**Remplacée par une version plus récente**

UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

# CCITT X.292

COMITÉ CONSULTATIF INTERNATIONAL TÉLÉGRAPHIQUE ET TÉLÉPHONIQUE

(09/92)

**RÉSEAUX DE COMMUNICATIONS DE DONNÉES**

# CADRE ET MÉTHODOLOGIE DES TESTS DE CONFORMITÉ OSI POUR LES RECOMMANDATIONS SUR LES PROTOCOLES POUR LES APPLICATIONS DU CCITT

## Notation combinée arborescente et tabulaire

Remplacée par une version plus récente

**Recommandation X.292**

# Remplacée par une version plus récente

## AVANT-PROPOS

Le CCITT (Comité consultatif international télégraphique et téléphonique) est l'organe permanent de l'Union internationale des télécommunications (UIT). Il est chargé de l'étude des questions techniques, d'exploitation et de tarification, et émet à ce sujet des Recommandations en vue de la normalisation des télécommunications à l'échelle mondiale.

L'Assemblée plénière du CCITT, qui se réunit tous les quatre ans, détermine les thèmes d'étude et approuve les Recommandations rédigées par ses Commissions d'études. Entre les Assemblées plénières, l'approbation des Recommandations par les membres du CCITT s'effectue selon la procédure définie dans la Résolution n° 2 du CCITT (Melbourne, 1988).

La Recommandation X.292, élaborée par la Commission d'études VII, a été approuvée le 10 septembre 1992 selon la procédure définie dans la Résolution n° 2.

---

REMARQUE

Dans cette Recommandation, le terme «Administration» désigne indifféremment une administration de télécommunication ou une exploitation privée reconnue.

# Remplacée par une version plus récente

**TABLE DES MATIÈRES**

*Page*

**Recommandation X.292**

# CADRE ET MÉTHODOLOGIE DES TESTS DE CONFORMITÉ OSI POUR LES RECOMMANDATIONS SUR LES PROTOCOLES POUR LES APPLICATIONS DU CCITT
## Notation combinée arborescente et tabulaire[1)]

*(1992)*

Le CCITT,

*considérant*

(a) que la Recommandation X.200 définit le modèle de référence pour l'interconnexion des systèmes ouverts (OSI) pour les applications du CCITT;

(b) que l'objectif de l'OSI ne sera complètement réalisé que si les systèmes peuvent être testés pour déterminer leur conformité aux Recommandations pertinentes sur les protocoles OSI;

(c) que des suites de tests normalisées devraient être élaborées pour chaque Recommandation sur les protocoles OSI, de telle façon:

– que les résultats des tests de conformité provenant de services de test différents soient largement acceptés et accueillis avec confiance;

– qu'elles assurent la possibilité d'interfonctionnement des équipements qui ont subi avec succès les tests de conformité normalisés;

(d) qu'il est nécessaire de normaliser le processus des tests de conformité afin d'atteindre un niveau de comparabilité utile et acceptable des résultats d'évaluation de conformité de produits similaires,

*déclare à l'unanimité*

que la notation à utiliser pour les tests génériques et abstraits doit être conforme à la présente Recommandation.

## 0 Introduction

La présente Recommandation définit une notation informelle de tests, appelée notation combinée arborescente et tabulaire (TTCN) (*tree and tabular combined notation*), destinée à être utilisée pour spécifier des suites de tests de conformité abstraites OSI.

Pour élaborer une suite de tests abstraite normalisée, on utilise une notation de test décrivant des tests élémentaires abstraits. Cette notation de test peut être une notation informelle (sans sémantique formellement définie) ou une technique de description formelle (FDT) (*formal description technique*). La notation TTCN est une notation informelle à sémantique expressément définie.

La notation TTCN est destinée à répondre aux objectifs suivants:

a) fournir une notation dans laquelle il soit possible d'exprimer des tests élémentaires abstraits dans des suites de tests normalisées;

b) fournir une notation indépendante des méthodes de test, des couches et des protocoles;

c) fournir une notation reflétant la méthodologie de test abstraite définie dans les Recommandations de la série X.290.

La méthodologie de test abstraite considère une suite de tests comme une hiérarchie allant de la suite de tests complète en passant par les groupes de tests, les tests élémentaires et les modules de tests jusqu'aux événements de tests. La notation TTCN donne une structure de dénomination reflétant la position des tests élémentaires dans cette

---

1) La Recommandation X.292 et la norme ISO/CEI 9646, Technologies de l'information – Interconnexion de systèmes ouverts – Méthodologie générale et procédures – Partie 3: Notation combinée arborescente et tabulaire (TTCN), sont alignées sur le plan technique.

hiérarchie. Elle permet également de structurer les tests élémentaires en une hiérarchie de modules de tests aboutissant à des événements de tests. En notation TTCN, les événements de tests de base sont l'émission et la réception de primitives de service abstraites (ASP) (*abstract service primitives*) et d'unités de données de protocole (PDU) (*protocol data units*) ainsi que les événements de temporisation.

Cette notation est donnée sous deux formes: une forme tabulaire ergonomique, appelée notation graphique TTCN.GR, destinée à être utilisée dans les normes relatives aux suites de tests de conformité OSI; une forme exploitable par machines, appelée notation informatisable TTCN.MP, destinée à être utilisée pour représenter la notation TTCN sous une forme canonique dans des systèmes informatiques et servant de syntaxe de transfert des tests élémentaires entre différents systèmes informatiques. Ces deux formes sont sémantiquement équivalentes.

La présente Recommandation est également publiée par l'ISO sous la norme ISO/CEI 9646, partie 3. L'annexe F de la norme ISO/CEI 9646, partie 3, présente un résumé des différences existant entre les versions du projet de norme internationale et de la norme internationale qui en est issue.

## 1 Domaine d'application

La présente Recommandation définit une notation informelle de tests, appelée notation combinée arborescente et tabulaire (TTCN), applicable aux suites de tests de conformité OSI. Cette notation indépendante des méthodes de test, des couches et des protocoles, reflète la méthodologie de test abstraite définie dans les Recommandations X.290 et X.291.

Elle fournit également des spécifications et des directives concernant l'utilisation de la notation TTCN pour spécifier des suites de tests de conformité indépendantes du système dans une ou plusieurs Recommandations OSI. Cette notation est donnée sous deux formes: la première, ergonomique, est applicable à l'élaboration de Recommandations traitant des suites de tests de conformité pour les protocoles OSI; la seconde, informatisable, s'applique au traitement et aux échanges informatiques.

La présente Recommandation s'applique à la spécification des tests élémentaires de conformité qui peuvent s'exprimer de manière abstraite en termes de contrôle et d'observation d'unités de données de protocole et de primitives de service abstraites. Cependant, certains protocoles peuvent requérir des tests élémentaires qui ne peuvent s'exprimer en ces termes. La spécification de ces tests élémentaires n'entre pas dans le cadre de la présente Recommandation, bien qu'ils puissent avoir à figurer dans une Recommandation sur les suites de tests de conformité.

*Remarque* – Certaines spécifications de conformité statique relatives à un service d'application peuvent, par exemple, nécessiter des techniques de test spécifiques à cette application particulière.

La présente Recommandation indique ce qu'une Recommandation sur des suites de tests peut spécifier à propos de conformité de réalisation d'une suite de tests, notamment à propos de la sémantique opératoire des suites de tests en notation TTCN.

*Remarque* – La Recommandation X.293 fournit des spécifications en matière de réalisation de tests, y compris la dérivation des suites de tests exécutables (ETS).

La présente Recommandation s'applique à la spécification de suites de tests de conformité pour les protocoles des couches 2 à 7 de l'OSI, et notamment aux protocoles exprimés en notation de syntaxe abstraite numéro un (ASN.1) (*abstract syntax notation one*). N'entrent pas dans le cadre de la présente Recommandation:

a) la spécification de suites de tests de conformité pour les protocoles entre entités homologues multiples ou les protocoles de couche physique;

b) les rapports entre la notation TTCN et les techniques de description formelle;

c) la spécification des tests élémentaires faisant intervenir simultanément des descriptions comportementales concurrentes;

*Remarque* – L'utilisation d'arbres parallèles et leur synchronisation seront traitées dans un prochain amendement à la présente Recommandation.

d) la réalisation de suites de tests exécutables (ETS) (*executable test suites*) à partir des suites de tests abstraites.

## 2 Références

- Rec. X.200 (1988), *Modèle de référence pour l'interconnexion des systèmes ouverts pour les applications du CCITT* (voir également ISO 7498).
- Rec. X.210 (1988), *Conventions relatives à la définition de service des couches de l'interconnexion des systèmes ouverts* (voir également ISO/TR 8509).
- Rec. X.208 (1988), *Spécification de la syntaxe abstraite numéro un (ASN.1)* (voir également ISO/CEI 8824).
- Rec. X.209 (1988), *Spécification des règles de codage pour la notation de syntaxe abstraite numéro un (ASN.1)* (voir également ISO/CEI 8825).
- Rec. X.290 (1992), *Cadre général et méthodologie des tests de conformité OSI pour les Recommandations sur les protocoles pour les applications du CCITT – Concepts généraux* (voir également ISO/CEI 9646, Partie 1).
- Rec. X.291 (1992), *Cadre général et méthodologie des tests de conformité OSI pour les Recommandations sur les protocoles pour les applications du CCITT – Spécification des suites de tests abstraites* (voir également ISO/CEI 9646, Partie 2).
- Rec. X.293 (1992), *Cadre général et méthodologie des tests de conformité OSI pour les Recommandations sur les protocoles pour les applications du CCITT – Réalisation des tests* (voir également ISO/CEI 9646, Partie 4).
- Rec. X.294 (1992), *Cadre général et méthodologie des tests de conformité OSI pour les Recommandations sur les protocoles pour les applications du CCITT – Conditions applicables aux laboratoires de test et aux clients pour le processus d'évaluation de conformité* (voir également ISO/CEI 9646, Partie 5).

  ISO/CEI 646:1991, *Technologies de l'information – Jeu ISO de caractères codés à 7 éléments pour l'échange d'information*.

  ISO/CEI 10646-1:. . .[1)], *Technologies de l'information – Jeu de caractères codés à octets multiples – Partie 1: Architecture et plan multilingue de base*.

## 3 Définitions

### 3.1 *Termes de base tirés de la Recommandation X.290*

Les termes suivants, définis dans la Recommandation X.290, sont utilisés:

a) primitive de service abstraite;

b) méthodologie de test abstraite;

c) test élémentaire abstrait;

d) méthode de test (abstraite);

e) suite de tests abstraite;

f) journal de conformité;

g) suite de tests (de conformité);

h) méthode de test coordonnée;

i) méthode de test répartie;

j) test élémentaire exécutable;

k) erreur de test élémentaire exécutable;

l) suite de tests exécutables;

m) échec (verdict);

---

1) Actuellement à l'état de projet.

n) état de test «au repos»;

o) instance sous test;

p) non concluant (verdict);

q) événement de test invalide;

r) méthode de test locale;

s) testeur inférieur;

t) moyens de tester;

u) succès (verdict);

v) formulaire PICS;

w) formulaire PIXIT;

x) déclaration de conformité d'instance de protocole;

y) informations supplémentaires sur l'instance de protocole destinées au test;

z) point de contrôle et d'observation;

aa) méthode de test à distance;

ab) état de test stable;

ac) suite de tests abstraite normalisée;

ad) condition de conformité statique;

ae) événement de test invalide sur le plan de la syntaxe;

af) système à tester;

ag) corps de test;

ah) test élémentaire;

ai) erreur de test élémentaire;

aj) procédure de coordination de tests;

ak) événement de test;

al) groupe de tests;

am) objectif de groupe de tests;

an) laboratoire de test;

ao) protocole de gestion de tests;

ap) résultat d'un test;

aq) épilogue de test;

ar) préambule (de test);

as) objectif d'un test;

at) réalisation d'un test;

au) réalisateur de tests;

av) module de test;

aw) suite de tests (de conformité);

ax) système de test;

ay) testeur supérieur;

az) verdict (d'un test);

ba) état de test.

3.2 *Termes tirés de la Recommandation X.200*

Les termes suivants, définis dans la Recommandation X.200, sont utilisés:

a) couche application;

b) unité de données de protocole;

c) point d'accès au service;

d) couche session;

e) sous-réseau;

f) syntaxe de transfert;

g) couche transport.

3.3 *Termes tirés de la Recommandation X.210*

Le terme suivant, défini dans la Recommandation X.210, est utilisé:

fournisseur de service

3.4 *Termes tirés de la Recommandation X.208*

Les termes suivants, définis dans la Recommandation X.208, sont utilisés:

a) type chaîne binaire;

b) type chaîne de caractères;

c) type énuméré;

d) type externe;

e) identificateur d'objet;

f) type chaîne d'octets;

g) type réel;

h) type sélection;

i) type séquence;

j) type séquence-de;

k) type ensemble;

l) type ensemble-de;

m) sous-type.

*Remarque* – Là où il y a risque de confusion avec la notation TTCN, ces termes sont précédés du préfixe «ASN.1».

3.5 *Termes tirés de la Recommandation X.209*

Le terme suivant, défini dans la Recommandation X.209, est utilisé:

codage

3.6 *Termes spécifiques à la notation TTCN*

Les définitions suivantes sont utilisées dans la présente Recommandation:

3.6.1 **construction ATTACH**

Déclaration en notation combinée arborescente et tabulaire rattachant un module d'événement à un arbre d'appel.

3.6.2 **contrainte de base**

Spécifie un ensemble de valeurs par défaut pour chaque champ dans une définition du type de primitive de service abstraite ou d'unité de données de protocole.

3.6.3 **type de base**

Type duquel est dérivé un type défini dans une suite de tests.

3.6.4 **ligne comportementale**

Entrée dans une table de comportement dynamique, représentant un événement de test ou une autre déclaration en notation combinée arborescente et tabulaire, éventuellement munie d'une étiquette, d'un verdict, d'une référence aux contraintes et d'un commentaire.

3.6.5 **arbre comportemental**

Spécification d'un ensemble de séquences d'événements de test et d'autres déclarations en notation combinée arborescente et tabulaire.

3.6.6 **entrée muette**

Dans une contrainte compacte modifiée, une entrée muette (laissée en blanc) dans un paramètre ou dans un champ de contrainte traduit le fait que sa valeur est fixée par héritage.

3.6.7 **arbre d'appel**

Arbre comportemental auquel un module de test est rattaché.

3.6.8 **table compacte des contraintes**

Déclaration sur une seule table d'un ensemble de contraintes pour une primitive de service abstraite, une unité de données de protocole ou un type structuré.

3.6.9 **table compacte des tests élémentaires**

Déclaration sur une seule table d'un ensemble de tests élémentaires appartenant à un même groupe de tests.

3.6.10 **partie contraintes**

Partie d'une suite de tests en notation combinée arborescente et tabulaire dans laquelle sont spécifiées les valeurs des paramètres de primitives de service abstraites ou des champs d'unités de données de protocole envoyées à l'instance sous test, ainsi que les conditions imposées aux paramètres de primitives de service abstraites et aux champs d'unités de données de protocole par l'instance sous test.

3.6.11 **référence aux contraintes**

Référence à une contrainte, indiquée dans une ligne comportementale.

3.6.12 **partie déclarative**

Partie d'une suite de tests en notation combinée arborescente et tabulaire se rapportant à la définition et à la déclaration de toutes les composantes non prédéfinies utilisées dans cette suite de tests.

3.6.13 **comportement par défaut**

Evénements et autres déclarations en notation combinée arborescente et tabulaire qui peuvent se produire à un niveau quelconque de l'arbre associé et qui sont indiqués dans le formulaire des comportements par défaut.

3.6.14 **groupe de comportements par défaut**

Ensemble nommé de comportements par défaut.

3.6.15 **référence de groupe de comportements par défaut**

Chemin d'accès spécifiant l'emplacement logique d'un comportement par défaut dans la bibliothèque des comportements par défaut.

3.6.16 **identificateur par défaut**

Nom unique d'un comportement par défaut.

3.6.17 **bibliothèque des comportements par défaut**

Ensemble des comportements par défaut dans une suite de tests.

3.6.18 **référence de comportement par défaut**

Référence à un comportement par défaut dans la bibliothèque des comportements par défaut, faite à partir d'une table de tests élémentaires ou de modules de tests.

3.6.19 **chemin de dérivation**

Identificateur composé d'un identificateur de contrainte de base concaténé avec un ou plusieurs identificateurs de contraintes modifiées séparés par des points et se terminant par un point.

3.6.20 **chaînage dynamique**

Etablissement d'un lien par paramétrage, allant des déclarations de contrainte d'un paramètre de primitive de service abstraite ou d'un champ d'unité de données de protocole vers la déclaration de contrainte d'une autre unité de données de protocole. La référence de contrainte d'une ligne comportementale spécifie quelles unités de données de protocole sont chaînées.

3.6.21 **partie dynamique**

Partie d'une suite de tests en notation combinée arborescente et tabulaire spécifiant les descriptions des comportements dynamiques des tests élémentaires, des modules de test et des comportements par défaut.

3.6.22 **événement d'envoi implicite**

Mécanisme utilisé dans des méthodes de test à distance pour demander à l'instance sous test de lancer une unité de données de protocole ou une primitive de service abstraite particulière.

3.6.23 **niveau d’indentation**

Indique la structure de l'arbre dans une description comportementale. Le niveau d'indentation du texte reflète la structure arborescente d'une description comportementale.

3.6.24 **arbre local**

Arbre comportemental défini dans le même formulaire que son arbre d'appel.

3.6.25 **contrainte modifiée**

Contrainte définie pour une primitive de service abstraite ou une unité de données de protocole ayant déjà une contrainte de base, et correspondant à une modification de cette contrainte de base.

3.6.26 **sémantique opératoire**

Sémantique expliquant l'exécution d'un arbre comportemental en notation combinée arborescente et tabulaire.

3.6.27 **événement sinon**

Mécanisme en notation combinée arborescente et tabulaire permettant de traiter des événements de test imprévus de manière contrôlée.

3.6.28 **partie présentation générale**

Partie d'une suite de tests en notation combinée arborescente et tabulaire relative à la présentation générale de la structure de la suite de tests, la structure de la bibliothèque des modules de test (si elle existe), la structure de la bibliothèque des comportements par défaut (si elle existe) et l'association des expressions de sélection (s'il y en a) aux tests élémentaires et aux groupes de tests. Cette partie fournit également les index des tests élémentaires, des modules de tests et des comportements par défaut.

3.6.29 **résultat préliminaire**

Résultat enregistré avant la fin d'un test élémentaire, indiquant si le verdict établi pour la partie associée du test élémentaire est un succès, un échec ou un verdict non concluant.

3.6.30 **pseudo-événement**

Expression en notation combinée arborescente et tabulaire ou opération de temporisation apparaissant sur une ligne de déclaration dans une description comportementale, sans être associée à un événement.

3.6.31 **événement qualifié**

Evénement associé à une expression booléenne.

3.6.32 **événement de réception**

Réception d'une primitive de service abstraite ou d'une unité de données de protocole en un point de contrôle et d'observation nommé ou implicite.

3.6.33 **arbre racine**

Arbre comportemental principal d'un test élémentaire, intervenant au niveau d'entrée de ce test élémentaire.

3.6.34 **événement d'envoi**

Envoi d'une primitive de service abstraite ou d'une unité de données de protocole vers un point de contrôle et d'observation nommé ou implicite.

3.6.35 **ensemble des propositions SI (ou ensemble des propositions conditionnelles multiples)**

Déclarations en notation combinée arborescente et tabulaire codées au même niveau d'indentation et raccordées au même nœud antécédant. Ces déclarations représentent les événements, pseudo-événements et constructions possibles, devant être examinés au point correspondant dans l'exécution du test élémentaire.

3.6.36 **table simple de contrainte**

Déclaration dans une table simple d'une contrainte pour une primitive de service abstraite ou une unité de données de protocole simple d'un type donné.

3.6.37 **sémantique d'image instantanée**

Modèle sémantique supprimant l'effet temporel sur l'exécution d'un test élémentaire, et défini en termes d'image instantanée de l'environnement du test pendant laquelle cet environnement est gelé pour une période spécifiée.

3.6.38 **valeur spécifique**

Valeur en notation combinée arborescente et tabulaire ne contenant ni mécanisme de correspondance, ni variable non liée.

3.6.39 **chaînage statique**

Etablissement d'un lien allant des déclarations de contrainte d'un paramètre de primitive de service abstraite ou d'un champ d'unité de données de protocole à la déclaration de contrainte d'un autre champ d'unité de données de protocole en prenant explicitement une contrainte pour valeur.

3.6.40 **sémantique statique**

Règles de sémantique limitant l'utilisation de la syntaxe en notation combinée arborescente et tabulaire.

3.6.41 **type structuré**

Collection d'un ou plusieurs paramètres de primitive de service abstraite ou champs d'unité de données de protocole pouvant apparaître dans une ou plusieurs définitions du type d'une primitive de service abstraite ou d'une unité de données de protocole, définie dans une déclaration distincte et pouvant servir à spécifier une partie d'une structure ou d'une sous-structure plate dans cette primitive de service abstraite ou cette unité de données de protocole.

3.6.42 **identificateur de test élémentaire**

Nom unique d'un test élémentaire.

3.6.43 **variable de test élémentaire**

Elément d'un ensemble de variables déclaré globalement pour la suite de tests, mais dont la valeur n'est utilisée que dans un seul test élémentaire.

3.6.44 **référence de groupe d'un test**

Chemin spécifiant l'emplacement logique d'un test élémentaire dans une structure de suite de tests abstraite.

3.6.45 **groupe de modules de test**

Ensemble nommé de modules de test.

3.6.46 **référence de groupe d'un module de test**

Trajet spécifiant l'emplacement logique d'un module de test dans la bibliothèque des modules de test.

3.6.47 **identificateur de module de test**

Nom unique d'un module de test.

3.6.48 **bibliothèque des modules de test**

Ensemble des descriptions des comportements dynamiques des modules de test autres que les modules locaux dans la suite de tests.

3.6.49 **objectif de module de test**

Déclaration informelle de l'objectif que le module de test doit réaliser.

3.6.50 **constante de suite de tests**

Elément d'un ensemble de constantes, *non* dérivé de la déclaration de conformité d'une instance de protocole ou des informations supplémentaires sur l'instance de protocole destinées au test, qui reste inchangé dans toute la suite de tests.

3.6.51 **paramètre de suite de tests**

Elément d'un ensemble de constantes, dérivé de la déclaration de conformité d'une instance de protocole ou des informations supplémentaires sur l'instance de protocole destinées au test, qui paramètre globalement une suite de tests.

3.6.52 **variable de suite de tests**

Elément d'un ensemble de variables déclaré globalement pour la suite de tests et dont la valeur est transmise entre les différents tests élémentaires.

3.6.53 **événement de fin de temporisation**

Evénement utilisé dans un arbre comportemental pour vérifier une fin de temporisation donnée.

3.6.54 **rattachement à un arbre**

Méthode permettant d'indiquer qu'un arbre comportemental spécifié ailleurs (en un autre point du formulaire courant ou comme module de test dans la bibliothèque des modules de test) doit être inclus dans l'arbre comportemental courant.

3.6.55 **en-tête d'arbre**

Identificateur d'arbre local suivi d'une liste optionnelle des paramètres formels de cet arbre.

3.6.56 **identificateur d'arbre**

Nom identifiant un arbre local.

3.6.57 **feuille d'arbre**

Déclaration en notation combinée arborescente et tabulaire dans un arbre comportemental ou un module de test pour laquelle aucun comportement subséquent n'est spécifié.

3.6.58 **nœud en arbre**

Déclaration en notation combinée arborescente et tabulaire simple.

3.6.59 **notation d'arbre**

Notation en notation combinée arborescente et tabulaire utilisée pour représenter les tests élémentaires sous forme d'arbres.

3.6.60 **déclaration en notation combinée arborescente et tabulaire**

Evénement, pseudo-événement ou construction spécifié(e) dans une description comportementale.

3.6.61 **événement de test imprévu**

Evénement de test non identifié comme tel dans les résultats prévus de test de la suite de tests. Il est normalement traité par l'événement SINON.

3.6.62 **événement non qualifié**

Evénement non associé à une expression booléenne.

## 4 Abréviations

### 4.1 *Abréviations définies dans la Recommandation X.290*

Les abréviations suivantes, définies dans le § 4 de la Recommandation X.290, sont utilisées:

ATS Suite de tests abstraite (*absract test suite*)

ASP Primitive de service abstraite (*abstract service primitive*)

ETS Suite de tests exécutables (*executable test suite*)

IUT Instance sous test (*implementation under test*)

LT Testeur inférieur(*lower tester*)

MOT Moyens de tester (*means of testing*)

PCO Point de contrôle et d'observation (*point of control and observation*)

PICS Déclaration de conformité d'une instance de protocole (*protocol implementation conformance statement*)

PIXIT Informations supplémentaires sur l'instance de protocole destinées au test(*protocol implementation extra information for testing*)

SUT Système à tester (*system under test*)

TMP Protocole de gestion de test (*test management protocol*)

TTCN Notation combinée arborescente et tabulaire (*tree and tabular combined notation*)

UT Testeur supérieur(*upper tester*)

### 4.2 *Abréviations définies dans la Recommandation X.291*

Les abréviations suivantes, définies dans le § 4 de la Recommandation X.291, sont utilisées:

CS Méthode de test monocouche coordonnée*coordinated single-layer (test method)*]

DS Méthode de test monocouche répartie [*distributed single-layer (test method)*]

LS Méthode de test monocouche locale [*local single-layer (test method)*]

RS Méthode de test monocouche à distance[*remote single-layer (test method)*]

### 4.3 *Autres abréviations*

Les abréviations suivantes sont également utilisées:

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

BNF Forme de Backus-Naur (étendue, utilisée en notation TTCN) (*Backus-Naur form extented BNF used in TTCN*)

DIS Projet de Norme internationale (*draft international standard*)

FDT Technique de description formelle (*formal description technique*)

FIFO Premier entré premier sorti (*first in first out*)

FTAM Transfert, accès et gestion de fichiers (*file transfer access and management*)

IS Norme internationale(*international standard*)

OSI Interconnexion de systèmes ouverts (*open systems interconnection*)

PDU Unité de données de protocole (*protocol data unit*)

SAP Point d'accès au service (*service access point*)

TCP Procédures de coordination des tests (*test coordination procedure*)

TTCN.GR Notation combinée arborescente et tabulaire, forme graphique (*tree and tabular combined notation, graphical form*)

TTCN.MP Notation combinée arborescente et tabulaire, forme informatisable (*tree and tabular combined notation, machine processable form*)

## 5 Formes syntaxiques de la notation TTCN

La notation TTCN existe sous deux formes:

a) une forme graphique (TTCN.GR), adaptée à la lecture par l'homme;

b) une forme informatisable (TTCN.MP), adaptée à la transmission de descriptions TTCN entre machines, pouvant également convenir à d'autres formes de traitement automatique.

La forme TTCN.GR est définie à l'aide de formulaires tabulaires, la forme TTCN.MP à l'aide de productions syntaxiques utilisant comme délimiteurs des mots clés TTCN spéciaux à la place des éléments de délimitation visuels des formulaires tabulaires (cadres et en-têtes par exemple). Dans les tables en notation TTCN.GR, les entrées sont définies par des productions syntaxiques ne faisant intervenir aucun mot clé en notation TTCN.MP; ces productions sont communes aux notations TTCN.GR et TTCN.MP.

Les productions syntaxiques en notation TTCN.MP sont spécifiées dans l'annexe A. Pour éclaircir la notation TTCN.GR, de nombreuses productions syntaxiques communes aux notations TTCN.GR et TTCN.MP sont incluses dans le corps de la présente Recommandation, où elles sont indiquées par le titre DÉFINITION SYNTAXIQUE. Pour une meilleure lisibilité, certaines productions apparaissent en plusieurs endroits du texte.

Les productions syntaxiques incluses dans ce texte sont censées être des copies exactes des productions correspondantes de l'annexe A; en cas de différence, l'annexe A fait foi.

La description textuelle en notation TTCN.GR est censée correspondre à la syntaxe sous-jacente définie dans les productions syntaxiques en notation TTCN.MP, à l'exception des différences indiquées au § A.5 et des restrictions de sémantique statique spécifiées à l'annexe A (qui sont communes aux notations TTCN.GR et TTCN.MP).

Si une contradiction apparaît entre la syntaxe en notation TTCN.GR et la sémantique statique décrite dans le texte et dans l'annexe A, les règles suivantes s'appliquent:

a) les productions syntaxiques en notation TTCN.MP ont la priorité sur la présentation textuelle et les productions syntaxiques du corps de la présente Recommandation, à l'exception des différences indiquées au § A.5;

b) les restrictions de sémantique statique spécifiées au § A.4 et dans les commentaires de sémantique statique (indiqués par la mention SÉMANTIQUE STATIQUE) des productions syntaxiques du § A.3 restreignent les formes valides en notation TTCN, en indiquant les formes de productions syntaxiques autorisées;

c) les restrictions de sémantique statique spécifiées dans l'annexe A ont priorité sur le corps de la présente Recommandation.

Si une suite ATS est spécifiée en notation TTCN.GR conformément à la présente Recommandation, il n'existe alors qu'une seule représentation correspondante en notation TTCN.MP de cette suite ATS ayant la même syntaxe sous-jacente. Ces deux représentations ont des sémantiques opératoires identiques. Deux représentations différentes d'une suite ATS sont équivalentes si et seulement si elles ont des sémantiques opératoires identiques.

*Remarque* – Si une suite ATS normalisée spécifiée en notation TTCN.GR a une représentation apparemment équivalente en notation TTCN.MP, mais que les interprétations de leurs sémantiques opératoires diffèrent, c'est la sémantique opératoire de la notation TTCN.GR qui prime car c'est elle qui constitue la suite ATS normalisée.

## 6 Conformité

6.1 Pour être conformes à la présente Recommandation les suites ATS répondront aux spécifications des notations TTCN.GR ou TTCN.MP.

*Remarque* – Le § 10 de la Recommandation X.290 explique l'utilisation du terme «conformité» dans les Recommandations de la série X.290.

6.2 Pour être conformes aux spécifications de la notation TTCN.GR les suites ATS répondront aux spécifications syntaxiques de la notation TTCN.GR énoncées dans les § 8 à 15 et dans le § A.4.

6.3 Pour être conformes aux spécifications de la notation TTCN.MP les suites ATS répondront aux spécifications syntaxiques de la notation TTCN.MP énoncées au § A.3.

6.4 Pour être conformes à la présente Recommandation et être sémantiquement valides, les suites ATS répondront aux spécifications de sémantique statique énoncées dans les § 7 à 15 et auront des sémantiques opératoires conformes à la définition de la sémantique opératoire donnée dans l'annexe B.

6.5 Pour être conforme à la présente Recommandation une suite ATS normalisée sera telle que toute instance de cette suite de tests se voulant conforme à cette suite ATS normalisée:

a) ait une sémantique opératoire équivalant à celle de la suite de tests comme défini dans l'annexe B;

b) soit conforme à la Recommandation X.293.

*Remarque* – Si une erreur de sémantique statique ou opératoire est décelée pendant l'exécution du test élémentaire exécutable conforme à la spécification TTCN du test élémentaire abstrait correspondant, un laboratoire d'essai conforme à la Recommandation X.294 consignera une erreur de test élémentaire abstrait ou exécutable, selon l'endroit où l'erreur se produit.

6.6 Si une suite ATS normalisée a atteint l'état de projet de Norme internationale en 1991 ou avant cette date, ou a été approuvée en tant que Recommandation du CCITT pendant la période d'études 1989-1992, peut avoir été déclarée conforme à la présente Recommandation, mais utiliser une ou toutes les caractéristiques de la notation TTCN définies dans le projet de Norme internationale (DIS) qui ont été modifiées par la suite entre le DIS et la Norme internationale (IS). Une telle suite devra faire référence à la présente Recommandation et comporter une description des différences existant entre les caractéristiques en notation TTCN qu'elle utilise et celles qui sont spécifiées dans la présente Recommandation.

## 7 Conventions

7.1 *Introduction*

Les conventions suivantes ont été utilisées pour définir les formulaires des tables TTCN.GR et la grammaire TTCN.MP.

7.2 *Métanotation syntaxique*

Le tableau définit la métanotation utilisée pour spécifier la forme étendue de la grammaire de la forme Backus-Naur pour la notation TTCN (dénotée BNF dans ce qui suit):

TABLEAU 1/X.292

**Métanotation syntaxique de la notation TTCN.MP**

| | |
|---|---|
| ::= | Est par définition |
| \| | Ou |
| [abc] | 0 ou 1 instance d'abc |
| {abc} | 0 ou plusieurs instances d'abc |
| {abc}+ | 1 ou plusieurs instances d'abc |
| (...) | Groupement textuel |
| abc | Symbole non terminal abc |
| **abc** | Symbole terminal abc |
| "abc" | Symbole terminal abc |

*Exemple 1* – Utilisation de la métanotation BNF

```
FormalParList ::= "("FormalPar&Type {SemiColon FormalPar&Type}")"
```

On utilise les conventions suivantes pour les textes des formulaires tabulaires:

a) un texte en caractères gras (**comme ceci**) apparaît mot pour mot dans chaque table réelle d'une suite de tests TTCN;

b) un texte en italique (*comme ceci*) n'apparaît pas mot pour mot dans une suite de tests TTCN. Cette police de caractères sert à indiquer que le texte réel doit remplacer le symbole en italique. Les spécifications syntaxiques concernant le texte réel peuvent être trouvées dans la production BNF correspondante en notation TTCN.MP.

*Exemple 2 – SuiteIdentifier* correspond à la production 3 de l'annexe A.

7.3 *Formulaires tabulaires en notation TTCN.GR*

7.3.1 *Introduction*

La notation TTCN.GR est définie à l'aide de deux types de tables:

a) les tables d'objets TTCN simples (voir le § 7.3.2), utilisées pour définir, déclarer ou décrire un objet TTCN simple, comme une déclaration d'unité PDU ou un comportement dynamique de test élémentaire;

b) les tables d'objets TTCN multiples (voir le § 7.3.3), utilisées pour définir un certain nombre d'objets TTCN du même type dans une seule table, comme des définitions de type simple ou des variables de test élémentaire.

7.3.2 *Tables d'objets TTCN simples*

La disposition générale d'une table pour un objet TTCN simple est indiquée à la figure 1/X.292.

L'en-tête de la table contient des informations générales relatives à l'objet défini par cette table. La première entrée de l'en-tête, appelée *Nom de l'objet (Object Name),* contient un identificateur de l'objet. La deuxième entrée, appelée *Commentaires (Comments)*, contient une description informelle de l'objet. Elle peut être omise.

Le corps de la table comprend une ou plusieurs colonnes portant chacune un titre. La colonne la plus à droite, appelée *Commentaires*, contient des descriptions informelles des composantes de l'objet spécifié dans le corps de la table. Elle n'est pas présente dans tous les formulaires et peut être omise si cette colonne est vide.

Le cadre de bas de page contient un article appelé *Commentaires détaillés (Detailed Comments)*. Il peut être utilisé aux mêmes fins que la colonne Commentaires du corps de la table. Le concepteur de la suite de tests peut utiliser ce cadre de bas de page en association avec la colonne Commentaires, à la place de celle-ci, ou ne pas l'utiliser du tout, auquel cas ce cadre peut être omis.

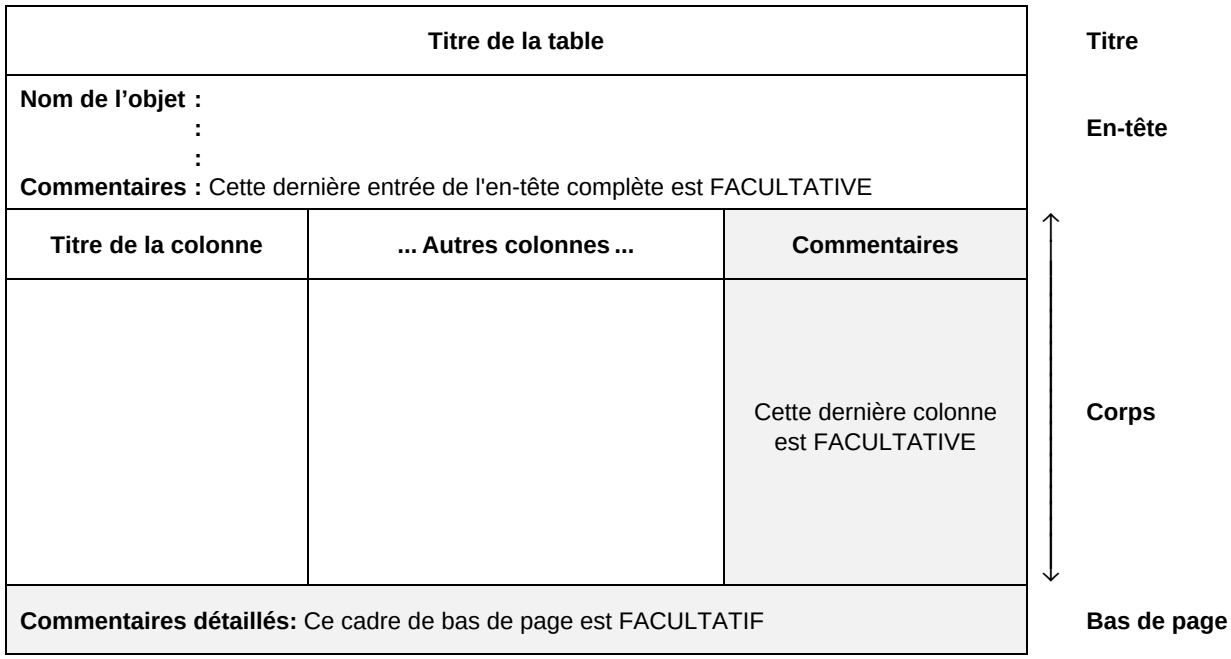



FIGURE 1/X.292

**Disposition générale d'une table déclarative simple**

7.3.3 *Tables d'objets TTCN multiples*

La disposition générale d'une table pour des objets TTCN multiples est indiquée à la figure 2/X.292.

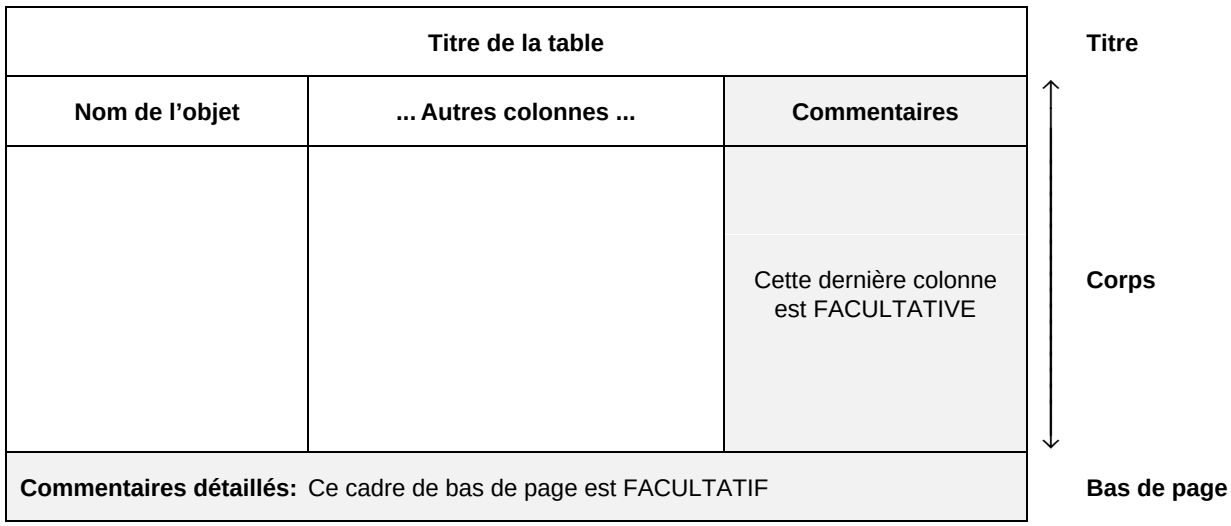



FIGURE 2/X.292

**Disposition générale d'une table déclarative multiple**

Ce type de table ne comporte pas d'en-tête. Le corps du tableau comporte une ou plusieurs colonnes munies chacune d'un titre. La colonne la plus à gauche, appelée *Nom de l'objet*, contient les identificateurs des objets définis ou déclarés dans cette table. La colonne la plus à droite, appelée *Commentaires*, contient des descriptions informelles des objets définis ou déclarés dans la table. Elle n'existe pas dans tous les formulaires. Lorsqu'elle existe, son utilisation est optionnelle pour le concepteur de la suite de tests. Le cadre de bas de page est identique à celui d'une table d'objet simple.

7.3.4 *Autres tables compactes*

Dans certains cas, il est possible de représenter plusieurs tables d'objets simples TTCN sous un format plus compact, c'est-à-dire de présenter plusieurs tables d'objets simples TTCN en une table compacte. Les seules tables pouvant être présentées ainsi sont les suivantes:

– contraintes de primitive ASP (tabulaires et en notation ASN.1);

– contraintes d'unité PDU (tabulaires et en notation ASN.1);

– contraintes du type structuré;

– contraintes du type ASN.1;

– comportements dynamiques de test élémentaire.

Les formats de ces formulaires compacts sont définis dans l'annexe C.

7.3.5 *Spécification des formulaires*

La présente Recommandation spécifie 32 types de tables en notation TTCN.GR et donne une représentation graphique des formulaires correspondants. Ces formulaires sont conformes à la disposition générale indiquée aux § 7.3.2 et 7.3.3. Les colonnes facultatives sont ombrées pour mémoire.

7.4 *Texte libre et texte libre borné*

Certaines entrées des tables permettent l'utilisation de texte libre, c'est-à-dire de caractères provenant de l'un quelconque des jeux de caractères définis dans la Norme ISO 10646, sous réserves des restrictions suivantes:

a) La combinaison de caractères «*/», utilisée en notation TTCN.MP pour indiquer la fin d'une chaîne de texte libre, ne doit pas apparaître dans le texte libre à moins d'être précédée par une barre oblique inverse (\). Une double barre oblique inverse (\\) sera utilisée pour représenter une barre oblique inverse dans le texte.

b) Les combinaisons des caractères «/*» et «*/» ouvrant et fermant les chaînes de texte libre borné BoundedFreeText en notation TTCN.MP seront omises en notation TTCN.GR, c'est-à-dire que si une chaîne de texte libre borné apparaît dans un champ de la table, par exemple dans le champ de l'identificateur complet, ces combinaisons de caractères ne seront pas imprimées.

## 8 Structure des suites de tests TTCN

8.1 *Introduction*

La notation TTCN permet de conférer une structure hiérarchique à une suite de tests conformément au § 8.1 de la Recommandation X.290. Cette structure comprend les composantes suivantes:

a) groupes de tests;

b) tests élémentaires;

c) modules de tests.

Une suite de tests en notation TTCN peut être complètement plate (c'est-à-dire dépourvue de structure); dans ce cas, il n'existe pas de groupes de tests.

La notation TTCN permet d'utiliser des groupes de modules de test et des groupes de comportements par défaut, conceptuellement similaires aux groupes de tests, pour structurer hiérarchiquement les modules de test et les comportements par défaut. Cette structure hiérarchique est facultative.

8.2 *Références de groupe de tests*

La notation TTCN prend en charge une structure de dénomination reflétant le regroupement conceptuel des tests élémentaires. Les groupes de tests peuvent être imbriqués. Les tests élémentaires peuvent également être autonomes (voir la figure 9 au § 8 de la Recommandation X.290). Les références des groupes de tests définissent la structure de la suite de tests; elles obéissent à la syntaxe suivante:

*Définition syntaxique*

```
235  TestGroupReference ::= [SuiteIdentifier "/"] {TestGroupIdentifier "/"}
```

*Exemple 3* – Référence de groupe de transport: TRANSPORT/CLASS0/CONN_ESTAB/

8.3 *Références de groupe de modules de test*

8.3.1 Les modules de test peuvent être explicitement identifiés en notation TTCN et utilisés pour structurer des tests élémentaires ou d'autres modules de test. Les modules de test peuvent aussi être explicites dans la description de comportement d'un test élémentaire. Les modules de test explicites peuvent être spécifiés

– soit localement dans un test élémentaire ou dans une description de comportement d'un module de test;

– soit globalement dans une bibliothèque des modules de test; celle-ci peut être structurée hiérarchiquement en groupes de modules de test.

*Remarque* – Un préambule peut, par exemple, comprendre quelques lignes d'une description de comportement d'un test élémentaire; il est alors implicite. Un préambule peut aussi être spécifié explicitement par sa propre description de comportement. Si un tel préambule explicite est utilisé uniquement dans un test élémentaire, il peut être spécifié localement dans ce même test. S'il est utilisé dans plusieurs tests élémentaires, il doit être spécifié dans la bibliothèque des modules de test.

8.3.2 Les modules de test locaux sont identifiés simplement par un identificateur d'arbre. Les modules de test globaux sont identifiés par un identificateur de module de test. Les modules de test globaux ont aussi une référence de groupe de modules de test qui indique l'emplacement d'un module de test dans la bibliothèque des modules de test. La structure de la bibliothèque des modules de test est indépendante de la structure de la suite de tests. Les références de groupe de modules de test obéissent à la syntaxe suivante:

*Définition syntaxique*

```
248  TestStepGroupReference ::= [SuiteIdentifier "/"] {TestStepGroupIdentifier "/"}
```

*Exemple 4* – Référence de groupe de modules de test de transport:
TRANSPORT/STEP_LIBRARY/CLASS0/CONN_ESTAB/

8.4 *Références de groupe de comportements par défaut*

Les comportements par défaut (lorsqu'ils existent) sont regroupés dans une bibliothèque de comportements par défaut.

Une référence de groupe de comportements par défaut spécifie l'emplacement du comportement par défaut dans la bibliothèque correspondante, qui peut être structurée hiérarchiquement. La structure de cette bibliothèque n'a aucune influence sur celle de la suite de tests elle-même. Les références de groupe de comportements par défaut obéissent à la syntaxe suivante:

*Définition syntaxique*

```
258  DefaultGroupReference ::= [SuiteIdentifier "/"] {DefaultGroupIdentifier "/"}
```

*Exemple 5* – Référence de groupe de comportements par défaut de transport:
TRANSPORT/DEFAULT_LIBRARY/CLASS0/

8.5 *Composantes d'une suite de tests TTCN*

Une suite ATS écrite en TTCN comporte dans l'ordre les quatre sections suivantes:

a) présentation générale de la suite (voir le § 9), contenant les informations nécessaires à la présentation générale et à la bonne compréhension de la suite de tests, par exemple les références de test et une description de son objectif global;

b) partie déclarative (voir le § 10), contenant les définitions ou déclarations de toutes les composantes de la suite de tests (par exemple, des points PCO, des temporisations, des primitives ASP, des unités PDU, et de leurs paramètres ou champs);

c) partie contraintes (voir les § 11, 12, 13), contenant les déclarations des valeurs des primitives ASP, des unités PDU, ainsi que de leurs paramètres utilisés dans la partie dynamique. Ces contraintes sont spécifiées à l'aide de:

   1) tables TTCN, ou

   2) la notation des valeurs en syntaxe ASN.1, ou

   3) des deux éléments ci-dessus à la fois;

d) partie dynamique (voir le § 14), comprenant trois sections contenant les tables spécifiant le comportement du test exprimé principalement en termes d'instances de primitives ASP ou d'unités PDU aux points PCO. Ces sections sont les suivantes:

   1) descriptions de comportements dynamiques de tests élémentaires;

   2) bibliothèque contenant les descriptions de comportements dynamiques de modules de test (s'il y en a);

   3) bibliothèque contenant les descriptions de comportements dynamiques par défaut (s'il y en a).

## 9 Présentation générale de la suite de tests

9.1 *Introduction*

La partie présentation générale d'une suite de tests a pour objet de fournir les informations nécessaires à la présentation générale et à la bonne compréhension de cette suite de tests. Elle comprend les éléments suivants:

a) structure de la suite de tests (voir le § 9.2);

b) index des tests élémentaires (voir le § 9.3);

c) index des modules de tests (voir le § 9.4);

d) index des comportements par défaut (voir le § 9.5).

9.2 *Structure de suite de tests*

La structure de la suite de tests contient l'identification des documents de référence correspondants, la spécification de la structure de la suite de tests, une brève description de sa finalité générale et des références aux critères de sélection des groupes de tests.

Cette section comporte au moins les informations suivantes:

a) le nom de la suite de tests;

b) les références aux Recommandations de base pertinentes;

c) une référence au formulaire de déclaration PICS;

d) une référence au formulaire partiel d'informations PIXIT (voir le § 15 de la Recommandation X.291);

e) une indication de la ou des méthodes de test auxquelles cette suite de tests s'applique, plus, dans le cas des méthodes de tests coordonnées, une référence à l'endroit où le protocole TMP est spécifié;

f) toutes autres informations pouvant aider à la compréhension de la suite de tests, par exemple son mode de dérivation; ces informations doivent apparaître sous forme de commentaires;

g) une liste des groupes de tests de la suite de tests (s'il y en a), comportant, pour chacun de ces groupes, les informations suivantes:

1) la référence de groupe de tests, dans laquelle le premier identificateur peut être le nom de la suite, et chacun des identificateurs suivants représente la suite de l'ordonnancement conceptuel de la suite de tests. Les groupes de tests seront énumérés dans l'ordre d'apparition des tests élémentaires correspondants dans la suite ATS. Ils doivent de plus être classés de telle sorte que tous les groupes d'un groupe simple suivent immédiatement celui-ci. Tous les groupes de tests d'une suite de tests doivent être énumérés;

2) un identificateur facultatif de l'expression de sélection, qui donne comme référence une entrée de la table des définitions des expressions de sélection du test élémentaire, servant à déterminer si les tests élémentaires du groupe s'appliquent à des instances sous test particulières. Cette colonne peut contenir l'identificateur d'une expression de sélection applicable au groupe de tests. Si un identificateur d'expression de sélection est indiqué pour un groupe et si l'expression de sélection mentionnée prend la valeur FALSE (faux), aucun test élémentaire de ce groupe n'est sélectionné pour exécution. Si l'expression de sélection prend la valeur TRUE (vrai), la sélection pour exécution de tests élémentaires appartenant à ce groupe dépendra alors de la valeur prise par les expressions de sélection correspondant aux sous-groupes de ce groupe et aux tests élémentaires individuels. L'omission d'un identificateur d'expression de sélection équivaut à la valeur booléenne TRUE;

3) l'objectif de groupe de tests, qui est une déclaration informelle de la finalité du groupe de tests;

4) un numéro de page, indiquant l'emplacement du premier test élémentaire du groupe dans la suite ATS. Le numéro de page accompagnant chaque référence de groupe de tests dans la table de la structure de la suite de tests est celui de la première description comportementale de test élémentaire de ce groupe.

Ces informations doivent être fournies dans le format de formulaire suivant:

| **Structure de la suite de tests** | | | |
|---|---|---|---|
| **Nom de la suite** : *SuiteIdentifier*<br>**Normes de référence** : *FreeText*<br>**Réf. de déclaration PICS** : *FreeText*<br>**Réf. d'informations PIXIT** : *FreeText*<br>**Méthode(s) de test** : *FreeText*<br>**Commentaires** : *[FreeText]* | | | |
| **Référence de groupe de tests** | **Référence de sélection** | **Objectif du groupe de tests** | **N° page** |
| .<br>.<br>.<br>.<br>*TestGroupReference*<br>.<br>.<br>.<br>. | .<br>.<br>.<br>.<br>*[SelectExpr-Identifier]*<br>.<br>.<br>.<br>. | .<br>.<br>.<br>.<br>*FreeText*<br>.<br>.<br>.<br>. | .<br>.<br>.<br>.<br>*Number*<br>.<br>.<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | | |

**Formulaire 1 – Structure de la suite de tests**

*Définition syntaxique*

```
3    SuiteIdentifier ::= Identifier
235  TestGroupReference ::= [SuiteIdentifier "/"] {TestGroupIdentifier "/"}
83   SelectExprIdentifier ::= Identifier
```

9.3 *Index des tests élémentaires*

L'index des tests élémentaires contient la liste complète des tests élémentaires de la suite ATS. Les informations suivantes seront données pour chacun d'eux:

a) une référence facultative de groupe de tests (si la suite ATS est structurée en groupes de tests), qui définit l'emplacement du test élémentaire dans la structure du groupe de la suite de tests. Si cette référence est omise, le test élémentaire est supposé situé dans le même groupe de tests que le test élémentaire le précédant dans l'index. Les groupes de tests sont énumérés dans l'ordre qu'ils ont dans la suite ATS. Une référence de groupe de tests explicite sera indiquée pour le premier test élémentaire de chaque groupe et pour chaque test élémentaire suivant immédiatement le dernier test élémentaire du groupe;

b) le nom du test élémentaire, c'est-à-dire l'identificateur indiqué dans la table des comportements dynamiques des tests élémentaires. Les tests élémentaires seront énumérés dans l'ordre de leur apparition dans la suite ATS;

c) un identificateur facultatif d'expression de sélection, qui donne comme référence une entrée de la table des définitions des expressions de sélection du test élémentaire, afin de déterminer si le test élémentaire doit être sélectionné pour exécution. Cette colonne peut contenir l'identificateur d'une expression de sélection applicable au test élémentaire. Si un identificateur d'expression de sélection est indiqué et si l'expression de sélection prend la valeur FALSE (faux), ce test élémentaire n'est pas sélectionné pour exécution. Si l'expression de sélection prend la valeur TRUE (vrai), la sélection pour exécution de tests élémentaires dépendra alors de la valeur prise par les expressions de sélection des groupes de tests contenant le test élémentaire. Un test élémentaire est sélectionné si son expression de sélection, ainsi que celle de tous les groupes le contenant, prennent la valeur TRUE (vrai). L'omission d'un identificateur d'expression de sélection équivaut à la valeur booléenne TRUE;

d) une description du test élémentaire, pouvant être un résumé de l'objectif du test;

e) un numéro de page, indiquant l'emplacement du test élémentaire dans la suite ATS. Le numéro de page figurant vis-à-vis de chaque identificateur de test élémentaire dans la table d'index des tests élémentaires est celui de la description comportementale qui lui correspond.

Ces informations sont fournies dans le format illustré dans le formulaire suivant:

| **Index des tests élémentaires** | | | | |
|---|---|---|---|---|
| **Référence du groupe de tests** | **Identificateur du test élémentaire** | **Réf. de sélection** | **Description** | **N° page** |
| . . . . *[TestGroupReference]* . . . . . | . . . . *TestCase Identifier* . . . . . | . . . . *[SelectExpr-Identifier]* . . . . . | . . . . *FreeText* . . . . . | . . . . *Number* . . . . . |
| **Commentaires détaillés:** *[FreeText]* | | | | |

**Formulaire 2 – Index des tests élémentaires**

*Définition syntaxique*

```
235  TestGroupReference ::= [SuiteIdentifier "/"] {TestGroupIdentifier "/"}
233  TestCaseIdentifier ::= Identifier
83   SelectExprIdentifier ::= Identifier
```

9.4 *Index des modules de test*

L'index des modules de test contient la liste complète de tous les modules de test de la suite ATS. Les informations suivantes seront fournies pour chaque module de test:

a) une référence facultative de groupe de modules de test (si la suite ATS est structurée en groupes de modules de test), qui définit l'emplacement du module de test dans la structure de la bibliothèque des modules de test. Si cette référence est omise, le module de test est supposé classé dans le même groupe que le module de test le précédant dans l'index. Les groupes de modules de test sont énumérés dans l'ordre qu'ils ont dans la suite ATS. Une référence explicite de groupe de modules de test sera fournie pour le premier module de test de chaque groupe et pour chaque module de test suivant immédiatement le dernier module de test du groupe; ceci est nécessaire si un groupe de modules de test contient aussi bien un groupe de modules de test et un module de test;

b) le nom du module de test, c'est-à-dire l'identificateur fourni dans la table des comportements dynamiques des modules de test. Les modules de test seront énumérés dans l'ordre de leur apparition dans la suite ATS;

c) une description du module de test, pouvant être un résumé de l'objectif du module de test;

d) un numéro de page, indiquant l'emplacement du module de test dans la suite ATS. Le numéro de page indiqué vis-à-vis de chaque identificateur de module de test dans la table d'index des modules de test est celui de la description comportementale qui lui correspond.

Ces informations seront fournies dans le format illustré dans le formulaire suivant:

| **Index des modules de test** | | | |
|---|---|---|---|
| **Référence du groupe de modules de test** | **Identificateur du module de test** | **Description** | **N° page** |
| . . . . *[TestStepGroupReference]* . . . . | . . . . *TestStep Identifier* . . . . | . . . . *FreeText* . . . . | . . . . *Number* . . . . |
| **Commentaires détaillés:** *[FreeText]* | | | |

**Formulaire 3 – Index des modules de test**

*Définition syntaxique*

```
248  TestStepGroupReference ::= [SuiteIdentifier "/"] {TestStepGroupIdentifier "/"}
246  TestStepIdentifier ::= Identifier
```

9.5 *Index des comportements par défaut*

L'index des comportements par défaut contient la liste complète de tous les comportements par défaut de la suite ATS. Les informations suivantes seront fournies pour chaque comportement par défaut:

a) une référence facultative de groupe de comportements par défaut (si la suite ATS est structurée en groupes de comportements par défaut), qui définit l'emplacement du comportement par défaut dans la structure de la bibliothèque des comportements par défaut. Si cette référence est omise, le comportement par défaut est supposé situé dans le même groupe que le comportement par défaut le précédant dans l'index. Les groupes de comportements par défaut seront énumérés dans l'ordre de leur apparition dans la suite ATS. Une référence explicite de groupe de comportements par défaut sera fournie pour le premier comportement par défaut de chaque groupe et pour chaque comportement par défaut suivant immédiatement le dernier comportement par défaut du groupe;

b) le nom du comportement par défaut, c'est-à-dire l'identificateur indiqué dans la table des comportements dynamiques par défaut. Les comportements par défaut seront énumérés dans l'ordre de leur apparition dans la suite ATS;

c) une description du comportement par défaut, pouvant être un résumé de l'objectif du comportement par défaut;

d) un numéro de page, indiquant l'emplacement du comportement par défaut dans la suite ATS. Le numéro de page indiqué vis-à-vis de chaque identificateur de comportement par défaut dans la table d'index des comportements par défaut est celui de la description comportementale qui lui correspond.

Ces informations seront fournies dans le format illustré dans le formulaire suivant:

| **Index des comportements par défaut** | | | |
|---|---|---|---|
| **Référence de groupe de comportements par défaut** | **Identificateur du comportement par défaut** | **Description** | **N° page** |
| . . . . *[DefaultGroupReference]* . . . . | . . . . *Default Identifier* . . . . | . . . . *FreeText* . . . . | . . . . *Number* . . . . |
| **Commentaires détaillés:** *[FreeText]* | | | |

**Formulaire 4 – Index des comportements par défaut**

*Définition syntaxique*

```
258  DefaultGroupReference ::= [SuiteIdentifier "/"] {DefaultGroupIdentifier "/"}
257  DefaultIdentifier ::= Identifier
```

# 10 Partie déclarative

## 10.1 *Introduction*

La partie déclarative d'une suite ATS a pour objet de définir et de déclarer toutes les composantes utilisées dans cette suite de tests. Les composantes suivantes d'une suite ATS citées en référence dans la partie présentation générale, la partie contraintes et la partie dynamique, devront avoir été déclarées dans la partie déclarative:

a) *définitions:*

1) types de suites de tests (voir le § 10.2.3);

2) opérations des suites de tests (voir le § 10.3.4);

b) *paramétrage et sélection des tests élémentaires:*

1) paramètres de suites de tests (voir le § 10.4);

2) expressions de sélection des tests élémentaires (voir le § 10.5);

c) *déclarations/définitions:*

1) constantes de suites de tests (voir le § 10.6);

2) variables de suites de tests (voir le § 10.7.1);

3) variables de test élémentaire (voir le § 10.7.3);

4) points de contrôle et d'observation (voir le § 10.8);

5) temporisateurs (voir le § 10.9);

6) types de primitives ASP (voir le § 10.10);

7) types d'unités PDU (voir le § 10.11);

8) alias (pseudonymes) (voir le § 10.15).

## 10.2 *Types TTCN*

### 10.2.1 *Introduction*

La notation TTCN prend en charge un certain nombre de types et de mécanismes prédéfinis qui permettent de définir des types spécifiques de suites de tests. Ces types peuvent être utilisés dans toute la suite de tests et référencés lors de la déclaration des paramètres, des constantes et des variables de suites de tests, ainsi que des paramètres de primitives ASP, des champs d'unités PDU, etc.

### 10.2.2 *Types TTCN prédéfinis*

Un certain nombre de types couramment employés sont prédéfinis pour être utilisés en notation TTCN. Tous les types définis en notation ASN.1 et dans ce paragraphe peuvent être référencés, même s'ils n'apparaissent pas dans une définition de type d'une suite de tests. Tous les autres types utilisés dans une suite de tests devront être déclarés dans les définitions des types, des primitives ASP et des unités PDU de la suite de tests. Ils seront référencés par leur nom.

Les types TTCN prédéfinis indiqués ci-dessous sont considérés comme identiques à leurs homologues en notation ASN.1:

a) **type prédéfini INTEGER** (Entier): Type dont les valeurs distinctives sont les entiers relatifs (positifs, négatifs ou nuls).

Les valeurs du type INTEGER sont représentées par un ou plusieurs chiffres, le premier étant différent de zéro, sauf si la valeur est nulle; la valeur nulle est représentée par un seul zéro;

b) **type prédéfini BOOLEAN** (Booléen): Type à deux valeurs distinctives: TRUE (vrai) et FALSE (faux);

c) **type prédéfini BITSTRING** (Chaîne binaire): Type dont les valeurs distinctives sont des séquences ordonnées de zéro, un ou plusieurs bits.

Les valeurs du type BITSTRING sont représentées par un nombre arbitraire (éventuellement zéro) de chiffres binaires (0 et 1), précédés d'une apostrophe ' et suivis des deux caractères 'B;

*Exemple 6* – '01101'B

d) **type prédéfini HEXSTRING** (Chaîne hexadécimale): Type dont les valeurs distinctives sont des séquences ordonnées de zéro, un ou plusieurs chiffres hexadécimaux, chaque chiffre correspondant à une séquence ordonnée de quatre bits.

Les valeurs du type HEXSTRING sont représentées par un nombre arbitraire (éventuellement zéro) de chiffres hexadécimaux:

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

précédés d'une apostrophe ' et suivis des deux caractères 'H; chaque chiffre hexadécimal (type HEX) est utilisé pour dénoter la valeur d'un demi-octet en utilisant une représentaion hexadécimale;

*Exemple 7* – 'AB01D'H

e) **type prédéfini OCTETSTRING** (Chaîne d'octets): Type dont les valeurs distinctives sont des séquences ordonnées comprenant zéro ou un nombre pair positif de chiffres hexadécimaux (chaque paire de chiffres correspondant à une séquence ordonnée de huit bits).

Les valeurs du type OCTETSTRING sont représentées par un nombre pair arbitraire (éventuellement nul) de chiffres hexadécimaux:

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

précédés d'une apostrophe et suivis des deux caractères 'O, chaque chiffre hexadécimal (type HEX) représentant un demi-octet;

*Exemple 8* – 'FF96'O

f) **types prédéfinis CharacterString** (Chaîne de caractères): Types dont les valeurs distinctives sont des séquences de zéro, un ou plusieurs caractères tirés d'un jeu de caractères donné; les types CharacterString énumérés dans le tableau 2/X.292 peuvent être utilisés; ils sont définis dans le § 31 de la Recommandation X.208.

Les valeurs des types CharacterString sont représentées par un nombre arbitraire (éventuellement nul) de caractères tirés du jeu de caractères indiqué par le type CharacterString, précédés et suivis de guillemets ("); un guillemet faisant partie de la chaîne de caractères sera dénoté par un double guillemet ("").

TABLEAU 2/X.292

**Types CharacterString prédéfinis**

| |
|---|
| *NumericString (Chaîne numérique)* |
| *PrintableString (Chaîne imprimable)* |
| *TeletexString (Chaîne télétex) (c'est-à-dire chaîne T.61)* |
| *VideotexString (Chaîne vidéotex)* |
| *VisibleString (Chaîne visible) (c'est-à-dire chaîne ISO646)* |
| *IA5String (Chaîne alphabet international 5)* |
| *GraphicString (Chaîne graphique)* |
| *GeneralString (Chaîne générale)* |

10.2.3 *Définitions des types de suites de tests*

10.2.3.1 *Introduction*

Les définitions des types devant servir de types d'objets de données et de sous-types de primitives ASP et d'unités PDU structurées, etc., peuvent être présentées en format tabulaire ou en notation ASN.1. La récursivité directe ou indirecte n'est pas admise dans les définitions de types de suites de tests lorsque des types y sont cités en référence.

10.2.3.2 *Définition d'un type simple à l'aide de tables*

Pour définir un nouveau type simple, les informations suivantes seront fournies:

a) un nom pour le type considéré;

b) le type de base, qui doit être un type prédéfini ou un type simple. Ce type de base est suivi par la restriction de type, qui prend l'une des formes suivantes:

1) une liste des valeurs distinctives du type de base; ces valeurs comprennent le nouveau type;

2) la spécification d'un intervalle de valeurs de type INTEGER; le nouveau type englobe les valeurs de cet ensemble, bornes comprises. Pour spécifier un intervalle infini, on le bornera à droite ou à gauche par le mot clé INFINITY;

3) la spécification d'une longueur particulière ou d'un intervalle de longueurs pour un type de chaîne prédéfini ou de chaîne de suite de tests; cette(ces) valeur(s) de longueur seront interprétées selon le tableau 4/X.292; pour la limite supérieure, seuls seront utilisés des libellés INTEGER non négatifs ou le mot clé INFINITY.

Ces informations sont données dans le format illustré dans le formulaire suivant:

| **Définitions d'un type simple** | | |
|---|---|---|
| **Nom du type** | **Définition du type** | **Commentaires** |
| .<br>.<br>*SimpleTypeIdentifier*<br>.<br>. | .<br>.<br>*Type&Restriction*<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | |

**Formulaire 5 – Définition d'un type simple**

*Définition syntaxique*

```
27  SimpleTypeIdentifier ::= Identifier
29  Type&Restriction ::= Type [Restriction]
331 Type ::= PredefinedType | Reference Type
332 PredefinedType ::= INTEGER | BOOLEAN | BITSTRING | HEXSTRING | OCTETSTRING |
    CharacterString
333 CharacterString ::= NumericString | PrintableString | TelexString | VideotexString |
    VisibleString | IA5String | GraphicString | GeneralString
334 ReferenceType ::= TS_TypeIdentifier | ASP_Identifier | PDU_Identifier
335 TS_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1_TypeIdentifier
30  Restriction ::= LengthRestriction | IntegerRange | SimpleValueList
31  LengthRestriction ::= SingleTypeLength | RangeTypeLength
32  SingleTypeLength ::= "["Number "]"
33  RangeTypeLength ::= "[" LowerTypeBound To UpperTypebound "]"
34  IntegerRange ::= "(" LowerTypeBound To UpperTypeBound ")"
35  LowerTypeBound ::= [Minus] Number | Minus INFINITY
36  UpperTypeBound ::= [Minus] Number | INFINITY
37  To ::= TO | ".."
38  SimpleValueList ::= "(" [Minus] LiteralValue {Comma [Minus] LiteralValue} ")"
```

Lorsqu'un type est défini sous la forme d'un intervalle de valeurs ou de longueurs (pour des chaînes), la borne inférieure sera celle de gauche. Un intervalle de valeurs entières ne sera utilisé que pour le type de base INTEGER ou pour un type dérivé de INTEGER. Dans ce dernier cas, l'intervalle de valeurs entières sera un sous-intervalle de l'ensemble des valeurs définies par le type de base.

Lorsqu'un type est défini par une liste de valeurs, celles-ci doivent appartenir au type de base et former un sous-ensemble strict des valeurs du type de base. Lorsqu'un type est défini par restriction de longueur, l'ensemble de ce type doit former un sous-ensemble strict des valeurs du type de base.

*Exemple 9* – Définition des types simples de suites de tests

| **Définitions de types simples** | | |
|---|---|---|
| **Nom du type** | **Définition du type** | **Commentaires** |
| Transport_classes | INTEGER(0, 1, 2, 3, 4) | Classes pouvant servir à la liaison de couche transport |
| String5 | IA5String[5] | Chaîne de longueur 5 |
| SeqNumbers | INTEGER(0..127) | Tous les nombres de 0 à 127 |
| PositiveNumbers | INTEGER(1..INFINITY) | Tous les entiers positifs |
| String10to20 | IA5String [10 .. 20] | Chaîne de longueur minimale (10 caractères) et de longueur maximale (20 caractères) |

10.2.3.3 *Définition des types structurés à l'aide de tables*

Les types structurés peuvent être définis sous forme de tables utilisables pour la déclaration d'objets structurés comme les sous-types dans des définitions de primitives ASP et d'unités PDU ou dans d'autres types structurés, etc.

Pour chaque type structuré, les informations suivantes seront fournies:

a) son nom, si nécessaire, il s'agira du nom complet, tel qu'il est indiqué dans la Recommandation sur les protocoles pertinente; lorsqu'une abréviation est utilisée, le nom complet devra suivre entre parenthèses;

b) la liste des éléments associés à ce type structuré, comportant, pour chacun de ces éléments, les informations suivantes:

   1) son nom, c'est-à-dire le nom complet, tel qu'il est indiqué dans la Recommandation sur les protocoles pertinente; lorsqu'on utilise une abréviation, le nom complet devra suivre entre parenthèses;

   2) son type et un attribut facultatif, ces éléments pouvant appartenir à un type structuré arbitrairement complexe; les références récursives (directes ou indirectes) ne sont pas admises. Il est possible d'utiliser l'élément facultatif de restriction de longueur pour indiquer les longueurs minimale et maximale d'un élément de type chaîne (voir le § 10.12).

Les éléments des définitions des types structurés sont considérés comme facultatifs, c'est-à-dire que dans certaines instances de ces types, ils peuvent ne pas être tous présents.

Ces informations sont données dans le format illustré dans le formulaire suivant:

| **Définition d'un type structuré** | | |
|---|---|---|
| **Nom du type :** *StructId&FullId*<br>**Commentaires :** *[FreeText]* | | |
| **Nom de l'élément** | **Définition du type** | **Commentaires** |
| .<br>.<br>*ElemId&FullId*<br>.<br>. | .<br>.<br>*Type&Attributes*<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>. |
| **Commentaires détaillés:** *[Free Text]* | | |

**Formulaire 6 – Définition d'un type structuré**

*Définition syntaxique*

```
42  StructId&FullId ::= StructIdentifier [FullIdentifier]
44  StructIdentifier ::= Identifier
48  ElemId&FullId ::= ElemIdentifier [FullIdentifier]
49  ElemIdentifier ::= Identifier
43  FullIdentifier ::= "(" BoundedFreeText ")"
154 Type&Attributes ::= (Type [LengthAttribute]) | PDU
331 Type ::= PredefinedType | ReferenceType
332 PredefinedType ::= INTEGER | BOOLEAN | BITSTRING | HEXSTRING | OCTETSTRING |
    CharacterString
333 CharacterString ::= NumericString | PrintableString | TeletextString | VideotextString |
    VisibleString | IA5String | GraphicString | GeneralString
334 ReferenceType ::= TS_TypeIdentifier | ASP_Identifier | PDU_Identifier
335 TS_TypeIdentifer ::= SimpleTypeIdentifier | StructIdentifier | ASN1_TypeIdentifier
155 LengthAttribute ::= SingleLength | RangeLength
156 SingleLength ::= "[" Bound "]"
157 Bound ::= Number | TS_ParIdentifier | TS_ConstIdentifier
158 RangeLength ::= "[" LowerBound To UpperBound "]"
159 LowerBound ::= Bound
37  To ::= TO | ".."
160 UpperBound ::= Bound | INFINITY
```

10.2.3.4 *Définition de types de suites de tests à l'aide de la notation ASN.1*

Il est possible de spécifier des types de suites de tests à l'aide de la notation ASN.1, en les définissant selon la syntaxe ASN.1 indiquée dans la Recommandation X.208. Les informations suivantes seront fournies pour chaque type ASN.1:

a) son nom, c'est-à-dire le nom complet, tel qu'il est indiqué dans la Recommandation sur les protocoles pertinente; si une abréviation est utilisée, le nom complet suivra entre parenthèses;

b) la définition du type ASN.1, qui respectera la syntaxe définie dans la Recommandation X.208. Le trait d'union (-) ne devra pas être utilisé dans les identificateurs de cette définition; il est possible d'utiliser à la place le caractère de soulignement (_). L'identificateur de type indiqué dans l'en-tête de la table est le nom du premier type défini dans le corps du tableau.

Les types cités en référence dans la définition de type sont définis dans d'autres tables de définition de types ASN.1, par référence dans la table de référence de types ASN.1 ou localement dans le même tableau, selon la première définition de type. Les types définis localement ne seront pas utilisés dans d'autres parties de la suite de tests.

Les définitions de types ASN.1 utilisées en notation TTCN ne feront pas référence avec types externes définis dans la Recommandation X.208. Des commentaires en notation ASN.1 peuvent figurer dans le corps de la table. La colonne Commentaires n'existe pas dans cette table.

Ces informations seront fournies dans le format de formulaire suivant:

| **Définition de type en notation ASN.1** |
|---|
| **Nom du type :** *ASN1_TypeId&FullId*<br>**Commentaires :** *[FreeText]* |
| **Définition du type** |
| *ASN1_Type&LocalTypes* |
| **Commentaires détaillés:** *[FreeText]* |

**Formulaire 7 – Définition de type en notation ASN.1**

*Définition syntaxique*

```
54   ASN1_TypeId&FullId ::= ASN1_TypeIdentifier [FullIdentifier]
55   ASN1_TypeIdentifier ::= Identifier
43   FullIdentifier ::= "(" BoundedFreeText ")"
57   ASN1_Type&LocalTypes ::= ASN1_Type {ASN1_LocalType}
58   ASN1_Type ::= Type
     /* REFERENCE – Où Type est un symbole non-terminal défini dans la Recommandation X.208 */
59   ASN1_LocalType ::= Typeassignment
     /* REFERENCE – Où Typeassignment est un symbole non-terminal défini dans la
      Recommandation X.208 */
```

*Exemple 10* – Définition d'un type de suite de tests en notation ASN.1

| **Définition de type en notation ASN.1** |
|---|
| **Nom du type :** DATE_type<br>**Commentaires :** afin d'illustrer la structure des définitions de types en notation ASN.1 |
| **Définition du type** |

```
SEQUENCE      {
                    jour      DAY_type,
                    mois      MONTH_type,
                    année     YEAR_type
              }

--type local DAY_type
DAY_type ::= INTEGER {first(1), last(31)}


--Les types MONTH_type et YEAR_Type sont définis dans d'autres tables de définition de type ASN.1
```

10.2.3.5 *Définition de types ASN.1 par référence*

Il est possible de spécifier des types par une référence précise à un type ASN.1 défini dans une Recommandation OSI ou à un type ASN.1 défini dans un module ASN.1 rattaché à la suite de tests. Les informations suivantes seront fournies pour chaque type:

a) son nom, qui pourra être utilisé dans toute la suite de tests; il sera spécifié sans identificateur complet FullIdentifier;

b) la référence de type, qui respectera les règles énoncées dans la Recommandation X.208 pour les identificateurs;

c) l'identificateur du module, composé d'une référence à un module respectant les règles énoncées dans la Recommandation X.208 pour les identificateurs, et d'un identificateur d'objet ObjectIdentifier facultatif; ce module sera unique dans le domaine considéré.

Ces informations seront fournies dans le formulaire suivant:

| **Définitions de types ASN.1 par référence** | | | |
|---|---|---|---|
| **Nom du type** | **Référence du type** | **Identificateur du module** | **Commentaires** |
| . . *ASN1_TypeId&FullId* . . | . . *TypeReference* . . | . . *ModuleIdentifier* . . | . . *[FreeText]* . . |
| **Commentaires détaillés:** *[FreeText]* | | | |

**Formulaire 8 – Définitions de types ASN.1 par référence**

*Définition syntaxique*

```
54  ASN1_TypeId&FullId ::= ASN1_TypeIdentifier [FullIdentifier]
55  ASN1_TypeIdentifier ::= Identifier
43  FullIdentifier ::= "(" BoundedFreeText ")"
63  TypeReference ::= typereference
    /* REFERENCE – Où typereference est défini au § 8.2 de la Recommandation X.208 */
65  ModuleIdentifier ::= ModuleIdentifier
    /* REFERENCE – Où ModuleIdentifier est un symbole non-terminal défini dans la
    Recommandation X.208 */
```

Les types importés de modules ASN.1 pouvant contenir des identificateurs, des références de types et des références de valeur respectant les règles énoncées dans la Recommandation X.208 pour les identificateurs, peuvent donc contenir des traits d'union. Pour pouvoir utiliser en notation TTCN les définitions importées, il est donc nécessaire de transformer les traits d'union contenus dans les identificateurs importés en caractères de soulignement. Cette modification s'effectuera au sein du processus d'importation.

*Exemple 11* – La définition suivante de type contenue dans un module ASN.1

```
module-1 DEFINITIONS BEGIN
        Type 1 ::= SEQUENCE {
                champ1 Sub-Type-1
                champ2 BIT STRING {first-bit(0), second-bit(1) }
                }
END
```

peut être importée en TTCN avec:

| Définition des types ASN.1 par référence | | | |
|---|---|---|---|
| **Nom du type** | **Référence du type** | **Identificateur du module** | **Commentaires** |
| Type_1<br>Sub_Type_1 | Type-1<br>Sub-Type-1 | module-1<br>module-1 | |

La définition par référence du Type-1, indiquée ci-dessus, équivaut à la définition suivante:

| Définition de type en notation ASN.1 |
|---|
| **Nom du type :** Type_1<br>**Commentaires :** |
| **Définition du type** |
| SEQUENCE { champ1 Sub_Type_1,<br>champ2 BIT STRING {first_bit(0), second_bit(1)} } |

10.3 *Opérateurs et opérations du TTCN*

10.3.1 *Introduction*

La notation TTCN prend en charge un certain nombre d'opérateurs, opérations et mécanismes prédéfinis permettant de définir des opérations de suite de tests. Ces opérateurs et opérations peuvent être utilisés dans toutes les descriptions et contraintes de comportement dynamique.

10.3.2 *Opérateurs TTCN*

10.3.2.1 *Introduction*

Les opérateurs prédéfinis sont classés en trois catégories:

a) arithmétiques;

b) relationnels;

c) booléens.

Les règles de préséance entre ces opérateurs sont indiquées au tableau 3/X.292. Les parenthèses servent à regrouper des opérandes en expressions, une expression entre parenthèses étant évaluée prioritairement.

Les opérateurs indiqués sur une même ligne du tableau 3/X.292 ont la même préséance. Lorsque plusieurs opérateurs de même préséance apparaissent, ils sont évalués de gauche à droite.

TABLEAU 3/X.292

**Priorité entre les opérateurs**

| | | |
|---|---|---|
| Maximale<br>↓<br>Minimale | Unaires<br><br>Binaires | ( )<br>+ – NOT<br>* / MOD AND<br>+ – OR<br>= < > <> >= <= |

*Définition syntaxique*

```
321  AddOp ::= "+" | "–" | OR
322  MultiplyOp ::= "*" | "/" | MOD | AND
323  UnaryOp ::= "+" | "–" | NOT
324  RelOp ::= "=" | "<" | ">" | "<>" | ">=" | "<="
```

10.3.2.2 *Opérateurs arithmétiques prédéfinis*

Les opérateurs arithmétiques prédéfinis sont:

"+", "–", "*", "/", **MOD**

Ils représentent les opérations d'addition, de soustraction, de multiplication, de division et de modulo. Leurs opérandes sont du type INTEGER (Entiers) (type TTCN et ASN.1 prédéfini) ou d'un type dérivé d'INTEGER (un sous-ensemble des Entiers). Les valeurs nommées de l'ASN.1 ne seront pas utilisées comme opérandes dans des expressions arithmétiques.

Le résultat des opérations arithmétiques est du type INTEGER.

Les mêmes règles s'appliquent également aux opérandes des opérateurs unaires (monadiques) + et –. L'opérateur "–" inverse le signe de l'opérande.

La division (/) d'un entier INTEGER par un autre donne le résultat de la division entière, c'est-à-dire un entier, la partie fractionnaire étant ignorée.

L'opération MOD entre deux entiers a pour résultat le reste de la division du premier entier par le second.

10.3.2.3 *Opérateurs relationnels prédéfinis*

Les opérateurs relationnels prédéfinis sont:

"=","<",">","<>",">=","<="

Ils représentent les relations d'égalité, d'infériorité, de supériorité, d'inégalité, de supériorité ou égalité et d'infériorité ou égalité. Les opérandes d'égalité (=) et d'inégalité (<>) peuvent être d'un type arbitraire. Les deux opérandes doivent être compatibles. Tous les autres opérateurs relationnels auront des opérandes de type entier ou dérivé. Ces opérations ont un résultat de type booléen (BOOLEAN).

Dans les comparaisons de chaînes, les chaînes binaires BITSTRING, héxadécimales HEXSTRING, d'octets OCTETSTRING et toutes les chaînes de caractères CharacterString peuvent comporter les caractères génériques (*) (AnyOrNone = zéro ou un caractère quelconque) et (?) (AnyOne = un caractère quelconque). Dans ce cas, la comparaison s'effectue conformément aux règles de concordance de modèle définies au § 11.6.5.

10.3.2.4 *Opérateurs booléens prédéfinis*

Les opérateurs booléens prédéfinis sont:

**NOT**, **AND**, **OR** (**NON**, **ET**, **OU**)

Ils représentent les opérations de négation, d'intersection logique et de réunion logique. Leurs opérandes sont de type booléen (TTCN, ASN.1 ou prédéfinis). Leur résultat est de type booléen.

L'opérateur logique ET prend la valeur TRUE (vrai) si ses deux opérandes ont la valeur «vrai», la valeur FALSE (faux) dans les autres cas. L'opérateur logique OU prend la valeur «vrai» si l'un au moins de ses opérandes a la valeur «vrai», la valeur «faux» si et seulement si ses deux opérandes ont la valeur «faux». Le NON logique est l'opérateur unaire qui prend la valeur «vrai» si son opérande a la valeur «faux», et la valeur «faux» si son opérande a la valeur «vrai».

### 10.3.3 *Opérations prédéfinies*

#### 10.3.3.1 *Introduction*

Les opérations prédéfinies se classent en deux catégories:

a) opérations de conversion;

b) autres opérations.

Les opérations prédéfinies peuvent être utilisées dans toute suite de tests. Elles n'ont pas besoin d'être explicitement définies par une table de définition d'opérations de suite de tests. Lorsqu'une opération prédéfinie est appelée:

a) le nombre des paramètres effectifs sera le même que celui des paramètres formels;

b) chaque paramètre effectif prendra une valeur correspondant au type du paramètre formel;

c) toutes les variables apparaissant dans la liste des paramètres seront liées.

Chacune des opérations prédéfinies se présente sous la forme suivante:

OPERATION_NAME (FORMAL_PARAMETER_LIST) ⇒ RESULT_TYPE

#### 10.3.3.2 *Opérations prédéfinies de conversion*

10.3.3.2.1 La notation TTCN prend en charge les opérations prédéfinies de conversion de types suivantes:

a) HEX_TO_INT, convertit une chaîne hexadécimale en un entier (HEXTRING en INTEGER);

b) BIT_TO_INT, convertit une chaîne binaire en un entier (BITSTRING en INTEGER);

c) INT_TO_HEX, convertit un entier en une chaîne hexadécimale (INTEGER en HEXSTRING);

d) INT_TO_BIT, convertit un entier en une chaîne binaire (INTEGER en BITSTRING).

Les règles de codage indiquées dans cette section ne s'appliquent que dans le contexte de ces opérations. Il serait erroné de supposer que ces règles s'appliquent en-dehors du domaine de ces opérations en notation TTCN.

10.3.3.2.2 HEX_TO_INT(hexvalue:HEXSTRING) ⇒ INTEGER

Cette opération convertit une valeur hexadécimale en un entier.

A cette fin, la chaîne hexadécimale est interprétée comme un entier positif en base 16, le poids des chiffres allant décroissant de gauche à droite. Les chiffres hexadécimaux de 0 à F représentent respectivement les valeurs décimales de 0 à 15.

10.3.3.2.3 BIT_TO_INT(bitvalue:BITSTRING) ⇒ INTEGER

Cette opération convertit une seule chaîne binaire en un entier.

A cette fin, la chaîne binaire est interprétée comme un entier positif en base 2, le poids des chiffres allant décroissant de gauche à droite. Les bits 0 et 1 représentent respectivement les valeurs décimales 0 et 1.

10.3.3.2.4 INT_TO_HEX(intvalue, slength:INTEGER) ⇒ HEXSTRING

Cette opération convertit un entier simple en une seule chaîne hexadécimale. La longueur de la chaîne résultante est de *slength* chiffres hexadécimaux.

A cette fin, une chaîne HEXSTRING est interprétée comme un entier positif en base 16, le poids des chiffres décroissant de gauche à droite. Les chiffres hexadécimaux de 0 à F représentent respectivement les valeurs décimales de 0 à 15.

Si la valeur résultante comporte moins de chiffres hexadécimaux que le nombre spécifié par le second paramètre, la chaîne HEXSTRING est complétée à gauche par des zéros.

Une erreur de test élémentaire intervient si l'entier *intvalue* est négatif ou si le résultat hexadécimal comporte plus de chiffres que ne le permet le second paramètre.

10.3.3.2.5 INT_TO_BIT(*intvalue, slength*: INTEGER) ⇒ BITSTRING

Cette opération convertit un entier en une chaîne binaire. La longueur de la chaîne résultante est de *slength* chiffres binaires.

A cette fin, une chaîne BITSTRING est interprétée comme un entier positif en base 2, le poids des chiffres décroissant de gauche à droite. Les bits 0 et 1 représentent respectivement les valeurs décimales 0 et 1.

Si la valeur résultante comporte moins de bits que le nombre spécifié par le second paramètre, la chaîne BITSTRING est complétée à gauche par des zéros.

Une erreur de test élémentaire intervient si l'entier *intvalue* est négatif ou si le résultat binaire comporte plus de bits que ne le permet le second paramètre.

10.3.3.3 *Autres opérations prédéfinies*

La notation TTCN comprend également les opérations prédéfinies suivantes:

a) IS_PRESENT (Est_présent);

b) NUMBER_OF_ELEMENTS (Nombre_d'éléments);

c) IS_CHOSEN (Est_choisi);

d) LENGTH_OF (Longueur_de).

10.3.3.3.1 IS_PRESENT(DataObjectReference) ⇒ BOOLEAN

Cette opération ne prend comme argument une référence à un champ dans un objet de données que si ce champ est défini comme facultatif ou s'il a une valeur par défaut DEFAULT. Le champ peut être d'un type quelconque. Cette opération a pour résultat la valeur booléenne vrai (TRUE) si et seulement si la valeur du champ est présente dans l'instance effective de l'objet de données. Dans les autres cas, le résultat a la valeur «faux» (FALSE).

L'argument de cette opération aura le format défini au § 10.3.4.

*Exemple 12* – Utilisation de IS_PRESENT

si l'unité received_PDU est de type ASN.1,

```
SEQUENCE    {    field_1  INTEGER OPTIONAL,
                 field_2 SEQUENCE OF INTEGER

                }
```

l'opération
IS_PRESENT(received_PDU.field_1)
prend la valeur «vrai» (TRUE) si le champ field_1 de l'instance effective de l'unité received_PDU est présent.

10.3.3.3.2 NUMBER_OF_ELEMENTS(DataObjectReference) ⇒ INTEGER

Cette opération renvoie le nombre effectif d'éléments d'un objet de données qui sont des types ASN.1 SEQUENCE OF ou SET OF. Elle ne s'applique pas aux objets de données ou aux champs d'objets de données d'un type autre que les types ASN.1 SEQUENCE OF ou SET OF. L'argument de cette opération aura le format défini au § 10.3.4.

*Exemple 13* – Utilisation de NUMBER_OF_ELEMENTS

si l'unité received_PDU est de type ASN.1,

```
SEQUENCE        {       field_1 INTEGER OPTIONAL,
                        field_2 SEQUENCE OF INTEGER

                }
```

l'opération

NUMBER_OF_ELEMENTS(received_PDU.field_2)

renvoie le nombre d'éléments de la séquence SEQUENCE OF INTEGER contenue dans l'unité received_PDU d'objet effectif de données.

10.3.3.3.3 IS_CHOSEN(DataObjectReference) ⇒ BOOLEAN

Cette opération renvoie la valeur booléenne «vrai» (TRUE) si et seulement si la référence de l'objet de données spécifie la variante du type CHOICE effectivement sélectionnée pour un objet de données particulier. Dans les autres cas, elle a pour résultat la valeur «faux» (FALSE). Cette opération ne s'applique pas à des objets de données ou à des champs d'objets de données d'un type autre que le type ASN.1 CHOICE. L'argument de cette opération aura le format défini au § 10.3.4.

*Exemple 14* – Utilisation de IS_CHOSEN

si l'unité received_PDU est de type ASN.1,

```
CHOICE          {       p1 PDU_type1,
                        p2 PDU_type2,
                        p3 PDU_type3

                }
```

l'opération

IS_CHOSEN(received_PDU.p2)

renvoie la valeur «vrai» (TRUE) si l'instance effective de l'unité received_PDU achemine une unité du type PDU_type2.

10.3.3.3.4 LENGTH_OF (DataObjectReference) ⇒ INTEGER

Cette opération renvoie la longueur effective d'un objet de données de type BITSTRING, HEXSTRING, OCTETSTRING ou CharacterString. Les unités de longueur sont définies pour chaque type de chaîne au tableau 4/X.292. L'argument de cette opération aura le format défini au § 10.3.4.

Cette opération ne s'applique pas aux objets de données ou aux champs d'objets de données autres que ceux de type BITSTRING, HEXSTRING, OCTETSTRING et CharacterString.

*Exemple 15* – Utilisation de LENGTH_OF

Si S = '010'B alors LENGTH_OF(S) renvoie 3

Si S = 'F3'H alors LENGTH_OF(S) renvoie 2

Si S = 'F2'O alors LENGTH_OF(S) renvoie 1

Si S = "EXEMPLE" alors LENGTH_OF(S) renvoie 7

10.3.4 *Définitions des opérations des suites de tests*

Le concepteur d'une suite ATS peut introduire des opérations spécifiques à cette suite de tests. Pour définir une nouvelle opération, il faut fournir les informations suivantes:

a) nom de l'opération;

b) liste des paramètres d'entrée avec leurs types;

   la liste des noms et des types des paramètres formels. Chaque nom de paramètre sera suivi de deux points (":"), puis du nom du type du paramètre.

Lorsque plusieurs paramètres sont du même type, ils pourront être regroupés en sous-listes. Dans une sous-liste, les noms des paramètres sont séparés par une virgule. Le dernier paramètre de la liste est suivi de deux points (":"), puis du nom du type commun.

Lorsqu'une liste comporte plus d'un couple paramètre/type (ou plus d'un couple liste de paramètres/type), les couples seront séparés par des points-virgules (";").

Seuls les types prédéfinis et les types de données définis dans les définitions de type de suite de tests, de type de primitive ASP et de type d'unité PDU pourront servir de types de paramètres formels. Les types points PCO ne seront pas utilisés comme types de paramètre formel.

*Exemple 16* – Listes des paramètres

Les deux manières suivantes de spécifier une liste de paramètres comportant deux paramètres INTEGER et un paramètre BOOLEAN sont équivalentes:

(A:INTEGER; B:INTEGER; C:BOOLEAN)
(A, B:INTEGER; C:BOOLEAN)

c) type du résultat, la mention du type obéira aux règles concernant les types de paramètres énoncées au point b);

d) description de l'opération, comprenant une explication de l'opération, plus au moins un exemple illustrant un appel de la fonction et le résultat correspondant; l'explication commencera par l'énoncé du nom de l'opération, suivi entre parenthèses d'une liste des noms des paramètres de l'opération; tout cela constitue un «canevas» d'appel de l'opération.

Ces informations seront fournies dans le format illustré dans le formulaire suivant:

| **Définition d'une opération de suite de tests** |
|---|
| **Nom de l'opération :** *TS_OpId&ParList*<br>**Type de résultat :** *Type*<br>**Commentaires :** *[FreeText]* |
| **Description** |
| .<br>.<br>*FreeText*<br>.<br>. |
| **Commentaires détaillés**: *[FreeText]* |

**Formulaire 9 – Définition d'une opération de suite de tests**

*Définition syntaxique*

```
69  TS_OpId&ParList ::= TS_OpIdentifier [FormalParList]
331 Type ::= PredefinedType | ReferenceType
332 PredefinedType ::= INTEGER | BOOLEAN | BITSTRING | HEXSTRING | OCTETSTRING |
    CharacterString
333 CharacterString ::= NumericString | PrintableString | TeletexString | VideotexString |
    VisibleString | IA5String | GraphicString | GeneralString
334 ReferenceType ::= TS_TypeIdentifier | ASP_Identifier | PDU_Identifier
335 TS_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1_TypeIdentifier
```

Une opération peut être comparée à une fonction d'un langage de programmation classique. Cependant, les paramètres de l'opération ne sont jamais modifiés par son exécution; de plus, l'opération n'a pas d'effets secondaires (elle n'entraîne, par exemple, aucune modification de variable de suite de tests ou de test élémentaire).

Lorsqu'on appelle une opération de suite de tests,

a) le nombre des paramètres effectifs sera le même que celui des paramètres formels;

b) chaque paramètre effectif prendra une valeur du même type que le paramètre formel correspondant;

c) toutes les variables apparaissant dans la liste de paramètres seront liées.

*Exemple 17* – Définition de l'opération SUBSTR (sous-chaîne)

| **Définition d'une opération de suite de tests** |
|---|
| **Nom de l'opération :** SUBSTR (source:IA5String; start_index, length:INTEGER)<br>**Type du résultat :** IA5String<br>**Commentaires :** |
| **Description** |
| *SUBSTR(source, start_index, length)* est la chaîne de longueur *length* commençant à l'indice *start_index* de la chaîne d'origine *source*<br>Par exemple: SUBSTR("abcde",3,2) = "cd"<br>SUBSTR("abcde",1,3) = "abc" |

10.4 *Déclarations des paramètres de suites de tests*

Cette section a pour objet de déclarer les constantes dérivées de la déclaration PICS et des informations PIXIT, qui servent au paramétrage global de la suite de tests. Ces constantes sont appelées paramètres de la suite de tests et servent de base à la sélection et au paramétrage des tests élémentaires.

La section déclaration fournira pour chaque paramètre de suite de tests les informations suivantes:

a) son nom;

b) son type, il s'agira d'un type prédéfini, d'un type ASN.1, d'un type suite de tests ou d'un type unité PDU;

c) la référence d'entrée PICS ou PIXIT, il s'agit d'une référence à une entrée individuelle PICS ou PIXIT identifiant clairement l'emplacement où se trouve la valeur à utiliser pour cette suite de tests.

Ces informations seront fournies dans le format illustré dans le formulaire suivant:

| **Déclaration de paramètres d'une suite de tests** | | | |
|---|---|---|---|
| **Nom de paramètre** | **Type** | **Réf. PICS/PIXIT** | **Commentaires** |
| .<br>.<br>*TS_ParIdentifier*<br>.<br>. | .<br>.<br>*Type*<br>.<br>. | .<br>.<br>*FreeText*<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | | |

**Formulaire 10 – Déclaration de paramètres d'une suite de tests**

*Définition syntaxique*

```
77  TS_ParIdentifier ::= Identifier
331 Type ::= PredefinedType | ReferenceType
332 PredefinedType ::= INTEGER | BOOLEAN | BITSTRING | HEXSTRING | OCTETSTRING |
    CharacterString
333 CharacterString ::= NumericString | PrintableString | TeletexString | VideotexString |
    VisibleString | IA5String | GraphicString | GeneralString
334 ReferenceType ::= TS_TypeIdentifier | ASP_Identifier | PDU_Identifier
335 TS_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1_TypeIdentifier
```

*Exemple 18* – Déclaration de paramètres d'une suite de tests

| Déclaration de paramètres d'une suite de tests | | | |
|---|---|---|---|
| **Nom du paramètre** | **Type** | **Réf. PICS/PIXIT** | **Commentaires** |
| PAR1<br>PAR2<br>PAR3 | INTEGER<br>INTEGER<br>INTEGER | Question xx du PICS<br>Question yy du PICS<br>Question zz du PIXIT | |

10.5 *Définition des expressions de sélection de tests élémentaires*

Cette section de la suite ATS a pour objet de définir les expressions booléennes à utiliser dans le processus de sélection des tests élémentaires. Elle répondra aux spécifications de la Recommandation X.291.

Une expression booléenne est associée à un ou plusieurs tests élémentaires ou groupes de tests en plaçant son identificateur dans la colonne «référence de sélection des tests élémentaires» de la structure de la suite de tests ou de l'index des tests élémentaires. Une même expression peut être indiquée en référence par plus d'un test élémentaire ou groupe de tests.

Un test élémentaire ne sera exécuté que si l'expression de sélection booléenne prend la valeur «vrai» (TRUE).

Les informations suivantes seront fournies pour chaque expression de sélection d'un test élémentaire:

a) son nom;

b) une expression de sélection, dont le résultat sera de type booléen et dont les termes seront exclusivement des littéraux, des paramètres de suites de tests, des constantes de suites de tests et d'autres identificateurs d'expressions de sélection.

Ces informations seront fournies dans le format illustré dans le formulaire suivant:

| Définition des expressions de sélection de tests élémentaires | | |
|---|---|---|
| **Nom de l'expression** | **Expression de sélection** | **Commentaires** |
| .<br>.<br>*SelectExprIdentifier*<br>.<br>. | .<br>.<br>*SelectionExpression*<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | |

**Formulaire 11 – Définition des expressions de sélection de tests élémentaires**

*Définition syntaxique*

```
83  SelectExprIdentifier ::= Identifier
85  SelectionExpression ::= Expression
```

10.6 *Déclaration des constantes de suites de tests*

Cette section a pour objet de déclarer un ensemble de noms pour des valeurs *non* dérivées de la déclaration PICS ou des informations PIXIT, et qui resteront constantes dans toute la suite de tests.

Pour chaque constante d'une suite de tests, les informations suivantes seront fournies:

a) son nom;

b) son type, il s'agira d'un type prédéfini, d'un type ASN.1, d'un type suite de tests ou d'un type unité PDU;

c) sa valeur, les termes de l'expression de cette valeur ne contiendront ni variables de suite de tests ni variables de tests élémentaires; la valeur prise correspondra au type indiqué dans la colonne type.

Ces informations seront fournies dans le format illustré dans le formulaire suivant:

| **Déclaration des constantes de suites de tests** | | | |
|---|---|---|---|
| **Nom de la constante** | **Type** | **Valeur** | **Commentaires** |
| .<br>.<br>*TS_ConstIdentifier*<br>.<br>. | .<br>.<br>*Type*<br>.<br>. | .<br>.<br>*Declaration Value*<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | | |

**Formulaire 12 – Déclaration des constantes de suites de tests**

*Définition syntaxique*

90 TS_ConstIdentifier ::= Identifier
331 Type ::= PredefinedType | ReferenceType
332 PredefinedType ::= **INTEGER** | **BOOLEAN** | **BITSTRING** | **HEXSTRING** | **OCTETSTRING** | CharacterString
333 CharacterString ::= **NumericString** | **PrintableString** | **TeletexString** | **VideotexString** | **VisibleString** | **IA5String** | **GraphicString** | **GeneralString**
334 ReferenceType ::= TS_TypeIdentifier | ASP_Identifier | PDU_Identifier
335 TS_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1_TypeIdentifier
93 DeclarationValue ::= Expression

*Exemple 19* – Déclaration des constantes de suites de tests

| **Déclaration des constantes de suites de tests** | | | |
|---|---|---|---|
| **Nom de la constante** | **Type** | **Valeur** | **Commentaires** |
| TS_CONST1 | BOOLEAN | TRUE | |
| TS_CONST2 | IA5String | «Chaîne quelconque» | |

## 10.7 *Variables TTCN*

### 10.7.1 *Déclaration des variables de suites de tests*

Une suite de tests peut utiliser un ensemble de variables définies globalement pour cette suite de tests, dont la valeur se transmet dans toute la suite de tests. Ces variables sont appelées variables de la suite de tests.

On utilise une variable de suite de tests chaque fois qu'il est nécessaire de transmettre des informations entre deux tests élémentaires.

Les informations suivantes seront fournies pour chacune des variables déclarées:

a) son nom;

b) son type, il s'agira d'un type prédéfini, d'un type ASN.1, d'un type suite de tests ou d'un type unité PDU;

c) sa valeur initiale (s'il y a lieu), la colonne valeur initiale servira à affecter une valeur initiale à une variable de suite de tests en son point de déclaration; les termes de l'expression de cette valeur ne contiendront ni variables de suite de tests ni variables de test élémentaire; la valeur prise par l'expression sera du type indiqué dans la colonne type. La spécification d'une valeur initiale est facultative.

Ces informations seront fournies dans le format illustré dans le formulaire suivant:

| **Déclaration des variables de suites de tests** | | | |
|---|---|---|---|
| **Nom de la variable** | **Type** | **Valeur** | **Commentaires** |
| .<br>.<br>*TS_VarIdentifier*<br>.<br>. | .<br>.<br>*Type*<br>.<br>. | .<br>.<br>*[DeclarationValue]*<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | | |

**Formulaire 13 – Déclaration des variables de suites de tests**

*Définition syntaxique*

97 TS_VarIdentifier ::= Identifier
331 Type ::= PredefinedType | ReferenceType
332 PredefinedType ::= **INTEGER** | **BOOLEAN** | **BITSTRING** | **HEXSTRING** | **OCTETSTRING** | CharacterString
333 CharacterString ::= **NumericString** | **PrintableString** | **TeletexString** | **VideotexString** | **VisibleString** | **IA5String** | **GraphicString** | **GeneralString**
334 ReferenceType ::= TS_TypeIdentifier | ASP_Identifier | PDU_Identifier
335 TS_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1_TypeIdentifier
93 DeclarationValue ::= Expression

L'utilisation des variables d'une suite de tests ne devra pas supposer un ordre particulier d'exécution des tests élémentaires car ces tests peuvent être exécutés indépendamment les uns des autres.

*Exemple 20* – Déclaration des variables de suites de tests

| **Déclaration des variables de suites de tests** | | | |
|---|---|---|---|
| **Nom de la variable** | **Type** | **Valeur** | **Commentaires** |
| Etat | IA5String | «repos» "idle" | Indique l'état stable final du test élémentaire précédent, s'il y en a, pour faciliter la détermination du préambule à utiliser. |

10.7.2 *Valuation des variables de suites de tests*

A l'origine, les variables de suite de tests ne sont pas valuées. Elles peuvent être valuées ou revaluées dans les contextes suivants:

a) au point de déclaration si une valeur initiale leur est affectée;

b) lorsque la variable de suite de tests apparaît dans le membre gauche d'une déclaration d'affectation (voir le § 14.10.4).

Une fois la variable de suite de tests valuée, elle conserve sa valeur jusqu'à ce qu'il lui soit affectée une autre valeur en cours d'exécution, sinon jusqu'à la fin de la suite de tests.

Si une variable non valuée d'une suite de tests apparaît dans le membre droit d'une affectation, il s'agit d'une erreur de test élémentaire.

10.7.3 *Déclarations des variables de test élémentaire*

Une suite de tests peut utiliser un ensemble de variables déclarées globalement pour cette suite de tests, mais dont le domaine d'application est défini comme local pour le test élémentaire. Ces variables sont appelées variables de test élémentaire.

Les informations suivantes sont fournies pour chaque variable déclarée:

a) nom;

b) type, il s'agira d'un type prédéfini, d'un type ASN.1, d'un type suite de tests ou d'un type unité PDU;

c) valeur initiale (le cas échéant), la colonne valeur initiale servira à affecter une valeur initiale à une variable de test élémentaire en son point de déclaration; les termes de l'expression de cette valeur ne contiendront ni variables de suite de tests ni variables de test élémentaire; la valeur prise par l'expression sera du type indiqué dans la colonne type. La spécification d'une valeur initiale est facultative.

Ces informations seront fournies dans le format de formulaire suivant:

| **Déclaration des variables de test élémentaire** | | | |
|---|---|---|---|
| **Nom de la variable** | **Type** | **Valeur** | **Commentaires** |
| .<br>.<br>*TS_VarIdentifier*<br>.<br>. | .<br>.<br>*Type*<br>.<br>. | .<br>.<br>*[DeclarationValue]*<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | | |

**Formulaire 14 – Déclaration des variables de test élémentaire**

*Remarque* – Lorsqu'on utilise des variables de test élémentaire comme variables locales dans un module de test, il faut prêter attention aux possibles conflits d'appellation avec les variables des autres modules de test et tests élémentaires. Pour éviter ces problèmes, un concepteur de suite de tests peut adopter une convention de dénomination garantissant l'unicité d'appellation de ces variables dans la suite de tests.

*Définition syntaxique*

103 TC_VarIdentifier ::= Identifier
331 Type ::= PredefinedType | ReferenceType
332 PredefinedType ::= **INTEGER** | **BOOLEAN** | **BITSTRING** | **HEXSTRING** | **OCTETSTRING** | CharacterString
333 CharacterString ::= **NumericString** | **PrintableString** | **TeletexString** | **VideotexString** | **VisibleString** | **IA5String** | **GraphicString** | **GeneralString**
334 ReferenceType ::= TS_TypeIdentifier | ASP_Identifier | PDU_Identifier
335 TS_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1_TypeIdentifier
93 DeclarationValue ::= Expression

10.7.4 *Valuation des variables de test élémentaire*

A l'origine, les variables de test élémentaire ne sont pas valuées. Elles peuvent être valuées ou revaluées dans les contextes suivants:

a) au point de déclaration si une valeur initiale leur est affectée;

b) lorsque la variable de test élémentaire apparaît dans le membre gauche d'une déclaration d'affectation (voir le § 14.10.4).

Une fois la variable de test élémentaire valuée, elle conserve sa valeur jusqu'à ce qu'il lui soit affecté une autre valeur en cours d'exécution, sinon jusqu'à la fin de l'exécution du test élémentaire. A la fin du test élémentaire, la variable de test élémentaire reprend sa valeur initiale, si elle en a une; sinon, elle devient non valuée.

Si une variable de test élémentaire non valuée apparaît dans le membre droit d'une affectation, il s'agit d'une erreur de test élémentaire.

10.8 *Déclaration de points PCO*

Cette partie de suite de tests abstraite (ATS) énumère l'ensemble des points de contrôle et d'observation (PCO) à utiliser dans la suite de tests et explique où ils existent dans l'environnement de test.

Le nombre des points PCO sera celui qui est spécifié au § 7.5 de la Recommandation X.290 et au § 12.6 de la Recommandation X.291, pour la ou les méthode(s) identifiée(s) dans la table de structure de la suite de tests.

Les déclarations comportementales TTCN spécifiées pour être exécutées au point PCO du testeur supérieur n'auront pas de spécifications allant au-delà de celles de la Recommandation X.291.

En notation TTCN, le modèle de point PCO repose sur deux files d'attente FIFO (premier entré, premier sorti):

– une file de sortie pour l'envoi de primitives ASP et/ou d'unités PDU

– une file d'entrée pour la réception de primitives ASP et/ou d'unités PDU

La file de sortie est supposée se trouver au niveau du fournisseur de service sous-jacent ou, dans le cas du testeur supérieur, au niveau de l'instance sous test (IUT).

Un événement SEND (Envoi) est réussi lorsqu'il est transmis du testeur inférieur au fournisseur de service, ou du testeur supérieur à l'instance IUT.

Le testeur est muni d'une file d'attente d'entrée pour la réception des événements. Tous les événements entrants sont placés à la suite dans cette file et traités dans leur ordre d'arrivée, sans qu'aucun soit perdu.

*Remarque* – Ce modèle de file d'attente est purement abstrait et ne suppose aucune mise en œuvre particulière.

Les informations suivantes seront fournies pour chaque point PCO utilisé dans la suite de tests:

a) son nom, utilisé dans les descriptions comportementales pour spécifier l'endroit où des événements particuliers interviennent;

b) son type, utilisé pour identifier la limite du service dans lequel le point PCO est situé;

c) son rôle, pour expliquer le type de testeur placé au point PCO. L'identificateur prédéfini **UT** (*upper tester*) indique que le point PCO est un point PCO de testeur supérieur, l'identificateur **LT** (*lower tester*) spécifiant qu'il s'agit d'un point PCO de testeur inférieur.

Ces informations seront fournies dans le format de formulaire suivant:

| **Déclaration des points PCO** | | | |
|---|---|---|---|
| **Nom du point PCO** | **Type PCO** | **Rôle** | **Commentaires** |
| . . *PCO_Identifier* . . | . . *PCO_TypeIdentifier* . . | . . *PCO_Role* . . | . . *[FreeText]* . . |
| **Commentaires détaillés:** *[FreeText]* | | | |

**Formulaire 15 – Déclaration des points PCO**

*Définition syntaxique*

109 PCO_Identifier ::= Identifier
111 PCO_TypeIdentifier ::= Identifier
113 PCO_Role ::= **UT** | **LT**

*Exemple 21* – Déclaration des points PCO

| **Déclaration des points PCO** | | | |
|---|---|---|---|
| **Nom du point PCO** | **Type PCO** | **Rôle** | **Commentaires** |
| L | TSAP | LT | Point d'accès au service transport, niveau testeur inférieur. |
| U | SSAP | UT | Point d'accès au service session, niveau testeur supérieur. |

Les points de contrôle et d'observation PCO sont d'habitude des points d'accès au service (SAP), mais plus généralement, il peut s'agir de n'importe quel point d'où il est possible d'observer et de contrôler les événements de test. Il est toutefois possible de définir un point PCO correspondant à un ***ensemble*** de points SAP, à condition que tous ces points SAP:

– soient situés au même endroit (c'est-à-dire dans le testeur inférieur ou dans le testeur supérieur);

– donnent accès au même service.

Lorsqu'un point PCO correspond à plusieurs points SAP, on identifie un point SAP particulier par son adresse. Les points PCO sont normalement associés à un point d'accès au service (SAP) de l'instance IUT ou du fournisseur de service de la couche (N-1).

*Remarque* – Un point PCO peut n'être associé à aucun point SAP, notamment dans le cas d'une couche composée de sous-couches (par exemple, dans la couche Application, ou dans les couches inférieures, où un point de rattachement de sous-réseau n'est pas un point SAP).

10.9 *Déclarations des temporisateurs*

Une suite de tests peut utiliser des temporisateurs. Les informations suivantes seront fournies pour chaque temporisateur:

a) son nom;

b) sa durée par défaut, où la durée par défaut de la temporisation est une expression qui pourra être omise si sa valeur ne peut être déterminée avant l'exécution de la suite de tests; les termes de l'expression ne contiendront ni variables de suite de tests ni variables de test élémentaire. L'expression de la durée de temporisation prendra une valeur entière non signée (donc positive);

c) l'unité de temps, c'est-à-dire:

1) **ps** (picoseconde);

2) **ns** (nanoseconde);

3) **μs** (microseconde);

4) **ms** (milliseconde);

5) **s** (seconde);

6) **min** (minute).

Les unités de temps sont déterminées par le concepteur de la suite de tests et fixées au moment de la spécification. Différents temporisateurs pourront utiliser des unités de temps différentes dans la même suite de tests. S'il existe une entrée de déclaration PICS ou d'information PIXIT, la déclaration de temporisateur doit spécifier les mêmes unités que dans l'entrée PICS/PIXIT.

Ces informations seront fournies dans le format illustré dans le formulaire suivant:

| **Déclaration des temporisateurs** | | | |
|---|---|---|---|
| **Nom du temporisateur** | **Durée** | **Unité** | **Commentaires** |
| . . *TimerIdentifier* . . | . . *[DeclarationValue]* . . | . . *TimeUnit* . . | . . *[FreeText]* . . |
| **Commentaires détaillés:** *[FreeText]* | | | |

**Formulaire 16 – Déclaration des temporisateurs**

*Définition syntaxique*

117 TimerIdentifier ::= Identifier
93 DeclarationValue ::= Expression
120 TimeUnit ::= **ps** | **ns** | **μs** | **ms** | **s** | **min**

*Exemple 22* – Déclaration des temporisateurs

| Déclaration des temporisateurs | | | |
|---|---|---|---|
| **Nom du temporisateur** | **Durée** | **Unité** | **Commentaires** |
| wait (attente) | 15 | **s** | Temporisation courante «à tout faire». |
| no_response (pas_de_réponse) | A | **min** | Sert à attendre que l'instance sous test se connecte ou réagisse à l'établissement d'une connexion, sa durée est supérieure à celle d'une temporisation courante. Elle tire sa valeur des informations PIXIT. |
| delay_time (délai) | | **ms** | Durée à déterminer pendant l'éxécution de la suite de tests. |

10.10 *Définition des types de primitives ASP*

10.10.1 *Introduction*

Cette partie d'une suite de tests TTCN a pour objet de déclarer les types de primitives ASP pouvant être envoyées ou reçues aux points PCO déclarés. Les définitions des types de primitives ASP peuvent inclure, si nécessaire, des définitions de types en notation ASN.1.

10.10.2 *Définition des types de primitives ASP à l'aide de tables*

Les informations suivantes seront fournies pour chaque primitive ASP:

a) son nom, c'est-à-dire le nom complet, tel qu'il est donné dans la Recommandation sur les protocoles correspondants; si une abréviation est utilisée, elle sera suivie du nom complet entre parenthèses;

b) le type du point PCO associé à la primitive ASP, ce type de point PCO étant l'un de ceux qui sont employés dans le formulaire de déclaration de point PCO. Si un seul point PCO est défini dans la suite de tests, la spécification du type de point PCO dans la définition du type de primitive ASP est facultative.

c) la liste des paramètres associés à la primitive ASP, donnant pour chaque paramètre les informations suivantes:

1) son nom, c'est-à-dire:

– soit le nom complet, tel qu'il est donné dans la Recommandation sur les protocoles correspondants; si une abréviation est utilisée, elle sera suivie du nom complet entre parenthèses;

– soit le macrosymbole (<–) indiquant que l'entrée dans la colonne type identifie un ensemble de paramètres qui doit être inséré directement dans la liste des paramètres de la primitive ASP; ce macrosymbole ne sera utilisé qu'avec les types structurés mentionnés dans les définitions des types structurés;

2) son type et un attribut facultatif, les paramètres pouvant être d'un type à structure arbitrairement complexe, notamment d'un type de suite de tests (prédéfini, simple, structuré ou ASN.1); si un paramètre est structuré comme une unité PDU, son type est alors déclaré:

– soit comme un identificateur d'unité PDU pour indiquer que, dans la contrainte de primitive ASP, ce paramètre peut être chaîné à une contrainte d'unité PDU d'un type PDU spécifique;

– soit sous la forme du symbole terminal **PDU** pour indiquer que, dans la contrainte de primitive ASP, ce paramètre peut être chaîné à une contrainte d'unité PDU d'un type PDU quelconque;

et l'attribut facultatif étant Length (longueur); la spécification de cet attribut peut restreindre ce paramètre à une longueur ou à un intervalle de longueurs particulier, conformément au § 10.12. Les valeurs des longueurs s'interprètent conformément au tableau 4/X.292. Les limites seront spécifiées sous forme d'entiers non négatifs, de paramètres de suite de tests, de constantes de suite de tests ou du mot clé INFINITY (infini).

Les spécifications de longueur définies pour le type de paramètre de la primitive ASP dans les définitions de type, de suite de tests ne doivent pas être contraires aux spécifications de longueur figurant dans la définition du type de la primitive ASP, c'est-à-dire que l'ensemble de chaînes défini par restriction de longueur dans la définition d'une primitive ASP doit former un sous-ensemble strict de l'ensemble de chaînes défini dans la définition du type de la suite de tests.

Le mot clé INFINITY peut servir de limite supérieure pour indiquer une longueur maximale illimitée.

*Remarque* – Il n'est généralement pas nécessaire de limiter la longueur des paramètres des primitives ASP; cependant, dans certains cas, cela peut s'avérer nécessaire pour limiter effectivement la longueur d'un champ de l'unité PDU correspondante dans un protocole sous-jacent.

Les paramètres des définitions de types de primitives ASP sont considérés comme facultatifs; c'est-à-dire que, dans des instances de ces types, des paramètres entiers peuvent être omis.

Ces informations seront fournies dans le format de formulaire suivant:

| **Définition de type de primitives ASP** | | |
|---|---|---|
| **Nom de la primitive :** *ASP_ Id&FullId*<br>**Type du point PCO :** *[PCO_TypeIdentifier]*<br>**Commentaires :** *[FreeText]* | | |
| **Nom du paramètre** | **Type du paramètre** | **Commentaires** |
| .<br>.<br>*ASP_ParIdOrMacro*<br>.<br>. | .<br>.<br>*Type&Attributes*<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | |

**Formulaire 17 – Définition de type de primitives ASP**

*Définition syntaxique*

```
127  ASP_Id&FullId ::= ASP_Identifier [FullIdentifier]
128  ASP_Identifier ::= Identifier
43   FullIdentifier ::= "(" BoundedFreeText ")"
111  PCO_TypeIdentifier ::= Identifier
132  ASP_ParIdOrMacro ::= ASP_ParId&FullId | MacroSymbol
133  ASP_ParId&FullId ::= ASP_ParIdentifier [FullIdentifier]
134  ASP_ParIdentifier ::= Identifier
150  MacroSymbol ::= "<–"
154  Type&Attributes ::= (Type [LengthAttribute]) | PDU
331  Type ::= PredefinedType | ReferenceType
332  PredefinedType ::= INTEGER | BOOLEAN | BITSTRING | HEXSTRING | OCTETSTRING |
     CharacterString
333  CharacterString ::= NumericString | PrintableString | TeletexString | VideotexString |
     VisibleString | IA5String | GraphicString | GeneralString
334  ReferenceType ::= TS_TypeIdentifier | ASP_Identifier | PDU_Identifier
335  TS_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1_TypeIdentifier
155  LengthAttribute ::= SingleLength | RangeLength
156  SingleLength ::= "[" Bound "]"
157  Bound ::= Number | TS_ParIdentifier | TS_ConstIdentifier
158  RangeLength ::= "[" LowerBound To UpperBound "]"
159  LowerBound ::= Bound
37   To ::= TO | ".."
160  UpperBound ::= Bound | INFINITY
```

*Exemple 23* – Primitive de service abstrait T_CONNECTrequest

La figure ci-dessous représente un exemple tiré du service transport [(Recommandation X.214)]. Il pourrait s'agir d'une des primitives ASP utilisées pour décrire le comportement d'un testeur supérieur abstrait dans une suite de tests DS (méthode de test monocouche répartie) pour le transport de classe 0. CDA, CGA et QOS sont des types de suite de tests.

| **Définition de type de primitive ASP** | | |
|---|---|---|
| **Nom de la primitive :** CONreq (T_CONNECTrequest)<br>**Type du point: :** TSAP<br>**Commentaires :** | | |
| **Nom du paramètre** | **Type du paramètre** | **Commentaires** |
| Cda (Called Address) | CDA | adresse appelée du testeur supérieur |
| Cga (Calling Address) | CGA | adresse appelante du testeur inférieur |
| QoS (Quality of Service) | QOS | qualité de service – doit vérifier que la classe 0 est utilisée |
| **Commentaires détaillés:** Primitive ASP à transmettre au point d'accès au service transport | | |

10.10.3 *Utilisation de types structurés dans les définitions de types des primitives ASP*

Les définitions de primitives ASP peuvent se référer de deux manières possibles à un type structuré:

a) si la définition contient un nom de paramètre, le type structuré mentionné forme une sous-structure. Ceci permet de définir des primitives ASP à structure paramétrique multiniveau;

b) si on utilise le macrosymbole (<–) à la place d'un nom de paramètre, ceci équivaut alors à un développement de macro; l'entrée dans la définition du type de primitive ASP se développe directement en une liste de paramètres sans introduire de niveau supplémentaire de sous-structure.

Ce macrosymbole ne doit pas être utilisé sur la même ligne que des références à des types ASN.1 ou à des types simples; en d'autres termes, seuls des types structurés définis sous forme tabulaire peuvent être développés dans d'autres types structurés en tant que développements de macro.

10.10.4 *Définitions de types de primitives ASP à l'aide de la notation ASN.1*

Les primitives ASP peuvent, si nécessaire, être spécifiées en notation ASN.1. On a alors recours à une définition ASN.1 en utilisant la syntaxe ASN.1 telle qu'elle est définie dans la Recommandation X.208. Les informations suivantes seront fournies pour chaque primitive ASP en notation ASN.1:

a) son nom, c'est-à-dire le nom complet, tel qu'il est donné dans la Recommandation sur les protocoles pertinente; si une abréviation est utilisée, elle sera suivie du nom complet entre parenthèses;

b) le type du point PCO associé à la primitive ASP, ce type de point PCO étant l'un de ceux employés dans le formulaire de déclaration des points PCO. Si un seul point PCO est défini dans une suite de tests, la spécification du type de point PCO dans la définition du type de la primitive ASP devient facultative;

c) la définition ASN1 du type de primitive ASP, qui doit respecter la syntaxe définie dans la Recommandation X.208. Le symbole trait d'union (-) ne doit pas figurer dans les identificateurs de cette définition; il peut être remplacé par le symbole de soulignement (_). L'identificateur de primitive ASP porté dans l'en-tête de la table est le nom du premier type défini dans le corps de la table.

Les types référencés à partir de la définition de la primitive ASP seront définis soit dans d'autres tables de définition de types en notation ASN.1, soit par référence dans la table de référence des types en notation ASN.1 soit enfin localement dans la même table, à la suite de la définition du premier type. Les types définis localement ne seront pas utilisés ailleurs dans la suite de tests.

Des commentaires ASN.1 peuvent être portés dans le corps de la table. La colonne commentaires n'existe pas.

Ces informations seront fournies dans le format illustré dans le formulaire suivant:

| **Définition de types de primitive ASP en notation ASN.1** |
|---|
| **Nom de la primitive :** *ASP_Id&FullId*<br>**Type de point PCO :** *[PCO_TypeIdentifier]*<br>**Commentaires :** *[FreeText]* |
| **Définition du type** |
| .<br>.<br>*ASN1_Type&LocalTypes*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* |

**Formulaire 18 – Définition de type de primitive ASP en notation ASN.1**

*Définition syntaxique*

```
127  ASP_Id&FullId ::= ASP_Identifier [FullIdentifier]
128  ASP_Identifier ::= Identifier
43   FullIdentifier ::= "(" BoundedFreeText ")"
111  PCO_TypeIdentifier ::= Identifier
57   ASN1_Type&LocalTypes ::= ASN1_Type {ASN1_LocalType}
58   ASN1_Type ::= Type
     /* REFERENCE – Type étant un symbole non terminal défini dans la Recommandation X.208 */
59   ASN1_LocalType ::= Typeassignment
     /* REFERENCE – Typeassignment étant un symbole non terminal défini dans la
     Recommandation X.208 */
```

10.10.5 *Définition par référence de types de primitives ASP en notation ASN.1*

Les primitives ASP peuvent être spécifiées par une référence précise à une primitive ASP en notation ASN.1 définie dans une Recommandation sur l'OSI ou par référence à un type ASN.1 défini dans un module ASN.1 rattaché à la suite de tests. Les informations suivantes seront fournies pour chaque primitive ASP:

a) son nom, qui peut être utilisé dans toute la suite de tests;

b) le type de point PCO associé à cette primitive ASP, ce type devant être l'un de ceux mentionnés dans le formulaire de déclaration de points PCO. Si un seul point PCO est défini dans une suite de tests, la spécification du type de point PCO devient facultative dans la définition de type de primitive ASP;

c) la référence du type, qui respectera les règles relatives aux identificateurs énoncées dans la Recommandation X.208;

d) l'identificateur du module, composé d'une référence de module conforme aux règles relatives aux identificateurs énoncées dans la Recommandation X.208, et d'un identificateur facultatif d'objets ObjectIdentifier.

Ces informations sont fournies dans le formulaire suivant:

| **Définition par référence de types ASN.1 de primitives ASP** | | | | |
|---|---|---|---|---|
| **Nom de la primitive ASP** | **Type de point PCO** | **Référence du Type** | **Identificateur de module** | **Commentaires** |
| . . *ASP_Id&FullId* . . | . . *[PCO_TypeIdentifier]* . . | . . *TypeReference* . . | . . *ModuleIdentifier* . . | . . *[FreeText]* . . |
| **Commentaires détaillés:** *[FreeText]* | | | | |

**Formulaire 19 – Définition par référence de types ASN.1 de primitives ASP**

*Définition syntaxique*

```
127  ASP_Id&FullId ::= ASP_Identifier [FullIdentifier]
128  ASP_Identifier ::= Identifier
43   FullIdentifier ::= "(" BoundedFreeText ")"
111  PCO_TypeIdentifier ::= Identifier
63   TypeReference ::= typereference
     /* REFERENCE – Typereference est défini au § 8.2 de la Recommandation X.208 */
65   ModuleIdentifier ::= ModuleIdentifier
     /* REFERENCE – ModuleIdentifier est un symbole non-terminal défini dans la
     Recommandation X.208 */
```

Les identificateurs ASN.1 des références aux types et aux valeurs peuvent comporter des traits d'union. Pour pouvoir importer des définitions en TTCN, il est nécessaire de remplacer ces traits d'union par des caractères de soulignement (voir le § A.4.2.1).

## 10.11 *Définition des types d'unité PDU*

### 10.11.1 *Introduction*

Cette partie d'une suite de tests abstraite en notation TTCN a pour objet de déclarer les types d'unités PDU qui peuvent être envoyées ou reçues, directement ou imbriquées dans des primitives ASP, aux points PCO déclarés. Les définitions des types d'unité PDU peuvent comporter si nécessaire des définitions de type en notation ASN.1. Les définitions d'unités PDU définissent l'ensemble des unités PDU échangées avec l'instance sous test et qui sont syntaxiquement valides par rapport à la suite ATS, mais pas nécessairement valides par rapport à la spécification de protocole.

Tous les champs des unités PDU définies dans la Recommandation sur les protocoles pertinente doivent être déclarés soit explicitement, soit implicitement par référence à des règles de codage (les règles de codage ASN.1, si elles sont applicables).

Le codage des champs d'unités PDU sera conforme à la spécification de protocole correspondante.

10.11.2 *Définition des types d'unités PDU à l'aide de tables*

La définition des unités PDU est semblable à celle des primitives ASP. Les informations suivantes seront fournies pour chaque unité PDU:

a) son nom, c'est-à-dire le nom complet, tel qu'il est donné dans la Recommandation sur les protocoles pertinente; si une abréviation est utilisée, elle sera suivie du nom complet entre parenthèses;

b) le type de point PCO associé à l'unité PDU, ce type de point PCO étant l'un de ceux mentionnés dans les déclarations de points PCO; si, dans toute la suite de tests, une unité PDU n'est envoyée ou reçue qu'imbriquée dans des primitives ASP, la spécification du type de point PCO devient facultative; si un seul point PCO est défini dans une suite de tests, la spécification du type de point PCO dans la définition du type d'unité PDU devient facultative;

c) la liste des champs associés à l'unité PDU, donnant, pour chaque champ, les informations suivantes:

1) son nom, c'est-à-dire:

– soit le nom complet, tel qu'il est donné dans la Recommandation sur les protocoles pertinente; si une abréviation est utilisée, elle sera suivie du nom complet entre parenthèses;

– soit le macrosymbole (<–) indiquant que l'entrée dans la colonne type identifie un ensemble de champs à insérer directement dans la liste des champs de l'unité PDU; ce macrosymbole ne sera utilisé qu'avec les types structurés mentionnés dans les définitions correspondantes des types structurés;

2) son type et un attribut facultatif, les champs pouvant être d'un type à structure arbitrairement complexe, notamment d'un type de suite de tests (prédéfini, simple, structuré ou ASN.1); si un champ doit être structuré comme une unité PDU, son type peut alors être déclaré:

– soit comme identificateur d'unité PDU pour indiquer que, dans la contrainte relative à l'unité PDU, ce champ peut être chaîné à une contrainte d'unité PDU d'un type PDU spécifique;

– soit sous la forme du symbole terminal **PDU** pour indiquer que, dans la contrainte relative à l'unité PDU, ce champ peut être chaîné à une contrainte d'unité PDU d'un type PDU quelconque, et l'attribut facultatif étant Length (longueur), la spécification de cet attribut pouvant restreindre ce champ à une longueur ou à un intervalle de longueurs particulier, conformément au § 10.12. Les valeurs de longueur s'interprètent conformément au tableau 4/X.292. Les limites seront spécifiées en termes de littéraux entiers non négatifs, de paramètres de suite de tests, de constantes de suite de tests ou du mot clé INFINITY (infini).

Les spécifications de longueur définies pour le type de champ d'unité PDU dans les définitions de suite de tests ne doivent pas être contraires aux spécifications de longueur figurant dans la définition du type d'unité PDU, c'est-à-dire que l'ensemble de chaînes défini par restriction de longueur figurant dans la définition d'une unité PDU doit former un sous-ensemble strict de l'ensemble de chaînes défini dans la définition du type de la suite de tests.

Le mot clé INFINITY peut servir de limite supérieure pour indiquer une longueur maximale illimitée.

Les champs des définitions des types d'unité PDU sont considérés comme facultatifs, c'est-à-dire que dans des instances de ces types, des champs entiers peuvent être omis.

Ces informations seront fournies dans le format de formulaire suivant:

| **Définition de type d'unité PDU** | | |
|---|---|---|
| **Nom de l'unité :** *PDU_ Id&FullId*<br>**Type du point :** *[PCO_TypeIdentifier]*<br>**Commentaires :** *[FreeText]* | | |
| **Nom du paramètre** | **Type du champ** | **Commentaires** |
| .<br>.<br>*PDU_FieldIdOrMacro*<br>.<br>. | .<br>.<br>*Type&Attributes*<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | |

**Formulaire 20 – Définition de type d'unité PDU**

*Définition syntaxique*

144 PDU_Id&FullId ::= PDU_Identifier [FullIdentifier]
145 PDU_Identifier ::= Identifier
43 FullIdentifier ::= "(" BoundedFreeText ")"
111 PCO_TypeIdentifier ::= Identifier
149 PDU_FieldIdOrMacro ::= PDU_FieldId&FullId | MacroSymbol
151 PDU_FieldId&FullId ::= PDU_FieldIdentifier [FullIdentifier]
152 PDU_FieldIdentifier ::= Identifier
150 MacroSymbol ::= "<"
154 Type&Attributes ::= (Type [LengthAttribute] | **PDU**
331 Type ::= PredefinedType | ReferenceType
332 PredefinedType ::= **INTEGER** | **BOOLEAN** | **BITSTRING** | **HEXSTRING** | **OCTETSTRING** | CharacterString
333 CharacterString ::= **NumericString** | **PrintableString** | **TeletexString** | **VideotexString** | **VisibleString** | **IA5String** | **GraphicString** | **GeneralString**
334 ReferenceType ::= TS_TypeIdentifier | ASP_Identifier | PDU_Identifier
335 TS_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1_TypeIdentifier
155 LengthAttribute ::= SingleLength | RangeLength
156 SingleLength ::= "[" Bound "]"
157 Bound ::= Number | TS_ParIdentifier | TS_ConstIdentifier
158 RangeLength ::= "[" LowerBound To UpperBound "]"
159 LowerBound ::= Bound
37 To ::= **TO** | ".."
160 UpperBound ::= Bound | **INFINITY**

*Exemple 24* – Définition d'un type courant d'unité PDU

| **Définition de type d'unité PDU** | | |
|---|---|---|
| **Nom de l'unité PDU :** INTC (Interrupt Confirm)<br>**Type du point PCO :** NSAP<br>**Commentaires :** | | |
| **Nom du champ** | **Type du champ** | **Commentaires** |
| GFI | BITSTRING | Identificateur général de format |
| LCGN | BITSTRING | Numéro du groupe de canaux logiques |
| LCN | BITSTRING | Identificateur de canal logique |
| PTI | OCTETSTRING | Identificateur de type de paquet |
| EXTRA | OCTETSTRING | Pour créer des paquets INTC (confirmation d'interruption) longs |

10.11.3 *Utilisation de types structurés dans les définitions d'unités PDU*

Les définitions d'unité PDU peuvent se référer de deux manières possibles à un type structuré:

a) si la définition contient un nom de champ, le type structuré mentionné forme une sous-structure. Ceci permet de définir des unités PDU à structure de champs multiniveau;

b) si on utilise le macrosymbole (<–) à la place d'un nom de champ, cela équivaut alors à un développement de macro; l'entrée dans la définition du type d'unité PDU se développe directement en une liste de champs sans introduire de niveau supplémentaire de sous-structure.

Ce macrosymbole ne doit pas être utilisé sur la même ligne que des références à des types ASN.1 ou à des types simples; en d'autres termes, seuls des types structurés définis sous forme tabulaire peuvent être développés dans d'autres types structurés en tant que développements de macro.

10.11.4 *Définition de types d'unités PDU à l'aide de la notation ASN.1*

Les unités PDU peuvent, si nécessaire, être spécifiées en notation ASN.1. On a alors recours à une définition ASN.1 en utilisant la syntaxe ASN.1 telle qu'elle est définie dans la Recommandation X.208. Les informations suivantes seront fournies pour chaque unité PDU en notation ASN.1:

a) son nom, c'est-à-dire le nom complet, tel qu'il est donné dans la Recommandation sur les protocoles correspondants, si une abréviation est utilisée, elle sera suivie du nom complet entre parenthèses;

b) le type du point PCO associé à l'unité PDU, ce type de point PCO étant l'un de ceux qui sont employés dans les déclarations des points PCO; si une unité PDU est toujours envoyée ou reçue imbriquée dans des primitives ASP, la spécification du type de point PCO dans la définition du type d'unité PDU devient facultative; si un seul point PCO est défini dans une suite de tests, la spécification du type de point PCO dans la définition du type de l'unité PDU devient facultative;

c) la définition ASN.1 du type d'unité PDU, qui doit respecter la syntaxe définie dans la Recommandation X.208. Le symbole trait d'union (-) ne doit pas figurer dans les identificateurs de cette définition; il peut être remplacé par le symbole de soulignement (_). L'identificateur d'unité PDU porté dans l'en-tête de la table est le nom du premier type défini dans le corps de la table.

Les types référencés à partir de la définition de l'unité PDU seront définis soit dans d'autres tables de définition de type en notation ASN.1, soit par référence dans la table de référence des types en notation ASN.1, soit enfin localement dans la même table, à la suite de la définition du premier type. Les types définis localement ne seront pas utilisés ailleurs dans la suite de tests.

Des commentaires ASN.1 peuvent être portés dans le corps de la table. La colonne commentaires n'existe pas.

Ces informations seront fournies dans le formulaire suivant:

| **Définition de type d'unité PDU en notation ASN.1** |
|---|
| **Nom de l'unité PDU :** *PDU_Id&FullId*<br>**Type de point PCO :** *[PCO_TypeIdentifier]*<br>**Commentaires :** *[FreeText]* |
| **Définition du type** |
| .<br>.<br>*ASN1_Type&LocalTypes*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* |

**Formulaire 21 – Définition de type d'unité PDU en notation ASN.1**

*Définition syntaxique*

144 PDU_Id&FullId ::= PDU_Identifier [FullIdentifier]
145 PDU_Identifier ::= Identifier
43 FullIdenfifier ::= "(" BoundedFreeText ")"
111 PCO_TypeIdentifier ::= Identifier
57 ASN1_Type&LocalTypes ::= ASN1_Type {ASN1_LocalType}
58 ASN1_Type ::= *Type*

/* REFERENCE – Type étant un symbole non terminal défini dans la Recommandation X.208 */

59 ASN1_LocalType ::= *Typeassignment*

/* REFERENCE – Typeassignment étant un symbole non terminal défini dans la Recommandation X.208 */

*Exemple 25* – Définition ASN.1 d'une unité FTAM (transfert, accès et gestion de fichiers)

| **Définition ASN.1 de types d'unité PDU** |
|---|
| **Nom de l'unité PDU :** F_INIT (F_INITIALIZE_response)<br>**Type de point PCO :**<br>**Commentaires :** |
| **Définition du type** |
| SEQUENCE {<br>state_result State_result DEFAULT success,<br>action_result Action_Result multiple success,<br>protocol_id Protocol_Version,<br><br>-- etc.<br>} |

10.11.5 *Définitions par référence de types d'unités PDU en notation ASN.1*

Les unités PDU peuvent être spécifiées par une référence précise à une unité PDU en notation ASN.1 définie dans une Recommandation sur l'OSI ou par référence à un type ASN.1 défini dans un module ASN.1 rattaché à la suite de tests. Les informations suivantes seront fournies pour chaque unité PDU:

a) son nom, qui peut être utilisé dans toute la suite de tests;

b) le type de point PCO associé à cette unité PDU, ce type devant être l'un de ceux qui sont mentionnés dans les déclarations de points PCO; si une unité PDU n'est envoyée ou reçue qu'imbriquée dans des primitives ASP dans toute la suite de tests, la spécification du type de point PCO devient facultative; si un seul point PCO est défini dans une suite de tests, la spécification du type de point PCO devient facultative dans la définition du type d'unité PDU;

c) la référence du type, qui respectera les règles relatives aux identificateurs énoncées dans la Recommandation X.208;

d) l'identificateur du module, composé d'une référence de module conforme aux règles relatives aux identificateurs énoncées dans la Recommandation X.208, et d'un identificateur facultatif d'objets ObjectIdentifier.

Ces informations seront fournies dans le format illustré dans le formulaire suivant:

| **Définition par référence de types ASN.1 d'unités PDU** | | | | |
|---|---|---|---|---|
| **Nom de l'unité PDU** | **Type de point PCO** | **Référence du type** | **Identificateur de module** | **Commentaires** |
| . . *PDU_Id&FullId* . . | . . *[PCO_TypeIdentifier]* . . | . . *TypeReference* . . | . . *ModuleIdentifier* . . | . . *[FreeText]* . . |
| **Commentaires détaillés:** *[FreeText]* | | | | |

**Formulaire 22 – Définition par référence de types ASN.1 d'unités PDU**

*Définition syntaxique*

```
144  PDU_Id&FullId ::= PDU_Identifier [FullIdentifier]
145  PDU_Identifier ::= Identifier
43   FullIdentifier ::= "(" BoundedFreeText ")"
111  PCO_TypeIdentifier ::= Identifier
63   TypeReference ::= typereference
     /* REFERENCE – Typereference est défini au § 8.2 de la Recommandation X.208 */
65   ModuleIdentifier ::= ModuleIdentifier
     /* REFERENCE – ModuleIdentifier est un symbole non-terminal défini dans la
     Recommandation X.208 */
```

Les identificateurs ASN.1 des références aux types et aux valeurs peuvent comporter des traits d'union. Pour pouvoir importer des définitions en TTCN, il est nécessaire de remplacer ces traits d'union par des caractères de soulignement (voir le § A.4.2.1).

10.12 *Spécifications de longueur des chaînes*

10.12.1 La notation TTCN prévoit la spécification de restrictions de longueur pour les types de chaînes (BITSTRING, HEXSTRING, OCTETSTRING et pour tous les types de chaînes de caractères CharacterString) dans les cas suivants:

a) lors de la déclaration de types de suite de tests en tant que restrictions de type;

b) lors de la déclaration de paramètres simples de primitives ASP, de champs simples d'unités PDU ou d'éléments simples de types structurés en tant qu'attributs de paramètre, de champ ou de type d'élément;

c) lors de la définition de contraintes de primitives ASP, d'unités PDU ou de type structuré en tant qu'attribut d'une valeur de contrainte.

10.12.2 Les spécifications de longueur peuvent être indiquées dans les formats suivants:

a) [Length] la longueur des valeurs des chaînes d'un type donné restreignant à la seule valeur *Length*;

b) [MinLength **TO** MaxLength] ou [MinLength .. MaxLength] spécifiant une longueur minimale et une longueur maximale pour les valeurs d'un type de chaîne donné.

Les limites de longueur: *Length*, *MinLength* et *MaxLength* auront divers degrés de complexité selon l'endroit où elles sont utilisées. Dans tous les cas, elles prendront des valeurs entières non négatives. On peut également utiliser le mot clé INFINITY pour indiquer que la longueur maximale est illimitée. Lorsqu'un intervalle de longueurs est spécifiée, la limite inférieure sera celle de gauche.

Dans le contexte des contraintes, des restrictions de longueur peuvent également être spécifiées pour les valeurs des types SEQUENCE OF et SET OF, limitant ainsi le nombre de leurs éléments.

Les unités de longueur des différents types de chaînes sont indiquées dans le tableau 4/X.292:

TABLEAU 4/X.292

**Unités de longueur utilisées pour les spécifications de longueurs de champs**

| Type | Unités de longueur |
|---|---|
| BITSTRING | Bits |
| HEXSTRING | Chiffres hexadécimaux |
| OCTETSTRING | Octets |
| CharacterString | Caractères |
| SEQUENCE OF | Nombre d'éléments du type de base |
| SET OF | Nombre d'éléments du type de base |

Les spécifications de longueur ne doivent pas être contradictoires, c'est-à-dire qu'une restriction portant sur un type (un ensemble de valeurs) lui-même déjà restreint doit spécifier un sous-ensemble des valeurs de ce type.

*Exemple 26* – Spécification de longueur

Si on considère les définitions suivantes de types ASN.1:

```
type 1 ::= OCTETSTRING [0 .. 25]
type 2 ::= type 1 [15 .. 24]
```

la restriction de longueur portant sur le type2 est correcte puisque le type2 comprend toutes les chaînes OCTETSTRING d'une longueur minimale de 15 et d'une longueur maximale de 24, qui constituent un sous-ensemble strict de toutes les chaînes OCTETSTRING d'une longueur maximale de 25. En revanche,

```
type 2 ::= type 1 [15 .. 30]
```

n'est pas valide puisqu'il contient des valeurs non incluses dans le type1.

10.13 *Définition de primitives ASP et d'unités PDU pour les événements SEND (envoi)*

Dans les primitives ASP et les unités PDU envoyées depuis le testeur, les valeurs des paramètres ASP et des champs PDU définies dans la partie Contraintes (voir les § 11, 12 et 13) correspondront à la définition de ces paramètres ou champs. En d'autres termes:

a) cette valeur sera du type spécifié pour ce paramètre de primitive ASP ou pour ce champ d'unité PDU; et

b) chacune des valeurs respectera toutes les restrictions de longueur associées à ce type.

10.14 *Définition de primitives ASP et d'unités PDU pour les événements RECEIVE (réception)*

Le type de primitives ASP et d'unités PDU pouvant être reçues par le testeur définit la classe de primitives ASP et d'unités PDU entrantes pouvant concorder avec une spécification d'événement de ce type. Une primitive ASP ou une unité PDU entrante est considérée comme appartenant à cette classe si et seulement si:

a) les valeurs des paramètres de la primitive ASP et des champs de l'unité PDU sont du type spécifié dans la définition de la primitive ASP et de l'unité PDU;

b) cette valeur répond à toutes les restrictions de longueur associées à ce type.

Dans tous les autres cas, une primitive ASP ou unité PDU est considérée comme ne concordant pas avec une spécification d'événement de ce type.

Dans le cas de primitives ASP ou d'unités PDU possédant une sous-structure, qu'il s'agisse de types structurés ou de types ASN.1, les règles ci-dessus s'appliquent récursivement aux champs des sous-structures.

10.15 *Définitions des alias (pseudonymes)*

10.15.1 *Introduction*

Pour améliorer la lisibilité des descriptions comportementales en notation TTCN, il est possible d'utiliser un alias (pseudonyme) pour faciliter la redésignation des identificateurs de primitives ASP ou d'unités PDU dans les descriptions comportementales. Cette redésignation peut servir à souligner l'échange d'unités PDU imbriquées dans des primitives ASP.

Les informations suivantes seront fournies pour chaque pseudonyme:

a) un identificateur d'alias (pseudonyme);

b) son développement, qui est lui-même un identificateur.

Ces informations seront fournies dans le format illustré dans le formulaire suivant:

| **Définition des alias (pseudonymes)** | | |
|---|---|---|
| **Nom des alias (pseudonymes)** | **Développement** | **Commentaires** |
| .<br>.<br>*AliasIdentifier*<br>.<br>. | .<br>.<br>*Expansion*<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | |

**Formulaire 23 – Définition des alias (pseudonymes)**

*Définition syntaxique*

```
168  AliasIdentifier ::= Identifier
170  Expansion ::= ASP_Identifier | PDU_Identifier
```

10.15.2 *Développement des alias*

On appliquera les règles suivantes:

a) un alias est un identificateur respectant les règles syntaxiques relatives aux identificateurs définies dans la notation TTCN.MP, c'est-à-dire qu'un alias est délimité par un caractère (symbole) quelconque non autorisé dans un identificateur en TTCN;

b) les alias ne sont pas transitifs: si un alias apparaît en tant que développement d'un autre alias, il ne sera pas développé (c'est-à-dire qu'il s'agit d'un développement en une passe);

c) un alias ne sera utilisé que pour remplacer un identificateur de primitive ASP ou d'unité PDU dans une seule déclaration TTCN dans un arbre comportemental. Il ne sera utilisé que dans une colonne description comportementale;

d) le développement d'un alias respectera les règles syntaxiques relatives aux identificateurs, définies dans la notation TTCN.MP.

*Remarque* – Les alias étant traités comme des développements de macroinstructions, le terme AliasIdentifier (identificateur d'alias) n'apparaît pas en format BNF dans les lignes d'événement TTCN.

*Exemple 27* – Définition d'alias à partir d'une suite de tests de transport

| **Définition des alias** | | |
|---|---|---|
| **Nom alias** | **Développement** | **Commentaires** |
| CR (Connection Request) | N_DATArequest | Alias de la primitive ASP N_DATArequest, utilisé pour acheminer une unité CR_TPDU |
| DR (Disconnect Request) | N_DATArequest | Alias de la primitive ASP N_DATArequest, utilisé pour acheminer une unité DR_TPDU |
| CC (Connection Confirm) | N_DATAindication | Alias de la primitive ASP N_DATAindication, utilisé pour acheminer une unité CC_TPDU |

*Remarque* – Les alias étant traités comme des développements de macroinstructions, le terme AliasIdentifier (identificateur d'alias) n'apparaît pas en format BNF dans les lignes d'événement TTCN.

# 11 Partie contraintes

## 11.1 *Introduction*

Une suite ATS spécifiera les valeurs des paramètres des primitives ASP et des champs des unités PDU envoyées et reçues par le système de test. Dans la notation TTCN, la partie contraintes remplit ce rôle.

Les descriptions de comportement dynamique (voir le § 14) font référence aux contraintes pour élaborer les primitives ASP et les unités PDU sortantes des événements SEND (envoi), et pour spécifier le contenu prévu des primitives ASP et des unités PDU entrantes des événements RECEIVE (réception).

Ces contraintes peuvent être spécifiées sous l'une des deux formes suivantes:

a) contraintes sous forme tabulaire (voir le § 12);

b) contraintes en notation ASN.1 (voir le § 13).

## 11.2 *Principes généraux*

Ce paragraphe décrit les principes généraux régissant l'élaboration des contraintes pour les événements SEND (envoi) et l'étude de la concordance des événements RECEIVE (réception), et en définit les mécanismes. Ces principes sont communs aux deux formes de contraintes, tabulaire et ASN1.

Les contraintes sont des spécifications détaillées des primitives ASP et des unités PDU. Normalement, chaque contrainte est définie spécifiquement aussi bien pour des événements SEND (envoi) que des événements RECEIVE (réception). Une même contrainte peut s'utiliser dans l'un ou l'autre contexte, sous réserve des restrictions de sémantique opératoire définies dans l'annexe B.

La spécification de contrainte d'une primitive ASP ou d'une unité PDU aura la même structure que celle de la définition de type de cette primitive ASP ou de cette unité PDU.

Si une primitive ASP ou une unité PDU possède une sous-structure, les contraintes applicables aux primitives ASP et aux unités PDU de ce type auront la même structure tabulaire ou une structure ASN.1 compatible (c'est-à-dire pouvant comporter des groupages).

Dans une définition de primitive ASP ou d'unité PDU, les types structurés construits à l'aide du macrosymbole (<–) ne sont pas considérés comme des sous-structures. Les contraintes relatives à ces primitives ASP ou à ces unités PDU soit auront une structure complètement plate (les éléments de toute structure développée étant explicitement énumérés dans la contrainte de primitive ASP ou d'unité PDU), soit feront référence à une contrainte de structure correspondante pour un développement de macroinstruction.

Les contraintes spécifient les valeurs des paramètres de primitives ASP et des champs d'unités PDU à l'aide de diverses combinaisons de littéraux, de références d'objets de données, d'expressions, de valeurs construites en notation ASN.1, de mécanismes spéciaux de concordance et de références à d'autres contraintes.

Les contraintes peuvent utiliser les valeurs de tous les types TTCN et ASN.1. Les expressions utilisées dans les contraintes prendront une valeur spécifique lors de l'utilisation de ces contraintes pendant les événements d'envoi et de réception.

De quelque manière qu'on les obtienne, ces valeurs correspondront aux entrées de paramètre ou de champ dans les définitions des types de primitives ASP ou d'unités PDU. En d'autres termes:

a) la valeur sera du type spécifié pour ce paramètre ou champ; et

b) la longueur obéira à toutes les restrictions associées à ce type.

Dans une contrainte, une expression ne comportera que des littéraux, des paramètres de suite de tests, des constantes de suite de tests, des paramètres formels et des opérations de suite de tests.

Pour faciliter le chaînage statique, il est également possible d'utiliser une référence à une contrainte (éventuellement paramétrée) comme valeur de paramètre ou de champ.

Les contraintes n'utiliseront ni des variables de suite de tests, ni des variables de test élémentaire, sauf si ces variables sont transmises comme paramètres effectifs. Dans ce cas, elles seront valuées et ne seront pas modifiées par l'occurrence d'un événement d'envoi ou de réception.

Les mécanismes de concordance sont définis au § 11.6.2.

## 11.3 *Paramétrage des contraintes*

Les contraintes peuvent être paramétrées. Dans ce cas, le nom de la contrainte est suivi de la liste des paramètres formels entre parenthèses. Les paramètres formels sont utilisés pour spécifier les valeurs des paramètres de primitives ASP et des champs d'unités PDU dans cette contrainte.

Chaque nom de paramètre formel sera suivi de deux points «:» et du nom du type du paramètre. Si plusieurs paramètres sont du même type, ils peuvent être regroupés en sous-liste. Dans ce cas, les noms des paramètres du même type seront séparés par une virgule, le dernier étant suivi de deux points «:» et du nom du type commun à la sous-liste. Lorsqu'on utilise plus d'un couple paramètre/type (ou couple sous-liste/type), les couples seront séparés par un point-virgule «;».

Les valeurs littérales, les paramètres de suite de tests, les constantes de suite de tests, les variables de suite de tests, les variables de test élémentaire et les contraintes de type PDU ou suite de tests peuvent être transmis comme paramètres effectifs à une contrainte par une référence de contrainte effectuée depuis une description comportementale. Ces paramètres ne seront pas du type point PCO ou primitive ASP.

## 11.4 *Chaînage des contraintes*

Les contraintes peuvent être chaînées en citant en référence une contrainte comme valeur de paramètre ou de champ dans une autre contrainte. Par exemple, la valeur du paramètre Data (données) d'une primitive ASP de demande de données de réseau N-DATAreq pourrait être une référence à une contrainte d'unité PDU de demande de connexion transport T-CRPDU, c'est-à-dire que la contrainte T-CRPDU est chaînée à la primitive de service restreint.

Le chaînage des contraintes peut s'effectuer de deux manières:

a) par chaînage statique, dans lequel la valeur dans une contrainte d'un paramètre de primitive ASP ou d'un champ d'unité PDU, est une référence explicite à une autre contrainte;

b) par chaînage dynamique, dans lequel la valeur dans une contrainte d'un paramètre de primitive ASP ou d'un champ d'unité PDU, est un paramètre formel de cette contrainte. Lorsqu'une telle référence est appelée dans un comportement dynamique, le paramètre effectif correspondant à cette contrainte est une référence à une autre contrainte (voir les exemples de chaînages statique et dynamique donnés dans l'annexe D).

Une contrainte ne peut faire référence à elle-même de manière récursive (directe ou indirecte).

## 11.5 *Contraintes relatives aux événements SEND (envoi)*

Les contraintes données en référence d'un événement SEND (envoi) ne doivent pas contenir de caractères génériques [c'est-à-dire AnyValue (?) (valeur quelconque) ou AnyOrOmit (*) (quelconque ou omission)], sauf si ceux-ci sont des valeurs spécifiques explicitement affectées dans la ligne d'événement SEND (envoi) de la description du comportement.

Dans une contrainte tabulaire, tous les paramètres de primitive ASP et les champs d'unité PDU sont facultatifs et peuvent donc être omis en utilisant le symbole d'omission pour indiquer que ces paramètres ou ces champs sont absents de l'événement d'envoi.

Dans une contrainte ASN.1, seuls les paramètres de primitive ASP et les champs d'unité PDU déclarés facultatifs peuvent être omis. On peut les omettre soit en utilisant le symbole d'omission, soit simplement en ne mentionnant pas le paramètre de primitive ASP ou le champ d'unité PDU pertinent.

Aucun des mécanismes de concordance définis au § 11.6.2, à l'exception de la valeur spécifique SpecificValue, ne fournit de valeur de paramètre de primitive ASP ou de champ d'unité PDU pour un événement d'envoi.

Lorsqu'on utilise des valeurs ASN.1 du type SET ou SET OF dans une contrainte, les valeurs des éléments de cet ensemble sont envoyées dans l'ordre spécifié par la contrainte correspondante.

## 11.6 *Contraintes relatives aux événements RECEIVE (réception)*

### 11.6.1 *Valeurs de concordance*

Si une contrainte est utilisée pour déterminer les valeurs des paramètres de primitive ASP ou des champs d'unité PDU avec lesquelles une primitive ASP ou une unité PDU reçue doit concorder, cette contrainte ne comportera que des valeurs spécifiques calculées conformément au § 11.6.3, ou des mécanismes spéciaux de concordance lorsqu'il n'est pas souhaitable ou possible de spécifier des valeurs particulières. Les mécanismes de concordance spécifient des concordances autres que la méthode d'égalité à une valeur spécifique.

Une primitive ASP ou une unité PDU entrante concorde avec une contrainte utilisée dans un événement RÉCEPTION si et seulement si tous les paramètres de la primitive ASP ou tous les champs de l'unité PDU sont du type spécifié dans les définitions de la primitive ASP ou de l'unité PDU, que la valeur, le jeu de caractères et la longueur obéissent aux restrictions associées à ce type, et que les valeurs des paramètres de la primitive ASP ou des champs de l'unité PDU concordent correctement avec celles de la contrainte.

Dans le cas de primitives ASP ou d'unités PDU munies de sous-structures, qu'elles soient définies par des types structurés ou en ASN.1, les règles ci-dessus s'appliquent récursivement aux champs de la ou des sous-structures.

*Remarque* – Si un événement RÉCEPTION est qualifié par une expression booléenne, il y aura concordance si la primitive ASP ou l'unité PDU concorde avec la contrainte et si le qualificateur prend la valeur VRAI (TRUE).

### 11.6.2 *Mécanismes de concordance*

Le tableau 5/X.292 donne un aperçu général des mécanismes de concordance pris en charge, ainsi que des symboles spéciaux et de leur domaine d'application. La colonne de gauche donne la liste des types ASN.1 et des types TTCN équivalents auxquels ces mécanismes de concordance s'appliquent. Les différents types de mécanismes de concordance, correspondant aux lignes du tableau, sont subdivisés verticalement en quatre groupes:

a) spécificateurs de valeurs;

b) spécificateurs de *substitution* (à des valeurs);

c) spécificateurs d'*appartenance* (à une valeur);

d) spécificateurs d'*attributs* (de valeurs).

Certains de ces spécificateurs peuvent être combinés comme indiqué dans les paragraphes suivants.

La partie ombrée du tableau 5/X.292 indique les mécanismes qui s'appliquent à la fois aux types prédéfinis TTCN et ASN.1.

Dans une spécification de contrainte, les mécanismes de concordance peuvent remplacer des valeurs de certains paramètres de primitive ASP ou champs d'unité PDU, voire le contenu entier d'une primitive ASP ou d'une unité PDU.

*Remarque* – Lorsque ces mécanismes de concordance sont utilisés seuls ou combinés, de nombreuses restrictions de protocole peuvent être spécifiées dans les contraintes, permettant ainsi d'éviter des détails de calcul inutiles dans la partie comportement.

TABLEAU 5/X.292

**Mécanismes de concordance en notation TTCN**

| | VALEUR | SPECIFICATEURS DE SUBSTITUTION | | | | | | | | SPECIFICATEURS D'APPARTENANCE | | | SPECIFICATEURS D'ATTRIBUTS | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TYPE | Specific Value (Valeur spécifique) | Complement | Omit (–) | AnyValue (?) | AnyOrOmit (*) | ValueList | Range | SuperSet | SubSet | AnyOne (?) | AnyOrNone (*) | Permutation | Length | IfPresent |
| BOOLEAN | • | • | • | • | • | • | | | | | | | | • |
| INTEGER | • | • | • | • | • | • | • | | | | | | | • |
| ENUMERATED | • | • | • | • | • | • | | | | | | | | • |
| BITSTRING | • | • | • | • | • | • | | | | • | • | | • | • |
| OCTETSTRING | • | • | • | • | • | • | | | | • | • | | • | • |
| HEXSTRING | • | • | • | • | • | • | | | | • | • | | • | • |
| CHARSTRINGS | • | • | • | • | • | • | | | | • | • | | • | • |
| SEQUENCE | • | • | • | • | • | • | | | | | | | | • |
| SEQUENCE OF | • | • | • | • | • | • | | | | • | • | • | • | • |
| SET | • | • | • | • | • | • | | | | | | | | • |
| SET OF | • | • | • | • | • | • | | • | • | • | • | | • | • |
| ANY | • | • | • | • | • | • | | | | | | | | • |
| CHOICE | • | • | • | • | • | • | | | | | | | | • |
| OBJECT ID | • | • | • | • | • | • | | | | | | | | • |

11.6.3 *Spécificateurs de valeurs*

Il s'agit du mécanisme de concordance de base. Les valeurs spécifiques indiquées dans les contraintes sont des expressions. Sauf spécification contraire, un paramètre de primitive ASP ou un champ d'unité PDU d'une contrainte concorde avec le paramètre correspondant de la primitive ASP entrante ou le champ correspondant de l'unité PDU entrante si et seulement si la valeur de ce paramètre de primitive ASP entrante ou de champ d'unité PDU entrante est identique à la valeur prise par l'expression de la contrainte.

Deux valeurs d'un type tabulaire de primitive ASP, d'unité PDU ou de structure, ou deux valeurs ASN.1 du type SEQUENCE ou SEQUENCE OF sont considérées comme identiques si tous leurs paramètres, champs ou éléments concordent et apparaissent dans le même ordre. Deux valeurs ASN.1 de type SET ou SET OF sont identiques si elles ont le même nombre d'éléments et si chacun des éléments d'une valeur concorde exactement avec un élément de l'autre et réciproquement. Les éléments de deux valeurs de type SET ou SET OF ne doivent pas obligatoirement apparaître dans le même ordre pour concorder.

11.6.4 *Spécificateurs de substitution*

11.6.4.1 *Complement (complément)*

Complement est une spécification de concordance qui peut s'appliquer à des valeurs de tous types. Cette opération est indiquée par le mot clé COMPLEMENT (complément) suivi de la liste des valeurs des contraintes. Chaque valeur de contrainte de cette liste doit être du type déclaré pour le paramètre de primitive ASP ou le champ d'unité PDU pour lequel le mécanisme Complement est utilisé.

*Définition syntaxique*

201 Complement ::= **COMPLEMENT** ValueList

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU spécifiée par le mécanisme Complement concorde avec le paramètre de primitive ASP ou le champ d'unité PDU correspondant si et seulement si le paramètre de primitive ASP entrante ou le champ d'unité PDU entrante ne concorde avec aucune des valeurs de la liste ValueList.

*Exemple 28* – Contraintes utilisant le mécanisme Complement à la place d'une valeur avec une liste des valeurs

| *Type* | *Contrainte* |
|---|---|
| INTEGER | COMPLEMENT (5) |
| INTEGER | COMPLEMENT (1, 3, 5) |

11.6.4.2 *Omit (omission)*

Omit est un spécificateur spécial de concordance qui peut s'appliquer à des valeurs de tous types, à condition que le paramètre de primitive ASP ou le champ d'unité PDU soit facultatif.

Dans les contraintes ASN.1, il est également possible d'omettre un paramètre de primitive ASP ou un champ d'unité PDU de type OPTIONAL (facultatif) au lieu d'utiliser explicitement le symbole OMIT.

Dans les contraintes tabulaires, le symbole Omit est représenté par un tiret (–). Dans les contraintes ASN.1, il est indiqué par **OMIT**.

*Définition syntaxique*

```
202  Omit ::= Dash | OMIT
```

Dans une contrainte, un symbole Omit indique qu'un paramètre de primitive ASP ou un champ d'unité PDU facultatif sera absent.

*Exemple 29* – Contraintes utilisant Omit en lieu et place d'une valeur, au niveau supérieur

| *Type* | *Contrainte* |
|---|---|
| INTEGER OPTIONAL | OMIT |

11.6.4.3 *AnyValue (valeur quelconque)*

AnyValue est un spécificateur spécial de concordance qui peut s'appliquer à des valeurs de tous types. Dans les contraintes tant ASN.1 que tabulaires, AnyValue est indiqué par le signe «?».

*Définition syntaxique*

```
203  AnyValue ::= "?"
```

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU utilisant ce symbole concorde avec le paramètre de primitive ASP entrante ou le champ d'unité PDU entrante correspondant si et seulement si ce paramètre ou ce champ entrant est valué par un élément simple du type spécifié.

*Exemple 30* – Contraintes utilisant une combinaison de valeurs spécifiques et de symboles AnyValue

| ***Type*** | ***Contrainte*** |
|---|---|
| SEQUENCE OF SET OF INTEGER | { {1, 2},<br>?,<br>{1, 2, ?} } |

11.6.4.4 *AnyOrOmit (élément quelconque ou omission)*

AnyOrOmit est un spécificateur spécial de concordance applicable à des valeurs de tous types, à condition que le paramètre de primitive ASP ou le champ d'unité PDU soit déclaré comme facultatif. Dans les contraintes tant ASN.1 que tabulaires, AnyOrOmit est indiqué par le signe «*».

*Remarque* – Le symbole «*» est utilisé pour les deux symboles AnyOrOmit et AnyOrNone (zéro ou plusieurs éléments). Cette ambiguïté d'interprétation est levée par les spécifications des § 11.6.4.4 et 11.6.5.2.

*Définition syntaxique*

```
204  AnyOrOmit ::= "*"
```

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU utilisant le symbole AnyOrOmit concorde avec le paramètre de primitive ASP entrante ou le champ d'unité PDU entrante correspondant si et seulement si ce paramètre ou ce champ entrant est absent ou prend la valeur d'un élément quelconque du type spécifié.

*Exemple 31* – Contraintes utilisant une combinaison de valeurs spécifiques et du symbole AnyOrOmit

| ***Type*** | ***Contrainte*** |
|---|---|
| SEQUENCE OF { id1 SET OF INTEGER<br>id2 SET OF INTEGER } | { id1 {2, 5},<br>id2 * } |

11.6.4.5 *ValueList (liste de valeurs)*

La spécification par liste ValueList peut s'utiliser avec des valeurs de tous types. Dans les contraintes tant ASN.1 que tabulaires, ces listes apparaissent sous forme d'une série de valeurs séparées par des virgules et placées entre parenthèses.

*Définition syntaxique*

```
205  ValueList ::=  "("ConstraintValue&Attributes {Comma ConstraintValue&Attributes}")"
```

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU utilisant une liste de valeurs concorde avec le paramètre de primitive ASP entrante ou le champ d'unité PDU entrante correspondant si et seulement si ce paramètre ou ce champ entrant concorde avec l'une quelconque des valeurs de la liste des valeurs ValueList. Chaque valeur de la liste ValueList doit appartenir au type déclaré pour le paramètre de primitive ASP ou le champ d'unité PDU pour lequel le mécanisme ValueList est utilisé.

*Exemple 32* – Contrainte utilisant le mécanisme ValueList en lieu et place d'une valeur spécifique, pour un type entier

| ***Type*** | ***Contrainte*** |
|---|---|
| INTEGER | (2, 4, 6) |

*Exemple 33* – Contrainte utilisant le mécanisme ValueList en lieu et place d'une valeur spécifique pour un type CHOICE

| ***Type*** | ***Contrainte*** |
|---|---|
| CHOICE {a INTEGER,<br>b BOOLEAN} | (a 2, b TRUE) |

11.6.4.6 *Range (intervalle)*

La spécification par intervalles Range ne s'applique qu'à des valeurs de type entier. Un intervalle est indiqué par deux limites, séparées par le signe «..» ou le mot TO (à), le tout entre parenthèses. Une limite sera:

a) soit INFINITY ou –INFINITY ($\pm\infty$);

b) soit une expression de contrainte de valeur entière.

La limite inférieure sera inscrite à la gauche du symbole «..» ou TO, la limite supérieure à droite. La limite inférieure doit être inférieure à la limite supérieure.

*Définition syntaxique*

```
206  ValueRange ::= "(" ValRange ")"
207  ValRange ::= (LowerRangeBound To UpperRangeBound)
208  LowerRangeBound ::= ConstraintExpression | Minus INFINITY
209  UpperRangeBound ::= ConstraintExpression | INFINITY
```

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU spécifiée par un intervalle correspond au paramètre de la primitive ASP entrante ou au champ d'unité PDU entrante correspondant si et seulement si la valeur de ce paramètre ou de ce champ entrant est égale à l'une de celles de l'intervalle.

*Exemple 34* – Contraintes utilisant le mécanisme d'intervalle Range en lieu et place d'une valeur

| ***Type*** | ***Contrainte*** |
|---|---|
| INTEGER | (1 .. 6)<br>(–INFINITY .. 8)<br>(12 .. INFINITY) |

11.6.4.7 *SuperSet (surensemble)*

La spécification par surensemble est un mécanisme de concordance qui ne s'applique qu'à des valeurs du type SET OF (ensemble de). Ce mécanisme ne sera utilisé que pour des contraintes ASN.1. Il est indiqué par la mention **SUPERSET**.

*Définition syntaxique*

210 SuperSet ::= **SUPERSET** "(" ConstraintValue&Attributes ")"

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU spécifiée par un surensemble concorde avec le paramètre de la primitive ASP entrante ou au champ de l'unité PDU entrante correspondant si et seulement si ce paramètre ou ce champ entrant contient tous les éléments du surensemble, plus éventuellement d'autres. L'argument du spécificateur SuperSet doit être du type déclaré pour le paramètre de la primitive ASP ou le champ de l'unité PDU pour lequel ce mécanisme est utilisé.

*Exemple 35* – Contrainte spécifiée par surensemble en lieu et place d'une valeur spécifique

| *Type* | *Contrainte* |
|---|---|
| SET OF INTEGER | SUPERSET ({1, 2, 3}) |

11.6.4.8 *SubSet (sous-ensemble)*

La spécification par sous-ensemble est un mécanisme de concordance qui ne s'applique qu'aux valeurs du type SET OF. Ce mécanisme ne sera utilisé que pour des contraintes ASN.1. Il est indiqué par la mention **SUBSET**.

*Définition syntaxique*

211 SubSet ::= **SUBSET** "(" ConstraintValue&Attributes ")"

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU spécifiée par sous-ensemble concorde avec le paramètre de la primitive ASP entrante ou le champ de l'unité PDU entrante correspondant si et seulement si ce paramètre ou ce champ entrant ne contient que des éléments du sous-ensemble, et pas nécessairement tous. L'argument du spécificateur Subset doit être du type déclaré pour le paramètre de primitive ASP ou le champ d'unité PDU pour lequel ce mécanisme est utilisé.

*Exemple 36* – Contrainte spécifiée par sous-ensemble en lieu et place d'une valeur spécifique

| *Type* | *Contrainte* |
|---|---|
| SET OF INTEGER | SUBSET ({2, 4, 6, 8, 10}) |

11.6.5 *Spécificateurs d'appartenance*

11.6.5.1 *AnyOne (un quelconque)*

AnyOne est un spécificateur spécial de concordance qui peut s'utiliser avec des valeurs des types chaînes, SEQUENCE OF et SET OF. Dans les contraintes tant ASN.1 que tabulaires, ce spécificateur est représenté par le signe «?».

*Définition syntaxique*

351 AnyOne ::= "?"

Dans une valeur de type chaîne, SEQUENCE OF ou SET OF, le signe «?» mis à la place d'un élément simple indique que tout élément est acceptable à cette position. Dans une chaîne CharacterString, le caractère «?» (point d'interrogation) sera représenté par le signe «\?»; quant au caractère «\» (barre oblique inverse), il sera représenté par le signe «\\».

*Exemple 37* – Contrainte utilisant AnyOne

| *Type* | *Contrainte* |
|---|---|
| IA5String | "a?cd" |
| SEQUENCE OF INTEGER | {1, 2, ?} |

*Remarque* – Dans le second exemple, le signe «?» peut être interprété comme le spécificateur «valeur quelconque» (AnyValue) représentant une valeur entière quelconque au niveau de la contrainte ou comme le spécificateur «un quelconque» (AnyOne) représentant une valeur quelconque parmi les valeurs de la suite d'entiers SEQUENCE OF INTEGER. Ces interprétations aboutissant toutes deux au même ensemble d'événements mis en concordance, l'ambiguïté ne pose pas de problème.

11.6.5.2 *AnyOrNone (quelconque ou aucun)*

AnyOrNone est un spécificateur spécial de concordance qui peut s'utiliser avec des valeurs des types chaînes, SEQUENCE OF et SET OF. Dans les contraintes tant ASN.1 que tabulaires, ce spécificateur est représenté par le signe «*».

Si le symbole «*» apparaît au niveau supérieur d'une valeur de type chaîne, SEQUENCE OF ou SET OF, il est interprété comme le spécificateur AnyOrNone.

*Remarque* – Cette règle évite l'interprétation possible du signe «*» en tant que symbole de AnyOrOmit, qui remplace un élément d'une valeur de type chaîne, SEQUENCE OF ou SET OF.

*Définition syntaxique*

352 AnyOrNone ::= "*"

Dans une valeur de type chaîne, SEQUENCE OF ou SET OF, un signe «*» mis à la place d'un élément simple remplace zéro ou plusieurs éléments consécutifs. Le mécanisme de concordance fait correspondre au symbole «*» la plus longue séquence possible d'éléments, conformément au modèle spécifié par les symboles entourant le signe «*». Dans une chaîne CharacterString, le caractère «*» (astérisque) est représenté par le signe «*»; quant au caractère «\» (barre oblique inverse), il est représenté par le signe «\\».

*Exemple 38* – Contraintes utilisant le spécificateur AnyOrNone

| *Type* | *Contrainte* |
|---|---|
| IA5String | "ab*z" |
| SEQUENCE OF INTEGER | {1, 2, *, 10} |
| SEQUENCE OF IA5String | { "ab*z",<br>*,<br>"abc"} |

11.6.5.3 *Permutation*

La Permutation est une opération de concordance qui ne s'applique qu'aux éléments d'une valeur de type SEQUENCE OF. Elle ne s'utilise qu'avec des contraintes ASN.1. Elle est indiquée par le spécificateur **PERMUTATION**.

*Définition syntaxique*

212 Permutation ::= **PERMUTATION** ValueList

La mention Permutation, en lieu et place d'un élément simple, signifie que cet élément peut prendre pour valeur n'importe quelle permutation des éléments de la liste de valeurs ValueList. Si les spécificateurs Permutation et AnyOrNone apparaissent tous deux dans une valeur, le spécificateur AnyOrNone sera évalué en premier. Chacun des éléments cités en argument de la Permutation doit être du type déclaré dans le type SEQUENCE OF du paramètre de la primitive ASP ou du champ de l'unité PDU.

*Exemple 39* – Contrainte spécifiée par Permutation

| *Type* | *Contrainte* |
|---|---|
| SEQUENCE OF INTEGER | {PERMUTATION (1, 2, 3), 5} |

*Exemple 40* – Contraintes spécifiées par une combinaison de Permutation et AnyOrNone

| *Type* | *Contrainte* |
|---|---|
| SEQUENCE OF INTEGER | {PERMUTATION (1, 2, 3), *}<br>{PERMUTATION (1, 2, 3, *)} |

A noter que la première contrainte concorde avec toute primitive ASP ou unité PDU entrante composée d'une séquence de valeurs entières, commençant par une des six permutations: 1,2,3; 1,3,2; 2,1,3; 2,3,1; 3,1,2; ou 3,2,1 et suivie d'un nombre quelconque d'entiers. La seconde contrainte concorde avec toute primitive ASP ou unité PDU entrante de type suite d'entiers contenant les éléments 1, 2 et 3 en position quelconque. Elle concorde par exemple, avec les suites {5,2,7,1,3} et {9,3,7,2,12,1,17}.

### 11.6.6 *Attributs de valeurs*

#### 11.6.6.1 *Length (longueur)*

La spécification de longueur est une opération de concordance qui ne s'utilise comme attribut que pour les mécanismes suivants: Specific Value (valeur spécifique), Complement, Omit, AnyValue, AnyOrOmit, AnyOne, AnyOrNone et Permutation.

Dans les contraintes tant ASN.1 que tabulaires, la longueur peut être spécifiée sous forme d'une valeur exacte ou d'un intervalle de valeurs de type chaîne, SEQUENCE OF et SET OF conformément au § 10.12. Les unités de longueur seront interprétées selon le tableau 4/X.292. Les limites seront des valeurs entières non négatives. Le mot clé INFINITY peut également servir de limite supérieure pour indiquer que la longueur maximale est illimitée.

Une spécification de longueur définie pour un type de paramètre de primitive ASP ou de champ d'unité PDU dans une définition de type de suite de tests respectera les spécifications de longueur de la contrainte de la primitive ASP ou de l'unité PDU, c'est-à-dire que l'ensemble de chaînes défini dans une contrainte de primitive ASP ou d'unité PDU par restriction de longueur doit former un sous-ensemble strict de l'ensemble de chaînes correspondant à la définition de la primitive ASP ou de l'unité PDU.

*Définition syntaxique*

```
214  ValueLength ::= SingleValueLength | RangeValueLength
215  SingleValueLength ::= "[" ValueBound "]"
216  ValueBound ::= Number | TS_ParIdentifier | TS_ConstIdentifier | FormalParIdentifier
217  RangeValueLength ::= "[" LowerValueBound To UpperValueBound "]"
218  LowerValueBound ::= ValueBound
37   To ::= TO | ".."
219  UpperValueBound ::= ValueBound | INFINITY
```

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU utilisant le mécanisme de spécification de l'attribut Length concorde avec le paramètre de la primitive ASP entrante ou avec le champ de l'unité PDU entrante correspondant si et seulement si ce paramètre ou ce champ entrant concorde avec la valeur de la contrainte tout en respectant la spécification de l'attribut associé. La valeur de l'attribut de longueur est considérée comme acceptable si la longueur du paramètre ou du champ entrant appartient à l'intervalle spécifié (limites comprises) lorsque la spécification d'attribut est donnée sous forme d'intervalle, ou est égale à la valeur spécifiée lorsque la spécification d'attribut est donnée sous forme de valeur unique.

*Exemple 41* – Contrainte utilisant une combinaison de spécification d'appartenance et d'attribut de longueur

| *Type* | *Contrainte* |
|---|---|
| IA5String | "ab*ab" [13] |

#### 11.6.6.2 *IfPresent (si présent)*

IfPresent est un spécificateur spécial de concordance pouvant être utilisé comme attribut de tous les mécanismes de concordance, à condition que le type soit déclaré comme facultatif. Dans les contraintes tant ASN.1 que tabulaires, cette spécification est indiquée par le spécificateur **IF_PRESENT**.

Une contrainte de paramètre de primitive ASP ou de champ d'unité PDU utilisant le spécificateur IfPresent comme attribut d'un autre spécificateur concorde avec le paramètre de la primitive ASP entrante ou le champ de l'unité PDU entrante correspondant si et seulement si ce paramètre ou ce champ entrant concorde avec la contrainte ou est absent.

*Remarque* – Le spécificateur AnyOrOmit (*) a exactement la même signification que le spécificateur ? muni de l'attribut IF_PRESENT.

*Exemple 42* – Contrainte utilisant une valeur combinée avec le spécificateur IfPresent

| *Type* | *Contrainte* |
|---|---|
| IA5String OPTIONAL | "abcdef" IF_PRESENT |

# 12 Spécification des contraintes à l'aide de tables

## 12.1 *Introduction*

Le présent paragraphe a pour objet de décrire la spécification sous forme tabulaire de contraintes imposées à des primitives ASP et à des unités PDU. Il indique comment utiliser des tables de contraintes simples pour imposer des contraintes à des primitives ASP ou à des unités PDU plates (non structurées) et comment spécifier des contraintes structurées en définissant des contraintes relatives à des types structurés déclarés dans la section types des suites de tests.

L'annexe C définit des tables complémentaires permettant de regrouper plusieurs définitions de contraintes dans un seul tableau.

## 12.2 *Déclarations de contraintes de structures*

Si une primitive ASP ou une unité PDU est définie à l'aide de types structurés, soit par développement de macros soit par des sous-structures, les contraintes imposées seront sous-structurées de la même façon. Les tables des contraintes structurées sont très proches de celles des contraintes d'unité PDU. La valeur des éléments des contraintes structurées sera fournie dans le format illustré dans le formulaire suivant:

| **Déclaration de contrainte de type structuré** | | |
|---|---|---|
| **Nom de la contrainte :** *ConsId&ParList*<br>**Type structuré :** *StructIdentifier*<br>**Chemin de dérivation :** *[DerivationPath]*<br>**Commentaires :** *[FreeText]* | | |
| **Nom de l'élément** | **Valeur de l'élément** | **Commentaires** |
| .<br>.<br>*ElemIdentifier*<br>.<br>. | .<br>.<br>*ConstraintValue&Attributes*<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | |

**Formulaire 24 – Déclaration de contrainte de type structuré**

*Définition syntaxique*

```
190  ConsId&ParList ::= ConstraintIdentifier [FormalParList]
191  ConstraintIdentifier ::= Identifier
44   StructIdentifier ::= Identifier
193  DerivationPath ::= {ConstraintIdentifier Dot}+
49   ElemIdentifier ::= Identifier
197  ConstraintValue&Attributes ::= ConstraintValue ValueAttributes
198  ConstraintValue ::= ConstraintExpression | MatchingSymbol | ConsRef
199  ConstraintExpression ::= Expression
200  MatchingSymbol ::= Complement | Omit | AnyValue | AnyOrOmit | ValueList | ValueRange |
     SuperSet | SubSet | Permutation
277  ConsRef ::= ConstraintIdentifier [ActualCrefParList]
278  ActualCrefParList ::= "(" ActualCrefPar {Comma ActualCrefPar} ")"
279  ActualCrefPar ::= Value | DataObjectIdentifier | ConsRef
213  ValueAttributes ::= [ValueLength] [IF_PRESENT]
214  ValueLength ::= SingleValueLength | RangeValueLength
215  SingleValueLength ::= "[" ValueBound "]"
216  ValueBound ::= Number | TS_ParIdentifier | TS_ConstIdentifier | FormalParIdentifier
217  RangeValueLength ::= "[" LowerValueBound To UpperValueBound "]"
218  LowerValueBound ::= ValueBound
37   To ::= TO | ".."
219  UpperValueBound ::= ValueBound | INFINITY
```

Ce formulaire est utilisé pour les structures et leurs éléments de la même manière que le formulaire de déclaration de contrainte d'unité PDU l'est pour les unités PDU et leurs champs (voir le § 12.4).

Si une définition de primitive ASP ou d'unité PDU se réfère à un type structuré comme sous-structure de paramètre ou de champ (c'est-à-dire avec le nom du paramètre ou du champ spécifié pour cette sous-structure), la contrainte correspondante indiquera alors le même nom de paramètre ou de champ dans la position correspondante de la colonne nom du paramètre ou nom du champ, et sa valeur sera une référence à une contrainte portant sur ce paramètre ou ce champ (c'est-à-dire sur cette sous-structure, conformément à la définition du type structuré). Si la définition de la primitive ASP ou de l'unité PDU se réfère à un paramètre ou à un champ spécifié comme étant du métatype unité PDU, la valeur de ce paramètre ou champ sera alors spécifiée dans une contrainte correspondante comme le nom d'une contrainte d'unité PDU ou comme paramètre formel.

## 12.3 *Déclaration des contraintes de primitive ASP*

Les valeurs des paramètres des contraintes de primitive ASP seront fournies dans le format illustré dans le formulaire suivant:

| **Déclaration de contrainte de primitive ASP** | | |
|---|---|---|
| **Nom de la contrainte :** *ConsId&ParList*<br>**Type de la primitive :** *ASP_Identifier*<br>**Chemin de dérivation :** *[DerivationPath]*<br>**Commentaires :** *[FreeText]* | | |
| **Nom du paramètre** | **Valeur du paramètre** | **Commentaires** |
| .<br>.<br>*ASP_ParIdOrMacro*<br>.<br>. | .<br>.<br>*ConstraintValue&Attributes*<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | |

**Formulaire 25 – Déclaration de contrainte de primitive ASP**

*Définition syntaxique*

```
190  ConsId&ParList ::= ConstraintIdentifier [FormalParList]
191  ConstraintIdentifier ::= Identifier
128  ASP_Identifier ::= Identifier
193  DerivationPath ::= {ConstraintIdentifier Dot}+
132  ASP_ParIdOrMacro ::= ASP_ParId&FullId | MacroSymbol
133  ASP_ParId&FullId ::= ASP_ParIdentifier [FullIdentifier]
134  ASP_ParIdentifier ::= Identifier
150  MacroSymbol ::= "<–"
197  ConstraintValue&Attributes ::= ConstraintValue ValueAttributes
198  ConstraintValue ::= ConstraintExpression | MatchingSymbol | ConsRef
199  ConstraintExpression ::= Expression
200  MatchingSymbol ::= Complement | Omit | AnyValue | AnyOrOmit | ValueList | ValueRange |
     SuperSet | SubSet | Permutation
277  ConsRef ::= ConstraintIdentifier [ActualCrefParList]
278  ActualCrefParList ::= "(" ActualCrefPar {Comma ActualCrefPar} ")"
279  ActualCrefPar ::= Value | DataObjectIdentifier | ConsRef
213  ValueAttributes ::= [ValueLength] [IF_PRESENT]
214  ValueLength ::= SingleValueLength | RangeValueLength
215  SingleValueLength ::= "[" ValueBound "]"
216  ValueBound ::= Number | TS_ParIdentifier | TS_ConstIdentifier | FormalParIdentifier
217  RangeValueLength ::= "[" LowerValueBound To UpperValueBound "]"
218  LowerValueBound ::= ValueBound
37   To ::= TO | ".."
219  UpperValueBound ::= ValueBound | INFINITY
```

Ce formulaire est utilisé pour spécifier les contraintes portant sur les primitives ASP et leurs paramètres de la même manière qu'un formulaire de déclaration de contrainte d'unité PDU (voir le § 12.4).

### 12.4 *Déclaration des contraintes d'unité PDU*

En format tabulaire, une contrainte est définie par la spécification d'une valeur et d'attributs facultatifs pour chaque champ d'unité PDU. Les informations suivantes seront fournies pour chaque contrainte d'unité PDU:

a) le nom de la contrainte, éventuellement suivi d'une liste facultative de paramètres formels;

b) le nom du type de l'unité PDU;

c) son chemin de dérivation (voir le § 12.6);

d) une valeur de contrainte pour chaque champ, donnant les informations suivantes pour chaque champ:

1) son nom,

chaque entrée de champ dans la colonne nom du champ devant avoir été déclarée dans la définition correspondante du type d'unité PDU. Si l'un des champs de l'unité PDU a été défini comme ayant un nom abrégé et un identificateur complet, la contrainte ne répétera pas l'identificateur complet;

si la définition de l'unité PDU se réfère à un type structuré résultant d'un développement de macro (c'est-à-dire dans laquelle le signe «<–» remplace le nom d'un champ, la contrainte correspondante prendra l'une des formes suivantes:

– ou bien chacun des champs correspondants au type structuré apparaîtra de manière explicite dans la contrainte;

– ou alors le macrosymbole (<–) sera placé dans la position correspondante de la colonne nom du champ d'unité PDU de la contrainte et sa valeur fera référence à une contrainte appliquée au type structuré mentionné dans la définition de l'unité PDU.

On n'utilisera pas de contraintes structurées par développement de macro, sauf si la définition de l'unité PDU correspondante fait elle aussi référence au même type structuré par développement de macro;

2) sa valeur et un attribut facultatif.

Ces informations doivent être fournies selon le format de formulaire suivant:

| **Déclaration de contrainte d'unité PDU** | | |
|---|---|---|
| **Nom de la contrainte :** *ConsId&ParList*<br>**Type de l'unité :** *PDU_Identifier*<br>**Chemin de dérivation :** *[DerivationPath]*<br>**Commentaires :** *[FreeText]* | | |
| **Nom du champ** | **Valeur du champ** | **Commentaires** |
| .<br>.<br>*PDU_FieldIdOrMacro*<br>.<br>. | .<br>.<br>*ConstraintValue&Attributes*<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | |

**Formulaire 26 – Déclaration de contrainte d'unité PDU**

*Définition syntaxique*

190 ConsId&ParList ::= ConstraintIdentifier [FormalParList]
191 ConstraintIdentifier ::= Identifier
145 PDU_Identifier ::= Identifier
193 DerivationPath ::= {ConstraintIdentifier Dot}+
149 PDU_FieldIdOrMacro ::= PDU_FieldId&FullId | MacroSymbol
151 PDU_FieldId&FullId ::= PDU_FieldIdentifier [FullIdentifier]
152 PDU_FieldIdentifier ::= Identifier
150 MacroSymbol ::= "<–"
197 ConstraintValue&Attributes ::= ConstraintValue ValueAttributes
198 ConstraintValue ::= ConstraintExpression | MatchingSymbol | ConsRef
199 ConstraintExpression ::= Expression
200 MatchingSymbol ::= Complement | Omit | AnyValue | AnyOrOmit | ValueList | ValueRange | SuperSet | SubSet | Permutation
277 ConsRef ::= ConstraintIdentifier [ActualCrefParList]
278 ActualCrefParList ::= "(" ActualCrefPar {Comma ActualCrefPar} ")"
279 ActualCrefPar ::= Value | DataObjectIdentifier | ConsRef
213 ValueAttributes ::= [ValueLength] [**IF_PRESENT**]
214 ValueLength ::= SingleValueLength | RangeValueLength
215 SingleValueLength ::= "[" ValueBound "]"
216 ValueBound ::= Number | TS_ParIdentifier | TS_ConstIdentifier | FormalParIdentifier
217 RangeValueLength ::= "[" LowerValueBound To UpperValueBound "]"
218 LowerValueBound ::= ValueBound
37 To ::= **TO** | ".."
219 UpperValueBound ::= ValueBound | **INFINITY**

*Exemple 43* – Contrainte C1 agissant sur l'unité PDU_A

| **Déclaration de contrainte d'unité PDU** | | |
|---|---|---|
| **Nom de la contrainte :** C1<br>**Type de l'unité :** PDU_A<br>**Chemin de dérivation :**<br>**Commentaires :** | | |
| **Nom du champ** | **Valeur du champ** | **Commentaires** |
| FIELD1<br>FIELD2<br>FIELD3 | (4 .. INFINITY)<br>TRUE<br>«CHAÎNE QUELCONQUE»<br>"A STRING" | |

12.5 *Paramétrage des contraintes*

Les contraintes peuvent être paramétrées à l'aide d'une liste de paramètres formels. Les paramètres effectifs sont transmis à une contrainte à partir d'une référence à une contrainte dans une description comportementale.

*Exemple 44* – Contrainte paramétrée

| **Déclaration de contrainte d'unité PDU** | | |
|---|---|---|
| **Nom de la contrainte :** C1<br>**Type de l'unité :** PDU_A<br>**Chemin de dérivation :**<br>**Commentaires :** | | |
| **Nom du champ** | **Valeur du champ** | **Commentaires** |
| FIELD1<br>FIELD2<br>FIELD3 | (4 .. INFINITY)<br>TRUE<br>«CHAÎNE QUELCONQUE»<br>"A STRING" | |

12.6 *Contraintes de base et contraintes modifiées*

Au moins une contrainte de base sera spécifiée pour chaque type défini d'unité PDU. Une contrainte de base spécifie un ensemble de valeurs de base ou par défaut pour chacun des champs définis. Une unité PDU peut comporter un nombre quelconque de contraintes de base (voir les exemples donnés dans l'annexe D).

Lorsqu'une contrainte est spécifiée comme étant une modification d'une contrainte de base, tous les champs non spécifiés à nouveau prennent par défaut la valeur des champs correspondants de la contrainte de base. Le nom de la contrainte modifiée sera un identificateur unique. Le nom de la contrainte de base dont elle dérive sera indiqué à la rubrique chemin de dérivation de l'en-tête de la contrainte. Cette rubrique est muette pour une contrainte de base. Une contrainte modifiée peut être modifiée à son tour. Dans ce cas, le chemin de dérivation indiquera les noms concaténés et séparés par des points ( . ) des contraintes de base et des contraintes successives déjà modifiées. Le nom de la dernière contrainte modifiée sera suivi d'un point. Les règles de formation d'une contrainte modifiée à partir d'une contrainte de base sont les suivantes:

a) si un paramètre ou un champ et sa valeur correspondante ne sont pas spécifiés dans la contrainte, ce paramètre ou ce champ prendra alors la valeur de la contrainte mère (c'est-à-dire que la valeur est héritée);

b) si un paramètre ou un champ et sa valeur correspondante sont spécifiés dans la contrainte, la valeur spécifiée remplace alors la valeur du paramètre ou du champ correspondant de la contrainte mère.

### 12.7 *Listes de paramètres formels dans les contraintes modifiées*

Si une contrainte de base est définie comme ayant une liste de paramètres formels, les règles suivantes s'appliqueront à toutes les contraintes modifiées dérivées de cette contrainte de base, que leur modification résulte d'une ou plusieurs dérivations successives:

a) la contrainte modifiée aura la même liste de paramètres formels que la contrainte de base. Aucun paramètre ne sera notamment omis ou ajouté;

b) la liste des paramètres formels suivra le nom de chaque contrainte modifiée;

c) les paramètres de primitive ASP et les unités PDU apparaissant dans les champs d'une contrainte de base ne doivent pas être modifiés ou explicitement omis dans une contrainte modifiée.

## 13 Spécification des contraintes à l'aide de l'ASN.1

### 13.1 *Introduction*

Ce paragraphe décrit une méthode pour spécifier les contraintes en ASN.1, d'une manière similaire à la définition des contraintes tabulaires. La déclaration normale des valeurs ASN.1 est étendue pour permettre l'utilisation de caractères génériques et de mécanismes de permutation. Des mécanismes assurant la substitution ou l'omission de parties de contraintes ASN.1 sont définis pour les besoins des contraintes modifiées.

### 13.2 *Déclaration des contraintes des types ASN.1*

Les contraintes des primitives ASP et des unités PDU en notation ASN.1 peuvent être structurées par des références à des contraintes de type suites de tests ASN.1 pour les valeurs de champs complexes. Les types suites de tests ASN.1 sont définis dans la section déclarations de la suite ATS.

Les tables des contraintes des types ASN.1 sont très proches de celles des primitives ASP. Les déclarations des contraintes des types ASN.1 seront données dans le format de formulaire suivant:

| **Déclaration de contrainte d'un type ASN.1** |
|---|
| **Nom de la contrainte :** *ConsId&ParList*<br>**Type structuré :** *ASN1_TypeIdentifier*<br>**Chemin de dérivation :** *[DerivationPath]*<br>**Commentaires :** *[FreeText]* |
| **Valeur de la contrainte** |
| .<br>.<br>C*onstraintValue&AttributesOrReplace*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* |

**Formulaire 27 – Déclaration d'une contrainte d'un type en ASN.1**

*Définition syntaxique*

```
190  ConsId&ParList ::= ConstraintIdentifier [FormalParList]
191  ConstraintIdentifier ::= Identifier
55   ASN1_TypeIdentifier ::= Identifier
193  DerivationPath ::= {ConstraintIdentifier Dot}+
223  ConstraintValue&AttributesOrReplace ::= ConstraintValue&Attributes | Replacement
     {Comma Replacement}
197  ConstraintValue&Attributes ::= ConstraintValue ValueAttributes
198  ConstraintValue ::= ConstraintExpression | MatchingSymbol | ConsRef
199  ConstraintExpression ::= Expression
200  MatchingSymbol ::= Complement | Omit | AnyValue | AnyOrOmit | ValueList | ValueRange |
     SuperSet | SubSet | Permutation
277  ConsRef ::= ConstraintIdentifier [ActualCrefParList]
278  ActualCrefParList ::= "(" ActualCrefPar {Comma ActualCrefPar} ")"
279  ActualCrefPar ::= Value | DataObjectIdentifier | ConsRef
213  ValueAttributes ::= [ValueLength] [IF_PRESENT]
214  ValueLength ::= SingleValueLength | RangeValueLength
215  SingleValueLength ::= "[" ValueBound "]"
216  ValueBound ::= Number | TS_ParIdentifier | TS_ConstIdentifier | FormalParIdentifier
217  RangeValueLength ::= "[" LowerValueBound To UpperValueBound "]"
218  LowerValueBound ::= ValueBound
37   To ::= TO | ".."
219  UpperValueBound ::= ValueBound | INFINITY
224  Replacement ::= (REPLACE ReferenceList BY ConstraintValue&Attributes) | (OMIT ReferenceList)
225  ReferenceList ::= (ArrayRef | ComponentIdentifier | ComponentPosition) {ComponentReference}
```

Ce formulaire est utilisé pour les types ASN.1 comme le formulaire de déclaration de contraintes d'unités PDU en notation ASN.1 (voir le § 13.4).

13.3 *Déclaration de contraintes de primitive ASP en notation ASN.1*

Les informations suivantes sont fournies pour chaque déclaration de contrainte de primitive ASP en notation ASN.1:

a) le nom de la contrainte, éventuellement suivi d'une liste facultative de paramètres formels;

b) le nom du type de la primitive ASP;

c) le chemin de dérivation (voir les § 12.6 et 13.6), si une déclaration de contrainte ASN.1 dérive d'une contrainte ASN.1 existante, le nom de la contrainte ASN.1 prise comme base de cette dérivation sera indiqué en référence dans la table, à la rubrique chemin de dérivation;

d) la valeur de la contrainte, le corps de la table de contrainte de la primitive ASP contenant la déclaration de la contrainte ASN.1 et les attributs facultatifs. Tous les attributs et valeurs de contrainte définis au § 11.6 sont utilisables dans des contraintes ASN.1.

Les déclarations de contraintes de primitives ASP en notation ASN.1 seront données dans le format de formulaire suivant:

| **Déclaration de contrainte de primitive ASP en notation ASN.1** |
|---|
| **Nom de la contrainte :** *ConsId&ParList*<br>**Type de la primitive :** *ASP_Identifier*<br>**Chemin de dérivation :** *[DerivationPath]*<br>**Commentaires :** *[FreeText]* |
| **Valeur de la contrainte** |
| .<br>.<br>*ConstraintValue&AttributesOrReplace*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* |

**Formulaire 28 – Déclaration de contrainte de primitive ASP en notation ASN.1**

*Définition syntaxique*

```
190  ConsId&ParList ::= ConstraintIdentifier [FormalParList]
191  ConstraintIdentifier ::= Identifier
128  ASP_Identifier ::= Identifier
193  DerivationPath ::= {ConstraintIdentifier Dot}+
223  ConstraintValue&AttributesOrReplace ::= ConstraintValue&Attributes | Replacement
       {Comma Replacement}
197  ConstraintValue&Attributes ::= ConstraintValue ValueAttributes
198  ConstraintValue ::= ConstraintExpression | MatchingSymbol | ConsRef
199  ConstraintExpression ::= Expression
200  MatchingSymbol ::= Complement | Omit | AnyValue | AnyOrOmit | ValueList | ValueRange |
       SuperSet | SubSet | Permutation
277  ConsRef ::= ConstraintIdentifier [ActualCrefParList]
278  ActualCrefParList ::= "(" ActualCrefPar {Comma ActualCrefPar} ")"
279  ActualCrefPar ::= Value | DataObjectIdentifier | ConsRef
213  ValueAttributes ::= [ValueLength] [IF_PRESENT]
214  ValueLength ::= SingleValueLength | RangeValueLength
215  SingleValueLength ::= "[" ValueBound "]"
216  ValueBound ::= Number | TS_ParIdentifier | TS_ConstIdentifier | FormalParIdentifier
217  RangeValueLength ::= "[" LowerValueBound To UpperValueBound "]"
218  LowerValueBound ::= ValueBound
37   To ::= TO | ".."
219  UpperValueBound ::= ValueBound | INFINITY
224  Replacement ::= (REPLACE ReferenceList BY ConstraintValue&Attributes) | (OMIT ReferenceList)
225  ReferenceList ::= (ArrayRef | ComponentIdentifier | ComponentPosition) {ComponentReference}
```

Ce formulaire est utilisé pour les primitives ASN.1 comme le formulaire de déclaration de contraintes d'unités PDU en notation ASN.1 (voir le § 13.4).

13.4 *Déclarations de contraintes d'unités PDU en notation ASN.1*

Les tables de contraintes d'unités PDU sont très proches de celles des primitives ASP. Les déclarations de contraintes d'unités PDU en notation ASN.1 seront données dans le format de formulaire suivant:

| **Déclaration de contrainte d'unité PDU en notation ASN.1** |
|---|
| **Nom de la contrainte :** *ConsId&ParList*<br>**Type de l'unité :** *PDU_Identifier*<br>**Chemin de dérivation :** *[DerivationPath]*<br>**Commentaires :** *[FreeText]* |
| **Valeur de la contrainte** |
| .<br>.<br>*ConstraintValue&AttributesOrReplace*<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* |

**Formulaire 29 – Déclaration de contrainte d'unité PDU en notation ASN.1**

*Définition syntaxique*

```
190  ConsId&ParList ::= ConstraintIdentifier [FormalParList]
191  ConstraintIdentifier ::= Identifier
145  PDU_Identifier ::= Identifier
193  DerivationPath ::= {ConstraintIdentifier Dot}+
223  ConstraintValue&AttributesOrReplace ::= ConstraintValue&Attributes | Replacement
     {Comma Replacement}
197  ConstraintValue&Attributes ::= ConstraintValue ValueAttributes
198  ConstraintValue ::= ConstraintExpression | MatchingSymbol | ConsRef
199  ConstraintExpression ::= Expression
200  MatchingSymbol ::= Complement | Omit | AnyValue | AnyOrOmit | ValueList | ValueRange |
     SuperSet | SubSet | Permutation
277  ConsRef ::= ConstraintIdentifier [ActualCrefParList]
278  ActualCrefParList ::= "(" ActualCrefPar {Comma ActualCrefPar} ")"
279  ActualCrefPar ::= Value | DataObjectIdentifier | ConsRef
213  ValueAttributes ::= [ValueLength] [IF_PRESENT]
214  ValueLength ::=  SingleValueLength | RangeValueLength
215  SingleValueLength ::= "[" ValueBound "]"
216  ValueBound ::= Number | TS_ParIdentifier | TS_ConstIdentifier | FormalParIdentifier
217  RangeValueLength ::= "[" LowerValueBound To UpperValueBound "]"
218  LowerValueBound ::= ValueBound
37   To ::= TO | ".."
219  UpperValueBound ::= ValueBound | INFINITY
224  Replacement ::= (REPLACE ReferenceList BY ConstraintValue&Attributes) | (OMIT ReferenceList)
225  ReferenceList ::= (ArrayRef | ComponentIdentifier | ComponentPosition) {ComponentReference}
```

13.5 *Paramétrage des contraintes ASN.1*

Les contraintes ASN.1 peuvent être paramétrées (voir le § 12.5).

13.6 *Contraintes ASN.1 modifiées*

Des contraintes ASN.1 peuvent être spécifiées par modification d'une contrainte ASN.1 existante. Des parties de contrainte peuvent être spécifiées à nouveau pour former une nouvelle contrainte à l'aide des mécanismes REPLACE et OMIT (remplacer et omettre).

Certains paramètres ou champs d'une contrainte de base ou modifiée peuvent être identifiés à l'aide d'une liste de sélecteurs de champs afin de remplacer leur valeur définie par une nouvelle, ou d'omettre la valeur définie. Une liste de références ReferenceList se compose des identificateurs des sélecteurs de champs (définis dans la définition de type correspondante) séparés par des points, identifiant de manière unique un champ particulier (éventuellement structuré) dans une unité PDU (ou une primitive ASP). Les champs du premier niveau sont identifiés par un seul sélecteur, les champs imbriqués nécessitant la connaissance du chemin de dérivation complet.

Les valeurs de remplacement ne seront utilisées que lorsqu'un chemin de dérivation est spécifié. Un ensemble complet de valeurs ASN.1 ne sera utilisé que lorsque aucun chemin de dérivation n'aura été spécifié. Les valeurs remplacées ou omises peuvent être structurées.

*Définition syntaxique*

**224 Replacement ::= (**REPLACE **ReferenceList** BY **ConstraintValue&Attributes) | (**OMIT **ReferenceList)**
**225 ReferenceList ::= (ArrayRef | ComponentIdentifier | ComponentPosition) {ComponentReference}**

Si un champ appartient à une structure SEQUENCE, SET ou CHOICE, la position du champ entre les parenthèses peut être utilisée en guise d'identificateur de sélecteur de champ. Cette technique sera utilisée lorsque aucun identificateur n'aura été indiqué dans la déclaration du champ.

## 13.7 *Listes des paramètres formels dans des contraintes ASN.1 modifiées*

Les spécifications du § 12.7 s'appliquent également aux contraintes ASN.1 modifiées.

## 13.8 *Noms des paramètres de primitive ASP et des champs d'unité PDU dans les contraintes ASN.1*

Lors de la spécification d'une contrainte de primitive ASP ou d'unité PDU en notation ASN.1, il est possible d'utiliser les identificateurs de paramètres ou de champs spécifiés dans la définition ASN.1 des types SEQUENCE, SET ou CHOICE pour identifier le paramètre de primitive ASP ou le champ d'unité PDU pour lequel la contrainte impose une valeur. Dans le cas des types CHOICE, on utilisera les identificateurs de variantes. Dans le cas des types SEQUENCE, les identificateurs de paramètre ou de champ seront utilisés chaque fois que l'omission de paramètres ou champs facultatifs risque de rendre ambiguë l'attribution de valeur. Pour ce qui est des types SET, les identificateurs de paramètre ou de champ seront toujours utilisés.

*Exemple 45* – Valeurs des champs dans une contrainte ASN.1 d'unité PDU. Soit la définition de type suivante:

| **Définition de type d'unité PDU en notation ASN.1** |
|---|
| **Nom de l'unité :** XY_PDU<br>**Type de point PCO :**<br>**Commentaires :** |
| **Définition du type** |
| SET { field_1 INTEGER OPTIONAL,<br>field_2 BOOLEAN,<br>field_3 INTEGER OPTIONAL,<br>field_4 INTEGER OPTIONAL } |

La contrainte suivante est alors possible:

| **Déclaration de contrainte ASN.1 d'unité PDU** |
|---|
| **Nom de la contrainte :** CONS1 (a:INTEGER)<br>**Type de l'unité PDU :** XY_PDU<br>**Chemin de dérivation :**<br>**Commentaires :** |
| **Valeur de la contrainte** |
| { field_1 5,<br>field_2 TRUE,<br>field_3 3<br><br>}<br>-- le champ 4 n'est pas spécifié => omis au moment de l'envoi<br>-- si l'identificateur field_3 n'avait pas été utilisé, l'attribution de la valeur 3 au champ_3 ou au<br>-- champ 4 aurait été ambiguë, ces champs étant tous deux facultatifs. |

## 14 Partie dynamique

### 14.1 *Introduction*

La partie dynamique constitue le corps de la suite de tests. Elle regroupe les descriptions comportementales des tests élémentaires, des modules de test et des comportements par défaut.

### 14.2 *Comportement dynamique de test élémentaire*

#### 14.2.1 *Spécification de la table de comportement dynamique d'un test élémentaire*

14.2.1.1 La table a pour titre «comportement dynamique de test élémentaire».

14.2.1.2 L'en-tête contient les informations suivantes:

a) le nom du test élémentaire, attribuant un identificateur unique au test élémentaire décrit dans la table;

b) la référence de groupe de tests, indiquant le nom complet (au niveau le plus bas du chemin d'accès) du groupe qui contient le test élémentaire; ce nom complet respectera les spécifications du § 8.2 et se terminera par une barre oblique (/);

c) l'objectif du test, déclaration informelle de l'objectif du test élémentaire, tel qu'il figure dans la Recommandation (si elle existe) relative à la structure de la suite de tests et aux objectifs du test, ou dans une section équivalente de la Recommandation sur les suites de tests (si elle existe);

d) la référence du comportement par défaut, identificateur (comprenant, si nécessaire, une liste des paramètres effectifs) d'une description de comportement par défaut, s'il y en a, s'appliquant à la description comportementale du test élémentaire (voir le § 14.4).

14.2.1.3 Le corps de la table comportera les colonnes suivantes avec les informations correspondantes:

a) une colonne numéro de ligne (facultative) (voir le § 14.2.4), placée, lorsqu'elle existe, à l'extrême gauche de la table;

b) une colonne étiquette, dans laquelle seront inscrites les étiquettes facultatives identifiant les déclarations TTCN et utilisées par les constructions de saut GOTO (voir le § 14.14);

c) une description comportementale, décrivant le comportement des testeurs LT et UT en termes de déclarations TTCN et de leurs paramètres, à l'aide de la notation arborescente (voir le § 14.6);

d) une colonne référence de contraintes, dans laquelle sont inscrites les références aux contraintes permettant d'associer les déclarations TTCN de l'arbre comportemental à une référence aux valeurs, définies dans la partie contraintes d'une primitive ASP ou d'une unité PDU spécifique (voir le § 11);

e) une colonne verdict, dans laquelle est inscrit le verdict ou l'information de résultat associés à des déclarations TTCN de l'arbre comportemental (voir le § 14.17);

f) une colonne commentaires (facultative), utilisée pour faciliter la compréhension des déclarations TTCN en les commentant par de courtes remarques ou une référence à un texte complémentaire figurant dans le champ facultatif de bas de page commentaires détaillés.

Les colonnes c), d), e) et f) seront présentées de gauche à droite dans cet ordre. Il est recommandé de placer la colonne obligatoire étiquette immédiatement à gauche de la description comportementale. Sinon, elle peut être placée immédiatement à droite de cette description.

14.2.1.4 Un cadre de bas de page (facultatif) recueillera les commentaires détaillés.

14.2.2 *Formulaire de comportement dynamique de tests élémentaires*

Le comportement dynamique d'un test élémentaire sera décrit sous le format de formulaire suivant:

| **Comportement dynamique de test élémentaire** | | | | | | |
|---|---|---|---|---|---|---|
| **Nom du test élémentaire :** *TestCaseIdentifier*<br>**Groupe :** *TestGroupReference*<br>**Objectif :** *FreeText*<br>**Comportement par défaut :** *[DefaultReference]*<br>**Commentaires :** *[FreeText]* | | | | | | |
| **N°** | **Etiquette** | **Description comportementale** | **Etiquette** | **Réf. des contraintes** | **Verdict** | **Commentaires** |
| 1<br>2<br>.<br>.<br>.<br>.<br>.<br>.<br>*n* | .<br>.<br>*[Label]*<br>.<br><br>.<br>.<br>.<br>. | .<br>.<br>*StatementLine*<br>.<br>.<br>*TreeHeader*<br>*StatementLine*<br>.<br>. | .<br>.<br>Autre emplacement possible de la colonne Etiquette<br>.<br>.<br>.<br>.<br>.<br>. | .<br>.<br>*[ConstraintReference]*<br>.<br>.<br>.<br>.<br>.<br>. | .<br>.<br>*[Verdict]*<br>.<br>.<br>.<br>.<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>.<br>.<br>.<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | | | | | |

**Formulaire 30 – Comportement dynamique de test élémentaire**

*Définition syntaxique*

```
233  TestCaseIdentifier ::= Identifier
235  TestGroupReference ::= [SuiteIdentifier "/"] {TestGroupIdentifier "/"}
238  DefaultReference ::= DefaultIdentifier [ActualParList]
299  ActualParList ::= "(" ActualPar {Comma ActualPar} ")"
300  ActualPar ::= Value | PCO_Identifier
274  Label ::= Identifier
286  StatementLine ::= (Event [Qualifier] [AssignmentList] [TimerOps]) | (Qualifier [AssignmentList]
     [TimerOps]) | (AssignmentList [TimerOps]) | TimerOps | Construct | ImplicitSend
264  TreeHeader ::= TreeIdentifier [FormalParList]
265  TreeIdentifier ::= Identifier
266  FormalParList ::= "(" FormalPar&Type {SemiColon FormalPar&Type} ")"
267  FormalPar&Type ::= FormalParIdentifier {Comma FormalParIdentifier} Colon FormalParType
268  FormalParIdentifier ::= Identifier
269  FormalParType ::= Type | PCO_TypeIdentifier | PDU
331  Type ::= PredefinedType | ReferenceType
277  ConsRef ::= ConstraintIdentifier [ActualCrefParList]
278  ActualCrefParList ::= "(" ActualCrefPar {Comma ActualCrefPar} ")"
279  ActualCrefPar ::= Value | DataObjectIdentifier | ConsRef
281  Verdict ::= Pass | Fail | Inconclusive | Result
282  Pass ::= PASS | P | "(" PASS ")" | "(" P ")"
283  Fail ::= FAIL | F | "(" FAIL ")" | "(" F ")"
284  Inconclusive ::= INCONC | I | "(" INCONC ")" | "(" I ")"
285  Result ::= Identifier
```

L'autre emplacement possible de la colonne étiquette est indiqué en pointillés.

Les en-têtes des colonnes de ce formulaire peuvent être abrégés en: **E**, **RefC**, **V** et **C**, ce qui permet d'élargir le plus possible la colonne description comportementale si la dimension du support physique est limitée.

14.2.3 *Structure du comportement d'un test élémentaire*

Chaque test élémentaire contient une description précise des séquences d'événements (anticipés) et des verdicts correspondants. Cette description est structurée en un arbre dont les nœuds sont les déclarations TTCN, et les feuilles les verdicts correspondants. Il est souvent plus efficace de se servir des modules de test pour sous-structurer cet arbre:

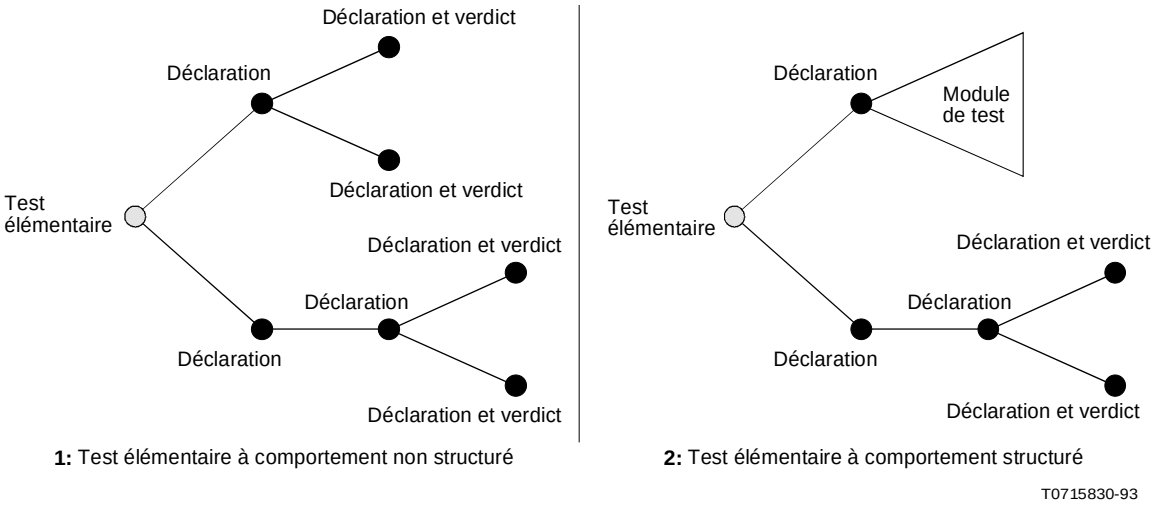



FIGURE 3/X.292

**Structure comportementale d'un test élémentaire**

En notation TTCN, cette modularisation explicite est exprimée à l'aide de modules de test et de la structure ATTACH.

14.2.4 *Numérotation et suite des lignes*

Une ligne (de description) comportementale pouvant être trop longue pour tenir sur une ligne du support physique, il est nécessaire de recourir à des symboles supplémentaires indiquant l'étalement des lignes comportementales. Deux techniques sont employées à cette fin:

a) indiquer le début d'une nouvelle ligne comportementale; une colonne supplémentaire de numérotation de lignes est ajoutée à gauche du corps de la table; cette colonne ne comporte de numérotation qu'au début de chaque nouvelle ligne comportementale; les numéros utilisés sont les entiers naturels 1, 2, 3..., sans réinitialisation pour les arbres locaux; en d'autres termes chaque ligne comportementale de la table aura un numéro de ligne unique;

*Remarque 1* – Les numéros de ligne peuvent être utilisés à des fins de consignation, pour désigner sans ambiguïté la ligne comportementale exécutée.

*Remarque 2* – Les numéros de ligne peuvent servir de références dans la section commentaires détaillés.

b) indiquer la suite de ligne; si une ligne comportementale doit se poursuivre sur la ligne physique suivante, un signe # sera inscrit au début de cette ligne de suite de texte dans la colonne comportement, il est recommandé de donner aux lignes suite la même indentation que la première ligne.

Si un report de ligne s'impose dans une colonne autre que celle de la description comportementale, le signe # n'est pas nécessaire.

*Exemple 46* – Impression de longues lignes comportementales

*46.1* Style recommandé:

| N° | Etiquette | Description comportementale | Réf. des contraintes | Verdict | Commentaires |
|---|---|---|---|---|---|
| 1 | | ceci est une déclaration TTCN trop longue pour être imprimée sur une seule ligne car la colonne est trop étroite | Ref1 | | |
| 2 | | ceci est la ligne de déclaration suivante | ceci est une référence de contrainte trop longue pour être imprimée sur une seule ligne | | |
| 3 | | ligne d'une autre déclaration | Ref2 | | |

*46.2* Autre style possible:

| Etiquette | Description comportementale | Réf. des contraintes | Verdict | Commentaires |
|---|---|---|---|---|
| | ceci est une déclaration TTCN trop longue pour être imprimée sur une seule ligne car la colonne est trop étroite | Ref1 | | |
| | ceci est la ligne de déclaration suivante | ceci est une référence de contrainte trop longue pour être imprimée sur une seule ligne | | |
| | ligne d'une autre déclaration | Ref2 | | |

## 14.3 *Comportement dynamique des modules de test*

### 14.3.1 *Spécification des tables de comportement dynamique des modules de test*

Le comportement dynamique des modules de test est défini de la même manière que celui des tests élémentaires, à ceci près que les modules de test peuvent être paramétrés (voir le § 14.7). Les tables de comportement dynamique des modules de test sont identiques à celles des tests élémentaires, exception faite des différences suivantes:

a) la table a pour titre «comportement dynamique de module de test»;

b) la première rubrique de l'en-tête est nom du module de test, qui est un identificateur unique du module de test suivi de la liste facultative des paramètres formels et des types associés. Ces paramètres peuvent être utilisés pour transmettre des points PCO, des contraintes et d'autres objets de données vers l'arbre racine du module de test;

c) la deuxième rubrique de l'en-tête est référence du groupe de modules de test, indiquant le nom complet (au niveau le plus bas du chemin d'accès) du groupe bibliothèque de modules de test contenant ce module de test; ce nom complet respectera les spécifications du § 8.3 et se terminera par une barre oblique (/);

d) la troisième rubrique de l'en-tête est objectif du module de test, déclaration informelle de l'objectif du module de test.

14.3.2 *Formulaire de comportement dynamique d'un module de test*

Le comportement dynamique d'un module de test est décrit sous le format de formulaire suivant:

| **Comportement dynamique de module de test** | | | | | | |
|---|---|---|---|---|---|---|
| **Nom du module de test** : *TestStepId&ParList*<br>**Groupe** : *TestStepGroupReference*<br>**Objectif** : *FreeText*<br>**Comportement par défaut** : *[DefaultReference]*<br>**Commentaires** : *[FreeText]* | | | | | | |
| **N°** | **Etiquette** | **Description comportementale** | **Etiquette** | **Réf. des contraintes** | **Verdict** | **Commentaires** |
| 1<br>2<br>.<br>.<br>.<br>.<br>.<br>.<br>*n* | .<br>.<br>*[Label]*<br>.<br><br>.<br>.<br>.<br>. | .<br>.<br>*StatementLine*<br>.<br>.<br>*TreeHeader*<br>*StatementLine*<br>.<br>. | .<br>.<br>Autre emplacement possible de la colonne Etiquette<br>. | .<br>.<br>*[ConstraintReference]*<br>.<br>.<br>.<br>.<br>.<br>. | .<br>.<br>*[Verdict]*<br>.<br>.<br>.<br>.<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>.<br>.<br>.<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | | | | | |

**Formulaire 31 – Comportement dynamique de module de test**

*Définition syntaxique*

245 TestStepId&ParList ::= TestStepIdentifier [FormalParList]
246 TestStepIdentifier ::= Identifier
266 FormalParList ::= "(" FormalPar&Type {SemiColon FormalPar&Type} ")"
267 FormalPar&Type ::= FormalParIdentifier {Comma FormalParIdentifier} Colon FormalParType
268 FormalParIdentifier ::= Identifier
269 FormalParType ::= Type | PCO_TypeIdentifier | **PDU**
331 Type ::= PredefinedType | ReferenceType
248 TestStepGroupReference ::= [SuiteIdentifier "/"] {TestStepGroupIdentifier "/"}
238 DefaultReference ::= DefaultIdentifier [ActualParList]
299 ActualParList ::= "(" ActualPar {Comma ActualPar} ")"
300 ActualPar ::= Value | PCO_Identifier
274 Label ::= Identifier
286 StatementLine ::= (Event [Qualifier] [AssignmentList] [TimerOps]) | (Qualifier [AssignmentList] [TimerOps]) I (AssignmentList [TimerOps]) | TimerOps | Construct | ImplicitSend
264 TreeHeader ::= TreeIdentifier [FormalParList]
265 TreeIdentifier ::= Identifier
266 FormalParList ::= "(" FormalPar&Type {SemiColon FormalPar&Type} ")"
267 FormalPar&Type ::= FormalParIdentifier {Comma FormalParIdentifier} Colon FormalParType
268 FormalParIdentifier ::= Identifier
269 FormalParType ::= Type | PCO_TypeIdentifier | **PDU**
331 Type ::= PredefinedType | ReferenceType
277 ConsRef ::= ConstraintIdentifier [ActualCrefParList]
278 ActualCrefParList ::= "(" ActualCrefPar {Comma ActualCrefPar} ")"
279 ActualCrefPar ::= Value | DataObjectIdentifier | ConsRef
281 Verdict ::= Pass | Fail | Inconclusive | Result
282 Pass ::= **PASS** | **P** | "(" **PASS** ")" | "(" **P** ")"
283 Fail ::= **FAIL** | **F** | "(" **FAIL** ")" | "(" **F** ")"
284 Inconclusive ::= **INCONC** | **I** | "(" **INCONC** ")" | "(" **I** ")"
285 Result ::= Identifier

L'autre emplacement possible de la colonne étiquette est indiqué en pointillés.

Les en-têtes des colonnes de ce formulaire peuvent s'abréger en: **L**, **Cref**, **V** et **C**.

14.4 *Comportement dynamique par défaut*

14.4.1 *Comportement par défaut*

Un test élémentaire TTCN spécifiera un comportement de remplacement pour *tous* les événements possibles (y compris les événements non valides). Dans un arbre comportemental, les séquences de propositions conditionnelles (proposition SI) aboutissent souvent toutes au même comportement. Ce comportement peut être mis en facteur comme comportement par défaut dans l'arbre. Ces descriptions de comportements par défaut sont regroupées dans la bibliothèque générale des comportements par défaut.

La dynamique des comportements par défaut est définie à l'aide des mêmes mécanismes que pour les modules de test, aux restrictions suivantes près:

a) un comportement par défaut n'a pas lui-même de comportement par défaut;

b) la description comportementale ne comportera qu'un seul arbre (c'est-à-dire qu'elle ne comportera pas d'arbres locaux);

c) l'arbre de description comportementale n'utilisera pas de rattachement d'un autre arbre (c'est-à-dire que les arbres de comportement par défaut ne peuvent se rattacher des modules de test).

Les points PCO et les autres paramètres effectifs peuvent être transmis à des descriptions de comportement par défaut de la même manière qu'ils peuvent l'être à des modules de test. On appliquera les mêmes règles de visibilité et de substitution des paramètres que pour le rattachement d'arbres (voir le § 14.13).

14.4.2 *Spécification des tables de comportement dynamique par défaut*

Les tables de comportement dynamique par défaut sont identiques à celles des modules de test, aux différences suivantes près:

a) la table a pour titre «comportement dynamique par défaut»;

b) la première rubrique de l'en-tête est nom du comportement par défaut, identificateur unique du comportement par défaut suivi de la liste facultative des paramètres formels et des types associés. Ces paramètres peuvent être utilisés pour transmettre des points PCO, des contraintes ou d'autres objets de données vers l'arbre racine du comportement par défaut;

c) la deuxième rubrique de l'en-tête est référence du groupe des comportements par défaut, indiquant le nom complet (au niveau le plus bas du chemin d'accès) du groupe des comportements par défaut contenant ce comportement par défaut; ce nom complet respectera les spécifications du § 8.4 et se terminera par une barre oblique (/);

d) la troisième rubrique de l'en-tête est objectif du comportement par défaut, déclaration informelle de l'objectif du comportement par défaut.

14.4.3 *Formulaire de comportement dynamique par défaut*

Le comportement dynamique par défaut sera décrit dans le format illustré dans le formulaire suivant:

| **Comportement dynamique par défaut** | | | | | | |
|---|---|---|---|---|---|---|
| **Nom du comportement par défaut :** *DefaultId&ParList*<br>**Groupe :** *DefaultGroupReference*<br>**Objectif :** *FreeText*<br>**Commentaires :** *[FreeText]* | | | | | | |
| **N°** | **Etiquette** | **Description comportementale** | **Etiquette** | **Réf. des contraintes** | **Verdict** | **Commentaires** |
| 1<br>2<br>.<br>.<br>.<br>.<br>.<br>.<br>*n* | .<br>.<br>*[Label]*<br>.<br><br>.<br>.<br>.<br>. | .<br>.<br>*StatementLine*<br>.<br>.<br>.<br>.<br>.<br>. | .<br>.<br>Autre emplacement possible de la colonne Etiquette<br>. | .<br>.<br>*[ConstraintReference]*<br>.<br>.<br>.<br>.<br>.<br>. | .<br>.<br>*[Verdict]*<br>.<br>.<br>.<br>.<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>.<br>.<br>.<br>.<br>. |
| **Commentaires détaillés:** *[FreeText]* | | | | | | |

**Formulaire 32 – Comportement dynamique par défaut**

*Définition syntaxique*

256 DefaultId&ParList ::= DefaultIdentifier [FormalParList]
257 DefaultIdentifier ::= Identifier
266 FormalParList ::= "(" FormalPar&Type {SemiColon FormalPar&Type} ")"
267 FormalPar&Type ::= FormalParIdentifier {Comma FormalParIdentifier} Colon FormalParType
268 FormalParIdentifier ::= Identifier
269 FormalParType ::= Type | PCO_TypeIdentifier | **PDU**
331 Type ::= PredefinedType | ReferenceType
238 DefaultReference ::= DefaultIdentifier [ActualParList]
299 ActualParList ::= "(" ActualPar {Comma ActualPar} ")"
300 ActualPar ::= Value | PCO_Identifier
274 Label ::= Identifier
286 StatementLine ::= (Event [Qualifier] [AssignmentList] [TimerOps]) | (Qualifier [AssignmentList] [TimerOps]) | (AssignmentList [TimerOps]) | TimerOps | Construct | ImplicitSend
277 ConsRef ::= ConstraintIdentifier [ActualCrefParList]
278 ActualCrefParList ::= "(" ActualCrefPar {Comma ActualCrefPar} ")"
279 ActualCrefPar ::= Value | DataObjectIdentifier | ConsRef
281 Verdict ::= Pass | Fail | Inconclusive I Result
282 Pass ::= **PASS** | **P** | "(" **PASS** ")" | "(" **P** ")"
283 Fail ::= **FAIL** | **F** | "(" **FAIL** ")" | "(" **F** ")"
284 Inconclusive ::= **INCONC** | **I** | "(" **INCONC** ")" | "(" **I** ")"
285 Result ::= Identifier

L'autre emplacement possible de la colonne étiquette est indiqué en pointillés.

Les en-têtes des colonnes de ce formulaire peuvent s'abréger en: **L**, **Cref**, **V** et **C**.

14.5 *Description comportementale*

La colonne description comportementale d'une table de comportement dynamique regroupe les combinaisons de déclarations TTCN jugées possibles par le concepteur de la suite de tests. L'ensemble de ces combinaisons est appelé arbre comportemental. Chaque déclaration TTCN constitue un nœud de l'arbre comportemental.

14.6 *Notation arborescente*

Chaque déclaration TTCN apparaîtra sur une ligne distincte. Les déclarations peuvent être reliées les unes aux autres de deux manières:

– en tant que séquences de déclarations TTCN;

– en tant que déclarations TTCN conditionnelles (propositions SI).

Les séquences de déclarations TTCN sont représentées sur des lignes de déclarations successives, chaque déclaration étant décalée d'un niveau d'indentation vers la droite par rapport à la précédente.

*Exemple 47* – Séquence de déclarations TTCN

```
EVENT_A
    CONSTRUCT_B
        EVENT_C
```

Les déclarations ayant le même niveau d'indentation et rattachées au même nœud antécédent représentent des déclarations conditionnelles (propositions SI) pouvant intervenir à un moment donné. Dès lors, cet ensemble de déclarations en notation TTCN est appelé *ensemble des propositions SI*, ou simplement *propositions SI* (conditionnelles).

*Exemple 48* – Déclarations TTCN conditionnelles

```
CONSTRUCT_A1
STATEMENT_A2
EVENT_A3
```

*Exemple 49* – Combinaison de séquences de déclarations et de propositions SI dans la construction d'un arbre

```
EVENT_A
    CONSTRUCT_B
        EVENT_C
    STATEMENT_D1
    EVENT_D2
```

Le résultat de l'évaluation d'une déclaration TTCN dépend des diverses conditions qui lui sont associées. Les conditions des différentes déclarations ne sont pas nécessairement mutuellement exclusives, c'est-à-dire qu'il est possible qu'à un moment donné, plusieurs déclarations prennent la valeur TRUE. Les déclarations des propositions SI étant évaluées dans leur ordre d'apparition, c'est la première pour laquelle les conditions seront remplies qui sera prise en considération. Ceci pourra engendrer des comportements non atteignables, lorsque des déclarations sont codées comme propositions SI à la suite d'autres déclarations toujours vraies.

Les opérations GOTO et REPEAT sont assimilées à des déclarations toujours vraies. De même, les opérations SEND (envoi), IMPLICIT SEND (envoi implicite), d'affectation et de temporisation sont également assimilées à des déclarations vraies sous réserve que le qualificateur d'accompagnement, s'il existe, ait la valeur TRUE (vrai).

En notation TTCN.GR l'indentation graphique des lignes de déclaration est mise en correspondance avec les valeurs d'indentation en notation TTCN.MP. Les déclarations du premier niveau d'indentation des propositions SI n'ayant pas d'antécédent dans l'arbre racine ou dans l'arbre local auquel elles appartiennent auront zéro pour valeur d'indentation. Les déclarations ayant un antécédent prendront comme valeur d'indentation celle de cet antécédent, plus un.

*Définition syntaxique*

```
271    Line ::= $Line  Indentation StatementLine
```

*Exemple 50* – $Line [6]+R1_POSTAMBLE

## 14.7 *Noms des arbres et listes de paramètres*

### 14.7.1 *Introduction*

Chaque description comportementale contiendra au moins un arbre comportemental. Chacun de ces arbres recevra un nom pour pouvoir être cité en référence (dans une construction ATTACH, par exemple) de manière non ambiguë.

Le premier arbre apparaissant dans une description comportementale est appelé arbre racine. Le nom d'un arbre racine est l'identificateur apparaissant dans l'en-tête de sa table de comportement dynamique. Par exemple, le nom de l'arbre racine d'un module de test est l'identificateur de ce module de test, et il en est de même pour les arbres racines des comportements dynamiques des tests élémentaires et des comportements dynamiques par défaut.

Les arbres autres que l'arbre racine qui apparaissent dans les tables de comportement dynamique sont appelés arbres locaux. Ils sont précédés d'un en-tête d'arbre contenant le nom de l'arbre.

*Définition syntaxique*

```
261    RootTree ::= {BehaviourLine}+
262    LocalTree ::= Header {BehaviourLine}+
```

### 14.7.2 *Arbres paramétrés*

Tous les arbres, à l'exception des arbres racines des tests élémentaires, peuvent être paramétrés. Les paramètres peuvent fournir des points PCO, des contraintes, des variables et d'autres éléments de la sorte à utiliser dans cet arbre. Les arbres racines des tests élémentaires ne sont pas paramétrés.

Si un arbre est paramétré, son nom sera directement suivi par une liste entre parenthèses des paramètres formels avec leurs types. Par exemple, la liste des paramètres formels de l'arbre racine d'un module de test sera indiquée entre parenthèses immédiatement après l'identificateur du module de test dans l'en-tête de la table de comportement dynamique du module de test. De même, la liste des paramètres formels d'un arbre local suivra immédiatement le nom de l'arbre dans son en-tête.

Dans la liste chaque paramètre formel sera suivi de deux points et de son type. S'il existe plusieurs paramètres formels du même type, ils pourront être regroupés dans une sous-liste. Dans ces sous-listes, les paramètres formels sont séparés les uns des autres par une virgule, le dernier étant suivi de deux points (:) et du type commun à tous les paramètres.

Lorsqu'il y a plus d'un couple paramètre formel/type (ou plus d'un couple sous-liste/type), ces couples sont séparés les uns des autres par un point-virgule (;).

Les paramètres formels peuvent appartenir aux types suivants: point PCO, primitive ASP, unité PDU, structuré, ou appartenir à l'un des autres types prédéfinis ou types de suite de tests.

Si le symbole terminal **PDU** est donné comme type de paramètre formel d'un arbre, il ne faudra pas faire référence dans l'arbre à des champs particuliers de l'unité PDU. Si le paramètre formel est l'identificateur d'une unité PDU particulière, il est alors possible de faire référence à des champs particuliers de cette unité PDU dans cet arbre.

*Exemple 51* – Module de test utilisant des paramètres formels

```
EXAMPLE_TREE (L:TSAP; X:INTEGER; Y:INTEGER)
```

*Exemple 52* – Module de test utilisant un paramètre formel avec sous-liste

```
EXAMPLE_TREE (L:TSAP; X, Y:INTEGER)
```

## 14.8 *Déclarations TTCN*

La notation arborescente permet de spécifier des événements de test déclenchés par le testeur inférieur ou supérieur [événements SEND (envoi) et IMPLICIT SEND (envoi implicite)], des événements de test reçus par le testeur inférieur ou supérieur [RECEIVE (réception), OTHERWISE (sinon) et TIMEOUT (fin de temporisation)], des constructions (ATTACH, GOTO et REPEAT) et des pseudo-événements comprenant des combinaisons de qualificateurs et d'opérations d'affectation et de temporisation. Tous ces événements et opérations sont collectivement appelés déclarations TTCN.

Les événements de test peuvent être accompagnés de qualificateurs (expressions booléennes), d'opérations d'affectation et de temporisation. Ces opérations et ces qualificateurs peuvent constituer à eux seuls des déclarations TTCN et sont alors appelés pseudo-événements.

## 14.9 *Evénements de test TTCN*

### 14.9.1 *Evénements d'envoi et de réception*

La notation TTCN prend en charge le lancement (l'envoi) de primitives ASP et d'unités PDU vers des points PCO nommés, ainsi que l'acceptation (la réception) de primitives ASP et d'unités PDU en des points PCO nommés. Le modèle de point PCO est défini aux § 10.8 et 14.9.5.2.

*Définition syntaxique*

```
289   Send ::= [PCO_Identifier | FormalParIdentifier] "!" (ASP_Identifier | PDU_Identifier)
291   Receive ::= [PCO_Identifier | FormalParIdentifier] "?" (ASP_Identifier | PDU_Identifier)
```

Dans la forme la plus simple, un identificateur de primitive ASP ou d'unité PDU suit le symbole d'envoi (!) pour les événements devant être déclenchés par les testeurs LT ou UT, ou le symbole de réception (?) pour les événements que ces testeurs peuvent accepter. Le nom facultatif d'un point PCO facultatif n'est pas indiqué. Cette forme n'est valide que lorsque la suite de tests ne comporte qu'un seul point PCO.

*Exemple 53* – !CONreq ou ?CONind

Si la suite de tests comporte plus d'un point PCO, le symbole d'envoi (!) ou de réception (?) sera précédé d'un nom de point PCO apparaissant dans la partie déclarative ou dans la liste des paramètres formels de l'arbre. Ce nom de point PCO désigne le point PCO auquel l'événement de test peut se produire.

*Exemple 54* – L!CONreq ou L?CONind

### 14.9.2 *Evénements de réception*

Le résultat d'une ligne d'événement RECEIVE est «vrai» si une primitive ASP ou une unité PDU entrante au point PCO spécifié concorde avec la ligne d'événement. Pour que cette concordance ait lieu, les conditions suivantes devront être réunies:

a) la primitive ASP ou l'unité PDU entrante est valide par rapport à la définition du type de primitive ASP ou d'unité PDU donné en référence par le nom d'événement sur cette ligne. En particulier, toutes les valeurs de paramètres et de champs seront du type défini et respecteront toutes les restrictions de longueur spécifiées;

b) la primitive ASP ou l'unité PDU concorde avec la référence de contrainte mentionnée sur cette ligne d'événement;

c) si un qualificateur est spécifié sur la ligne d'événement, son résultat doit être «vrai» (TRUE); ce qualificateur peut faire référence à des paramètres de primitive ASP et à des champs d'unité PDU.

Un événement entrant n'est retiré de la file d'entrée d'un point PCO que s'il concorde avec une ligne d'événement de réception.

14.9.3 *Evénements d'envoi*

Le résultat d'une ligne d'événement SEND comportant un qualificateur est «vrai» si l'expression de ce qualificateur prend la valeur «vrai» (TRUE). Les événements d'envoi non qualifiés sont assimilés à des événements toujours «vrais». La primitive ASP ou l'unité PDU sortante résultant d'un événement d'envoi sera formée de la manière suivante:

a) toutes les valeurs des paramètres de la primitive ASP ou des champs de l'unité PDU seront du type spécifié dans les définitions correspondantes et respecteront toutes les restrictions de longueur contenues dans ces définitions;

b) les paramètres de la primitive ASP ou les champs de l'unité PDU prendront les valeurs spécifiées dans la contrainte indiquée en référence sur la ligne d'événement (voir les § 11, 12 et 13 pour l'explication des constructions de primitives ASP et d'unités PDU avec contraintes);

c) toute valeur directement affectée aux paramètres de primitive ASP ou aux champs d'unité PDU sur la ligne d'événement prendra le pas sur la valeur correspondante éventuellement spécifiée dans la contrainte;

d) tous les paramètres ou les champs de la primitive ASP ou de l'unité PDU sortante seront valués ou explicitement omis avant la réalisation de l'événement d'envoi SEND.

La génération, par contrainte ou affectation d'une valeur de paramètre de primitive ASP ou de champ d'unité PDU enfreignant la déclaration de type ou la restriction longueur entraînera une erreur de test élémentaire.

14.9.4 *Durée de vie des événements*

Les identificateurs de paramètres de primitive ASP et de champs d'unité PDU associés aux événements SEND et RECEIVE ne pourront faire référence aux valeurs des paramètres de primitive ASP et des champs d'unité PDU que sur la ligne de déclaration elle-même.

Dans le cas des événements SEND, les paramètres des primitives ASP et les champs des unités PDU correspondants peuvent être valués, si nécessaire, dans des affectations réalisées sur la ligne d'envoi.

*Exemple 55* – !A_PDU (A_PDU.FIELD := 3)

Une telle affectation n'a pas d'effet en dehors de la ligne d'événement dans laquelle elle intervient; elle ne survit pas à l'événement d'envoi.

Dans le cas des événements RECEIVE, s'il est nécessaire de faire référence par la suite à des valeurs de paramètres de primitive ASP ou de champs d'unité PDU, on affectera tout ou partie de la primitive ASP ou de l'unité PDU à des variables sur la ligne de réception elle-même. Il est alors possible de faire référence à ces variables dans les lignes suivantes.

*Exemple 56* – ?A_PDU (VAR := A_PDU.FIELD),

VAR pouvant être utilisé sur des lignes d'événement après la réception de A_PDU.

14.9.5 *Exécution de l'arbre comportemental*

14.9.5.1 *Introduction*

Le concepteur de la suite de tests organisera l'arbre comportemental représentant un test élémentaire ou un module de test conformément aux règles suivantes relatives à l'exécution des tests:

a) démarrant depuis la racine de l'arbre, le testeur LT ou UT reste au premier niveau d'indentation jusqu'à ce qu'un événement concorde. Si un événement doit être déclenché, le testeur LT ou UT le déclenche; si un événement doit être reçu, il n'est considéré comme concordant que si un événement réel est reçu et qu'il concorde avec la ligne d'événement;

b) dès qu'un événement a concordé, le testeur LT ou UT passe au niveau suivant d'indentation. Il n'est possible de revenir à un niveau d'indentation précédent qu'en utilisant la construction de saut GOTO;

c) les lignes d'événements de même niveau d'indentation venant après une même ligne d'événement antécédente représentent les propositions SI pouvant être mises en concordance à un moment donné. Les propositions SI seront indiquées dans l'ordre dans lequel le concepteur de la suite de tests demande au testeur LT ou UT de tenter de les déclencher ou de les recevoir, si nécessaire de manière récurrente, jusqu'à ce que l'une de ces propositions SI concorde.

*Exemple 57* – Exemple d'arbre comportemental TTCN

Supposons que la séquence d'événements suivante puisse intervenir pendant un test dont l'objectif est d'établir une liaison, d'échanger des données et de libérer cette liaison. Ces événements se produisent au point PCO du testeur inférieur appelé L:

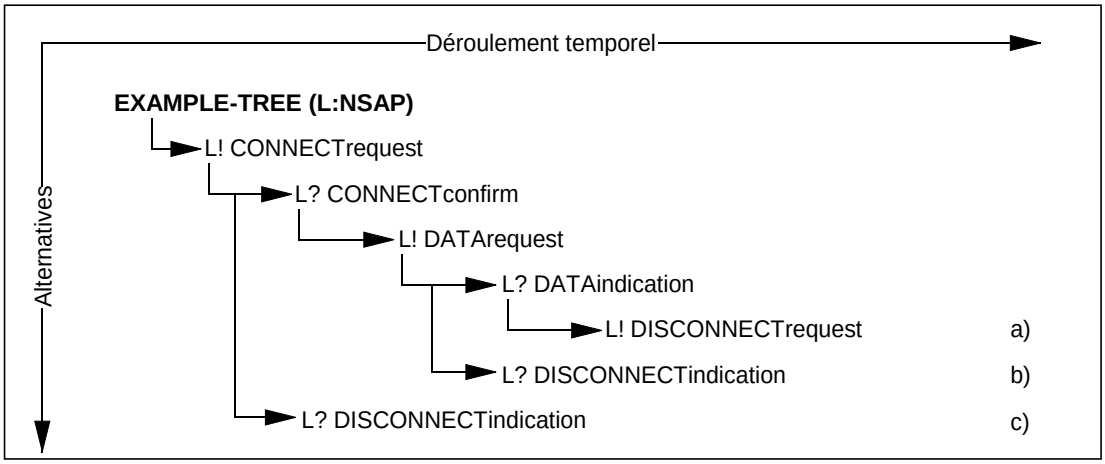
a) CONNECTrequest (demande de connexion), CONNECTconfirm (confirmation de connexion), DATArequest (demande de données), DATAindication (indication de données), DISCONNECTrequest (demande de déconnexion);

   leur déroulement peut être bloqué à tout moment par l'instance sous test ou par le fournisseur de service, ce qui génère deux autres séquences:

b) CONNECTrequest, CONNECTconfirm, DATArequest, DISCONNECTindication;

c) CONNECTrequest, DISCONNECTindication.

Ces trois séquences d'événements peuvent s'exprimer sous forme d'un arbre comportemental TTCN. Il existe cinq niveaux de propositions SI, et trois feuilles seulement (de a à c), les événements SEND L! étant toujours «vrais». L'exécution doit se dérouler de gauche à droite (déroulement séquentiel) et de haut en bas (choix conditionnels). La figure suivante illustre ce déroulement, ainsi que le principe d'arbre comportemental TTCN:




En fait, il n'existe ni traits, ni flèches, ni noms de feuilles en notation TTCN, et l'arbre comportemental de l'exemple précédent devrait être représenté comme suit:

*Exemple 58* – Arbre comportemental TTCN

| **Comportement dynamique de module de test** | | | | | |
|---|---|---|---|---|---|
| **Nom du module de test** : TREE_EX_1(L:NSAP)<br>**Groupe** : TTCN_EXAMPLES/TREE_EXAMPLE_1/<br>**Objectif** : Illustrer l'utilisation des arbres<br>**Comportement par défaut :**<br>**Commentaires** : *Remarque* – Cet exemple peut être simplifié en utilisant des comportements par défaut | | | | | |
| **N°** | **Etiquette** | **Description comportementale** | **Réf. des contraintes** | **Verdict** | **Commentaires** |
| 1 | | L! CONNECTrequest | CR1 | | Demande de connexion |
| 2 | | L? CONNECTconfirm | CC1 | | Confirmation de connexion |
| 3 | | L! DATArequest | DTR1 | | Demande d'envoi de données |
| 4 | | L? DATAindication | DTI1 | | Réception de données |
| 5 | | L! DISCONNECTrequest | DSCR1 | PASS | Acceptation |
| 6 | | L? DISCONNECTindication | DSCI1 | INCONC | Déconnexion prématurée |
| 7 | | L? DISCONNECTindication | DSCR1 | INCONC | Déconnexion prématurée |

14.9.5.2 *Sémantique d'image instantanée*

Les propositions conditionnelles du niveau d'indentation courant sont traitées dans leur ordre d'apparition. La sémantique opératoire TTCN (voir l'annexe B) suppose que l'état de tous les événements reste inchangé durant tout le processus de recherche de concordance avec une des propositions conditionnelles appartenant au même ensemble de propositions conditionnelles. Cela implique que la sémantique d'image instantanée est utilisée pour les événements reçus et les fins de temporisation; en d'autres termes, chaque fois qu'on aborde un ensemble de propositions conditionnelles, on mémorise une image instantanée de l'ensemble des événements reçus et des fins de temporisation apparues à cet instant. Seuls les événements et fins de temporisation identifiés dans cette image instantanée pourront intervenir dans le processus de concordance au cours du cycle suivant de parcours des propositions conditionnelles.

14.9.5.3 *Limites d'utilisation des événements*

Afin d'éviter toute erreur de test élémentaire, les règles suivantes s'appliquent:

a) un test élémentaire ou un module de test ne doit pas contenir de comportements pour lesquels la vitesse de traitement relative des moyens de tester risque d'influer sur les résultats. Pour éviter de tels problèmes, une ligne d'événement RECEIVE (réception), OTHERWISE (sinon), ou TIMEOUT (fin de temporisation) sera exclusivement suivie, dans un ensemble de propositions SI, par d'autres lignes d'événement du même type (événements réception, sinon ou fin de temporisation). Par conséquent, les arbres de comportement par défaut ne contiennent que des lignes événements RECEIVE, OTHERWISE et TIMEOUT au premier niveau d'indentation des propositions SI;

b) lorsqu'un événement se trouve sur une file d'attente de point PCO ou lorsqu'une fin de temporisation se trouve sur la liste des fins de temporisation, il ne peut en être retiré qu'après une recherche de concordance réussie avec la déclaration TTCN correspondante. Dans le cas d'un ensemble de propositions conditionnelles incluant des déclarations RECEIVE, l'ensemble des événements de réception attendus doit être spécifié dans sa totalité. Cela signifie qu'il devra se produire une erreur de test élémentaire si, pendant l'exécution, aucune concordance de déclaration de réception ne se produit et que cependant l'exécution passe au niveau de possibilités conditionnelles suivant du fait de l'apparition d'une fin de temporisation après la réception, sur l'une des files de l'un quelconque des points PCO pertinents, d'une primitive ASP ou d'une unité PDU qui n'était pas spécifiée dans l'ensemble des déclarations de réception.

*Exemple 59* – Ensemble incomplet d'événements RECEIVE (réception)

```
PARTIAL_TREE
!A START T
    ?B
    ?TIMEOUT T
        !C
            ?D          PASS
```

a)

```
PARTIAL_TREE
!A START T
    ?B
    ?OTHERWISE          FAIL
    ?TIMEOUT T
        !C
            ?D          PASS
            ?OTHERWISE  FAIL
```

b)

Dans le cas a), si D est reçu en réponse à !A, le test élémentaire affectera un verdict «succès» erroné dès la fin de la temporisation. Cette procédure peut être évitée par une déclaration OTHERWISE (sinon).

### 14.9.6 *Evénement IMPLICIT SEND (envoi implicite)*

Dans les méthodes de test distant, bien qu'il n'y ait pas de point PCO explicite au-dessus de l'instance sous test, il est nécessaire de pouvoir spécifier, en un point donné de la description comportementale du testeur LT, qu'il faut faire en sorte que cette instance IUT lance une primitive ASP ou une unité PDU particulière. A cette fin, on définit l'événement envoi implicite répondant à la syntaxe suivante:

*Définition syntaxique*

290 ImplicitSend ::= "<" **IUT** "!" (ASP_Identifier | PDU_Identifier) ">"

Dans cette syntaxe, l'identificateur du point PCO utilisé avec un événement normal SEND ou RECEIVE est remplacé par le symbole terminal **IUT**, pour indiquer que l'instance sous test (IUT) doit envoyer la primitive ASP ou l'unité PDU spécifiée. Les parenthèses angulaires (chevrons couchés) signifient qu'il s'agit d'un événement implicite, c'est-à-dire qu'on ne spécifie pas les actions à exécuter sur l'instance IUT pour déclencher cette réaction, mais seulement la réaction voulue elle-même.

Un événement IMPLICIT SEND est toujours considéré comme réussi, en ce sens que toute proposition SI codée à la suite de cet événement au même niveau d'indentation sera non atteignable.

On n'utilisera un IMPLICIT SEND que lorsque les Recommandations OSI correspondantes autorisent l'instance IUT à envoyer la primitive ASP ou l'unité PDU spécifiée depuis ce point lors d'une communication avec le testeur LT.

Pour chaque événement IMPLICIT SEND dans une suite de tests, le concepteur de cette suite de tests créera et référencera une question dans le formulaire PIXIT partiel, indiquant s'il est possible d'appeler à la demande cet événement IMPLICIT SEND.

On n'utilisera un événement IMPLICIT SEND que si la méthode de test utilisée est une méthode de test distant et qu'il soit possible d'obtenir le même effet avec la méthode de test monocouche répartie.

*Remarque* – Par exemple, lorsqu'on teste une instance de protocole de transport orientée connexion, si cette restriction n'existait pas, il serait possible d'utiliser un IMPLICIT SEND pour demander à l'instance IUT de lancer une unité de données de protocole de transport «demande de connexion» CR TPDU, car la méthode de test monocouche répartie permet de parvenir au même effet en demandant au testeur UT d'envoyer une primitive ASP de demande de connexion-transport T-CONreq. Par ailleurs, il ne serait pas possible d'utiliser un IMPLICIT SEND pour demander à l'instance IUT de lancer une primitive ASP de demande de rétablissement-réseau N-RstReq, cet effet ne pouvant être commandé à travers la limite du service transport. Cette restriction a pour objet d'empêcher des tests élémentaires d'exiger d'avoir un contrôle externe sur une instance IUT plus important que celui qui est prévu dans les Recommandations correspondantes sur les protocoles.

Lorsqu'un IMPLICIT SEND est spécifié, on exécute également les événements internes associés dans l'instance IUT (par exemple établir une temporisation, initialiser des variables d'état) qui sont nécessaires au respect des spécifications de la Recommandation relative au protocole testé.

La sémantique IMPLICIT SEND fait que le système à tester reçoit les commandes nécessaires au lancement des primitives ASP ou unités PDU spécifiées. Le formulaire PIXIT (ou les documents cités en référence par celui-ci) spécifiera le mode de commande du système à tester.

Ni un verdict final ni un résultat préliminaire ne sera codé sur la ligne événement d'un IMPLICIT SEND.

Un événement de réception doit se trouver en un point approprié à la suite d'une opération IMPLICIT SEND afin de vérifier la concordance de la primitive ASP ou de l'unité PDU envoyée par l'instance IUT.

*Exemple 60* – Exemple d'utilisation d'un IMPLICIT SEND

| **Comportement dynamique de test élémentaire** | | | | | |
|---|---|---|---|---|---|
| **Nom du test élémentaire** : IMP1<br>**Groupe** : TTCN_EXAMPLES/IMPLICIT_SEND1/<br>**Objectif** : Arbre partiel illustrant l'utilisation d'un envoi implicite<br>**Comportement par défaut** :<br>**Commentaires** : | | | | | |
| **N°** | **Etiquette** | **Description comportementale** | **Réf. des contraintes** | **Verdict** | **Commentaires** |
| .<br>.<br>5<br>6<br>7<br>.<br>.<br>12<br>.<br>. | | .<br>.<br><IUT ! CR><br>L?CR<br>L!CC<br>.<br>.<br>L?OTHERWISE<br>.<br>. | <br><br>CR1<br>CR1<br>CC1 | | |

14.9.7 *Evénement OTHERWISE (sinon)*

L'événement prédéfini OTHERWISE est le mécanisme TTCN permettant de traiter de manière contrôlée les événements de test imprévus. Il a la syntaxe suivante:

*Définition syntaxique*

**292 Otherwise ::= [PCO_Identifier | FormalParIdentifier] "?"** OTHERWISE

L'événement OTHERWISE est utilisé pour indiquer que le testeur inférieur ou supérieur accepte *tout* événement entrant n'ayant pas précédemment concordé avec l'une des propositions SI concurrentes de l'événement OTHERWISE.

Lorsqu'une suite de tests comprend plus d'un point PCO, l'événement OTHERWISE sera préfixé par un nom de point PCO apparaissant dans la partie déclarative ou dans la liste des paramètres formels de l'arbre. Le nom de point PCO indique le point PCO où l'événement de test peut intervenir. Les événements entrants, y compris l'événement OTHERWISE, ne sont considérés que du point de vue du point PCO donné.

*Exemple 61* – Utilisation de l'événement OTHERWISE avec des identificateurs de point PCO

| | |
|---|---|
| PARTIAL_TREE | |
| PCO1?A | |
| PCO2?B | PASS |
| PCO1?C | INCONC |
| PCO2?OTHERWISE | FAIL |

Si aucun événement n'est reçu au point PCO1, la réception de l'événement B au point PCO2 conduit à un verdict de succès. La réception d'un autre événement en ce point PCO2 conduit à un verdict d'échec.

Etant donné l'importance de l'ordre des propositions SI, les événements entrants constituant des propositions SI à la suite d'un événement OTHERWISE inconditionnel au même point PCO ne concorderont jamais.

*Exemple 62* – Evénements entrants à la suite d'un événement OTHERWISE

| | |
|---|---|
| PARTIAL_TREE | |
| PCO1?A | PASS |
| PCO1?OTHERWISE | FAIL |
| PCO1?C | INCONC |

L'événement OTHERWISE concorde avec tout événement entrant différent de A et de B. La dernière proposition SI (?C), ne pourra jamais concorder.

14.9.8 *Evénement TIMEOUT (fin de temporisation)*

L'événement TIMEOUT permet de vérifier dans un test élémentaire l'expiration d'une ou de toutes les temporisations. Lorsqu'un temporisateur s'arrête (conceptuellement immédiatement avant le traitement par image instantanée d'un ensemble de propositions SI concurrentes), un événement TIMEOUT est placé dans une liste de fins de temporisations. Le temporisateur se désactive immédiatement. Pour un temporisateur donné, une seule entrée peut apparaître dans cette liste à un moment donné. Les fins de temporisations n'étant pas associées à des points PCO, les fins de temporisations sont regroupées en une seule liste.

Si un nom de temporisateur est indiqué dans le traitement d'un événement TIMEOUT, un tel événement correspondant à ce nom est recherché dans la liste des fins de temporisations et, s'il est trouvé, il est retiré de la liste et il est considéré comme ayant réussi.

Si aucun nom de temporisateur n'est indiqué, tout événement TIMEOUT de la liste concorde. L'événement TIMEOUT est réussi si la liste n'est pas vide. Dès qu'il y a une concordance, toute la liste de fins de temporisations est vidée.

L'événement TIMEOUT obéit à la syntaxe suivante:

*Définition syntaxique*

293 Timeout ::= "?" **TIMEOUT** [TimerIdentifier]

*Exemple 63* – Utilisation de TIMEOUT

| | | | | |
|---|---|---|---|---|
| | ?TIMEOUT T | | | |

Les événements de TIMEOUT n'étant pas des événements de RECEIVE, les propositions OTHERWISE antérieurement codées ne les rendent pas non atteignables.

14.10 *Expressions TTCN*

14.10.1 *Introduction*

La notation TTCN comporte deux sortes d'expressions: les affectations et les expressions booléennes, qui peuvent toutes deux contenir des valeurs explicites et les formes suivantes de référence à des objets de données:

a) paramètres de suite de tests;

b) constantes de suite de tests;

c) variables de suite de tests et de test élémentaire;

d) paramètres formels d'un arbre de module de test, de comportement par défaut ou d'un arbre local;

e) primitives ASP et unités PDU (sur les lignes d'événement).

Toute variable apparaissant dans une expression booléenne ou dans le membre droit d'une affectation sera valuée. Si une variable non valuée est utilisée, il s'agit d'une erreur de test élémentaire.

*Définition syntaxique*

```
303  Expression ::= SimpleExpression [RelOp SimpleExpression]
304  SimpleExpression ::= Term {AddOp Term}
305  Term ::= Factor {MultiplyOp Factor}
306  Factor ::= [UnaryOp] Primary
307  Primary ::= Value I DataObjectReference I OpCall I SelectExprIdentifier I "(" Expression ")"
336  Value ::= LiteralValue I ASN1_Value
337  LiteralValue ::= Number I BooleanValue I Bstring I Hstring I Ostring I Cstring
338  Number ::= (NonZeroNum {Num} ) I 0
339  NonZeroNum ::= 1 I 2 I 3 I 4 I 5 I 6 I 7 I 8 I 9
340  Num ::=  0 I NonZeroNum
341  BooleanValue ::= TRUE I FALSE
342  Bstring ::= " ' " {Bin I Wildcard} " ' " B
343  Bin ::= 0 I 1
344  Hstring ::=  " ' " {Hex I Wildcard} " ' " H
345  Hex ::= Num I A I B I C I D I E I F
346  Ostring ::= " ' " {Oct I Wildcard} " ' " O
347  Oct ::= Hex Hex
348  Cstring ::= " " " {Char I Wildcard I "\"} " " "
349  Char ::= /* REFERENCE – A character defined by the relevant character string type */
350  Wildcard ::= AnyOne I AnyOrNone
351  AnyOne ::=  "?"
352  AnyOrNone ::= "*"
308  DataObjectReference ::= DataObjectIdentifier {ComponentReference}
309  DataObjectIdentifier ::= TS_ParIdentifier I TS_ConstIdentifier I TS_VarIdentifier I TC_VarIdentifier I
       FormalParIdentifier I ASP_Identifier I PDU_Identifier
310  ComponentReference ::= RecordRef I ArrayRef I BitRef
311  RecordRef ::= Dot (ComponentIdentifier I PDU_Identifier I StructIdentifier I ComponentPosition)
312  ComponentIdentifier ::= ASP_ParIdentifier I PDU_FieldIdentifier I ElemIdentifier I ASN1_Identifier
314  ComponentPosition ::= "("Number")"
315  ArrayRef ::= Dot "[" ComponentNumber "]"
316  ComponentNumber ::= Expression
317  BitRef ::= Dot (BitIdentifier I "[" BitNumber "]")
318  BitIdentifier ::= Identifier
319  BitNumber ::= Expression
320  OpCall ::= TS_OpIdentifier ( ActualParList I "(" ")" )
321  AddOp ::= "+" | "–" | OR
322  MultiplyOp ::= "*" I "/" | MOD I AND
323  UnaryOp ::= "+" I "–" I NOT
324  RelOp ::= "=" | "<" | ">" | "< >" | ">=" | "<="
```

14.10.2 *Références à des objets de données définis en notation ASN.1*

Pour pouvoir se référer à des composantes d'objets de données structurés, les mécanismes d'accès suivants ont été prévus:

a) une référence à une composante d'un des types suivants: SEQUENCE, SET et CHOICE est formée à l'aide d'une notation à points, c'est-à-dire en ajoutant un point et le nom (l'identificateur) de la composante voulue à l'identificateur de l'objet de données; l'identificateur de la composante sera utilisé si cela est spécifié;

b) une référence à une composante sans nom est formée en donnant entre parenthèses la position de cette composante dans la définition de type; la numérotation des composantes de ces types commence à zéro.

*Définition syntaxique*

```
310  ComponentReference ::= RecordRef I ArrayRef | BitRef
311  RecordRef ::= Dot (ComponentIdentifier | PDU_Identifier | StructIdentifier | ComponentPosition)
312  ComponentIdentifier ::= ASP_ParIdentifier | PDU_FieldIdentifier | ElemIdentifier | ASN1_Identifier
314  ComponentPosition ::= "("Number")"
315  ArrayRef ::= Dot "[" ComponentNumber "]"
316  ComponentNumber ::= Expression
```

Un index indiqué entre crochets sert à pointer une composante d'un type ASN.1 SEQUENCE OF ou SET OF.

La première composante porte le numéro zéro.

L'expression prendra une valeur entière non négative.

*Exemple 64* – Références à des composantes

```
Example_type ::= SEQUENCE  {
                                field_1      INTEGER,
                                field_2      BOOLEAN,
                                             OCTET STRING }
```

Si var1 est du type ASN.1 Example_type, il est possible d'écrire:

var1.field_1, pour désigner le premier champ INTEGER

var1.(3), pour désigner le troisième champ (sans nom)

*Exemple 65* – Références à des champs d'unité PDU

```
XY_PDUtype ::= SEQUENCE  {
                                    :
                                user_data        OCTET STRING,
                                    : }
```

Sur une ligne de déclaration contenant le type XY_PDUtype, il est possible d'écrire: L? XY_PDU(buffer := XY_PDUtype.user_data).

La notation par index est utilisée pour pointer des éléments (bits) du type ASN.1 BITSTRING. Ce type est sensé être défini comme une SEQUENCE OF {BOOLEAN}. Si certains bits d'une chaîne BITSTRING sont associés à un identificateur (bits nommés), il est possible de les désigner à l'aide de cet identificateur ou de la notation à points.

*Définition syntaxique*

```
317    BitRef ::= Dot (BitIdentifier | "[" BitNumber "]")
318    BitIdentifier ::= Identifier
319    BitNumber ::= Expression
```

Le bit le plus à gauche porte le numéro zéro.

L'expression prendra une valeur entière non négative.

*Exemple 66* – Références à des composantes pour des types BITSTRING

```
B_type ::= BIT STRING { ack(0), poll(3) }
```

où le bit zéro est appelé «ack» (accusé de réception), le bit trois «poll» (interrogation).

Si b_str est du type ASN.1 B_type, on peut écrire:

b_str.ack := TRUE

b_str.[2] := FALSE

Il est à noter que b_str.poll := TRUE et b_str.[3] := TRUE sont des expressions qui toutes deux affectent la valeur TRUE au bit «poll».

La Recommandation X.208 définit des types SET et SET OF comme des ensembles non ordonnés de composantes. Cette condition n'est valable que si les valeurs de ces types sont codées et envoyées par l'intermédiaire du fournisseur de service sous-jacent. La notation TTCN traite donc les objets de données des types SET et SET OF de la même manière que les objets des types SEQUENCE et SEQUENCE OF, c'est-à-dire en se référant aux composantes par des numéros, le numéro *i* renvoyant toujours au *ième* champ déclaré pour ce type. Les identificateurs ASN.1 des composantes comprennent le nom ASN.1 des paramètres des primitives ASP, des champs d'unités PDU ou de leurs sous-composantes.

Une fois reçue la primitive ASP ou l'unité PDU, l'appel de la composante d'index *i* donnera toujours la même valeur. Aucune opération TTCN ne modifie l'ordre des éléments d'un type SET ou SET OF.

14.10.3 *Références à des objets de données définis à l'aide de tables*

La syntaxe définie au § 14.10.2 servira aussi à former des pointeurs désignant les paramètres des primitives ASP, les champs des unités PDU et les éléments des types structurés définis sous forme tabulaire. Pour désigner un paramètre, un champ ou un élément, l'identificateur de l'objet de données sera suivi d'un point (.) et de l'identificateur du paramètre, du champ ou de l'élément.

Lorsqu'un paramètre, un champ ou un élément est défini comme une sous-structure de type structuré à définition tabulaire, la référence aux éléments de cette sous-structure sera formée en faisant suivre l'identificateur du paramètre, du champ ou de l'élément par un point puis par l'identificateur de l'élément de cette sous-structure.

Si une structure est utilisée dans un développement de macro, il sera fait référence à ses éléments comme si cette structure avait été développée dans la structure qui s'y réfère.

Si un paramètre, un champ ou un élément est défini comme étant du métatype PDU, aucune référence ne sera faite aux champs de cette sous-structure .

14.10.4 *Affectations*

14.10.4.1 *Introduction*

Des événements de test peuvent être associés à une liste d'affectations et à un qualificateur. Les affectations sont séparées par des virgules et la liste est mise entre parenthèses.

*Définition syntaxique*

```
301   AssignmentList ::= "(" Assignment {Comma Assignment} ")"
302   Assignment ::= DataObjectReference ":=" Expression
```

Dans une affectation, le membre droit prendra une valeur du même type que le membre gauche.

Une affectation a pour effet de donner à la variable d'un test élémentaire ou d'une suite de tests (ou au paramètre de primitive ASP ou au champ d'unité PDU) la valeur de l'expression. Cette expression ne contiendra aucune variable non valuée.

Toutes les affectations sont effectuées dans l'ordre de leur apparition, c'est-à-dire de gauche à droite.

*Exemple 67* – Utilisation des affectations sur des lignes d'événement

```
(X:=1)
    (Y:=2)
        L!A (Y:=0, X:=Y, A.field1:=Y)
            L?B (Y:=B.field2, X:=X+1)
```

Lorsque la transmission de l'unité PDU A est réussie, les variables de test élémentaire X et Y ainsi que le champ field1 de l'unité A ont un contenu nul. A la réception de l'unité PDU B, la variable Y se voit affecter le contenu du champ field2 de l'unité B et la variable de test élémentaire X est incrémentée.

14.10.4.2 *Règles d'affectation pour les types chaînes*

Les règles suivantes s'appliquent aux affectations faisant intervenir des types de chaînes avec restriction de longueur:

a) si la chaîne d'origine est plus longue que la longueur définie pour le type de chaîne de destination, elle sera tronquée à droite à la longueur maximale du type de la chaîne de destination;

b) si la chaîne d'origine est plus courte que la longueur définie pour le type de la chaîne de destination, elle sera alignée à gauche et complétée par des caractères de remplissage jusqu'à correspondre à la taille maximale du type de la chaîne de destination.

Les caractères de remplissage sont les suivants:

" " (blanc) pour toutes les chaînes de caractères CharacterString;

"0" (zéro) pour les chaînes BITSTRING, HEXSTRING et OCTETSTRING.

Lorsqu'une variable de type chaîne illimitée (c'est-à-dire d'une longueur infinie) est utilisée à gauche d'une affectation, elle sera limitée à la valeur du côté droit sans remplissage. Le remplissage ne doit être utilisé que lorsque la variable est du type chaîne de longueur fixe.

14.10.5 *Qualificateurs*

Un événement pourra être qualifié par une expression booléenne entre crochets placée à sa suite. L'interprétation de cette qualification est que la déclaration n'est exécutée que si l'événement concorde et si le qualificateur prend la valeur «vrai» (TRUE).

Si un qualificateur et une affectation sont associés au même événement, le qualificateur apparaîtra en premier, chacun de ses termes étant évalué avec les valeurs en vigueur avant l'exécution de l'affectation.

*Définition syntaxique*

```
288    Qualifier ::= "[" Expression "]"
```

14.10.6 *Lignes d'événement comportant des affectations et des qualificateurs*

Un événement peut être associé à une affectation, à un qualificateur ou aux deux. Dans le premier cas, l'affectation n'est exécutée que si l'événement concorde. Dans le deuxième cas, l'événement ne peut concorder que si le qualificateur prend la valeur «vrai» (TRUE). Dans le dernier cas, l'événement ne concorde que si le qualificateur prend la valeur «vrai», et l'affectation n'est exécutée que si l'événement concorde.

Si un événement «réception» est spécifié avec un qualificateur, et qu'un événement qui s'est réalisé concorde potentiellement avec l'événement spécifié, le qualificateur sera évalué dans le contexte de l'événement intervenu. Si le qualificateur fait référence à des paramètres de primitive ASP ou à des champs d'unité PDU, les valeurs de ces paramètres ou de ces champs seront extraites de l'événement intervenu.

Les règles d'utilisation des affectations dans les événements sont les suivantes:

a) dans un événement SEND, toutes les affectations sont effectuées *après* le calcul du qualificateur et *avant* la transmission de la primitive ASP ou de l'unité PDU;

b) dans les événements SEND, les affectations sont permises pour les champs de la primitive ou de l'unité PDU en cours transmise;

c) dans un événement RECEIVE, les affectations sont effectuées *après* que la réalisation de l'événement s'est produite et ne peuvent concerner les champs de la primitive ASP ou de l'unité PDU qui vient d'être reçue.

Une affectation dans la partie comportementale portant sur une contrainte de paramètre de primitive ASP, de champ d'unité PDU ou d'élément de structure provoque le remplacement des valeurs de contrainte au niveau de la ligne d'événement SEND.

*Exemple 68* – Utilisation d'un événement SEND qualifié

```
PARTIAL_TREE
!A[X=3]
!B
```

Dans le traitant de ces événements SEND concurrents, le testeur n'envoie A que si la variable X vaut 3. Sinon il envoie B.

*Exemple 69* – Utilisation de l'événement OTHERWISE, des qualificateurs et des affectations

Il est possible d'utiliser l'événement OTHERWISE avec des qualificateurs et des affectations. Si on utilise un qualificateur, son expression booléenne devient une condition supplémentaire à l'acceptation d'un événement entrant. Si on utilise une déclaration d'affectation, cette affectation n'intervient que si toutes les conditions de concordance de l'événement OTHERWISE sont respectées, par exemple:

```
PARTIAL_TREE (PCO1:XSAP; PCO2:YSAP)
PCO1?A                                                          PASS
PCO2?B[X=2]                                                     INCONC
PCO1?C                                                          PASS
PCO2?OTHERWISE[X<>2](Motif:="Xdifférent de 2")                  FAIL
PCO2?OTHERWISE(Motif:="X égal à 2 mais l'événement n'est pas B") FAIL
```

Si on suppose qu'aucun événement n'est reçu au point PCO1, la réception de l'événement B au point PCO2 lorsque X = 2 donne un verdict non concluant. La réception d'un autre événement au point PCO2 lorsque X <> 2 donne un verdict d'échec et affecte la valeur «X différent de 2» à la variable motif de type chaîne de caractères. Si un événement reçu au point PCO2 ne répond à aucun de ces scénarios, c'est le dernier événement OTHERWISE qui concorde.

### 14.11 *Pseudo-événements*

Les affectations, les qualificateurs et les temporisateurs peuvent se retrouver seuls sur une ligne de déclaration d'un arbre comportemental, sans événement associé. De telles expressions autonomes sont appelées pseudo-événements.

Un pseudo-événement a la signification suivante:

a) qualificateur seul: celui-ci est évalué s'il prend la valeur «vrai» (TRUE) et l'exécution se poursuit avec la déclaration comportementale suivante; s'il prend la valeur «faux» (FALSE), on passe à la proposition SI suivante. S'il n'existe aucune autre proposition SI, il s'agit d'une erreur de test élémentaire;

b) affectations et temporisateurs: les affectations sont exécutées de gauche à droite et les opérations de temporisation sont exécutées de gauche à droite;

c) affectations et temporisateurs précédés d'un qualificateur: le qualificateur est d'abord évalué; s'il prend la valeur «vrai» (TRUE), les affectations et les opérations de temporisation sont exécutées.

### 14.12 *Gestion des temporisateurs*

#### 14.12.1 *Introduction*

La gestion des temporisateurs est modélisée par un ensemble d'opérations, qui peuvent être combinées à des événements ou apparaître sous forme de pseudo-événements autonomes.

Les opérations de temporisation peuvent s'appliquer:

– à un temporisateur individuel, spécifié en faisant suivre l'opération de temporisation par le nom du temporisateur;

– à tous les temporisateurs, ce qui est spécifié en omettant le nom du temporisateur.

On suppose que les temporisateurs utilisés dans une suite de tests sont inactifs ou déclenchés. Tous les temporisateurs déclenchés sont automatiquement annulés à la fin de chaque test élémentaire. Il existe trois opérations de temporisation prédéfinies: START (déclenchement), CANCEL (annulation) et READ-TIMER (lecture de temporisation). Il est possible de spécifier plus d'une opération de temporisateur sur une même ligne d'événement si cela est nécessaire. Dans ce cas, les opérations sont séparées par des virgules.

Lorsqu'une opération de temporisation apparaît sur la même ligne de déclaration qu'un événement ou un qualificateur, elle est exécutée si et seulement si l'événement concorde et/ou si le qualificateur prend la valeur «vrai» (TRUE).

*Définition syntaxique*

```
325  TimerOps ::= TimerOp {Comma TimerOp}
326  TimerOp ::= StartTimer | CancelTimer | ReadTimer
```

### 14.12.2 *Opération de déclenchement START*

L'opération START sert à déclencher un temporisateur.

*Définition syntaxique*

```
327  StartTimer ::= START TimerIdentifier [ "(" TimerValue ")" ]
117  TimerIdentifier ::= Identifier
329  TimerValue ::= Expression
```

Le paramètre facultatif valeur de temporisation doit être donné si aucune durée par défaut n'a été spécifiée lors de la déclaration du temporisateur ou si on souhaite affecter au temporisateur une heure d'expiration (c'est-à-dire une durée) différente de la valeur par défaut si celle-ci a été spécifiée.

Les valeurs de temporisation seront du type entier. Le rédacteur du test élémentaire s'assurera que le paramètre facultatif valeur de temporisation prend une valeur entière strictement positive. Si un temporisateur est déclenché avec une valeur négative ou nulle, il en résultera une erreur de test élémentaire.

Toute variable figurant dans l'expression spécifiant la valeur facultative de la temporisation sera valuée. Si une variable non valuée est utilisée, il s'agit d'une erreur de test élémentaire.

Si une nouvelle durée est imposée à un temporisateur, cette nouvelle valeur ne s'appliquera qu'à l'instance courante du temporisateur: toutes les opérations START ultérieures de déclenchement de ce temporisateur ne spécifiant pas de durée adopteront la durée par défaut déclarée dans la section des déclarations de temporisation.

*Exemple 70* – Utilisation de l'opération START

Les $T_i$ sont des identificateurs de temporisateurs et les $V_i$ leurs valeurs de temporisation:

START T0

START T0 (V0)

START T1, START T2 (V2)

L'opération START peut s'appliquer à un temporisateur déjà déclenché: dans ce cas, le temporisateur est annulé, réinitialisé et redéclenché. Toutes les entrées qui s'y rapportent dans la liste de fins de temporisation en seront retirées.

### 14.12.3 *Opération CANCEL (annulation)*

L'opération CANCEL sert à arrêter un temporisateur déclenché.

*Définition syntaxique*

```
328  CancelTimer ::= CANCEL [TimerIdentifier]
117  TimerIdentifier ::= Identifier
329  TimerValue ::= Expression
```

Un temporisateur annulé devient inactif. Si un événement TIMEOUT figure dans la liste de fins de temporisation, il en sera retiré. Si le nom du temporisateur est omis dans l'opération CANCEL, tous les temporisateurs en fonctionnement sont annulés et la liste de fins de temporisation est vidée de son contenu.

L'annulation d'un temporisateur inactif est une opération valide, mais n'a bien sûr aucun effet.

*Exemple 71* – Exemples d'utilisation de l'opération CANCEL

CANCEL

CANCEL T0

CANCEL T1, CANCEL T2

CANCEL T1, START T3

Les $T_i$ sont des identificateurs de temporisateurs.

14.12.4 *Opération READTIMER (lecture de temporisateur)*

L'opération READTIMER sert à obtenir le temps écoulé depuis le déclenchement de la temporisation et à l'enregistrer dans la variable spécifiée de suite de tests ou de test élémentaire. Cette variable est du type entier. La valeur temporelle qui lui est affectée est exprimée dans l'unité de temps spécifiée pour ce temporisateur lors de sa déclaration. Par convention, l'application de l'opération READTIMER à un temporisateur inactif renvoie la valeur zéro.

*Définition syntaxique*

330 ReadTimer ::= **READTIMER** TimerIdentifier "(" DataObjectReference ")"
117 TimerIdentifier ::= Identifier

*Exemple 72* – Utilisation de l'opération de lecture READTIMER

```
:
START TimerName (TimerVal)
        ?EVENT_A
            +Tree_A
        ?EVENT_B
            +Tree_B
        ?EVENT_C
            READTIMER TimerName (CurrTime)
                +Tree_C
        ?TIMEOUT TimerName
:
```

Si EVENT_C est reçu avant l'expiration de la temporisation TimerName, le temps écoulé depuis son déclenchement est enregistré dans la variable de test élémentaire ou de suite de tests CurrTime. Le comportement de Tree_C pourra utiliser cette valeur.

*Exemple 73* – Utilisation combinée de l'opération READTIMER et d'autres opérations de temporisation

READTIMER T1 (PASSED_TIME), CANCEL T1

READTIMER T1 (V1), START NEW_TIMER (V1)

14.13 *Construction ATTACH (rattachement)*

14.13.1 *Introduction*

Des arbres peuvent être rattachés à d'autres à l'aide de la construction ATTACH définie par la syntaxe suivante:

*Définition syntaxique*

296 Attach ::= "+" TreeReference [ActualParList]
298 TreeReference ::= TestStepIdentifier | TreeIdentifier
299 ActualParList ::= "(" ActualPar {Comma ActualPar} ")"
300 ActualPar ::= Value | PCO_Identifier

Les variables de suite de tests et de test élémentaire sont communes à l'arbre de rattachement (l'arbre principal) et à l'arbre rattaché, c'est-à-dire que toute modification apportée à une telle variable dans un arbre rattaché sera également effective dans l'arbre principal. Chaque construction de rattachement d'arbre occupera seule une ligne de déclaration complète.

14.13.2 *Visibilité des arbres après rattachement*

Les descriptions comportementales peuvent contenir plus d'un arbre. Toutefois, seul l'arbre *principal* de la description comportementale est accessible depuis l'extérieur de celle-ci. Tous les arbres imbriqués sont considérés comme des modules de test locaux de cette description comportementale et donc inaccessibles depuis l'extérieur.

A noter que seuls les tests élémentaires sont directement exécutables, les modules de test n'étant exécutés que s'ils sont rattachés à un test élémentaire, soit directement soit par l'intermédiaire d'un ou plusieurs autres modules de tests. Les tests élémentaires ne peuvent pas être rattachés.

Une référence à un arbre peut être un identificateur de module de test ou un identificateur d'arbre:

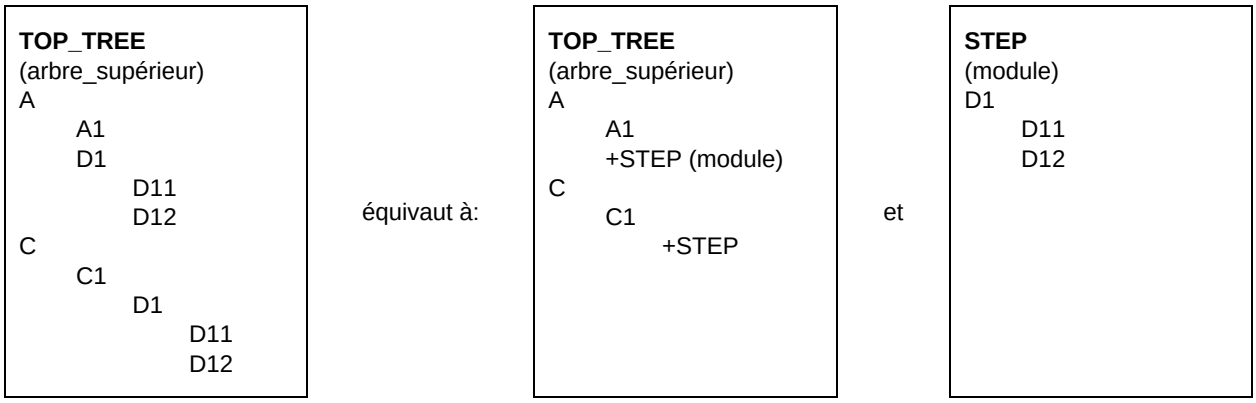
a) l'identificateur de module de test indiquant le rattachement d'un module de test situé dans la bibliothèque des modules de test; ce module de test est désigné par son identificateur unique;

b) l'identificateur d'arbre étant le nom de l'un des arbres figurant dans la description comportementale courante; il s'agit donc du rattachement d'un arbre local.

14.13.3 *Règles de base du rattachement des arbres*

Il est possible de détacher certaines parties d'un arbre comportemental pour former des arbres comportementaux distincts, c'est-à-dire des modules de test. Les points d'où un module de test a été séparé de l'arbre d'origine sont indiqués par le symbole de rattachement (+) suivi du nom attribué à ce module de test.

*Exemple 74* – Fractionnement d'un grand arbre en deux plus petits




Cette opération peut s'effectuer non seulement sur l'arbre comportemental principal (l'arbre racine) du test élémentaire, mais également sur les modules de test qui en ont été détachés. L'arbre rattaché est un arbre local ou un élément de la bibliothèque des modules de test.

Le rattachement d'arbre peut être défini d'une manière plus générale qu'une simple insertion de modules de test:

– un arbre rattaché ne contient pas nécessairement des trajets complets allant jusqu'à des déclarations qui constitueront les feuilles de l'arbre auquel il est rattaché (son *arbre d'appel*): certaines «branches», communes à tous les trajets de l'arbre rattaché, pourront être spécifiées dans l'arbre d'appel, c'est-à-dire comme des comportements ultérieurs au point de rattachement;

– certaines déclarations (même du niveau supérieur) du module de test rattaché peuvent prendre elles-mêmes la forme +SOME_SUBTREE (+SOUS_ARBRE_DONNE), permettant ainsi le rattachement d'autres modules de test;

– il est possible de paramétrer des modules de test rattachés.

14.13.4 *Signification du rattachement d'arbre*

14.13.4.1 La liste suivante définit la sémantique opératoire du rattachement d'arbre:

a) la ligne de rattachement (par exemple, +STEP) de l'arbre de comportement (par exemple, TOP_TREE) constitue formellement l'une des propositions conditionnelles (par exemple, $A_i$) de l'ensemble ordonné:

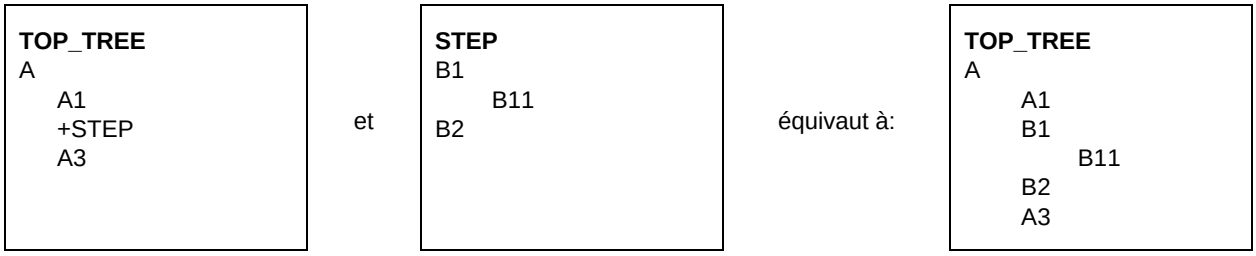
$$(A_1, \ldots, A_i, \ldots, A_n)$$

Le rattachement de STEP en cette position signifie qu'on développe TOP_TREE en insérant les propositions conditionnelles du niveau d'indentation supérieur (par exemple, $B_1, \ldots, B_m$) du module de test STEP dans cette séquence, pour former ainsi une nouvelle séquence de propositions conditionnelles:

$$(A_1, \ldots, A_{(i-1)}, B_1, \ldots, B_m, A_{(i+1)}, \ldots, A_n)$$

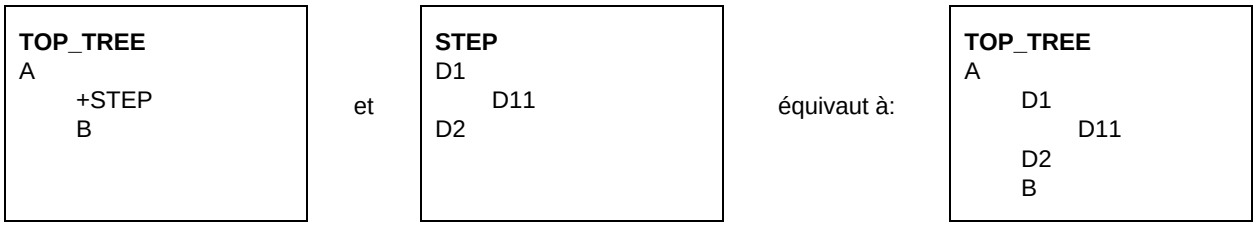
Tous les comportements de niveau d'indentation inférieur dépendant des comportements B seront rattachés en même temps qu'eux;

*Exemple 75* – Développement d'un module de test




b) tous les comportements consécutifs à la ligne +STEP dans cet arbre, deviennent des comportements consécutifs à toutes les feuilles du STEP rattaché après développement;

*Exemple 76* – Comportement consécutif à une construction de rattachement




c) lorsqu'on utilise une liste de paramètres effectifs dans une construction ATTACH, chacun des paramètres formels se verra substituer le paramètre effectif correspondant par simple substitution textuelle. Cette substitution s'effectue conformément aux règles de visibilité suivantes:

1) les paramètres effectifs de la construction ATTACH à un arbre local ne sont substitués aux paramètres formels correspondants qu'à l'intérieur de cet arbre local;

2) les paramètres effectifs de la construction ATTACH à l'arbre racine d'un module de test sont substitués à toutes les occurrences des paramètres formels correspondants dans l'arbre racine et dans tous les arbres locaux à l'intérieur du module de test;

3) lorsqu'un arbre paramétré est rattaché:

A) le nombre des paramètres effectifs est le même que celui des paramètres formels,

B) chaque paramètre actuel prend une valeur du type du paramètre formel correspondant.

*Exemple 77* – Substitution de paramètres

```
TOP_TREE(L:NSAP; U:TSAP)
L!CONreq (M:=1)
    +STEP (U, M, 2)
```

et

```
STEP(PCO:TSAP; X,Y:INTEGER)
PCO?CONind (X:=Y)
```

équivaut à:

```
TOP_TREE(L:NSAP; U:TSAP)
L!CONreq (M:=1)
    U?CONind (M:=2)
```

*Exemple 78* – Règles de visibilité applicables à la substitution de paramètres

| **Comportement dynamique de module de test** | | | | | |
|---|---|---|---|---|---|
| **Nom de module de test** : TEST_STEP_1(X, Y:INTEGER)<br>**Groupe** : TTCN_EXAMPLES/PARAMS/MODULES/<br>**Objectif** : Illustration des règles de visibilité en substitution de paramètres<br>**Comportement par défaut** :<br>**Commentaires** : | | | | | |
| **N°** | **Etiquette** | **Description comportementale** | **Réf. des contraintes** | **Verdict** | **Commentaires** |
| 1<br>2<br>3<br><br><br><br>4<br>5 | | ?A<br>+TEST_STEP_2(X)<br>+LOCAL(5)<br><br><br>LOCAL(F:INTEGER)<br>!B<br>(TC_VAR:=F+Y) | A1<br><br><br><br><br>B1 | <br><br><br><br><br>PASS | |
| **Commentaires détaillés:** Lorsque TEST_STEP1 est rattaché par un arbre d'appel, toutes les occurrences des paramètres formels X et Y figurant dans la totalité du module de test (y compris dans l'arbre LOCAL) sont remplacées par les paramètres effectifs d'appel. A noter que les paramètres formels X et Y ne sont pas remplacés automatiquement par les paramètres effectifs dans TEST_STEP2. Cependant, la valeur du paramètre effectif se substitue à celle du paramètre formel X dans la construction ATTACH "+TEST_STEP2(X)", ce qui aboutit à ce que la valeur du paramètre effectif X du paramètre (dans TEST_STEP1) se substitue au paramètre formel du TEST_STEP2 quel que soit le nom qui lui est donné dans la déclaration de ce dernier module. A noter enfin que le paramètre effectif 5 (constante) se substitue au paramètre formel "F" lors du rattachement de l'arbre LOCAL. Cette substitution n'intervient que dans l'arbre local. | | | | | |

### 14.13.5 *Transfert de contraintes paramétrées*

Il est possible de transférer des contraintes sous forme de paramètres aux modules de test. Si la contrainte possède une liste de paramètres formels, elle sera transférée avec une liste de paramètres effectifs. Les paramètres effectifs de la contrainte seront déjà valués au point de rattachement.

*Exemple 79* – Transfert d'une contrainte paramétrée

Supposons que la contrainte C1 possède un seul paramètre formel de type INTEGER, TOP_TREE rattache STEP et transfère C1 sous forme de paramètre. A noter que la référence à la contrainte dans STEP n'est pas paramétrée:

```
TOP_TREE
    :
    +STEP(C1(3))
    :
```

```
STEP(PAR:A_PDU)
    :
    !A_PDU    PAR
    :
```

14.13.6 *Rattachement récursif d'arbre*

Le rattachement d'arbre fonctionnant récursivement (STEP pouvant contenir une ligne +SOME_OTHER_TREE), la sémantique de développement d'un arbre peut ne jamais aboutir à un arbre exempt de lignes de rattachement.

*Exemple 80* – Rattachement récursif autorisé d'un arbre

```
TOP_TREE
A
    +STEP
    B
```

et

```
STEP
C
        +TOP_TREE
D
```

un développement équivaut à:

```
TOP_TREE
A
    C
        +TOP_TREE
            B
    D
        B
```

Un arbre ne peut se rattacher lui-même, directement ou indirectement, à son niveau supérieur d'indentation.

*Remarque* – Il n'est pas nécessaire de développer ni un module de test qui ne sera pas exécuté, ni des propositions SI situées au-delà du niveau d'indentation courant tant qu'une proposition SI de ce niveau n'aura pas été sélectionnée.

*Exemple 81* – Rattachement récursif non autorisé d'un arbre

```
TOP_TREE
A
    +STEP
    B
```

et

```
STEP
C
      D
+TOP_TREE
```

un développement équivaut à:

```
TOP_TREE
A
C
    D
      B
+TOP_TREE
    B
```

14.13.7 *Rattachement d'arbre et comportements par défaut*

Les comportements par défaut dans un arbre seront développés avant tout rattachement de celui-ci (voir le § 14.18.2).

*Remarque* – L'utilisation simultanée dans une description comportementale de rattachements d'arbre et de comportements par défaut nécessite une attention particulière.

14.14 *Etiquettes et construction GOTO (saut)*

Toute ligne de déclaration de l'arbre comportemental peut recevoir une étiquette dans la colonne étiquettes.

*Remarque* – Chaque fois qu'une entrée étiquetée est exécutée dans l'arbre comportemental, il faut enregistrer son étiquette qui sera consignée dans le journal de conformité de façon à pouvoir l'associer à l'enregistrement de l'exécution de cette entrée.

Il est possible dans un arbre comportemental de spécifier une construction de saut GOTO vers une étiquette à condition que cette étiquette soit associée à la première proposition d'un ensemble de propositions concurrentes dont l'une forme un nœud antécédent du point depuis lequel l'instruction de saut GOTO sera exécutée. Seuls sont autorisés les sauts à l'intérieur d'un même arbre, c'est-à-dire à l'intérieur d'un arbre racine de test élémentaire, d'un arbre de module de test, d'un arbre de comportement par défaut ou d'un arbre local. Par conséquent, toute étiquette utilisée dans une construction GOTO se trouvera dans l'arbre même où est située cette construction. Les sauts vers le premier niveau d'indentation des arbres locaux, des modules de test et des comportements par défaut ne sont pas autorisés.

La construction GOTO est spécifiée par une flèche (–>) ou le mot clé GOTO, suivis du nom de l'étiquette, et occupera une ligne de déclaration indépendante dans l'arbre comportemental.

*Définition syntaxique*

295 GoTo ::= ("–>" | **GOTO)** Label

Une étiquette est unique dans un arbre. Si une construction GOTO est exécutée, le test élémentaire procède au traitement de l'ensemble de propositions SI auquel l'étiquette se réfère.

Les constructions GOTO sont toujours inconditionnelles et donc toujours exécutées.

*Remarque* – Il est possible de placer une expression booléenne comme antécédent immédiat d'une construction de saut GOTO pour obtenir l'effet d'un saut conditionnel.

*Exemple 82* – Utilisation de la construction de saut GOTO

| **Comportement dynamique de test élémentaire** | | | | | |
|---|---|---|---|---|---|
| **Nom du test élémentaire** : GOTO_EX1<br>**Groupe** : TTCN_EXAMPLES/GOTO_EXAMPLE1/<br>**Objectif** : Illustrer l'utilisation d'étiquettes avec la construction GOTO<br>**Comportement par défaut :**<br>**Commentaires :** | | | | | |
| **N°** | **Etiquette** | **Description comportementale** | **Réf. des contraintes** | **Verdict** | **Commentaires** |
| 1 | LA | !A | A1 | | |
| 2 | LB | ?B | B1 | | |
| 3 | LB2 | +B-tree | | | |
| 4 | LC | ?C | C1 | | |
| 5 | LD | [D=1] | | | |
| 6 | | –>LA | | | |
| 7 | LE | [E=1] | | | |
| 8 | LF | !F | F1 | | |
| **Commentaires détaillés:** Cet exemple illustre un saut vers l'étiquette LA. Il est également possible de sauter de cette position vers LB ou LD, mais pas vers LB2 ou LF (car les ensembles correspondants de propositions SI ne contiennent pas de nœud antécédent du point de saut, ni vers LC ou LE (car ces étiquettes ne sont pas les premières d'un ensemble de propositions SI). | | | | | |

14.15 *Construction REPEAT (répétition)*

Ce point décrit un mécanisme utilisable dans les descriptions comportementales pour exécuter un module de test de manière itérative. Cette construction obéit à la syntaxe suivante:

*Définition syntaxique*

297 Repeat ::= **REPEAT** TreeReference [ActualParList] UNTIL Qualifier

La référence d'arbre TreeReference pointera vers un arbre local ou à un module de test défini dans la bibliothèque des modules de test. Se reporter au § 14.13 pour les règles de rattachement. La signification de la construction REPEAT est la suivante: tout d'abord, l'arbre désigné par la référence d'arbre est exécuté. Puis le qualificateur est évalué. S'il prend la valeur «vrai» (TRUE), la construction de répétition REPEAT est achevée. Sinon, l'exécution de l'arbre reprend, suivie de l'évaluation du qualificateur. Ce traitement se répète jusqu'à ce que le qualificateur prenne la valeur «vrai» (TRUE).

Une construction REPEAT peut toujours être exécutée; elle constituera normalement la dernière proposition SI d'une série de déclarations TTCN du même niveau d'indentation, ainsi que le prévoit le § 14.9.5.3 a).

*Remarque* – Il est recommandé d'utiliser si possible la construction de répétition REPEAT de préférence à la construction GOTO.

*Exemple 83* – Utilisation de la construction de répétition REPEAT (voir aussi l'annexe D)

| **Comportement dynamique de test élémentaire** | | | | | |
|---|---|---|---|---|---|
| **Nom du test élémentaire :** RPT_EX1<br>**Groupe :** TTCN_EXAMPLES/REPEAT_EXAMPLE1/<br>**Objectif :** Illustrer l'utilisation de la construction REPEAT<br>**Comportement par défaut :**<br>**Commentaires :** | | | | | |
| **N°** | **Etiquette** | **Description comportementale** | **Réf. des contraintes** | **Verdict** | **Commentaires** |
| 1<br>2<br>3<br>4<br><br><br>5<br>6 | | (FLAG:=FALSE)<br>!A<br>REPEAT STEP1 (FLAG) UNTIL [FLAG]<br>!D<br><br>STEP1 (F:BOOLEAN)<br>?B (F:=TRUE)<br>?C (F:=FALSE) | <br>A1<br><br>D1<br><br><br>B1<br>C1 | <br><br><br>PASS | |
| **Commentaires détaillés:** Cet exemple décrit un test pouvant recevoir toutes sortes de messages au point PCO du testeur inférieur jusqu'à la réception du message B attendu. | | | | | |

14.16 *Référence de contraintes*

14.16.1 *Objectif de la colonne référence de contraintes*

Cette colonne permet de se référer à une contrainte particulière imposée à une primitive ASP ou à une unité PDU. Ces contraintes sont définies dans la partie «contraintes» (voir les § 11, 12 et 13). La référence de contrainte sera présente avec des événements SEND (envoi), IMPLICIT SEND (envoi implicite) et RECEIVE (réception). Elle est facultative lorsqu'une primitive ASP n'a pas de paramètres; elle n'accompagnera aucune autre sorte de déclaration TTCN.

Une référence de contrainte obéit à la syntaxe suivante:

*Définition syntaxique*

```
277  ConsRef ::= ConstraintIdentifier [ActualCrefParList]
278  ActualCrefParList ::= "(" ActualCrefPar {Comma ActualCrefPar} ")"
279  ActualCrefPar ::= Value | DataObjectIdentifier | ConsRef
```

*Exemple 84* – Référence de contrainte sans liste de paramètres

| | | | | |
|---|---|---|---|---|
| | N_SAP? CR_PDU | CR1 | | |

14.16.2 *Transfert de paramètres dans des références de contraintes*

Une référence de contrainte peut avoir une liste de paramètres facultatifs permettant de manipuler des valeurs de contraintes particulières depuis l'arbre comportemental.

La liste des paramètres effectifs respectera les conditions suivantes:

a) le nombre des paramètres effectifs sera le même que celui des paramètres formels;

b) chaque paramètre effectif prendra une valeur du type du paramètre formel correspondant.

Si une contrainte est transférée en tant que paramètre effectif et qu'elle soit déclarée avec une liste de paramètres formels, elle comportera également une liste de paramètres effectifs (éventuellement imbriquée). Toutes les variables apparaissant dans la liste de paramètres seront valuées au moment de l'utilisation de cette contrainte. Si une variable non valuée est utilisée, il s'agit d'une erreur de test élémentaire.

*Exemple 85* – Référence de contrainte avec une liste de paramètres

| | | | | |
|---|---|---|---|---|
| | N_SAP? N_DATAreq | D1(P1, CR1(P2)) | | |

Où D1 est une contrainte portant sur la primitive N_DATAreq (demande de données_réseau) et comportant deux paramètres (les paramètres effectifs P1 et CR1) et où CR1 est une contrainte comportant un paramètre (le paramètre effectif P2).

14.16.3 *Contraintes, qualificateurs et affectations*

Si un événement est qualifié et comporte aussi une référence de contrainte, on considère qu'il concorde si et seulement si le qualificateur *et* la contrainte sont respectés.

Si un événement est suivi d'une affectation et comporte une référence de contrainte ou un qualificateur, on l'interprétera comme suit: l'affectation sera exécutée si et seulement si l'événement se produit conformément à la définition donnée ci-dessus.

14.17 *Verdicts*

14.17.1 *Introduction*

Dans les tables de comportement dynamique, les entrées de la colonne verdict sont:

– soit un résultat préliminaire, indiqué entre parenthèses;

– soit un verdict final explicite.

Aucune entrée de l'un de ces types n'apparaîtra sur une ligne vide ou dans les déclarations TTCN suivantes:

a) construction de rattachement ATTACH;

b) construction de répétition REPEAT;

c) construction de saut GOTO;

d) envoi implicite IMPLICIT SEND

*Définition syntaxique*

```
281  Verdict ::= Pass | Fail | Inconclusive | Result
282  Pass ::= PASS | P | "(" PASS ")" | "(" P ")"
283  Fail ::= FAIL | F | "(" FAIL ")" | "(" F ")"
284  Inconclusive ::= INCONC | I | "(" INCONC ")" | "(" I ")"
285  Result ::= Identifier
```

*Remarque* – Pendant l'exécution d'un test élémentaire, chaque fois que dans l'arbre comportemental un verdict est associé à une entrée exécutée, il sera consigné dans le journal de conformité de façon à être associé à l'enregistrement de cette entrée dans l'arbre comportemental.

14.17.2 *Résultats préliminaires*

La variable prédéfinie R sert dans chaque test élémentaire à enregistrer tout résultat intermédiaire. Elle peut prendre les valeurs *pass* (succès), *fail* (échec), *inconc* (non concluant) et *none* (néant). Ces valeurs sont des identificateurs prédéfinis pour lesquels, par conséquent, la différence minuscules-majuscules est significative.

R peut s'utiliser partout où on peut utiliser d'autres variables de test élémentaire, sauf dans le membre gauche d'une déclaration d'affectation. Il s'agit donc d'une variable accessible en lecture seulement, à l'exception des modifications apportées à sa valeur dans la colonne verdict (selon les spécifications ci-dessous).

Un résultat préliminaire spécifié dans la colonne verdict, aura l'une des trois valeurs suivantes:

a) **(P)** ou **(PASS)**, signifiant que l'objectif du test est atteint sous un certain aspect;

b) **(I)** ou **(INCONC)**, signifiant que quelque chose a rendu le test élémentaire non concluant par rapport à un aspect de l'objectif du test;

c) **(F)** ou **(FAIL)**, signifiant qu'une erreur de protocole s'est produite ou qu'un certain aspect de l'objectif du test a échoué.

*Remarque* – PASS ou P, FAIL ou F, INCONC ou I sont des mots clés qui ne s'utilisent que dans les colonnes verdict. Les identificateurs prédéfinis *pass*, *fail*, *inconc* et *none* sont des valeurs représentant le contenu possible de la variable prédéfinie R. Ces identificateurs prédéfinis ne s'utilisent que pour tester la variable R dans des lignes comportementales.

Chaque fois qu'un résultat préliminaire est consigné à la suite de l'exécution de l'entrée correspondante dans l'arbre comportemental, la valeur de la variable prédéfinie R du test élémentaire est modifiée conformément au tableau 6/X.292:

TABLEAU 6/X.292

**Calcul de la valeur de la variable R**

| Valeur courante de R | Entrée dans la colonne verdict | | |
|---|---|---|---|
| | **(PASS)** | **(INCONC)** | **(FAIL)** |
| none | pass | inconc | fail |
| pass | pass | inconc | fail |
| inconc | inconc | inconc | fail |
| fail | fail | fail | fail |

*Remarque* – L'ordre de priorité (inférieur → supérieur) est donc: N, P, I, F. Même si R a la valeur *fail*, il peut être utile d'enregistrer un résultat préliminaire P ou I pour consigner dans le journal de conformité que P ou I peut convenir pour certains aspects de l'objectif du test, bien que ces valeurs ne modifient pas celle de R.

14.17.3 *Verdict final*

S'il faut le spécifier, un verdict final explicite dans la colonne verdict, prendra une des valeurs suivantes:

a) **P** ou **PASS**, signifiant qu'il faut consigner un verdict de succès;

b) **I** ou **INCONC**, signifiant qu'il faut consigner un verdict non concluant;

c) **F** ou **FAIL**, signifiant qu'il faut consigner un verdict d'échec;

d) la variable prédéfinie R, signifiant que la valeur de R est le verdict final, sauf si elle est égale à *none*, auquel cas on enregistre une erreur de test élémentaire à la place du verdict final.

TABLEAU 7/X.292

**Détermination du verdic final R**

| Valeur courante de R | Entrée dans la colonne verdict | | | |
|---|---|---|---|---|
| | **PASS** | **INCONC** | **FAIL** | R |
| none | pass | inconc | fail | *error* |
| pass | pass | inconc | fail | pass |
| inconc | *error* | inconc | fail | inconc |
| fail | *error* | *error* | fail | fail |

Chaque fois qu'un verdict final explicite est spécifié pendant l'exécution d'un test élémentaire, celui-ci prend fin. Pour être conforme aux dispositions de la Recommandation X.291, un verdict final explicite n'est spécifié que si le test élémentaire a repris un état stable approprié (par exemple, l'état de test au repos).

*Remarque* – Il est nécessaire que la spécification d'un verdict final explicite puisse mettre fin à un test élémentaire, lorsqu'on atteint par exemple un état stable dans un module de test rattaché et qu'un comportement ultérieur est spécifié dans l'arbre d'appel.

Si on atteint la feuille de l'arbre comportemental sans qu'un verdict final explicite soit spécifié, le verdict final est alors déterminé comme dans le cas d) ci-dessus (c'est-à-dire comme si on avait introduit la variable R dans la colonne verdict).

S'il faut enregistrer un verdict final explicite autre que R, il faut alors le comparer à la valeur de R pour déterminer s'ils sont cohérents. Si la valeur de R est *fail*, un verdict final **PASS** ou **INCONC** est alors considéré comme incohérent; si la valeur de R est *inconc*, un verdict final **PASS** est considéré comme incohérent. Chacune de ces incohérences constitue une erreur de test élémentaire.

*Remarque* – Dans ce cas, on consigne le message «erreur de test élémentaire» dans le journal de conformité.

14.17.4 *Verdicts et déclaration OTHERWISE (sinon)*

Une déclaration OTHERWISE (sinon) n'aboutira pas à un verdict de succès; elle devrait plutôt se terminer par un verdict d'échec, car elle correspond généralement à un événement de test invalide.

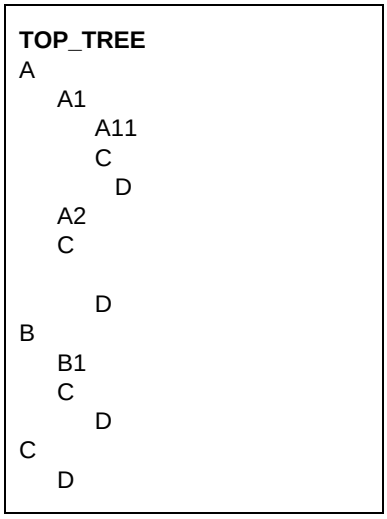
14.18 *Signification des comportements par défaut*

14.18.1 *Introduction*

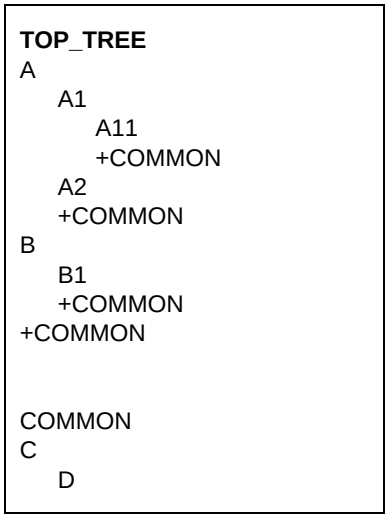
On utilise très souvent un comportement par défaut pour privilégier un ensemble de trajets intéressants dans un test en déclarant comme comportements par défaut les propositions SI communes moins intéressantes (ainsi que leurs comportements consécutifs).

Le même effet pourrait être obtenu, mais avec une concision moindre, en rattachant un module de test (par exemple +DEFAULT) comme dernière proposition SI générale supplémentaire. Contrairement au rattachement d'arbre, le comportement par défaut se développe en de nombreux points de l'arbre auquel il est associé. Cette propriété impose une utilisation attentive des comportements par défaut.

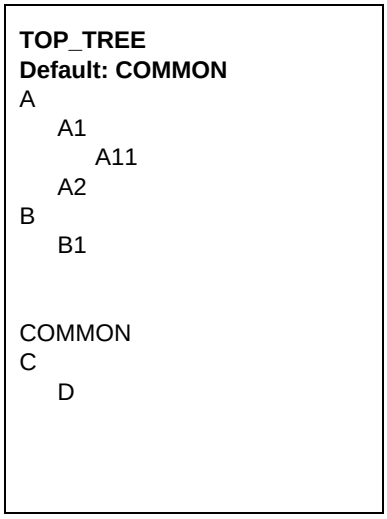
*Exemple 86* – Identification d'un arbre de comportement par défaut




1: ensemble complet des propositions SI




2: rattachement explicite d'arbre




3: un comportement par défaut donne le même résultat que 2

Aucun comportement par défaut n'est spécifié pour un autre, c'est-à-dire qu'un comportement par défaut ne peut avoir lui-même de comportement par défaut. Le rattachement d'arbres ne sera pas utilisé dans des arbres de comportement par défaut, c'est-à-dire que des arbres de comportement par défaut ne peuvent pas se rattacher des modules de test. Les tests élémentaires et les modules de test ne peuvent être désignés comme des comportements par défaut.

L'exécution d'un test élémentaire ne nécessite pas le développement des comportements par défaut dans tous les arbres qui s'y réfèrent. Cela apparaît dans la description opératoire de la signification des comportements par défaut: quand une concordance est recherchée dans une séquence de propositions SI (ce qui peut nécessiter des tentatives répétées) et qu'aucune n'est trouvée, on essaie également le premier niveau d'indentation des propositions SI du comportement par défaut. Si aucune d'elles ne concorde, on répète l'opération avec de nouvelles valeurs de temporisateurs et de files d'attente en tous les points PCO concernés. Si une concordance est trouvée dans le comportement par défaut, on le poursuit en ce point.

Pour s'assurer qu'aucun comportement ultérieur n'interviendra après l'exécution d'un comportement par défaut, on affecte un verdict final à chaque feuille de l'arbre de comportement par défaut. Si le verdict final est associé à une déclaration OTHERWISE dans l'arbre de comportement par défaut, le verdict final sera FAIL.

### 14.18.2 *Comportements par défaut et rattachement d'arbre*

Lorsqu'on utilise un rattachement d'arbre, il est important de bien comprendre comment les comportements par défaut se raccrochent tant à l'arbre d'appel qu'au module de test rattaché. Pour éviter les effets collatéraux masqués, les comportements par défaut applicables au module de test rattaché sont définis comme étant ceux qui sont spécifiés dans la table définissant ce module de test. Ainsi, si le module de test est défini dans la bibliothèque des modules de test, les comportements par défaut qui lui sont applicables sont spécifiés dans l'en-tête de la table comportementale du module de test. Ou alors, si le module de test est défini localement dans la même table comportementale que l'arbre d'appel, les mêmes comportements par défaut s'appliquent à la fois à l'arbre d'appel et au module de test rattaché.

Pour éviter la multiplication des insertions de comportements par défaut dans un ensemble de propositions SI, le comportement par défaut spécifié pour un arbre donné ne s'applique pas au niveau d'indentation supérieur des propositions SI de cet arbre, sauf s'il s'agit de l'arbre racine d'un test élémentaire.

Pour développer correctement un arbre, il est nécessaire de développer les comportements par défaut à la fois:

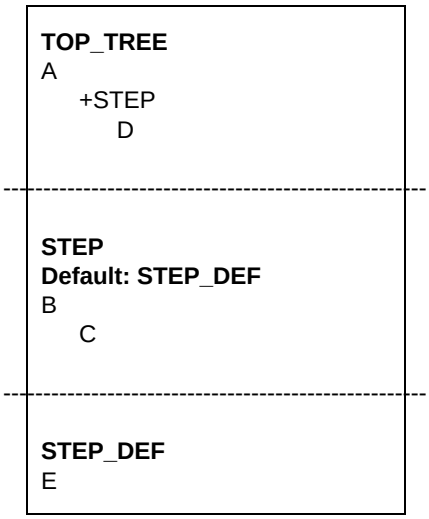
a) avant de développer l'arbre en tant qu'arbre rattaché; et

b) avant de développer les modules de test rattachés de cet arbre.

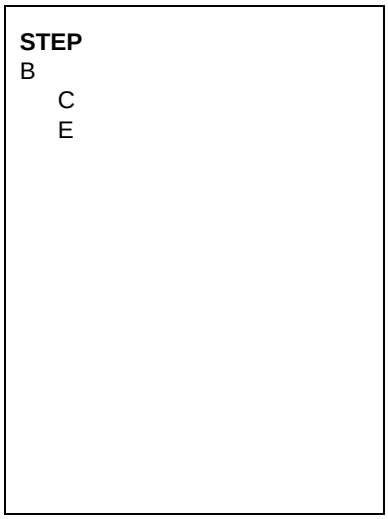
Le développement des comportements par défaut s'effectue donc localement pour un arbre simple et comprend le rattachement de l'arbre de comportement par défaut au bas de chacun des ensembles de propositions SI de cet arbre (à l'exception du niveau supérieur d'indentation pour les arbres autres que l'arbre racine d'un test élémentaire).

Les règles de développement des comportements par défaut s'appliquent de la même manière à un ensemble de propositions SI contenant un événement OTHERWISE.

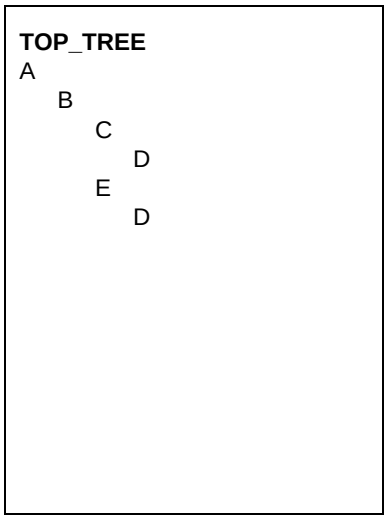
*Exemple 87* – Caractère local d'un comportement par défaut par rapport à un module de test




1: TOP_TREE rattache STEP, qui a le comportement par défaut STEP_DEF

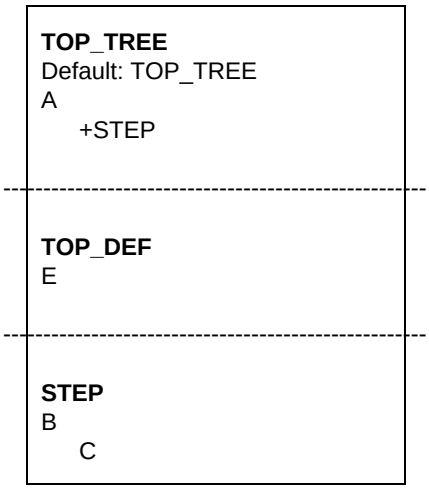



2: Développement de STEP_DEF dans STEP

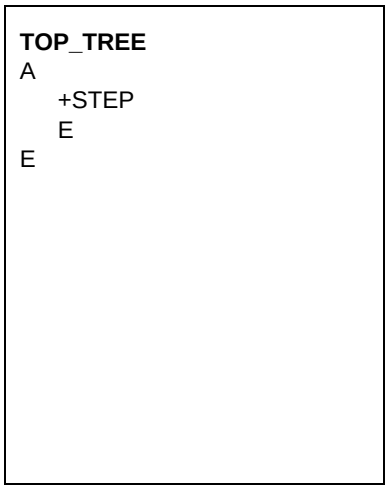



3: Développement de STEP dans TOP_TREE

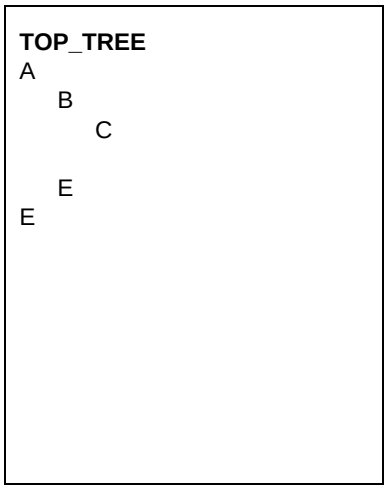
*Exemple 88* – Caractère local d'un comportement par défaut par rapport à un arbre d'appel




1: TOP_TREE rattache STEP. Le comportement par défaut de TOP_TREE est TOP_DEF

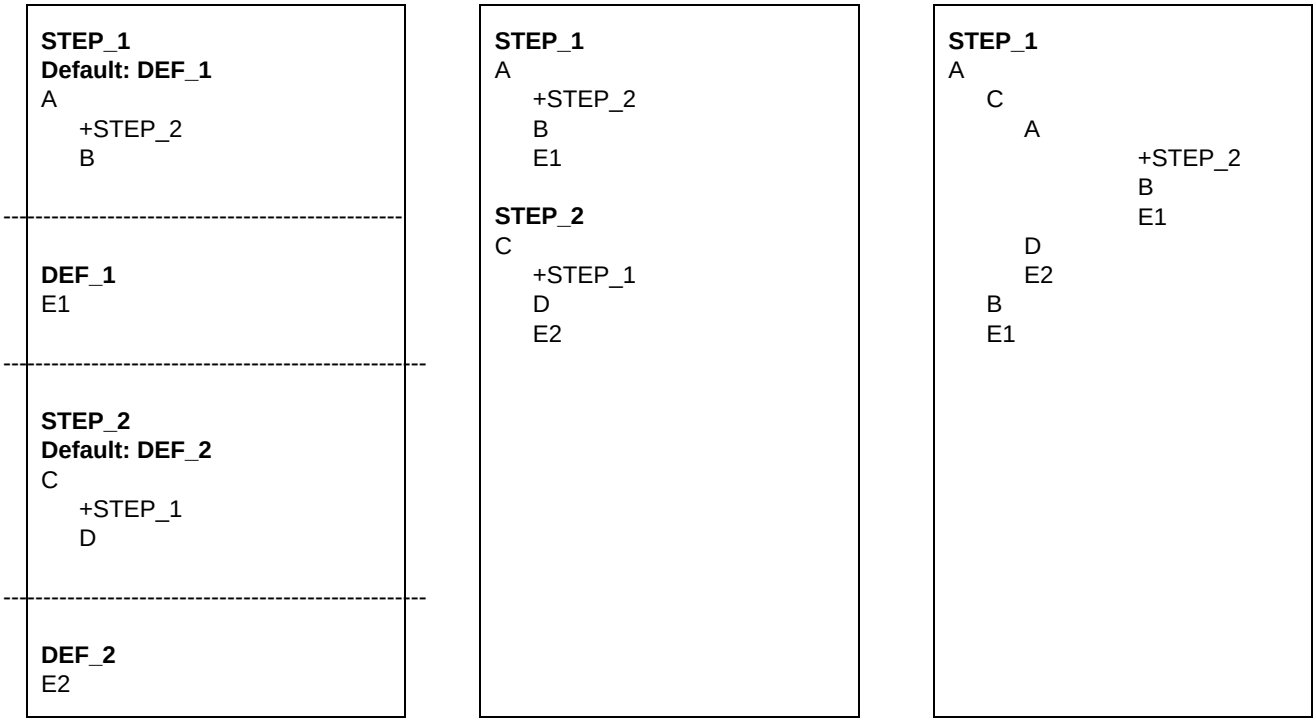



2: Développement de TOP_DEF dans TOP_TREE




3: Développement de STEP dans TOP_TREE

*Exemple 89* – Cas d'un rattachement cyclique d'arbre




1: STEP_1 et STEP_2 sont rattachés l'un à l'autre. Le comportement par défaut de STEP_1 est DEF_1, celui de STEP_2 est DEF_2

2: Développement de DEF_1 dans STEP_1 et de DEF_2 dans STEP_2

3: Après un développement de STEP_2 sans comportement par défaut et un développement de STEP_1 sans comportement par défaut

*Remarque* – Il est déconseillé d'utiliser ces rattachements cycliques.

## 14.19 *Références de comportement par défaut*

Les comportements des tests élémentaires et des modules de test font référence à un comportement par défaut de la bibliothèque des comportements par défaut au niveau de la rubrique Default (comportement par défaut) de l'en-tête de la table. Cette référence localise ce comportement par défaut à l'aide de son identificateur unique. Il est possible de paramétrer des comportements par défaut. La liste des paramètres effectifs doit respecter les conditions suivantes:

a) le nombre des paramètres effectifs sera le même que celui des paramètres formels;

b) chaque paramètre effectif prendra une valeur du type du paramètre formel correspondant;

c) toutes les variables apparaissant dans la liste des paramètres seront valuées au moment où la contrainte sera appelée.

*Définition syntaxique*

238 DefaultReference ::= DefaultIdentifier [ActualParList]

L'identificateur du comportement par défaut DefaultIdentifier pointera un comportement par défaut défini dans la bibliothèque des comportements par défaut.

*Exemple 90* – Référence de comportement par défaut

**90.1**

| **Comportement dynamique de test élémentaire** | | | | | |
|---|---|---|---|---|---|
| **Nom du test élémentaire :** DEF_EX1<br>**Groupe :** TTCN_EXAMPLES/DEFAULT_EXAMPLE1/<br>**Objectif :** Illustrer l'utilisation des comportements par défaut<br>**Comportement par défaut :** DEF1 (L)<br>**Commentaires :** Il est possible de fractionner l'arbre de l'exemple** en ce test élémentaire plus le comportement par défaut DEF1 | | | | | |
| **N°** | **Etiquette** | **Description comportementale** | **Réf. des contraintes** | **Verdict** | **Commentaires** |
| 1 | | L!CONNECTrequest | CR1 | | Demande de connexion |
| 2 | | L?CONNECTconfirm | CC1 | | Confirmation de connexion |
| 3 | | L!DATArequest | DTR1 | | Envoi de données |
| 4 | | L?DATAindication | DTI1 | | Réception de données |
| 5 | | L!DISCONNECTrequest | DSC1 | PASS | Acceptation |

**90.2**

| **Comportement dynamique de comportement par défaut** | | | | | |
|---|---|---|---|---|---|
| **Nom du test élémentaire :** DEF1(X:XSAP)<br>**Groupe :** TTCN_EXAMPLES/DEFAULTS_LIB/DEFAULT_1/<br>**Objectif :** Illustrer un comportement par défaut simple<br>**Commentaires :** | | | | | |
| **N°** | **Etiquette** | **Description comportementale** | **Réf. des contraintes** | **Verdict** | **Commentaires** |
| 1 | | X?DISCONNECTindication | DSC2 | INCONC | Prématuré |

*Remarque* – Syntaxiquement, dans l'exemple ci-dessus le comportement par défaut de la seconde table rattache l'indication X?DISCONNECTindication comme proposition SI à chacune des déclarations L! et L? de la première table. Toutefois, le rattachement de l'arbre de comportement par défaut en tant que proposition SI à une déclaration L! dont le résultat est toujours «vrai» n'a pas de sens.

## 15 Suite de page

### 15.1 *Suite de page dans les tables en notation TTCN*

Lorsqu'une table TTCN est trop longue pour tenir sur une seule page, on utilise le mécanisme suivant:

a) on imprime les mots «suite page suivante» *après* la ligne de la table à laquelle la coupure intervient;

b) on imprime les mots «suite de la page précédente» *avant* la suite de la table sur la page suivante.

Il est possible de couper les tables en un endroit quelconque, c'est-à-dire dans l'en-tête, dans le corps ou dans le cadre de bas de page. Dans tous les cas, les titres des sections (par exemple, les en-têtes des colonnes) sont répétés sur la nouvelle page. Il n'est pas obligatoire de reproduire l'en-tête complet.

*Exemple 91* – Coupure d'une table de paramètres dans une suite de tests

| Déclarations des paramètres de la suite de tests | | | |
|---|---|---|---|
| **Nom du paramètre** | **Type** | **Réf. PICS/PIXIT** | **Commentaires** |
| PAR1<br>PAR2<br>PAR3 | INTEGER<br>BOOLEAN<br>IA5String | Question aa du PICS<br>Question bb du PICS<br>Question cc du PIXIT | |

Suite page suivante page *n*

---

Suite de la page précédente page *n* + 1

| Déclarations des paramètres de la suite de tests | | | |
|---|---|---|---|
| **Nom du paramètre** | **Type** | **Réf. PICS/PIXIT** | **Commentaires** |
| PAR4<br>PAR5 | BOOLEAN<br>HEXSTRING | Question dd du PICS<br>Question ee du PICS | |

15.2 *Suite d'une page dans les tables de comportement dynamique*

Lorsqu'il est nécessaire de couper une table de comportement dynamique, on utilise l'un des deux mécanismes suivants:

a) la modularisation, c'est-à-dire la spécification d'une partie de l'arbre comportemental comme un module de test (non local) de bibliothèque, ce qui permet de modulariser l'arbre et de réduire, pour le formulaire courant, le nombre des comportements à celui qui peut figurer sur une seule page;

b) le mécanisme de suite de page, c'est-à-dire dans le cas des tables de comportement dynamique, la présentation des informations supplémentaires suivantes, qui permettent de faciliter l'alignement des niveaux d'indentation:

   1) l'impression, avant les mots «suite page suivante», du niveau d'indentation (entre crochets) de la dernière déclaration TTCN avant la coupure de page;

   2) sur la nouvelle page, l'impression, après les mots «suite de la page précédente», du niveau d'indentation (entre crochets) de la première déclaration TTCN de la suite de la table.

Lorsque des tests élémentaires sont longs, il peut être nécessaire d'adopter un niveau d'indentation apparent différent de celui déclaré. Dans ce cas, le niveau d'indentation déclaré, indiqué entre crochets, sera aligné avec une valeur d'indentation choisie pour la première ligne de déclaration de la suite de la table. D'autres indications concernant les niveaux d'indentation peuvent également être données pour en faciliter encore l'alignement.

*Exemple 92* – Coupure de page avec réalignement de l'indentation à l'aide d'indicateurs entre crochets

| **Comportement dynamique de test élémentaire** | | | | | |
|---|---|---|---|---|---|
| **Nom du test élémentaire :** SPLIT2<br>**Groupe :** TTCN_EXAMPLES/PAGE_SPLITTING2/<br>**Objectif :** Expliquer une coupure de page dans une table de comportement dynamique<br>**Comportement par défaut :**<br>**Commentaires :** | | | | | |
| **N°** | **Etiquette** | **Description comportementale** | **Réf. des contraintes** | **Verdict** | **Commentaires** |
| 1<br>2<br>3<br>4 | | ?A<br>?H1<br>?J<br>?K | CA<br>CH1<br>CJ<br>CK | | |

[1] [3]

Suite page suivante page *n*

---

Suite de la page précédente page *n* + 1

[1] [3]

| **N°** | **Etiquette** | **Description comportementale** | **Réf. des contraintes** | **Verdict** | **Commentaires** |
|---|---|---|---|---|---|
| 5<br>6<br>7 | | ?L<br>?M<br>?H2 | CL<br>CM<br>CH2 | <br>PASS<br>FAIL | |

A titre d'option, les rédacteurs des suites de tests peuvent utiliser une grille de tabulation en haut et en bas d'une table comportementale. Lorsqu'une page se poursuit sur la suivante, cette grille peut être décalée vers la gauche, permettant ainsi de décaler également l'indentation.

*Exemple 93* – Réalignement de l'indentation à l'aide de grilles de tabulation

| **Comportement dynamique de test élémentaire** | | | | | |
|---|---|---|---|---|---|
| **Nom du test élémentaire :** SPLIT2<br>**Groupe :** TTCN_EXAMPLES/PAGE_SPLITTING3/<br>**Objectif :** Expliquer une coupure de page dans une table de comportement dynamique à l'aide d'une grille de tabulation<br>**Comportement par défaut :**<br>**Commentaires :** | | | | | |
| **N°** | **Etiquette** | **Description comportementale**<br><br>0 1 2 3 | **Réf. des contraintes** | **Verdict** | **Commentaires** |
| 1<br>2<br>3<br>4 | | ?A<br>?H1<br>?J<br>?K<br>0 1 2 3 | CA<br>CH1<br>CJ<br>CK | | |

Suite page suivante page *n*

---

Suite de la page précédente page *n* + 1

| **N°** | **Etiquette** | **Description comportementale**<br><br>1 2 3 4 ... | **Réf. des contraintes** | **Verdict** | **Commentaires** |
|---|---|---|---|---|---|
| 5<br>6<br>7 | | ?L<br>?M<br>?H2<br>1 2 3 4 ... | CL<br>CM<br>CH2 | PASS<br>FAIL | |

ANNEXE A

(à la Recommandation X.292)

(Cette annexe fait partie intégrante de la présente Recommandation)

**Syntaxe et sémantique statique de la notation TTCN**

## A.1 *Introduction*

La présente annexe définit la syntaxe et la sémantique statique de la notation TTCN. Cette notation se présente sous deux formes, une forme graphique (TTCN.GR) et une forme informatisable (TTCN.MP). Pour l'utilisateur, la forme graphique de la notation TTCN, la notation TTCN.GR, a l'avantage d'une interprétation visuelle facile à comprendre. Toutefois, cette forme de notation ne se prête pas facilement au traitement par machine. La notation TTCN.MP répond à cette question et poursuit les objectifs suivants:

a) offrir une syntaxe formelle destinée à la notation TTCN dans la forme Backus-Naur (BNF);

b) servir de syntaxe de transfert;

c) faciliter la génération automatique des suites de tests exécutables (ETS) à partir des suites de tests abstraites (ATS);

d) autre traitement informatique.

*Remarque* – La génération automatique des suites ETS n'entre pas dans le cadre de la présente Recommandation.

La présente annexe définit également la sémantique statique des notations TTCN.GR et TTCN.MP.

## A.2 *Conventions appliquées à la description de la syntaxe*

### A.2.1 *Métanotation syntaxique*

Le tableau A-1/X.292 définit la métanotation utilisée pour spécifier l'extension de la forme de la grammaire BNF à la notation TTCN (dorénavant appelée «forme BNF»):

TABLEAU A-1/X.292

**Métanotation syntaxique de la notation TTCN.MP**

| | |
|---|---|
| ::= | Est par définition |
| \| | Ou |
| [abc] | 0 ou 1 instance d'abc |
| {abc} | 0 ou plusieurs instances d'abc |
| {abc}+ | 1 ou plusieurs instances d'abc |
| (...) | Groupement textuel |
| abc | Symbole non terminal abc |
| **abc** | Symbole terminal abc |
| "abc" | Symbole terminal abc |

### A.2.2 *Définitions de la syntaxe de la notation TTCN.MP*

A.2.2.1 Les tables complètes définies en notation TTCN.GR sont représentées en notation TTCN.MP par des productions du type:

**$Begin_KEYWORD .... .... .... $End_KEYWORD**

*Exemple A.1* – TS_PARdcls ::= $Begin_TS_PARdcls {TS_PARdcl}+ $End_TS_PARdcls

Normalement, ces productions comprennent au moins un champ obligatoire.

A.2.2.2 Tant les ensembles de lignes d'une table que les lignes individuelles (à savoir les ensembles de champs dans une table) sont représentés par des productions du type:

**$KEYWORD .... .... .... .... .... $End_KEYWORD**

**Begin** n'apparaît pas dans le mot clé ouvrant.

*Exemple A.2* – TS_PARdcl ::= **$TS_PARdcl** TS_PARid TS_PARtype PICS_PIXIT [Comment] **$END_TS_PARdcl**

A.2.2.3 Les champs individuels d'une ligne sont représentés par:

**$KEYWORD .... .... .... .... .... .... ....**

Il n'y a pas de mot clé fermant.

*Exemple A.3* – TS_Parlid ::= **$TS_Parld** TS_Parlidentifier

*Exemple A.4* – TS_Parlidentifier ::= Identifier

A.2.2.4 Les ensembles de tables jusqu'à et y compris la suite de tests, sont représentés par des productions du type:

**$KEYWORD .... .... .... .... .... $End_KEYWORD**

*Exemple A.5* – ASP_TypeDefs ::= **$ASP_TypeDefs** [TTCN_ASP_TypeDefs] [ASN1_ASP_TypeDefs] **$End_ASP_TypeDefs**

A.2.2.5 Pour toutes les autres productions définissant des symboles non terminaux, le membre droit de l'affectation ne comportera de mot clé ni à son début, ni à sa fin.

*Exemple A.6* – TimerIdentifier ::= Identifier

## A.3 *Productions syntaxiques en notation TTCN.MP dans la forme BNF*

### A.3.1 *Suite de tests*

1 Suite ::= **$Suite** SuiteId SuiteOverviewPart DeclarationsPart ConstraintsPart DynamicPart **$End_Suite**

2 SuiteId ::= **$SuiteId** SuiteIdentifier

3 SuiteIdentifier ::= Identifier

### A.3.2 *Description générale de la suite de tests*

#### A.3.2.1 *Considérations générales*

4 SuiteOverviewPart ::= **$SuiteOverviewPart** SuiteStructure TestCaseIndex [TestStepIndex] [DefaultIndex] **$End_SuiteOverviewPart**

#### A.3.2.2 *Structure de la suite de tests*

5 SuiteStructure ::= **$Begin_SuiteStructure** SuiteId StandardsRef PICSref PIXITref TestMethods [Comment] Structure&Objectives [Comment] **$End_SuiteStructure**

6 StandardsRef ::= **$StandardsRef** BoundedFreeText

7 PICSref ::= **$PICSref** BoundedFreeText

8 PIXITref ::= **$PIXITref** BoundedFreeText

9 TestMethods ::= **$TestMethods** BoundedFreeText

10 Comment ::= **$Comment** [BoundedFreeText]

11 Structure&Objectives ::= **$Structure&Objectives** {Structure&Objective} **$End_Structure&Objectives**

12 Structure&Objective ::= **$Structure&Objective** TestGroupRef SelExprId Objective **$End_Structure&Objective**

13 SelExprId ::= **$SelectExprId** [SelectExprIdentifier]

A.3.2.3 *Indice des tests élémentaires*

14 TestCaseIndex ::= **$Begin_TestCaseIndex** {CaseIndex}+ [Comment] **$End_TestCaseIndex**

15 CaseIndex ::= **$CaseIndex** TestGroupRef TestCaseId SelExprId Description **$End_CaseIndex**

/*SÉMANTIQUE STATIQUE – Les tests élémentaires doivent être énumérés dans l'ordre où ils apparaissent dans la partie dynamique*/

/*SÉMANTIQUE STATIQUE – Une référence explicite de groupe de tests doit être fournie pour le premier test élémentaire de chaque groupe de modules de test*/

/*SÉMANTIQUE STATIQUE – Une référence explicite de groupe de tests doit être fournie pour chaque test élémentaire qui suit immédiatement un groupe de modules de test*/

16 Description ::= **$Description** BoundedFreeText

A.3.2.4 *Indice des modules de test*

17 TestStepIndex ::= **$Begin_TestStepIndex** {StepIndex} [Comment] **$End_TestStepIndex**

18 StepIndex ::= **$StepIndex** TestStepRef TestStepId Description **$End_StepIndex**

/*SÉMANTIQUE STATIQUE – L'identificateur de module de test TestStepId ne comportera pas de liste de paramètres formels*/

/*SÉMANTIQUE STATIQUE – Les modules de test doivent être énumérés dans l'ordre où ils apparaissent dans la partie dynamique*/

/*SÉMANTIQUE STATIQUE – Une référence explicite de groupe de modules de test doit être fournie pour le premier module de test de chaque groupe de modules de test*/

/*SÉMANTIQUE STATIQUE – Une référence explicite de groupe de modules de test doit être fournie pour chaque module de test qui suit immédiatement un groupe de modules de test*/

A.3.2.5 *Indice par défaut*

19 DefaultIndex ::= **$Begin_DefaultIndex** {DefIndex} [Comment] **$End_DefaultIndex**

20 DefIndex ::= **$DefIndex** DefaultRef DefaultId Description **$DefIndex**

/*SÉMANTIQUE STATIQUE – L'identificateur par défaut DefaultId ne comportera pas de liste de paramètres formels*/

/*SÉMANTIQUE STATIQUE – Les indices par défaut seront énumérés dans l'ordre où ils apparaissent dans la partie dynamique*/

/*SÉMANTIQUE STATIQUE – Une référence explicite de groupe par défaut sera fournie pour le premier indice par défaut de chaque groupe de comportements par défaut*/

/*SÉMANTIQUE STATIQUE – Une référence explicite de groupe par défaut sera fournie pour chaque indice par défaut qui suit immédiatement un groupe de comportements par défaut*/

A.3.3 *Partie déclarative*

A.3.3.1 *Considérations générales*

21 DeclarationsPart ::= **$DeclarationsPart** Definitions Parameterization&Selection Declarations ComplexDefinitions **$End_DeclarationsPart**

A.3.3.2 *Définitions*

A.3.3.2.1 *Considérations générales*

22 Definitions ::= [TS_TypeDefs] [TS_OpDefs]

A.3.3.2.2 *Définitions de type de suite de tests*

23 TS_TypeDefs ::= **$TS_TypeDefs** [SimpleTypeDefs] [StructTypeDefs] [ASN1_TypeDefs] [ASN1_TypeRefs] **$End_TS_TypeDefs**

A.3.3.2.3 *Définitions de type simple*

24 SimpleTypeDefs ::= **$Begin_SimpleTypeDefs** {SimpleTypeDef}+ [Comment] **$End_SimpleTypeDefs**

25 SimpleTypeDef ::= **$SimpleTypeDef** SimpleTypeId SimpleTypeDefinition [Comment] **$End_SimpleTypeDef**

26 SimpleTypeId ::= **$SimpleTypeId** SimpleTypeIdentifier

27 SimpleTypeIdentifier ::= Identifier

28 SimpleTypeDefinition ::= **$SimpleTypeDefinition** Type&Restriction

29 Type&Restriction ::= Type [Restriction]

30 Restriction ::= LengthRestriction | IntegerRange | SimpleValueList

/*SÉMANTIQUE STATIQUE – L'ensemble des valeurs définies par Restriction formera un sous-ensemble vrai des valeurs du type de base*/

31 LengthRestriction ::= SingleTypeLength | RangeTypeLength

/*SÉMANTIQUE STATIQUE – L'attribut LengthRestriction (restriction de longueur) ne sera fourni que si le type de base est un type chaîne (c'est-à-dire BITSTRING (CHAÎNE BINAIRE), HEXSTRING (CHAÎNE HEXADÉCIMALE), OCTETSTRING (CHAÎNE D'OCTETS) ou CharacterString (chaîne de caractères) ou dérivé d'un type chaîne*/

32 SingleTypeLength ::= "[" Number "]"

33 RangeTypeLength ::= "[" LowerTypeBound To UpperTypeBound "]"

34 IntegerRange ::= "(" LowerTypeBound To UpperTypeBound ")"

/*SÉMANTIQUE STATIQUE – La valeur LowerTypeBound (borne inférieure) doit être inférieure à la valeur UpperTypeBound (borne supérieure)*/

35 LowerTypeBound ::= [Minus] Number | Minus **INFINITY**

36 UpperTypeBound ::= [Minus] Number | **INFINITY**

37 To ::= TO | ".."

38 SimpleValueList ::= "(" [Minus] LiteralValue {Comma [Minus] LiteralValue} ")"

/*SÉMANTIQUE STATIQUE – Si la valeur Minus (signe moins) est utilisée dans l'attribut SimpleValueList (liste des valeurs simples), la valeur littérale (LiteralValue) doit être un nombre*/

/*SÉMANTIQUE STATIQUE – Les valeurs LiteralValues doivent appartenir au type de base et former un sous-ensemble vrai des valeurs définies par le type de base*/

A.3.3.2.4 *Définitions de type structuré*

39 StructTypeDefs ::= **$StructTypeDefs** {StructTypeDef}+ **$End_StructTypeDefs**

40 StructTypeDef ::= **$Begin_StructTypeDef** StructId [Comment] ElemDcls [Comment] **$End_StructTypeDef**

41 StructId ::= **$StructId** StructId&FullId

42 StructId&FullId ::= StructIdentifier [FullIdentifier]

43 FullIdentifier ::= "(" BoundedFreeText ")"

/*SÉMANTIQUE STATIQUE – Certains objets en notation TTCN permettent l'abréviation des dénominations telles qu'elles apparaissent dans la Recommandation de protocole appropriée. Si une abréviation est utilisée, l'identificateur complet FullIdentifier doit figurer dans la déclaration de l'objet*/

44 StructIdentifier ::= Identifier

45 ElemDcls ::= **$ElemDcls** {ElemDcl}+ **$End_ElemDcls**

46 ElemDcl ::= **$ElemDcl** ElemId ElemType [Comment] **$End_ElemDcl**

47 ElemId ::= **$ElemId** ElemId&FullId

48 ElemId&FullId ::= ElemIdentifier [FullIdentifier]

49 ElemIdentifier ::= Identifier

50 ElemType ::= **$ElemType** Type&Attributes

/*SÉMANTIQUE STATIQUE – Il ne peut y avoir de références récursives (ni directement ni indirectement) dans l'attribut Types&Attributes (types et attributs)*/

/*SÉMANTIQUE STATIQUE – Un type d'élément de structure doit être un type prédéfini PredefinedType, un identificateur de type de suite de tests TS_TypeIdentifier, un identificateur d'unité PDU PDU_Identifier ou une unité PDU*/

A.3.3.2.5 *Définitions de type de suite de tests en notation ASN.1*

51 ASN1_TypeDefs ::= **$ASN1_TypeDefs** {ASN1_TypeDef}+ **$End_ASN1_TypeDefs**

52 ASN1_TypeDef ::= **$Begin_ASN1_TypeDef** ASN1_TypeId [Comment] ASN1_TypeDefinition [Comment] **$End_ASN1_TypeDef**

53 ASN1_TypeId ::= **$ASN1_TypeId** ASN1_TypeId&FullId

54 ASN1_TypeId&FullId ::= ASN1_TypeIdentifier [FullIdentifier]

55 ASN1_TypeIdentifier ::= Identifier

56 ASN1_TypeDefinition ::= **$ASN1_TypeDefinition** ASN1_Type&LocalTypes **$End_ASN1_TypeDefinition**

57 ASN1_Type&LocalTypes ::= ASN1_Type {ASN1_LocalType}

/*SÉMANTIQUE STATIQUE – Les types mentionnés provenant de la définition de type en notation ASN.1, ASN1_Type, doivent être définis soit dans d'autres tables de définition de type en notation ASN.1, soit encore par référence dans la table des références à un type en notation ASN.1, soit enfin localement (c'est-à-dire les types locaux en notation ASN.1 ASN1_LocalTypes) dans la même table après la première définition de type*/

/*SÉMANTIQUE STATIQUE – Les types ASN1_LocalTypes ne doivent pas être utilisés dans d'autres parties de la suite de tests*/

58 ASN1_Type ::= Type

/*RÉFÉRENCE – Où Type est un symbole non terminal défini dans la Recommandation X.208*/

/*SÉMANTIQUE STATIQUE – Chaque référence terminale de type utilisée dans la production Type sera une des suivantes: une référence à un type ASN1_LocalType, un identificateur TS_TypeIdentifier ou un identificateur PDU_Identifier*/

/*SÉMANTIQUE STATIQUE – Les définitions de type en notation ASN.1 utilisées en notation TTCN ne doivent pas utiliser de références à un type externe, comme défini dans la Recommandation X.208*/

59 ASN1_LocalType ::= Typeassignment

/*RÉFÉRENCE – Où Typeassignment (affectation de type) est un symbole non terminal défini dans la Recommandation X.208*/

/*SÉMANTIQUE STATIQUE – Les définitions de type en notation ASN.1 utilisées en notation TTCN n'utiliseront pas de références à un type externe, comme défini dans la Recommandation X.208*/

A.3.3.2.6 *Types de suite de tests en notation ASN.1 par référence*

60 ASN1_TypeRefs ::= **$Begin_ASN1_TypeRefs** {ASN1_TypeRef}+ [Comment] **$End_ASN1_TypeRefs**

61 ASN1_TypeRef ::= **$ASN1_TypeRef** ASN1_TypeId ASN1_TypeReference ASN1_ModuleId [Comment] **$End_ASN1_TypeRef**

/*SÉMANTIQUE STATIQUE – L'identificateur ASN1_TypeId ne doit pas être déclaré en faisant usage d'un identificateur FullIdentifier*/

62 ASN1_TypeReference ::= **$ASN1_TypeReference** TypeReference

63 TypeReference ::= typereference

/*RÉFÉRENCE – Où la référence type TypeReference est définie dans le § 8.2 de la Recommandation X.208*/

64 ASN1_ModuleId ::= **$ASN1_ModuleId** ModuleIdentifier

65 ModuleIdentifier ::= ModuleIdentifier

/*RÉFÉRENCE – Où ModuleIdentifier est un identificateur non terminal défini dans la Recommandation X.208*/

/*SÉMANTIQUE STATIQUE – L'identificateur ModuleIdentifier doit être unique dans le domaine considéré*/

A.3.3.2.7 *Définitions d'opérations de suite de tests*

66 TS_OpDefs ::= **$TS_OpDefs** {TS_OpDef}+ **$End_TS_OpDefs**

67 TS_OpDef ::= **$Begin_TS_OpDef** TS_OpId TS_OpResult [Comment] TS_OpDescription [Comment] **$End_TS_OpDef**

68 TS_OpId ::= **$TS_OpId** TS_OpId&ParList

69 TS_OpId&ParList ::= TS_OpIdentifier [FormalParList]

/*SÉMANTIQUE STATIQUE – Le type de paramètre formel d'opération de suite de tests TS_OpIdentifier doit être un type PredefinedType, un identificateur TS_TypeIdentifier ou un identificateur PDU_Identifier*/

70 TS_OpIdentifier ::= Identifier

71 TS_OpResult ::= **$TS_OpResult** Type

/*SÉMANTIQUE STATIQUE – Type doit être un type PredefinedType, un identificateur TS_TypeIdentifier ou un identificateur PDU_Identifier*/

72 TS_OpDescription ::= **$TS_OpDescription** BoundedFreeText

A.3.3.3 *Paramétrage et sélection*

A.3.3.3.1 *Considérations générales*

73 Parameterization&Selection ::= [TS_ParDcls] [SelectExprDefs]

A.3.3.3.2 *Déclarations des paramètres de suite de tests*

74 TS_ParDcls ::= **$Begin_TS_ParDcls** {TS_ParDcl}+ [Comment] **$End_TS_ParDcls**

75 TS_ParDcl ::= **$TS_ParDcl** TS_ParId TS_ParType PICS_PIXITref [Comment] **$End_TS_ParDcl**

76 TS_ParId ::= **$TS_ParId** TS_ParIdentifier

77 TS_ParIdentifier ::= Identifier

78 TS_ParType ::= **$TS_ParType** Type

/*SÉMANTIQUE STATIQUE – Type doit être un type PredefinedType, un identificateur TS_TypeIdentifier ou un identificateur PDU_Identifier*/

79 PICS_PIXITref ::= **$PICS_PIXITref** BoundedFreeText

A.3.3.3.3 *Définitions de l'expression de sélection de test élémentaire*

80 SelectExprDefs ::= **$Begin_SelectExprDefs** {SelectExprDef}+ [Comment] **$End_SelectExprDefs**

81 SelectExprDef ::= **$SelectExprDef** SelectExprId SelectExpr [Comment] **$End_SelectExprDef**

82 SelectExprId ::= **$SelectExprId** SelectExprIdentifier

83 SelectExprIdentifier ::= Identifier

84 SelectExpr ::= **$SelectExpr** SelectionExpression

85 SelectionExpression ::= Expression

/*SÉMANTIQUE STATIQUE – L'expression de sélection (SelectionExpression) ne doit contenir que les valeurs LiteralValues, des identificateurs de paramètre de suite de tests (TS_ParIdentifiers), des identificateurs de constante de suite de tests (TS_ConstIdentifiers) et des identificateurs d'expression de sélection (SelectExprIdentifiers)*/

/*SÉMANTIQUE STATIQUE – L'expression (SelectionExpression) doit prendre une valeur booléenne spécifique*/

/*SÉMANTIQUE STATIQUE – L'expression ne peut se rapporter de manière récursive (ni directement, ni indirectement) à l'identificateur SelectExprIdentifier qu'elle définit*/

### A.3.3.4 *Déclarations*

#### A.3.3.4.1 *Considérations générales*

86 Declarations ::= [TS_ConstDcls] [TS_VarDcls] [TC_VarDcls] PCO_Dcls [TimerDcls]

#### A.3.3.4.2 *Déclarations de constantes de suite de tests*

87 TS_ConstDcls ::= **$Begin_TS_ConstDcls** {TS_ConstDcl}+ [Comment] **$End_TS_ConstDcls**

88 TS_ConstDcl ::= **$TS_ConstDcl** TS_ConstId TS_ConstType TS_ConstValue [Comment] **$End_TS_ConstDcl**

89 TS_ConstId ::= **$TS_ConstId** TS_ConstIdentifier

90 TS_ConstIdentifier ::= Identifier

91 TS_ConstType ::= **$TS_ConstType** Type

/*SÉMANTIQUE STATIQUE – Type doit être un PredefinedType, un identificateur TS_TypeIdentifier ou un identificateur PDU_Identifier*/

92 TS_ConstValue ::= **$TS_ConstValue** DeclarationValue

93 DeclarationValue ::= Expression

/*SÉMANTIQUE STATIQUE – La valeur de déclaration (DeclarationValue) ne contiendra que des littéraux (LiteralValues), des identificateurs TS_ParIdentifiers ou des identificateurs TS_ConstIdentifiers*/

/*SÉMANTIQUE STATIQUE – La valeur prise par DeclarationValue sera conforme à son type déclaré*/

#### A.3.3.4.3 *Déclarations de variables de suite de tests*

94 TS_VarDcls ::= **$Begin_TS_VarDcls** {TS_VarDcl}+ [Comment] **$End_TS_VarDcls**

95 TS_VarDcl ::= **$TS_VarDcl** TS_VarId TS_VarType TS_VarValue [Comment] **$End_TS_VarDcl**

96 TS_VarId ::= **$TS_VarId** TS_VarIdentifier

97 TS_VarIdentifier ::= Identifier

98 TS_VarType ::= **$TS_VarType** Type

/*SÉMANTIQUE STATIQUE – Type sera un type PredefinedType, un identificateur TS_TypeIdentifier ou un identificateur PDU_Identifier*/

99 TS_VarValue ::= **$TS_VarValue** [DeclarationValue]

A.3.3.4.4 *Déclarations de variables de test élémentaire*

100 TC_VarDcls ::= **$Begin_TC_VarDcls** {TC_VarDcl}+ [Comment] **$End_TC_VarDcls**

101 TC_VarDcl ::= **$TC_VarDcl** TC_VarId TC_VarType TC_VarValue [Comment] **$End_TC_VarDcl**

102 TC_VarId ::= **$TC_VarId** TC_VarIdentifier

103 TC_VarIdentifier ::= Identifier

104 TC_VarType ::= **$TC_VarType** Type

/*SÉMANTIQUE STATIQUE – Type sera un type PredefinedType, un identificateur TS_TypeIdentifier ou un identificateur PDU_Identifier*/

105 TC_VarValue ::= **$TC_VarValue** [DeclarationValue]

A.3.3.4.5 *Déclarations de points de contrôle et d'observation (PCO)*

106 PCO_Dcls ::= **$Begin_PCO_Dcls** {PCO_Dcl}+ [Comment] **$End_PCO_Dcls**

/*SÉMANTIQUE STATIQUE – Conformément à la Recommandation X.290, le nombre de points PCO correspondra à la méthode de test utilisée*/

107 PCO_Dcl ::= **$PCO_Dcl** PCO_Id PCO_TypeId P_Role [Comment] **$End_PCO_Dcl**

108 PCO_Id ::= **$PCO_Id** PCO_Identifier

109 PCO_Identifier ::= Identifier

110 PCO_TypeId ::= **$PCO_TypeId** PCO_TypeIdentifier

111 PCO_TypeIdentifier ::= Identifier

112 P_Role ::= **$PCO_Role** PCO_Role

113 PCO_Role ::= **UT** | **LT**

A.3.3.4.6 *Déclarations de temporisation*

114 TimerDcls ::= **$Begin_TimerDcls** {TimerDcl}+ [Comment] **$End_TimerDcls**

115 TimerDcl ::= **$TimerDcl** TimerId Duration Unit [Comment] **$End_TimerDcl**

116 TimerId ::= **$TimerId** TimerIdentifier

117 TimerIdentifier ::= Identifier

118 Duration ::= **$Duration** [DeclarationValue]

/*SÉMANTIQUE STATIQUE – DeclarationValue aura une valeur de type INTEGER (entière) strictement positive (non nulle)*/

119 Unit ::= **$Unit** TimeUnit

120 TimeUnit ::= **ps** | **ns** | **μs** | **ms** | **s** | **min**

/*SÉMANTIQUE STATIQUE – Si une temporisation est dérivée d'une déclaration de conformité d'instance de protocole (PICS) des informations supplémentaires sur l'instance de protocole destinée au test (PIXIT), la déclaration de temporisation doit alors spécifier les mêmes unités que l'entrée PICS/PIXIT*/

A.3.3.5 *Définitions de types de primitive ASP et de types d'unité PDU*

A.3.3.5.1 *Considérations générales*

121 ComplexDefinitions ::= [ASP_TypeDefs] PDU_TypeDefs [AliasDefs]

A.3.3.5.2 *Définitions de types de primitive de service abstrait (ASP)*

122 ASP_TypeDefs ::= **$ASP_TypeDefs** [TTCN_ASP_TypeDefs] [ASN1_ASP_TypeDefs] [ASN1_ASP_TypeDefsByRef] **$End_ASP_TypeDefs**

A.3.3.5.3 *Définitions de types de primitive ASP sous forme tabulaire*

123 TTCN_ASP_TypeDefs ::= **$TTCN_ASP_TypeDefs** {TTCN_ASP_TypeDef}+ **$End_TTCN_ASP_TypeDefs**

124 TTCN_ASP_TypeDef ::= **$Begin_TTCN_ASP_TypeDef** ASP_Id PCO_Type [Comment] ASP_ParDcls [Comment] **$End_TTCN_ASP_TypeDef**

125 PCO_Type ::= **$PCO_Type** [PCO_TypeIdentifier]

/*SÉMANTIQUE STATIQUE – L'identificateur de type de point PCO PCO_TypeIdentifier doit être vu des types de points PCO utilisés dans le formulaire de déclaration de point PCO*/

/*SÉMANTIQUE STATIQUE – Si un seul point PCO est défini dans une suite de tests, l'identificateur PCO_TypeIdentifier est facultatif*/

126 ASP_Id ::= **$ASP_Id** ASP_Id&FullId

127 ASP_Id&FullId ::= ASP_Identifier [FullIdentifier]

128 ASP_Identifier ::= Identifier

129 ASP_ParDcls ::= **$ASP_ParDcls** {ASP_ParDcl} **$End_ASP_ParDcls**

130 ASP_ParDcl ::= **$ASP_ParDcl** ASP_ParId ASP_ParType [Comment] **$End_ASP_ParDcl**

131 ASP_ParId ::= **$ASP_ParId** ASP_ParIdOrMacro

132 ASP_ParIdOrMacro ::= ASP_ParId&FullId I MacroSymbol

/*SÉMANTIQUE STATIQUE – Un MacroSymbol ne sera utilisé qu'en combinaison avec une référence à un type structuré*/

133 ASP_ParId&FullId ::= ASP_ParIdentifier [FullIdentifier]

134 ASP_ParIdentifier ::= Identifier

135 ASP_ParType ::= **$ASP_ParType** Type&Attributes

/*SÉMANTIQUE STATIQUE – Type doit être un type PredefinedType, un identificateur TS_TypeIdentifier, un identificateur PDU_Identifier ou une unité **PDU**/

A.3.3.5.4 *Définitions de types de primitive ASP en notation ASN.1*

136 ASN1_ASP_TypeDefs ::= **$ASN1_ASP_TypeDefs** {ASN1_ASP_TypeDef} **$End_ASN1_ASP_TypeDefs**

137 ASN1_ASP_TypeDef ::= **$Begin_ASN1_ASP_TypeDef** ASP_Id PCO_Type [Comment] ASN1_TypeDefinition [Comment] **$End_ASN1_ASP_TypeDef**

A.3.3.5.5 *Définitions de types de primitive ASP en notation ASN.1 par référence*

138 ASN1_ASP_TypeDefsByRef ::= **$Begin_ASN1_ASP_TypeDefsByRef** {ASN1_ASP_TypeDefByRef}+ [Comment] **$End_ASN1_ASP_TypeDefsByRef**

139 ASN1_ASP_TypeDefByRef ::= **$ASN1_ASP_TypeDefByRef** ASP_Id PCO_Type ASN1_TypeReference ASN1_ModuleId [Comment] **$End_ASN1_ASP_TypeDefByRef**

/*SÉMANTIQUE STATIQUE – L'identificateur ASP ASP_Id ne doit pas être déclaré en faisant usage de l'identificateur FullIdentifier*/

A.3.3.5.6 *Définitions de types d'unité PDU*

140 PDU_TypeDefs ::= **$PDU_TypeDefs** [TTCN_PDU_TypeDefs] [ASN1_PDU_TypeDefs] [ASN1_PDU_TypeDefsByRef] **$End_PDU_TypeDefs**

A.3.3.5.7 *Définitions de types d'unité PDU sous forme tabulaire*

141 TTCN_PDU_TypeDefs ::= **$TTCN_PDU_TypeDefs** {TTCN_PDU_TypeDef}+ **$End_TTCN_PDU_TypeDefs**

142 TTCN_PDU_TypeDef ::= **$Begin_TTCN_PDU_TypeDef** PDU_Id PCO_Type [Comment] PDU_FieldDcls [Comment] **$End_TTCN_PDU_TypeDef**

/*SÉMANTIQUE STATIQUE – Si une unité PDU est envoyée ou reçue en étant seulement incluse dans des primitives ASP à l'intérieur d'une suite entière de tests, l'identificateur PCO_TypeIdentifier (dans le type PCO_Type) est facultatif*/

143 PDU_Id ::= **$PDU_Id** PDU_Id&FullId

144 PDU_Id&FullId ::= PDU_Identifier [FullIdentifier]

145 PDU_Identifier ::= Identifier

146 PDU_FieldDcls ::= **$PDU_FieldDcls** {PDU_FieldDcl}+ **$End_PDU_FieldDcls**

147 PDU_FieldDcl ::= **$PDU_FieldDcl** PDU_FieldId PDU_FieldType [Comment] **$End_PDU_FieldDcl**

148 PDU_FieldId ::= **$PDU_FieldId** PDU_FieldIdOrMacro

149 PDU_FieldIdOrMacro ::= PDU_FieldId&FullId I MacroSymbol

/*SÉMANTIQUE STATIQUE – Le MacroSymbol ne doit être utilisé qu'en combinaison avec une référence à un type structuré*/

150 MacroSymbol ::= "<–"

151 PDU_FieldId&FullId ::= PDU_FieldIdentifier [FullIdentifier]

152 PDU_FieldIdentifier ::= Identifier

153 PDU_FieldType ::= **$PDU_FieldType** Type&Attributes

/*SÉMANTIQUE STATIQUE – Type doit être un type PredefinedType, un identificateur TS_TypeIdentifier, un identificateur PDU_Identifier ou une unité **PDU**/

154 Type&Attributes ::= (Type [LengthAttribute]) I **PDU**

/*SÉMANTIQUE STATIQUE – L'ensemble des valeurs définies par l'attribut ValueLength doit être un sous-ensemble vrai des valeurs du type de base*/

/*SÉMANTIQUE STATIQUE – L'attribut LengthAttribute ne sera indiqué que si le type de base est un type chaîne (à savoir BITSTRING, HEXSTRING, OCTETSTRING ou CharacterString) ou s'il en dérive*/

155 LengthAttribute ::= SingleLength I RangeLength

156 SingleLength ::= "[" Bound "]"

157 Bound ::= Number I TS_ParIdentifier I TS_ConstIdentifier

/*SÉMANTIQUE STATIQUE – L'attribut Bound (borne) prendra une valeur non négative de type INTEGER (entier) ou la valeur INFINITY (infini)*/

158 RangeLength ::= "[" LowerBound To UpperBound "]"

/*SÉMANTIQUE STATIQUE – La valeur de LowerBound (borne inférieure) doit être inférieure à celle de UpperBound (borne supérieure)*/

159 LowerBound ::= Bound

160 UpperBound ::= Bound I **INFINITY**

A.3.3.5.8 *Définitions de types d'unité PDU en notation ASN.1*

161 ASN1_PDU_TypeDefs ::= **$ASN1_PDU_TypeDefs** {ASN1_PDU_TypeDef} **$End_ASN1_PDU_TypeDefs**

162 ASN1_PDU_TypeDef ::= **$Begin_ASN1_PDU_TypeDef** PDU_Id PCO_Type [Comment] ASN1_TypeDefinition [Comment] **$End_ASN1_PDU_TypeDef**

/*SÉMANTIQUE STATIQUE – Si une unité PDU est envoyée ou reçue en étant incluse seulement dans des primitives ASP à l'intérieur d'une suite entière de tests, l'identificateur PCO_TypeIdentifier (dans le type PCO_Type) est facultatif*/

A.3.3.5.9 *Définitions de types d'entité PDU en notation ASN.1 par référence*

163 ASN1_PDU_TypeDefsByRef ::= **$Begin_ASN1_PDU_TypeDefsByRef** {ASN1_PDU_TypeDefByRef}+ [Comment] **$End_ASN1_PDU_TypeDefsByRef**

164 ASN1_PDU_TypeDefByRef ::= **$ASN1_PDU_TypeDefByRef** PDU_Id PCO_Type ASN1_TypeReference ASN1_ModuleId [Comment] **$End_ASN1_PDU_TypeDefByRef**

/*SÉMANTIQUE STATIQUE – Si une unité PDU est envoyée ou reçue en étant seulement incluse dans des primitives ASP à l'intérieur d'une suite entière de tests, l'identificateur PCO_TypeIdentifier (dans le type PCO_Type) est facultatif*/

/*SÉMANTIQUE STATIQUE – L'identificateur PDU_Id ne doit pas être déclaré avec un identificateur FullIdentifier*/

A.3.3.5.10 *Définitions des alias (pseudonymes)*

165 AliasDefs ::= **$Begin_AliasDefs** {AliasDef}+ [Comment] **$End_AliasDefs**

166 AliasDef ::= **$AliasDef** AliasId ExpandedId [Comment] **$End_AliasDef**

167 AliasId ::= **$AliasId** AliasIdentifier

168 AliasIdentifier ::= Identifier

/*SÉMANTIQUE STATIQUE – Les identificateurs d'alias (AliasIdentifier) ne seront utilisés que sur une ligne de déclaration d'une description de comportement*/

/*SÉMANTIQUE STATIQUE – L'identificateur AliasIdentifier ne sera utilisé que pour un identificateur ASP_Identifier ou PDU_Identifier valide*/

169 ExpandedId ::= **$ExpandedId** Expansion

170 Expansion ::= ASP_Identifier I PDU_Identifier

A.3.4 *Partie contraintes*

A.3.4.1 *Considérations générales*

171 ConstraintsPart ::= **$ConstraintsPart** [TS_TypeConstraints] [ASP_Constraints] [PDU_Constraints] **$End_ConstraintsPart**

A.3.4.2 *Déclarations de contraintes relatives au type de suite de tests*

172 TS_TypeConstraints ::= **$TS_TypeConstraints** [StructTypeConstraints] [ASN1_TypeConstraints] **$End_TS_TypeConstraints**

A.3.4.3 *Déclarations de contraintes de type structuré*

173 StructTypeConstraints ::= **$StructTypeConstraints** {StructTypeConstraint}+ **$End_StructTypeConstraints**

174 StructTypeConstraint ::= **$Begin_StructTypeConstraint** ConsId StructId DerivPath [Comment] ElemValues [Comment] **$End_StructTypeconstraint**

/*SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur de structure (Struct_Id) ne sera pas utilisé*/

/*SÉMANTIQUE STATIQUE – Une contrainte modifiée aura la même liste de paramètres que sa contrainte de base. En particulier, aucun paramètre ne sera omis ou ajouté à cette liste*/

175 Elem Values ::= **$ElemValues** {ElemValue}+ **$End_ElemValues**

176 ElemValue ::= **$ElemValue** ElemId ConsValue [Comment] **$End_ElemValue**

/*SÉMANTIQUE STATIQUE – Les valeurs ElemValue paramétrées dans une contrainte de base ne seront ni modifiées ni explicitement omises dans une contrainte modifiée*/

A.3.4.4 *Déclarations de contraintes de type en notation ASN.1*

177 ASN1_TypeConstraints ::= **$ASN1_TypeConstraints** {ASN1_TypeConstraint}+
**$End_ASN1_TypeConstraints**

178 ASN1_TypeConstraint ::= **$Begin_ASN1_TypeConstraint** ConsId ASN1_TypeId DerivPath [Comment]
ASN1_ConsValue [Comment] **$End_ASN1_TypeConstraint**

/*SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur ASN1_TypeId ne sera pas utilisé*/

/*SÉMANTIQUE STATIQUE – Une contrainte modifiée aura la même liste de paramètres que sa contrainte de base. En particulier, aucun paramètre ne sera omis ou ajouté à cette liste*/

A.3.4.5 *Déclarations des contraintes de primitive ASP*

179 ASP_Constraints ::= **$ASP_Constraints** [TTCN_ASPConstraints] [ASN1_ASP_Constraints]
**$End_ASP_Constraints**

A.3.4.6 *Déclarations des contraintes de primitive ASP sous forme tabulaire*

180 TTCN_ASP_Constraints ::= **$TTCN_ASP_Constraints** {TTCN_ASP_Constraint}+
**$End_TTCN_ASP_Constraints**

181 TTCN_ASP_Constraint ::= **$Begin_TTCN_ASP_Constraint** ConsId ASP_Id DerivPath [Comment]
ASP_ParValues [Comment] **$End_TTCN_ASP_Constraint**

/*SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur ASP_Id ne sera pas utilisé*/

/*SÉMANTIQUE STATIQUE – La spécification de contrainte d'une primitive ASP aura la même structure que celle de la définition de type de cette primitive ASP*/

/*SÉMANTIQUE STATIQUE – Pour chaque définition de type de primitive ASP, une contrainte de base, et au moins une, sera spécifiée*/

/*SÉMANTIQUE STATIQUE – Une contrainte modifiée aura la même liste de paramètres que sa contrainte de base. En particulier, aucun paramètre ne sera omis ou ajouté à cette liste*/

182 ASP_ParValues ::= **$ASP_ParValues** {ASP_ParValue}+ **$End_ASP_ParValues**

183 ASP_ParValue ::= **$ASP_ParValue** ASP_ParId ConsValue [Comment] **$End_ASP_ParValue**

/*SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur ASP_ParId ne sera pas utilisé*/

/*SÉMANTIQUE STATIQUE – Si une définition de primitive ASP fait référence à un type structuré en tant que sous-structure d'un paramètre (c'est-à-dire avec un nom de paramètre), la contrainte correspondante aura le même nom de paramètre dans la position correspondante de la colonne nom du paramètre de la contrainte et la valeur sera une référence à une contrainte s'appliquant à ce paramètre (c'est-à-dire s'appliquant à cette sous-structure conformément à la définition du type structuré)*/

/*SÉMANTIQUE STATIQUE – Si une définition de primitive ASP fait référence à un paramètre spécifié comme étant du métatype PDU, alors, dans une contrainte correspondante, la valeur de ce paramètre sera spécifiée comme étant le nom d'une contrainte d'unité PDU ou d'un paramètre formel*/

/*SÉMANTIQUE STATIQUE – Dans une contrainte, on ne pourra utiliser des contraintes structurées par développement de macros que si la définition de primitive ASP correspondante renvoie aussi au même type structuré par développement de macro*/

/*SÉMANTIQUE STATIQUE – Les paramètres de primitive ASP paramétrée dans une contrainte de base ne seront ni modifiés ni explicitement omis dans une contrainte modifiée*/

A.3.4.7 *Déclarations de contraintes de primitive ASP en notation ASN.1*

184 ASN1_ASP_Constraints ::= **$ASN1_ASP_Constraints** {ASN1_ASP_Constraint}+
**$End_ASN1_ASP_Constraints**

185 ASN1_ASP_Constraint ::= **$Begin_ASN1_ASP_Constraint** ConsId ASP_Id DerivPath [Comment]
ASN1_ConsValue [Comment] **$End_ASN1_ASP_Constraint**

/*SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur ASP_Id ne sera pas utilisé*/

/*SÉMANTIQUE STATIQUE – Si une primitive ASP comporte une sous-structure, les contraintes s'appliquant à ces primitives ASP auront une structure ASN.1 compatible (c'est-à-dire comportant éventuellement certains groupements)*/

/*SÉMANTIQUE STATIQUE – Une contrainte modifiée aura la même liste de paramètres que la contrainte de base. En particulier, aucun paramètre ne sera omis ou ajouté à cette liste*/

A.3.4.8 *Déclarations de contraintes d'unité PDU*

186 PDU_Constraints ::= **$PDU_Constraints** [TTCN_PDU_Constraints] [ASN1_PDU_Constraints]
**$End_PDU_Constraints**

A.3.4.9 *Déclarations de contraintes d'unité PDU en notation tabulaire*

187 TTCN_PDU_Constraints ::= **$TTCN_PDU_Constraints** {TTCN_PDU_Constraint}+
**$End_TTCN_PDU_Constraints**

188 TTCN_PDU_Constraint ::= **$Begin_TTCN_PDU_Constraint** ConsId PDU_Id DerivPath [Comment]
PDU_FieldValues [Comment] **$End_TTCN_PDU_Constraint**

/*SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur PDU_Id ne sera pas utilisé*/

/*SÉMANTIQUE STATIQUE – Si une unité PDU comporte une sous-structure, les contraintes s'appliquant à ces unités PDU auront la même structure*/

/*SÉMANTIQUE STATIQUE – Pour chaque définition de type d'unité PDU, une contrainte de base, et au moins une, sera spécifiée*/

/*SÉMANTIQUE STATIQUE – Une contrainte modifiée aura la même liste de paramètres que sa contrainte de base. En particulier, aucun paramètre ne sera omis ou ajouté à cette liste*/

189 ConsId ::= **$ConsId** ConsId&ParList

190 ConsId&ParList ::= ConstraintIdentifier [FormalParList]

191 ConstraintIdentifier ::= Identifier

192 DerivPath ::= **$DerivPath** [DerivationPath]

193 DerivationPath ::= {ConstraintIdentifier Dot}+

/*SÉMANTIQUE STATIQUE – Si une définition de contrainte est une modification d'une contrainte existante, le nom de cette dernière figurera dans la table de l'entrée du trajet de dérivation*/

/*SÉMANTIQUE STATIQUE – Le premier identificateur de contrainte ConstraintIdentifier dans le (mode de dérivation) DerivationPath doit être un identificateur de contrainte de base*/

/*SÉMANTIQUE STATIQUE – Les contraintes doivent être énumérées dans l'ordre d'application des modifications qu'elles apportent à la contrainte de base*/

/*SÉMANTIQUE STATIQUE – Il n'y aura pas de blanc entre l'identificateur ConstraintIdentifier et la production Dot*/

194 PDU_FieldValues ::= **$PDU_FieldValues** {PDU_FieldValue}+ **$End_PDU_FieldValues**

195 PDU_FieldValue ::= **$PDU_FieldValue** PDU_FieldId ConsValue [Comment] **$End_PDU_FieldValue**

/*SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur PDU_FieldId ne sera pas utilisé*/

/*SÉMANTIQUE STATIQUE – Si une définition d'unité PDU fait référence à un type structuré en tant que sous-structure d'un champ (c'est-à-dire avec un nom de champ qui y fait référence), la contrainte correspondante aura le même nom de champ dans la position correspondante de la colonne nom du champ de la contrainte et la valeur sera une référence à une contrainte s'appliquant à ce champ (c'est-à-dire s'appliquant à cette sous-structure conformément à la définition du type structuré)*/

/*SÉMANTIQUE STATIQUE – Si une définition d'unité PDU fait référence à un champ spécifié comme étant du métatype PDU, alors, dans une contrainte correspondante, la valeur de ce champ sera spécifiée comme étant le nom d'une contrainte d'unité PDU ou d'un paramètre formel*/

/*SÉMANTIQUE STATIQUE – On ne pourra utiliser dans une contrainte des contraintes structurées par développement de macros que si la définition d'unité PDU correspondante renvoie aussi au même type structuré par développement de macro*/

/*SÉMANTIQUE STATIQUE – Les valeurs de champ d'unité PDU paramètrées dans une contrainte de base ne seront ni modifiées, ni explicitement omises dans une contrainte modifiée*/

196 ConsValue ::= **$ConsValue** ConstraintValue&Attributes

/*SÉMANTIQUE STATIQUE – ConsValue prendra pour valeur un élément du type spécifié pour le paramètre de primitive ASP, le champ d'unité PDU ou l'élément de structure*/

197 ConstraintValue&Attributes ::= ConstraintValue ValueAttributes

/*SÉMANTIQUE STATIQUE – La valeur de contrainte ConstraintValue respectera toutes les restrictions définies pour le paramètre de primitive ASP, pour le champ d'unité PDU ou pour le type d'élément de structure, notamment les plages de valeurs, les listes de valeurs, les restrictions alphabétiques et les restrictions de longueur*/

/*SÉMANTIQUE STATIQUE – Aucune spécification de longueur définie pour le paramètre de primitive ASP ou pour le type de champ d'unité PDU dans les déclarations de type de suite de tests ne doit contrevenir aux spécifications de longueur figurant dans la définition de type de primitive ASP ou d'unité PDU*/

/*SÉMANTIQUE STATIQUE – Ni les variables de suite de tests ni celles de tests élémentaires ne doivent être utilisées dans les contraintes, à moins d'être passées en tant que paramètres effectifs, auquel cas une valeur leur sera attribuée et elles ne seront plus modifiées*/

198 ConstraintValue ::= ConstraintExpression I MatchingSymbol I ConsRef

/*SÉMANTIQUE STATIQUE – Les littéraux LiteralValue, les identificateurs de paramètre de suite de tests (TS_ParIdentifier), de constante de suite de tests (TS_ConstIdentifier), de variable de suite de tests (TS_VarIdentifier), de variable de test élémentaire (TC_VarIdentifier), les (autres) références aux attributs et valeur de contrainte ConsRef et les identificateurs de paramètres formels (FormalParIdentifier) peuvent être passés en tant que paramètres effectifs dans le type ConsRef*/

199 ConstraintExpression ::= Expression

/*SÉMANTIQUE STATIQUE – Les termes de l'expression ConstraintExpression ne comporteront que des littéraux LiteralValue, des identificateurs TS_ParIdentifier, TS_ConstIdentifier et FormalParIdentifier et des OpCall (appels d'opérateur)*/

/*SÉMANTIQUE STATIQUE – L'expression ConstraintExpression prendra pour valeur un élément du type spécifié*/

200 MatchingSymbol ::= Complement I Omit I AnyValue I AnyOrOmit I ValueList I ValueRange I SuperSet I SubSet I Permutation

/**Remarque* – Aucun symbole de concordance n'est considéré comme valeur spécifique*/

201 Complement ::= **COMPLEMENT** ValueList

202 Omit ::= Dash I **OMIT**

/*SÉMANTIQUE STATIQUE – Dans les contraintes en notation ASN.1, l'attribut Omit (omission) ne sera utilisé que pour les paramètres de primitive ASP et champs d'unité PDU déclarés être de type OPTIONAL (facultatifs) ou DEFAULT (par défaut)*/

203 AnyValue ::= "?"

204 AnyOrOmit ::= "*"

205 ValueList ::= "(" ConstraintValue&Attributes {Comma ConstraintValue&Attributes} ")"

/*SÉMANTIQUE STATIQUE – Chaque attribut ConstraintValue&Attributes (attributs et valeur de contrainte) sera du type déclaré pour le paramètre de primitive ASP, pour le champ d'unité PDU ou pour l'élément de structure dans lequel la liste de valeurs ValueList est utilisée*/

206 ValueRange ::= "(" ValRange ")"

/*SÉMANTIQUE STATIQUE – L'attribut d'intervalle de valeurs (ValueRange) ne sera utilisé que pour les paramètres de primitive ASP, les champs d'unité PDU et les éléments de structure de type INTEGER (entier)*/

/*SÉMANTIQUE STATIQUE – L'ensemble des valeurs définies par l'intervalle ValueRange sera un sous-ensemble vrai des valeurs autorisées par le type déclaré du paramètre de primitive ASP, du champ d'unité PDU ou de l'élément de structure*/

207 ValRange ::= (LowerRangeBound To UpperRangeBound)

/*SÉMANTIQUE STATIQUE – La limite inférieure d'intervalle (LowerRangeBound), sera inférieure à la limite supérieure (UpperRangeBound)*/

208 LowerRangeBound ::= ConstraintExpression | Minus **INFINITY**

/*SÉMANTIQUE STATIQUE – L'expression ConstraintExpression prendra une valeur de type INTEGER (entière)*/

209 UpperRangeBound ::= ConstraintExpression | **INFINITY**

/*SÉMANTIQUE STATIQUE – L'expression ConstraintExpression prendra une valeur de type INTEGER (entière)*/

210 SuperSet ::= **SUPERSET** "(" ConstraintValue&Attributes ")"

/*SÉMANTIQUE STATIQUE – L'argument de l'attribut SuperSet (surensemble), c'est-à-dire ConstraintValue&Attributes, sera du type SET OF (ensemble de)*/

211 SubSet ::= **SUBSET** "(" ConstraintValue&Attributes ")"

/*SÉMANTIQUE STATIQUE – L'argument de l'attribut SubSet (sous-ensemble), c'est-à-dire ConstraintValue&Attributes, sera du type SET OF*/

212 Permutation ::= **PERMUTATION** ValueList

/*SÉMANTIQUE STATIQUE – L'attribut Permutation ne sera utilisé qu'à l'intérieur d'une valeur de type SEQUENCE OF (séquence de)*/

/*SÉMANTIQUE STATIQUE – La liste ValueList sera du type spécifié par SEQUENCE OF*/

213 ValueAttributes ::= [ValueLength] [**IF_PRESENT**]

/*SÉMANTIQUE STATIQUE – Dans les contraintes ASN.1, la déclaration IF_PRESENT (si présente) ne sera utilisée que pour les paramètres de primitive ASP et champs d'unité PDU déclarés OPTIONAL ou DEFAULT*/

214 ValueLength ::= SingleValueLength I RangeValueLength

/*SÉMANTIQUE STATIQUE – L'attribut de longueur ValueLength ne sera utilisé que pour les paramètres de primitive ASP, les champs d'unité PDU et les éléments de structure qui sont déclarés BITSTRING, HEXSTRING, OCTETSTRING, CharacterString, SEQUENCE OF ou SET OF*/

/*SÉMANTIQUE STATIQUE – L'attribut ValueLength ne sera utilisé qu'en combinaison avec les mécanismes suivants: SpecificValue (valeur spécifique), Complement (complément), Omit (omission), AnyValue (valeur quelconque), AnyOrOmit (quelconque ou omission), AnyOrNone (quelconque ou aucun) et Permutation (permutation)*/

/*SÉMANTIQUE STATIQUE – L'ensemble des valeurs définies par l'attribut ValueLength doit être un sous-ensemble vrai des valeurs autorisées par le type déclaré des paramètres de primitive ASP, des champs d'unité PDU et des éléments de structure*/

215 SingleValueLength ::= "[" ValueBound "]"

216 ValueBound ::= Number I TS_ParIdentifier I TS_ConstIdentifier I FormalParIdentifier

/*SÉMANTIQUE STATIQUE – L'attribut ValueBound (limite de valeur) prendra une valeur de type INTEGER (entière) non négative*/

217 RangeValueLength ::= "[" LowerValueBound To UpperValueBound "]"

/*SÉMANTIQUE STATIQUE – La limite inférieure LowerValueBound sera inférieure à la limite supérieure UpperValueBound*/

218 LowerValueBound ::= ValueBound

219 UpperValueBound ::= ValueBound I **INFINITY**

A.3.4.10 *Déclarations de contraintes de primitive ASP en notation ASN.1*

220 ASN1_PDU_Constraints ::= **$ASN1_PDU_Constraints** {ASN1_PDU_Constraint}+ **$End_ASN1_PDU_Constraints**

221 ASN1_PDU_Constraint ::= **$Begin_ASN1_PDU_Constraint** Consld PDU_Id DerivPath [Comment] ASN1_ConsValue [Comment] **$End_ASN1_PDU_Constraint**

/*SÉMANTIQUE STATIQUE – L'identificateur FullIdentifier qui fait partie de l'identificateur PDU_Id ne sera pas utilisé*/

/*SÉMANTIQUE STATIQUE – Si une unité PDU comporte une sous-structure, les contraintes s'appliquant à ces unités PDU auront une structure ASN.1 compatible (c'est-à-dire comportant éventuellement certains groupements)*/

/*SÉMANTIQUE STATIQUE – Une contrainte modifiée aura la même liste de paramètres que sa contrainte de base. En particulier, aucun paramètre ne sera omis ou ajouté à cette liste*/

222 ASN1_ConsValue ::= **$ASN1_ConsValue** ConstraintValue&AttributesOrReplace **$End_ASN1_ConsValue**

223 ConstraintValue&AttributesOrReplace ::= ConstraintValue&Attributes | Replacement {Comma Replacement}

224 Replacement ::= (**REPLACE** ReferenceList **BY** ConstraintValue&Attributes) | (**OMIT** ReferenceList)

/*SÉMANTIQUE STATIQUE – L'attribut Replacement (remplacement) ne sera utilisé que si l'attribut DerivPath est spécifié*/

/*SÉMANTIQUE STATIQUE – Les valeurs Replacement paramétrées dans une contrainte de base ne seront ni modifiées ni explicitement omises dans une contrainte modifiée*/

225 ReferenceList ::= (ArrayRef | ComponentIdentifier | ComponentPosition) {ComponentReference}

A.3.5 *Partie dynamique*

A.3.5.1 *Considérations générales*

226 DynamicPart ::= **$DynamicPart** TestCases [TestStepLibrary] [DefaultsLibrary] **$End_DynamicPart**

A.3.5.2 *Tests élémentaires*

227 TestCases ::= **$TestCases** ({TestGroup I TestCase}+) **$End_TestCases**

228 TestGroup ::= **$TestGroup** TestGroupId {TestGroup I TestCase}+ **$End_TestGroup**

229 TestGroupId ::= **$TestGroupId** TestGroupIdentifier

230 TestGroupIdentifier ::= Identifier

231 TestCase ::= **$Begin_TestCase** TestCaseId TestGroupRef TestPurpose DefaultRef [Comment] BehaviourDescription [Comment] **$End_TestCase**

232 TestCaseId ::= **$TestCaseId** TestCaseIdentifier

233 TestCaseIdentifier ::= Identifier

234 TestGroupRef ::= **$TestGroupRef** TestGroupReference

235 TestGroupReference ::= [SuiteIdentifier "/"] {TestGroupIdentifier "/"}

/*SÉMANTIQUE STATIQUE – Il n'y aura pas d'espace de chaque côté du signe "/"*/

236 TestPurpose ::= **$TestPurpose** BoundedFreeText

237 DefaultRef ::= **$DefaultRef** [DefaultReference]

238 DefaultReference ::= DefaultIdentifier [ActualParList]

A.3.5.3 *Bibliothèque des modules de test*

239 TestStepLibrary ::= **$TestStepLibrary** ({TestStepGroup I TestStep}+) **$End_TestStepLibrary**

240 TestStepGroup ::= **$TestStepGroup** TestStepGroupId {TestStepGroup I TestStep}+ **$End_TestStepGroup**

241 TestStepGroupId ::= **$TestStepGroupId** TestStepGroupIdentifier

242 TestStepGroupIdentifier ::= Identifier

243 TestStep ::= **$Begin_TestStep** TestStepId TestStepRef Objective DefaultsRef [Comment] BehaviourDescription [Comment] **$End_TestStep**

244 TestStepId ::= **$TestStepId** TestStepId&ParList

245 TestStepId&ParList ::= TestStepIdentifier [FormalParList]

246 TestStepIdentifier ::= Identifier

247 TestStepRef ::= **$TestStepRef** TestStepGroupReference

248 TestStepGroupReference ::= [SuiteIdentifier "/"] {TestStepGroupIdentifier "/"}

/*SÉMANTIQUE STATIQUE – Il n'y aura pas d'espace de chaque côté du signe "/"*/

249 Objective ::= **$Objective** BoundedFreeText

A.3.5.4 *Bibliothèque de comportements par défaut*

250 DefaultsLibrary ::= **$DefaultsLibrary** ({DefaultGroup I Default}+) **$End_DefaultsLibrary**

251 DefaultGroup ::= **$DefaultGroup** DefaultGroupId {DefaultGroup I Default}+ **$End_DefaultGroup**

252 DefaultGroupId ::= **$DefaultGroupId** DefaultGroupIdentifier

253 Default ::= **$Begin_Default** DefaultId DefaultRef Objective [Comment] BehaviourDescription [Comment] **$End_Default**

/*SÉMANTIQUE STATIQUE – La description de comportement (BehaviourDescription) sera constituée d'un arbre unique (c'est-à-dire qu'il n'y aura pas d'arbres locaux)*/

/*SÉMANTIQUE STATIQUE – La description BehaviourDescription n'utilisera pas le rattachement à un arbre (c'est-à-dire que les arbres de comportement par défaut ne rattacheront pas de modules de test)*/

/*SÉMANTIQUE STATIQUE – Un verdict final sera affecté à chaque feuille d'un arbre de comportements par défaut. Si ce verdict final résulte d'une déclaration OTHERWISE (sinon) dans l'arbre par défaut, il aura la valeur FAIL (échec)*/

254 DefaultRef ::= **$DefaultRef** DefaultGroupReference

255 DefaultId ::= **$DefaultId** DefaultId&ParList

256 DefaultId&ParList ::= DefaultIdentifier [FormalParList]

257 DefaultIdentifier ::= Identifier

258 DefaultGroupReference ::= [SuiteIdentifier "/"] {DefaultGroupIdentifier "/"}

/*SÉMANTIQUE STATIQUE – Il n'y aura pas d'espace de chaque côté du signe "/"*/

259 DefaultGroupIdentifier ::= Identifier

A.3.5.5 *Descriptions de comportement*

260 BehaviourDescription ::= **$BehaviourDescription** RootTree {LocalTree} **$End_BehaviourDescription**

261 RootTree ::= {BehaviourLine}+

262 LocalTree ::= Header {BehaviourLine}+

263 Header ::= **$Header** TreeHeader

264 TreeHeader ::= TreeIdentifier [FormalParList]

265 TreeIdentffier ::= Identifier

266 FormalParList ::= "(" FormalPar&Type {SemiColon FormalPar&Type} ")"

267 FormalPar&Type ::= FormalParIdentifier {Comma FormalParIdentifier} Colon FormalParType

268 FormalParIdentifier ::= Identifier

269 FormalParType ::= Type | PCO_TypeIdentifier | **PDU**

/*SÉMANTIQUE STATIQUE – Si un paramètre formel de module de test est de type **PDU**, les champs spécifiques de l'unité PDU ne doivent pas être mentionnés dans l'arbre de comportement de module de test*/

A.3.5.6 *Lignes de comportement*

270 BehaviourLine ::= **$BehaviourLine** LabelId Line Cref VerdictId [Comment] **$End_BehaviourLine**

271 Line ::= **$Line** Indentation StatementLine

272 Indentation ::= "[" Number "]"

/*SÉMANTIQUE STATIQUE – Les déclarations du premier niveau de propositions (propositions conditionnelles) multiples dans une description comportementale auront la valeur d'indentation zéro*/

/*SÉMANTIQUE STATIQUE – Les déclarations des propositions SI imbriquées auront comme valeur d'indentation celle de la proposition SI du niveau précédent plus un*/

273 LabelId ::= **$LabelId** [Label]

274 Label ::= Identifier

275 Cref ::= **$Cref** [ConstraintReference]

276 ConstraintReference ::= ConsRef | FormalParIdentifier

/*SÉMANTIQUE STATIQUE – La présence de l'attribut ConsRef sera concomitante des déclarations SEND (envoi), IMPLICIT SEND (envoi implicite) et RECEIVE (réception). L'attribut ConstraintReference (référence à une contrainte) n'est pas nécessaire aux primitives ASP dépourvues de paramètre. Il n'apparaîtra dans aucun autre type de déclaration TTCN*/

/*SÉMANTIQUE STATIQUE – L'identificateur FormalParIdentifier renverra un attribut ConsRef*/

/*SÉMANTIQUE STATIQUE – Les littéraux LiteralValue, les identificateurs TS_ParIdentifier, TS_ConstIdentifier, TS_VarIdentifier, TC_VarIdentifier, ConsRef et FormalParIdentifier pourront être transférés comme paramètres effectifs à une contrainte dans une déclaration ConstraintReference faisant partie d'une description de comportement*/

/*SÉMANTIQUE STATIQUE – L'attribut ConstraintReference donné en référence d'un événement SEND (envoi) ne contiendra pas de caractères génériques sauf si ceux-ci sont des valeurs spécifiques explicitement affectées dans la ligne d'événement SEND (envoi)*/

277 ConsRef ::= ConstraintIdentifier [ActualCrefParList]

278 ActualCrefParList ::= "(" ActualCrefPar {Comma ActualCrefPar} ")"

/*SÉMANTIQUE STATIQUE – Voir la sémantique statique relative à la production 299*/

279 ActualCrefPar ::= Value | DataObjectIdentifier | ConsRef

280 VerdictId ::= **$VerdictId** [Verdict]

281 Verdict ::= Pass | Fail | Inconclusive | Result

/*SÉMANTIQUE STATIQUE – L'attribut Verdict n'apparaîtra pas dans les entrées des types suivants de l'arbre de comportement: entrées vides, constructions ATTACH, constructions REPEAT, constructions GOTO, déclarations IMPLICIT SEND*/

282 Pass ::= **PASS** | **P** | "(" **PASS** ")" | "(" **P** ")"

283 Fail ::= **FAIL** | **F** | "(" **FAIL** ")" | "(" **F** ")"

284 Inconclusive ::= **INCONC** | **I** | "(" **INCONC** ")" | "(" **I** ")"

285 Result ::= Identifier

/*SÉMANTIQUE STATIQUE – L'attribut Result (résultat) aura pour valeur exclusive l'identificateur prédéfini R*/

/*SÉMANTIQUE STATIQUE – L'identificateur R ne sera jamais utilisé dans le membre gauche d'une affectation*/

A.3.5.7 *Déclarations en notation TTCN*

286 StatementLine ::= (Event [Qualifier] [AssignmentList] [TimerOps]) | (Qualifier [AssignmentList] [TimerOps]) | (AssignmentList [TimerOps]) | TimerOps | Construct | ImplicitSend

287 Event ::= Send | Receive | Otherwise | Timeout

/*SÉMANTIQUE STATIQUE – Un événement receive (réception), otherwise (sinon) ou timeout (fin de temporisation) ne peut être suivi que par d'autres événements receive, otherwise et timeout dans l'ensemble des autres propositions SI de l'arbre complètement développé. En conséquence, les arbres par défaut ne comporteront que des événements receive, otherwise et timeout au premier niveau d'indentation des propositions SI*/

288 Qualifier ::= "[" Expression "]"

/*SÉMANTIQUE STATIQUE – L'attribut Qualifier (qualificateur) aura une valeur BOOLEAN*/

289 Send ::= [PCO_Identifier | FormalParIdentifier] "|" (ASP_Identifier | PDU_ Identifier)

/*SÉMANTIQUE STATIQUE – Les identificateurs PCO_Identifier et FormalParIdentifier ne seront présents que si la suite de tests utilise plus d'un point PCO*/

/*SÉMANTIQUE STATIQUE – L'identificateur FormalParIdentifier renverra un identificateur PCO_Identifier. Celui-ci doit être présent si la suite de tests utilise plus d'un point PCO*/

290 ImplicitSend ::= "<" **IUT** "|" (ASP_Identifier | PDU_Identifier) ">"

/*SÉMANTIQUE STATIQUE – L'attribut ImplicitSend ne sera utilisé que pour les méthodes de test à distance*/

291 Receive ::= [PCO_Identifier | FormalParIdentifier] "?" (ASP_Identifier | PDU_Identifier)

/*SÉMANTIQUE STATIQUE – Les identificateurs PCO_Identifier et FormalParIdentifier ne seront présents que si la suite de tests utilise plus d'un point PCO*/

292 Otherwise ::= [PCO_Identifier | FormalParIdentifier] "?" **OTHERWISE**

/*SÉMANTIQUE STATIQUE – Les identificateurs PCO_Identifier et FormalParIdentifier ne seront présents que si la suite de tests utilise plus d'un point PCO*/

293 Timeout ::= "?" **TIMEOUT** [TimerIdentifier]

294 Construct ::= GoTo | Attach | Repeat

295 GoTo ::= ("–>" | **GOTO**) Label

/*SÉMANTIQUE STATIQUE – La colonne des étiquettes ne doit contenir que des étiquettes mentionnées dans des déclarations GoTo*/

/*SÉMANTIQUE STATIQUE – L'étiquette Label sera associée à la première proposition SI d'un ensemble de propositions conditionnelles multiples dont l'une constitue un nœud antécédent du point à partir duquel le renvoi GoTo intervient*/

/*SÉMANTIQUE STATIQUE – Le renvoi GoTo ne sera utilisé qu'à l'intérieur d'un même arbre, c'est-à-dire dans un arbre racine de test élémentaire, dans un arbre de module de test, dans un arbre par défaut ou dans un arbre local*/

/*SÉMANTIQUE STATIQUE – Chaque étiquette utilisée dans une construction GoTo figurera à l'intérieur du module auquel appartient cette construction*/

/*SÉMANTIQUE STATIQUE – Aucune déclaration GoTo ne renverra au premier niveau de proposition dans les arbres locaux, les modules de test et les arbres par défaut*/

296 Attach ::= "+" TreeReference [ActualParList]

/*SÉMANTIQUE STATIQUE – Une référence TreeReference ne se rattachera jamais à elle-même, ni directement ni indirectement, à son niveau supérieur d'indentation*/

/*SÉMANTIQUE STATIQUE – Le nombre de paramètres effectifs sera le même que le nombre de paramètres formels*/

/*SÉMANTIQUE STATIQUE – Les littéraux LiteralValue, les identificateurs TS_ParIdentifier, TS_ConstIdentifier, TS_VarIdentifier, TC_VarIdentifier, ConstraintIdentifier et les points PCO peuvent être transférés en tant que paramètres effectifs vers un arbre rattaché*/

297 Repeat ::= **REPEAT** TreeReference [ActualParList] **UNTIL** Qualifier

/*SÉMANTIQUE STATIQUE – Une référence TreeReference ne se rattachera jamais à elle-même, ni directement ni indirectement, à son niveau supérieur d'indentation*/

/*SÉMANTIQUE STATIQUE – Le nombre de paramètres effectifs sera le même que le nombre de paramètres formels*/

/*SÉMANTIQUE STATIQUE – Les littéraux LiteralValue, les identificateurs TS_ParIdentifier, TS_ConstIdentifier, TS_VarIdentifier, TC_VarIdentifier, ConstraintIdentifier et les points PCO peuvent être transférés vers l'arbre en tant que paramètres effectifs dans une déclaration REPEAT*/

298 TreeReference ::= TestStepIdentifier | TreeIdentifier

/*SÉMANTIQUE STATIQUE – L'identificateur d'arbre (TreeIdentifier) désignera un des arbres de la description comportementale courante, c'est-à-dire que les arbres locaux ne seront pas accessibles depuis l'extérieur de la description comportementale dans laquelle ils sont spécifiés*/

299 ActualParList ::= "(" ActualPar {Comma ActualPar} ")"

/*SÉMANTIQUE STATIQUE – Le nombre de paramètres effectifs sera le même que le nombre de paramètres formels*/

/*SÉMANTIQUE STATIQUE – Chaque paramètre effectif prendra une valeur compatible avec le type du paramètre formel correspondant*/

/*SÉMANTIQUE STATIQUE – Si un paramètre est une contrainte paramétrée, cette contrainte sera transférée avec sa liste de paramètres effectifs*/

/*SÉMANTIQUE STATIQUE – Les paramètres effectifs seront valués*/

/*SÉMANTIQUE STATIQUE – Si le type de paramètre formel est PDU, le paramètre effectif sera alors déclaré de type PDU ou de type **PDU** spécifique*/

300 ActualPar ::= Value | PCO_Identifier

A.3.5.8 *Expressions*

301 AssignmentList ::= "(" Assignment {Comma Assignment} ")"

302 Assignment ::= DataObjectReference ":=" Expression

/*SÉMANTIQUE STATIQUE – Le membre gauche de l'affectation renverra toujours à un identificateur TS_VarIdentifier, à un identificateur TC_VarIdentifier, à une référence à un champ de variable ou à une référence à un paramètre de primitive ASP ou à un champ d'unité PDU à transmettre*/

/*SÉMANTIQUE STATIQUE – Une expression ne contiendra pas de variables non valuées*/

/*SÉMANTIQUE STATIQUE – L'expression du membre droit de l'affectation prendra une valeur explicite du même type que le membre gauche*/

303 Expression ::= SimpleExpression [RelOp SimpleExpression]

/*SÉMANTIQUE STATIQUE – S'il existe deux expressions simples (SimpleExpressions) liées par un opérateur relationnel RelOp, ces expressions prendront des valeurs spécifiques de types compatibles*/

/*SÉMANTIQUE STATIQUE – Si l'opérateur RelOp est "<" | ">" | ">=" | "<=" chaque expression simple prendra une valeur spécifique de type INTEGER (entière)*/

/*SÉMANTIQUE STATIQUE – Les valeurs nommées de l'ASN.1 ne seront pas utilisées comme opérandes dans les expressions arithmétiques*/

304 SimpleExpression ::= Term {AddOp Term}

/*SÉMANTIQUE STATIQUE – Chaque terme prendra une valeur spécifique. Si plus d'un terme existe et si l'opérateur additif AddOp est «OR» (ou), les termes seront du type BOOLEAN; si l'opérateur AddOp est "+" ou "–", les termes seront du type INTEGER*/

305 Term ::= Factor {MultiplyOp Factor}

/*SÉMANTIQUE STATIQUE – Chaque facteur prendra une valeur spécifique. Si plus d'un facteur existe et si l'opérateur multiplicatif MultiplyOp est «AND» (et), les facteurs seront du type BOOLEAN; si l'opérateur MultiplyOp est "*" ou "/" les facteurs seront du type INTEGER*/

306 Factor ::= [UnaryOp] Primary

/*SÉMANTIQUE STATIQUE – L'élément Primary (primaire) prendra une valeur spécifique. Si l'opérateur unaire UnaryOp existe et est «NOT» (non), l'élément primaire sera du type BOOLEAN; si l'opérateur UnaryOp est "+" ou "–", l'élément primaire sera du type INTEGER*/

307 Primary ::= Value | DataObjectReference | OpCall | SelectExprIdentifier | "(" Expression ")"

/*SÉMANTIQUE STATIQUE – L'identificateur d'expression de sélection (SelectExprIdentifier) ne sera utilisé que dans des expressions de sélection*/

308 DataObjectReference ::= DataObjectIdentifier {ComponentReference}

/*SÉMANTIQUE STATIQUE – Les identificateurs de paramètres de primitives ASP et de champs d'unités PDU associés aux événements SEND et RECEIVE ne seront utilisés que pour renvoyer aux valeurs de ces paramètres et champs sur la ligne même de déclaration*/

/*SÉMANTIQUE STATIQUE – Chacune des références ComponentReference désignera un paramètre de primitive ASP, un champ d'unité PDU, un élément de structure ou une valeur ASN.1 déclarée explicitement dans l'objet qui la précède immédiatement du membre droit de l'affectation*/

309 DataObjectIdentifier ::= TS_ParIdentifier | TS_ConstIdentifier | TS_VarIdentifier | TC_VarIdentifier | FormalParIdentifier | ASP_Identifier | PDU_Identifier

310 ComponentReference ::= RecordRef | ArrayRef | BitRef

/*SÉMANTIQUE STATIQUE – La référence d'enregistrement RecordRef sera utilisée pour désigner les composantes des types SEQUENCE (séquence), SET (ensemble) et CHOICE (choix) à l'exclusion de tout autre type ASN.1*/

/*SÉMANTIQUE STATIQUE – RecordRef sera utilisée pour faire référence aux paramètres de primitives ASP, aux champs d'unités PDU et aux éléments de structure sous forme tabulaire*/

/*SÉMANTIQUE STATIQUE – La référence de tableau ArrayRef sera utilisée pour désigner les composantes des types SEQUENCE OF (séquence de) et SET OF (ensemble de) à l'exclusion de tout autre type ASN.1*/

311 RecordRef ::= Dot (ComponentIdentifier | PDU_Identifier | StructIdentifier | ComponentPosition)

/*SÉMANTIQUE STATIQUE – La forme RecordRef avec utilisation de l'identificateur ComponentIdentifier sera toujours utilisée pour faire référence aux composantes des types SEQUENCE, SET et CHOICE de l'ASN.1 quand un identificateur est déclaré pour cette composante*/

/*SÉMANTIQUE STATIQUE – La forme du type de RecordRef avec utilisation de l'identificateur ComponentIdentifier sera toujours utilisée pour faire référence aux paramètres de primitives ASP, aux champs d'unités PDU et aux éléments de structure déclarés sous forme tabulaire*/

/*SÉMANTIQUE STATIQUE – La forme de RecordRef avec utilisation de l'attribut de position de composante ComponentPosition sera toujours utilisée pour faire référence aux composantes des types SEQUENCE, SET et CHOICE de l'ASN.1 quand un identificateur n'est pas déclaré pour cette composante*/

/*SÉMANTIQUE STATIQUE – L'identificateur de structure (StructIdentifier) ne sera pas utilisé si la structure pertinente est utilisée comme macro. L'identificateur StructIdentifier et l'identificateur PDU PDU_Identifier seront explicitement compris dans RecordRef chaque fois qu'un paramètre, un champ ou un élément concaténé à une unité PDU ou à une structure et que RecordRef devra identifier une composante de cette unité PDU ou de cette structure*/

/*SÉMANTIQUE STATIQUE – Lorsqu'une structure est utilisée comme développement de macro, il sera fait référence aux éléments de la structure comme si cette dernière avait été développée dans la primitive ASP ou l'unité PDU qui s'y réfère*/

/*SÉMANTIQUE STATIQUE – Si un paramètre, champ ou élément est défini en tant que métatype PDU, aucune référence aux champs de cette sous-structure ne sera faite*/

312 ComponentIdentifier ::= ASP_ParIdentifier | PDU_FieldIdentifier | ElemIdentifier | ASN1_Identifier

313 ASN1_Identifier ::= Identifier

/**Remarque* – L'identificateur ASN1_Identifier identifie un champ à l'intérieur d'un type SEQUENCE, SET ou CHOICE de l'ASN.1*/

/*SÉMANTIQUE STATIQUE – Un identificateur ASN1_Identifier associé à une valeur nommée NamedValue ne sera pas utilisé à moins que cette valeur n'apparaisse dans un type SEQUENCE, SET ou CHOICE*/

/*SÉMANTIQUE STATIQUE – Un identificateur ASN1_Identifier sera prévu pour identifier la variante dans un type CHOICE*/

/*SÉMANTIQUE STATIQUE – Un identificateur ASN1_Identifier sera prévu chaque fois que la définition de valeur devient ambiguë parce que des valeurs OPTIONAL sont omises dans un type SEQUENCE*/

314 ComponentPosition ::= "(" Number ")"

315 ArrayRef ::= Dot "[" ComponentNumber "]"

316 ComponentNumber ::= Expression

/*SÉMANTIQUE STATIQUE – Le numéro de composant ComponentNumber prendra une valeur spécifique de type INTEGER (entière) non négative*/

317 BitRef ::= Dot (BitIdentifier | "[" BitNumber "]")

318 BitIdentifier ::= Identifier

/**Remarque* – L'identificateur de bits BitIdentifier identifie un bit particulier dans une chaîne BIT STRING de l'ASN.1*/

319 BitNumber ::= Expression

/*SÉMANTIQUE STATIQUE – Le numéro de bit BitNumber prendra une valeur spécifique de type INTEGER (entière) non négative*/

320 OpCall ::= TS_OpIdentifier (ActualParList | "(" ")")

/*SÉMANTIQUE STATIQUE – Voir la sémantique statique relative à la production 299*/

321 AddOp ::= "+" | "–" | **OR**

/*SÉMANTIQUE STATIQUE – Les opérandes des opérateurs "+" et "–" seront de type INTEGER (c'est-à-dire prédéfinis en notation TTCN ou ASN.1) ou dérivés de type INTEGER (des sous-intervalles). Les opérandes de l'opérateur OR seront du type BOOLEAN (prédéfinis en notation TTCN ou ASN.1) ou dérivés du type BOOLEAN*/

322 MultiplyOp ::= "*" | "/" | **MOD** | **AND**

/*SÉMANTIQUE STATIQUE – Les opérandes des opérateurs "*", "/" et MOD (Modulo) seront du type INTEGER (c'est-à-dire prédéfinis en notation TTCN ou ASN.1) ou dérivés du type INTEGER (des sous-intervalles). Les opérandes de l'opérateur AND seront du type BOOLEAN (prédéfinis en notation TTCN ou ASN.1) ou dérivés du type BOOLEAN*/

323 UnaryOp ::= "+" | "–" | **NOT**

/*SÉMANTIQUE STATIQUE – Les opérandes des opérateurs "+", "–" seront du type INTEGER (c'est-à-dire prédéfinis en notation TTCN ou ASN.1) ou des dérivés du type INTEGER (des sous-intervalles). Les opérandes de l'opérateur NOT seront du type BOOLEAN (prédéfinis en notation TTCN ou ASN.1) ou dérivés du type BOOLEAN*/

324 RelOp ::= "=" | "<" | ">" | "<>" | ">=" | "<="

A.3.5.9 *Opérations de temporisation*

325 TimerOps ::= TimerOp {Comma TimerOp}

326 TimerOp ::= StartTimer I CancelTimer I ReadTimer

327 StartTimer ::= **START** TimerIdentifier [ "("TimerValue")" ]

328 CancelTimer ::= **CANCEL** [TimerIdentifier]

329 TimerValue ::= Expression

/*SÉMANTIQUE STATIQUE – TimerValue prendra une valeur de type INTEGER strictement positive*/

330 ReadTimer ::= **READTIMER** TimerIdentifier "(" DataObjectReference ")"

/*SÉMANTIQUE STATIQUE – La référence DataObjectReference renverra toujours à un identificateur TS_VarIdentifier, à un identificateur TC_VarIdentifier, à une référence au champ de variable ou à une référence à un paramètre de primitive ASP ou à un champ d'unité PDU à envoyer*/

/*SÉMANTIQUE STATIQUE – La référence DataObjectReference prendra une valeur de type INTEGER*/

A.3.6 *Types*

A.3.6.1 *Considérations générales*

331 Type ::= PredefinedType | ReferenceType

A.3.6.2 *Types prédéfinis*

332 PredefinedType ::= **INTEGER I BOOLEAN I BITSTRING I HEXSTRING I OCTETSTRING I** CharacterString

333 CharacterString ::= **NumericString I PrintableString I TeletexString I VideotexString I VisibleString I IA5String I GraphicString I GeneralString**

A.3.6.3 *Types de référenciation*

334 ReferenceType ::= TS_TypeIdentifier | ASP_Identifier | PDU_Identifier

/*SÉMANTIQUE STATIQUE – Exception faite des types prédéfinis, tous les types utilisés dans une suite de tests seront déclarés dans les définitions de type de suites de tests ou dans les définitions de type de primitives ASP ou dans les définitions de type d'unités PDU et recevront un nom pour y faire référence*/

335 TS_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1_TypeIdentifier

A.3.7 *Valeurs*

336 Value ::= LiteralValue | ASN1_Value

/*RÉFÉRENCE – Où ASN1_Value est un type valeur, comme défini dans la Recommandation X.208*/

/*Dans la Recommandation X.208, DefinedValue (valeur définie) est produite comme suit: DefinedValue ::= Externalvaluereference | valuereference. Pour les besoins de la notation TTCN, cette production est redéfinie de la manière suivante: DefinedValue ::= ConstraintValue&Attributes. On notera qu'en d'autres termes, les références externes ne sont pas admises en notation TTCN*/

/*SÉMANTIQUE STATIQUE – Les valeurs nommées de l'ASN.1 ne seront pas utilisées comme opérandes dans les expressions arithmétiques*/

337 LiteralValue ::= Number I BooleanValue I Bstring I Hstring I Ostring I Cstring

338 Number ::= (NonZeroNum {Num}) | **0**

339 NonZeroNum ::= **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

340 Num ::= **0** | NonZeroNum

341 BooleanValue ::= **TRUE** | **FALSE**

342 Bstring ::= "'" {Bin I Wildcard} "'" **B**

343 Bin ::= **0** | **1**

344 Hstring ::= "'" {Hex I Wildcard} "'" **H**

345 Hex ::= Num | **A** | **B** | **C** | **D** | **E** | **F**

346 Ostring ::= "'" {Oct I Wildcard} "'" **O**

347 Oct ::= Hex Hex

348 Cstring ::= """ {Char I Wildcard I "\"} """

349 Char ::= /* RÉFÉRENCE – Un caractère défini par le type chaîne de caractères correspondant*/

/*SÉMANTIQUE STATIQUE – Si le type CharacterString comporte un guillemet ", ce caractère sera dédoublé dans la dénomination d'une valeur quelconque*/

350 Wildcard ::= AnyOne | AnyOrNone

351 AnyOne ::= "?"

/*SÉMANTIQUE STATIQUE – Le caractère générique "?" (AnyOne = un caractère quelconque) ne sera utilisé que dans des valeurs de type chaîne, SEQUENCE OF et SET OF*/

352 AnyOrNone ::= "*"

/*SÉMANTIQUE STATIQUE – Le caractère générique "*" (AnyOrNone = zéro ou un caractère quelconque) ne sera utilisé que dans les valeurs de type chaîne, SEQUENCE OF et SET OF*/

353 Identifier ::= Alpha{AlphaNum | Underscore}

/*SÉMANTIQUE STATIQUE – Tous les identificateurs auxquels il est fait référence dans une suite de tests en notation TTCN seront soit déclarés explicitement dans la suite des tests, soit encore déclarés explicitement dans une définition de type en notation ASN.1 citée elle-même en référence par la suite de tests, ou seront des identificateurs prédéfinis TTCN*/

354 Alpha ::= UpperAlpha I LowerAlpha

355 AlphaNum ::= Alpha I Num

356 UpperAlpha ::= **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** | **L** | **M** | **N** | **O** | **P** | **Q** | **R** | **S** | **T** | **U** | **V** | **W** | **X** | **Y** | **Z**

357 LowerAlpha ::= **a** | **b** | **c** | **d** | **e** | **f** | **g** | **h** | **i** | **j** | **k** | **l** | **m** | **n** | **o** | **p** | **q** | **r** | **s** | **t** | **u** | **v** | **w** | **x** | **y** | **z**

358 ExtendedAlphaNum ::= /* RÉFÉRENCE – Un caractère provenant de l'un quelconque des jeux de caractères définis dans l'ISO/CEI 10646 */

359 BoundedFreeText ::= "/*" FreeText "*/"

360 FreeText ::= {ExtendedAlphaNum}

/*SÉMANTIQUE STATIQUE – Une chaîne FreeText (texte libre) ne contiendra pas le groupement "*/" à moins de faire précéder un tel groupement par une barre oblique inversée ("\")*/

A.3.8 *Productions diverses*

/**Remarque* – Les productions suivantes n'étant d'aucune aide à un locuteur français n'ont pas été utilisées dans la version française*/

```
361  Comma ::= ","
362  Dot ::= "."
363  Dash ::= "–"
364  Minus ::= "–"
365  SemiColon ::= ";"
366  Colon ::= ":"
367  Underscore ::= "_"
```

A.4 *Spécifications générales de la sémantique statique*

A.4.1 *Introduction*

Les spécifications de la sémantique statique liées à des productions spécifiques de la forme BNF sont indiquées sous forme de commentaires à la suite des productions correspondantes, dans le format suivant:

/*SÉMANTIQUE STATIQUE – ...*/

Toutes les autres spécifications de la sémantique statique commune aux notations graphique TTCN.GR et informatisable TTCN.MP sont spécifiées dans le reste du § A.4. La sémantique statique additionnelle appliquée en notation TTCN.MP est spécifiée au § A.5.2.

A.4.2 *Unicité des identificateurs*

A.4.2.1 Dans certains cas, les suites de tests renvoient à des éléments définis dans d'autres Recommandations OSI. Les définitions de types peuvent notamment faire référence à des modules de définition de type ASN.1 selon la Recommandation X.208. Les noms provenant de ces modules (tels que les identificateurs de sous-champs à l'intérieur des définitions de types structurés ASN.1) peuvent être utilisés dans toute la suite de tests.

Etant donné que les règles concernant les identificateurs en ASN.1 et en notation TTCN ne sont pas totalement compatibles, les conventions suivantes s'appliquent:

a) les références aux types et les identificateurs de modules qui figurent dans les différentes tables de définitions de types ASN.1 respecteront les conventions définies dans la Recommandation X.208 relatives aux identificateurs;

b) dans les identificateurs utilisés dans les autres parties d'une suite de tests, les tirets (–) seront remplacés par des soulignés ( _ ).

Dans certaines tables TTCN, une partie de la syntaxe ASN.1 peut être utilisée pour la définition de types. Dans ce cas, les identificateurs se conformeront aux règles de l'ASN.1 sauf que les tirets (–) ne seront pas utilisés. Les soulignés (_) peuvent être utilisés à leur place. Toutes les autres spécifications établies dans la Recommandation X.208 (par exemple, les identificateurs de type commenceront par une majuscule et les identificateurs de champs dans les définitions structurées de l'ASN.1 commenceront par une minuscule) s'appliquent aux suites de tests TTCN chaque fois que la notation ASN.1 est utilisée.

A.4.2.2 Tous les identificateurs des objets TTCN suivants seront uniques dans toute la suite de tests:

a) types de suite de tests;

b) opérations de suite de tests;

c) paramètres de suite de tests;

d) expressions de sélection de test élémentaire;

e) constantes de suite de tests;

f) variables de suite de tests;

g) variables de test élémentaire;

h) types de point PCO;

i) points PCO;

j) temporisations;

k) types de primitive ASP;

l) types d'unité PDU;

m) types structurés;

n) alias (pseudonymes);

o) contraintes de primitive ASP;

p) contraintes d'unité PDU;

q) contraintes de structure;

r) tests élémentaires;

s) modules de tests;

t) arbres par défaut.

A.4.2.3 Toutes les références aux objets TTCN suivants seront uniques dans toute la suite de tests:

a) références de groupe de tests;

b) références de groupe de modules de test;

c) références de groupe de comportements par défaut.

A.4.2.4 Les tableaux A-2/X.292 et A-3/X.292 énumèrent les mots clés réservés de la notation TTCN. Ces mots clés réservés ne doivent pas être utilisés comme identificateurs dans une suite de tests TTCN. Pour tous les mots clés de la notation TTCN, les identificateurs prédéfinis (types, opérations, variables, valeurs prédéfinis) et les identificateurs TTCN, la différence majuscule/minuscule est significative.

TABLEAU A-2/X.292

**Mots clés de la notation TTCN**

| | | |
|---|---|---|
| AND | ms | REPLACE |
| BY | NOT | s |
| CANCEL | ns | START |
| F | OMIT | SUPERSET |
| FAIL | OR | SUBSET |
| GOTO | OTHERWISE | TIMEOUT |
| I | P | TO |
| IF_PRESENT | PASS | UNTIL |
| INCONC | PDU | µs |
| IUT | PERMUTATION | UT |
| LT | ps | |
| min | READTIMER | |
| MOD | REPEAT | |

TABLEAU A-3/X.292

**Identificateurs prédéfinis de la notation TTCN**

| | | |
|---|---|---|
| BITSTRING | inconc | NumericString |
| BOOLEAN | INFINITY | OCTETSTRING |
| BIT_TO_INT | INTEGER | pass |
| fail | INT_TO_HEX | PrintableString |
| FALSE | INT_TO_BIT | R |
| GeneralString | IS_CHOSEN | TRUE |
| GraphicString | IS_PRESENT | TeletexString |
| HEXSTRING | LENGTH_OF | VideotexString |
| HEX_TO_INT | none | VisibleString |
| IA5String | NUMBER_OF_ELEMENTS | |

A.4.2.5 Le tableau A-4/X.292 énumère les mots réservés de la notation ASN.1. Ces mots réservés ne seront pas utilisés comme identificateurs dans une suite de tests TTCN.

TABLEAU A-4/X.292

**Mots réservés de la notation ASN.1**

| | | |
|---|---|---|
| ABSENT | FROM | PRESENT |
| ANY | GeneralStringGeneralizedTime | PRIVATE |
| APPLICATION | GraphicString | PrintableString |
| BEGIN | IA5String | REAL |
| BIT | IDENTIFIER | SEQUENCE |
| BOOLEAN | IMPLICIT | SET |
| CHOICE | IMPORT | SIZE |
| COMPONENT | INCLUDES | STRING |
| COMPONENTS | INTEGER | T61String |
| DEFAULT | ISO646String | TRUE |
| DEFINED | MAX | TeletexString |
| DEFINITIONS | MIN | UNIVERSAL |
| END | NULL | UTCTime |
| ENUMERATED | NumericString | VideotexString |
| EXPLICIT | OBJECT | VisibleString |
| EXPORT | OCTET | WITH |
| EXTERNAL | OF | |
| FALSE | OPTIONAL | |

A.4.2.6 Quand la notation ASN.1 est utilisée dans une suite de tests TTCN, les identificateurs ASN.1 de la liste suivante seront uniques dans toute la suite de tests, que la définition ASN.1 soit explicite ou implicite par référence:

a) identificateurs de type (*TypeIdentifiers*) d'une définition de type de l'ASN.1;

b) identificateurs apparaissant dans un type ENUMERATED (énuméré) de l'ASN.1 en tant que valeurs distinctives;

c) identificateurs apparaissant dans une liste de nombres-nommés, (*NamedNumberList*) d'un type INTEGER de l'ASN.1.

A.4.2.7 Les noms de paramètres de primitives ASP seront uniques dans la primitive ASP où ils sont déclarés. Les noms de champs d'unités PDU seront uniques dans l'unité PDU où ils sont déclarés.

A.4.2.8 Si un type structuré est utilisé comme développement de macro, les noms des éléments contenus dans le type structuré seront uniques dans chaque primitive ASP ou unité PDU où cette structure sera développée.

A.4.2.9 Les étiquettes utilisées dans un arbre seront uniques dans cet arbre (par exemple, arbre racine de test élémentaire, arbre de module de test, arbre par défaut, arbre local).

A.4.2.10 L'identificateur d'en-tête d'arbre utilisé pour les arbres locaux sera unique dans la description de comportement dynamique où ces arbres apparaissent et sera différent de tout identificateur ayant une signification unique dans toute la suite de tests.

*Remarque* – Cela signifie qu'un identificateur d'arbre local peut avoir le même nom qu'un identificateur d'arbre local dans une autre description comportementale, mais pas le même qu'un autre module de test dans la bibliothèque des modules de test.

A.4.2.11 Les noms de paramètres formels qui peuvent apparaître en option comme faisant partie des éléments suivants seront uniques dans cette liste de paramètres formels et ne coïncideront avec aucun autre identificateur ayant une signification unique dans toute la suite de tests:

a) définition des opérations de suite de tests;

b) en-tête d'un arbre local;

c) identificateur de module de test;

d) identificateur d'arbre par défaut;

e) déclaration de contrainte paramétrée.

A.4.2.12 Un nom de paramètre formel figurant dans la liste des paramètres formels d'un en-tête d'arbre local supplantera, dans le champ de visibilité de cette liste locale, un nom identique de paramètre formel figurant dans la liste des paramètres formels du module de test où l'arbre local est défini.

## A.5 *Différences entre la notation TTCN.GR et la notation TTCN.MP*

### A.5.1 *Différences syntaxiques*

Les différences syntaxiques entre notations informatisable TTCN.MP et graphique TTCN.GR sont énumérées ci-dessous:

a) la notation TTCN.MP utilise des mots clés comme délimiteurs entre les entrées, tandis que la notation TTCN.GR utilise des encadrés;

b) la notation TTCN.MP utilise une dénotation numérique explicite des niveaux d'indentation des événements de test, tandis que cet ordre d'imbrication est inhérent à l'aspect graphique de la notation TTCN.GR;

c) la notation TTCN.MP comporte une occurrence supplémentaire de l'identificateur de suite, afin de faciliter l'identification des suites de tests abstraites (ATS) en méthode automatisée;

d) en notation TTCN.MP, les descriptions comportementales des tests élémentaires sont explicitement regroupées par l'insertion de l'identificateur du groupe de tests en tête des descriptions comportementales des tests élémentaires appartenant à ce groupe; cette information reproduit celle qui figure dans l'index des tests élémentaires et dans les références de groupe de tests des descriptions comportementales des tests élémentaires;

e) les tables de structure de suites de tests, d'indices des tests élémentaires, d'indices de modules de tests et d'indices par défaut exigent un numéro de page par entrée. Etant donné que de tels numéros n'ont pas de signification sémantique en traitement machine, ils ne sont pas pris en compte dans la notation TTCN.MP;

f) la notation TTCN.GR peut gérer les formulaires tant simples que compacts, utilisés pour les contraintes de primitives ASP et d'unités PDU et pour les tests élémentaires; pour le format tabulaire simple, la notation TTCN ne peut gérer que la forme BNF et la représentation d'un certain nombre de tables simples en format compact TTCN.GR est un problème de sortie; quand on établit la correspondance entre une table compacte de contraintes et la notation TTCN.MP (c'est-à-dire le format simple), les champs supprimés par les modifications seront omis;

g) les symboles "/*" et "*/" ouvrant et fermant les chaînes de texte libre borné BoundedFreeText en notation TTCN.MP n'apparaîtront pas en notation TTCN.GR;

h) il y a deux positions possibles pour la colonne étiquettes des tableaux de description comportementale en notation TTCN.GR et une seule pour les étiquettes en notation TTCN.MP;

i) les suites de pages et de lignes n'apparaissent qu'en notation TTCN.GR et n'existent pas en notation TTCN.MP;

j) la numérotation des pages et des lignes n'apparaît qu'en notation TTCN.GR et n'existe pas en notation TTCN.MP.

A.5.2 *Eléments additionnels de sémantique statique en notation TTCN.MP*

On trouvera ci-dessous une liste d'éléments additionnels de sémantique statique utilisés en notation TTCN.MP:

a) en notation TTCN.MP, les déclarations du premier niveau de propositions SI (propositions conditionnelles) n'ayant pas de prédécesseur dans l'arbre racine ou l'arbre local auquel elles appartiennent ont la valeur d'indentation zéro; les déclarations des propositions SI imbriquées ont une valeur d'indentation égale à celle de la proposition SI du niveau précédent plus un;

b) en notation TTCN.MP, l'information de structure de suite de tests, représentée sous la forme d'identificateurs de groupe de tests précédant les descriptions comportementales des tests élémentaires, sera la même que celle qui est définie par la partie de la structure de suite de tests correspondant aux groupes de tests et celle qui est définie par l'indice des tests élémentaires.

ANNEXE B

(à la Recommandation X.292)

**Sémantique opératoire de la notation TTCN**

(Cette annexe fait partie intégrante de la présente Recommandation)

## B.1 *Introduction*

La présente annexe décrit la manière dont sont définis les éléments sémantiques en notation TTCN. Leur définition comporte deux phases. Dans la première, des textes habituels en notation TTCN sont ramenés à une structure simplifiée manipulable en deuxième phase. La deuxième phase peut être considérée comme une machine TTCN qui interprète les textes TTCN simplifiés.

La première phase se décompose en trois étapes:

a) *Définition syntaxique*

Aucun contenu sémantique ne peut être affecté à un langage dépourvu d'une grammaire établie (indépendante du contexte). L'annexe A décrit la syntaxe TTCN par des règles de production en forme BNF.

b) *Définition sémantique statique*

La sémantique statique précise lesquels des textes générés conformément à la grammaire indépendante du contexte sont significatifs (par exemple, les points PCO utilisés doivent être déclarés, les contraintes auxquelles il est fait référence dans la partie comportementale doivent être présentes dans la partie contraintes de la suite de tests, etc.); les spécifications de la sémantique statique de la notation TTCN sont données en annexe A.

c) *Définition de la transformation d'arbre*

La troisième étape décrit la construction d'un arbre abstrait d'évaluation à partir d'un texte TTCN réel correct du point de vue de la syntaxe et de la sémantique statique. Les algorithmes transformationnels sont décrits au § B.4.

La deuxième phase définit la machine TTCN abstraite. Il est peu vraisemblable que cette machine, munie de l'architecture décrite, puisse être un jour construite. Les problèmes de réalisation ne sont pas pris en compte dans la définition de la sémantique dynamique, ou opératoire. La machine TTCN abstraite est décrite au § B.5. Sa partie principale est un processus appelé EVALUATE_TEST_CASE (évaluation de test élémentaire) qui interprète l'arbre abstrait d'évaluation. Cet arbre est un paramètre du processus considéré.

## B.2 *Priorité*

La sémantique opératoire appliquée à la notation TTCN est présentée dans ce qui suit sous deux formes possibles de notation pour permettre au lecteur de choisir la méthode qui lui convient le mieux: le langage pseudo-formalisé et le langage naturel. La convergence de ces deux notations n'est pas fortuite puisqu'elle représente un contenu identique. Par contre, si le langage pseudo-formalisé et le langage naturel se contredisent, il s'agit d'une erreur qui doit être portée à la connaissance de l'organisme de normalisation sous forme de rapport d'anomalie. En ce cas, le langage pseudo-formalisé prendra le pas sur le langage naturel jusqu'à ce que des corrections soient apportées par l'organisme de normalisation.

## B.3 *Traitement des erreurs de test élémentaire*

Dans le corps de la présente Recommandation et dans la présente annexe, des conditions sont décrites qui conduisent à la détection d'erreurs de test élémentaire. Chaque fois qu'une telle erreur est décelée, elle sera consignée dans le journal de conformité.

## B.4 *Algorithmes de transformation*

### B.4.1 *Introduction*

Le présent paragraphe définit la manière de construire un arbre abstrait d'évaluation à partir d'un test élémentaire en notation TTCN correct du point de vue de la syntaxe et de la sémantique statique. L'opération se déroule en trois étapes successives:

a) adjonction des comportements par défaut;

b) suppression des structures REPEAT;

c) développement des arbres rattachés.

Ces transformations sont décrites sous forme d'algorithmes en pseudo-langage de programmation ou pseudo-code. En outre, une description en langage naturel est donnée afin d'expliquer le fonctionnement de ces algorithmes.

### B.4.2 *Adjonction de comportement par défaut*

L'adjonction effective de compléments par défaut est effectuée en ajoutant la construction «+ DefaultReference» à la fin de chaque ensemble de propositions SI dans le test élémentaire. Dans le pseudo-code suivant, le paramètre Level représente l'ensemble des propositions SI dans l'arbre de comportement courant. Si $A_i$ est une proposition SI parmi M dans l'arbre de comportement courant, Level est alors l'ensemble ordonné ($A_1$, $A_2$, . . . $A_m$) où $A_i$ est un événement, un pseudo-événement ou une construction. A chaque $A_i$ est associé un paramètre Level de niveau inférieur regroupant zéro ou plusieurs propositions SI. Chaque $A_i$ représente donc un sous-arbre de l'arbre initial.

- **function APPEND_DEFAULT** (TestCaseOrStep,Tree,Level)**: BOOLEAN**

```
begin
if (TestCaseOrStep.DefaultReference <> empty) then
      begin

       (*adjonction de comportements par défaut au premier niveau du premier arbre de test élémentaire*)


      if ((TestCaseOrStep is a TestCase)) and
         (Tree=ROOT_TREE(TestCaseOrStep) and
         (Level=FIRST_LEVEL(Tree))) then
         APPEND('+' TestCaseOrStep.DefaultReference, Level);

       (*adjonction de comportements par défaut à tous les niveaux en dessous du premier niveau*)

            if (Level <>  FIRST_LEVEL(Tree)) then
                  APPEND('+' TestCaseOrStep.DefaultReference, Level)
        end;
      RETURN(TRUE);
end
```

L'adjonction de tous les comportements par défaut dans un test élémentaire ou un module de test peut être effectuée en parcourant l'arbre de comportement tout entier et en utilisant la jonction APPEND_DEFAULT à chaque niveau d'imbrication de proposition SI. Cela est possible avec la procédure récursive suivante, appelée de la manière suivante: TestCaseOrStep:=TestCase, Tree:=ROOT_TREE(TestCase), and Level:=FIRST_LEVEL(Tree)

- **procedure APPEND_TRAV** (TestCaseOrStep, Tree, Level)

```
begin
if (Am in Level is not '+' TestCaseOrStep.DefaultReference) then
        begin (*N'exécuter que si les comportements par défaut ne sont pas encore annexés*)
                for (every alternative Ai in Level) do

                begin
                        if (Ai is not a leaf) then

                        begin
                                NextLevel:=NEXT_LEVEL(Ai);
                                APPEND_TRAV(TestCaseOrStep, Tree, NextLevel)

                        end
                        if (Ai is an ATTACH construct) then
                                if (Ai.AttachedTree is TestStepIdentifier) then

                        begin
                                NextTestStep:=Ai.AttachedTree;
                                NextTree:=ROOT_TREE(NextTestStep);
                                NextLevel:=FIRST_LEVEL(NextTree);
                                APPEND_TRAV(NextStep, NextTree, NextLevel);

                        end
                        else (*l'arbre rattaché est un identificateur TreeIdentifier*)

                        begin
                                NextTestTree:=Ai.AttachedTree;
                                NextLevel:=FIRST_LEVEL(NextTree);
                                APPEND_TRAV(TestCaseOrStep, NextTree, NextLevel)

                        end

                end
        APPEND_DEFAULT(TestCaseOrStep, NextTree, NextLevel)

    end
end
```

B.4.3 *Suppression des constructions REPEAT (répétition)*

Si l'objet *TreeAndParameters* (arbre et paramètres) représente un identificateur d'arbre donné *TreeIdentifier* suivi d'une liste de paramètres effectifs *ActualPARlist*, et si «condition» est une expression booléenne donnée, la construction REPEAT *TreeAndParameters* UNTIL [*Condition*] peut être alors remplacée par:

```
                [TRUE]
label_A             [TRUE]
                        + TreeAndParameters
                          [NOT (condition)]
                                –> label_A
                          [condition]
                                :
```

Les lignes décrivant la suite du comportement de la construction REPEAT viennent après ce développement, avec une indentation supplémentaire de deux niveaux.

B.4.4 *Développement des arbres ATTACHED (rattachés)*

Les arbres rattachés sont développés en remplaçant d'abord la construction de rattachement + *TestStep* par l'arbre *TestStep* puis si un comportement est spécifié avec indentation à la suite de la construction ATTACH, en insérant avec indentation ce comportement après chaque feuille de l'arbre rattaché.

Lors de l'évaluation de la sémantique d'un arbre comportemental après rattachement de comportements par défaut et suppression des constructions REPEAT, il est recommandé de développer les arbres rattachés pour chaque niveau d'indentation (ensemble de propositions SI) dès que ce niveau est atteint pendant l'exécution d'un test élémentaire. Les arbres rattachés pour le niveau Level sont développés en appliquant la procédure suivante:

- **procedure EXPAND_LEVEL** (Level)

```
begin
    for (every alternative Ai in Level) do
    begin
        if (Ai is an ATTACH construct) then
        begin
                Subsequent:=SUBSEQUENT_BEHAVIOUR_TO(Ai);
                EXPAND(Ai.AttachedTree, Subsequent, Ai, Level)
        end
    end
end
```

- **procedure EXPAND** (SubTree, Subsequent, Alternative, Level)

```
begin
REPLACE_PARAMETERS(Alternative, SubTree);
```

(*Cette instruction remplace par substitution textuelle les paramètres formels du (sous-arbre) par les paramètres formels spécifiés dans la liste des paramètres effectifs de la proposition SI*)

```
for (every leaf Li in SubTree) do APPEND_SUBSEQUENT_BEHAVIOUR_TO(Li, Subsequent);

New_Level:=FIRST_LEVEL(SubTree);
Level:=REPLACE(Level, New_Level, Alternative)
```

(*Cette instruction remplace la proposition SI (Alternative) du niveau d'indentation Level par le tableau New_Level; par exemple, REPLACE((A,B,C), (X,Y,Z), B) donne: (A,X,Y,Z,C)*)

```
end
```

Le développement des arbres rattachés est aussi expliqué au § 14.13.

B.5 *Sémantique opératoire de la notation TTCN*

B.5.1 *Introduction*

On suppose que tous les sous-arbres (rattachements directs et indirects) et les arbres REPEAT ont été préalablement développés conformément aux règles définies dans le corps de la présente Recommandation. On suppose aussi que tous les arbres comportementaux par défaut ont été rattachés conformément aux règles définies au § B.4.2.

On considère qu'un événement, un pseudo-événement ou une structure en notation TTCN *concorde* quand on peut l'évaluer correctement. Les conditions de concordance d'une déclaration TTCN dépendent de ce qui est codé sur la ligne comportementale et de ce qui est décrit dans le texte sémantique.

B.5.2 *Introduction à la notation pseudo-formalisée*

La sémantique de la notation TTCN est définie à l'aide d'une méthode fonctionnelle simple qui explique l'exécution d'un arbre comportemental TTCN et les composants qui forment les nœuds de cet arbre. Les fonctions utilisées ont pour but de mieux faire comprendre la sémantique de la notation TTCN et ne sont pas destinées à être associées à un quelconque modèle d'exécution ou langage de programmation de haut niveau. Elles ne sont pas censées être des méthodes directes destinées à exécuter des déclarations TTCN.

Ces fonctions supposent l'existence de deux variables (qui n'ont rien à voir avec les variables TTCN):

| | |
|---|---|
| *SendObject* (envoi d'objet) | est une structure de données globale temporaire dont la valeur est une primitive ASP ou une unité PDU à envoyer. Cette valeur est créée conformément aux spécifications de contrainte correspondantes. |
| *ReceivedObject* (réception d'objet) | est une structure de données globale temporaire dont la valeur est la copie d'une primitive ASP ou d'une unité PDU reçue. |

L'exécution d'un test élémentaire TTCN commence en appelant la fonction EVALUATE_TEST_CASE.

### B.5.3 *Exécution d'un test élémentaire*

#### B.5.3.1 *Exécution d'un test élémentaire: description en langage pseudo-formalisé*

- **function EVALUATE_TEST_CASE** (BehaviourTree)**: BOOLEAN**

  (*Le paramètre Level est une variable locale de la fonction EVALUATE_TEST_CASE. Si $A_i$ est une possibilité parmi *m* propositions conditionnelles multiples ou propositions SI de l'arbre comportemental courant, alors Level est l'ensemble ordonné ($A_1$, $A_2$ . . . $A_m$) où $A_i$ est un événement, un pseudo-événement ou une construction. L'arbre étant complètement développé, on notera que la seule déclaration TTCN qui puisse apparaître est GOTO*)

```
begin
Level:=FIRST_LEVEL;
EVALUATE_LEVEL (Level)

end
```

- **function EVALUATE_LEVEL** (Level): **BOOLEAN**

  (*Les propositions SI $A_1$ . . . $A_m$ du même niveau d'indentation Level sont traitées dans l'ordre de leur apparition. La sémantique opératoire TTCN suppose que le traitement de l'ensemble des propositions SI est instantané, c'est-à-dire qu'aucun état d'événement n'est modifié pendant le calcul de la concordance. On notera que Level est mis à jour et passe au niveau d'indentation suivant par une déclaration spécifique*)

```
begin
repeat
        TAKE_SNAPSHOT;
        if EVALUATE_EVENT_LINE(A1, Level) then EVALUATE_LEVEL(Level);
        if EVALUATE_EVENT_LINE(A2, Level) then EVALUATE_LEVEL(Level);
        if              ...
                        ...
                        ...
        if EVALUATE_EVENT_LINE(Am , Level) then EVALUATE_LEVEL(Level)
until SNAPSHOT_FIXED(Level)
RETURN(TestCaseError)
```

  (*Où SNAPSHOT_FIXED (image instantanée figée) renvoie la valeur TRUE (vrai) si toutes les files d'attente correspondantes des points PCO contiennent un ou plusieurs événements et si toutes les temporisations correspondantes sont arrivées à expiration, et renvoie sinon FALSE (faux)*)

```
end
```

- **function EVALUATE_EVENT_LINE** ($A_i$, Level)**: BOOLEAN**

  (*Cette fonction appelle les fonctions EVALUATE_EVENT, EVALUATE_PSEUDO_EVENT ou EVALUATE_CONSTRUCT suivant le type d'événement de la proposition SI ($A_i$)*)

```
case Ai of
EVENT:              if EVALUATE_EVENT (Ai, Level) then RETURN(TRUE) else
                    RETURN(FALSE);
PSEUDO_EVENT:       if EVALUATE_PSEUDO_EVENT (Ai, Level) then
                    RETURN(TRUE) else RETURN(FALSE);
TTCN_CONSTRUCT:     if EVALUATE_CONSTRUCT (Ai, Level) then
                    RETURN(TRUE) else RETURN(FALSE)

end
```

B.5.3.2 *Exécution d'un test élémentaire: description en langage naturel*

Etape 1 L'évaluation d'un test élémentaire commence au niveau d'indentation le plus à gauche de l'arbre (c'est-à-dire par les lignes non indentées dans l'arbre TTCN.GR).

Etape 2 On prend une image instantanée des files d'attente des points PCO et de la liste des fins de temporisations.

*Remarque* – Le fait de prendre une image instantanée n'extrait pas un événement de la file d'attente du point PCO. Dans la ligne comportementale au niveau courant d'indentation des propositions SI, prendre la première spécifiée.

Etape 3 Evaluer la déclaration TTCN de la ligne comportementale courante en fonction de ce qui est spécifié dans l'élément le plus à gauche, sauf pour les déclarations TTCN commençant par un identificateur de point PCO qui sont évaluées en fonction de ce qui suit cet identificateur.

L'évaluation de chaque type de déclaration TTCN est spécifiée dans la sémantique opératoire propre à ce type de déclaration. On considère que les affectations, les opérations de temporisation et les événements SEND génèrent une concordance positive sauf s'ils sont qualifiés par une expression booléenne qui prend la valeur False (faux). Les déclarations IMPLICIT SEND et GOTO sont toujours considérées comme des concordances positives. L'événement RECEIVE peut générer ou non une concordance positive, en fonction à la fois de l'expression booléenne si elle existe, et du premier événement dans la file d'attente correspondante du point PCO.

Etape 4 Si l'évaluation de la déclaration TTCN aboutit à une concordance positive, passer à l'étape 5.

Sinon, s'il reste d'autres propositions SI dans l'ensemble courant des propositions SI, passer à la ligne comportementale suivante de cet ensemble et aller à l'étape 3.

S'il n'existe pas d'autre proposition SI et que cependant toutes les files d'attente de point PCO se rapportant à ce niveau d'indentation contiennent au moins un événement et que tous les temporisateurs intervenant dans les déclarations de fin de temporisation de ce même niveau figurent dans la liste de fins de temporisation, il y a alors une erreur de test élémentaire et ce test sera interrompu en indiquant *Test case error* (erreur de test élémentaire).

*Remarque* – Il s'agit d'une erreur de test élémentaire parce que dans ces conditions aucune des propositions SI de l'ensemble ne pourra jamais concorder.

Dans tous les autres cas, prendre une nouvelle image instantanée des files d'attente de point PCO et de la liste de fins de temporisation et prendre à nouveau la première déclaration TTCN de l'ensemble des propositions SI; passer ensuite à l'étape 3.

Etape 5 Si un nœud feuille de l'arbre est atteint, passer à l'étape 6.

Sinon, prendre un nouvel ensemble de propositions SI pour évaluation. (Cet ensemble est composé des déclarations TTCN du niveau d'indentation immédiatement inférieur à la déclaration TTCN qui vient de produire une concordance positive.) Passer à l'étape 2.

Etape 6 Lorsqu'un nœud feuille de l'arbre est atteint, utiliser la valeur courante de la variable R du résultat préliminaire comme verdict final du test élémentaire, comme indiqué au § B.5.16.2.

B.5.4 *Fonctions appliquées aux événements TTCN*

B.5.4.1 *Fonctions appliquées aux événements TTCN: description en langage pseudo-formalisé*

- **function EVALUATE_EVENT** ($A_i$, Level)**: BOOLEAN**

  (*Cette fonction appelle les événements SEND, IMPLICIT SEND, RECEIVE, OTHERWISE ou TIMEOUT en fonction du type d'événement que constitue la proposition SI ($A_i$)*)

```
case Ai of
  SEND:          if  SEND (Ai, Level) then RETURN(TRUE) else RETURN(FALSE);
  RECEIVE:       if RECEIVE (Ai, Level) then RETURN(TRUE) else RETURN(FALSE);
  OTHERWISE:     if OTHERWISE (Ai, Level) then RETURN(TRUE) else RETURN(FALSE);
  TIMEOUT:       if TIMEOUT (Ai, Level) then RETURN(TRUE) else RETURN(FALSE);
  IMPLICIT_SEND: if IMPLICIT_SEND (Level) then RETURN(TRUE)
end
```

B.5.4.2 *Fonctions appliquées aux événements TTCN: description en langage naturel*

Si la déclaration TTCN est un événement, elle sera évaluée comme spécifié au § B.5.5.2 s'il s'agit d'un événement SEND, au § B.5.6.2 s'il s'agit d'un événement RECEIVE, au § B.5.7.2 s'il s'agit d'un événement OTHERWISE, au § B.5.8.2 s'il s'agit d'un événement TIMEOUT, ou au § B.5.9.2 s'il s'agit d'un événement IMPLICIT SEND.

B.5.5 *Exécution de l'événement SEND*

B.5.5.1 *Exécution de l'événement SEND: description en langage pseudo-formalisé*

- **function SEND** (PCOidentifier,
  ASPidentifier or PDUidentifier,
  Qualifier,
  Assignment,
  TimerOperation,
  ConstraintsReference,
  Verdict,
  Level): **BOOLEAN**

  (*Tous les paramètres sauf Level sont tirés de $A_i$*)

  **begin**
  **if** EVALUATE_BOOLEAN (Qualifier) **then**
  **begin**
  BUILD_SEND_OBJECT (ASPidentifierOrPDUidentifier, ConstraintsReference);
  EXECUTE_ASSIGNMENT (Assignment);
  SEND_EVENT (PCOidentifier);
  TIMER_OP (TimerOperation);
  VERDICT (Verdict);
  Level:=NEXT_LEVEL;
  LOG (PCOidentifier, SendObject);
  RETURN(TRUE)
  **end**
  **else** RETURN(FALSE)

  **end**

- **function BUILD_SEND_OBJECT** (ASPidentifierOrPDUidentifier**,**
  ConstraintsReference): **BOOLEAN**

  **begin**
  SendObject := (**an instance of** ASPidentifierOrPDUidentifier, **whose parameters/fields have the values specified by** ConstraintsReference);
  RETURN(TRUE)

  **end**

B.5.5.2 *Exécution de l'événement SEND: description en langage naturel*

Le contenu de la primitive ASP ou de l'unité PDU, comme spécifié dans l'entrée de référence de contraintes nommées, doit être envoyé. On notera qu'en présence d'un qualificateur, l'événement SEND n'est exécuté que si ce qualificateur a la valeur TRUE.

Etape 1 S'il existe un qualificateur, il est évalué avant tout autre traitement.

– Si le qualificateur prend la valeur FALSE, l'événement SEND n'est pas exécuté.

– Si le qualificateur prend la valeur TRUE, passer à l'étape 2.

Etape 2 Les valeurs spécifiées dans la référence de contraintes nommées seront affectées à la déclaration de primitive ASP ou d'unité PDU.

Etape 3 Si la caractéristique de chaînage dynamique a été utilisée, la valeur spécifiée dans l'entrée de référence de contraintes sera affectée au paramètre ou au champ correspondant de la primitive ASP ou de l'unité PDU à envoyer.

Le fait d'utiliser la caractéristique de chaînage dynamique a pour effet de mettre en mémoire une copie de la contrainte nommée dans le paramètre ou champ nommé de la primitive ASP ou de l'unité PDU constituée à des fins de comparaison. La structure définie pour la référence de contraintes associées est utilisée pour ce paramètre ou ce champ nommé.

Etape 4 S'il y a une déclaration d'affectation, cette affectation sera effectuée comme indiqué au § B.5.12.2.

Etape 5 La primitive ASP ou l'unité PDU est maintenant entièrement renseignée conformément aux spécifications données. Le testeur inférieur (LT) ou le testeur supérieur (UT) enverra la primitive ASP ou l'unité PDU. (Si un point PCO a été déclaré, la primitive ASP ou l'unité PDU doit être envoyée au niveau de ce point PCO. Si le point PCO n'a pas été déclaré, c'est-à-dire si le test utilise un seul point PCO, la primitive ASP ou l'unité PDU est envoyée à partir du point PCO inférieur.)

Etape 6 Si une ou plusieurs opérations de temporisation ont été codées sur la ligne comportementale, ces opérations seront effectuées comme indiqué au § B.5.13.

Etape 7 Si un verdict est codé, l'exécuter comme indiqué au § B.5.16.2.

Etape 8 Consigner dans le journal de conformité les informations suivantes ainsi que les informations spécifiées au § B.5.17.2:

– le point PCO au niveau duquel l'événement SEND a eu lieu;

– la primitive ASP, l'unité PDU ou les procédures de coordination des tests (TCP) entièrement définie(s) qui a(ont) été envoyée(s).

### B.5.6 *Exécution de l'événement RECEIVE*

#### B.5.6.1 *Exécution de l'événement RECEIVE: description en langage pseudo-formalisé*

- **function RECEIVE** (PCOidentifier,
  ASPidentifier or PDUidentifier,
  Qualifier,
  Assignment,
  TimerOperation,
  ConstraintsReference,
  Verdict,
  Level): **BOOLEAN**

(*Tous les paramètres sauf Level sont tirés de $A_i$*)

```
begin
    if  RECEIVE_EVENT (PCOidentifier) then
      begin
        if (RECEIVED_OBJECT(ASPidentifierOrPDUidentifier, ConstraintsReference)
            AND EVALUATE_BOOLEAN (Qualifier))
            then
              begin
                EXECUTE_ASSIGNMENT (Assignment);
                TIMER_OP (TimerOperation);
                REMOVE_OBJECT (PCOidentifier);
                VERDICT (Verdict);
                Level:=NEXT_LEVEL;
                LOG (PCOidentifier, ReceivedObject);
                RETURN(TRUE)
              end
        else RETURN(FALSE)
      end
    else RETURN(FALSE)
end
```

- **function RECEIVE_EVENT** (PCOidentifier)**: BOOLEAN**

(*L'objet réel N'EST PAS enlevé de la file PCOidentifier*)

```
begin
    if INPUT_Q (PCOidentifier) NOT empty then
        begin
          ReceivedObject := copy of object at head of INPUT_Q (PCOidentifier);
          RETURN(TRUE)
        end
    else RETURN(FALSE)
end
```

- **function RECEIVED_OBJECT** (ASPidentifierOrPDUidentifier,
  ConstraintsReference): **BOOLEAN**

```
begin
    if ( (ReceivedObject is ASPidentifierOrPDUidentifier)
        AND
        (parameters/fields of ReceivedObject have the values specified by the
        ConstraintsReference) )
    then RETURN(TRUE)
  else  RETURN(FALSE)
end
```

B.5.6.2 *Exécution de l'événement RECEIVE: description en langage naturel*

Le testeur LT ou UT vérifiera si le contenu de la primitive ASP ou de l'unité PDU, comme spécifié dans l'entrée de référence des contraintes nommées, a été reçu. On notera qu'en présence d'un qualificateur, l'événement RECEIVE ne peut concorder que si le qualificateur a la valeur TRUE.

Etape 1 Si l'image instantanée prise au début de l'itération en cours du contrôle de concordance pour ce niveau d'indentation de propositions SI montre qu'il n'y a pas de primitive ASP ou d'unité PDU entrante, cet événement RECEIVE ne peut pas être mis en concordance.

Sinon, passer à l'étape 2.

Etape 2 Une copie de la primitive ASP ou de l'unité PDU est constituée, qui utilise la structure définie dans la déclaration de primitive ASP ou d'unité PDU plus les valeurs spécifiées dans la référence aux contraintes nommées. On comparera cette copie à la primitive ASP ou l'unité PDU entrante (le cas échéant) en vue de déterminer si l'événement RECEIVE peut concorder comme spécifié.

Etape 3 Si la caractéristique de chaînage dynamique a été utilisée, la valeur spécifiée dans l'entrée de référence des contraintes sera affectée au paramètre ou champ approprié de la primitive ASP ou de l'unité PDU qui sera utilisée dans la comparaison.

L'utilisation de la caractéristique de chaînage dynamique a pour effet d'enregistrer une copie de la contrainte nommée dans le paramètre ou le champ nommé de la primitive ASP ou de l'unité PDU en cours d'élaboration à des fins de comparaison. La structure définie pour la référence des contraintes associées est utilisée pour ce paramètre ou champ nommé.

Etape 4 La primitive ASP ou l'unité PDU est maintenant entièrement renseignée conformément aux spécifications données. Le testeur LT ou UT comparera la copie créée à partir des données spécifiées avec la primitive ASP ou l'unité PDU reçue le cas échéant dans l'image instantanée.

Si un point PCO a été déclaré, la primitive ASP ou l'unité PDU aura été reçue au niveau de ce point PCO. Si aucun point PCO n'a été déclaré – c'est-à-dire si le test utilise un seul point PCO – la primitive ASP ou l'unité PDU aura été reçue au point PCO inférieur.

Etape 5 S'il y a un qualificateur, il sera évalué.

– Si le qualificatif prend la valeur FALSE, l'événement RECEIVE ne peut pas concorder.

– Si le qualificatif prend la valeur TRUE, passer à l'étape 6.

Etape 6 Si le testeur LT ou UT n'est pas en mesure de faire concorder la primitive ASP ou l'unité PDU comme spécifié (par exemple, aucune primitive ASP ou unité PDU n'est parvenue, une primitive ASP ou une unité PDU est parvenue mais sans concorder avec les données spécifiées, ou la primitive ASP ou l'unité PDU concorde avec ces données mais le qualificateur n'a pas la valeur «vrai»), cet événement RECEIVE doit être considéré comme non concordant, ce qui veut dire qu'une nouvelle mise en concordance devra être tentée sur l'occurrence suivante de l'événement RECEIVE.

Si l'événement RECEIVE concorde avec les données spécifiées, passer à l'étape 7. (La primitive ASP ou l'unité PDU entrante qui vient de concorder sera extraite de la file d'attente du point PCO d'arrivée et ne pourra plus être mise en concordance avec un événement subséquent quelconque du test élémentaire.)

Etape 7 S'il y a une déclaration d'affectation, elle sera exécutée comme indiqué au § B.5.12.2.

Etape 8 Si une ou plusieurs opérations de temporisation ont été codées sur la ligne comportementale, elles seront exécutées comme indiqué au § B.5.13.

Etape 9 Si un verdict est codé, le traiter comme indiqué au § B.5.16.2.

Etape 10 Consigner dans le journal de conformité les informations suivantes ainsi que les informations spécifiées au § B.5.17.2:

– le point PCO au niveau duquel l'événement RECEIVE a eu lieu;

– la primitive ASP, l'unité PDU ou la procédure TCP reçue avec sa définition complète.

B.5.7 *Exécution de l'événement OTHERWISE (sinon)*

B.5.7.1 *Exécution de l'événement OTHERWISE: description en langage pseudo-formalisé*

```
• function OTHERWISE (PCOidentifier,
                      Qualifier,
                      Assignment,
                      TimerOperation,
                      Verdict,
                      Level): BOOLEAN

    (*Tous les paramètres sauf Level sont tirés de Ai*)

begin
    if ( (RECEIVE_EVENT (PCOidentifier))
          AND EVALUATE_BOOLEAN (Qualifier) )
      then
          begin
              EXECUTE_ASSIGNMENT (Assignment);
              TIMER_OP (TimerOperation);
              REMOVE_OBJECT (PCOidentifier);
              VERDICT (Verdict);
              Level:=NEXT_LEVEL;
              LOG (PCOidentifier, ReceivedObject);
              RETURN(TRUE)
            end
    else RETURN(FALSE)
end
```

B.5.7.2 *Exécution de l'événement OTHERWISE (sinon): description en langage naturel*

Le testeur acceptera toute donnée entrante n'ayant pas encore concordé avec une proposition SI antérieure à cet événement OTHERWISE. On notera qu'en présence d'un qualificateur, l'événement «sinon» ne concorde que si le qualificateur a la valeur TRUE.

Etape 1 Si un point PCO a été déclaré, la primitive ASP ou l'unité PDU aura été reçue au niveau de ce point. Faute d'une telle déclaration – c'est-à-dire si le test utilise un seul point PCO – la primitive ASP ou l'unité PDU sera reçue au point PCO inférieur.

Etape 2 S'il y a un qualificateur, il sera évalué avant tout autre traitement:

– si le qualificateur a la valeur FALSE, l'événement OTHERWISE ne peut concorder;

– si le qualificateur a la valeur TRUE, passer à l'étape 3.

Etape 3 Si le testeur n'est pas en mesure de recevoir une primitive ASP ou une unité PDU (c'est-à-dire si aucune primitive ASP ou unité PDU n'est parvenue), on considère que l'événement OTHERWISE n'a pas concordé et on passe à la proposition SI qui suit cet événement OTHERWISE.

Si l'événement OTHERWISE s'est déroulé avec succès, passer à l'étape 4. (La primitive ASP ou l'unité PDU entrante qui vient d'être mise en concordance ne pourra être réutilisée pour une nouvelle concordance avec un événement ultérieur quelconque du test élémentaire.)

Etape 4 S'il y a une déclaration d'affectation, l'exécuter comme indiqué au § B.5.12.2.

Etape 5 Si une ou plusieurs opérations de temporisation ont été codées sur la ligne comportementale, les exécuter comme indiqué au § B.5.13.

Etape 6 Si un verdict est codé, le traiter comme indiqué au § B.5.16.2.

Etape 7 Consigner dans le journal de conformité les informations suivantes ainsi que les informations spécifiées au § B.5.17.2:

– le point PCO au niveau duquel l'événement OTHERWISE a eu lieu;

– la primitive ASP ou l'unité PDU reçue avec la définition complète.

### B.5.8 *Exécution de l'événement TIMEOUT (fin de temporisation)*

#### B.5.8.1 *Exécution de l'événement TIMEOUT: description en langage pseudo-formalisé*

```
function TIMEOUT (TimerIdentifier,
                  Qualifier,
                  Assignment,
                  TimerOperation,
                  Verdict,
                  Level): BOOLEAN

   (*Tous les paramètres sauf Level sont tirés de Ai*)

   begin
   if EVALUATE_BOOLEAN (Qualifier) then
         begin
               if (TIMER_EXPIRED (TimerIdentifier)) then
                     begin
                           EXECUTE_ASSIGNMENT (Assignment);
                           TIMER_OP (TimerOperation);
                           VERDICT (Verdict);
                           Level:=NEXT_LEVEL;
                           LOG (TimerIdentifier);
                           RETURN(TRUE)
                     end
               else RETURN(FALSE)
         end
   else RETURN(FALSE)

   end

   function TIMER_EXPIRED (TimerIdentifier): BOOLEAN

   begin
   if (timer has expired) then

   begin
   reset expired timer; (*voir le § B.5.8.2*)
         RETURN(TRUE)
   end

   else
   RETURN(FALSE)

end
```

B.5.8.2 *Exécution de l'événement TIMEOUT: description en langage naturel*

Le testeur vérifiera si la temporisation nommée est arrivée à expiration. (Si aucun nom de temporisateur n'est donné, le testeur vérifiera si une temporisation quelconque est arrivée à expiration.) On notera qu'en présence d'un qualificateur, l'événement TIMEOUT ne concordera que si ce qualificateur a la valeur TRUE.

Etape 1 S'il y a un qualificateur, il sera évalué avant tout autre traitement.

– Si le qualificateur a la valeur FALSE, l'événement TIMEOUT ne concorde pas.

– Si le qualificateur a la valeur TRUE, passer à l'étape 2.

Etape 2 Voir si l'une quelconque des temporisations explicitement ou implicitement nommées dans l'événement TIMEOUT est arrivée à expiration après déclenchement.

– Si aucun identificateur de temporisateur n'est spécifié, le testeur recherchera une fin de temporisation *quelconque*. Dans ce cas, tous les temporisateurs arrivés à expiration sont réinitialisés (et laissés à l'arrêt). Les entrées de fin de temporisation sont supprimées de la liste de fins de temporisation.

– Si un identificateur de temporisateur est spécifié, le testeur vérifiera si ce temporisateur a été déclenché et s'il est arrivé à expiration. Si oui, ce temporisateur est réinitialisé (et laissé à l'arrêt). L'entrée de fin de temporisation est supprimée de la liste de fins de temporisation.

– Si aucun temporisateur n'est arrivé à expiration, l'événement TIMEOUT ne peut pas concorder, c'est-à-dire que le système passe à la proposition SI suivante.

Etape 3 S'il y a une déclaration d'affectation, elle est exécutée comme indiqué au § B.5.12.2.

Etape 4 Si une ou plusieurs opérations de temporisation ont été codées sur la ligne comportementale, ces opérations sont exécutées comme indiqué au § B.5.13.

Etape 5 Si un verdict est codé, le traiter comme indiqué au § B.5.16.2.

Etape 6 Consigner dans le journal de conformité les informations suivantes ainsi que les informations spécifiées au § B.5.17.2:

– nom du temporisateur arrivé à expiration.

B.5.9 *Exécution de l'événement IMPLICIT SEND (envoi implicite)*

B.5.9.1 *Exécution de l'événement IMPLICIT SEND: description en langage pseudo-formalisé*

- **function IMPLICIT_SEND** (Level): **BOOLEAN**

**begin**

(*Evaluer la fonction IMPLICIT_SEND conformément au § 14.9.6**)

Level:=NEXT_LEVEL;

RETURN(TRUE)

**end**

B.5.9.2 *Exécution de l'événement IMPLICIT SEND: description en langage naturel*

L'instance sous test fait le nécessaire pour envoyer le contenu de la primitive ASP ou de l'unité PDU, comme spécifié dans l'entrée de référence des contraintes nommées.

Si la caractéristique de chaînage dynamique a été utilisée, la valeur spécifiée dans l'entrée de référence des contraintes sera affectée au paramètre ou champ approprié de la primitive ASP ou de l'unité PDU à envoyer.

B.5.10 *Exécution de PSEUDO_EVENT (pseudo-événement)*

B.5.10.1 *Exécution des PSEUDO_EVENTS: description en langage pseudo-formalisé*

- **function EVALUATE_PSEUDO_EVENT** (Qualifier,
  Assignment,
  TimerOperation,
  Verdict,
  Level): **BOOLEAN**

(*Tous les paramètres sauf Level sont tirés de $A_i$*)

```
begin
if EVALUATE_BOOLEAN (Qualifier)
      then
         begin
            ASSIGNMENT (Assignment);
            TIMER_OP (TimerOperation);
            VERDICT (Verdict);
            Level:=NEXT_LEVEL;
            LOG ( );
            RETURN(TRUE)
         end
      else RETURN(FALSE)
end
```

B.5.10.2 *Exécution des PSEUDO_EVENTS: description en langage naturel*

Si la déclaration TTCN est un pseudo-événement, elle sera évaluée comme spécifié au § B.5.11.2 s'il s'agit d'une expression booléenne, au § B.5.12.2 s'il s'agit d'une déclaration d'affectation, au § B.5.13.2 s'il s'agit d'un pseudo-événement START (déclenchement) de temporisation, au § B.5.13.3 s'il s'agit d'un pseudo-événement CANCEL (annulation) de temporisation ou au § B.5.13.4 s'il s'agit d'un pseudo-événement READ (lecture) de temporisation.

Etape n Après l'achèvement du pseudo-événement, consigner dans le journal de conformité les informations spécifiées au § B.5.17.2.

B.5.11 *Exécution des expressions BOOLEAN (booléennes)*

B.5.11.1 *Exécution des expressions BOOLEAN (booléennes): description en langage pseudo-formalisé*

- **function EVALUATE_BOOLEAN** (Qualifier): **BOOLEAN**

(*Pour une définition plus détaillée, voir le § B.5.11.2*)

```
begin
if no argument then RETURN(TRUE)
else   begin
            if Qualifier then RETURN(TRUE)
            else RETURN(FALSE)
       end

end
```

B.5.11.2 *Exécution des expressions BOOLEAN (booléennes): description en langage naturel*

Une expression booléenne (c'est-à-dire un qualificateur) spécifie une condition qui doit être testée. Cette condition est soit vraie (TRUE), soit fausse (FALSE). Une expression booléenne peut être déclarée comme partie de ligne de déclaration (c'est-à-dire sur la même ligne qu'un événement SEND, RECEIVE, TIMEOUT ou OTHERWISE) ou comme ligne de déclaration à part entière (c'est-à-dire comme pseudo-événement).

Etape 1 L'expression booléenne est évaluée pour déterminer si la condition spécifiée est TRUE ou FALSE. Les règles normales de la logique booléenne s'appliquent ainsi que les règles de priorité spécifiées au § 11.4.2.1.

Etape 2 Si l'expression booléenne prend la valeur FALSE, cette expression n'est pas un événement concordant. Sauter les étapes restantes du présent paragraphe.

Si l'expression booléenne concorde, la suite du traitement diffère suivant que cette expression a été codée comme partie d'un événement (à savoir SEND, RECEIVE, TIMEOUT ou OTHERWISE) ou comme pseudo-événement.

Si l'expression booléenne a été codée comme partie d'un événement, cette partie de la ligne de déclaration est considérée comme concordante et le restant de la ligne de déclaration sera évalué conformément à la sémantique prévue pour ce type d'événement. Sauter les étapes suivantes, qui ne s'appliquent qu'aux expressions booléennes codées comme pseudo-événements.

Si cette expression booléenne a été codée comme pseudo-événement, elle est considérée comme un comportement concordant. Passer à l'étape suivante pour traiter le restant de la ligne comportementale de l'expression booléenne.

Etape 3 S'il y a une déclaration d'affectation, l'exécuter comme indiqué au § B.5.12.2.

Etape 4 Si une ou plusieurs opérations de temporisation ont été codées sur la ligne comportementale, les exécuter comme indiqué au § B.5.13.

Etape 5 Si un verdict est codé, le traiter comme indiqué au § B.5.16.2.

### B.5.12 *Exécution de ASSIGNMENT (affectation)*

#### B.5.12.1 *Exécution de la fonction EXECUTE_ASSIGNMENT: description en langage pseudo-formalisé*

- **function EXECUTE_ASSIGNMENT** (Assignment): **BOOLEAN**

(*On notera que les affectations du type <*PDUidentifier*>. <*FIELDidentifier*>, etc., affectent (réaffectent) des valeurs aux champs/paramètres de l'objet SendObject. Ce type d'affectation ne doit pas être utilisé avec un événement RECEIVE*)

```
begin
    if no argument then RETURN(TRUE)
    else begin
        execute the clauses in Assignment in a left-to-right order;
        RETURN(TRUE)
    end
end
```

#### B.5.12.2 *Exécution des ASSIGNMENTS: description en langage naturel*

Une déclaration d'affectation spécifie que la variable située dans le membre gauche de la déclaration doit prendre la valeur du membre droit de cette déclaration. Une déclaration d'affectation peut faire partie d'une ligne de déclaration (c'est-à-dire sur la même ligne qu'un événement SEND, RECEIVE, TIMEOUT ou OTHERWISE), être combinée avec une expression booléenne, ou constituer une ligne comportementale à part entière (c'est-à-dire un pseudo-événement).

Etape 1 La manière dont l'affectation est exécutée dépend du format utilisé pour spécifier le membre droit. Les termes sont évalués de gauche à droite, en respectant les règles de priorité indiquées au tableau 3/X.292.

Etape 2 Une fois l'affectation exécutée, la suite du traitement diffère suivant que la déclaration d'affectation a été codée comme partie d'un événement (à savoir SEND, RECEIVE, TIMEOUT ou OTHERWISE), ou comme pseudo-événement.

Si cette déclaration d'affectation a été codée comme pseudo-événement, passer à l'étape suivante pour traiter la suite de la ligne comportementale de la déclaration d'affectation.

Etape 3 Si une ou plusieurs opérations de temporisation ont été codées sur la ligne comportementale, les exécuter comme indiqué au § B.5.13.

Etape 4 Si un verdict est codé, le traiter comme indiqué au § B.5.16.2.

B.5.13 *Exécution des opérations TIMER (temporisation)*

B.5.13.1 *Exécution des opérations TIMER: description en langage pseudo-formalisé*

- **function TIMER_OP** (TimerOperation): **BOOLEAN**

```
begin
if no argument then RETURN(TRUE)
else  case TimerOperation of
            START_TIMER:    if    START_TIMER(TimerOperation) then RETURN(TRUE)
                            else RETURN(FALSE); (*voir le § B.5.13.2 pour l'opération
                            START_TIMER*)
            CANCEL_TIMER:   if    CANCEL_TIMER(TimerOperation) then RETURN(TRUE)
                            else RETURN(FALSE); (*voir le § B.5.13.3 pour l'opération
                            CANCEL_TIMER*)
            READ_TIMER:     if    READ_TIMER(TimerOperation) then RETURN(TRUE)
                            else RETURN(FALSE); (*voir le § B.5.13.4 pour l'opération
                                        READ_TIMER*)
      end
end
```

B.5.13.2 *Exécution de START (déclenchement de temporisation): description en langage naturel*

L'opération START spécifie le déclenchement d'une temporisation. Cette opération peut être codée comme partie d'une ligne de déclaration (c'est-à-dire sur la même ligne qu'un événement SEND, RECEIVE, OTHERWISE ou TIMEOUT), comme ligne comportementale à part entière, ou combinée avec un qualificateur ou une déclaration d'affectation.

Etape 1 Déterminer la durée à adopter pour cette instance de temporisation. Si aucune durée n'est spécifiée telle qu'une valeur ou une expression entière, on utilisera la durée par défaut spécifiée dans la partie déclarations des temporisations.

Si une durée a été spécifiée, elle aura priorité sur la durée par défaut donnée dans la déclaration de la temporisation. Le remplacement d'une durée de temporisation par défaut par la spécification d'une durée en argument de l'opération START ne vaut que pour cette instance du temporisateur – si ce temporisateur est déclenché en un autre point quelconque du test, il n'est pas affecté par cette substitution de durée.

Etape 2 Si le temporisateur est déjà déclenché, il doit être annulé et redéclenché sans délai. Si le temporisateur n'est pas encore déclenché, il doit l'être avec une valeur initiale indiquant un temps écoulé nul. Toute entrée relative à ce temporisateur dans la liste de fins de temporisation est supprimée de la liste.

Etape 3 Si une autre opération de temporisation est codée à la suite de cette opération START, l'exécuter (sauf s'il s'agit d'un verdict).

Etape 4 Si un verdict est codé, l'exécuter comme indiqué au § B.5.16.2.

B.5.13.3 *Exécution de CANCEL (annulation de temporisation)*

L'opération d'annulation de temporisation CANCEL spécifie l'arrêt d'un ou plusieurs temporisateurs. Cette opération peut être codée comme partie d'une ligne de déclaration (c'est-à-dire sur la même ligne qu'un événement SEND, RECEIVE, OTHERWISE ou TIMEOUT), comme ligne comportementale à part entière ou en combinaison avec un qualificateur ou une déclaration d'affectation.

Etape 1 Déterminer le nom du ou des temporisations à annuler:

– si aucun identificateur de temporisation n'est spécifié, le testeur LT ou UT doit annuler *toutes* les temporisations en marche;

– si un identificateur est spécifié, le testeur LT ou UT doit annuler la temporisation en marche correspondante.

Etape 2 Le descripteur d'état du ou des temporisateurs nommés ou impliqués est mis sur "not running" («arrêté»). Le ou les temporisateurs sont réinitialisés à zéro. Si la liste de fins de temporisation contient une entrée concernant ce ou ces temporisateurs, ces entrées sont supprimées de la liste.

Etape 3 Si une autre opération de temporisation est codée à la suite de cette opération CANCEL, l'exécuter (sauf s'il s'agit d'un verdict).

Etape 4 Si un verdict est codé, l'exécuter comme indiqué au § B.5.16.2.

B.5.13.4 *READTIMER (lecture de temporisation): description en langage naturel*

L'opération de lecture de temporisation READTIMER spécifie que le temps compté par un temporisateur déclenché doit être mémorisé dans une variable. Le temporisateur n'est pas interrompu. Cette opération peut être codée comme partie d'une ligne de déclaration (c'est-à-dire sur la même ligne qu'un événement SEND, RECEIVE, OTHERWISE ou TIMEOUT), ou comme ligne comportementale à part entière, ou en combinaison avec un qualificateur ou une déclaration d'affectation.

Etape 1 Lire la valeur du temporisateur dont le nom est spécifié. Mémoriser le temps écoulé dans la variable nommée. Les unités renvoyées sont les mêmes que les unités déclarées pour ce type de temporisateur.

Si le temporisateur est à l'arrêt, la variable nommée sera mise à zéro.

Etape 2 Si une autre opération de temporisation est codée à la suite de cette opération READTIMER, l'exécuter (sauf s'il s'agit d'un verdict).

Etape 3 Si un verdict est codé, l'exécuter comme indiqué au § B.5.16.2.

B.5.14 *Fonctions appliquées aux constructions TTCN*

B.5.14.1 *Fonctions appliquées aux constructions TTCN: description en langage pseudo-formalisé*

- **function EVALUATE_CONSTRUCT** (Construct, Level): **BOOLEAN**

(*Tous les paramètres sauf Level sont tirés de $A_i$*)

(*Etant donné que l'arbre de test élémentaire est complètement développé avant le traitement, les constructions REPEAT et ATTACH ne sont jamais rencontrées, voir les § B.4.3 et B.4.4*)

**case** Construct **of**

**GOTO:** GOTO(Label, Level);

RETURN(TRUE)

**end**

B.5.14.2 *Fonctions appliquées aux constructions TTCN: description en langage naturel*

Si la déclaration TTCN est une construction TTCN de type GOTO, elle sera évaluée comme spécifié au § B.5.15.2; pour la description des constructions REPEAT et ATTACH, voir le § B.4.

B.5.15 *Exécution de la construction GOTO*

B.5.15.1 *Exécution de la construction GOTO: description en langage pseudo-formalisé*

- **function GOTO** (Label, Level): **BOOLEAN**

**begin**

Level:=LABELED_LEVEL(Label);

RETURN(TRUE)

**end**

B.5.15.2 *Exécution de la construction GOTO: description en langage naturel*

Le testeur LT ou UT doit transférer le contrôle de l'événement courant (c'est-à-dire la construction GOTO) à l'ensemble des propositions SI portant l'étiquette spécifiée. L'exécution se poursuit alors à ce nouveau niveau d'indentation.

B.5.16 *Verdict*

B.5.16.1 *Verdict: description en langage pseudo-formalisé*

- **function VERDICT** (VerdictColumnEntry): **BOOLEAN**

```
begin
if VerdictColumnEntry is blank then RETURN(TRUE)
else begin
        if VerdictColumnEntry = R then LOG(Verdict implied by R);
        else LOG(VerdictColumn);
        if VerdictColumnEntry is preliminary result then (*comporte un verdict*)
            begin
            if ( (R = none) or (R = pass and VerdictColumnEntry <> (PASS))
              or (R = inconc and VerdictColumnEntry = (FAIL) ) then
            begin
                if VerdictColumnEntry = (PASS) then R := pass;
                if VerdictColumnEntry  = (INCONC) then R := inconc;
                if VerdictColumnEntry = (FAIL) then R := fail;
              end
    end

else   (*verdict final*)
begin
    reset test case variables all timers;
    if ( (VerdictColumnEntry = R and R = none) or
        (VerdictColumnEntry = PASS and R = inconc) or
        (VerdictColumnEntry = PASS and R = fail) or
        (VerdictColumnEntry = INCONC and R = fail) )
      then raise test case error;
      STOP (*fin du test élémentaire*)
end
end
end
```

B.5.16.2 *Verdict: description en langage naturel*

Si un verdict est codé, l'exécuter.

– Si le verdict est mis entre parenthèses, la variable R de résultat provisoire sera mise à jour conformément à l'algorithme de verdict du § 15.17.2. Le verdict déclaré est consigné dans le journal de conformité.

– Si le verdict est R, la valeur courante de la variable R de résultat provisoire sera utilisée comme verdict du test élémentaire. Si R à la valeur «néant», signaler une erreur de test élémentaire.

Si le verdict est PASS (succès), INCONC (non concluant) ou FAIL (échec), le verdict déclaré sera pris comme verdict final du test élémentaire. Si le verdict final contredit le verdict préliminaire, signaler une erreur *TestCaseError*.

B.5.17 *Journal de conformité (log)*

B.5.17.1 *Journal LOG: description en langage pseudo-formalisé*

- **procedure LOG (Variable number of arguments)**

```
begin
log the sequence number of the event line (if any);
log the label associated with the event line (if any);

log the arguments passed to LOG;

log the assignment(s) made (if any);
log the timer operation(s) performed (if any);
log the verdict or preliminary result associated with the event line (if any);
log current time; (*l'instant courant peut être absolu ou relatif*)
end
```

B.5.17.2 *Journal de conformité: description en langage naturel*

Consigner les informations suivantes dans le journal de conformité:

– numéro de séquence de la ligne événement (le cas échéant);

– étiquette associée à la ligne événement (le cas échéant);

– affectations effectuées (le cas échéant);

– opérations de temporisation exécutées (le cas échéant);

– verdict ou résultat préliminaire associé à la ligne événement (le cas échéant);

– horodatage.

B.5.18 *Autres fonctions diverses utilisées par le langage pseudo-formalisé*

- **function FIRST_LEVEL: LEVEL**

  **begin**
  RETURN **(set of alternatives at first level of indentation)**
  **end**

- **function NEXT_LEVEL: LEVEL**

  **begin**
  **if** (set of alternatives at next level of indentation exist) **then**
  RETURN (set of alternatives at next level of indentation)
  **else**
  VERDICT(R)
  STOP (*fin du test élémentaire*)
  **end**

- **function LABELED_LEVEL** (Label): **LEVEL**

  **begin**
  RETURN (**set of alternatives at the level of indentation indicated by** Label)
  **end**

- **function RETURN** (argument): **BOOLEAN or LEVEL or QUEUE**

  exit the current function immediately and return the value of argument
  **end**

- **function OUTPUT_Q** (PCOidentifier): **QUEUE**

(*En notation TTCN, chaque point PCO est modélisé par deux files d'attente de type FIFO (premier entré premier sorti) non bornées: une file d'entrée INPUT (vers le testeur LT ou UT) et une file de sortie OUTPUT (depuis le testeur LT ou UT). Dans les suites de tests qui n'utilisent qu'un seul point PCO, lorsque ce point PCO n'est pas explicitement associé à un événement, le seul point PCO (par défaut) est obtenu des déclarations de points PCO*)

**begin**
**if no argument then** RETURN(**default PCO output queue**)
**else** RETURN(**output queue identified by** PCOidentifier)
**end**

- **function INPUT_Q** (PCOidentifier): **QUEUE**

(*En notation TTCN, chaque point PCO est modélisé par deux files d'attente de type FIFO non bornées: une file d'entrée INPUT (vers le testeur LT ou UT) et une file de sortie OUTPUT (depuis le testeur LT ou UT). Dans les suites de tests qui n'utilisent qu'un seul point PCO, lorsque ce point PCO n'est pas explicitement associé à un événement, le seul point PCO (par défaut) est obtenu des déclarations de points PCO*)

**begin**

**if no argument then** RETURN(**default PCO input queue**)

**else** RETURN(**input queue identified by** PCOidentifier)

**end**

- **procedure TAKE_SNAPSHOT**

(*La sémantique d'image instantanée est utilisée pour les événements RECEIVE et les fins de temporisation, c'est-à-dire chaque fois que pour un ensemble donné des propositions SI une image instantanée est prise fixant les événements reçus et les fins de temporisation intervenues. Seuls les événements identifiés dans cette image instantanée seront utilisés pour les concordances du cycle suivant de parcours des propositions SI*)

**begin**

**update PCO input queues;**

**update TIMER queue;**

**end**

- **procedure STOP**

**begin**

**reset test case variables;**

**reset PCO queue;**

**reset TIMER queue;**

**reset timers;**

**terminate execution immediately**

**end**

# Remplacée par une version plus récente

ANNEXE C

(à la Recommandation X.292)

**Formulaires compacts**

(Cette annexe fait partie intégrante de la présente Recommandation)

## C.1 *Introduction*

Sur option, il est possible de présenter un grand nombre de contraintes et de tests élémentaires sur une seule table. Cela peut être utile pour mettre en lumière les relations existant entre les contraintes simples ou les tests élémentaires simples. La présente annexe énonce les spécifications d'utilisation des formulaires compacts de contraintes et des formulaires compacts de tests élémentaires et en donne quelques exemples. Ces formulaires sont spécifiques et diffèrent des présentations générales du § 7.3. Etant donné que les nouveaux formulaires ne sont qu'une autre manière de présenter les mêmes informations, aucune notation TTCN.MP n'y est associée. Les informations contenues dans une table compacte des contraintes ou une table compacte de tests élémentaires peuvent être traduites dans la notation TTCN.MP associée aux nombreuses tables de contraintes simples et aux nombreuses tables de tests élémentaires qui contiennent les mêmes informations.

## C.2 *Formulaires compacts pour les contraintes*

### C.2.1 *Spécifications*

Le regroupement de plusieurs tables de contrainte simple en une seule table compacte des contraintes est soumis aux conditions suivantes:

a) les contraintes ont le même type de primitive ASP, d'unité PDU, structuré ou ASN.1;

b) il n'y a aucune entrée dans la colonne commentaires de l'une quelconque des tables de contrainte simple.

*Remarque* – Si les tables de contrainte simple ne comportent des commentaires que sur la ligne de commentaire détaillé en bas de page (c'est-à-dire si la colonne commentaires est vide), il est possible de regrouper plusieurs contraintes en un format compact. En ce cas, les commentaires détaillés particuliers des formulaires simples sont regroupés en un commentaire unique sur la ligne de commentaire détaillé en bas de page du formulaire compact.

### C.2.2 *Formulaires compacts pour les contraintes de primitive ASP*

Lorsqu'une contrainte ne comporte que quelques paramètres ou que le nombre de contraintes est peu élevé, elles peuvent être présentées dans la version compacte du formulaire de contraintes de primitive ASP.

| **Déclarations de contraintes de primitive ASP** | | | | | |
|---|---|---|---|---|---|
| **Type de primitive ASP:** *ASP_Identifier* | | | | | |
| **Nom de contrainte** | **Chemin de dérivation** | **Nom de paramètre** | | | **Commentaires** |
| | | *ASP_ParIdentifier*$_1$ | | *ASP_ParIdentifier*$_n$ | |
| *ConsId-&ParList*$_1$ | *Derivation-Path*$_1$ | *ConstraintValue-&Attributes*$_{1,1}$ | | *ConstraintValue-&Attributes*$_{1,n}$ | *[FreeText]*$_1$ |
| *ConsId-&ParList*$_2$ | *Derivation-Path*$_2$ | *ConstraintValue-&Attributes*$_{2,1}$ | | *ConstraintValue-&Attributes*$_{2,n}$ | *[FreeText]*$_2$ |
| ⋮ | | ⋮ | | ⋮ | ⋮ |
| *ConsId-&ParList*$_m$ | *Derivation-Path*$_m$ | *ConstraintValue-&Attributes*$_{m,1}$ | | *ConstraintValue-&Attributes*$_{m,n}$ | *[FreeText]*$_m$ |

**Formulaire C-1 – Déclarations de contraintes de primitive ASP – Forme compacte**

Ce formulaire est utilisé pour les primitives ASP et leurs paramètres tout comme le formulaire de déclaration de contraintes d'unité PDU est utilisé pour les unités PDU et leurs champs (voir le § C.2.3).

C.2.3 *Formulaires compacts pour les contraintes d'unité PDU*

C.2.3.1 *Introduction*

Lorsqu'une contrainte ne comporte que quelques champs ou que le nombre de contraintes est peu élevé, elles peuvent être présentées dans la version compacte du formulaire de contraintes d'unité PDU.

| **Déclarations de contraintes d'unité PDU** | | | | | |
|---|---|---|---|---|---|
| **Type d'unité PDU:** *PDU_Identifier* | | | | | |
| **Nom de contrainte** | **Chemin de dérivation** | **Nom de champ** | | | **Commentaires** |
| | | *ASP_ParIdentifier*$_1$ | | *ASP_ParIdentifier*$_n$ | |
| *ConsId-&ParList*$_1$ | *Derivation-Path*$_1$ | *ConstraintValue-&Attributes*$_{1,1}$ | | *ConstraintValue-&Attributes*$_{1,n}$ | *[FreeText]*$_1$ |
| *ConsId-&ParList*$_2$ | *Derivation-Path*$_2$ | *ConstraintValue-&Attributes*$_{2,1}$ | | *ConstraintValue-&Attributes*$_{2,n}$ | *[FreeText]*$_2$ |
| ⋮ | | ⋮ | | ⋮ | ⋮ |
| *ConsId-&ParList*$_m$ | *Derivation-Path*$_m$ | *ConstraintValue-&Attributes*$_{m,1}$ | | *ConstraintValue-&Attributes*$_{m,n}$ | *[FreeText]*$_m$ |

**Formulaire C-2 – Déclarations de contraintes d'unité PDU – Forme compacte**

Dans le formulaire compact de contraintes, les colonnes représentent les champs, et les lignes représentent les différentes instances de contraintes d'unité PDU. S'il y a *n* champs dans la définition de type d'unité PDU, il y aura *n* colonnes champ dans le formulaire compact des contraintes.

La colonne de chemin de dérivation est facultative: elle sera utilisée pour spécifier le chemin de dérivation des contraintes modifiées (voir le § 12.6). Une table compacte peut regrouper plusieurs contraintes de base (comme l'illustre l'exemple C.1) ou une contrainte de base et ses contraintes modifiées (comme dans l'exemple C.2). Lorsque des contraintes modifiées sont déclarées dans une table compacte, les champs non modifiés des contraintes modifiées sont représentés par des cases laissées en blanc à l'intersection de la ligne de la contrainte modifiée et de la colonne champ. Lorsqu'on réexprime une table compacte en notation TTCN.MP (c'est-à-dire en format simple), les champs vides (et donc hérités) seront omis. Les champs non spécifiés des contraintes modifiées sont laissés vides.

*Exemple C.1* – Contraintes utilisant le formulaire compact des contraintes

**C.1.1** Etant donné la déclaration de l'unité PDU_B

| **Définition de Type d'unité PDU** | | |
|---|---|---|
| **Nom d'unité PDU :** PDU_B<br>**Type de point PCO :** XSAP<br>**Commentaires :** | | |
| **Nom de champ** | **Type de champ** | **Commentaires** |
| FIELD1<br>FIELD2<br>FIELD3 | INTEGER<br>BOOLEAN<br>IA5String | |

**C.1.2** Les contraintes s'appliquant à l'unité PDU_B pourraient être représentées de la manière suivante sur le formulaire compact de contraintes:

| **Déclarations de contraintes d'unité PDU** | | | | |
|---|---|---|---|---|
| **Type d'unité PDU:** PDU_B | | | | |
| **Nom de contrainte** | **Nom de champ** | | | **Commentaires** |
| | FIELD1 | FIELD2 | FIELD3 | |
| CN1 | 3 | TRUE | «une chaîne» | |
| CN2 | (4, 5, 6) | FALSE | «une chaîne» | |
| CN3 | 0 | ? | – | |

La référence aux contraintes dans la partie dynamique pourrait comporter des entrées telles que PDU_B[CN1] et PDU_B[CN2].

*Exemple C.2* – Mécanisme d'héritage utilisant le formulaire compact des contraintes

| **Déclarations de contraintes d'unité PDU** | | | | | | |
|---|---|---|---|---|---|---|
| **Type d'unité PDU:** PDU_A | | | | | | |
| **Nom de contrainte** | **Non de dérivation** | **Nom de champ** | | | | **Commentaires** |
| | | FIELD1 | FIELD2 | FIELD3 | FIELD4 | |
| CN0 | | 0 | 'FF'H | '00'B | TRUE | |
| CN1 | CN0. | 1 | | | | |
| CN2 | CN0.CN | | – | ? | | |

C.2.3.2 *Contraintes compactes paramétrées*

Les contraintes (en représentation) compactes peuvent aussi être paramétrées. En ce cas, la liste de paramètres sera ajoutée au nom de la contrainte et figurera dans la colonne du nom de contrainte des formulaires compacts des contraintes.

*Exemple C.3* – Contrainte compacte paramétrée

L'appel des contraintes de l'unité PDU_X dans un module de test peut s'effectuer comme suit: S1, S2, S3, S4, S5(0), S5(1) ou S5(Var) où Var est une variable de test élémentaire ou de suite de tests.

| **Déclarations de contraintes d'unité PDU** | | | |
|---|---|---|---|
| **Type d'unité PDU:** PDU_X | | | |
| **Nom de contrainte** | **Nom de champ** | | **Commentaires** |
| | P1 | P2 | |
| S1 | 0 | 0 | |
| S2 | 0 | 1 | |
| S3 | 1 | 0 | |
| S4 | 1 | 1 | |
| S5(A:INTEGER) | 1 | A | |

C.2.4 *Formulaires compacts utilisés pour les contraintes de type structuré*

Pour les contraintes compactes de type structuré, on utilisera le formulaire suivant:

| **Déclarations de contraintes de type structuré** | | | | | |
|---|---|---|---|---|---|
| **Type de structure:** *StructIdentifier* | | | | | |
| **Nom de contrainte** | **Chemin de dérivation** | **Nom dechamp** | | | **Commentaires** |
| | | *ASP_ParIdentifier*$_1$ | | *ASP_ParIdentifier*$_n$ | |
| *ConsId-&ParList*$_1$ | *Derivation-Path*$_1$ | *ConstraintValue-&Attributes*$_{1,1}$ | | *ConstraintValue-&Attributes*$_{1,n}$ | *[FreeText]*$_1$ |
| *ConsId-&ParList*$_2$ | *Derivation-Path*$_2$ | *ConstraintValue-&Attributes*$_{2,1}$ | | *ConstraintValue-&Attributes*$_{2,n}$ | *[FreeText]*$_2$ |
| ⋮ | | ⋮ | | ⋮ | ⋮ |
| *ConsId-&ParList*$_m$ | *Derivation-Path*$_m$ | *ConstraintValue-&Attributes*$_{m,1}$ | | *ConstraintValue-&Attributes*$_{m,n}$ | *[FreeText]*$_m$ |

**Formulaire C-3 – Déclarations de contraintes de type structuré (version compacte)**

*Exemple C.4* – Utilisation des contraintes compactes de type structuré

L'unité PDU_Y comporte cinq champs: Y1 à Y5. Les champs Y1, Y2 et Y3 ont été combinés en un type structuré A. Dans les tables suivantes, la première montre les contraintes définies sur l'unité PDU_Y. Les deuxième et troisième tables renferment les mêmes informations que la dernière table.

Les deuxième et troisième tables montrent la spécification des contraintes du type structuré A à l'aide de formulaires de contrainte simple, tandis que la dernière table montre les contraintes du type structuré A au moyen du formulaire compact des contraintes. Dans les deux cas, le mécanisme de modification est utilisé.

Dans les tables suivantes, on voit que l'application de la contrainte YY1 impose aux champs Y1 à Y5 respectivement les valeurs 0,0,0,0,1, les valeurs des champs Y1 à Y3 étant calculées à partir du type structuré A en utilisant la contrainte A1. Si la contrainte YY2 est appliquée, les valeurs des champs Y1 à Y5 seront respectivement 0,3,0,1,0, les valeurs des champs Y1 à Y3 étant calculées à partir du type structuré A en utilisant la contrainte A2.

**C.4.1** Table de contraintes d'unité PDU utilisant un type structuré (appelé A)

| **Déclarations de contraintes d'unité PDU** | | | | |
|---|---|---|---|---|
| **Type d'unité PDU:** PDU_Y | | | | |
| **Nom de contrainte** | **Nom de champ** | | | **Commentaires** |
| | A | Y4 | Y5 | |
| YY1 | A1 | 0 | 1 | |
| YY2 | A2 | 1 | 0 | |
| YY3 | A2 | 0 | 1 | |

**C.4.2** A1 est une contrainte de base du type structuré A

| **Déclaration de contrainte de type structuré** | | |
|---|---|---|
| **Nom de contrainte :** A1<br>**Type de structure :** A<br>**Chemin de dérivation :**<br>**Commentaire :** | | |
| **Nom de l'élément** | **Valeur de l'élément** | **Commentaires** |
| Y1<br>Y2<br>Y3 | 0<br>0<br>0 | |

**C.4.3** La contrainte A2 de type structuré est une contrainte modifiée dérivée de A1

| **Déclaration de contrainte de type structuré** | | |
|---|---|---|
| **Nom de contrainte :** A2<br>**Type de structure :** A<br>**Chemin de dérivation :** A1.<br>**Commentaire :** | | |
| **Nom d'élément** | **Valeur d'élément** | **Commentaires** |
| Y2 | 3 | |

**C.4.4** Contraintes A1 et A2 du type structuré A sous forme compacte

| **Déclarations de contraintes de type structuré** | | | | | |
|---|---|---|---|---|---|
| **Nom du type de structure:** A | | | | | |
| **Nom de contrainte** | **Chemin de dérivation** | **Nom d'élément** | | | **Commentaires** |
| | | Y1 | Y2 | Y3 | |
| A1 | | 0 | 0 | 0 | |
| A2 | A1. | | 3 | | |

Lorsqu'on utilise les types structurés dans les déclarations de contraintes d'unité PDU, chaque nom de champ utilisé dans la définition du type structuré correspondra exactement au nom (ou nom abrégé, si le nom abrégé et le nom complet ont tous deux été définis) du champ d'unité PDU correspondant tel qu'il apparaît dans la définition originale du type d'unité PDU.

C.2.5 *Formulaires compacts pour les contraintes ASN.1*

Les formulaires suivants seront utilisés pour les définitions sous forme compacte des contraintes de primitive ASP ASN.1, d'unité PDU ASN.1 et de type ASN.1:

| **Déclarations de contraintes de primitive ASP en notation ASN.1** | |
|---|---|
| **Type de primitive ASP:** *ASP_Identifier* | |
| **Nom de contrainte** | **Valeur ASN.1** |
| *ConsId&ParList*$_1$ | *ConstraintValue&Attribute*$_1$ |
| ⋮ | ⋮ |
| *ConsId&ParList*$_m$ | *ConstraintValue&Attribute*$_m$ |

**Formulaire C-4 – Déclarations de contraintes de primitive ASP en notation ASN.1 – Forme compacte**

| **Déclarations de contraintes d'unité PDU en notation ASN.1** | |
|---|---|
| **Type d'unité PDU:** *PDU_Identifier* | |
| **Nom de contrainte** | **Valeur ASN.1** |
| *ConsId&ParList*$_1$ | *ConstraintValue&Attributes*$_1$ |
| ⋮ | ⋮ |
| *ConsId&ParList*$_m$ | *ConstraintValue&Attributes*$_m$ |

**Formulaire C-5 – Déclarations de contraintes d'unité PDU en notation ASN.1 – Forme compacte**

| **Déclarations de contraintes de type en notation ASN.1** | |
|---|---|
| **Nom du Type:** *ASN1_TypeIdentifier* | |
| **Nom de contrainte** | **Valeur ASN.1** |
| *ConsId&ParList*$_1$ | *ConstraintValue&Attributes*$_1$ |
| ⋮ | ⋮ |
| *ConsId&ParList*$_m$ | *ConstraintValue&Attributes*$_m$ |

**Formulaire C-6 – Déclarations de contraintes de type en notation ASN.1 – Forme compacte**

## C.3 *Formulaire compact pour les tests élémentaires*

### C.3.1 *Spécifications*

Le regroupement de plusieurs tables simples de comportement dynamique de test élémentaire en une seule table compacte de comportements dynamiques de tests élémentaires n'est autorisé que lorsque les règles suivantes s'appliquent:

a) toutes les tables simples de comportement dynamique de test élémentaire appartiennent au même groupe de tests;

b) toutes les tables simples de comportement dynamique de test élémentaire ont le même arbre comportemental par défaut ou n'en possèdent pas du tout; il est recommandé de ne pas en avoir;

c) la description comportementale de chaque table simple de comportement dynamique de test élémentaire comporte une simple construction de rattachement ATTACH.

### C.3.2 *Formulaire compact pour les comportements dynamiques de tests élémentaires*

Lorsqu'une série de tests élémentaires ont essentiellement le même comportement dynamique, les différences n'apparaissant que dans les contraintes en référence (par exemple, les tests utilisés de variation des paramètres des primitives ASP ou des unités PDU), ils peuvent être représentés dans la version compacte du formulaire des comportements dynamiques de tests élémentaires.

| **Comportements dynamiques de tests élémentaires** | | | |
|---|---|---|---|
| **Groupe :** *TestGroupReference*<br>**Comportement par défaut :** *DefaultReference* | | | |
| **Nom du test élémentaire** | **Objet** | **Rattachement de module de test** | **Commentaires** |
| .<br>.<br>*TestCaseIdentifier*<br>.<br>. | .<br>.<br>*FreeText*<br>.<br>. | .<br>.<br>*Attach*<br>.<br>. | .<br>.<br>*[FreeText]*<br>.<br>. |

**Formulaire C-7 – Comportements dynamiques de tests élémentaires (version compacte)**

Chaque ligne du formulaire décrit un test élémentaire simple. La table de ce formulaire remplace à elle seule une série de tables de comportement dynamique de test élémentaire dans la partie comportementale de la suite de tests.

La dernière colonne comporte des commentaires concernant chacun des tests élémentaires en ce qui concerne leur rattachement.

Les tests élémentaires figurant dans le formulaire compact des tests élémentaires peuvent former un sous-ensemble de leur groupe et doivent apparaître dans l'ordre indiqué dans l'index des tests élémentaires.

*Exemple C.5* – Table compacte des tests élémentaires définissant une série de tests pour le service de transfert, accès et gestion de fichiers (FTAM)

| **Comportements dynamiques de tests élémentaires** | | |
|---|---|---|
| **Groupe** : R/BV/PV/LM/CR/OV<br>**Comportement par défaut :** | | |
| **Nom du test élémentaire** | **Objet** | **Rattachement de module de test** |
| OVERIDE1 | Omettre le paramètre substitution quand le fichier existe | +OVERRIDE (FCRERQ_001,FCRERP_001) |
| OVERIDE2 | Omettre le paramètre substitution quand le fichier n'existe pas | +OVERRIDE (FCRERQ_002,FCRERP_002) |

ANNEXE D

(à la Recommandation X.292)

**Exemples**

(Cette annexe ne fait pas partie intégrante de la présente Recommandation)

D.1 *Exemples de contraintes sous forme tabulaire*

D.1.1 *Définitions de primitive ASP et d'unité PDU*

D.1.1.1 *Définition de type plat*

| **Définition de type d'unité PDU** | | |
|---|---|---|
| **Nom d'unité PDU** : T_CONNECT1<br>**Type du point PCO :**<br>**Commentaire** : Illustration des mécanismes TTCN | | |
| **Nom du champ** | **Nom du type** | **Commentaires** |
| Source | BITSTRING [4] | Longueur de 4 bits |
| Destination | BITSTRING [4] | Longueur de 4 bits |
| T_Class | INTEGER0to4 | Défini comme type simple |
| UserData | IA5String | |

D.1.1.2 *Définition de type structuré*

| **Définition de type d'unité PDU** | | |
|---|---|---|
| **Nom d'unité PDU :** T_CONNECT2<br>**Type du point PCO :**<br>**Commentaire :** Illustration des mécanismes TTCN | | |
| **Nom du champ** | **Nom du type** | **Commentaires** |
| T_Addresses | T_AddressInfo | |
| T_Class | INTEGER0to4 | Défini comme type simple |
| UserData | IA5String | |

| **Définition de type structuré** | | |
|---|---|---|
| **Nom du type :** T_AddressInfo<br>**Commentaire :** Peut être utilisé dans tous les exemples d'unité PDU de transport | | |
| **Nom d'élément** | **Définition du type** | **Commentaires** |
| Source | BITSTRING [4] | Longueur de 4 bits |
| Destination | BITSTRING [4] | Longueur de 4 bits |

D.1.1.3 *Type spécial d'unité PDU, pour permettre l'utilisation du chaînage (statique) des contraintes*

| **Définition de type primitive ASP** | | |
|---|---|---|
| **Nom de primitive ASP :** N_DATArequest<br>**Type de point PCO :** N_SAP<br>**Commentaire :** Pour illustration seulement | | |
| **Nom de paramètre** | **Type de paramètre** | **Commentaires** |
| CallingNetworkAddress | HEXSTRING | |
| CalledNetworkAddress | HEXSTRING | |
| ConnectionIdentifier | HEXSTRING | |
| Data | PDU | Pour permettre le chaînage des contraintes |

D.1.2 *Contraintes de primitive ASP/d'unité PDU*

D.1.2.1 Contraintes de type plat

| **Déclaration de contrainte d'unité PDU** | | |
|---|---|---|
| **Nom de contrainte :** TCON_Class4_1<br>**Type d'unité PDU :** T_CONNECT1<br>**Chemin de dérivation :**<br>**Commentaire :** | | |
| **Nom de champ** | **Valeur du champ** | **Commentaires** |
| Source | TS_Par1 | |
| Destination | TS_Par2 | |
| T_Class | 4 | |
| UserData | "testing, testing" | |

D.1.2.2 *Contraintes de type structuré, faisant référence aux groupes de champs*

| **Déclaration de contrainte d'unité PDU** | | |
|---|---|---|
| **Nom de contrainte :** TCON_Class4_2<br>**Type d'unité PDU :** T_CONNECT2<br>**Chemin de dérivation :**<br>**Commentaire :** | | |
| **Nom de champ** | **Valeur du champ** | **Commentaires** |
| T_Addresses | WrongAddress | WrongAddress (fausse adresse) est une référence à une contrainte de type structuré |
| T_Class | 4 | |
| UserData | "one, two, three" | |

| **Déclaration de contrainte de type structuré** | | |
|---|---|---|
| **Nom de contrainte :** WrongAddress<br>**Type de structure :** T_AddressInfo<br>**Chemin de dérivation :**<br>**Commentaire :** | | |
| **Nom d'élément** | **Valeur d'élément** | **Commentaires** |
| Source | TS_Par1 | |
| Destination | ’0000’B | |

D.1.2.3 *Chaînage, utile pour les unités PDU (imbriquées) dans des primitives ASP*

| **Déclaration de contrainte de primitive ASP** | | |
|---|---|---|
| **Nom de contrainte :** N_DATAreq_With_T_CON_Class4_1<br>**Type de primitive ASP :** N_DATArequest<br>**Chemin de dérivation :**<br>**Commentaire :** TCON_Class4_1 est une contrainte d'unité PDU (c'est-à-dire un chaînage) | | |
| **Nom de paramètre** | **Valeur du paramètre** | **Commentaires** |
| CallingNetworkAddress | TS_Par3 | |
| CalledNetworkAddress | TS_Par4 | |
| ConnectionIdentifier | 'ABCDEF'H | |
| Data | TCON_Class4-1 | |

D.1.2.4 *Contraintes paramétrées*

Il est possible de paramétrer des contraintes de types plat, structuré et chaîné. L'exemple suivant montre le paramétrage utilisé pour transférer une valeur:

| **Déclaration de contrainte d'unité PDU** | | |
|---|---|---|
| **Nom de contrainte :** TCON_1(class:INTEGER)<br>**Type d'unité PDU :** T_CONNECT1<br>**Chemin de dérivation :**<br>**Commentaire :** | | |
| **Nom de champ** | **Valeur du champ** | **Commentaires** |
| Source | ?1000'B | |
| Destination | ? | |
| T_Class | class | class(classe) est un paramètre formel |
| UserData | ? | |

On peut faire référence à cette contrainte paramétrée depuis des tables de test élémentaire, de module de test ou de comportement par défaut, par exemple sous forme de:

TCON_1(4) ou TCON_1(TCvariable)

Les valeurs de champ peuvent être des unités PDU (chaînées) entières.

| **Déclaration de contrainte de primitive ASP** | | |
|---|---|---|
| **Nom de contrainte :** N_DATAreq_With_T_CON(A_Constraint:T_CONNECT2)<br>**Type de primitive ASP :** N_DATArequest<br>**Chemin de dérivation :**<br>**Commentaire :** TCON_Class4_1 est une contrainte d'unité PDU (à savoir, chaînage) | | |
| **Nom de paramètre** | **Valeur du paramètre** | **Commentaires** |
| CallingNetworkAddress | TS_Par3 | |
| CalledNetworkAddress | TS_Par4 | |
| ConnectionIdentifier | '1234567'H | |
| Data | A_Constraint | A_Constraint est un paramètre formel |

Cette contrainte peut être appelée, par exemple de la manière suivante:

N_DATAreq_With_TCON(TCON_Class4_2)

Comme le paramètre effectif est un nom de contrainte lui-même paramétrable, il est possible ainsi de représenter une profondeur arbitraire d'imbrication des unités PDU.

D.1.2.5 *Contraintes modifiées*

Il est possible d'utiliser des contraintes existantes et de les modifier pour en définir de nouvelles. Cela est vrai pour les contraintes de tous types: plat, structuré et paramétré.

| **Déclaration de contrainte d'unité PDU** | | |
|---|---|---|
| **Nom de contrainte** : TCON_Class0_1<br>**Type d'unité PDU** : T_CONNECT1<br>**Chemin de dérivation** : TCON_Class4_1<br>**Commentaire** : La classe 0 est acceptable | | |
| **Nom de champ** | **Valeur de champ** | **Commentaires** |
| T_Class | 0 | |

Des caractères génériques peuvent être utilisés pour les valeurs du type suivant.

| **Déclaration de contrainte d'unité PDU** | | |
|---|---|---|
| **Nom de contrainte** : TCON_AnyClass<br>**Type d'unité PDU** : T_CONNECT1<br>**Chemin de dérivation** : TCON_Class4_1<br>**Commentaire** : Toute classe (0..4) est acceptable | | |
| **Nom de champ** | **Valeur du champ** | **Commentaires** |
| T_Class | ? | |

Cette méthode n'est toutefois pas recommandée. Il est préférable d'utiliser la contrainte plus générale comme base.

Il est aussi possible de supprimer des champs entiers.

| **Déclaration de contrainte d'unité PDU** | | |
|---|---|---|
| **Nom de contrainte** : TCON_Erroneous_NoClass<br>**Type d'unité PDU** : T_CONNECT1<br>**Chemin de dérivation** : TCON_Class4_1<br>**Commentaire** : Aucune classe présente | | |
| **Nom de champ** | **Valeur du champ** | **Commentaires** |
| T_Class | – | T_Class omis |

D.2 *Exemples de contraintes ASN.1*

D.2.1 *Définitions de primitive ASP et d'unité PDU*

D.2.1.1 *Type plat*

| **Définition de type unité PDU en notation ASN.1** |
|---|
| **Nom d'unité PDU :** T_CONNECT1<br>**Type de point PCO :**<br>**Commentaire :** |
| **Définition de type** |
| -- à la seule fin d'illustrer l'utilisation de la notation ASN.1 en notation TTCN<br><br>SEQUENCE { source BITSTRING [SIZE(4..4)],<br>destination BITSTRING [SIZE(4..4)],<br>t_Class INTEGER(0..4),<br>userData IA5String OPTIONAL<br>} |

D.2.1.2 *Type structuré*

| **Définition de type unité PDU en notation ASN.1** |
|---|
| **Nom d'unité PDU :** T_CONNECT2<br>**Type de point PCO :**<br>**Commentaire :** |
| **Définition de type** |
| -- à la seule fin d'illustrer l'utilisation de la notation ASN.1 en notation TTCN<br><br>SEQUENCE { t_Addresses T_AddressInfo,<br>t_Class INTEGER (0..4),<br>userData IA5String<br>}<br><br>-- on peut trouver un développement de T_AddressInfo dans une table séparée |

Les productions connexes en notation ASN.1 qui sont normalement dans un seul module ASN.1 peuvent être réparties entre plusieurs tables TTCN.

| **Définition de type** |
|---|
| **Nom du type :** T_AddressInfo<br>**Commentaire :** |
| **Définition du type** |
| SEQUENCE { source BITSTRING [SIZE(4..4)],<br>Destination BITSTRING [SIZE(4..4)],<br>} |

D.2.1.3 *Définition de primitive ASP*

| **Définition de type de primitive ASP en notation ASN.1** |
|---|
| **Nom de primitive ASP :** N_DATArequest<br>**Type de point PCO :** N_SAP<br>**Commentaire :** |
| **Définition de type** |
| SEQUENCE { callingNetworkAddress OCTETSTRING, -- nombre pair d'octets<br>calledNetworkAddress OCTETSTRING, -- nombre pair d'octets<br>connectionIdentifier OCTETSTRING, -- nombre pair d'octets<br>data T_PDUS<br>} |

| **Définition de type ASN.1** |
|---|
| **Nom du type :** T_PDUS<br>**Commentaire :** |
| **Définition de type** |
| CHOICE { t1 T_CONNECT1,<br>t2 T_CONNECT2<br>} |

D.2.2 *Contraintes de primitive ASP/d'unité PDU en notation ASN.1*

D.2.2.1 *Type plat*

| **Déclaration de contrainte d'unité PDU en notation ASN.1** |
|---|
| **Nom de contrainte :** TCON_Class4_1<br>**Type d'unité PDU :** T_CONNECT1<br>**Chemin de dérivation :**<br>**Commentaire :** |
| **Valeur de contrainte** |
| { source TS_PAR1,<br>ts_PAR2, -- l'identificateur de champ peut être omis<br>t_Class 4,<br>userData "testing, testing"<br>} |

D.2.2.2 *Type structuré*

| **Déclaration de contrainte d'unité PDU en notation ASN.1** |
|---|
| **Nom de contrainte** : TCON_Class4_2<br>**Type d'unité PDU** : T_CONNECT2<br>**Chemin de dérivation** :<br>**Commentaire** : |
| **Valeur de contrainte** |
| { t_Addresses WrongAddress, -- référence à une contrainte de champ d'unité PDU<br>t_Class 4,<br>userData "one, two, three"<br>} |

| **Déclaration de contrainte de type ASN.1** |
|---|
| **Nom de contrainte** : WrongAddress<br>**Type de structure** : T_AddressInfo<br>**Chemin de dérivation** :<br>**Commentaire** : |
| **Valeur de contrainte** |
| { source TS_PAR1,<br>destination '0000'B<br>} |

D.2.2.3 *Chaînage d'une contrainte d'unité PDU*

| **Déclaration de contrainte de primitive ASP en notation ASN.1** |
|---|
| **Nom de contrainte** : N_DATAreq_With_TCON_Class4_1<br>**Type de primitive ASP** : N_DATArequest<br>**Chemin de dérivation** :<br>**Commentaire** : |
| **Valeur de contrainte** |
| { callingNetworkAddress TS_PAR_3,<br>calledNetworkAddress TS_PAR_4,<br>connectionIdentifier 'ABCDEF'H,<br>data {t1 TCON_Class_4_1} -- chaînage vers une contrainte d'unité PDU<br>} |

D.2.2.4 *Contraintes paramétrées*

Les contraintes ASN.1 peuvent être paramétrées tout comme les contraintes tabulaires TTCN, par exemple:

| **Déclaration de contrainte d'unité PDU en notation ASN.1** |
|---|
| **Nom de contrainte** : TCON_1(Class:INTEGER)<br>**Type d'unité PDU** : T_CONNECT1<br>**Chemin de dérivation** :<br>**Commentaires** : |
| **Valeur de contrainte** |
| { source ’0000’B,<br>destination ? -- caractère générique<br>t_Class class, -- paramètre formel<br>userData ?<br>} |

On peut faire référence à cette contrainte paramétrée depuis les tables de test élémentaire, de module de test ou de comportement par défaut, par exemple sous la forme:

TCON_1(4) ou TCON_1(TCvariable)

Un paramètre peut aussi représenter une unité PDU chaînée complète.

| **Déclaration de contrainte de primitive ASP en notation ASN.1** |
|---|
| **Nom de contrainte** : N_DATAreq_With_TCON(a_constraint:T_CONNECT2)<br>**Type de primitive ASP** : T_DATArequest<br>**Chemin de dérivation** :<br>**Commentaire** : |
| **Valeur de contrainte** |
| { callingNetworkAddress TS_PAR_3,<br>calledNetworkAddress TS_PAR_4,<br>connectionIdentifier’1234567’H,<br>data {t2 a_constraint}<br><br>-- a_constraint est un paramètre formel contenant une unité PDU complète<br>} |

On peut faire référence à cette contrainte depuis les tables de test élémentaire, de module de test ou de comportement par défaut, par exemple sous la forme:

N_DATAreq_With_TCON(TCON_Class4_2)

Comme le paramètre effectif est un nom de contrainte lui-même paramétrable, il est possible de représenter une profondeur arbitraire d'imbrication.

D.2.2.5 *Contraintes modifiées*

De nouvelles contraintes peuvent être créées, en modifiant des contraintes déjà définies à l'aide du mécanisme REPLACE (remplacement).

| **Déclaration de contrainte d'unité PDU en notation ASN.1** |
|---|
| **Nom de contrainte** : TCON_Class0_1<br>**Type d'unité PDU** : T_CONNECT1<br>**Chemin de dérivation** : T_CON_Class4_1.<br>**Commentaire** : |
| **Valeur de contrainte** |
| REPLACE t_Class BY 0 |

Des caractères génériques peuvent aussi être utilisés comme suit.

| **Déclaration de contrainte d'unité PDU en notation ASN.1** |
|---|
| **Nom de contrainte** : TCON_AnyClass<br>**Type d'unité PDU** : T_CONNECT1<br>**Chemin de dérivation** : T_CON_Class4_1.<br>**Commentaire** : |
| **Valeur de contrainte** |
| REPLACE t_Class BY ? |

Pour spécifier les champs qui seront omis, on utilise le mécanisme OMIT (omission). Cela n'est permis que si le champ est déclaré OPTIONAL (facultatif).

| **Déclaration de contrainte d'unité PDU en notation ASN.1** |
|---|
| **Nom de contrainte** : TCON_NoUserData<br>**Type d'unité PDU** : T_CONNECT1<br>**Chemin de dérivation** : TCON_CLASS4_1.TCON_AnyClass.<br>**Commentaire** : |
| **Valeur de contrainte** |
| OMIT UserData |

Il est possible de modifier les contraintes paramétrées ASN.1, mais on notera que les champs paramétrés eux-mêmes ne peuvent pas être remplacés.

| **Déclaration de contrainte d'unité PDU en notation ASN.1** |
|---|
| **Nom de contrainte** : TCON_2(class:INTEGER)<br>**Type d'unité PDU** : T_CONNECT1<br>**Chemin de dérivation** : TCON_1.<br>**Commentaire** : |
| **Valeur de contrainte** |
| REPLACE userData BY "CPS" |

D.2.3 *Autres exemples de contraintes en notation ASN.1*

D.2.3.1 *Définition d'une unité PDU F-INITIALIZEresponse pour le service FTAM, à l'aide d'une table de définition de type unité PDU en notation ASN.1*

| **Définition de type unité PDU en notation ASN.1** | |
|---|---|
| **Nom d'unité PDU** : F_INITIALIZEresponse<br>**Type de point PCO** :<br>**Commentaire** : | |
| **Définition de type** | |
| SEQUENCE { | |
| state_result | State_Result DEFAULT success, |
| action_result | Action_Result DEFAULT success, |
| protocol_version | Protocol_Version DEFAULT { version_1}, |
| implementation_information | Implementation_Information OPTIONAL, |
| presentation_context_management | [2] IMPLICIT BOOLEAN DEFAULT FALSE, |
| service_class | Service_Class DEFAULT { transfer_class}, |
| functional_units | Functional_Units, |
| attribute_groups | Attribute_Groups DEFAULT { }, |
| shared_ASE_information | Shared_ASE_Information OPTIONAL, |
| ftam_quality_of_service | FTAM_Quality_Of_Service, |
| contents_type_list | Contents_Type_List OPTIONAL, |
| diagnostic | Diagnostic OPTIONAL, |
| checkpoint_window | [8] IMPLICIT INTEGER DEFAULT 1 |
| } | |

Les champs de l'unité PDU (State_Result, Action_Result, etc.) sont déclarés dans les définitions de type ASN.1

Le type Functional_Units est par exemple déclaré comme ceci.

| **Définition de type en notation ASN.1** |
|---|
| **Nom du type :** Functional_Units<br>**Commentaire :** |
| **Définition de type** |
| [4] IMPLICIT BITSTRING<br>{ read (2),<br>write (3),<br>file_access (4)<br>limited_file_management (5),<br>enhanced_file_management (6),<br>grouping (7),<br>fadu_locking (8)<br>recovery (9),<br>restart_data_transfer (10)<br>} |

Une contrainte de base, F_INITrsp_001, s'appliquant à une réponse F_INITIALIZEresponse, est déclarée dans la partie contraintes.

| **Déclaration de contrainte d'unité PDU en notation ASN.1** | |
|---|---|
| **Nom de contrainte :** F_INITrsp_001<br>**Type d'unité PDU :** F_INITIALIZEresponse<br>**Chemin de dérivation :**<br>**Commentaire :** | |
| **Valeur de contrainte** | |
| { | |
| state_result | State_Result_001, |
| action_result | Action_Result_001, |
| protocol_version | Protocol_Version_001, |
| implementation_information | Implementation_Information_001, |
| presentation_context_management | FALSE, |
| service_class | Service_Class_001, |
| functional_units | Functional_Units_001, |
| attribute_groups | Attribute_Groups_0 |
| shared_ASE_information | Shared_ASE_Information_001 |
| ftam_quality_of_service | FTAM_Quality_Of_Service_001, |
| contents_type_list | Contents_Type_List_001, |
| diagnostic | Diagnostic_001, |
| checkpoint_window | 1 |
| } | |

Une contrainte Functional_Units_001, appliquée au type Functional_Units, est déclarée dans une déclaration de contrainte de champ d'unité PDU en notation ASN.1:

| **Déclaration de contrainte de type ASN.1** |
|---|
| **Nom de contrainte** : Functional_Units_001<br>**Type de structure** : Functional_Units<br>**Chemin de dérivation** :<br>**Commentaire** : |
| **Valeur de contrainte** |
| '001'B -- en écriture seulement |

Une deuxième contrainte, F_INITrsp_002, peut être créée en modifiant la contrainte de base, F_INIT_rsp001:

| **Déclaration de contrainte d'unité PDU en notation ASN.1** | | |
|---|---|---|
| **Nom de contrainte** : F_INITrsp_002<br>**Type d'unité PDU** : F_INITIALIZEresponse<br>**Chemin de dérivation** : F_INITrsp_001.<br>**Commentaires** : | | |
| **Valeur de contrainte** | | |
| OMIT | implementation_information, | |
| REPLACE | presentation_context_management | BY TRUE, |
| REPLACE | functional_units | BY Functional_Units_002, |
| REPLACE | checkpoint_window | BY ? |

où Functional_Units_002 est une déclaration de contrainte d'unité PDU en notation ASN.1.

D.3 *Contraintes de base et contraintes modifiées*

Supposons que nous ayons la définition de type unité PDU suivante:

| **Définition de type unité PDU** | | |
|---|---|---|
| **Nom d'unité PDU** : PDU_B<br>**Type de point PCO** :<br>**Commentaire** : Ceci est la déclaration de l'unité de données de protocole PDU-B | | |
| **Nom de champ** | **Type de champ** | **Commentaires** |
| FIELD1 | INTEGER | |
| FIELD2 | HEXSTRING | |
| FIELD3 | BITSTRING | |
| FIELD4 | BOOLEAN | |

Une contrainte de base pour PDU_B pourrait être

| Déclaration de contrainte d'unité PDU | | |
|---|---|---|
| **Nom de contrainte :** C0<br>**Type d'unité PDU :** PDU_B<br>**Chemin de dérivation :**<br>**Commentaires :** | | |
| **Nom de champ** | **Valeur du champ** | **Commentaires** |
| FIELD1<br>FIELD2<br>FIELD3<br>FIELD4 | 0<br>'FF'H<br>'00'B<br>TRUE | |

Une contrainte modifiée C1 dérivée de la contrainte de base C0 pourrait être

| Déclaration de contrainte d'unité PDU | | |
|---|---|---|
| **Nom de contrainte :** C1<br>**Type d'unité PDU :** PDU_B<br>**Chemin de dérivation :** C0.<br>**Commentaire :** | | |
| **Nom de champ** | **Valeur du champ** | **Commentaires** |
| FIELD1 | 1 | Dans la contrainte de base C0, cette valeur de champ est 0 |

Il est possible de poursuivre le processus de dérivation à partir de C1:

| Déclaration de contrainte d'unité PDU | | |
|---|---|---|
| **Nom de contrainte :** C2<br>**Type d'unité PDU :** PDU_B<br>**Chemin de dérivation :** C0.C1.<br>**Commentaire :** | | |
| **Nom de champ** | **Valeur du champ** | **Commentaires** |
| FIELD2<br>FIELD3 | –<br>? | Ce champ est omis<br>Toute valeur licite est acceptée |

On fait référence à une contrainte modifiée dans un arbre de comportement par son nom.

D.4 *Définitions de type utilisant des macroinstructions*

D.4.1 *Définition de type unité PDU à l'aide d'un symbole de macroinstruction*

| **Définition de type unité PDU** | | |
|---|---|---|
| **Nom d'unité PDU :** T_CONNECT3<br>**Type de point PCO :**<br>**Commentaire :** Illustration du mécanisme des macromécanismes TTCN | | |
| **Nom de champ** | **Type de champ** | **Commentaires** |
| <– | T_AddressGroup | |
| T_Class | INTEGER0to4 | Défini comme type simple |
| UserData | IA5String | |

| **Définition de type structuré** | | |
|---|---|---|
| **Nom du type :** T_AddressGroup<br>**Commentaire :** | | |
| **Nom d'élément** | **Définition du type** | **Commentaires** |
| Source | BITSTRING[4] | La longueur est de 4 bits |
| Destination | BITSTRING[4] | La longueur est de 4 bits |

| **Déclaration de contrainte d'unité PDU** | | |
|---|---|---|
| **Nom de contrainte :** TCON_Class4_3<br>**Type d'unité PDU :** T_CONNECT3<br>**Chemin de dérivation :**<br>**Commentaire :** | | |
| **Nom de champ** | **Valeur du champ** | **Commentaires** |
| <– | GoodAddress | Référence à la déclaration de contrainte de type structuré |
| T_Class | 4 | |
| UserData | "one, two, three" | |

| **Déclaration de contrainte de type structuré** | | |
|---|---|---|
| **Nom de contrainte :** GoodAddress<br>**Type de structure :** T_AddressGroup<br>**Chemin de dérivation :**<br>**Commentaire :** | | |
| **Nom d'élément** | **Valeur de l'élément** | **Commentaires** |
| Source | ’0101’B | |
| Destination | ’1111’B | |

D.5 *Utilisation de (REPEAT) (répétition)*

| **Comportement dynamique de test élémentaire** | | | | | |
|---|---|---|---|---|---|
| **Nom du test élémentaire :** RPT_EX2<br>**Groupe :** TTCN_EXAMPLES/REPEAT_EXAMPLE2/<br>**Objectif :** Illustrer l'utilisation de REPEAT<br>**Comportement par défaut :**<br>**Commentaires :** | | | | | |
| **N°** | **Etiquette** | **Description comportementale** | **Réf. des contraintes** | **Verdict** | **Commentaires** |
| 1<br>2<br>3<br><br>4<br>5<br>6<br>7<br><br><br><br>8<br>9 | | (FLAG:=FALSE, COUNTER:=0)<br>!A<br>REPEAT STEP2 (FLAG, COUNTER)<br>UNTIL [FLAG OR COUNTER=3]<br>[FLAG]<br>!D<br>[COUNTER=3]<br>!E<br><br>STEP2 (F:BOOLEAN; NUMBER:INTEGER)<br>?B (F:=TRUE)<br>?C (F:=FALSE, NUMBER:=NUMBER+1) | <br>A1<br><br><br><br>D1<br><br>E1<br><br><br><br>B1<br>C1 | <br><br><br><br>PASS<br><br>FAIL | |
| **Commentaires détaillés:** Cet exemple illustre l'exécution répétée de STEP2 (module 2), peut se terminer par la réception du message B ou de trois autres messages. Dans les lignes suivant la construction REPEAT, les expressions booléennes indiquent qu'il faut envoyer le message E si le message B est reçu et le message F si trois messages différents de B sont reçus. | | | | | |

D.6 *Opérations de suite de tests*

Utilisation d'une opération de suite de tests pour effectuer un total de contrôle:

| **Définition d'opération de suite de tests** |
|---|
| **Nom de l'opération :** CRC(P:A_PDU)<br>**Type de résultat :** INTEGER<br>**Commentaire :** |
| **Description** |
| Calculer et vérifier par total de contrôle l'unité PDU P à l'aide de l'algorithme de contrôle de redondance cyclique (CRC) |

*Remarque* – Dans une suite ATS réelle, cette opération serait décrite de manière plus détaillée.

| Déclaration de contrainte d'unité PDU | | |
|---|---|---|
| **Nom de contrainte** : CONS1<br>**Type d'unité PDU** : A_PDU<br>**Chemin de dérivation** :<br>**Commentaire** : | | |
| **Définition du type** | **Valeur du champ** | **Commentaires** |
| .<br>.<br>Checksum<br>.<br>. | .<br>.<br>?<br>.<br>. | |
| L'affectation A_PDU.Checksum := CRC(CONS1) dans l'événement SEND approprié d'une description comportementale effectuera le total de contrôle dans la contrainte CONS1. | | |

D.7 *Exemple d'une description générale de suite de tests*

La table de structure de suite de tests ci-dessous définit une hiérarchie des groupes de tests et des tests élémentaires dans la suite considérée. Dans cette structure, des expressions de sélection de test sont identifiées qui régissent la sélection des groupes de tests et des tests élémentaires pour exécution. Par exemple, SELEXP_100 est référencé comme expression de contrôle pour la caractéristique X du protocole. Si la caractéristique X n'est pas prise en charge, aucun des tests élémentaires de la suite qui se trouvent dans le groupe de cette caractéristique ne sera sélectionné.

| Structure de la suite de tests | | | |
|---|---|---|---|
| **Nom de la suite** : TEST_SUITE_A<br>**Référence normalisée** : ISO/CEI xxxx<br>**Référence de déclaration PICS** : ISO/CEI aaaa<br>**Référence d'information PIXIT** : ISO/CEI bbbb<br>**Notation(s) des tests** : Méthode de test DS<br>**Commentaire** : Ceci n'est qu'un exemple | | | |
| **Référence de groupe de test** | **Référence de sélection** | **Objectif du groupe de test** | **N° de Page** |
| FEATURE_X | SELEXP_100 | Caractéristique X facultative dans le test | 50 |
| FEATURE_X/ATTR_A | | Attribut A obligatoire dans le test | 50 |
| FEATURE_X/ATTR_A/NEGOTIATION | SELEXP_101 | Négociation de l'attribut A facultative dans le test | 50 |
| FEATURE_X/ATTR_A/USAGE | | Usage de l'attribut A dans le test | 60 |
| FEATURE_X/ATTR_B | | Caractéristique Y obligatoire dans le test | 80 |

Pour déterminer si la caractéristique X est prise en charge ou non, il faut évaluer SELEXP_100. Pour cela, on détermine si le paramètre de suite de tests dans SELEXP_100, c'est-à-dire TST_FX, a la valeur TRUE. Si tel est le cas, le traitement à l'intérieur du groupe continue. On notera que les tests concernant l'attribut A seront sélectionnés (aucune expression), mais que ceux qui concernent la caractéristique facultative de négociation de cet attribut ne le seront que si SELEXP_101 a la valeur TRUE.

| Index de tests élémentaires | | | | |
|---|---|---|---|---|
| **Référence de groupe de test** | **Identificateur des tests élémentaires** | **Référence de sélection** | **Description** | **N° de page** |
| FEATURE_X/ATTR_A/NEGOTIATION | FX_ANEG_1 | SELEXP_102 | Demande de l'attr. A, nég. valide | 50 |
| | FX_ANEG_2 | SELEXP_102 | Demande de l'attr. A, nég. non val. | 52 |
| | FX_ANEG_3 | | Réception de l'attr. A, nég. non val. | 54 |
| | FX_ANEG_4 | | Réception de l'attr. A, nég. non val. | 56 |
| FEATURE_X/ATTR_A/USAGE | FX_AUSE_1 | SELEXP_103 | Usage de l'attribut A (VAL=0) | 60 |
| | FX_AUSE_2 | | Réception de l'attribut A | 62 |
| | FX_AUSE_3 | | Réception de l'attribut A | 64 |

Si la négociation de l'attribut A est prise en charge, les tests élémentaires de FX_ANEG_01 à FX_ANEG_04 sont tous candidats pour la sélection. Toutefois, les tests élémentaires '01' et '02' ne seront choisis que si l'expression complémentaire de sélection SELEXP_102 a la valeur TRUE. Le test élémentaire FX_ANEG_01 ne sera sélectionné que si la déclaration PICS indique qu'une valeur zéro peut être prise en charge pour l'attribut A.

Les questions relatives à la déclaration PICS et aux informations PIXIT utilisées dans les expressions de sélection de test sont déclarées en tant que paramètres de suite de tests.

| Déclarations de paramètres de suite de tests | | | |
|---|---|---|---|
| **Nom de paramètre** | **Type** | **PICS/PIXIT** | **Commentaires** |
| TSP_FX | BOOLEAN | PICS question FX1 | Q: caractéristique X prise en charge? |
| TSP_FXA_N | BOOLEAN | PICS question FX2 | Q: nég. de caractéristique X prise en charge? |
| TSP_FXA_NINIT | BOOLEAN | PICS question FX3 | Q: nég. nécessaire à l'instance IUT? |
| TSP_FXA_MINVAL | INTEGER | PIXIT question FXVAL | Q: valeur VAL = 0 utilisée par instance IUT? |

Ces expressions de sélection de test sont déclarées en tant qu'expressions booléennes, comme défini au § 10.5.

| Définitions des expressions de sélection de test élémentaire | | |
|---|---|---|
| **Nom de l'expression** | **Expression de sélection** | **Commentaires** |
| SELEXP_100 | TSP_FX | Caractéristique X prise en charge |
| SELEXP_101 | TSP_FXA_N | Négociation de caractéristique X |
| SELEXP_102 | TSP_FXA_NINIT | Demande de négociation de caractéristique X |
| SELEXP_103 | TSP_FXA_VAL=0 | Caractéristique X accepte val = 0 |

D.8 *Exemple de test élémentaire présenté en notation TTCN.MP*

Pour l'échantillon de test élémentaire donné ci-dessous:

| **Comportement dynamique de test élémentaire** | | | | | |
|---|---|---|---|---|---|
| **Nom du test élémentaire :** PACKET/P4/PROPER/T_02<br>**Référence :** T_7_02<br>**Objectif :** Vérifier que l'instance IUT acquitte un code de cause de libération 05 quand elle est dans l'étape p4<br>**Comportement par défaut :**<br>**Commentaires :** | | | | | |
| **N°** | **Etiquette** | **Description de comportement** | **Réf. des contraintes** | **Verdict** | **Commentaires** |
| 0 | | +R1_PREAMBLE(SVC) | | | |
| 1 | | +P4D1_PREAMBLE | | | |
| 2 | | !CLEAR START TD | CLR_0(LC) | | Cause de libération=5 |
| 3 | L1 | ?CLEAR CANCEL TD | CLRC_0(LC) | PASS | |
| 4 | | +R1_POSTAMBLE | | | |
| 5 | | ?CLEAR CANCEL TD | CLR_L0(LC) | PASS | |
| 6 | | +R1_POSTAMBLE | | | |
| 7 | | ?RESTART [RST_ON_ERR] CANCEL TD | STRT_DTEA | PASS | |
| 8 | | !RESTARTC | STRTC | | |
| 9 | | +R1_POSTAMBLE | | | |
| 10 | | +D1C_UNEXPECTED | | | |
| 11 | | –> L1 | | | |
| 12 | | +RSRT_UNEXPECTED | | | |
| 13 | | ?TIMEOUT TD | | FAIL | |
| 14 | | ?OTHERWISE CANCEL TD | | FAIL | |

La description TTCN.MP correspondante est la suivante:

**$BeginTestCase**

**$TestCaseId** T_7_02

**$TestGroupRef** PACKET/P4/PROPERT/T_02

**$TestPurpose** /* Vérifier que l'instance IUT acquitte un code de cause de libération quand elle est dans l'état p4 */

**$DefaultsRef**

**$BehaviourDescription**

**$BehaviourLine**

**$Label**

**$Line** [0] +R1_PREAMBLE(SVC)

**$Cref**

**$Verdict**

**$End_BehaviourLine**

**$BehaviourLine**

**$Label**

**$Line** [1] +P4D1_PREAMBLE

**$Cref**

**$Verdict**

**$End_BehaviourLine**

**$BehaviourLine**

**$Label**

**$Line** [2] !CLEAR START TD

**$Cref** CLR_0(LC)

**$Verdict**

**$Comment** /* Cause de libération = 5 */

**$End_BehaviourLine**

**$BehaviourLine**

**$Label** L1

**$Line** [3] ?CLEARC CANCEL TD

**$Cref** CLRC_0(LC)

**$Verdict** (PASS)

**$End_BehaviourLine**

**$BehaviourLine**

**$Label**

**$Line** [4] +R1_POSTAMBLE

**$Cref**

**$Verdict**

**$End_BehaviourLine**

**$BehaviourLine**

**$Label**

**$Line** [3] ?CLEAR CANCEL TD

**$Cref** CLR_L0(LC)

**$Verdict** (PASS)

**$End_BehaviourLine**

**$BehaviourLine**

**$Label**

**$Line** [4] +R1_POSTAMBLE

**$Cref**

**$Verdict**

**$End_BehaviourLine**

**$BehaviourLine**

**$Label**

**$Line** [3] ?RESTART [RST_ON_ERR] CANCEL TD

**$Cref** STRT_DTEA

**$Verdict** (PASS)

**$End_BehaviourLine**

**$BehaviourLine**

**$Label**

**$Line** [4] !RESTARTC

**$Cref** STRTC

**$Verdict**

**$End_BehaviourLine**

**$BehaviourLine**

**$Label**

**$Line** [5] +R1_POSTAMBLE

**$Cref**

**$Verdict**

**$End_BehaviourLine**

**$BehaviourLine**

**$Label**

**$Line** [3] +D1C_UNEXPECTED

**$Cref**

**$Verdict**

**$End_BehaviourLine**

**$BehaviourLine**

**$Label**

**$Line** [4] –> L1

**$Cref**

**$Verdict**

**$End_BehaviourLine**

**$BehaviourLine**

**$Label**

**$Line** [3] +RSRT_UNEXPECTED

**$Cref**

**$Verdict**

**$End_BehaviourLine**

**$BehaviourLine**

**$Label**

**$Line** [3] ?TIMEOUT TD

**$Cref**

**$Verdict** FAIL

**$End_BehaviourLine**

**$BehaviourLine**

**$Label**

**$Line** [3] ?OTHERWISE CANCEL TD

**$Cref**

**$Verdict** FAIL

**$End_BehaviourLine**

**$End_BehaviourDescription**

**$End_TestCase**

La présentation adoptée ici a pour seul but de faciliter la lisibilité.

ANNEXE E

(à la Recommandation X.292)

## Guide stylistique

(Cette annexe ne fait pas partie intégrante de la présente Recommandation)

### E.1 *Introduction*

La présente annexe à caractère informatif présente certaines règles de style qu'il est recommandé de suivre en notation TTCN. Son but est d'assurer une cohérence de base entre styles TTCN utilisés par différents développeurs de suite de tests.

### E.2 *Structure du test élémentaire*

Afin de mieux analyser les résultats d'un test et de déterminer facilement si l'objectif du test est atteint, il est souhaitable d'observer les points suivants relatifs à la manière de structure des tests élémentaires:

a) le développeur d'une suite de tests doit clairement identifier les sous-arbres de préambule et d'épilogue;

b) l'épilogue et le préambule doivent être spécifiés comme un rattachement d'un arbre de test simple (arbre comportemental local du test élémentaire ou tiré de la bibliothèque des modules de tests) dans l'arbre comportemental principal de test élémentaire. Ces arbres de test peuvent se voir eux-mêmes rattacher d'autres sous-arbres;

c) une fois que les sous-arbres de préambule et d'épilogue sont identifiés dans l'arbre comportemental principal de test élémentaire, les autres événements subsistant de cet arbre peuvent être considérés comme se rapportant au corps de test (c'est-à-dire qu'il s'agit d'événements se rapportant à l'objet du test).

Ce procédé permet de délimiter facilement le préambule, le corps et l'épilogue du test élémentaire. Des étiquettes peuvent être utilisées pour indiquer le début et la fin du corps de test dans le journal de conformité.

**Comportement dynamique de test élémentaire**

**Nom du test élémentaire** : TTCN_EXAMPLES/STYLE1
**Référence** : ST_EX1
**Objet** : Illustrer l'identification du préambule et de l'épilogue
**Comportement par défaut** :
**Commentaire** :

| N° | Etiquette | Description du comportement | Réf. aux contraintes | Verdict | Commentaires |
|---|---|---|---|---|---|
| 1 | | +Preamble | | | Lié à l'objectif |
| 2 | | !A | A1 | | Lié à l'objectif |
| 3 | Body | ?B | B1 | | Lié à l'objectif |
| 4 | CinBody | ?C | C1 | (PASS) | |
| 5 | | +postamble_1 | | | Lié à l'objectif |
| 6 | DinBody | ?D | D1 | (PASS) | |
| 7 | | +postamble_2 | | | Lié à l'objectif |
| 8 | | ?E | E1 | INCONC | Lié à l'objectif |
| 9 | | ?OTHERWISE | | FAIL | |

**Figure E-1/X.292 – Identification du préambule et de l'épilogue**

Etant donné que les verdicts finals mettent fin à l'exécution du test élémentaire, un spécificateur de suite de tests ne peut affecter un verdict final dans le corps de test s'il est nécessaire de passer à l'épilogue. Il est toutefois souhaitable de rendre le verdict au niveau du point de test élémentaire où le résultat est effectivement atteint et de ne pas noyer ce verdict dans l'épilogue. Il est donc recommandé d'énoncer les résultats préliminaires dans la colonne verdict lorsqu'un résultat est atteint mais qu'un épilogue doit encore être exécuté. Dans la définition de l'épilogue, un développeur de suite de tests peut utiliser la variable R de résultat en tant que verdict affecté au niveau des feuilles de l'arbre comportemental pour signaler que si aucune erreur n'a été rencontrée dans l'épilogue, le verdit est défini dans le corps du test.

## E.3 *Utilisation de la notation TTCN dans différentes méthodes de test abstraites*

### E.3.1 *Introduction*

Le présent paragraphe lie la notation TTCN aux méthodes de test abstraites définies dans la Recommandation X.291. Il présente la syntaxe TTCN utilisée pour exprimer l'occurrence d'événements aux points PCO, ainsi que les références aux contraintes pour les différentes méthodes de test abstraites considérées.

On suppose que les définitions de type primitive ASP définissent le type du paramètre UserData (données d'usager) en tant qu'unité PDU. Il est alors possible d'utiliser le chaînage des contraintes (c'est-à-dire de renvoyer à une contrainte dans le cas d'une primitive ASP qui comporte une unité PDU dans le paramètre UserData) pour faire référence à une contrainte de primitive ASP comportant une contrainte d'unité PDU en tant que paramètre effectif.

### E.3.2 *Notation TTCN et méthode de test monocouche locale (LS)*

Evénements TTCN possibles:

| *Description comportementale* | *Référence aux contraintes* |
|---|---|
| LT!N_ASP | N_ASPconstraint(N_PDUconstraint) |
| LT?N_ASP | N_ASPconstraint(N_PDUconstraint) |
| UT!T_ASP | T_ASPconstraint |
| UT?T_ASP | T_ASPconstraint |

E.3.3 *Notation TTCN et méthode de test monocouche répartie (DS)*

Evénements TTCN possibles:

| *Description comportementale* | *Référence aux contraintes* |
|---|---|
| LT!N_ASP | N_ASPconstraint(T_PDUconstraint) |
| LT?N_ASP | N_ASPconstraint(T_PDUconstraint) |
| UT!T_ASP | T_ASPconstraint |
| UT?T_ASP | T_ASPconstraint |

E.3.4 *Notation TTCN et méthode de test monocouche coordonnée (CS)*

Evénements TTCN possibles:

| *Description comportementale* | *Référence aux contraintes* |
|---|---|
| LT!N_ASP | N_ASPconstraint(T_PDUconstraint) |
| LT?N_ASP | N_ASPconstraint(T_PDUconstraint) |

En échangeant des unités TM_PDU entre le testeur LT et l'instance de protocole de gestion de test (TM) dans l'instance sous test, via la liaison utilisée pour le test. On notera que dans ce cas la définition d'unité PDU doit avoir déclaré son champ UserData comme étant de type PDU.

| | |
|---|---|
| LT!N_ASP | N_ASPconstraint(T_PDUconstraint(TM_PDUconstraint)) |
| LT?N_ASP | N_ASPconstraint(T_PDUconstraint(TM_PDUconstraint)) |

E.3.5 *Notation TTCN et méthode de test monocouche à distance (RS)*

Evénements TTCN possibles:

| *Description comportementale* | *Référence aux contraintes* |
|---|---|
| LT!N_ASP | N_ASPconstraint(T_PDUconstraint) |
| LT?N_ASP | N_ASPconstraint(T_PDUconstraint) |

Etant donné qu'il n'y a pas de testeur UT ni de protocole TMP, l'événement IMPLICIT SEND est utilisé pour décrire les événements envoi côté instance sous test de la liaison.

| | |
|---|---|
| <IUT!N_ASP> | N_ASPconstraint(T_PDUconstraint) |
| <IUT!T_PDU> | T_PDUconstraint |

E.4 *Utilisation des comportements par défaut*

Question style, un rédacteur de suite de tests doit éviter les situations où l'essai d'une proposition conditionnelle de comportement par défaut est la spécification normale du comportement *prévu* de l'instance sous test. Ce serait par exemple le cas si un module de test représente le comportement du testeur LT ou UT et de l'instance sous test (IUT) quand les événements valides de test sont envoyés, tandis que les réponses de l'instance IUT à des événements de tests non valides ou inopportuns envoyés par le testeur LT ou UT sont spécifiées dans des comportements par défaut implicitement rattachés à ce module de test, lorsqu'il y a appel par d'autres tests élémentaires. Ces comportements par défaut devront alors prendre en charge le verdict «succès».

Cette pratique n'est pas recommandée quand le rattachement d'un arbre par défaut est laissé sans spécification et revêt quelque incertitude. Il faut plutôt utiliser des arbres explicitement rattachés ou l'arbre principal lui-même.

E.5 *Limitation du temps d'exécution d'un test élémentaire*

Dans les versions précédentes du TTCN, une déclaration ELAPSE (temps écoulé) a été définie, permettant au développeur de test élémentaire de limiter la durée d'un test élémentaire si, par exemple, le traitement d'une image instantanée se poursuit indéfiniment ou si une récursion non contrôlée de rattachement à un arbre se produit.

La déclaration ELAPSE ne fait plus partie de la notation TTCN, le problème qu'elle était censée résoudre étant considéré comme sortant du cadre de la spécification des suites de tests.

Pour limiter le temps d'exécution d'un test élémentaire, il est maintenant recommandé aux développeurs de tests d'élaborer des mécanismes locaux de limitation en même temps que les autres dispositifs de test. Lorsqu'une limite dans le temps pour la réalisation d'un événement doit être fixée, on pourra utiliser des temporisations explicites avec l'événement TIMEOUT (fin de temporisation).

### E.6 *Types structurés*

a) Dans les versions préDIS (projet de norme internationale) du TTCN, des champs et valeurs génériques ont été définis comme caractéristiques permettant de regrouper plusieurs champs ou valeurs dans une table de contraintes, ou de réutiliser ce groupe dans plusieurs tables de contraintes de même contenu.

b) Dans la présente version, le groupement des paramètres de primitive ASP et des champs d'unité PDU (ex-types de données) est introduit en premier dans la partie déclarative afin que cette partie soit complète et que la cohérence soit maintenue entre les notations ASN.1 et TTCN. Le § 10.2.3.3 donne une définition des tables de définition de type structuré. Une fois qu'un type structuré est déclaré, il peut être utilisé par une ou plusieurs définitions de type primitive ASP ou unité PDU. La table de définition de primitive ASP et d'unité PDU peut donc être de type «plat» (aucun groupe ou un groupe introduit par un appel de macroinstruction) ou structuré (au moyen de spécifications de structures s'appliquant aux paramètres nommés de primitive ASP ou aux champs nommés d'unité PDU).

c) Dans la partie contraintes, des valeurs doivent être affectées aux éléments de structure dans les tables de contraintes de type structuré. Les noms de ces contraintes peuvent être utilisés comme valeurs dans les tables de contraintes de base de primitive ASP ou d'unité PDU.

   Les tables des contraintes de primitive ASP et d'unité PDU peuvent donc aussi être:

   – de type plat, c'est-à-dire qu'elles affectent des valeurs à chaque paramètre ou champ et ne renvoient aux tables de contraintes de structure que par appel de macroinstruction; ou

   – de type structuré, c'est-à-dire qu'elles remplacent les valeurs des groupes déclarés de paramètres ou de champs par des noms de contraintes de groupe.

d) Si la primitive ASP ou l'unité PDU déclarée est structurée au moyen de certains paramètres de primitive ASP ou certains champs d'unité PDU spécifiés par référence à des éléments structurés, les contraintes doivent avoir alors la même structure.

e) Quelle que soit la forme utilisée, les contraintes de primitive ASP ou d'unité PDU peuvent aussi être:

   – modifiées;

   – paramétrées au moyen d'un paramètre qui se verra affecter une valeur de champ ou de paramètre ou une contrainte de type structuré.

f) Les tables de contraintes de type structuré remplacent les tables de champs génériques des versions précédentes de la notation TTCN.

g) Le concept de valeurs génériques est supprimé.

h) On trouvera des exemples dans l'annexe D.

### E.7 *Abréviations*

Dans les versions précédentes de la notation TTCN, il était permis de déclarer, dans une table spécifique, les abréviations à utiliser dans les colonnes comportement des tests élémentaires et des modules de test. Cette facilité s'est avérée prêter à confusion et a été limitée à l'abréviation des seuls noms de primitives ASP et d'unités PDU, lorsque ces noms apparaissent sur les lignes événements. Cette facilité a reçu le nom de fonction Alias (pseudonyme).

### E.8 *Descriptions de tests*

Les descriptions informelles de comportement donnant plus de détails que les objets de test, mais moins que les spécifications TTCN des tests élémentaires peuvent, si on le désire, être incluses dans une suite de tests abstraite (ATS) normalisée.

Ces descriptions de tests peuvent utiliser le langage naturel, les diagrammes chronologiques ou toute autre notation et être situées dans la partie commentaires des tables, dans une annexe informative ou dans les deux.

Les spécifications TTCN des tests élémentaires ont toujours la priorité sur les descriptions informelles de test.

E.9 *Affectations relatives aux événements SEND*

La notation TTCN permet de remplacer des valeurs de contrainte par d'autres avant un événement SEND, dans une déclaration d'affectation sur la ligne événement. Cela signifie que les données à envoyer sont d'abord construites à partir de la définition des contraintes et que les affectations sont ensuite exécutées.

Cette caractéristique doit être utilisée avec prudence car la détermination de la valeur effectivement envoyée risque d'être confuse pour le lecteur de la suite de tests. On estime notamment qu'il est mal venu d'utiliser la même contrainte pour l'envoi et la réception.

E.10 *Points PCO multiservice*

La spécification précise d'un point PCO couvrant plus d'un point d'accès au service (SAP), est donnée par l'ensemble des primitives ASP et des unités PDU qui peuvent y apparaître.

*Exemple E.1* – Un point PCO pour le service

| **Déclarations de point PCO** | | | |
|---|---|---|---|
| **Nom du point PCO** | **Type de point PCO** | **Rôle** | **Commentaires** |
| L | A_P_SAPs | LT | Point PCO par lequel on peut observer toutes les primitives ASP d'élément ACSE et toutes les primitives ASP de présentation, sauf P-CONNECT, P-RELEASE et P_ABORT |

Le point PCO «L» est de type A_P_SAP qui est capable d'observer toutes les primitives ASP de présentation et d'éléments de service de contrôle d'association (ACSE), à l'exclusion de P-CONNECT, P-RELEASE et P-ABORT. La colonne type montre que les points SAP «A» et «P» appartiennent à l'ensemble qui doit être observé par les points PCO, chaque point SAP étant séparé par un souligné ("_"). La colonne commentaires décrit exactement ce qui peut être vu par le point PCO.

Cette méthode peut s'étendre à de nombreux points SAP, qui seront tous séparés les uns des autres par un soulignement.

ANNEXE F

(à la Recommandation X.292)

**Liste des numéros des productions BNF**

F.1 *Introduction*

La présente annexe est l'index alphabétique des productions BNF qui apparaissent dans l'annexe A. L'index renvoie au numéro de chaque production (et non au numéro de la page).

F.2 *Index des productions*

# A

## B



## C



## D

## O



## P



## Q



## R



## S

## T

## U



## V



## W

# Remplacée par une version plus récente

ANNEXE G

(à la Recommandation X.292)

**Index**

(Cette annexe ne fait pas partie intégrante de la présente Recommandation)

G.1 *Introduction*

La présente annexe est l'index alphabétique anglais des termes et acronymes utilisés dans la Norme ISO/CEI 9646 (1991) partie 3. Pour chaque terme ou acronyme, l'index donne un ensemble de références renvoyant aux numéros d'articles, de figures et de tableaux du corps principal et des annexes de la partie 3.

La signification de chaque référence est indiquée comme suit:

a) en **gras**, les définitions des termes et acronymes;

b) en *italique*, les principaux emplois des termes et acronymes;

c) en caractères normaux, les autres emplois.

G.2 *Index*

# B



# C

# D

# E



# F



# G



# H



# I

# L



# M

## N



## O



## P

# Q

# R



# S

# T

## U



## V



## W

# Remplacée par une version plus récente