cyrillic, BasicGreek FROM ASN1-CHARACTER-MODULE

        { joint-iso-itu-t asn1(1) specification(0) modules(0) iso10646(0) };

MyAlphabet ::= UniversalString (FROM (BasicLatin | Cyrillic | BasicGreek))

mystring MyAlphabet ::= "POINT"
```

36.8 La valeur «DefinedValue» de «CharsDefn» renverra à une valeur de ce type.

36.9 Les numéros «number» dans les productions «Plane», «Row» et «Cell» seront inférieurs à 256, dans la production «Group» ils seront inférieurs à 128.

36.10 «Group» spécifie un groupe de l'espace de codage du système de codage universel UCS, «Plane» spécifie un plan du groupe, «Row» spécifie une rangée dans le plan, et «Cell» spécifie une cellule dans la rangée. Le caractère abstrait identifié par cette notation est celui de la cellule spécifiée par les valeurs «Group», «Plane», «Row», et «Cell». Dans tous les cas, l'ensemble des caractères autorisés peut être restreint par sous-typage.

NOTE – Les concepteurs d'applications doivent faire attention aux problèmes de conformité qui pourraient se poser lorsqu'ils utilisent des types chaînes de caractères ouverts tels que des chaînes générales «GeneralString» ou graphiques «GraphicString» sans leur appliquer de contraintes. Un texte précis sur la conformité est aussi nécessaire pour les types chaînes de caractères de taille limitée mais grande, tels que les chaînes télétex «TeletexString».

36.11 Le numéro «number» dans la production «TableColumn» sera compris entre zéro et sept, et le numéro «number» dans la production «TableRow» sera compris entre zéro et quinze. «TableColumn» désigne une colonne et «TableRow» une rangée de la grille des codes des caractères conformément à la disposition représentée Figure 1 de la norme ISO/CEI 2022. Cette notation n'est utilisée que pour les chaînes du type «IA5String» lorsque la grille de codage contient l'entrée de registre 1 en colonnes 0 et 1 et l'entrée de registre 6 dans les colonnes 2 à 7 (voir le *Registre international de l'ISO des jeux de caractères codés à utiliser avec les séquences d'échappement*).

36.12 Le type chaîne multilingue «BMPString» est un sous-type du type chaîne universelle «UniversalString» qui possède une étiquette en propre et modélise la table multilingue BMP (*basic multilingual plane*) de la grille ISO/CEI 10646-1 (les premières 64K - 2 cellules). Son type associé est:

UniversalString (Bmp)

où «Bmp» est défini dans le module «ASN1-CHARACTER-MODULE» (voir l'article 37) comme le sous-type du type «UniversalString» correspondant au nom de collection «BMP» défini à l'Annexe A de la norme ISO/CEI 10646-1.

NOTE 1 – Comme «BMPString» est un type prédéfini, il n'est pas défini dans le module «ASN1-CHARACTER-MODULE».

NOTE 2 – La définition du type «BMPString» en tant que type prédéfini a pour but de permettre aux règles de codage (comme les règles de base BER) obéissant à des contraintes d'utiliser un codage sur 16 bits plutôt que sur 32 bits.

NOTE 3 – Toutes les valeurs «BMPString» sont également des notations de valeurs valides des types «UniversalString» et «UTF8String».

36.13 Le type «UTF8String» est synonyme du type «UniversalString» au niveau abstrait et peut être utilisé chaque fois que ce dernier est utilisé (compte tenu des règles exigeant des étiquettes distinctes), mais il possède une étiquette différente et constitue un type distinct.

NOTE – Le codage est différent de celui d'une valeur «UniversalString» et sera moins verbeux pour la plupart des textes.

37 Dénomination des caractères et collections de caractères définis dans la norme ISO/CEI 10646

Le présent article définit un module ASN.1 prédéfini qui donne la définition d'un nom de référence de valeur pour chaque caractère défini dans la norme ISO/CEI 10646-1. Chacun de ces noms renvoie à une valeur de chaîne universelle «UniversalString» de taille 1. Ce module définit également un nom de référence de type pour chacune des collections de caractères de la norme ISO/CEI 10646-1, chacun de ces noms renvoyant à un sous-ensemble du type «UniversalString».

NOTE – Ces valeurs peuvent être utilisées pour noter les valeurs du type «UniversalString» et des sous-types qui en dérivent. Toutes les références de valeurs et de types spécifiées au 37.1 sont exportées, et doivent être importées par tout module qui les utilise.

37.1 Spécification du module «ASN1-CHARACTER-MODULE»

Ce module n'est pas reproduit ici en entier. Ce qui a été plutôt exposé est la manière dont il a été défini.

37.1.1 Le module commence comme suit:

```
ASN1-CHARACTER-MODULE {joint-iso-itu-t asn1(1) specification(0) modules(0) iso10646(0)}
DEFINITIONS ::= BEGIN
-- Toutes les références de valeurs et de types définies dans ce module sont implicitement exportées
-- et peuvent être importées par tout module.
-- Caractères de contrôle de la norme ISO/CEI 646

nul   IA5String ::= {0, 0}
soh   IA5String ::= {0, 1}
stx   IA5String ::= {0, 2}
etx   IA5String ::= {0, 3}
eot   IA5String :: := {0, 4}
```

```
enq  IA5String ::= {0, 5}
ack  IA5String ::= {0, 6}
bel  IA5String ::= {0, 7}
bs   IA5String ::= {0, 8}
ht   IA5String ::= {0, 9}
lf   IA5String ::= {0,10}
vt   IA5String ::= {0,11}
ff   IA5String ::= {0,12}
cr   IA5String ::= {0,13}
so   IA5String ::= {0,14}
si   IA5String ::= {0,15}
dle  IA5String ::= {1, 0}
dc1  IA5String ::= {1, 1}
dc2  IA5String ::= {1, 2}
dc3  IA5String ::= {1, 3}
dc4  IA5String ::= {1, 4}
nak  IA5String ::= {1, 5}
syn  IA5String ::= {1, 6}
etb  IA5String ::= {1, 7}
can  IA5String ::= {1, 8}
em   IA5String ::= {1, 9}
sub  IA5String ::= {1,10}
esc  IA5String ::= {1,11}
is4  IA5String ::= {1,12}
is3  IA5String ::= {1,13}
is2  IA5String ::= {1,14}
is1  IA5String ::= {1,15}
del  IA5String ::= {7,15}
```

37.1.2 Pour chaque entrée de chaque liste de noms de caractères graphiques (glyphes) figurant dans les articles 24 et 25 de la norme ISO/CEI 10646-1, le modèle inclut une déclaration de la forme:

```
<namedcharacter> BMPString ::= <tablecell>
    -- représentant le caractère <iso10646name>, voir la norme ISO/CEI 10646-1
```

où:

a) <iso10646name> est le nom du caractère dérivé de la liste donnée dans la norme ISO/CEI 10646-1;

b) <namedcharacter> est une chaîne obtenue en appliquant les procédures spécifiées au 37.2 au caractère <iso10646name>;

c) <tablecell> est le glyphe de la cellule du tableau de la norme ISO/CEI 10646-1 correspondant à l'entrée dans la liste.

EXEMPLE

```
latinCapitalLetterA  BMPString  ::=  {0, 0, 0, 65}
    -- représente le caractère A MAJUSCULE LATIN, voir la norme ISO/CEI 10646-1
greekCapitalLetterSigma BMPString  ::=  {0, 0, 3, 145}
    -- représente le caractère SIGMA MAJUSCULE GREC, voir la norme ISO/CEI 10646-1
```

37.1.3 Pour chaque nom de collection de caractères graphiques spécifié dans l'Annexe A de la norme ISO/CEI 10646-1, le module comporte une déclaration de la forme:

```
<namedcollectionstring> ::= BMPString
    (FROM ( <alternativelist>))
    -- représente la collection de caractères <collectionstring>,
    -- voir la norme ISO/CEI 10646-1.
```

où:

a) <collectionstring> est le nom de la collection de caractères affecté par la norme ISO/CEI 10646-1;

b) <namedcollectionstring> est formé en appliquant à <collectionstring> les procédures du 37.3;

c) <alternativelist> est formé en utilisant les caractères <namedcharacter> engendrés comme l'indique le paragraphe 37.2 pour chacun des caractères spécifiés dans la norme ISO/CEI 10646-1.

La référence de type résultante <namedcollectionstring> forme un sous-ensemble limité de la collection (voir les exemples didactiques de l'Annexe D).

NOTE – Un sous-ensemble limité est une liste de caractères appartenant à un sous-ensemble spécifié, par opposition à un sous-ensemble sélectionné, qui est une des collections de caractères énumérées à l'Annexe A de la norme ISO/CEI 10646-1, plus la collection latine de base.

EXEMPLE (partiel):

```
space BMPString ::= {0, 0, 0, 32}
exclamationMark BMPString ::= {0, 0, 0, 33}
quotationMark BMPString ::= {0, 0, 0, 34}
...          -- et ainsi de suite
tilde BMPString ::= {0, 0, 0, 126}

BasicLatin ::= BMPString
    (FROM (space
    | exclamationMark
    | quotationMark
    | ...          -- et ainsi de suite
    | tilde)
    )
-- représente la collection de caractères latins de base, voir la norme ISO/CEI 10646-1.
-- Les points de suspension, utilisés dans cet exemple par souci d'abréviation, signifient «et ainsi de
-- suite»; il ne faut pas les utiliser sous cette forme dans un véritable module ASN.1.
```

37.1.4 La norme ISO/CEI 10646-1 définit trois niveaux de réalisation. Tous les types définis dans le module «ASN1-CHARACTER-MODULE» à l'exception de «Level1» et de «Level2» sont par défaut conformes à une réalisation de niveau 3, car de tels types n'imposent aucune restriction quant à l'utilisation des caractères de combinaison. «Level1» indique qu'une réalisation de niveau 1 est requise; «Level2» indique qu'une réalisation de niveau 2 est requise, et «Level3» indique qu'une réalisation de niveau 3 est requise. Ces types sont définis dans le module «ASN1-CHARACTER-MODULE»:

```
Level1 ::= BMPString (FROM (ALL EXCEPT CombiningCharacters))

Level2 ::= BMPString (FROM (ALL EXCEPT CombiningCharactersB-2))

Level3 ::= BMPString
```

NOTE 1 – «CombiningCharacters» et «CombiningCharactersB-2» sont les chaînes de collections nommées <namedcollectionstring> correspondant respectivement aux caractères de combinaison (COMBINING CHARACTERS) et aux caractères de combinaison de type 2 (COMBINING CHARACTERS B-2) définies à l'Annexe A de la norme ISO/CEI 10646-1.

NOTE 2 – «Level1» et «Level2» sont utilisés à la suite d'un signe «IntersectionMark» (voir l'article 46) ou comme seule contrainte dans une spécification «ConstraintSpec». Un exemple en est donné au C.2.7.1.

NOTE 3 – Pour plus de détails, voir D.2.5.

37.1.5 Le module se termine par la déclaration:

```
END
```

37.1.6 Un équivalent défini par l'utilisateur de l'exemple donné au 37.1.3 pourrait être:

```
BasicLatin ::= BMPString (FROM (space..tilde))
-- représente la collection de caractères latins de base (BASIC LATIN) voir la norme ISO/CEI 10646-1.
```

37.2 Un nom <namedcharacter> est une chaîne dérivée d'un nom <iso10646name> (voir 37.1.2) en lui appliquant l'algorithme suivant:

a) chaque majuscule de <iso10646name> est mise en minuscule, sauf si elle est précédée par un caractère d'espacement, auquel cas elle reste inchangée;

b) les chiffres et les traits d'union restent inchangés;

c) les caractères d'espacement sont supprimés.

NOTE – L'algorithme ci-dessus, appliqué conjointement avec les directives d'appellation des caractères énoncées à l'Annexe K de la norme ISO/CEI 10646-1, produira toujours une notation de valeur non ambiguë pour chacun des noms de caractère figurant dans la norme ISO/CEI 10646-1.

EXEMPLE – Le caractère ISO/CEI 10646-1, rangée 0, cellule 60, appelé «signe inférieur à» (LESS-THAN SIGN), dont la représentation graphique est «<», peut être désigné par la chaîne de type «DefinedValue»:

less-thanSign

37.3 Le nom <namedcollectionstring> est la chaîne dérivée du nom <collectionstring> en lui appliquant l'algorithme suivant:

a) chaque majuscule du nom de la collection de la norme ISO/CEI 10646-1 est mise en minuscule, sauf si elle est précédée par un caractère d'espacement ou est la première lettre du nom, auquel cas elle reste inchangée;

b) les chiffres et les traits d'union restent inchangés;

c) les caractères d'espacement sont supprimés.

EXEMPLES

1 La collection identifiée dans l'Annexe A de la norme ISO/CEI 10646-1 sous le nom:

BASIC LATIN

a la référence de type ASN.1

BasicLatin

2 Un type chaîne de caractères composé des caractères de la collection BASIC LATIN, ainsi que de la collection BASIC ARABIC peut être défini de la manière suivante:

Ma-Chaine-De-Caracteres ::= BMPString (FROM (BasicLatin | BasicArabic))

NOTE – La formulation ci-dessus est nécessaire car la formulation apparemment plus simple:

Ma-Chaine-De-Caracteres ::= BMPString (BasicLatin | BasicArabic)

ne permettrait d'écrire que des chaînes dont tous les caractères appartiennent à la collection soit latine BASIC LATIN soit arabe BASIC ARABIC mais pas à un mélange des deux.

38 Ordre canonique des caractères

38.1 A des fins de sous-typage par la notation d'intervalle de valeurs «ValueRange» et d'utilisation possible par les règles de codage, un ordre canonique des caractères a été spécifié pour les types de chaînes universelles «UniversalString», multilingues «BMPString», numériques «NumericString», imprimables «PrintableString», visibles «VisibleString» et alphabet international nº 5 «IA5String».

38.2 Aux fins de ce seul article, les caractères sont en correspondance biunivoque avec les cellules d'une grille de codage, qu'un nom ou une forme de caractère ait été ou non affecté à ces cellules, qu'il s'agisse de caractères de contrôle ou de caractères imprimables, et qu'il s'agisse ou non de caractères de combinaison.

38.3 L'ordre canonique d'un caractère abstrait est défini par l'ordre canonique de sa cellule.

38.4 Pour les chaînes universelles «UniversalString», l'ordre canonique des cellules est défini par (voir la norme ISO/CEI 10646-1):

256*(256*(128*(numéro de groupe)+(numéro de plan))+(numéro de rangée))+(numéro de cellule)

Le jeu de caractères complet contient exactement 128*256*256*256 caractères. Les extrémités des intervalles de valeurs «ValueRange» dans les notations d'alphabet permis «PermittedAlphabet» (ou les différents caractères pris individuellement) peuvent être désignées soit en utilisant la référence de valeur ASN.1 définie dans le module «ASN1-CHARACTER-MODULE», soit (lorsque le symbole graphique n'est pas ambigu dans le contexte de la spécification) en indiquant le symbole graphique dans une chaîne «cstring» (le module «ASN1-CHARACTER-MODULE» est défini au 37.1). Il n'est pas possible de spécifier une cellule comme extrémité d'intervalle ou de la désigner individuellement lorsque aucun nom ou symbole graphique n'a été affecté à cette cellule.

38.5 Pour les chaînes multilingues «BMPString», l'ordre canonique des cellules est défini par (voir la norme ISO/CEI 10646-1):

256*(numéro de rangée)+(numéro de cellule)

Le jeu de caractères complet contient exactement 256*256 caractères. Les extrémités des intervalles de valeurs «ValueRange» dans les notations d'alphabet permis «PermittedAlphabet» (ou les différents caractères pris individuellement) peuvent être désignées soit en utilisant la référence de valeur ASN.1 définie dans le module «ASN1-CHARACTER-MODULE», soit (lorsque le symbole graphique n'est pas ambigu dans le contexte de la spécification) en indiquant le symbole graphique dans une chaîne «cstring». Il n'est pas possible de spécifier une cellule comme extrémité d'intervalle ou de la désigner individuellement lorsque aucun nom ou symbole graphique n'a été affecté à cette cellule.

38.6 Pour les chaînes de type numérique «NumericString», l'ordre canonique est défini dans un ordre croissant de gauche à droite par (voir le Tableau 4 du 36.2):

(espacement) 0 1 2 3 4 5 6 7 8 9

Le jeu de caractères complet contient exactement 11 caractères. L'extrémité d'un intervalle de valeur «ValueRange» (ou un caractère isolé) peut être désignée en indiquant le symbole graphique correspondant dans une chaîne «cstring».

NOTE – Cet ordre est le même que les caractères correspondants de la collection latine de base BASIC LATIN de la norme ISO/CEI 10646-1.

38.7 Pour les chaînes de type imprimable «PrintableString», l'ordre canonique est défini dans un ordre croissant de gauche à droite puis de haut en bas par (voir le Tableau 5 du 36.4):

(Espacement) (Apostrophe) (Parenthèse gauche) (Parenthèse droite) (Signe plus) (Virgule) (Trait d'union) (Point) (Barre oblique) 0123456789 (Deux-points) (Signe égal) (Point d'interrogation)
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

Le jeu de caractères complet contient exactement 74 caractères. L'extrémité d'un intervalle de valeur «ValueRange» (ou un caractère isolé) peut être désignée en indiquant le symbole graphique correspondant dans une chaîne «cstring».

NOTE – Cet ordre est le même que les caractères correspondants de la collection latine de base «BASIC LATIN» de la norme ISO/CEI 10646-1.

38.8 Pour les chaînes de type visible «VisibleString», l'ordre canonique des cellules est défini à partir du codage ISO 646 (appelé ISO 646 ENCODING) de la manière suivante:

(ISO 646 ENCODING) - 32

NOTE – C'est-à-dire que l'ordre canonique est celui des caractères correspondants dans les cellules 2/0 à 7/14 de la grille de codage ISO 646.

Le jeu de caractères complet contient exactement 95 caractères. L'extrémité d'un intervalle de valeur «ValueRange» (ou un caractère isolé) peut être désignée en indiquant le symbole graphique correspondant dans une chaîne «cstring».

38.9 Pour les chaînes de type alphabet international n° 5 «IA5String», l'ordre canonique des cellules est défini à partir du codage ISO 646 de la manière suivante:

(ISO 646 ENCODING)

Le jeu de caractères complet contient exactement 128 caractères. L'extrémité d'un intervalle de valeur «ValueRange» (ou un caractère isolé) peut être désignée en indiquant le symbole graphique correspondant dans une chaîne «cstring» ou une référence de valeur de caractère de contrôle ISO 646 définie au 37.1.1.

39 Définition des types chaînes de caractères à alphabet non restreint

Cet article définit un type dont les valeurs sont celles de n'importe quelle syntaxe abstraite de caractères. Cette syntaxe abstraite peut faire partie de l'ensemble contextuel défini fixé dans une instance de communication, ou être indiquée en référence directement dans chaque instance d'utilisation du type chaîne de caractères à alphabet non restreint.

NOTE 1 – Une syntaxe abstraite de caractères (et une ou plusieurs syntaxes correspondantes de transfert de caractères) peut être définie par tout organisme habilité à attribuer des identificateurs d'objets ASN.1.

NOTE 2 – Les profils établis par une communauté d'intérêt détermineront normalement les syntaxes abstraites de caractères et les syntaxes de transfert de caractères devant être prises en charge pour des instances ou groupes d'instances donnés de chaînes de caractères «CHARACTER STRING». On indiquera généralement les syntaxes admises dans un formulaire PICS (déclaration de conformité d'une instance de protocole). Il est à noter que le groupement des instances aux fins de la spécification de la couche application pourra être réalisé à l'aide de différentes références à des types ASN.1 (chacun étant une référence au type chaîne de caractères «CHARACTER STRING»).

39.1 Le type chaîne de caractères à alphabet non restreint (voir 3.8.69) est désigné au moyen de la notation «UnrestrictedCharacterStringType»:

UnrestrictedCharacterStringType ::= CHARACTER STRING

39.2 Ce type porte l'étiquette numéro 29 de la classe universelle.

39.3 Ce type est composé de valeurs représentant:

a) une chaîne de caractères qui peut être du type ASN.1 chaîne de caractères mais qui ne l'est pas nécessairement;

b) des identificateurs (ensembles ou séparés) indiquant:

1) une classe de valeurs contenant cette chaîne de caractères (syntaxe abstraite de caractères);

2) le codage utilisé (syntaxe de transfert de caractères) permettant de distinguer cette chaîne de caractères des autres valeurs de la même classe.

39.4 Le type chaîne de caractères à alphabet non restreint possède un type associé qui sert uniquement à spécifier la notation de ses valeurs et sous-types.

39.5 En supposant un environnement d'étiquetage automatique, le type associé servant à la définition des valeurs et au sous-typage est le suivant (les commentaires ont force de norme):

```
SEQUENCE {
    identification                  CHOICE {
        syntaxes                        SEQUENCE {
            abstract                        OBJECT IDENTIFIER,
            transfer                        OBJECT IDENTIFIER }
                -- Identificateurs d'objet de la syntaxe abstraite et de la syntaxe de transfert --,

        syntax                          OBJECT IDENTIFIER
                -- Identificateur d'objet unique pour identifier la classe et le codage --,
        presentation-context-id         INTEGER
                -- (Ne s'applique que dans les environnements OSI)
                -- Le contexte de présentation négocié identifie la classe de la valeur et son codage --,

        context-negotiation             SEQUENCE {
            presentation-context-id         INTEGER,
            transfer-syntax                 OBJECT IDENTIFIER }
                -- (Ne s'applique que dans les environnements OSI)
                -- Négociation de contexte en cours pour identifier
                -- la classe de la valeur et son codage --,

        transfer-syntax                 OBJECT IDENTIFIER
                -- La classe de la valeur (une spécification indiquant par exemple qu'il s'agit d'une
                -- valeur d'un type ASN.1) est fixée par le concepteur de l'application (et est donc connue
                -- à la fois de l'expéditeur et du destinataire). Ce cas est prévu avant tout pour prendr en
                -- charge le chiffrement sélectif par champ (ou d'autres transformations de codage) d'un
                -- type ASN.1 --,

        fixed                           NULL
                -- La valeur de donnée est une valeur d'un type ASN.1 fixe
                -- (qui est donc connu à la fois de l'expéditeur et du destinataire) -- },

    data-value-descriptor               ObjectDescriptor  OPTIONAL
                -- Il s'agit de l'identification en langage naturel de la classe de la valeur --,
                                    string-value    OCTET STRING }
( WITH COMPONENTS {
    ... ,
    data-value-descriptor  ABSENT } )
```

NOTE – Le type chaîne de caractères à alphabet non restreint ne permet pas d'inclure une valeur de descripteur de valeur de donnée «data-value-descriptor» en même temps qu'une «identification». Toutefois, la définition du type associé donnée ici souligne les points communs existant entre le type valeur de donnée de présentation enchâssée, le type externe et le type chaîne de caractères à alphabet non restreint.

39.6 La notation de la valeur sera celle du type associé, la valeur de la chaîne d'octets «string-value» représentant un codage utilisant la syntaxe de transfert spécifié par «identification».

UnrestrictedCharacterStringValue ::= SequenceValue -- *valeur du type associé défini au 39.5*

39.7 Un exemple de chaîne de caractères à alphabet non restreint est donné au C.2.8.

40 Notation des types définis dans les articles 41 à 43

40.1 La notation permettant de faire référence à un type défini dans les articles 41 à 43 est la suivante:

UsefulType ::= typereference

où «typereference» est l'une des références définies dans les articles 41 à 43 en notation ASN.1.

40.2 L'étiquette de chaque type «UsefulType» est spécifiée dans les articles 41 à 43.

41 Temps généralisé

41.1 Ce type est désigné par le nom:

GeneralizedTime

41.2 Ce type est composé de valeurs représentant:

a) une date calendaire, telle que celle-ci est définie dans la norme ISO 8601;

b) une heure, à une des précisions définies dans la norme ISO 8601, à l'exception de la valeur horaire 24 qui n'est pas utilisée;

c) le facteur de décalage horaire local, tel qu'il est défini dans la norme ISO 8601.

41.3 Ce type est défini en ASN.1 comme suit:

GeneralizedTime ::= [UNIVERSAL 24] IMPLICIT VisibleString

les valeurs de chaîne visible «VisibleString» étant restreintes à l'une des chaînes de caractères suivantes:

a) une chaîne représentant la date calendaire selon les spécifications de la norme ISO 8601, l'année étant représentée par quatre chiffres, le mois par deux chiffres et le jour par deux chiffres, sans caractères séparateurs, suivie d'une chaîne représentant l'heure selon les spécifications de la norme ISO 8601, sans autre caractère séparateur que la virgule ou le point décimal, (comme prévu dans la norme ISO 8601) et sans lettre finale Z (comme le prévoit la norme ISO 8601);

b) les caractères de l'alinéa a) ci-dessus, suivis d'une lettre Z majuscule;

c) les caractères de l'alinéa a) ci-dessus, suivis d'une chaîne représentant un décalage horaire local selon les spécifications de la norme ISO 8601, sans caractère séparateur.

Dans le cas a), l'heure correspond à l'heure locale. Dans le cas b), le temps représente l'heure universelle coordonnée (heure UTC). Dans le cas c), la partie de la chaîne formée comme dans le cas a) correspond à l'heure locale (t_1), le décalage horaire (t_2) permettant de déterminer l'heure UTC de la manière suivante:

temps universel coordonné (UTC) = $t_1 - t_2$

EXEMPLES:

Cas a)

"19851106210627.3"
heure locale 21 heures 6 minutes 27,3 secondes, le 6 novembre 1985.

Cas b)

"19851106210627.3Z"
temps universel coordonné correspondant à l'heure ci-dessus.

Cas c)

"19851106210627.3-0500"
même heure locale que dans le cas a), avec un retard local de 5 heures sur l'heure UTC.

41.4 L'étiquette de ce type sera conforme à la définition donnée au 41.3.

41.5 La notation des valeurs sera identique à celle de la chaîne visible «VisibleString» définie au 41.3.

42 Temps universel

42.1 Ce type sera désigné par le nom:

UTCTime

42.2 Le type est composé de valeurs représentant:

a) une date calendaire;

b) une heure, à la précision de la minute ou de la seconde;

c) et optionnellement, le décalage de l'heure locale par rapport au temps universel coordonné.

42.3 Le type est défini en ASN.1 de la manière suivante:

UTCTime ::= [UNIVERSAL 23] IMPLICIT VisibleString

les valeurs de la chaîne visible «VisibleString» étant limitées aux chaînes de caractères résultant de la juxtaposition des éléments suivants:

a) six chiffres AAMMJJ, où AA représente les deux chiffres de plus faible poids de l'année grégorienne, MM le mois (01 désignant le mois de janvier) et JJ le quantième du mois (de 01 à 31);

b) l'une des deux chaînes suivantes au choix:

1) quatre chiffres hhmm où hh représente l'heure (00 à 23) et mm les minutes (00 à 59);

2) six chiffres hhmmss où hh et mm sont comme en 1) ci-dessus, et ss représente les secondes (00 à 59);

c) l'une des deux chaînes suivantes au choix:

1) le caractère Z;

2) le caractère + ou le caractère –, suivi de hhmm, hh représentant les heures et mm les minutes.

Les deux formes de l'alinéa b) ci-dessus permettent de spécifier le temps avec différentes précisions.

Dans la forme c) 1), le temps représenté est le temps universel coordonné (UTC). Dans la forme c) 2), le temps t_1 indiqué par les chaînes a) et b) est l'heure locale; le décalage horaire (t_2) indiqué par la chaîne c) 2) permet de déterminer le temps UTC de la manière suivante:

Temps universel coordonné = $t_1 - t_2$

EXEMPLE 1 – Si le temps local est 7 heures du matin du 2 janvier 1982, et que le temps universel coordonné est midi du 2 janvier 1982, la valeur du temps UTCTime est représenté par l'une des chaînes suivantes:

"8201021200Z", ou
"8201020700-0500".

EXEMPLE 2 – Si le temps local est 7 heures du matin du 2 janvier 2001, et que le temps universel coordonné est midi du 2 janvier 2001, la valeur du temps UTCTime est représenté par l'une des chaînes suivantes:

"0101021200Z", ou
"0101020700-0500".

42.4 L'étiquette de ce type sera conforme à la définition donnée au 42.3.

42.5 La notation des valeurs sera identique à celle de la chaîne visible «VisibleString» définie au 42.3.

43 Type descripteur d'objets

43.1 Ce type est désigné par le nom:

ObjectDescriptor

43.2 Ce type est constitué d'un texte en langage naturel décrivant un objet donné. Ce texte ne constitue pas une identification non ambiguë de l'objet, mais des objets différents ne devraient logiquement pas être décrits par un même texte.

NOTE – Il est recommandé que l'autorité affectant un identificateur «OBJECT IDENTIFIER» à un objet lui affecte également une valeur de descripteur d'objet «ObjectDescriptor».

43.3 Ce type est défini en notation ASN.1 de la manière suivante:

ObjectDescriptor ::= [UNIVERSAL 7] IMPLICIT GraphicString

La chaîne graphique «GraphicString» contient le texte décrivant l'objet.

43.4 L'étiquette est celle qui est définie au 43.3.

43.5 Une valeur sera déclarée par la notation de valeur de chaîne graphique «GraphicString» définie au 43.3.

44 Types contraints

44.1 La notation du type contraint «ConstrainedType» permet soit d'imposer une contrainte à un type (parent) pour en restreindre l'ensemble des valeurs à un sous-type donné, soit d'imposer, dans un type ensemble ou un type séquence, des contraintes relationnelles entre composants qui s'appliquent aux valeurs du type parent et aux valeurs d'un autre composant du même type ensemble ou séquence. Elle permet aussi d'associer un identificateur d'exception à une contrainte.

ConstrainedType ::=
Type Constraint |
TypeWithConstraint

Dans la première forme, le type parent est «Type», et la contrainte est spécifiée par «Constraint», elle est définie au 44.5. La seconde forme est définie au 44.4.

44.2 Lorsqu'une contrainte est indiquée à la suite d'une notation de type ensemble-de ou séquence-de, elle s'applique au «Type» intérieur à la notation ensemble-de ou séquence-de, et non au type ensemble-de ou séquence-de lui-même.

NOTE – Dans la notation suivante par exemple, la contrainte de taille «(SIZE(1..64))» s'applique à la chaîne visible «VisibleString» et non au type séquence-de:

NomDesPaysMembres ::= SEQUENCE OF VisibleString (SIZE(1..64))

44.3 Lorsqu'une contrainte est indiquée à la suite d'une notation de type étiqueté «TaggedType», l'interprétation de la notation globale est la même, que l'on considère le type étiqueté ou le type non étiqueté comme le type parent.

44.4 Par suite de l'interprétation donnée au 44.2, une notation spéciale est prévue pour permettre d'appliquer une contrainte à un type ensemble-de ou séquence-de. Il s'agit de la contrainte «TypeWithConstraint»:

TypeWithConstraint ::=
SET Constraint OF Type |
SET SizeConstraint OF Type |
SEQUENCE Constraint OF Type |
SEQUENCE SizeConstraint OF Type

Le type parent est, dans les deux premières formes, «SET OF Type», et dans les deux dernières, «SEQUENCE OF Type». La contrainte est de type général «Constraint» (voir 44.5) dans les première et troisième formes, et porte sur la taille «SizeConstraint» dans les deuxième et quatrième formes (voir 48.5).

NOTE – Bien que la forme «Constraint» recouvre les possibilités de «SizeConstraint», cette dernière forme, non parenthésée, a été prévue pour des considérations de compatibilité amont avec les dispositions de la Rec. X.208 du CCITT | ISO/CEI 8824.

44.5 Une contrainte est spécifiée par la notation «Constraint»:

Constraint ::= "(" ConstraintSpec ExceptionSpec ")"

ConstraintSpec ::=
SubtypeConstraint |
GeneralConstraint

La spécification d'exception «ExceptionSpec» est définie à l'article 45. Sauf si elle est utilisée conjointement à un «marqueur d'extension» (voir l'article 47), elle n'apparaîtra que lorsqu'une occurrence de référence muette «DummyReference» (voir le paragraphe 8.3 de la Rec. UIT-T X.683 | ISO/CEI 8824-4) ou une contrainte définie par l'utilisateur «UserDefinedConstraint» (voir l'article 9 de la Rec. UIT-T X.682 | ISO/CEI 8824-3) figurera dans la spécification de contrainte «ConstraintSpec».

44.6 La notation «SubtypeConstraint» est la notation d'ordre général «ElementSetSpec» (voir l'article 46):

SubtypeConstraint ::= ElementSetSpecs

Dans ce contexte, les éléments sont des valeurs du type parent (le gouvernant de l'ensemble est le type parent). L'ensemble comprendra au moins un élément.

45 Identificateur d'exception

45.1 Dans une spécification ASN.1 complexe, il existe plusieurs situations dans lesquelles il est admis que les décodeurs auront à traiter des éléments incomplètement spécifiés. Ces situations résultent notamment de l'utilisation d'une contrainte définie au moyen d'un paramètre de la syntaxe abstraite (voir l'article 10 de la Rec. UIT-T X.683 | ISO/CEI 8824-4).

45.2 Dans de telles situations, les concepteurs d'applications doivent déterminer les mesures à prendre lorsqu'une contrainte donnée dépendant de l'application est transgressée. L'identificateur d'exception est prévu comme un moyen non ambigu de faire référence à des éléments de spécification ASN.1 indiquant les actions à exécuter dans ces cas. L'identificateur est constitué du caractère «!», suivi d'un type ASN.1 optionnel et d'une valeur de ce type. Si le type optionnel n'est pas mentionné, la valeur est supposée être du type entier «INTEGER».

45.3 Si un identificateur d'exception est indiqué, il signifie qu'il existe un texte dans le corps de la norme disant comment traiter la transgression de contrainte associé au signe «!». En l'absence d'un tel identificateur et si une transgression de contrainte a lieu, les concepteurs devront soit identifier le texte décrivant les mesures à prendre, soit décider de ces mesures en fonction de l'application.

45.4 La notation d'exception «ExceptionSpec» est définie comme suit:

```
ExceptionSpec ::= "!"  ExceptionIdentification | empty

ExceptionIdentification ::= SignedNumber  |
                            DefinedValue   |
                            Type ":" Value
```

Les deux premières formes correspondent à des identificateurs d'exception de type entier. La troisième correspond à un identificateur d'exception «Value» de type arbitraire «Type».

45.5 Lorsqu'un type est soumis à des contraintes multiples dont plus d'une possède un identificateur d'exception, l'identificateur d'exception de la contrainte la plus externe sera considéré comme l'identificateur d'exception pour ce type.

45.6 L'identificateur d'exception est ignoré et n'est pas hérité par le type qui est soumis à une contrainte résultant de l'arithmétique sur des ensembles, lorsqu'un marqueur d'exception est présent pour des types utilisés dans une telle arithmétique.

46 Spécification d'un ensemble d'éléments

46.1 Dans certaines notations, on peut spécifier un ensemble d'éléments appartenant à une classe déterminée (la classe gouvernante). A cette fin, on utilise la notation «ElementSetSpec»:

```
ElementSetSpecs ::=
        RootElementSetSpec |
        RootElementSetSpec  ","  "..." |
        "..."  ","  AdditionalElementSetSpec |
        RootElementSetSpec  ","  "..."  ","  AdditionalElementSetSpec

RootElementSetSpec ::= ElementSetSpec
```

AdditionalElementSetSpec ::= ElementSetSpec

ElementSetSpec ::= Unions |
ALL Exclusions

Unions ::= Intersections |
UElems UnionMark Intersections

UElems ::= Unions

Intersections ::= IntersectionElements |
IElems IntersectionMark IntersectionElements

IElems ::= Intersections

IntersectionElements ::= Elements | Elems Exclusions

Elems ::= Elements

Exclusions ::= EXCEPT Elements

UnionMark ::= "|" | UNION

IntersectionMark ::= "^" | INTERSECTION

NOTE 1 – Le symbole accent circonflexe «^» et le mot «INTERSECTION» sont synonymes. Le symbole barre «|» et le mot «UNION» sont synonymes. Dans un souci de cohérence stylistique, il est recommandé d'utiliser la notation soit par symboles soit par mots dans l'ensemble de la spécification. La déclaration «EXCEPT» peut être utilisée avec l'une ou l'autre notation.

NOTE 2 – L'ordre de priorité décroissant des opérateurs est: «EXCEPT», «^», «|». A noter que la déclaration «ALL EXCEPT» est spécifiée de telle manière qu'elle ne puisse être insérée au milieu d'autres contraintes sans que l'expression «ALL EXCEPT xxx» ne figure entre parenthèses.

NOTE 3 – A chaque occurrence «Elements» peut correspondre soit à une contrainte non parenthésée [par exemple INTEGER (1..4)] soit à une contrainte parenthésée de sous-type [par exemple INTEGER ((1..4) | 9))].

NOTE 4 – A noter que deux opérateurs «EXCEPT» seront toujours séparés par un signe «|», «^», «(« ou «)». Ainsi, l'expression «(A EXCEPT B EXCEPT C)» est incorrecte; elle devrait être remplacée par «((A EXCEPT B) EXCEPT C)» ou par «(A EXCEPT (B EXCEPT C))».

NOTE 5 – A noter que l'expression «((A EXCEPT B) EXCEPT C)» est équivalente à «(A EXCEPT (B | C))».

NOTE 6 – Les éléments auxquels fait référence la spécification «ElementSetSpecs» sont l'union des éléments auxquels font référence les spécifications «RootElementSetSpec» et «AdditionalElementSetSpec».

46.2 Les éléments composant l'ensemble sont:

a) si la première forme de «ElementSetSpec» est utilisée, les éléments spécifiés par la notation «Union» [voir alinéa b)], sinon les éléments du type gouvernant à l'exception des éléments spécifiés par la notation «Elements» de l'expression «Exclusions»;

b) si la première forme de «Unions» est utilisée, les éléments spécifiés par la notation «Intersections» [voir alinéa c)], sinon les éléments figurant au moins une fois dans «Uelems» ou dans «Intersections»;

c) si la première forme de «Intersections» est utilisée, les éléments spécifiés dans «IntersectionElements» [voir alinéa d)], sinon les éléments spécifiés à la fois dans «Ielems» et dans «IntersectionElements»;

d) si la première forme de «IntersectionElements» est utilisée, les éléments spécifiés dans «Elements», sinon les éléments spécifiés dans «Elems» à l'exception de ceux spécifiés dans «Exclusions».

46.3 La notation «Elements» est définie comme suit:

Elements ::=
SubtypeElements |
ObjectSetElements |
"(" ElementSetSpec ")"

Les éléments spécifiés par cette notation sont:

a) comme décrit dans l'article 48 ci-dessous si l'expression utilise la forme «SubtypeElements». Cette notation ne sera utilisée que lorsque le gouvernant est un type, et que le type concerné doit servir à imposer des contraintes supplémentaires aux possibilités de la notation. Dans un tel contexte, le gouvernant est considéré comme le type parent;

b) ceux décrits au paragraphe 12.6 de la Rec. UIT-T X.681 | ISO/CEI 8824-2 si l'expression utilise la forme «ObjectSetElements». Cette notation ne sera utilisée que lorsque le gouvernant est une classe d'objets informationnels;

c) les éléments spécifiés par la notation «ElementSetSpec» si la troisième forme est utilisée.

47 Marqueur d'extension

NOTE – Comme c'est le cas d'une manière générale pour la notation de contrainte, le marqueur d'extension n'a aucun effet sur certaines règles de codage de la notation ASN.1, telles que les règles de codage de base, mais en a sur certaines autres, telles que les règles de codage compactes.

47.1 Le marqueur d'extension, représenté par des points de suspension, est une indication que des additions d'extension sont attendues. Il ne fait aucune déclaration au sujet de la manière de traiter de telles additions, à part le fait qu'elles ne seront pas traitées comme des erreurs lors du processus de décodage.

47.2 L'utilisation conjointe du marqueur d'extension et d'un identificateur d'exception est une indication que des additions d'extension sont attendues, qu'elles ne doivent donc pas être traitées comme des erreurs lors du processus de décodage et que les normes de l'application prescrivent d'effectuer des actions spécifiques si une contrainte est transgressée. L'utilisation de cette notation est recommandée pour des situations dans lesquelles un stockage avec retransmission ou toute autre forme de relais est utilisé, de manière à indiquer que toute addition d'extension non reconnue doit être renvoyée à l'application pour un nouveau codage et un relais éventuel.

47.3 L'arithmétique sur des ensembles utilisée éventuellement dans la notation «ElementSetSpecs» sera effectuée sans prendre en considération la présence du marqueur d'extension.

NOTE – En d'autres termes, la présence d'un marqueur d'extension n'a aucun effet sur l'arithmétique sur des ensembles.

47.4 Si un type défini avec une contrainte extensible est référencé dans un sous-type «ContainedSubtype», le type nouvellement défini n'hérite pas du marqueur d'extension et de toute addition d'extension qu'il contient. Si le type nouvellement défini est destiné à être extensible, un marqueur d'extension devra alors être ajouté d'une manière explicite à ses spécifications «ElementSetSpecs». A titre d'exemple:

```
A ::= INTEGER (0..10, …, 12)          -- A est extensible.

B ::= INTEGER (A)                     -- B n'est pas extensible et limité à l'intervalle 0-10.

C ::= INTEGER (A, …)                  -- C est extensible et limité à l'intervalle 0-10.
```

47.5 Si un type défini avec une contrainte extensible est soumis en plus à une contrainte dont la spécification «ElementSetSpecs» ne contient pas de marqueur d'extension, le type résultant est un type dont les contraintes ne sont pas extensibles et qui n'hérite d'aucune des extensions pouvant être présentes dans le type parent. A titre d'exemple:

```
A ::= INTEGER (0..10, ...)            -- A est extensible.

B ::= A (2..5)                        -- B n'est pas extensible.

C ::= A                               -- C est extensible.
```

47.6 Les composants d'un type ensemble, séquence et choix qui sont soumis à la contrainte d'être absents ne pourront pas être présents, que le type ensemble, séquence ou choix soit extensible ou non.

NOTE – Des contraintes de type interne n'ont aucun effet sur l'extensibilité.

Par exemple:

```
A ::= SEQUENCE  {
      a     INTEGER
      b     BOOLEAN OPTIONAL,
      ...
}

B ::= A (WITH COMPONENTS {b ABSENT})    -- B est extensible, mais «b» ne figurera
                                        -- dans aucune de ses valeurs.
```

47.7 Les transformations suivantes doivent être appliquées d'une manière conceptuelle avant d'effectuer la vérification d'unicité des étiquettes lorsque la présente Recommandation | Norme internationale prescrit des étiquettes distinctes (voir 24.5, 24.6, 26.3 et 28.2).

47.7.1 Un nouveau composant ou une nouvelle forme (dans le cas d'un choix), appelé alors élément ajouté conceptuellement (voir 47.7.2) est ajouté au point d'extension dans l'un des cas suivants:

a) il n'existe pas de marqueurs d'extension, mais l'extensibilité est implicitement prévue dans l'entête du module, un marqueur d'extension est ajouté par la suite et le nouvel élément est inséré comme premier ajout après ce marqueur d'extension;

b) il existe un seul marqueur d'extension dans une notation «CHOICE», «SEQUENCE» ou «SET» et le nouvel élément est ajouté à la fin de la notation en question immédiatement avant l'accolade fermante;

c) il existe deux marqueurs d'extension dans une notation «CHOICE», «SEQUENCE» ou «SET» et le nouvel élément est ajouté immédiatement avant le deuxième marqueur d'extension.

47.7.2 L'élément ajouté au niveau conceptuel existe uniquement à des fins de vérification de légalité par application des règles qui prescrivent des étiquettes distinctes (voir 24.5, 24.6, 26.3 et 28.2). Il est ajouté au niveau conceptuel après l'application éventuelle de l'étiquetage automatique et l'expansion de la notation «COMPONENTS OF».

47.7.3 L'élément ajouté conceptuellement est défini comme ayant une étiquette distincte de l'étiquette de tout type ASN.1 normal, mais qui correspond à l'étiquette de tous les éléments conceptuellement ajoutés ainsi qu'à l'étiquette indéterminée du type ouvert comme le spécifie la Note 2 du paragraphe 14.2 de la Rec. UIT-T X.681 | ISO/CEI 8824-2.

NOTE – En ce qui concerne les éléments conceptuellement ajoutés et le type ouvert, les règles relatives à l'unicité ainsi que les règles imposant des étiquettes distinctes (voir 24.5 à 24.6, 26.3 et 28.2) sont nécessaires et suffisantes pour garantir:

a) que tout ajout d'extension inconnu puisse être attribué sans ambiguïté à un seul point d'insertion lors du décodage d'un élément codé selon les règles de codage de base;

b) que les additions d'extension inconnue ne pourront jamais être confondues avec les éléments optionnels.

Dans les codages compacts, les règles ci-dessous sont suffisantes mais ne sont pas nécessaires pour garantir ces propriétés. Elles sont néanmoins imposées comme règles ASN.1 pour garantir l'indépendance de la notation par rapport aux règles de codage.

47.7.4 La spécification utilise d'une manière illégale la notation d'extensibilité si les éléments ajoutés sur le plan conceptuel transgressent les règles d'unicité d'étiquette.

NOTE - Les règles précédentes ont pour but de fournir des contraintes précises au sujet de l'utilisation des points d'insertion (en particulier pour ceux qui ne sont pas situés à la fin de types «SEQUENCE», «SET» ou «CHOICE»). Ces limitations ont été conçues afin de garantir qu'il est possible d'affecter un point d'insertion spécifique à un élément inconnu reçu par un système en version 1 lors de l'utilisation des règles de codage BER, DER et CER. Ce point est important si le traitement d'exception de tels éléments ajoutés n'est pas le même pour différents points d'insertion.

47.8 Exemples

47.8.1 Exemple 1

```
A ::= SET {
    a    A,
    b    CHOICE {
         c    C,
         ... ,
         ... ,
         d    D
    }
}
```

est légal, car il n'existe pas d'ambiguïté sur le fait que toute information ajoutée doit faire partie de l'élément «b».

47.8.2 Exemple 2

```
A ::= SET {
    a    A,
    b    CHOICE {
         c         C,
              ... ,
              ... ,
         d         D
    },
    ... ,
    e    E
}
```

est illégal, car les informations ajoutées peuvent faire partie de l'élément «b» ou peuvent appartenir au niveau externe de l'élément «A»; un système en version 1 ne peut pas lever l'ambiguïté.

47.8.3 Exemple 3

```
A ::= SET {
        a    A,
        b    CHOICE {
             c        C,
             ...
        } ,
        d    CHOICE {
             e        E,
             ...
        }
}
```

est également illégal, car les informations ajoutées peuvent appartenir à l'élément «b» ou «d».

47.8.4 Il est possible de construire des exemples plus complexes, avec des choix extensibles situés au sein d'autres choix extensibles ou au sein d'éléments d'une séquence avec des marqueurs «OPTIONAL» ou «DEFAULT»; les règles précédentes sont toutefois nécessaires et suffisantes pour garantir qu'un élément qui n'est pas présent dans la version 1 peut être attribué d'une manière non ambiguë à un point d'insertion et un seul par un système en version 1.

48 Eléments de sous-type

48.1 Généralités

Plusieurs formes sont prévues pour noter les éléments de sous-type «SubtypeElements». Elles sont identifiées ci-dessous, leur syntaxe et leur sémantique étant définies dans les articles suivants. Le Tableau 6 récapitule pour chaque type parent les notations qui peuvent lui être appliquées.

```
SubtypeElements ::=
    SingleValue                |
    ContainedSubtype           |
    ValueRange                 |
    PermittedAlphabet          |
    SizeConstraint             |
    TypeConstraint             |
    InnerTypeConstraints
```

48.2 Valeur unique

48.2.1 La notation de valeur unique «SingleValue» est la suivante:

```
SingleValue ::= Value
```

où «Value» est la notation de la valeur appartenant au type parent.

48.2.2 La notation «SingleValue» détermine la valeur unique du type parent spécifiée par «Value».

48.3 Contenance de type

48.3.1 La notation de sous-type contenu «ContainedSubtype» est la suivante:

```
ContainedSubtype ::= Includes Type
Includes ::= INCLUDES | empty
```

La forme vide «empty» de la production «Includes» ne sera pas utilisée si le «Type» du sous-type contenu «ContainedSubtype» est la notation du type néant.

48.3.2 La notation «ContainedSubtype» spécifie toutes les valeurs du type parent qui résultent de l'intersection du type parent et de «Type», qui doit lui-même dériver du même type prédéfini que le type parent.

Tableau 6 – Applicabilité de la notation de sous-typage selon le type parent

Type	Valeur unique (Single Value)	Contenance de type (Contained Subtype)	Intervalle (Value Range)	Contrainte de taille (Size Constraint)	Alphabet permis (Permitted Alphabet)	Contrainte de type (Type Constraint)	Sous-typage interne (Inner Subtyping)
Chaîne binaire (Bit String)	Oui	Oui	Non	Oui	Non	Non	Non
Booléen (Boolean)	Oui	Oui	Non	Non	Non	Non	Non
Choix (Choice)	Oui	Oui	Non	Non	Non	Non	Oui
Valeur de donnée de présentation enchâssée (embedded-pdv)	Oui	Non	Non	Non	Non	Non	Oui
Enuméré (Enumerated)	Oui	Oui	Non	Non	Non	Non	Non
Externe (External)	Oui	Non	Non	Non	Non	Non	Oui
Instance-de (Instance-of)	Oui	Oui	Non	Non	Non	Non	Oui
Entier (Integer)	Oui	Oui	Oui	Non	Non	Non	Non
Néant (Null)	Oui	Oui	Non	Non	Non	Non	Non
Type champ de classe d'objets (Object class field type)	Oui	Oui	Non	Non	Non	Non	Non
Identificateur d'objet (Object Identifier)	Oui	Oui	Non	Non	Non	Non	Non
Chaîne d'octets (Octet String)	Oui	Oui	Non	Oui	Non	Non	Non
Type ouvert (open type)	Non	Non	Non	Non	Non	Oui	Non
Réel (Real)	Oui	Oui	Oui	Non	Non	Non	Oui
Types de chaînes de caractères à alphabet restreint (Restricted Character String Types)	Oui	Oui	Oui[a)]	Oui	Oui	Non	Non
Séquence (Sequence)	Oui	Oui	Non	Non	Non	Non	Oui
Séquence-de (Sequence-of)	Oui	Oui	Non	Oui	Non	Non	Oui
Ensemble (Set)	Oui	Oui	Non	Non	Non	Non	Oui
Ensemble-de (Set-of)	Oui	Oui	Non	Oui	Non	Non	Oui
Types de chaînes de caractères à alphabet non restreint (Unrestricted Character String Type)	Oui	Non	Non	Oui	Non	Non	Oui

a) Autorisé seulement avec l'alphabet permis «PermittedAlphabet» des types chaînes multilingues «BMPString», alphabet international n° 5 «IA5String», numériques «NumericString, imprimables «PrintableString», visibles «VisibleString» et universelles «UniversalString».

48.4 Intervalle de valeurs

48.4.1 La notation d'intervalle de valeurs «ValueRange» est la suivante:

ValueRange ::= LowerEndpoint ".." UpperEndpoint

48.4.2 La notation «ValueRange» spécifie toutes les valeurs d'un intervalle désigné en spécifiant les valeurs de ses extrémités. Cette notation ne peut être utilisée qu'avec le type entier, certains types de chaînes de caractères à alphabet restreint (chaînes de l'alphabet international n° 5 «IA5String», numériques «NumericString», imprimables «PrintableString», visibles «VisibleString», multilingues «BMPString» et universelles «UniversalString» seulement) et le type réel.

NOTE – Pour les besoins du sous-typage, «PLUS-INFINITY» est supérieur à toute valeur réelle et «MINUS-INFINITY» est inférieur à toute valeur réelle.

48.4.3 Chaque extrémité de l'intervalle peut être soit autorisée (auquel cas elle appartient à l'intervalle), soit non autorisée (auquel cas elle n'appartient pas à l'intervalle). Quand l'extrémité n'appartient pas à l'intervalle, la définition inclut un symbole inférieur-à «<»:

LowerEndpoint ::= LowerEndValue | LowerEndValue "<"

UpperEndpoint ::= UpperEndValue | "<" UpperEndValue

48.4.4 Une extrémité peut également ne pas être spécifiée, auquel cas l'intervalle s'étend dans ce sens aussi loin que le type parent l'autorise:

LowerEndValue ::= Value | MIN

UpperEndValue ::= Value | MAX

48.5 Contrainte de taille

48.5.1 La notation de contrainte de taille «SizeConstraint» est la suivante:

SizeConstraint ::= SIZE Constraint

48.5.2 Une contrainte de taille «SizeConstraint» ne peut s'appliquer qu'aux types chaîne binaire, chaîne d'octets, chaîne de caractères, ensemble-de ou séquence-de, ou aux types obtenus à partir de ces derniers par étiquetage.

48.5.3 La contrainte «Constraint» spécifie les valeurs entières autorisées pour la longueur des valeurs spécifiées; elle a la forme de n'importe quelle contrainte pouvant être appliquée au type parent suivant:

INTEGER (0 .. MAX)

La contrainte «Constraint» sera exprimée au moyen de la forme d'une contrainte de sous-type «SubtypeConstraint» dans la notation de spécification de contrainte «ConstraintSpec».

48.5.4 L'unité de mesure dépend du type parent de la manière suivante:

Type	*Unité de mesure*
Chaîne binaire	Bit
Chaîne d'octets	Octet
Chaîne de caractères	Caractère
Ensemble-de	Valeur composant l'ensemble
Séquence-de	Valeur composant la séquence

NOTE – On distinguera clairement le décompte des caractères utilisé dans ce paragraphe pour déterminer la taille des chaînes de caractères, d'un quelconque décompte d'octets. Le décompte des caractères sera interprété selon la définition de la collection de caractères utilisée dans le type, et plus particulièrement par rapport aux normes, grilles ou numéros d'enregistrement dans un registre pouvant apparaître dans une telle définition.

48.6 Contrainte de type

48.6.1 La notation de contrainte de type «TypeConstraint» est la suivante:

TypeConstraint ::= Type

48.6.2 Cette notation ne s'applique qu'à la notation d'un type ouvert et restreint ce type aux valeurs du type «Type».

48.7 Alphabet permis

48.7.1 La notation d'alphabet permis «PermittedAlphabet» est la suivante:

PermittedAlphabet ::= FROM Constraint

48.7.2 La notation «PermittedAlphabet» spécifie toutes les valeurs qui peuvent être obtenues en utilisant un sous-alphabet de la chaîne parentale. Cette notation ne s'applique qu'aux types chaînes de caractères à alphabet restreint.

48.7.3 La contrainte «Constraint» est n'importe quelle contrainte pouvant s'appliquer au type parent (voir le Tableau 6), sauf qu'elle doit utiliser la forme de contrainte de sous-type «SubtypeConstraint» de la production «ConstraintSpec». Le sous-alphabet inclut tous les caractères qui figurent dans une ou plusieurs valeurs (chaînes de caractères) du type parent et qui sont autorisés par la contrainte «Constraint» et seulement ces caractères.

48.8 Sous-typage interne

48.8.1 La notation de contrainte de type interne «InnerTypeConstraints» est la suivante:

InnerTypeConstraints ::=
WITH COMPONENT SingleTypeConstraint |
WITH COMPONENTS MultipleTypeConstraints

48.8.2 La notation «InnerTypeConstraints» spécifie les valeurs qui satisfont à un ensemble de contraintes portant sur la présence et la valeur des composants du type parent. Une valeur du type parent ne satisfait la contrainte que si elle satisfait à toutes les contraintes explicites ou implicites (voir 48.8.6). Cette notation peut être appliquée aux types ensemble-de, séquence-de, ensemble, séquence et choix, ainsi qu'aux types qui en dérivent par étiquetage.

48.8.3 Une contrainte ayant la forme d'une spécification de valeur de sous-type est prévue pour les types (ensemble-de, séquence-de) qui sont définis en termes d'un seul autre type (interne). La notation de cette contrainte de type unique «SingleTypeConstraint» est la suivante:

SingleTypeConstraint ::= Constraint

La contrainte «Constraint» définit un sous-type du type unique (interne). Une valeur du type parent satisfait la contrainte si et seulement si chaque valeur interne appartient au sous-type obtenu en appliquant la contrainte au type interne.

48.8.4 Lorsqu'un type (choix, ensemble, séquence) est défini en termes de plusieurs autres types (internes), plusieurs contraintes peuvent s'appliquer aux types internes qui le composent. La notation de ces contraintes à types multiples «MultipleTypeConstraints» est la suivante:

MultipleTypeConstraints ::= FullSpecification | PartialSpecification

FullSpecification ::= "{" TypeConstraints "}"

PartialSpecification ::= "{" "..." "," TypeConstraints "}"

TypeConstraints ::=
NamedConstraint |
NamedConstraint "," TypeConstraints

NamedConstraint ::=
identifier ComponentConstraint

48.8.5 Les contraintes de types «TypeConstraints» contiennent une liste de contraintes s'appliquant aux types composants du type parent. Pour un type séquence, les contraintes doivent figurer dans l'ordre des composants de la séquence. Le type interne auquel les contraintes s'appliquent est repéré par son identificateur. Il y aura au plus une contrainte nommée «NamedConstraint» par composant.

48.8.6 Les contraintes de types multiples «MultipleTypeConstraints» comprennent soit une spécification complète «FullSpecification», soit une spécification partielle «PartialSpecification». Lorsqu'une spécification complète «FullSpecification» est fournie, il existe une contrainte de présence implicite selon laquelle la déclaration «ABSENT» est apposée à tous les types internes qui peuvent recevoir une telle contrainte (voir 48.8.9) et qui ne sont pas explicitement énumérés. Lorsqu'une spécification partielle «PartialSpecification» est fournie il n'y a pas de contrainte implicite, et tout type interne peut être omis de la liste.

48.8.7 Les contraintes s'appliquant à un type interne donné peuvent concerner sa présence (exprimée à partir du type parent), ses valeurs, ou les deux. La notation de la contrainte de composant «ComponentConstraint» est la suivante:

ComponentConstraint ::= ValueConstraint PresenceConstraint

48.8.8 Une contrainte sur la valeur d'un type interne est définie par la notation «ValueConstraint»:

ValueConstraint ::= Constraint | empty

La contrainte est satisfaite par une valeur du type parent si et seulement si la valeur interne appartient au sous-type spécifié par la contrainte «Constraint» appliquée au type interne.

48.8.9 Une contrainte portant sur la présence d'un type interne est exprimée par la notation «PresenceConstraint»:

PresenceConstraint ::= PRESENT | ABSENT | OPTIONAL | empty

La signification de ces formes et les situations dans lesquelles elles sont autorisées sont définies du 48.8.9.1 au 48.8.9.3.

48.8.9.1 Si le type parent est une séquence ou un ensemble, un type composant déclaré «OPTIONAL» peut être contraint par la déclaration «PRESENT», (auquel cas la contrainte est satisfaite si et seulement si la valeur du composant correspondant figure), par la déclaration «ABSENT» (auquel cas la contrainte est satisfaite si et seulement si la valeur du composant correspondant ne figure pas) ou par la déclaration «OPTIONAL» (auquel cas aucune contrainte n'est imposée quant à la présence de la valeur du composant correspondant).

48.8.9.2 Si le type parent est un choix, un type composant peut être contraint par la déclaration «ABSENT» (auquel cas la contrainte est satisfaite si et seulement si le type composant correspondant n'est pas utilisé dans la valeur), ou par la déclaration «PRESENT» (auquel cas la contrainte est satisfaite si et seulement si le type composant correspondant figure dans la valeur); il y aura au plus un mot-clé «PRESENT» dans une notation de contraintes de types multiples «MultipleTypeConstraints».

NOTE – Le paragraphe C.4.6 fournit un exemple illustrant ce texte.

48.8.9.3 La signification d'une contrainte de présence «PresenceConstraint» vide dépend de la nature (complète ou partielle) de sa définition:

a) dans une spécification complète «FullSpecification», elle est équivalente à la contrainte de présence «PRESENT» pour un composant d'ensemble ou de séquence déclarée «OPTIONAL» et n'impose autrement aucune autre contrainte;

b) dans une spécification partielle «PartialSpecification», aucune contrainte n'est imposée.

Annexe A

Utilisation de la notation ASN.1-88/90

(Cette annexe fait partie intégrante de la présente Recommandation | Norme internationale)

A.1 Maintenance

Le terme de **notation ASN.1-88/90** désigne la notation ASN.1 spécifiée par la Rec. X.208 du CCITT | ISO/CEI 8824. Le terme de **notation ASN.1 actuelle** désigne la notation spécifiée par la présente Recommandation | Norme internationale.

Au moment de la publication de la présente Recommandation | Norme internationale, la maintenance de la Rec. X.208 du CCITT | ISO/CEI 8824 était encore assurée. Cette maintenance, dont l'existence dépend d'une résolution annuelle du JTC 1/SC21 de l'ISO/CEI, ne pourra pas se poursuivre indéfiniment. Elle est assurée pour donner aux utilisateurs de la notation ASN.1 le temps de remplacer certains éléments (et en particulier la déclaration «ANY» et l'utilisation de la notation de macro) de la notation ASN.1-88/90 par la notation ASN.1 actuelle (un tel remplacement peut être réalisé sans modification des messages binaires transmis).

A.2 Panachage de la notation ASN.1-88/90 et de la notation ASN.1 actuelle

La notation ASN.1-88/90 et la notation ASN.1 actuelle spécifient toutes deux une structure syntaxique de haut niveau qui est le module ASN.1. L'utilisateur de la notation ASN.1 écrit une série de modules ASN.1 et peut importer des définitions d'autres modules ASN.1.

Dans chaque module, la notation utilisée doit se conformer entièrement soit à la notation ASN.1-88/90 soit à la notation ASN.1 actuelle; une spécification d'utilisateur indiquera clairement la notation adoptée (en se référant à la Recommandation | Norme internationale appropriée) au niveau de chaque module dont le texte est donné dans la Spécification.

A noter qu'il peut arriver qu'un utilisateur souhaite modifier une partie d'un module et utiliser à cette fin la nouvelle notation, tout en laissant les autres parties du module dans l'ancienne notation. Ceci est parfaitement possible mais ne peut être réalisé qu'en éclatant le module en deux.

Lorsqu'un module est conforme à la notation ASN.1-88/90, il peut importer des références de types et de valeurs depuis un module écrit en notation ASN.1 actuelle. Ces types et valeurs doivent être associés à des types pouvant être définis au moyen de la seule notation ASN.1-88/90. Par exemple, un module écrit en notation ASN.1-88/90 ne peut importer une valeur du type «UniversalString», car celui-ci est défini dans la notation actuelle et non dans la notation ASN.1-88/90; en revanche, il peut importer des valeurs de type «INTEGER» ou «IA5String» par exemple.

Lorsqu'un module est conforme à la notation ASN.1 actuelle, il peut importer des références de types et de valeurs depuis un module écrit en notation ASN.1-88/90. Par contre, il n'importera pas de notations de macro ASN.1. La notation de valeur d'un type importé ne sera utilisée dans le module importateur que si les identificateurs des valeurs des types ensemble, séquence et choix utilisés dans la notation des valeurs sont présents, et qu'il n'existe pas dans cette notation de valeur une valeur déclarée avec le type «ANY». Une contrainte de type interne «inner type constraint» ne sera pas appliquée à un type importé si le composant sur lequel porte la contrainte n'a pas d'identificateur.

A.3 Migration vers la notation ASN.1 actuelle

Lorsqu'un module (rédigé à l'origine conformément à la notation ASN.1-88/90) est modifié pour le rendre conforme à la notation ASN.1 actuelle, il faut observer les points suivants:

a) Chacun des composants d'un type ensemble, séquence ou choix recevra un identificateur non ambigu dans le cadre de ce type, et ces identificateurs devront être incorporés dans la notation de valeur.

NOTE 1 – La notation de valeur d'un type choix contient le signe deux-points «:».

b) Tous les cas d'utilisation des déclarations «ANY» et «ANY DEFINED BY» seront repris avec une définition de classe d'objets informationnels appropriée, les déclarations «ANY» et «ANY DEFINED BY» (ainsi que le composant concerné) étant remplacées par des références appropriées à des champs de cette classe d'objets. Dans la plupart des cas, la spécification peut être grandement améliorée par l'insertion judicieuse de contraintes tabulaires et de contraintes relationnelles entre composants. Dans beaucoup de cas, la spécification peut être encore améliorée si les contraintes tabulaires ou les contraintes relationnelles entre composants sont données sous forme de paramètres du type.

c) Toutes les définitions de macro seront remplacées par la définition d'une classe d'objets informationnels, d'un type paramétré ou d'une valeur paramétrée. Si la déclaration «WITH SYNTAX» est judicieusement utilisée dans la définition d'une classe d'objets informationnels, la notation utilisée pour définir un objet de cette classe peut être rendue très semblable aux anciennes notations de macro.

d) Toutes les instances d'utilisation de macro seront remplacées soit par des définitions équivalentes d'objets informationnels, soit par des renvois à des types de champs de classe d'objets, des types paramétrés ou des valeurs paramétrées. Dans la plupart des cas, la spécification d'objets informationnels peut être grandement améliorée en regroupant ces définitions dans des ensembles d'objets informationnels, et en donnant des directives claires quant à l'obligation de prendre en charge tous les objets informationnels de l'ensemble, et à la possibilité pour les applications réceptrices d'apporter des extensions dépendant de l'application à cet ensemble d'objets informationnels, et, dans l'affirmative, la manière de traiter la réception de valeurs «inconnues». Il peut également être souhaitable d'envisager la possibilité d'élargir la classe d'objets informationnels dans une version ultérieure de la spécification d'utilisateur, et de donner en conséquence des directives aux utilisateurs sur la manière de traiter de telles extensions.

e) Toutes les occurrences de la déclaration «EXTERNAL» doivent être examinées avec soin; bien qu'une telle notation reste licite en notation ASN.1 actuelle, une spécification d'utilisateur peut sans doute être améliorée en procédant de la manière suivante:

   1) Envisager l'utilisation d'une notation «INSTANCE OF» (de préférence avec une contrainte tabulaire pouvant prendre la forme d'un paramètre du type, d'une manière similaire à la solution proposée ci-dessus pour les déclarations «ANY» et «ANY DEFINED BY») à la place de la notation «EXTERNAL»; dans de nombreux cas, cela ne modifiera pas le contenu des messages binaires transmis.

   2) Si la notation «EXTERNAL» est conservée, le recours au sous-typage interne du type associé (voir 33.5) peut contribuer à préciser la spécification quant à la possibilité ou non d'utiliser des identificateurs de contexte de présentation. Dans cette situation s'appliquent également les directives données précédemment (voir l'article 33) à propos des valeurs de type «EXTERNAL» devant être prises en charge, et du traitement par les applications des valeurs reçues non prises en charge.

   3) Envisager le passage à la notation:

      CHOICE {external EXTERNAL, embedded-pdv EMBEDDED PDV}

      (avec encore une fois un sous-typage interne s'il y a lieu) pour permettre une migration progressive des applications homologues réparties vers la nouvelle notation. Une telle transformation peut affecter les messages binaires transmis, et devrait donc normalement s'effectuer dans le cadre d'un changement de version de protocole. L'utilisation de la déclaration de valeur de donnée de présentation enchâssée «EMBEDDED PDV» (en particulier pour les nouvelles spécifications) devrait normalement conférer plus de souplesse, comme le laisse voir la comparaison des types associés; de plus, le type valeur de donnée de présentation enchâssée «EMBEDDED PDV» est codé plus efficacement que le type «EXTERNAL» par toutes les règles de codage spécifiées dans la Rec. UIT-T X.690 | ISO/CEI 8825-1.

f) La lisibilité de la notation des modules ASN.1 existants pourra éventuellement être améliorée (sans modification du train binaire transmis) en insérant la déclaration d'étiquetage automatique «AUTOMATIC TAGS» dans l'en-tête du module et en supprimant tout ou partie des étiquettes déclarées.

NOTE 2 – Une telle opération doit être effectuée avec prudence, et en ayant à l'esprit la manière dont l'étiquetage automatique fonctionne; incorrectement appliqué, l'étiquetage automatique peut provoquer la modification du train binaire transmis.

g) Si l'étiquetage automatique n'est pas appliqué à un module existant comme le décrit l'alinéa f) ci-dessus, il sera normalement préférable de ne pas ajouter de nouvelles définitions de types à ce module, mais de créer plutôt un nouveau module (avec étiquetage automatique) pour les nouvelles définitions de types. Ceci permettra de profiter des facilités de l'étiquetage automatique sans affecter le train binaire transmis.

h) Une attention particulière sera prêtée aux champs contenant des chaînes de caractères pour savoir s'il convient d'utiliser la notation «CHARACTER STRING», «BMPString» ou «UniversalString». Toutefois, ceci devrait normalement modifier le train binaire transmis, et devrait donc être effectué dans le cadre d'un changement de version.

i) Les identificateurs «mantissa», «base» et «exponent» doivent être ajoutés à toute notation de valeur réelle utilisant la forme «NumericRealValue» de la production «RealValue». On envisagera de restreindre la base aux valeurs 2 et 10 dans cette notation de type.

D'une manière générale, l'utilisation de la nouvelle notation ASN.1 devrait apporter des améliorations significatives en lisibilité, efficacité, précision et souplesse, notamment si les facilités offertes par les contraintes tabulaires et les contraintes relationnelles entre composants, la paramétrage et les nouveaux types de chaînes de caractères sont pleinement exploités. Tous les utilisateurs de la notation ASN.1-88/90 sont expressément invités à procéder à la migration chaque fois qu'une spécification est révisée, ou, si aucune révision n'est envisagée à terme, dans le cadre d'une activité spécifique.

L'amendement de modules existants en utilisant une notation non conforme à la présente spécification ASN.1 peut être considéré comme erroné, même s'il est fait référence aux spécifications ASN.1-88/90 dans ces modules. On évitera en particulier l'utilisation de la notation de macro, de déclarations «ANY» ou «ANY DEFINED BY», ainsi que la définition de nouvelles structures d'ensemble, de séquence ou de choix si ces structures ne sont pas pourvues d'identificateurs non ambigus.

Annexe B

Affectation de valeurs d'identificateurs d'objets

(Cette annexe fait partie intégrante de la présente Recommandation | Norme internationale)

La présente Recommandation | Norme internationale affecte les valeurs suivantes:

Référence	**Valeur d'identificateur d'objet**
36.3	{ joint-iso-itu-t asn1(1) specification(0) characterStrings(1) numericString(0) }
	Valeur de descripteur d'objet
	"NumericString ASN.1 type"
Référence	**Valeur d'identificateur d'objet**
36.5	{ joint-iso-itu-t asn1(1) specification(0) characterStrings(1) printableString(1) }
	Valeur de descripteur d'objet
	"PrintableString ASN.1 type"
Référence	**Valeur d'identificateur d'objet**
37.1	{ joint-iso-itu-t asn1(1) specification(0) modules(0) iso10646(0) }
	Valeur de descripteur d'objet
	"ASN1 Character Module"

Annexe C

Exemples et conseils stylistiques

(Cette annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

Cette annexe présente des exemples d'utilisation de la notation ASN.1 pour décrire des structures de données (fictives). Elle comporte également des conseils stylistiques et des directives concernant l'utilisation des diverses caractéristiques de la notation ASN.1. Sauf mention contraire, on suppose se trouver dans un environnement d'étiquetage automatique.

C.1 Exemple d'un enregistrement «salarié»

L'utilisation de la notation ASN.1 est illustrée par un enregistrement simple concernant un salarié fictif.

C.1.1 Description informelle de l'enregistrement «salarié»

La structure de l'enregistrement «salarié» est décrite ci-dessous, avec sa valeur pour un individu donné.

Nom: Jean P Martin

Fonction: Directeur

Matricule: 51

Date d'embauche: 17 septembre 1971

Nom du conjoint: Marie T Martin

Nombre d'enfants: 2

Renseignements enfant

- Nom: Marc T Martin
- Date de naissance: 11 novembre 1957

Renseignements enfant

- Nom: Anne B Dubois
- Date de naissance: 17 juillet 1959

C.1.2 Description en notation ASN.1 de la structure de l'enregistrement

La structure de chaque enregistrement «salarié» est décrite formellement au moyen de la notation normalisée des types de données.

```
EnregistrementSalarie ::= [APPLICATION 0] SET
{  nom                  Nom,
   fonction             VisibleString,
   matricule            Matricule,
   dateEmbauche         Date,
   nomDuConjoint        Nom,
   enfants              SEQUENCE OF InformationEnfant DEFAULT { }
}

InformationEnfant ::= SET
{  nom                  Nom,
   dateDeNaissance      [0] Date
}

Nom ::= [APPLICATION 1] SEQUENCE
{  prenom               VisibleString,
   initiale             VisibleString,
   nomDeFamille         VisibleString
}

Matricule ::= [APPLICATION 2] INTEGER

Date ::= [APPLICATION 3] VisibleString  -- AAAAMMJJ
```

Cet exemple illustre un aspect d'analyse de la syntaxe ASN.1. La structure syntaxique «DEFAULT» ne peut être appliquée qu'à un composant d'une séquence ou d'un ensemble; elle ne peut pas être appliquée à un composant d'un type séquence-de. La déclaration «DEFAULT { }» de «EnregistrementSalarie» s'applique par conséquent à la composante «enfants» et non à la notation «InformationEnfant».

C.1.3 Description en notation ASN.1 d'une valeur d'enregistrement

La valeur de l'enregistrement «salarié» de Jean P. Martin est décrite formellement ci-dessous en utilisant la notation normalisée des valeurs de données.

```
{ nom                {prenom "Jean", initiale "P", nomDeFamille "Martin"},
  fonction           "Directeur",
  matricule          51,
  dateEmbauche       "19710917",
  nomDuConjoint      {prenom "Marie", initiale "T", nomDeFamille "Martin"},
  enfants
   { {nom  {prenom "Marc", initiale "T", nomDeFamille "Martin"},
     dateDeNaissance "19571111"},
     {nom {prenom "Anne", initiale "B", nomDeFamille "Dubois"},
     dateDeNaissance "19590717" }
   }
}
```

C.2 Directives pour l'utilisation de la notation

La souplesse des types de données et de la notation formelle définis dans la présente Recommandation | Norme internationale permet de conceptualiser une grande variété de protocoles. Toutefois, cette souplesse peut parfois porter à confusion, en particulier lors d'une première utilisation de la notation. Cette annexe vise à minimiser les risques de confusion, en proposant des directives et des exemples concernant l'utilisation de la notation. Pour chacun des types de données prédéfinis, une ou plusieurs directives d'emploi sont proposées. Les types chaînes de caractères (les chaînes visibles par exemple) et les types définis dans les articles 41 à 43 ne sont pas traités ici.

C.2.1 Type booléen (BooleanType)

C.2.1.1 Utiliser un type booléen pour modéliser les valeurs d'une variable logique (c'est-à-dire à deux états), par exemple la réponse par oui ou par non à une question.

EXEMPLE

Salarie ::= BOOLEAN

C.2.1.2 Pour affecter un nom de référence à un type booléen, en choisir un qui représentee l'état **vrai**.

EXEMPLE

Marie ::= BOOLEAN

et non:

SituationDeFamille ::= BOOLEAN

C.2.2 Type entier (IntegerType)

C.2.2.1 Utiliser un type entier pour modéliser les valeurs (de grandeur illimitée en pratique) d'une variable cardinale ou entière.

EXEMPLE

VerificationBalanceDesComptes ::= INTEGER -- *en centimes; négatif signifie découvert*

equilibre VerificationBalanceDesComptes ::= 0

C.2.2.2 Définir les valeurs minimale et maximale autorisées d'un type entier comme nombres nommés.

EXEMPLE

Quantieme ::= INTEGER {premier(1), dernier(31)}

aujourdhui Quantieme ::= premier

dateInconnue Quantieme ::= 0

A noter que les nombres nommés «premier» et «dernier» ont été choisis à cause de leur contenu sémantique pour le lecteur, et n'excluent pas la possibilité d'avoir des Quantièmes d'une valeur inférieure à 1, supérieure à 31 ou compris entre 1 et 31.

Pour restreindre les valeurs de Quantième aux seules valeurs «premier» et «dernier», il faudrait écrire:

Quantieme ::= INTEGER {premier(1), dernier(31)} (premier | dernier)

et pour restreindre les valeurs de Quantième à toutes les valeurs comprises entre 1 et 31 inclusivement, il faudrait écrire:

Quantieme ::= INTEGER {premier(1), dernier(31)} (premier .. dernier)

quantieme Quantieme ::= 4

C.2.3 Type énuméré (EnumeratedType)

C.2.3.1 Utiliser un type énuméré pour modéliser les valeurs d'une variable qui possède trois états ou plus. Attribuer des valeurs en partant de zéro si la seule contrainte est qu'elles soient distinctes.

EXEMPLE

JourDeLaSemaine ::= ENUMERATED {dimanche(0), lundi(1), mardi(2),
mercredi(3), jeudi(4), vendredi(5), samedi(6)}

debutDeLaSemaine JourDeLaSemaine ::= dimanche

A noter que les valeurs énumérées «dimanche», «lundi», etc., ont été choisies à cause de leur contenu sémantique pour le lecteur, et qu'à partir de ce moment, le type «JourDeLaSemaine» est restreint à ces seules valeurs. Ainsi, une valeur ne peut se voir affecter que les seuls noms «dimanche», «lundi», etc., et non par exemple les entiers équivalents.

C.2.3.2 Utiliser un type énuméré extensible pour modéliser les valeurs d'une variable qui ne possède que deux états pour le moment, mais qui pourra en avoir d'autres dans une version ultérieure du protocole.

EXEMPLE

SituationDeFamille ::= ENUMERATED {celibataire, marie}
-- première version de la situation de famille

en prévision de:

SituationDeFamille ::= ENUMERATED {celibataire, marie, ..., veuf}
-- deuxième version de la situation de famille

et par la suite:

SituationDeFamille ::= ENUMERATED {celibataire, marie, ..., veuf, divorce}
-- troisième version de la situation de famille

C.2.4 Type réel (RealType)

C.2.4.1 Utiliser un type réel pour représenter un nombre irrationnel.

EXEMPLE

Angle-en-radians ::= REAL

pi REAL ::= {mantissa 314159265358979323846264338327 9, base 10, exponent –30}

C.2.4.2 Les concepteurs d'applications peuvent souhaiter garantir une interfonctionnement complet avec les valeurs réelles malgré les différences de représentation des nombres en virgule flottante dans les différents matériels, et décider par exemple d'utiliser des nombres en virgule flottante de longueur simple ou double dans une application X donnée. Ceci peut être réalisé de la manière suivante:

```
Reel-App-X ::= REAL (WITH COMPONENTS {
                    mantissa (–16777215..16777215),
                    base (2),
                    exponent (–125..128) } )
-- Les expéditeurs ne transmettront pas de valeurs hors de ces intervalles;
-- les destinataires conformes seront capables de recevoir et de traiter
-- toutes les valeurs respectant ces limites d'intervalles.

circonference Reel-App-X ::= {mantissa 16, base 2, exponent 1}
```

C.2.5 Type chaîne binaire (BitStringType)

C.2.5.1 Utiliser une chaîne binaire pour modéliser des données binaires dont le format et la longueur ne sont pas spécifiés, ou sont spécifiés ailleurs, et dont la longueur n'est pas nécessairement un multiple de huit bits.

EXEMPLE

```
TelecopieG3 ::= BIT STRING
  -- séquence binaire conforme à la Recommandation T.4

image TelecopieG3 ::= '100110100100001110110'B

finDeChaine BIT STRING ::= ' 0123456789ABCDEF'H

corps1 TelecopieG3 ::= '1101'B

corps2 TelecopieG3 ::= '1101000'B
```

A noter que «corps1» et «corps2» sont des valeurs abstraites différentes car les bits à 0 en fin de chaîne sont significatifs (en raison de l'absence d'une liste «NamedBitList» dans la définition de TélécopieG3).

C.2.5.2 Utiliser une chaîne binaire avec une contrainte de taille pour modéliser les valeurs d'un champ binaire de taille fixe.

EXEMPLE

```
Champ-binaire ::= BIT STRING (SIZE (12))

valeur1 Champ-binaire ::= '100110100100'B

valeur2 Champ-binaire ::= ' 9A4'H

valeur3 Champ-binaire ::= '1001101001'B      -- forme illicite: elle transgresse la contrainte de taille
```

A noter que «valeur1» et «valeur2» sont une même valeur abstraite, les 4 bits en fin de «valeur2» ne sont pas significatifs.

C.2.5.3 Utiliser une chaîne binaire pour modéliser les valeurs d'un **champ d'indicateurs**, d'un ensemble ordonné de variables logiques correspondant à une collection ordonnée d'objets et indiquant chacun si une condition particulière est remplie par l'objet correspondant de la collection.

```
JourDeLaSemaine ::= BIT STRING {
                        dimanche(0), lundi(1), mardi(2),
                        mercredi(3), jeudi(4), vendredi(5),
                        samedi(6) } (SIZE (0..7))

joursEnsoleillesDeLaSemaine1 JourDeLaSemaine ::= {dimanche, lundi, mercredi}
joursEnsoleillesDeLaSemaine2 JourDeLaSemaine ::= '1101'B
joursEnsoleillesDeLaSemaine3 JourDeLaSemaine ::= '1101000'B

joursEnsoleillesDeLaSemaine4 JourDeLaSemaine ::= '11010000'B -- forme illicite:
                                                              -- elle transgresse la contrainte de taille
```

A noter que si la longueur de la chaîne binaire est inférieure à 7, cela signifie que les jours correspondant aux bits manquants ont été maussades; les trois premières valeurs ont donc la même valeur abstraite.

C.2.5.4 Utiliser une chaîne binaire pour modéliser les valeurs d'un **champ d'indicateurs**, d'un ensemble ordonné de taille fixe de variables logiques correspondant à une collection ordonnée d'objets et indiquant chacun si une condition particulière est remplie par l'objet correspondant de la collection.

```
JourDeLaSemaine ::= BIT STRING {
                        dimanche(0), lundi(1), mardi(2),
                        mercredi(3), jeudi(4), vendredi(5),
                        samedi(6) } (SIZE (7))

joursEnsoleillesDeLaSemaine1 JourDeLaSemaine ::= {dimanche, lundi, mercredi}

joursEnsoleillesDeLaSemaine2 JourDeLaSemaine ::= '1101'B      -- forme illicite:
                                                              -- elle transgresse la contrainte de taille
joursEnsoleillesDeLaSemaine3 JourDeLaSemaine ::= '1101000'B

joursEnsoleillesDeLaSemaine4 JourDeLaSemaine ::= '11010000'B -- forme illicite:
                                                              -- elle transgresse la contrainte de taille
```

A noter que la première et la troisième valeur ont donc la même valeur abstraite.

C.2.5.5 Utiliser une chaîne binaire avec des bits nommés pour modéliser les valeurs d'un ensemble de variables logiques liées.

EXEMPLE

```
StatutPersonnel ::= BIT STRING
    {marie(0), salarie(1), ancienCombattant(2), diplomeUniversitaire(3)}

billClinton StatutPersonnel ::= {marie, salarie, diplomeUniversitaire}

hillaryClinton StatutPersonnel ::= '110100'B
```

A noter que «billClinton» et «hillaryClinton» ont les mêmes valeurs abstraites.

C.2.6 Type chaîne d'octets (OctetStringType)

C.2.6.1 Utiliser un type chaîne d'octets pour modéliser des données binaires dont le format et la longueur ne sont pas spécifiés, ou sont spécifiés ailleurs, et dont la longueur est un multiple de huit bits.

EXEMPLE

```
TelecopieG4 ::= OCTET STRING
-- séquence d'octets conforme aux
-- Recommandations T.5 et T.6

image TelecopieG4 ::= ' 3FE2EBAD471005'H
```

C.2.6.2 Utiliser un type chaîne de caractères à alphabet restreint approprié de préférence à un type chaîne d'octets lorsque cela est possible.

EXEMPLE

```
NomDeFamille ::= PrintableString

president NomDeFamille ::= "Clinton"
```

C.2.7 Types chaînes universelles (UniversalString) et chaînes multilingues (BMPString)

Utiliser le type chaîne multilingue «BMPString» pour modéliser toute chaîne informationnelle constituée des seuls caractères de la table multilingue (BMP, *basic multilingual plane*) de la norme ISO/CEI 10646-1, et le type chaîne universelle «UniversalString» pour modéliser toute chaîne constituée de caractères de la norme ISO/CEI 10646-1 mais non limitée à la table BMP.

C.2.7.1 Utiliser les déclarations «Level1» ou «Level2» pour indiquer que des restrictions sont imposées à l'utilisation de caractères de combinaison.

EXEMPLE

```
NomRusse ::= Cyrillic (Level1)                  -- Un NomRusse n'utilise pas
                                                -- de caractères de combinaison

NomSaoudien ::= BasicArabic (SIZE (1..100) ^ Level2)  -- Un NomSaoudien utilise un sous-ensemble
                                                -- de caractères de combinaison
```

C.2.7.2 Une collection de caractères peut être étendue et transformée en un sous-ensemble sélectionné (c'est-à-dire en lui ajoutant tous les caractères du jeu latin de base) au moyen du symbole de réunion «UnionMark» (voir l'article 46).

EXEMPLE

```
KatakanaEtLatinDeBase ::= UniversalString (FROM(Katakana | BasicLatin))
```

C.2.8 Type chaîne de caractères (CharacterStringType)

Utiliser un type chaîne de caractères à alphabet non restreint pour modéliser toute chaîne informationnelle qui ne peut être modélisée par l'un des types de chaînes de caractères à alphabet restreint. Ne pas oublier de spécifier le répertoire de caractères et leur codage en octets.

EXEMPLE

```
ChaineDCBCondensee ::= CHARACTER STRING ( WITH COMPONENTS {
                                                identification  ( WITH COMPONENTS {
                                                    fixed  PRESENT } )
                        -- Les syntaxes abstraite et de transfert seront les syntaxes
                        -- SyntaxeAbstraiteDeChaineDCBCondensee et
                        -- SyntaxeDeTransfertDeChaineDCBCondensee définies ci-dessous
                    } )

-- valeur d'identificateur d'objet pour une syntaxe abstraite de caractères (jeu de caractères)
-- dont l'alphabet est constitué des chiffres de 0 à 9.
identificateurDeSyntaxeAbstraiteDeChaineDCBCondensee  OBJECT  IDENTIFIER ::=
    {joint-iso-itu-t xxx(999) yyy(999) zzz(999) dCBCondense(999) jeuDeCaracteres(0) }

-- valeur d'identificateur d'objet pour une syntaxe de transfert de caractères qui condense deux
-- chiffres par octet, chaque octet étant codé de 0000 à 1001, 1111₂ servant au remplissage.
identificateurDeSyntaxeDeTransfertDeChaineDCBCondensee  OBJECT IDENTIFIER ::=
    {joint-iso-itu-t xxx(999) yyy(999) zzz(999) dCBCondense(999) syntaxeDeTransfertDeCaracteres(1) }

-- Le codage d'une valeur du type ChaîneDCBCondensee ne comportera que le codage défini des
-- caractères avec le champ de longueur nécessaire, ainsi que le champ étiquette en cas de codage selon
-- les règles de codage de base BER. Les valeurs d'identificateurs d'objets ne sont pas
-- transmises, la notation spécifiant le mode «fixed».
```

NOTE – Les règles de codage ne codent pas toujours les valeurs du type chaîne de caractères «CHARACTER STRING» sous une forme qui comporte nécessairement des valeurs d'identificateurs d'objets, bien que de tels identificateurs garantissent la préservation de la valeur abstraite dans le codage.

C.2.9 Type néant (NullType)

Utiliser un type néant pour indiquer l'absence effective d'un élément d'une séquence.

EXEMPLE

```
IdentificationMalade ::= SEQUENCE {
    nom                 VisibleString,
    numeroChambre       CHOICE {
        chambre             INTEGER,
        nonHospitalise      NULL  -- s'il s'agit d'un malade non hospitalisé  --
    }
}
```

```
dernierMalade IdentificationMalade ::= {
    nom                 "Jean Durand",
    numeroChambre       nonHospitalise: NULL
}
```

C.2.10 Types séquence et séquence-de (Sequence, SequenceOfType)

C.2.10.1 Utiliser un type séquence-de pour modéliser une collection de variables appartenant à un même type, en nombre élevé ou imprévisible, et dont l'ordre est significatif.

EXEMPLE

```
NomsDesPaysMembres ::= SEQUENCE OF VisibleString
-- par ordre alphabétique

deuxPremiers  NomsDesPaysMembres ::= {"Australie", "Autriche"}
```

C.2.10.2 Utiliser un type séquence pour modéliser une collection de variables de types différents, en nombre limité et connu, et dont l'ordre est significatif, la composition de la collection ayant peu de chance de changer d'une version du protocole à la suivante.

EXEMPLE

```
NomDesMembresDuBureau ::= SEQUENCE {
    president           VisibleString,
    vicePresident       VisibleString,
    secretaire          VisibleString}

cieAcme NomDesMembresDuBureau ::= {
    president           "Eric Martin",
    vicePresident       "Jean Martin",
    secretaire          "Paul Martin"}
```

C.2.10.3 Utiliser un type séquence inextensible pour modéliser une collection de variables de types différents, en nombre connu et limité, et dont l'ordre est significatif, la composition de cette collection ayant peu de chance de changer d'une version du protocole à la suivante.

EXEMPLE

```
Habilitation ::= SEQUENCE {
    nomUtilisateur      VisibleString,
    motDePasse          VisibleString,
    numeroDeCompte      INTEGER}
```

C.2.10.4 Utiliser un type séquence extensible pour modéliser une collection de variables dont l'ordre est significatif et dont le nombre actuel est faible et connu mais dont il est prévu qu'il puisse être augmenté.

EXEMPLE

```
Enregistrement ::= SEQUENCE {                   -- première version d'Enregistrement
    nomUtilisateur      VisibleString,
    motDePasse          VisibleString,
    numeroDeCompte      INTEGER,
    ...,
    ...
}
```

en prévision de:

```
Enregistrement ::= SEQUENCE {                   -- deuxième version d'Enregistrement
    nomUtilisateur      VisibleString,
    motDePasse          VisibleString,
    numeroDeCompte      INTEGER,
    ...,
    [[                                          -- extension faite en version 2
      derniereConnexion         GeneralizedTime OPTIONAL,
      minutesDerniereConnexion  INTEGER
    ]],
    ...
}
```

et par la suite:

```
Enregistrement ::= SEQUENCE {                                   -- troisième version d'Enregistrement
    nomUtilisateur              VisibleString,
    motDePasse                  VisibleString,
    numeroDeCompte              INTEGER,
    …,
    [[                                                          -- extension faite en version 2
      derniereConnexion         GeneralizedTime OPTIONAL,
      minutesDerniereConnexion  INTEGER
    ]],
    [[                                                          -- extension faite en version 3
      certificat                Certificat,
      empreinte                 Empreinte OPTIONAL
    ]],
    …
}
```

C.2.11 Types ensemble et ensemble-de (SetType, SetOfType)

C.2.11.1 Utiliser un type ensemble pour modéliser une collection de variables en nombre connu et limité, et dont l'ordre n'est pas significatif. En l'absence d'étiquetage automatique, identifier chaque variable par un étiquetage spécifique au contexte comme le montre l'exemple suivant (en étiquetage automatique, les étiquettes ne sont pas nécessaires).

EXEMPLE

```
NomUtilisateur ::= SET {
    nomPersonnel                [0] VisibleString,
    nomDOrganisation            [1] VisibleString,
    nomDePays                   [2] VisibleString}

utilisateur NomUtilisateur ::= {
    nomDePays                   "Nigeria",
    nomPersonnel                "Jonas Maruba",
    nomDOrganisation            "Meteorology, Ltd."}
```

C.2.11.2 Utiliser un type ensemble avec déclaration «OPTIONAL» pour modéliser une collection de variables qui est un sous-ensemble (strict ou non strict) d'une autre collection de variables, dont le nombre est connu et raisonnablement petit, et dont l'ordre n'est pas significatif. En l'absence d'étiquetage automatique, identifier chaque variable par un étiquetage spécifique au contexte comme le montre l'exemple suivant (en étiquetage automatique, les étiquettes ne sont pas nécessaires).

EXEMPLE

```
NomUtilisateur ::= SET {
    nomPersonnel                [0] VisibleString,
    nomDOrganisation            [1] VisibleString OPTIONAL,
        -- par défaut, nom de l'organisation locale --
    nomDePays                   [2] VisibleString OPTIONAL
        -- par défaut, nom du pays local -- }
```

C.2.11.3 Utiliser un type ensemble extensible pour modéliser une collection de variables dont la composition risque de changer d'une version du protocole à la suivante. On fait l'hypothèse, dans ce qui suit, qu'une déclaration «AUTOMATIC TAGS» a été faite lors de la définition du module.

EXEMPLE

```
NomUtilisateur ::= SET {
    nomPersonnel                VisibleString,              -- première version de NomUtilisateur
    nomDOrganisation            VisibleString OPTIONAL,
    nomDePays                   VisibleString OPTIONAL,
    …,
    …
}

utilisateur  NomUtilisateur ::= {nomPersonnel  "Jonas Maruba" }
```

en prévision de:

```
NomUtilisateur ::= SET {                                    -- deuxième version de NomUtilisateur
    nomPersonnel                VisibleString,
    nomDOrganisation            VisibleString OPTIONAL,
    nomDePays                   VisibleString OPTIONAL,
    ...,
    [[                                                      -- extension faite dans la version 2
      adresseMessagerieInternet VisibleString,
      numeroDeFax               VisibleString OPTIONAL
    ]],
    ...
}

utilisateur NomUtilisateur ::= {
    nomPersonnel                "Jonas Maruba",
    adresseMessagerieInternet   " jonas@meteor.ngo.com"
}
```

et par la suite:

```
NomUtilisateur ::= SET {                                    -- troisième version de nomUtilisateur
    nomPersonnel                VisibleString,
    nomDOrganisation            VisibleString OPTIONAL,
    nomDePays                   VisibleString OPTIONAL,
    ...,
    [[                                                      -- extension faite dans la version 2
      adresseMessagerieInternet VisibleString,
      numeroDeFax               VisibleString OPTIONAL
    ]],
    numeroDeTelephone           VisibleString OPTIONAL,     -- extension faite dans la version 3
    ...
}

utilisateur  NomUtilisateur ::= {
    nomPersonnel                "Jonas Maruba",
    adresseMessagerieInternet   "jonas@meteor.ngo.com"
}
```

C.2.11.4 Utiliser un type ensemble-de pour modéliser une collection de variables appartenant à un même type et dont l'ordre n'est pas significatif.

EXEMPLE

```
MotsCles ::= SET OF VisibleString  --  en ordre arbitraire

quelquesMotsClesASN1  MotsCles ::= {"INTEGER", "BOOLEAN", "REAL"}
```

C.2.12 Type étiqueté (TaggedType)

Avant l'introduction de la déclaration d'étiquetage automatique «AUTOMATIC TAGS», les spécifications ASN.1 comportaient souvent des étiquettes. Les points suivants décrivent la manière dont l'étiquetage était typiquement appliqué. Avec l'introduction de l'étiquetage automatique, les auteurs de spécifications ASN.1 n'ont plus besoin d'utiliser la notation d'étiquettes, mais ceux qui apportent des modifications à des spécifications anciennes doivent en comprendre le fonctionnement.

C.2.12.1 Les étiquettes de la classe universelle ne sont utilisées que dans la présente Recommandation | Norme internationale. La notation «[UNIVERSAL 30]» (par exemple) n'est indiquée que pour assurer la précision de la définition des types utiles normalisés à l'échelle internationale. Elle ne doit pas être utilisée ailleurs.

C.2.12.2 Un cas fréquent d'utilisation des étiquettes est l'affectation d'une seule étiquette de la classe application à l'ensemble de la spécification, cette étiquette servant à identifier un type qui s'avère être largement utilisé un peu partout dans la spécification. On utilise souvent aussi une seule étiquette de la classe application (une seule fois) pour étiqueter le type choix le plus externe d'une application, en assurant l'identification des différents messages par l'étiquette de la classe application. Le premier cas est illustré dans l'exemple ci-dessous:

EXEMPLE

```
NomDeFichier ::= [APPLICATION 8] SEQUENCE {
    nomDeRepertoire         VisibleString,
    nomDeFichierSimple      VisibleString}
```

C.2.12.3 L'étiquetage propre au contexte est fréquemment appliqué de manière algorithmique à tous les composants d'une structure ensemble, séquence ou choix. A noter toutefois que l'étiquetage automatique décharge le concepteur de cette tâche.

EXEMPLE

```
EnregistrementClient ::= SET {
    nom                     [0] VisibleString,
    adressePostale          [1] VisibleString,
    numeroDeCompte          [2] INTEGER,
    solde                   [3] INTEGER -- en centimes --}

AttributClient ::= CHOICE {
    nom                     [0] VisibleString,
    adressePostale          [1] VisibleString,
    numeroDeCompte          [2] INTEGER,
    solde                   [3] INTEGER -- en centimes --}
```

C.2.12.4 L'étiquetage en classe privée ne devrait normalement pas être utilisé dans les spécifications internationales (bien que ceci ne puisse être interdit). Les applications développées par les entreprises devraient normalement utiliser les classes d'étiquetage application et propres au contexte. Mais une situation particulière peut se présenter dans laquelle une spécification propre à une entreprise apporte une extension à une spécification internationale; dans une telle circonstance, l'utilisation d'étiquettes de la classe privée a l'avantage de protéger partiellement la spécification propre à l'entreprise d'une modification de la spécification internationale.

EXEMPLE

```
NumeroDeBadgeAcme ::= [PRIVATE 2] INTEGER

numeroDeBadge NumeroDeBadgeAcme ::= 2345
```

C.2.12.5 L'ajout de la déclaration «IMPLICIT» à chaque étiquette n'est généralement trouvé que sur les spécifications les plus anciennes. Les règles de codage de base (BER) produisent une représentation moins compacte avec l'étiquetage explicite qu'avec l'étiquetage implicite. Les règles de codage compact (PER) aboutissent à la même compacité de codage dans les deux cas. Avec les règles BER et l'étiquetage explicite, il y a une meilleure visibilité du type sous-jacent (entier, réel, booléen, etc.) dans les données codées. Les présentes directives utilisent l'étiquetage implicite dans les exemples chaque fois qu'il est licite de le faire. Selon les règles de codage, cela peut aboutir à une représentation compacte, particulièrement recherchée dans certaines applications. Dans d'autres applications, la compacité peut être moins importante par exemple que la possibilité d'effectuer une solide vérification de type. Dans une telle situation, on peut recourir à l'étiquetage explicite.

EXEMPLE

```
EnregistrementClient ::= SET {
    nom                     [0] IMPLICIT VisibleString,
    adressePostale          [1] IMPLICIT VisibleString,
    numeroDeCompte          [2] IMPLICIT INTEGER,
    solde                   [3] IMPLICIT INTEGER -- en centimes --}

AttributClient ::= CHOICE {
    nom                     [0] IMPLICIT VisibleString,
    adressePostale          [1] IMPLICIT VisibleString,
    numeroDeCompte          [2] IMPLICIT INTEGER,
    solde                   [3] IMPLICIT INTEGER -- en centimes --}
```

C.2.12.6 Les directives d'étiquetage pour les nouvelles spécifications ASN.1 d'utilisateur faisant référence à la présente Recommandation | Norme internationale sont très simples: NE PAS ÉTIQUETER. Mettre la déclaration «AUTOMATIC TAGS» dans l'en-tête du module, et ne plus s'en occuper. S'il s'avère nécessaire d'ajouter un nouveau composant à une structure ensemble, séquence ou choix dans une version ultérieure, l'ajouter à la fin.

C.2.13 Type choix

C.2.13.1 Utiliser un type choix pour modéliser une variable choisie dans un ensemble de variables en nombre connu et limité.

EXEMPLE

```
IdentificateurDeFichier ::= CHOICE {
    nomRelatif          VisibleString,
        -- nom du fichier (par exemple, "RapportAvancementMars")
    nomAbsolu           VisibleString,
        -- noms du fichier et du répertoire qui le contient
        -- (par exemple, "<Williams>RapportAvancementMars")
    numeroDeSerieINTEGER
        -- identificateur affecté par le système au fichier  -- }

fichier IdentificateurDeFichier ::= numeroDeSerie: 106448503
```

C.2.13.2 Utiliser un type choix extensible pour modéliser une variable choisie dans un ensemble de variables dont la composition risque de changer d'une version du protocole à la suivante.

EXEMPLE

```
IdentificateurDeFichier ::= CHOICE {                    -- première version d'IdentificateurDeFichier
    nomRelatif          VisibleString,
    nomAbsolu           VisibleString,
    ...,
    ...
}

idChamp1 IdentificateurDeFichier ::= nomRelatif: "RapportAvancementMars.doc"
```

en prévision de:

```
IdentificateurDeFichier ::= CHOICE {                    -- deuxième version d'IdentificateurDeFichier
    nomRelatif          VisibleString,
    nomAbsolu           VisibleString,
    ...,
    numeroDeSerieINTEGER,                       -- addition d'extension faite en version 2
    ...
}

idChamp1  IdentificateurDeFichier ::= nomRelatif: "RapportAvancementMars.doc"

idChamp2  IdentificateurDeFichier ::= numeroDeSerie: 214
```

et par la suite:

```
IdentificateurDeFichier ::= CHOICE {                    -- troisième version d'IdentificateurDeFichier
    nomRelatif          VisibleString,
    nomAbsolu           VisibleString,
    ...,
    numeroDeSerieINTEGER,                       -- addition d'extension faite en version 2
    [[                                              -- addition d'extension faite en version 3
      specifiqueFournisseur   ExtFournisseur,
      nonIdentifie          NULL
    ]],
    ...
}
```

```
idChamp1  IdentificateurDeFichier ::= nomRelatif: "RapportAvancementMars.doc"

idChamp2  IdentificateurDeFichier ::= numeroDeSerie: 214

idChamp3  IdentificateurDeFichier ::= nonIdentifie: NULL
```

C.2.13.3 Utiliser un type choix extensible avec un seul type lorsqu'il est envisagé d'avoir ultérieurement plusieurs types possibles.

EXEMPLE

```
Voeux     ::= CHOICE {                                  -- première version de Voeux
     cartePostale          VisibleString,
     …,
     …
}
```

en prévision de:

```
Voeux     ::= CHOICE {                                  -- deuxième version de Voeux
     cartePostale          VisibleString,
     …,
     [[                                                  -- addition d'extension faite en version 2
       audio          Audio,
       video          Video
     ]],
     …
}
```

C.2.13.4 Utiliser plusieurs signes «deux-points» lorsqu'un type choix est imbriqué dans un autre type choix.

EXEMPLE

```
Voeux     ::= [APPLICATION 12] CHOICE {
     cartePostale          VisibleString,
     enregistrement        Voix}

Voix ::= CHOICE {
     anglais               OCTET STRING,
     swahili               OCTET STRING }

mesVoeux Voeux::=  enregistrement: anglais: ' 019838547E0'H
```

C.2.14 Type sélection

C.2.14.1 Utiliser un type sélection pour modéliser une variable dont le type est celui d'une des formes particulières d'un choix défini antérieurement.

C.2.14.2 Considérons la définition:

```
AttributFichier ::= CHOICE {
     date-derniere-utilisation   INTEGER,
     nom-fichier                 VisibleString}
```

La définition suivante est alors possible:

```
ListeDesAttributs ::= SEQUENCE {
     premier-attribut          date-derniere-utilisation < AttributFichier,
     second-attribut       nom-fichier < AttributFichier}
```

avec la notation de valeur possible suivante:

```
listeDesAttributs  ListeDesAttributs ::=  {
     premier-attribut          27,
     second-attribut       "PROGRAMME" }
```

C.2.15 Type champ de classe d'objets

C.2.15.1 Utiliser un type champ de classe d'objets pour identifier un type défini au moyen d'une classe d'objets informationnels (voir la Rec. UIT-T X.681 | ISO/CEI 8824-2). Par exemple, les champs de la classe d'objets informationnels «ATTRIBUT» peuvent être utilisés pour définir un type «Attribut».

EXEMPLE

```
ATTRIBUT ::= CLASS
{
    &TypeDeLAttribut,
    &idDeLAttribut      OBJECT IDENTIFIER UNIQUE
}
Attribut ::= SEQUENCE {
    idDeLAttribut       ATTRIBUT.&idDeLAttribut,        -- normalement, une contrainte est imposée.
    valeurDeLAttribut   ATTRIBUT.&TypeDeLAttribut       -- normalement, une contrainte est imposée.
}
```

«ATTRIBUT.&idDeLAttribut» et «ATTRIBUT.&TypeDeLAttribut» sont tous deux des types de champ de classe d'objets, en ce sens qu'il s'agit de types définis par référence à une classe d'objets informationnels «ATTRIBUT». Le type «ATTRIBUT.&idDeLAttribut» est fixé parce qu'il est explicitement défini dans «ATTRIBUT» comme identificateur d'objet «OBJECT IDENTIFIER». Par contre, le type «ATTRIBUT.&TypeDeLAttribut» peut contenir une valeur de n'importe quel type défini en ASN.1, puisque son type n'est pas fixé dans la définition de la classe «ATTRIBUT». Une notation qui possède cette propriété de pouvoir véhiculer une valeur de n'importe quel type est dite «notation de type ouvert». Ainsi, «ATTRIBUT.&TypeDeLAttribut» est un type ouvert.

C.2.16 Type valeur de donnée de présentation enchâssée (EmbeddedPDVType)

C.2.16.1 Utiliser un type valeur de donnée de présentation enchâssée pour modéliser une variable dont le type n'est pas spécifié, ou l'est ailleurs sans restriction quant à la notation utilisée pour le spécifier.

EXEMPLE

```
ContenuDuFichier ::= EMBEDDED PDV

ListeDeDocuments ::= SEQUENCE OF EMBEDDED PDV
```

C.2.17 Type externe (ExternalType)

Le type externe est similaire au type valeur de donnée de présentation enchâssée, mais dispose d'un nombre plus restreint de possibilités d'identification. On préférera généralement dans les nouvelles spécifications utiliser le type valeur de donnée de présentation enchâssée à cause de sa plus grande souplesse et parce que certaines règles de codage encodent les valeurs de manière plus efficace.

C.2.18 Type instance-de (InstanceOfType)

C.2.18.1 Utiliser une déclaration instance-de pour spécifier un type contenant un champ identificateur d'objet et un type ouvert dont la valeur appartient à un type déterminé par cet identificateur d'objet. Les types instance-de sont astreints à véhiculer une valeur de la classe identificateur de type «TYPE-IDENTIFIER» (voir les Annexes A et C de la Rec. UIT-T X.681 | ISO/CEI 8824-2).

EXEMPLE

```
CLASSE-CONTROLE-D-ACCES ::= TYPE-IDENTIFIER

RequeteDeLecture ::= SEQUENCE {
    classeDObjet        ClasseDObjet,
    instanceDObjet      InstanceDObjet,
    controleDAcces      INSTANCE OF CLASSE-CONTROLE-D-ACCES,
                                -- normalement, une contrainte est imposée
    idDeLAttribut       ATTRIBUT.&idDeLAttribut
}
```

Le type «RequeteDeLecture» est alors équivalent à:

```
RequeteDeLecture ::= SEQUENCE {
    classeDObjet        ClasseDObjet,
    instanceDObjet      InstanceDObjet,
    controleDAcces      [UNIVERSAL 8] IMPLICIT SEQUENCE {
        idDuType        CLASSE-CONTROLE-D-ACCES.&id,
                                -- normalement, une contrainte est imposée
        valeur          [0] CLASSE-CONTROLE-D-ACCES.&Type
                                -- normalement, une contrainte est imposée
    },
    idDeLAttribut       ATTRIBUT.&idDeLAttribut
}
```

La véritable utilité du type instance-de n'apparaît que lorsqu'il est contraint au moyen d'un ensemble d'objets informationnels, mais un tel exemple sort du cadre de la présente Recommandation | Norme internationale. Se reporter à la Rec. UIT-T X.682 | ISO/CEI 8824-3 pour la définition de l'ensemble d'objets informationnels, et à l'Annexe A du même document pour la manière d'utiliser un ensemble d'objets informationnels pour contraindre un type instance-de. A noter que le codage d'une instance-de «CLASSE-CONTROLE-D-ACCES» est le même que celui obtenu par une valeur de type externe n'ayant qu'un identificateur d'objet et une valeur de donnée.

C.3 Identification des syntaxes abstraites

C.3.1 L'utilisation du service de présentation défini dans la Rec. UIT-T X.216 | ISO/CEI 8822 impose de spécifier des valeurs appelées valeurs de donnée de présentation et le groupement de ces valeurs en des ensembles appelés syntaxes abstraites. Chacun de ces ensembles reçoit un nom de syntaxe abstraite du type ASN.1 identificateur d'objet.

C.3.2 La notation ASN.1 peut servir d'outil général pour la spécification des valeurs de donnée de présentation et leur groupement en syntaxes abstraites nommées.

C.3.3 Dans le cas le plus simple d'une telle utilisation, il existe un seul type ASN.1 et chaque valeur de donnée de présentation de la syntaxe abstraite nommée est une valeur de ce type ASN.1. Normalement, il s'agira d'un type choix, et chaque valeur de donnée de présentation sera une forme possible de ce type choix. Dans ce cas, il est recommandé de définir au début du module ASN.1 ce type choix, suivi de la définition des types (non universels) auxquels ce type choix se réfère directement ou non.

NOTE – Cette disposition n'exclut pas la référence à des types définis dans d'autres modules.

C.3.4 Il est recommandé que l'affectation d'un identificateur d'objet et d'un descripteur d'objet à une syntaxe abstraite soit effectuée au moyen de la classe d'objets informationnels utile «ABSTRACT-SYNTAX» définie dans la Rec. UIT-T X.681 | ISO/CEI 8824-2. Il est également recommandé que toutes les utilisations de la classe «ABSTRACT-SYNTAX» soient regroupées dans un même module «racine» identifiant toutes les syntaxes abstraites utilisées par une norme d'application.

C.3.5 Ce qui suit est un exemple de texte pouvant apparaître dans une norme d'application:

EXEMPLE

```
ISOxxxx-yyyy {iso standard xxxx asn1-modules(...) yyyy-pdu(...)} DEFINITIONS ::=
BEGIN
    EXPORTS YYYY-PDU;

    YYYY-PDU ::= CHOICE {
        pduDeConnexion    ..... ,
        pduDeDonnees    CHOICE {
            ..... ,
            .....
        },
        .....
    }
    ......
END

ISOxxxx-yyyy-Abstract-Syntax-Module {iso standard xxxx asn1-modules(...) } DEFINITIONS ::=
BEGIN
    IMPORTS YYYY-PDU FROM ISOxxxx-yyyy {iso standard xxxx asn1-modules(...) yyyy-pdu(...)};
```

```
-- La présente Recommandation | Norme internationale définit la syntaxe abstraite suivante:

    YYYY-Abstract-Syntax ABSTRACT-SYNTAX ::=
        { YYYY-PDU  IDENTIFIED BY  yyyy-abstract-syntax-object-id }

    yyyy-abstract-syntax-object-id OBJECT IDENTIFIER ::= {iso standard yyyy(xxxx) abstract-syntax(...) }

-- Le descripteur d'objet correspondant est:

    yyyy-abstract-syntax-descriptor ObjectDescriptor ::= "....................."

-- Les valeurs d'identificateur d'objet et de descripteur d'objet ASN.1:
                        -- identificateur d'objet de règles de codage
                        -- descripteur d'objet de règles de codage
-- affectées aux règles de codage dans les Rec. UIT-T X.690 | ISO/CEI 8825-1
-- et Rec. UIT-T X.691 | ISO/CEI 8825-2 peuvent être utilisées comme identificateur de la syntaxe de
-- transfert en conjonction avec la syntaxe abstraite présente ISOxxxx-yyyy-Abstract-Syntax.

END
```

C.3.6 Pour garantir l'interfonctionnement, la norme peut rendre de surcroît obligatoire la prise en charge de la syntaxe de transfert obtenue en appliquant les règles de codage mentionnées dans son module de syntaxe abstraite.

C.4 Sous-types

C.4.1 Utiliser des sous-types pour restreindre un type existant à des valeurs particulières.

EXEMPLE

```
NumeroAtomique  ::=  INTEGER (1..104)

SuiteDeTouchesTelephoniques ::=  IA5String
      (FROM  ("0123456789" | "*" | "#" )) (SIZE (1..63))

ListeDesParametres ::= SET SIZE (1..63) OF Parametre

PetitNombrePremier  ::=  INTEGER (2|3|5|7|11|13|17|19|23|29)
```

C.4.2 Utiliser une contrainte de sous-type extensible pour modéliser un type «INTEGER» dont l'ensemble de valeurs autorisées est petit et bien défini, mais dont il est prévu qu'il puisse croître.

EXEMPLE

```
PetitNombrePremier ::= INTEGER (2 | 3, ...)          -- première version de PetitNombrePremier
```

en prévision de:

```
PetitNombrePremier ::= INTEGER (2 | 3, ..., 5 | 7 | 11)          -- deuxième version de PetitNombrePremier
```

et par la suite:

```
PetitNombrePremier ::= INTEGER (2 | 3, ..., 5 | 7 | 11 | 13 | 17 | 19)
                                                  -- troisième version de PetitNombrePremier
```

NOTE – Certaines règles de codage (par exemple les règles PER) fournissent, pour certains types, un codage hautement optimisé pour les valeurs de racine d'extension de contrainte de sous-type (c'est-à-dire pour les valeurs apparaissant avant la notation «...») et un codage moins optimisé des valeurs d'addition d'extension de contrainte de sous-type (c'est-à-dire pour les valeurs apparaissant après la notation «...»), tandis que les contraintes de sous-type n'ont aucun effet sur le codage effectué par d'autres règles (par exemple les règles BER).

C.4.3 Lorsque deux types apparentés ou plus ont de nombreux points communs, envisager de définir explicitement leur parent commun comme un type, et d'en dériver chacun des types par sous-typage. Cette solution met en relief leurs relations et leurs points communs et permet (sans aucune obligation) de maintenir ces relations au fur et à mesure de l'évolution des types. Elle facilite donc l'adoption d'approches communes pour le traitement des valeurs de ces types.

EXEMPLE

```
Enveloppe ::=  SET {
               typeA TypeA,
               typeB TypeB   OPTIONAL,
               typeC TypeC   OPTIONAL}
               -- le parent commun
```

```
EnveloppeAB ::= Enveloppe (WITH COMPONENTS
                {     … ,
                      typeB      PRESENT, typeC     ABSENT})
                      -- le typeB devant toujours apparaître mais pas le typeC

EnveloppeAC ::= Enveloppe (WITH COMPONENTS
                {     … ,
                      typeB      ABSENT, typeC      PRESENT})
                      -- le typeC devant toujours apparaître mais pas le typeB
```

Ces dernières définitions peuvent aussi être exprimées de la manière suivante:

```
EnveloppeAB ::= Enveloppe (WITH COMPONENTS {typeA, typeB})

EnveloppeAC ::= Enveloppe (WITH COMPONENTS {typeA, typeC})
```

Le choix entre les différentes formes dépend de facteurs comme le nombre de composants du type parent et, parmi ceux-ci, le nombre de ceux qui sont optionnels, l'ampleur des différences entre les types concernés et la stratégie d'évolution probable.

C.4.4 Recourir au sous-typage pour définir partiellement une valeur, lorsqu'une unité de données de protocole par exemple doit être soumise à un test de conformité ne portant que sur certains de ses composants.

EXEMPLE

Soit:

```
PDU  ::=  SET
 {alpha          INTEGER,
  beta           IA5String  OPTIONAL,
  gamma          SEQUENCE OF Parametre,
  delta          BOOLEAN}
```

alors, pour établir un test exigeant que le booléen soit faux et l'entier négatif, écrire:

```
PDUTest   ::=  PDU (WITH COMPONENTS
   {… ,
    delta (FALSE),
    alpha (MIN..<0)})
```

et si, de plus, la chaîne du type alphabet international n° 5, beta, doit être présente et longue de 5 ou 12 caractères, écrire:

```
PDUTestSupplementaire ::= PDUTest (WITH COMPONENTS { ... , beta (SIZE (5|12)) PRESENT } )
```

C.4.5 Si un type de données d'usage général a été défini comme un type séquence-de, recourir au sous-typage pour définir un sous-type restreint de ce type général:

EXEMPLE

```
BlocDeTexte ::= SEQUENCE OF VisibleString

Adresse ::= BlocDeTexte (SIZE (1..6)) (WITH COMPONENT (SIZE (1..32)))
```

C.4.6 Si un type de données d'usage général a été défini comme un type choix, recourir au sous-typage pour définir un sous-type restreint de ce type général:

EXEMPLE

```
Z ::= CHOICE {
      a          A,
      b          B,
      c          C,
      d          D,
      e          E
}

V ::= Z (WITH COMPONENTS {…, a  ABSENT,  b  ABSENT })  -- «a» et «b» doivent être absents, et
                                                       -- soit «c», soit «d», soit «e» peut être
                                                       -- présent dans la valeur.
```

```
W ::= Z (WITH COMPONENTS { ..., a  PRESENT })        -- seul «a» doit être présent (voir 48.8.9.2).

X ::= Z (WITH COMPONENTS { a  PRESENT })             -- seul «a» doit être présent (voir 48.8.9.2).

Y ::= Z (WITH COMPONENTS { a  ABSENT,  b,  c })      -- «a», «d» et «e» doivent être absents, et
                                                     -- soit «b», soit «c» peut être présent dans
                                                     -- la valeur.
```

NOTE – Les types «W» et «X» sont sémantiquement identiques.

C.4.7 Utiliser le sous-typage par contenance de type pour former de nouveaux sous-types à partir de sous-types existants:

EXEMPLE

```
Mois      ::=  ENUMERATED {
                 janvier          (1),
                 fevrier          (2),
                 mars             (3),
                 avril            (4),
                 mai              (5),
                 juin             (6),
                 juillet          (7),
                 aout             (8),
                 septembre(9),
                 octobre          (10),
                 novembre         (11),
                 decembre         (12) }

Premier-trimestre ::= Mois (
                          janvier    |
                          fevrier    |
                          mars )

Deuxieme-trimestre ::= Mois (
                          avril      |
                          mai        |
                          juin )

Troisieme-trimestre ::= Mois (
                          juillet    |
                          aout       |
                          septembre )

Quatrieme-trimestre ::= Mois (
                          octobre    |
                          novembre   |
                          decembre )

Premier-semestre ::= Mois ( Premier-trimestre | Deuxieme-trimestre )

Deuxieme-semestre ::= Mois ( Troisieme-trimestre | Quatrieme-trimestre )
```

Annexe D

Annexe didactique sur les chaînes de caractères ASN.1

(Cette annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

D.1 Prise en charge des chaînes de caractères en notation ASN.1

D.1.1 La notation ASN.1 prend en charge quatre groupes de chaînes de caractères, à savoir:

a) les types de chaînes de caractères basés sur le *Registre international des jeux de caractères codés à utiliser avec une séquence d'échappement de l'ISO* (c'est-à-dire la structure de la norme ISO/CEI 646) et le Registre international associé des jeux de caractères codés, constitués par les types de chaîne visible «VisibleString», alphabet international n° 5 «IA5String», télétex «TeletexString», vidéotex «VideotexString», graphique «GraphicString» et générale «GeneralString»;

b) les types de chaînes de caractères basés sur la norme ISO/CEI 10646-1, obtenus par restriction des types chaîne universelle «UniversalString», format de transformation UCS 8 «UTF8String» ou multilingue «BMPString» à des sous-ensembles définis dans la norme ISO/CEI 10646-1 ou en utilisant des caractères nommés;

NOTE 1 – L'utilisation de ces types sans contrainte conduit à une transgression des prescriptions de conformité concernant l'échange d'informations telles qu'elles sont spécifiées dans la norme ISO/CEI 10646-1 car aucun sous-ensemble adopté n'a été spécifié.

NOTE 2 – Malgré ce qui précède, l'utilisation de ce type avec une simple contrainte de sous-typage utilisant un paramètre de la syntaxe abstraite (restreinte à un sous-type défini du type «UniversalString») peut constituer un outil puissant et souple pour le traitement des caractères en s'appuyant sur des profils pour déterminer la valeur du paramètre permettant de répondre aux exigences d'un domaine d'intérêt commun. Cependant, l'utilisation de la déclaration «CHARACTER STRING» doit généralement être préférée dans les Recommandations | Normes internationales (voir ci-dessous).

c) les types de chaînes de caractères fournissant une petite collection simple de caractères, spécifiés dans la présente Recommandation | Norme internationale, et destinés à des utilisations particulières; il s'agit des types chaîne numérique «NumericString» et imprimable «PrintableString»;

d) le type «CHARACTER STRING» utilisé avec négociation du jeu de caractères à utiliser (ou l'annonce du jeu utilisé); cette possibilité permet à une application d'utiliser tous les jeux de caractères et codages auxquels des identificateurs d'objets «OBJECT IDENTIFIER» ont été affectés, y compris ceux du *Registre international des jeux de caractères codés à utiliser avec les séquences d'échappement de l'ISO*, de la norme ISO/CEI 7350 et de la norme ISO/CEI 10646-1, et les jeux de caractères et codages privés; (les profils peuvent imposer des spécifications ou des restrictions aux jeux de caractères – les syntaxes abstraites de caractères – à utiliser).

D.2 Les types chaîne universelle «UniversalString», chaîne «UTF8String» et table multilingue «BMPString»

D.2.1 Les types chaîne universelle «UniversalString» et chaîne «UTF8String» contiennent tous les caractères définis dans la norme ISO/CEI 10646-1. Ce jeu est généralement trop grand pour imposer en pratique une contrainte de conformité, et il doit normalement être restreint à des sous-ensembles combinant les jeux de caractères normalisés définis dans l'Annexe A de la norme ISO/CEI 10646-1.

D.2.2 Le type chaîne de table multilingue «BMPString» contient tous les caractères de la grille ISO/CEI 10646-1 (les 64K – 2 premiers caractères). Ce jeu est généralement restreint à une combinaison des jeux de caractères normalisés définis dans l'Annexe A de la norme ISO/CEI 10646-1.

D.2.3 Il existe pour les jeux définis dans l'Annexe A de la norme ISO/CEI 10646-1 des références de type définies dans le module prédéfini «ASN1-CHARACTER-MODULE» (voir l'article 37). Le mécanisme de contrainte de sous-typage permet de définir de nouveaux sous-types de «UniversalString» par combinaison de sous-types existants.

D.2.4 Parmi les références de type définies dans le module «ASN1-CHARACTER-MODULE» et le nom de collection correspondant dans la norme ISO/CEI 10646-1, on peut citer à titre d'exemple:

BasicLatin	BASIC LATIN (Latin de base)
Latin-1Supplement	LATIN-1 SUPPLEMENT (Supplément Latin-1)

LatinExtended-a	LATIN EXTENDED-A (Latin étendu-a)
LatinExtended-b	LATIN EXTENDED-B (Latin étendu-b)
IpaExtensions	IPA EXTENSIONS (Extensions IPA)
SpacingModifierLetters	SPACING MODIFIER LETTERS (Lettres modifiant les espacements)
CombiningDiacriticalMarks	COMBINING DIACRITICAL MARKS (Signes diacritiques de combinaison)

D.2.5 La norme ISO/CEI 10646-1 spécifie trois «niveaux d'application», et impose dans toute utilisation de la norme ISO/CEI 10646-1 de spécifier le niveau de l'application.

Le niveau d'application a trait à l'étendue de la prise en charge des *caractères de combinaison* dans le répertoire de caractères, et par là, en termes ASN.1, il définit un sous-ensemble des types de chaînes de caractères à alphabet restreint «UniversalString» et «BMPString».

Dans le niveau 1 de mise en œuvre, les caractères de combinaison sont interdits, et il existe normalement une correspondance biunivoque entre les caractères abstraits (références de cellules) des chaînes de caractères ASN.1 et les caractères matériels imprimés ou affichés de la chaîne.

Dans le niveau 2 de mise en œuvre, il est possible d'utiliser certains caractères de combinaison (énumérés dans l'Annexe B de la norme ISO/CEI 10646-1), mais d'autres sont interdits.

Dans le niveau 3 de mise en œuvre, il n'y a aucune restriction à l'utilisation des caractères de combinaison.

D.2.6 Un type «BMPString» ou «UniversalString» peut faire l'objet d'une limitation de manière à exclure toutes les fonctions de commande en utilisant comme suit la notation de sous-type:

```
VanillaBMPString ::= BMPString(FROM (ALL EXCEPT ({0,0,0,0}..{0,0,0,31} | {0,0,0,128}..{0,0,0,159})))
```

ou, de manière équivalente:

```
C0 ::= BMPString (FROM ( {0,0,0,0} .. {0,0,0,31} ))          -- fonctions de commande C0
C1 ::= BMPString (FROM ( {0,0,0,128} .. {0,0,0,159} ))       -- fonctions de commande C1
VanillaBMPString ::= BMPString (FROM (ALL EXCEPT (C0 | C1)))
```

D.3 A propos des prescriptions de conformité à la norme ISO/CEI 10646-1

L'utilisation d'un type (ou d'un sous-type de) «UniversalString», «BMPString» ou «UTF8String» dans une définition de type ASN.1 impose d'étudier les prescriptions de conformité de la norme ISO/CEI 10646-1.

Ces prescriptions de conformité imposent aux développeurs d'une certaine norme (disons la norme X) utilisant de tels types ASN.1 de produire une déclaration (dans la déclaration PICS de conformité d'une instance de protocole) précisant le sous-ensemble de caractères de la norme ISO/CEI 10646-1 qu'ils utilisent dans leur instance de la norme X, ainsi que le niveau d'application (prise en charge des caractères de combinaison) de cette instance.

L'utilisation d'un sous-type de «UniversalString», de «UTF8String» ou de «BMPString» dans une spécification impose la prise en charge par l'instance de tous les caractères de la norme ISO/CEI 10646-1 inclus dans ce sous-type ASN.1, et donc que ces caractères (au moins) soient présents dans le sous-ensemble adopté pour l'instance de protocole. De même, il est nécessaire que le niveau de mise en œuvre déclaré soit pris en charge par tous les sous-types ASN.1 ainsi définis.

NOTE – En l'absence de paramètres dans la syntaxe abstraite et de spécifications d'exceptions, une spécification ASN.1 détermine à la fois le jeu (maximal) de caractères pouvant être utilisé en émission et le jeu (minimal) de caractères qui doit pouvoir être traité en réception. L'adoption du jeu ISO/CEI 10646-1 signifie qu'aucun caractère en dehors de ce jeu ne doit être transmis et que tous les caractères de ce jeu sont pris en charge à la réception. Le jeu adopté doit donc correspondre très précisément à l'ensemble de tous les caractères autorisés par la spécification ASN.1. Le cas où intervient un paramètre de syntaxe abstraite est discuté ci-dessous.

D.4 Recommandations aux utilisateurs ASN.1 à propos de la conformité à la norme ISO/CEI 10646-1

Les utilisateurs ASN.1 doivent indiquer clairement le jeu de caractères ISO/CEI 10646-1 (et le niveau d'application correspondant) qui devra être adopté par les instances de protocole pour que celles-ci puissent être conformes aux prescriptions de leur norme.

Une bonne façon de le faire est de définir un sous-type ASN.1 du type de «UniversalString», «UTF8String» ou «BMPString» contenant tous les caractères nécessaires à la norme, et d'en restreindre s'il y a lieu le niveau au niveau 1 ou 2. On pourra par exemple désigner ce type par «ISO-10646-String».

EXEMPLE

```
ISO-10646-String ::= BMPString
                    (FROM(Level2 INTERSECTION (BasicLatin UNION HebrewExtended UNION Hiragana)))
                    -- Il s'agit du type qui définit le jeu minimal de caractères dans la collection
                    -- adoptée pour les instances de cette norme.
                    -- Le niveau d'application requis est le niveau 2 au moins.
```

La déclaration PICS contiendrait alors une simple déclaration indiquant que le sous-ensemble de caractères ISO/CEI 10646-1 adopté est le jeu (et le niveau) défini par le type «ISO-10646-String», et ce type (ou un sous-type de) «ISO-10646-String» sera utilisé partout dans la norme où apparaîtront des chaînes de caractères ISO/CEI 10646-1.

EXEMPLE DE DÉCLARATION PICS

Le sous-jeu de caractères ISO/CEI 10646-1 adopté est le sous-ensemble limité constitué de tous les caractères du type ASN.1 «ISO-10646-String» défini dans le module <mettez ici le nom de votre module>, avec un niveau d'application de 2.

EXEMPLE D'UTILISATION DANS UN PROTOCOLE

```
Message ::= SEQUENCE {
    premier-champ ISO-10646-String,
            -- tous les caractères du sous-jeu peuvent être utilisés
    deuxieme-champ      ISO-10646-String (FROM (latinSmallLetterA .. latinSmallLetterZ)),
            -- minuscules latines seulement
    troisieme-champ     ISO-10646-String (FROM (digitZero .. digitNine))
            -- chiffres seulement
}
```

D.5 Sous-jeux adoptés comme paramètres de la syntaxe abstraite

La norme ISO/CEI 10646-1 impose de définir *explicitement* le sous-jeu de caractères et le niveau d'application adoptés. Lorsqu'un utilisateur ASN.1 ne souhaite pas imposer de contrainte au jeu de caractères ISO/CEI 10646-1 dans une partie quelconque de la norme qu'il définit, il peut l'exprimer en définissant «ISO-10646-String» (par exemple) comme un sous-type de «UniversalString», «BMPString» ou «UTF8String» avec une contrainte de sous-typage constituée de (ou comportant) un type «SousJeuApplication» qui est laissé comme paramètre de la syntaxe abstraite.

Les utilisateurs de la notation ASN.1 doivent savoir que dans ce cas, un expéditeur conforme peut transmettre à un destinataire conforme des caractères qui ne pourront pas être traités par le destinataire parce qu'ils n'appartiennent pas au sous-jeu ou au niveau d'application (dépendants de l'application) adopté par le destinataire; il est recommandé dans ce cas d'inclure dans la définition du type «ISO-10646-String» une spécification de traitement d'exception.

EXEMPLE

```
ISO-10646-String {UniversalString: SousJeuApplication, NiveauApplication} ::=
                    UniversalString (FROM((SousJeuApplication UNION BasicLatin)
                                INTERSECTION NiveauApplication) !problemeJeuDeCaracteres)
                    -- Le sous-jeu pris dans la norme ISO/CEI 10646-1 comprendra le jeu «BasicLatin»,
                    -- mais pourra également inclure tout caractère spécifié dans «SousJeuApplication»,
                    -- qui est un paramètre de la syntaxe abstraite. «NiveauApplication», qui est aussi un
                    -- paramètre de la syntaxe abstraite, définit le niveau de l'application. Un destinataire
                    -- conforme devra être préparé à recevoir des caractères n'appartenant pas au sous-jeu
                    -- de caractères et au niveau de son application. Dans un tel cas, le système invoquera le
                    -- traitement d'exception spécifié au paragraphe <ajouter ici votre numéro de
                    -- paragraphe> pour «problemeJeuDeCaracteres». A noter qu'un tel traitement ne sera
                    -- jamais invoqué par un destinataire conforme si les caractères utilisés dans la
                    -- communication sont limités au jeu «BasicLatin».

Mon-Type-Chaîne-de-Niveau-2 ::= ISO-10646-String { { HebrewExtended UNION Hiragana }, Level2 }
```

D.6 Le type chaîne de caractères «CHARACTER STRING»

D.6.1 Le type «CHARACTER STRING» offre une totale souplesse de choix du jeu de caractères et de la méthode de codage. Lorsqu'une connexion simple (sans application relais) assure un transfert de données de bout en bout, la négociation des jeux de caractères à utiliser et du codage peut s'effectuer dans le cadre de la définition des contextes de présentation pour les syntaxes abstraites de caractères.

D.6.2 Il est important de comprendre qu'une syntaxe abstraite de caractères est une syntaxe abstraite ordinaire avec des restrictions apportées à ses valeurs possibles (ce sont toutes des chaînes de caractères formées à partir d'une certaine collection de caractères). L'enregistrement de telles syntaxes, ainsi que la négociation d'un contexte de présentation, s'effectue donc de manière normale.

D.6.3 Le codage du type «CHARACTER STRING» permet aussi de déclarer sans négociation les syntaxes abstraite et de transfert utilisées dans des environnements où il convient de procéder de la sorte.

NOTE 1 – Les concepteurs d'applications peuvent interdire ou imposer la négociation au niveau de la couche présentation pour ces champs, ou en laisser le choix à l'expéditeur.

NOTE 2 – Lorsqu'il donne la préférence au mode de déclaration plutôt que de négociation, le concepteur de l'application doit examiner à la fois comment l'expéditeur peut déterminer les syntaxes abstraites (et de transfert) de caractères acceptables pour le destinataire (en ayant recours par exemple au service d'annuaire ou en utilisant la technique des profils), et aussi les mesures qu'un destinataire doit prendre s'il reçoit une valeur de type «CHARACTER STRING» provenant d'une syntaxe abstraite de caractères qu'il ne prend pas en charge.

D.6.4 Si la négociation est utilisée, le concepteur de la couche application peut contrôler cette négociation en spécifiant les moments où de tels contextes de présentation doivent être établis et en spécifiant le paramètre «données d'utilisateur» des primitives de modification de contexte de présentation «P-ALTER-CONTEXT», ou peut simplement supposer qu'un profil donné aura déterminé la syntaxe abstraite à utiliser, établissant un contexte de présentation pour celle-ci au moment de la connexion de présentation «P-CONNECT».

D.6.5 Les fonctionnalités de gestion du contexte de service de présentation permettent à l'initiateur de la connexion de proposer (dans une primitive «P-CONNECT», ou, sur une liaison établie, dans une primitive «P-ALTER-CONTEXT») une liste de nouvelles syntaxes abstraites (pouvant inclure des syntaxes abstraites de caractères) ou d'éliminer certaines syntaxes abstraites, et permettent au répondeur d'effectuer son choix dans cette liste.

D.6.6 L'initiateur de la connexion peut exprimer ses préférences par l'ordre des syntaxes abstraites dans la liste, ou en utilisant le paramètre données d'utilisateur qui peut être utilisé par le concepteur de l'application pour expliquer pourquoi l'utilisation de cette nouvelle syntaxe abstraite est proposée. Il peut indiquer par exemple que toutes les syntaxes abstraites (de caractères) sont proposées pour être utilisées dans un même but donné, ou que son intention est de permettre le choix d'une seule de ces syntaxes à des fins multiples.

D.6.7 Des syntaxes abstraites de caractères (et les syntaxes de transfert correspondantes) sont définies dans plusieurs Recommandations UIT-T et Normes internationales; des syntaxes abstraites (ou des syntaxes de transfert) de caractères supplémentaires peuvent également être définies par tout organisme habilité à affecter des identificateurs d'objets.

D.6.8 Dans la norme ISO/CEI 10646-1, une syntaxe abstraite de caractères est définie (et des identificateurs d'objets affectés) pour la collection complète de caractères, pour chacune des collections de caractères définies comme sous-ensemble de la collection complète (BASIC LATIN, BASIC SYMBOLS, etc.) et pour chaque combinaison possible des collections de caractères définies. Deux syntaxes de transfert de caractères sont aussi définies pour identifier les différentes options (en particulier les représentations sur 16 bits et sur 32 bits) dans la norme ISO/CEI 10646-1.

Annexe E

Caractéristiques obsolètes

(Cette annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

Un certain nombre de caractéristiques figurant dans les versions antérieures de la présente Recommandation | Norme internationale (à savoir la Rec. X.208 du CCITT | ISO/CEI 8824) ont été remplacées et ne font plus partie de la notation ASN.1. Elles peuvent toutefois se trouver dans certains modules ASN.1 existants. La présente annexe décrit ces caractéristiques et explique comment obtenir les fonctions avec les notations qui les remplacent.

E.1 Utilisation des identificateurs devenus obligatoires

Un type nommé «NamedType» et une valeur nommée «NamedValue» étaient notés auparavant:

NamedType ::= identifier Type | Type | SelectionType
NamedValue ::= identifier Value | Value

Ces notations sont remplacées par:

NamedType ::= identifier Type
NamedValue ::= identifier Value

car les premières pouvaient conduire à des ambiguïtés syntaxiques.

Des identificateurs peuvent être ajoutés aux notations de types nommés «NamedType» dans les anciennes spécifications ASN.1 sans affecter le codage du type (bien qu'il soit nécessaire ensuite d'apporter des modifications aux notations ASN.1 pour toute utilisation de la notation de valeur correspondante). Une telle modification doit être apportée au titre d'un rapport de défaut ou dans le cadre d'une révision de la Recommandation | Norme internationale concernée.

E.2 Valeur du type choix

Une valeur de type choix était notée auparavant:

ChoiceValue ::= NamedValue

NamedValue ::= identifier Value | Value

Cette notation est remplacée par:

ChoiceValue ::= identifier ":" Value

car la première pouvait conduire à des ambiguïtés syntaxiques.

E.3 Type quelconque (any type)

Le type quelconque était défini dans les versions antérieures de la présente Recommandation | Norme internationale.

Le type quelconque était normalement utilisé pour laisser «en blanc» un élément d'une spécification, à charge de le remplir dans une autre spécification. La notation «AnyType», autorisée comme forme possible de «Type», était spécifiée comme suit:

AnyType ::= ANY | ANY DEFINED BY identifier

Il était vivement recommandé d'utiliser la seconde forme de cette notation. Dans cette forme, qui n'était autorisée que lorsque le type quelconque était l'un des types composant un type ensemble ou séquence, un autre composant de l'ensemble ou de la séquence (celui dont l'identificateur était désigné) indiquait, par sa valeur d'entier ou d'identificateur d'objet (ou un choix de composé de ces types), le type effectif gouvernant le composant de type quelconque. La correspondance entre ces valeurs et des types ASN.1 donnés pouvait être considérée comme un «tableau» faisant partie de la syntaxe abstraite. En l'absence de l'identificateur déclaré par «DEFINED BY», (première forme de la notation du type), la notation ne comportait aucune indication quant à la manière de déterminer le type de champ. Cela aboutissait fréquemment à des spécifications dans lesquelles le «blanc» persistait même à l'étape de mise en œuvre.

Le type quelconque est maintenant remplacé par la possibilité de spécifier des classes d'objets informationnels, puis de se référer aux champs de ces classes depuis l'intérieur des définitions de types (voir la Rec. UIT-T X.681 | ISO/CEI 8824-2). Comme les champs peuvent être définis de manière à accueillir un type ASN.1 arbitraire, la possibilité de laisser des éléments «en blanc» est fondamentalement conservée. Cependant, cette nouvelle caractéristique permet également de spécifier une «contrainte tabulaire» dans laquelle un ensemble d'objets informationnels donné (un ensemble d'objets de la classe considérée) est explicitement cité comme contraignant le type. Cette dernière capacité englobe celle qu'offrait la notation «**ANY DEFINED BY** identifier».

De plus, des utilisations prédéfinies de ces nouvelles capacités sont établies et correspondent à diverses utilisations courantes du type quelconque (voir la Rec. UIT-T X.681 | ISO/CEI 8824-2). Par exemple, une séquence contenant un identificateur d'objet et un type quelconque, souvent utilisée auparavant pour acheminer une valeur arbitraire en même temps qu'une indication de son type, peut maintenant être décrite par:

INSTANCE OF TYPE-QUELCONQUE

où «**TYPE-QUELCONQUE»** se définit ainsi:

TYPE-QUELCONQUE ::= TYPE-IDENTIFIER

Cette notation entraîne le remplacement de «INSTANCE OF TYPE-QUELCONQUE» par un identificateur d'objet de la classe «**TYPE-QUELCONQUE»** accompagné du type associé à cet identificateur. Un exemple d'une telle utilisation est donné au C.2.18.

Des couples particuliers d'identificateurs d'objets et de types sont définis comme les objets informationnels de la classe «**TYPE-QUELCONQUE**» et, si nécessaire, des ensembles particuliers de ceux-ci peuvent être définis et utilisés pour contraindre la structure «**INSTANCE OF»** de façon que seuls les objets de ces ensembles puissent apparaître.

La possibilité de définir des macros était souvent utilisée comme un moyen semi-formel de définir des tableaux d'objets informationnels régissant l'utilisation d'un type «ANY» associé. Elle est aussi remplacée par les nouvelles notations.

E.4 Possibilité de définir des macros

La définition de macros permettait à l'utilisateur de la notation ASN.1 d'étendre cette notation. A titre d'information, la partie de la version précédente de la présente Recommandation | Norme internationale dans laquelle cette possibilité était spécifiée est reproduite à l'Annexe G.

La possibilité de définir des macros était essentiellement utilisée pour définir une notation spécifiant des objets informationnels. Elle est maintenant directement incorporée dans la notation ASN.1 (voir la Rec. UIT-T X.681 | ISO/CEI 8824-2) et n'a plus besoin du caractère très général (et des risques correspondants) d'une notation définie par l'utilisateur.

En outre, la seule autre utilisation des macros semble être la définition d'expressions devant être complétées par des paramètres quelconques avant de devenir des types ASN.1 complètement définis. Cette opération est maintenant assurée par la capacité plus générale de paramétrage (voir la Rec. UIT-T X.683 | ISO/CEI 8824-4).

Annexe F

Annexe didactique traitant du modèle ASN.1 d'extension de type

(Cette annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

F.1 Présentation générale

F.1.1 Un type ASN.1 peut évoluer dans le temps à partir d'un type **racine d'extension** par le biais de séries d'extensions appelées **additions d'extension**.

F.1.2 Un type ASN.1 disponible pour une extension donnée peut être le type racine d'extension ou peut être ce type complété par une ou plusieurs extensions. Tout type ASN.1 qui contient une addition d'extension contient également la totalité des additions d'extension définies précédemment.

F.1.3 Les définitions de type ASN.1 de ces séries sont dites «**apparentées par extension**» (se reporter au paragraphe 3.8.32 pour une définition plus précise de ce terme) et des règles de codage sont nécessaires afin de coder des types apparentés par extension de manière telle que si deux systèmes utilisent deux types qui sont apparentés par extension, une transmission entre les deux systèmes réussira à transmettre les contenus d'information des parties des types apparentés par extension qui sont communes aux deux systèmes. Il est également nécessaire que celles des parties qui ne sont pas communes aux deux systèmes puissent être délimitées et retransmises (éventuellement à un tiers) lors d'une transmission ultérieure, sous condition que la syntaxe de transfert utilisée soit la même.

NOTE – L'émetteur peut utiliser un type qui est plus ancien ou plus récent dans la série d'additions d'extension.

F.1.4 Les séries de types obtenus par addition progressive à un type racine sont appelées **séries d'extensions.** De tels types (y compris les types racine) doivent être repérés par un indicateur syntaxique, afin que les règles de codage puissent prendre les dispositions idoines pour la transmission de types apparentés par extension (qui peuvent nécessiter plus de bits sur la ligne). Cet indicateur constitué de points de suspension «...» est appelé **marqueur d'extension**.

EXEMPLE

Type racine d'extension

```
A ::= SEQUENCE {
    a    INTEGER,
    ...
}
```

première extension

```
A ::= SEQUENCE {
    a    INTEGER,
    ...,
    b    BOOLEAN,
    c    INTEGER
}
```

deuxième extension

```
A ::= SEQUENCE {
    a    INTEGER,
    ...,
    b    BOOLEAN,
    c    INTEGER,
    d    SEQUENCE {
        e    INTEGER,
        ...,
        ...,
        f    IA5String
    }
}
```

troisième extension

```
A ::= SEQUENCE {
    a    INTEGER,
    ...,
    b    BOOLEAN,
    c    INTEGER,
    d    SEQUENCE {
        e    INTEGER,
        ...,
        g    BOOLEAN OPTIONAL,
        h    BMPString,
        ...,
        f    IA5String
    }
}
```

F.1.5 Toutes les additions d'extension sont insérées entre des paires de marqueurs d'extension. Un marqueur d'extension unique est autorisé si le type racine d'extension apparaît comme dernier élément dans le type, auquel cas on fait l'hypothèse qu'un marqueur d'extension correspondant existe immédiatement avant l'accolade fermante du type; toutes les additions d'extension sont alors insérées à la fin du type.

F.1.6 Un type possédant un marqueur d'extension peut être imbriqué au sein d'un type qui n'en possède pas, être imbriqué au sein d'un type dans une racine d'extension ou être imbriqué dans un type d'addition d'extension. Les séries d'extensions sont traitées dans de tels cas d'une manière indépendante et le type imbriqué avec le marqueur d'extension n'a aucun effet sur le type au sein duquel il est imbriqué. Un **seul point d'insertion d'extension** peut exister dans toute expression spécifique; il est situé à la fin du type si un seul marqueur d'extension est utilisé, ou immédiatement avant le deuxième marqueur d'extension si une paire de marqueurs d'extension est utilisée.

F.1.7 Une nouvelle addition d'extension est définie dans la série d'extensions sous la forme d'un **groupe d'additions d'extension** (un ou plusieurs types imbriqués dans une structure «[[» «]]») ou sous la forme d'un type unique ajouté au niveau du point d'insertion d'extension. Dans l'exemple qui suit, la première extension définit un groupe d'additions d'extension pour lequel les éléments «b» et «c» doivent être tous deux présents ou absents dans une valeur de type «A». La deuxième extension définit un type composant unique «d» qui peut être absent dans une valeur de type «A». La troisième extension définit un groupe d'additions d'extension dans lequel l'élément «h» doit être présent dans une valeur de type «A» chaque fois que le groupe d'additions d'extension figure dans une valeur.

EXEMPLE

Type racine d'extension

```
A ::= SEQUENCE {
        a    INTEGER,
        ...
}
```

première extension

```
A ::= SEQUENCE {
        a    INTEGER,
        ...,
        [[
          b  BOOLEAN,
          c  INTEGER
        ]]
}
```

deuxième extension

```
A ::= SEQUENCE {
        a    INTEGER,
        ...,
        [[
          b  BOOLEAN,
          c  INTEGER
        ]],
        d    SEQUENCE {
               e    INTEGER,
               ...,
               ...,
               f    IA5String
             }
}
```

troisième extension

```
A ::= SEQUENCE {
        a    INTEGER,
        ...,
        [[
          b  BOOLEAN,
          c  INTEGER
        ]],
        d    SEQUENCE {
               e    INTEGER,
               ...,
               [[
                 g  BOOLEAN OPTIONAL,
                 h  BMPString
               ]],
               ...,
               f    IA5String
             }
}
```

F.1.8 Bien que la démarche normale consiste à ajouter des extensions au cours du temps, le modèle ASN.1 sous-jacent et les spécifications n'impliquent pas de facteur temps. Deux types sont apparentés par extension si l'un des deux peut être produit à partir de l'autre au moyen d'additions d'extension. Ceci signifie que la seule contrainte est que ce type contienne tous les composants présents dans le premier. Il est possible (mais peu probable) qu'il existe des types qui doivent évoluer dans la direction opposée. Il est même possible qu'un type *démarre* avec un grand nombre d'extensions qui sont supprimées progressivement. La notation ASN.1 et ses règles de codage s'intéressent uniquement au fait que deux spécifications de type sont apparentées par extension ou non. Si elles le sont, **toutes** les règles de codage ASN.1 assureront un interfonctionnement entre leurs utilisateurs.

F.1.9 Nous partons d'un type donné et décidons ensuite si nous souhaitions pouvoir assurer un interfonctionnement avec des versions antérieures lorsque nous devons procéder à des extensions. Si c'est le cas, le marqueur d'extension sera inclus **à cet instant**. Nous pourrons alors ajouter ultérieurement des additions d'extension sans avoir à changer les bits sur la ligne pour des valeurs antérieures et avec des procédures définies pour le traitement de valeurs étendues par les anciens systèmes. Il est toutefois important de noter que l'ajout d'un marqueur d'extension à un type qui n'en possédait pas précédemment (ou la suppression d'un marqueur d'extension) **modifiera** en général les bits sur la ligne et fera obstacle à l'interfonctionnement. De telles modifications nécessitent en général un changement de numéro de version pour tous les protocoles affectés.

F.1.10 Le Tableau F.1 indique les types ASN.1 qui peuvent servir de type de racine d'extension pour une série d'extensions ASN.1, ainsi que la nature de l'addition d'extension unique qui est autorisée pour ce type (les additions d'extension multiples sont effectuées successivement ou ensemble).

F.2 Incidence sur la numérotation des versions, etc.

F.2.1 Lorsqu'une spécification ASN.1 est rééditée avec des modifications de définition de type faites sous la forme de définitions de type apparenté par extension, ces modifications ne nécessitent pas, d'une manière intrinsèque et pour toutes les utilisations, de changement de l'identificateur d'objet du module ou du numéro de version du protocole.

F.2.2 Il est possible, pour d'autres raisons, que de telles modifications s'accompagnent de changements de numéro de version, mais ceci n'est pas exigé.

Tableau F.1 – Additions d'extension

Type de racine d'extension	Nature de l'addition d'extension
ENUMERATED	Ajout d'une énumération supplémentaire unique à la fin de la liste d'énumérations «AdditionalEnumeration» avec une valeur d'énumération supérieure à celle de toute énumération déjà ajoutée.
SEQUENCE et SET	Ajout d'un type unique ou d'un groupe d'additions d'extension à la fin de la liste «ExtensionAdditionList». Les types de composants «ComponentType» qui constituent les additions d'extension (n'appartenant pas à un groupe d'additions d'extension) ne sont pas obligatoirement marqués comme «OPTIONAL» ou «DEFAULT» bien que ce soit souvent le cas.
CHOICE	Ajout d'un type unique «NamedType» à la fin de la liste «ExtensionAdditionAlternativesList».
Notation de contrainte	Ajout d'un élément unique «AdditionalElementSetSpec» à la notation «ElementSetSpecs».

F.2.3 En revanche, l'ajout d'un marqueur d'extension à un type qui n'en possédait pas précédemment, ou l'ajout de composant à un type séquence ou ensemble (par exemple) ne possédant pas de marqueur d'extension crée un nouveau type qui n'est **pas** apparenté par extension avec l'ancien type et le module qui le contient doit recevoir un nouvel identificateur d'objet; un nouveau numéro de version serait également adéquat pour le protocole associé.

F.3 Prescriptions concernant les règles de codage

F.3.1 Une syntaxe abstraite peut être définie comme les valeurs d'un type ASN.1 unique qui est un type extensible. Il contient alors toutes les valeurs qui peuvent être obtenues par ajout et suppression d'additions d'extension. Une telle syntaxe abstraite est appelée une syntaxe abstraite apparentée par extension.

F.3.2 Un ensemble de règles de codage bien formées pour une syntaxe abstraite apparentée par extension satisfait aux prescriptions supplémentaires formulées dans les paragraphes F.3.3 à F.3.5.

NOTE – Toutes les règles de codage ASN.1 satisfont à ces prescriptions.

F.3.3 La définition des procédures de transformation d'une valeur abstraite en un codage effectué à des fins de transfert et des procédures de transformation d'un codage reçu en une valeur abstraite doit tenir compte du fait que l'émetteur et le récepteur utilisant les syntaxes abstraites ne sont pas identiques, mais sont apparentés par des extensions.

F.3.4 Les règles de codage garantiront que lorsque l'émetteur utilise une spécification de type qui est plus ancienne dans la série d'extensions que la spécification utilisée par le récepteur, les valeurs de l'émetteur seront transférées de telle manière que le récepteur puisse déterminer que des additions d'extension ne sont pas présentes.

F.3.5 Les règles de codage garantiront que lorsque l'émetteur utilise une spécification de type qui est plus récente dans la série d'extensions que la spécification utilisée par le récepteur, un transfert de valeurs de ce type à destination du récepteur sera possible.

Annexe G

Récapitulatif de la notation ASN.1

(Cette annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

Les unités lexicales ci-dessous sont définies dans l'article 11:

typereference
identifier
valuereference
modulereference
comment
empty
number
bstring
hstring
cstring
"::="
[[
]]
".."
"..."
"{"
"}"
"<"
","
"."
"("
")"
"["
"]"
"-"
":"
";"
"@"
"|"
"!"
"^"
ABSENT
ABSTRACT-SYNTAX
ALL
APPLICATION
AUTOMATIC
BEGIN
BIT
BMPString
BOOLEAN
BY
CHARACTER
CHOICE
CLASS
COMPONENT
COMPONENTS
CONSTRAINED
DEFAULT
DEFINITIONS
EMBEDDED
END
ENUMERATED
EXCEPT
EXPLICIT
EXPORTS
EXTENSIBILITY
EXTERNAL
FALSE
FROM
GeneralizedTime
GeneralString
GraphicString
IA5String
IDENTIFIER
IMPLICIT
IMPLIED
IMPORTS
INCLUDES
INSTANCE
INTEGER
INTERSECTION
ISO646String
MAX
MIN
MINUS-INFINITY
NULL
NumericString
OBJECT
ObjectDescriptor
OCTET
OF
OPTIONAL
PDV
PLUS-INFINITY
PRESENT
PrintableString
PRIVATE
REAL
SEQUENCE
SET
SIZE
STRING
SYNTAX
T61String
TAGS
TeletexString
TRUE
TYPE-IDENTIFIER
UNION
UNIQUE
UNIVERSAL
UniversalString
UTCTime
UTF8String
VideotexString
VisibleString
WITH

Les productions suivantes sont utilisées dans la présente Recommandation | Norme internationale, avec les items ci-dessus comme symboles terminaux:

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

ModuleIdentifier ::=  modulereference
                       DefinitiveIdentifier

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

DefinitiveObjIdComponentList  ::=
          DefinitiveObjIdComponent     |
          DefinitiveObjIdComponent  DefinitiveObjIdComponentList

DefinitiveObjIdComponent  ::=
          NameForm      |
          DefinitiveNumberForm    |
          DefinitiveNameAndNumberForm

DefinitiveNumberForm    ::= number

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

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

ExtensionDefault ::=
     EXTENSIBILITY IMPLIED | empty

ModuleBody ::=      Exports Imports AssignmentList |
                    empty

Exports ::=         EXPORTS SymbolsExported ";" |
                    empty

SymbolsExported ::=      SymbolList |
                         empty

Imports ::=         IMPORTS SymbolsImported ";" |
                    empty

SymbolsImported ::=      SymbolsFromModuleList |
                         empty

SymbolsFromModuleList ::=
                    SymbolsFromModule |
                    SymbolsFromModuleList  SymbolsFromModule

SymbolsFromModule ::= SymbolList      FROM   GlobalModuleReference

GlobalModuleReference ::= modulereference AssignedIdentifier

AssignedIdentifier ::=      ObjectIdentifierValue |
                            DefinedValue |
                            empty

SymbolList   ::= Symbol | Symbol "," SymbolList

Symbol  ::= Reference | ParameterizedReference

Reference ::=
          typereference          |
          valuereference         |
```

```
            objectclassreference        |
            objectreference       |
            objectsetreference

AssignmentList ::=      Assignment  | AssignmentList Assignment

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

Externaltypereference ::=
                  modulereference
                  "."
                  typereference

Externalvaluereference ::=
                  modulereference
                  "."
                  valuereference

DefinedType ::=
            Externaltypereference |
            typereference |
            ParameterizedType |
            ParameterizedValueSetType

DefinedValue ::=
            Externalvaluereference |
            valuereference |
            ParameterizedValue

AbsoluteReference ::= "@" GlobalModuleReference
            "."
            ItemSpec

ItemSpec ::=
            typereference |
            ItemId "." ComponentId

ItemId ::= ItemSpec

ComponentId ::=
            identifier | number | "*"

TypeAssignment ::=      typereference
                        "::="
                        Type

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

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

ValueSet ::= "{" ElementSetSpecs "}"

Type ::= BuiltinType | ReferencedType | ConstrainedType

BuiltinType ::=
      BitStringType |
      BooleanType |
      CharacterStringType |
      ChoiceType |
      EmbeddedPDVType |
      EnumeratedType |
      ExternalType |
```

```
        InstanceOfType |
        IntegerType |
        NullType |
        ObjectClassFieldType |
        ObjectIdentifierType |
        OctetStringType |
        RealType |
        SequenceType |
        SequenceOfType |
        SetType |
        SetOfType |
        TaggedType

NamedType ::= identifier   Type

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

Value  ::=   BuiltinValue | ReferencedValue

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

ReferencedValue ::=
        DefinedValue |
        ValueFromObject

NamedValue ::= identifier Value

BooleanType  ::=BOOLEAN

BooleanValue::=      TRUE | FALSE

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

NamedNumberList ::=
                NamedNumber        |
                NamedNumberList "," NamedNumber

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

SignedNumber ::= number | "-" number

IntegerValue ::=      SignedNumber | identifier

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

```
Enumerations ::= RootEnumeration |
        RootEnumeration  ","  "..." |
        RootEnumeration  ","  "..."  ","  AdditionalEnumeration

RootEnumeration ::= Enumeration

AdditionalEnumeration ::= Enumeration

Enumeration ::=
        EnumerationItem | EnumerationItem "," Enumeration

EnumerationItem ::=
        identifier | NamedNumber

EnumeratedValue  ::=
        identifier

RealType ::=   REAL

RealValue ::=
        NumericRealValue | SpecialRealValue

NumericRealValue ::= 0 |
        SequenceValue          -- valeur du type séquence associé

SpecialRealValue ::=
        PLUS-INFINITY | MINUS-INFINITY

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

NamedBitList::=         NamedBit | NamedBitList "," NamedBit

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

BitStringValue  ::=     bstring | hstring | "{" IdentifierList "}" | "{"  "}"

IdentifierList  ::=     identifier | IdentifierList "," identifier

OctetStringType ::=     OCTET STRING

OctetStringValue ::=    bstring | hstring

NullType    ::=         NULL

NullValue   ::=NULL


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

ExtensionAndException ::=  "…"  |  "…"  ExceptionSpec

OptionalExtensionMarker ::=  ","  "…"  |  empty

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

RootComponentTypeList  ::=  ComponentTypeList

ExtensionEndMarker  ::=  ","  "…"

ExtensionAdditions  ::=  ","  ExtensionAdditionList  |  empty

ExtensionAdditionList  ::=  ExtensionAddition  |
        ExtensionAdditionList  ","  ExtensionAddition

ExtensionAddition  ::= ComponentType  |  ExtensionAdditionGroup

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

ComponentTypeList  ::=          ComponentType  |
                                ComponentTypeList "," ComponentType
```

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

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

ComponentValueList ::=        NamedValue   |
                        ComponentValueList "," NamedValue

SequenceOfType  ::=     SEQUENCE OF Type

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

ValueList   ::= Value | ValueList "," Value

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

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

SetOfType   ::=     SET OF Type

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

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

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

RootAlternativeTypeList  ::=  AlternativeTypeList

ExtensionAdditionAlternatives  ::=  ","  ExtensionAdditionAlternativesList  |  empty

ExtensionAdditionAlternativesList  ::= ExtensionAdditionAlternative  |
          ExtensionAdditionGroupAlternativesList  ","  ExtensionAdditionAlternative

ExtensionAdditionAlternative  ::=  ExtensionAdditionGroupAlternatives | NamedType

ExtensionAdditionGroupAlternatives  ::=  "[["  AlternativeTypeList  "]]"

AlternativeTypeList ::=             NamedType    |
                                    AlternativeTypeList "," NamedType

ChoiceValue  ::=          identifier ":" Value

SelectionType   ::=       identifier "<" Type

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

Tag     ::=                 "[" Class ClassNumber "]"

ClassNumber  ::= number | DefinedValue

Class   ::=     UNIVERSAL       |
                APPLICATION     |
                PRIVATE         |
                empty

TaggedValue  ::=    Value

EmbeddedPDVType ::=             EMBEDDED PDV

EmbeddedPDVValue ::=            SequenceValue

ExternalType ::= EXTERNAL

ExternalValue ::= SequenceValue

ObjectIdentifierType  ::=       OBJECT IDENTIFIER
```

```
ObjectIdentifierValue ::=       "{" ObjIdComponentList "}"  |
                                "{" DefinedValue ObjIdComponentList "}"

ObjIdComponentList  ::=         ObjIdComponent     |
                                ObjIdComponent ObjIdComponentList

ObjIdComponent  ::=     NameForm            |
                        NumberForm   |
                        NameAndNumberForm

NameForm     ::=        identifier

NumberForm     ::=      number | DefinedValue

NameAndNumberForm ::=           identifier "(" NumberForm ")"

CharacterStringType ::= RestrictedCharacterStringType | UnrestrictedCharacterStringType

RestrictedCharacterStringType ::= BMPString     |

                        GeneralString               |
                        GraphicString               |
                        IA5String                   |
                        ISO646String                |
                        NumericString               |
                        PrintableString             |
                        TeletexString               |
                        T61String                   |
                        UniversalString             |
                        UTF8String                  |
                        VideotexString              |
                        VisibleString

RestrictedCharacterStringValue ::= cstring | CharacterStringList | Quadruple | Tuple

CharacterStringList  ::=  "{" CharSyms "}"
CharSyms  ::=  CharsDefn | CharSyms "," CharsDefn
CharsDefn  ::=  cstring | Quadruple | Tuple | DefinedValue

Quadruple       ::= "{"  Group  ","  Plane  ","  Row  ","  Cell "}"
Group           ::= number
Plane           ::= number
Row             ::= number
Cell            ::= number

Tuple ::= "{" TableColumn "," TableRow "}"
TableColumn ::= number
TableRow ::= number

UnrestrictedCharacterStringType ::= CHARACTER STRING

CharacterStringValue ::= RestrictedCharacterStringValue | UnrestrictedCharacterStringValue

UnrestrictedCharacterStringValue ::= SequenceValue

UsefulType    ::= typereference
```

Les types de chaîne de caractères ci-dessous sont définis au 36.1:

```
NumericStringVisibleString

PrintableString         ISO646String

TeletexString           IA5String

T61String               GraphicString

VideotexString          GeneralString

UniversalString         BMPString
```

Les types utiles ci-dessous sont définis dans les articles 41 à 43:

```
GeneralizedTime

UTCTime

ObjectDescriptor
```

Les productions ci-dessous sont utilisées dans les articles 44 à 48:

```
ConstrainedType ::=
        Type   Constraint |
        TypeWithConstraint

TypeWithConstraint ::=
        SET Constraint OF Type |
        SET SizeConstraint OF Type |
        SEQUENCE Constraint OF Type |
        SEQUENCE SizeConstraint OF Type

Constraint ::= "(" ConstraintSpec  ExceptionSpec ")"

ConstraintSpec ::=
        SubtypeConstraint |
        GeneralConstraint

ExceptionSpec ::= "!"  ExceptionIdentification | empty

ExceptionIdentification ::= SignedNumber    |
                    DefinedValue   |
                    Type ":" Value

SubtypeConstraint ::= ElementSetSpecs

ElementSetSpecs ::=
            RootElementSetSpec |
            RootElementSetSpec  ","   "..."  |
            "..."  ","  AdditionalElementSetSpec |
            RootElementSetSpec  ","  "..."  ","  AdditionalElementSetSpec

RootElementSetSpec ::= ElementSetSpec

AdditionalElementSetSpec ::= ElementSetSpec

ElementSetSpec ::= Unions | ALL  Exclusions

Unions ::= Intersections |
            UElems UnionMark Intersections

UElems ::= Unions

Intersections ::= IntersectionElements |
            IElems IntersectionMark IntersectionElements

IElems ::= Intersections

IntersectionElements ::= Elements | Elems Exclusions

Elems ::= Elements

Exclusions ::= EXCEPT Elements

UnionMark  ::=  "|" |  UNION

IntersectionMark  ::=  "^"     |INTERSECTION

Elements  ::=
        SubtypeElements |
        ObjectSetElements |
        "(" ElementSetSpec ")"

SubtypeElements ::=
        SingleValue                |
        ContainedSubtype           |
        ValueRange                 |
        PermittedAlphabet          |
        SizeConstraint             |
        TypeConstraint             |
        InnerTypeConstraints

SingleValue    ::=  Value
```

```
ContainedSubtype ::= Includes Type

Includes ::= INCLUDES | empty

ValueRange ::= LowerEndpoint ".." UpperEndpoint

LowerEndpoint  ::=  LowerEndValue | LowerEndValue "<"

UpperEndpoint  ::=  UpperEndValue | "<" UpperEndValue

LowerEndValue ::= Value | MIN

UpperEndValue ::= Value | MAX

SizeConstraint ::= SIZE Constraint

PermittedAlphabet ::= FROM Constraint

TypeConstraint ::= Type

InnerTypeConstraints ::=
      WITH COMPONENT SingleTypeConstraint  |
      WITH COMPONENTS MultipleTypeConstraints

SingleTypeConstraint::= Constraint

MultipleTypeConstraints ::= FullSpecification | PartialSpecification

FullSpecification   ::= "{" TypeConstraints "}"

PartialSpecification ::= "{"  "..."   ","   TypeConstraints "}"

TypeConstraints ::=
      NamedConstraint |
      NamedConstraint "," TypeConstraints

NamedConstraint ::=
      identifier ComponentConstraint

ComponentConstraint ::= ValueConstraint PresenceConstraint

ValueConstraint ::=  Constraint | empty

PresenceConstraint  ::= PRESENT | ABSENT | OPTIONAL | empty
```

SÉRIES DES RECOMMANDATIONS UIT-T

Série A	Organisation du travail de l'UIT-T
Série B	Moyens d'expression: définitions, symboles, classification
Série C	Statistiques générales des télécommunications
Série D	Principes généraux de tarification
Série E	Exploitation générale du réseau, service téléphonique, exploitation des services et facteurs humains
Série F	Services de télécommunication non téléphoniques
Série G	Systèmes et supports de transmission, systèmes et réseaux numériques
Série H	Systèmes audiovisuels et multimédias
Série I	Réseau numérique à intégration de services
Série J	Transmission des signaux radiophoniques, télévisuels et autres signaux multimédias
Série K	Protection contre les perturbations
Série L	Construction, installation et protection des câbles et autres éléments des installations extérieures
Série M	RGT et maintenance des réseaux: systèmes de transmission, de télégraphie, de télécopie, circuits téléphoniques et circuits loués internationaux
Série N	Maintenance: circuits internationaux de transmission radiophonique et télévisuelle
Série O	Spécifications des appareils de mesure
Série P	Qualité de transmission téléphonique, installations téléphoniques et réseaux locaux
Série Q	Commutation et signalisation
Série R	Transmission télégraphique
Série S	Equipements terminaux de télégraphie
Série T	Terminaux des services télématiques
Série U	Commutation télégraphique
Série V	Communications de données sur le réseau téléphonique
Série X	**Réseaux pour données et communication entre systèmes ouverts**
Série Y	Infrastructure mondiale de l'information
Série Z	Langages et aspects informatiques généraux des systèmes de télécommunication