UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

CCITT

COMITÉ CONSULTATIF
INTERNATIONAL
TÉLÉGRAPHIQUE ET TÉLÉPHONIQUE

LIVRE ROUGE

TOME VI – FASCICULE VI.10

LANGAGE DE SPÉCIFICATION ET DE DESCRIPTION FONCTIONNELLES (LDS)

RECOMMANDATIONS Z.100 À Z.104

VIII^e ASSEMBLÉE PLÉNIÈRE
MALAGA-TORREMOLINOS, 8-19 OCTOBRE 1984

Genève 1985

UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

CCITT

COMITÉ CONSULTATIF
INTERNATIONAL
TÉLÉGRAPHIQUE ET TÉLÉPHONIQUE

LIVRE ROUGE

TOME VI – FASCICULE VI.10

LANGAGE DE SPÉCIFICATION ET DE DESCRIPTION FONCTIONNELLES (LDS)

RECOMMANDATIONS Z.100 À Z.104

VIIIe ASSEMBLÉE PLÉNIÈRE
MALAGA-TORREMOLINOS, 8-19 OCTOBRE 1984

Genève 1985

ISBN 92-61-02232-4

CONTENU DU LIVRE DU CCITT EN VIGUEUR APRÈS LA HUITIÈME ASSEMBLÉE PLÉNIÈRE (1984)

LIVRE ROUGE

Tome I – Procès-verbaux et rapports de l'Assemblée plénière.

Vœux et résolutions.

Recommandations sur:

- l'organisation du travail du CCITT (série A);
- les moyens d'expression (série B);
- les statistiques générales des télécommunications (série C).

Liste des Commissions d'études et des Questions mises à l'étude.

Tome II – *(Divisé en 5 fascicules vendus séparément)*

FASCICULE II.1 – Principes généraux de tarification – Taxation et comptabilité dans les services internationaux de télécommunications – Recommandations de la série D (Commission d'études III).

FASCICULE II.2 – Service téléphonique international – Exploitation – Recommandations E.100 à E.323 (Commission d'études II).

FASCICULE II.3 – Service téléphonique international – Gestion du réseau – Ingénierie du trafic – Recommandations E.401 à E.600 (Commission d'études II).

FASCICULE II.4 – Services télégraphiques – Exploitation et qualité de service – Recommandations F.1 à F.150 (Commission d'études I).

FASCICULE II.5 – Services de télématique – Exploitation et qualité de service – Recommandations F.160 à F.350 (Commission d'études I).

Tome III – *(Divisé en 5 fascicules vendus séparément)*

FASCICULE III.1 – Caractéristiques générales des communications et des circuits téléphoniques internationaux – Recommandations G.101 à G.181 (Commissions d'études XV, XVI et CMBD).

FASCICULE III.2 – Systèmes internationaux analogiques à courants porteurs – Caractéristiques des moyens de transmission – Recommandations G.211 à G.652 (Commissions d'études XV et CMBD).

FASCICULE III.3 – Réseaux numériques – Systèmes de transmission et équipement de multiplexage – Recommandations G.700 à G.956 (Commissions d'études XV et XVIII).

FASCICULE III.4 – Utilisation des lignes pour les transmissions des signaux autres que téléphoniques – Transmissions radiophoniques et télévisuelles – Recommandations des séries H et J (Commission d'études XV).

FASCICULE III.5 – Réseau numérique avec intégration des services (RNIS) – Recommandations de la série I (Commission d'études XVIII).

Tome IV – *(Divisé en 4 fascicules vendus séparément)*

FASCICULE IV.1 – Maintenance: principes généraux, systèmes de transmission internationaux, circuits téléphoniques internationaux – Recommandations M.10 à M.762 (Commission d'études IV).

FASCICULE IV.2 – Maintenance des circuits internationaux pour la transmission de télégraphie harmonique ou de télécopie – Maintenance des circuits internationaux loués – Recommandations M.800 à M.1375 (Commission d'études IV).

FASCICULE IV.3 – Maintenance des circuits radiophoniques internationaux et transmissions télévisuelles internationales – Recommandations de la série N (Commission d'études IV).

FASCICULE IV.4 – Spécifications des appareils de mesure – Recommandations de la série O (Commission d'études IV).

Tome V – Qualité de la transmission téléphonique – Recommandations de la série P (Commission d'études XII).

Tome VI – *(Divisé en 13 fascicules vendus séparément)*

FASCICULE VI.1 – Recommandations générales sur la commutation et la signalisation téléphoniques – Interface avec le service maritime et le service mobile terrestre – Recommandations Q.1 à Q.118 *bis* (Commission d'études XI).

FASCICULE VI.2 – Spécifications des Systèmes de signalisation n° 4 et 5 – Recommandations Q.120 à Q.180 (Commission d'études XI).

FASCICULE VI.3 – Spécifications du Système de signalisation n° 6 – Recommandations Q.251 à Q.300 (Commission d'études XI).

FASCICULE VI.4 – Spécifications des Systèmes de signalisation R1 et R2 – Recommandations Q.310 à Q.490 (Commission d'études XI).

FASCICULE VI.5 – Centraux numériques de transit dans les réseaux numériques intégrés et les réseaux mixtes analogiques-numériques. Centraux numériques locaux et mixtes – Recommandations Q.501 à Q.517 (Commission d'études XI).

FASCICULE VI.6 – Interfonctionnement des systèmes de signalisation – Recommandations Q.601 à Q.685 (Commission d'études XI).

FASCICULE VI.7 – Spécifications du Système de signalisation n° 7 – Recommandations Q.701 à Q.714 (Commission d'études XI).

FASCICULE VI.8 – Spécifications du Système de signalisation n° 7 – Recommandations Q.721 à Q.795 (Commission d'études XI).

FASCICULE VI.9 – Système de signalisation avec accès numérique – Recommandations Q.920 à Q.931 (Commission d'études XI).

FASCICULE VI.10 – Langage de spécification et de description fonctionnelles (LDS) – Recommandations Z.101 à Z.104 (Commission d'études XI).

FASCICULE VI.11 – Langage de spécification et de description fonctionnelles (LDS), annexes aux Recommandations Z.101 à Z.104 (Commission d'études XI).

FASCICULE VI.12 – Langage évolué du CCITT (CHILL) – Recommandation Z.200 (Commission d'études XI).

FASCICULE VI.13 – Langage homme-machine (LHM) – Recommandations Z.301 à Z.341 (Commission d'études XI).

Tome VII – *(Divisé en 3 fascicules vendus séparément)*

FASCICULE VII.1 – Transmission télégraphique – Recommandations de la série R (Commission d'études IX). – Equipements terminaux pour les services de télégraphie – Recommandations de la série S (Commission d'études IX).

FASCICULE VII.2 – Commutation télégraphique – Recommandations de la série U (Commission d'études IX).

FASCICULE VII.3 – Equipements terminaux et protocoles pour les services de télématique – Recommandations de la série T (Commission d'études VIII).

Tome VIII – *(Divisé en 7 fascicules vendus séparément)*

FASCICULE VIII.1 – Communication de données sur le réseau téléphonique – Recommandations de la série V (Commission d'études XVII).

FASCICULE VIII.2 – Réseaux de communications de données; services et facilités – Recommandations X.1 à X.15 (Commission d'études VII).

FASCICULE VIII.3 – Réseaux de communications de données; interfaces – Recommandations X.20 à X.32 (Commission d'études VII).

FASCICULE VIII.4 – Réseaux de communications de données; transmission, signalisation et commutation, réseau, maintenance et dispositions administratives – Recommandations X.40 à X.181 (Commission d'études VII).

FASCICULE VIII.5 – Réseaux de communications de données: interconnexion de systèmes ouverts (OSI), techniques de description du système – Recommandations X.200 à X.250 (Commission d'études VII).

FASCICULE VIII.6 – Réseaux de communications de données: interfonctionnement entre réseaux, systèmes mobiles de transmission de données – Recommandations X.300 à X.353 (Commission d'études VII).

FASCICULE VIII.7 – Réseaux de communications de données: systèmes de traitement des messages – Recommandations X.400 à X.430 (Commission d'études VII).

Tome IX – Protection contre les perturbations – Recommandations de la série K (Commission d'études V) – Construction, installation et protection des câbles et autres éléments d'installations extérieures – Recommandations de la série L (Commission d'études VI).

Tome X – *(Divisé en 2 fascicules vendus séparément)*

FASCICULE X.1 – Termes et définitions.

FASCICULE X.2 – Index du Livre rouge.

PAGE INTENTIONALLY LEFT BLANK

PAGE LAISSEE EN BLANC INTENTIONNELLEMENT

TABLE DES MATIÈRES DU FASCICULE VI.10 DU LIVRE ROUGE

Recommandations Z.100 à Z.104

Langage de spécification et de description fonctionnelles (LDS)

NOTES PRÉLIMINAIRES

1 Les Questions confiées à chaque Commission d'études pour la période 1985-1988 figurent dans la contribution N° 1 de la Commission correspondante.

2 Dans ce fascicule, l'expression «Administration» est utilisée pour désigner de façon abrégée aussi bien une administration de télécommunications qu'une exploitation privée reconnue de télécommunications.

3 La Conférence de plénipotentiaires, Nairobi, 1982, a décidé que le terme «Avis» du CCITT et du CCIR devrait être remplacé par le terme «Recommandation» dans les publications de l'UIT. Pour simplifier le traitement des textes du présent Livre, le mot «Avis» avec «A» majuscule a été systématiquement remplacé par le mot «Recommandation»; en conséquence, les Avis des CCI publiés antérieurement au Livre rouge seront désignés, à partir de maintenant, par le mot «Recommandation».

FASCICULE VI.10

Recommandations Z.100 à Z.104

LANGAGE DE SPÉCIFICATION ET DE DESCRIPTION FONCTIONNELLES (LDS)

PAGE INTENTIONALLY LEFT BLANK

PAGE LAISSEE EN BLANC INTENTIONNELLEMENT

INTRODUCTION AU LDS

1 Introduction

La présente Recommandation constitue une explication et une introduction générales du langage de spécification et de description du CCITT (LDS). Ce langage est défini en détail dans les Recommandations Z.101 à Z.104.

1.1 *Observations générales*

Le but poursuivi, en recommandant l'utilisation du LDS, est d'avoir un langage permettant de *spécifier* et de *décrire* sans ambiguïté le comportement des *systèmes* de télécommunications. Les *spécifications* et les *descriptions* faites à l'aide du LDS doivent être formelles dans ce sens qu'il doit être possible de les analyser et de les interpréter sans ambiguïté.

Une *spécification de système* se compose d'une spécification du *comportement fonctionnel* et d'un ensemble de *paramètres généraux du système*. Le LDS ne vise qu'à décrire les aspects relatifs au comportement d'un *système*; les *paramètres généraux* concernent des propriétés telles que la capacité et le poids, qui doivent être décrites à l'aide de techniques différentes.

Les termes *«spécification»* et *«description»* sont utilisés dans le sens ci-après:

- la *spécification* d'un *système* est la description du *comportement* souhaité de celui-ci, et
- la *description* d'un *système* est la description du *comportement* réel de celui-ci.

Le LDS décrit le *comportement* sous forme de stimulus/réaction, étant admis que les stimuli aussi bien que les réactions sont des entités discrètes. L'approche se fonde sur la notion de *machines à état fini étendues*.

Le *comportement* du *système*, tel qu'il est décrit dans le LDS, est formé par la suite des réactions provoquées par une suite quelconque de stimuli, le phénomène étant vu de l'extérieur du *système*. Deux *systèmes* quelconques décrits dans le LDS et ayant le même *comportement* à cet égard sont dits *fonctionnellement équivalents*. La notion d'*équivalence fonctionnelle* sert à comparer des *systèmes*, par exemple, comparaison de la *spécification* d'un *système* requis avec la *description* d'un *système* proposé. Les Directives pour les usagers du LDS, annexées aux Recommandations, fournissent un examen critique des critères et des méthodes retenues pour effectuer cette comparaison.

Le LDS contient également des principes de structuration qui permettent de découper un *système* en parties pouvant être définies, développées et comprises l'une après l'autre.

Ces principes sont utiles en premier lieu pour spécifier un système lorsque des aspects différents peuvent être traités indépendamment les uns des autres et, à un stade ultérieur, pour décrire un *système* lorsque les structures de description doivent correspondre à la structure du *système*.

1.2 *Objectifs*

Les objectifs généraux qui ont été pris en compte lors de la définition du LDS sont de fournir un langage:

- facile à apprendre, à utiliser et à interpréter en fonction des besoins des organismes exploitants;
- permettant l'élaboration de spécifications et de descriptions dépourvues d'ambiguïtés pour faciliter la soumission des offres et la commande;
- assez souple pour permettre un développement ultérieur;
- permettant l'application de plusieurs méthodologies de spécification et de conception de système, sans postuler a priori l'une quelconque de ces méthodologies.

1.3 *Domaine d'application*

Le domaine d'application principal du LDS est la description du comportement des *systèmes* de télécommunication dans certains de leurs aspects. Ces applications comprennent:

- le traitement des appels (par exemple: écoulement, signalisation téléphonique, comptage aux fins de taxation, etc.) dans les systèmes de commutation SPC;
- la maintenance et la relève des dérangements (par exemple: alarme, relève automatique des dérangements, essais périodiques, etc.) dans les systèmes généraux de télécommunication;
- la commande du système (par exemple: protection contre les surcharges, procédure de modification et d'extension, etc.);
- les protocoles de communications de données.

Il va de soi que le LDS peut aussi servir à décrire tout comportement pouvant être décrit au moyen d'un modèle discret, c'est-à-dire en communiquant avec son environnement par messages discrets.

2 Analyse générale du langage

L'analyse du LDS que l'on trouvera ci-après est conçue comme une introduction aux Recommandations Z.101 à Z.104. Ces explications ne sont pas, à l'exception du § 2.1, des définitions formelles mais uniquement des explications à caractère didactique.

2.1 *Quelques définitions fondamentales*

Les Recommandations Z.100 à Z.104 font appel à des conventions et à des concepts généraux dont les définitions sont données ci-après:

Type, définition et instance

Dans les Recommandations, une entité est rigoureusement séparée de sa *définition*. On utilise la terminologie définie ci-après et illustrée par le schéma de la figure 1/Z.100.

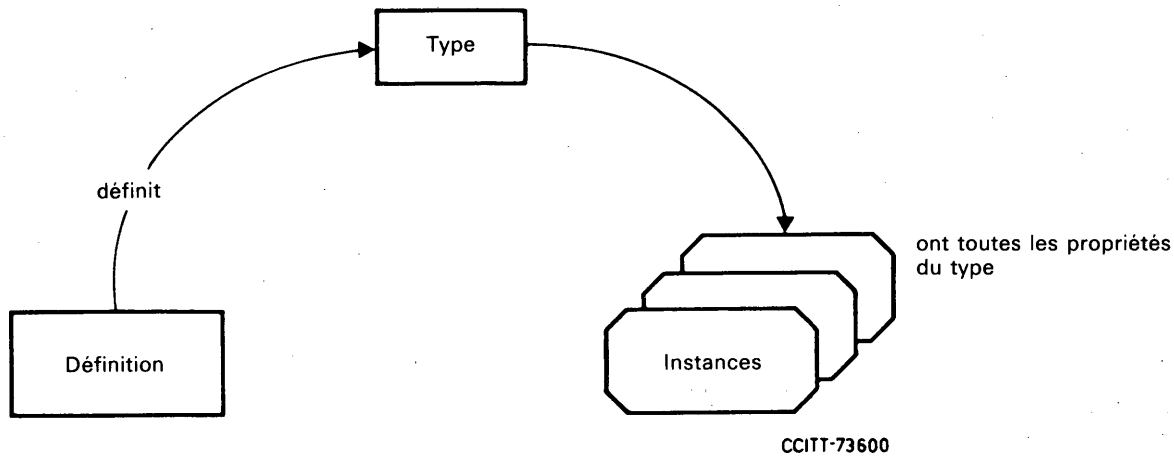


FIGURE 1/Z.100

Concept de type

Un *type* se définit par sa *définition*, laquelle définit toutes les propriétés associées à ce *type*. Un *type* peut être décomposé en un nombre quelconque d'*instances*. Toute *instance* d'un *type* particulier possède toutes les propriétés définies pour le *type*.

Le schéma s'applique à plusieurs concepts du LDS, c'est-à-dire qu'il existe des *définitions de système* et des *instances de système*, des *définitions de processus* et *des instances de processus.*

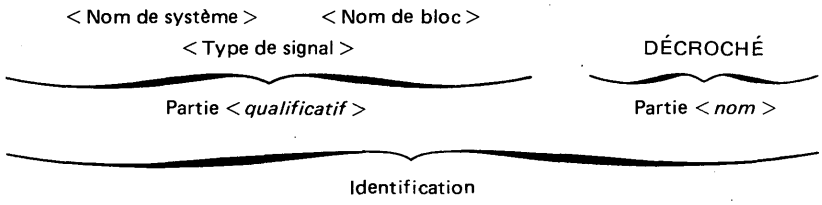
Les *instances* d'un *type* peuvent être elles-mêmes des *types*. Les propriétés associées aux *types* sont léguées en fonction des modalités de l'application hiérarchique; par exemple, les propriétés du *système* de type générique se retrouvent dans toutes les *instances* des *définitions d'un système* spécifique.

Pour éviter d'alourdir le texte, on applique la convention suivante: lorsque le contexte indique sans ambiguïté qu'on se trouve en présence de la *définition* ou de *l'instance* d'un type, l'attribut n'est pas mentionné.

Noms

Les conventions et la terminologie ci-après sont adoptées pour désigner les *définitions* et les *instances*.

Tous les identificateurs utilisés dans une représentation sont uniques. L'identification d'une entité se compose de deux parties: une partie *«nom»* et une partie *«qualificatif»* :



La partie *«qualificatif»* est tirée du contexte hiérarchique complet dans lequel est définie l'entité et du *type* de l'entité. La *partie «nom»* doit être un *nom* significatif par rapport à l'objectif ou à l'effet de l'entité nommée.

Lorsque l'identification apparaît dans la syntaxe concrète, les *qualificatifs* évidents peuvent être simplement sous-entendus. Lorsqu'il n'y a aucune ambiguïté, seule la *partie nom* doit être utilisée.

Règles de visibilité

Lorsqu'un *identificateur* est introduit dans une *spécification* ou dans une *description*, cet *identificateur* est *visible* au point d'introduction et aux niveaux situés directement au-dessous du point d'introduction dans la hiérarchie de la *spécification* ou de la *description.* Le niveau le plus élevé auquel l'*identificateur* est visible correspond au niveau hiérarchique le plus bas indiqué dans la partie *«qualificatif»* de l'*identificateur.*

Erreur de spécification

Lorsque les propriétés d'une *spécification* du LDS sont incohérentes ou ambiguës, la *spécification* est invalide. Lorsque l'interprétation d'une *spécification* LDS est en contradiction avec les propriétés d'une *spécification* valide, cette interprétation est *erronée.* Toute interprétation d'un *système* conduisant à une erreur implique que le comportement futur du *système* ne peut être prédit sur la base de la spécification.

2.2 *Le LDS de base*

Le *comportement* dynamique d'un *système* LDS est produit par des *instances de processus*, agissant simultanément. Un *processus* est modélisé sous la forme d'une *machine à état fini étendue.* Il ne réagit qu'à des stimuli externes discrets, et peut alors renvoyer des réponses discrètes à son *environnement.* D'autres *processus* peuvent accepter les réponses créées en tant que stimuli.

Dans le LDS, un *processus* se trouve en attente dans un certain *état* jusqu'à ce que celui-ci reçoive un *signal* valide de son *environnement.* Puis il effectuera une *transition* aboutissant à un nouvel *état.* Pendant cette *transition,* il pourra réaliser des *actions*; il peut s'agir soit de manipulation de l'information locale au *processus,* soit d'*émission de signaux* à d'autres *processus* ou à l'*environnement* externe du *système.*

Tout *processus* d'un *système* ou de son *environnement* a accès au «temps absolu», et peut effectuer des mesures de durée ou des temporisations.

Les *instances de processus* peuvent être *créées* et *arrêtées* de façon dynamique. Une *instance* peut être créée par une autre *instance,* ou bien elle peut exister au moment de l'activation du *système.* Il ne peut être *mis fin* à une *instance de processus* que par une *action* d'*arrêt* explicite réalisée par elle-même.

Plusieurs *instances de signaux* peuvent être en attente pour être absorbées par un *processus*. Afin de traiter cette file de *signaux,* on associe un *point d'entrée* à chaque *instance de processus.* Le système de file fonctionne essentiellement d'après le principe «premier arrivé, premier sorti», mais le *processus* peut lui-même traiter cette file grâce à un *ensemble de signaux mis en réserve* associé à un *état.*

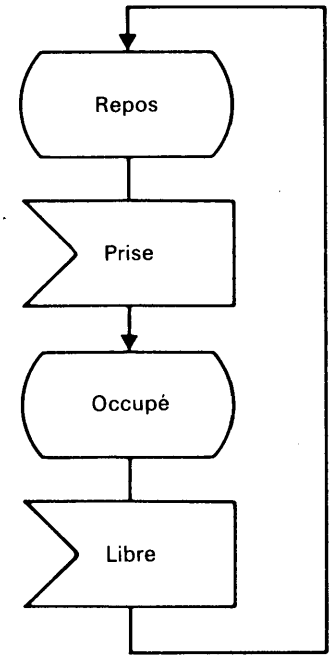


FIGURE 2/Z.100

Définition d'un processus

Les *instances de signaux* peuvent véhiculer des *valeurs de données* dont disposera l'*instance de processus* lorsqu'elle recevra le *signal*. Ces *valeurs de données* peuvent être mémorisées dans les *variables* locales du *processus*, et en être extraite.

On peut décrire la composition de la *structure* statique d'un *système* LDS au moyen de *système*, *blocs* et *canaux*, comme indiqué dans la figure 3/Z.100.

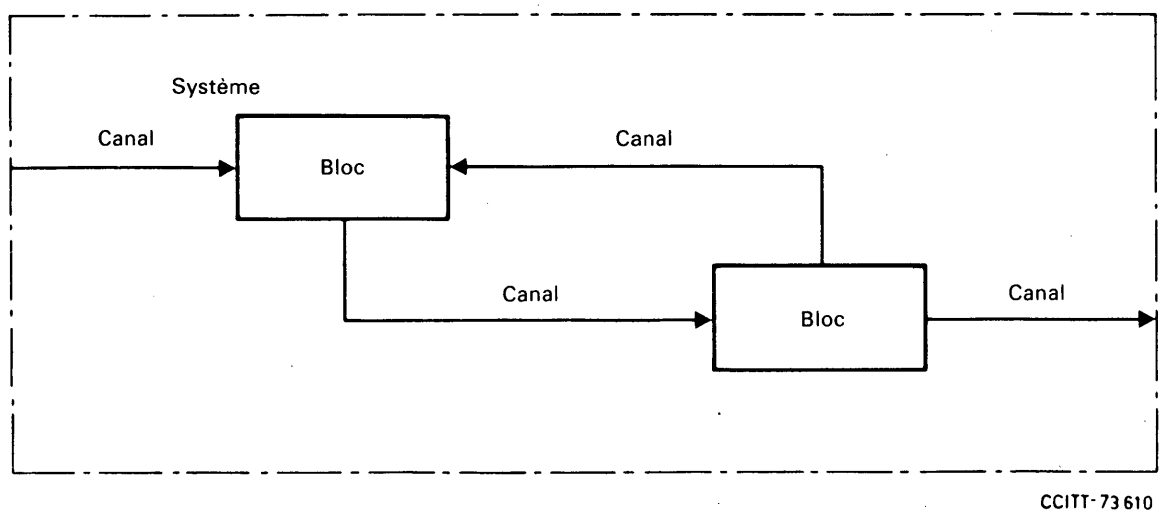


FIGURE 3/Z.100

Structure statique d'un système LDS

Le *système* se compose de *blocs* liés les uns aux autres et à la *frontière du système* au moyen de *canaux*. La *frontière du système* sépare le *système* de son *environnement*. On suppose que l'*environnement* «a un comportement conforme au LDS», c'est-à-dire qu'il émettra des *signaux* vers le *système* et qu'il acceptera des réponses sous forme de *signaux* provenant du *système*.

Les *canaux* servent à véhiculer les *signaux* entre les *blocs*, et entre les *blocs* et la *frontière du système*. Les *canaux* sont unidirectionnels. Les *blocs* servent de contenant aux *processus* et à structurer le *système* en construisant des blocs.

2.3 *Données dans le LDS*

Les processus du LDS peuvent conserver et manipuler des *valeurs de données*. Celles-ci sont *liées* à des *variables* qui sont locales par rapport à un *processus*. Toutefois, les *valeurs de données* peuvent être véhiculées entre des *processus* et échangées avec l'*environnement* au moyen de *signaux*. Les *variables* locales à un *processus* sont définies dans la *définition du processus*.

Les données traitées sont classées par *types*. Le LDS définit un certain nombre de *types de données prédéfinis*. De son côté, l'usager peut définir de nouveaux *types de données*. Les *types de données prédéfinis* sont les suivants:

- *Booléen* — ayant des *valeurs* VRAI et FAUX et permettant des opérations logiques normales sur ces *valeurs*.
- *Entier* — au sens mathématique habituel des nombres entiers.
- *Naturel* — au sens mathématique habituel des nombres naturels.
- *Réel* — au sens mathématique habituel des nombres réels.
- *Caractère* — au sens habituel (comme dans les langages de programmation), avec des *valeurs* et une représentation comme dans l'alphabet n° 5 du CCITT.
- *Tableau* — collection indexée d'objets appartenant à un même *type de données*. Extension, au sens normal de tableau («array») utilisé dans les langages de programmation.
- *Structure* — assemblage d'un certain nombre de *types de données* qui peuvent être différents les uns des autres.
- *Chaîne* — liste d'objets appartenant à un *type de données* quelconque et de longueur arbitraire.
- *Chaîne de caractères* — liste de caractères de longueur arbitraire. *Remarque* — Une *chaîne de caractères* est une *chaîne* de caractères formée à partir d'une *chaîne* de *types de données* avec les *caractères* de *types de données*.
- *Mode ensembliste* — au sens mathématique habituel, tel qu'une *valeur ensembliste* est un ensemble ordonné.
- *PId* — utilisé pour identifier des *instances de processus*.
- *Temps* — temps absolu.
- *Durée* — intervalles entre deux instants dans le temps absolu.
- *Temporisateur* — le *type de données* utilisé pour des temporisateurs et qui définit les opérations d'initialisation et de réinitialisation pour les variables du temporisateur.

Le concept STRUCT permet de mettre en œuvre d'autres types de données renfermant des valeurs composites composées à partir d'un nombre de types de données (éventuellement différents) à construire.

Les *types de données* définis par l'usager peuvent être définis au moyen de *types de données prédéfinis* avec des restrictions telles que l'intervalle d'un *nombre entier*. Il peut également s'agir de types de données, abstraits additionnels. Les *types de données abstraits* sont définis par une méthode «axiomatique».

Les *types de données* peuvent être génériques pour un *système*, ou contenues dans une *définition de bloc*, dans une *définition de canal* ou dans une *définition de processus*. La *définition* est *visible*, et donc utilisable, dans la portée de l'objet dans lequel elle est contenue. Par exemple, si une *définition de type* est contenue dans un *bloc*, les *données* de ce *type* pourront être utilisées dans tous les *processus* contenus dans le *bloc* et dans les *sous-blocs* du *bloc*.

2.4 *Concepts structurels dans le LDS*

Les concepts structurels qu'offre le LDS visent à faciliter les descriptions de *systèmes* complexes et/ou grands. Les concepts sont définis de façon à permettre:

- la *subdivision* de grandes descriptions en modules afin que les parties puissent être traitées et comprises indépendamment;
- la description d'un *système*, ou de parties d'un *système*, à plusieurs niveaux d'abstraction;
- la description de la structure effective d'un *système*.

Il convient de préciser, lors de l'utilisation de ces concepts structurels, s'ils représentent la *structure* requise ou effective ou s'ils sont utilisés seulement pour faciliter la représentation.

Dans le LDS de base, un *système* se compose de *blocs*, de *canaux* et de *processus*. Ces composantes peuvent être à leur tour subdivisées, à savoir les *blocs* en *sous-blocs* et en *sous-canaux*, les *canaux* en *blocs* et en *canaux* et les *processus* en *sous-processus*. Une *définition de processus* peut être aussi décomposée à nouveau en un certain nombre d'échelons, à l'aide de *procédures* et de *macros*.

Lorsque l'on *subdivise les blocs* en *sous-blocs* et en *sous-canaux*, on obtient une description hiérarchique du *système* à plusieurs niveaux. L'information contenue dans les *niveaux* supérieurs doit être contenue dans les interfaces des *niveaux* inférieurs. La structure ainsi obtenue peut être représentée sous la forme d'un diagramme d'*«arbre de blocs»*. Voir la figure 4/Z.100.

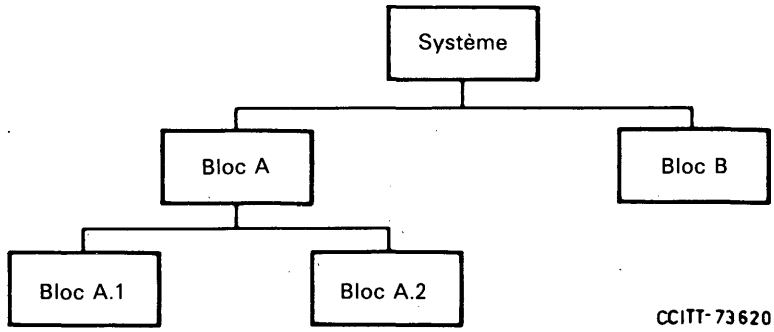


FIGURE 4/Z.100

Diagramme d'arbre de blocs

On peut aussi décrire cette structure de façon plus détaillée grâce à un *diagramme d'interaction de blocs*, comme celui représenté à la figure 5/Z.100.

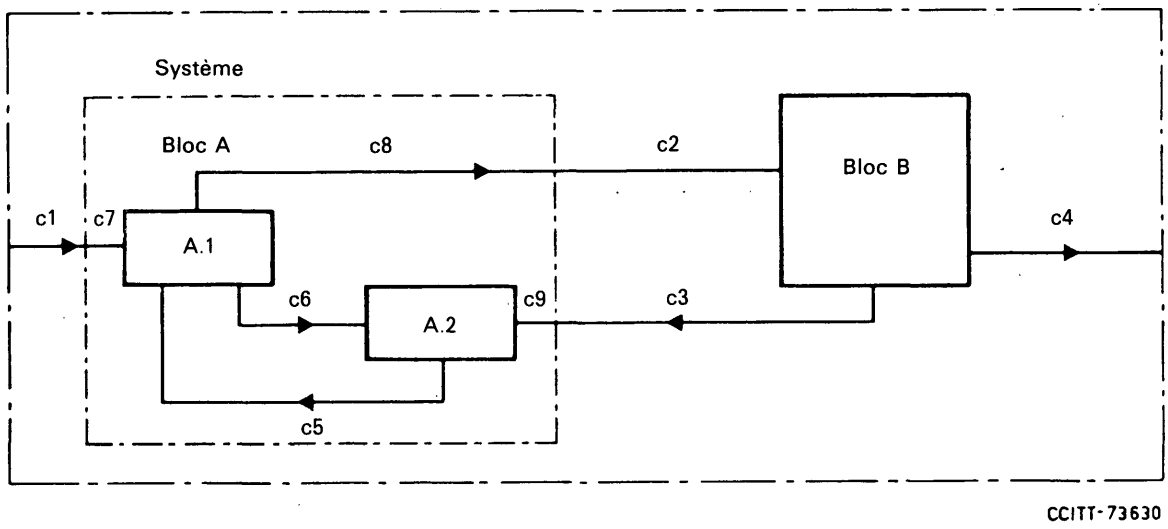
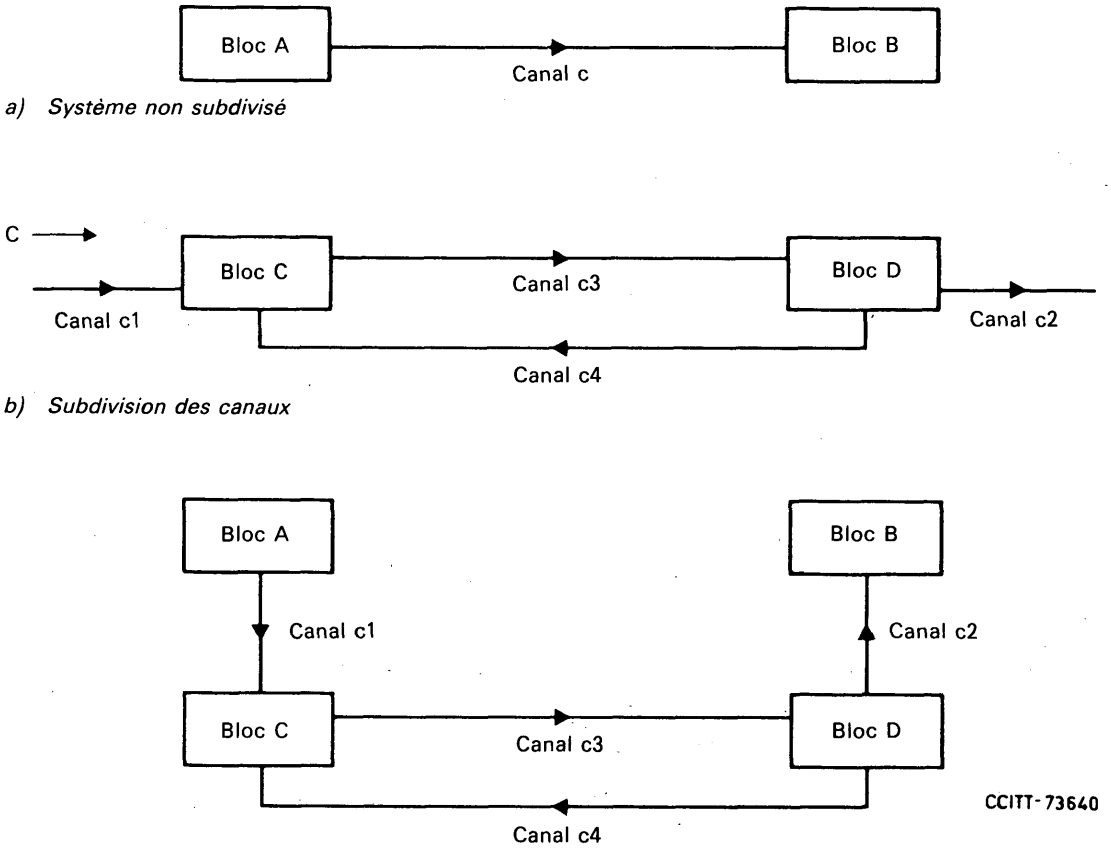


FIGURE 5/Z.100

Diagramme d'interaction de blocs

Ce dernier diagramme montre également les *canaux* et les *sous-canaux* qui relient la structure des *blocs* et des *sous-blocs*. Lorsqu'on utilise cette *subdivision des blocs*, il suffit pour que la description soit significative, que les *sous-blocs* les plus détaillés (c'est-à-dire les «blocs-feuilles» représentés dans le diagramme de l'*arbre de blocs*) contiennent les *processus*.

La subdivision des *canaux* en *canaux* et en *blocs* donne également une structure hiérarchique. On en trouvera un exemple simple dans la figure 6/Z.100.



a) Système non subdivisé

b) Subdivision des canaux

c) Système avec un canal subdivisé

FIGURE 6/Z.100

Subdivision des canaux

La *subdivision* d'un processus en un ensemble de *sous-processus* est liée à la *subdivision des blocs*, puisque les *sous-processus* d'un *processus* doivent toujours être situés dans le(s) *sous-bloc(s)* du *bloc* contenant le *processus*, comme cela est illustré par la figure 7/Z.100.

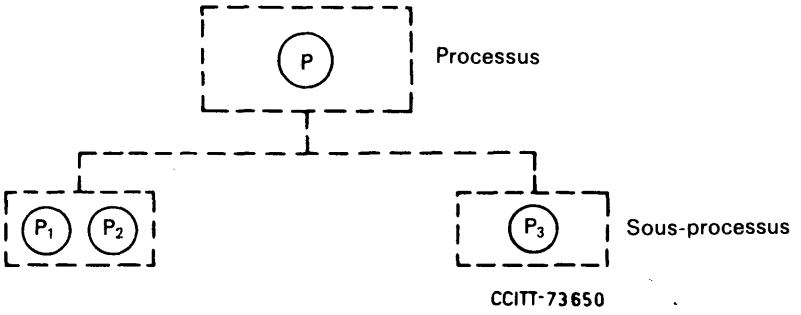


FIGURE 7/Z.100

Subdivision d'un processus

Les *sous-processus* d'un processus pris dans leur ensemble représentent une autre description, plus détaillée, du comportement décrit par le *processus*. Donc, lorsqu'on a une description comme celle représentée à la figure 7/Z.100, il faut choisir entre l'interprétation du *comportement* détaillé, représenté par les *sous-processus*, et celle du comportement moins détaillé, représenté par le *processus*. Dans le LDS, on dit que les autres descriptions possibles soutiennent des *niveaux d'abstraction* différents.

La *subdivision* d'un *processus* est illustrée par un diagramme d'*arbre de processus* comme celui qui est représenté à la figure 8/Z.100.

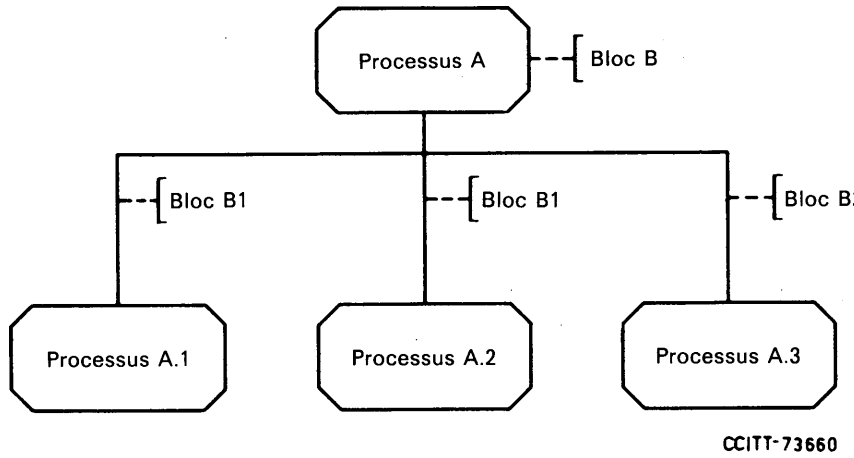


FIGURE 8/Z.100

Diagramme d'arbre de processus

Un *processus* peut aussi être structuré et détaillé grâce à l'utilisation de *procédures*. Une *procédure* dans le LDS est semblable aux procédures dans les langages de programmation. Elle représente un *comportement* paramétré et prédéfini, qui peut être appelé par tout *processus* ayant accès à sa définition. Les *procédures* peuvent être prédéfinies dans des bibliothèques ou définies par l'usager.

Une *procédure* est définie comme un ensemble d'*actions* et peut comprendre des *états*, tout comme un *processus*. Un exemple d'utilisation des procédures est donné dans la figure 9/Z.100.

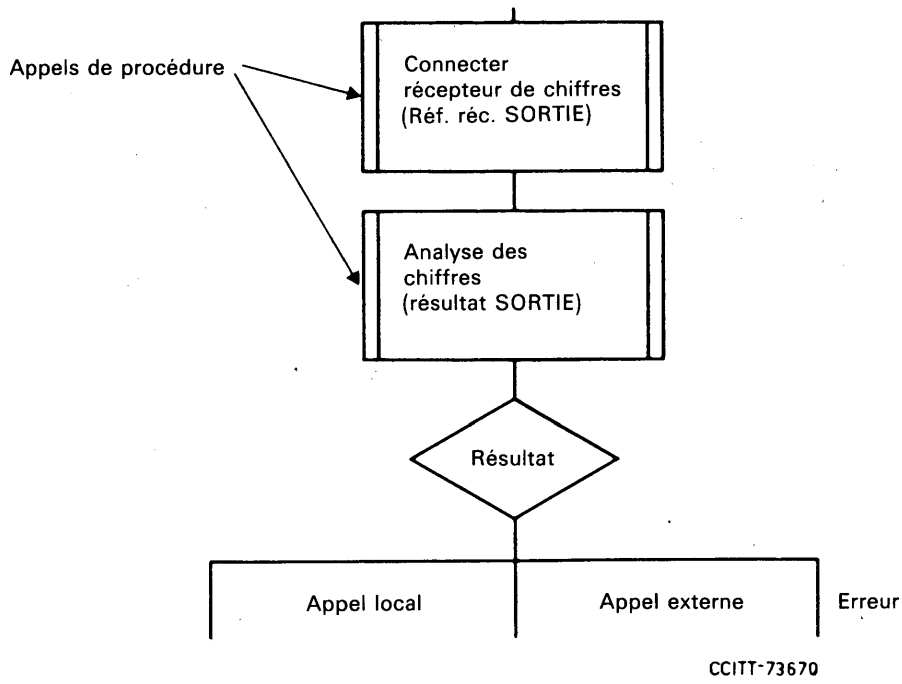
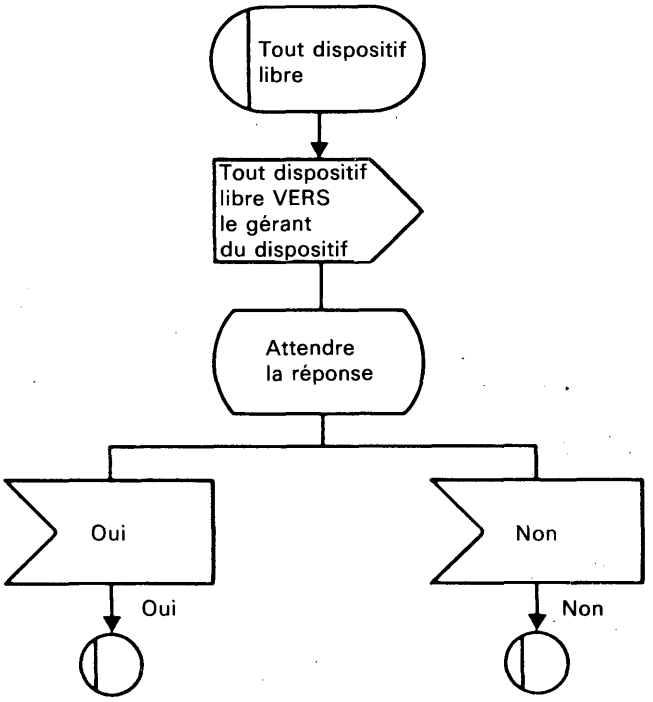


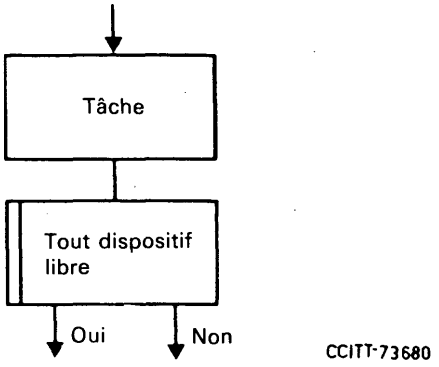
FIGURE 9/Z.100

Exemple d'utilisation des procédures dans une transition

Les représentations LDS peuvent aussi être structurées grâce à l'utilisation de *macros*. Une *macro* est un moyen syntaxique facilitant le dessin/la rédaction des représentations LDS et la compréhension. Une *macro* est un ensemble nommé d'éléments de syntaxe défini par l'usager. Chaque fois qu'une référence à une *macro* apparaît dans une description, on peut l'interpréter en la remplaçant par les éléments qui la définissent, c'est-à-dire qu'elle n'a pas de sémantique propre. Les *macros* peuvent être utilisées dans n'importe quelle représentation LDS, par exemple, pour les représentations de *processus*, pour les diagrammes structurels, etc. Un exemple d'utilisation d'une macro est donné dans la figure 10/Z.100.



a) Définition d'une macro



b) Utilisation d'une macro dans une transition

FIGURE 10/Z.100

Exemple d'utilisation d'une macro

Tous les concepts de structuration dans le LDS peuvent, bien entendu, être utilisés conjointement les uns avec les autres.

2.5 *Opérations composites*

Les *opérations composites* dans le LDS sont des notations abrégées normalisées fournies dans le langage pour simplifier la conception des *processus* LDS. Elles sont définies par rapport à d'autres opérations du LDS et doivent être interprétées comme si elles étaient remplacées par ces opérations.

Normalement, les *opérations composites* impliquent un échange d'*états* et de *signaux* cachés avec d'autres processus, raison pour laquelle il convient quelquefois de veiller aux effets de bord.

Les *opérations composites* fournies sont les suivantes:

Import/export des *valeurs de données*

Notation abrégée pour faire accéder aux *valeurs des données* locales aux autres processus au moyen d'un échange implicite de *signaux.*

Conditions de validation

Notation abrégée relative à la capacité soit d'accepter un *signal* en tant qu'*entrée* ou de *mettre en réserve* le *signal,* en fonction d'une *condition* qui peut contenir des *valeurs* importées. La modélisation comporte plusieurs *états* dans lesquels le *signal* est accepté comme une *entrée* et d'autres dans lesquels le *signal* est mis en réserve. L'*état* implicite est choisi après évaluation de la condition. L'opération peut aussi impliquer un échange de *signaux.*

Signaux continus

Notation abrégée pour quitter un *état* et entrer dans une *transition* lorsqu'une condition, qui peut utiliser des *valeurs importées,* est vérifiée. Un *signal continu* a une *priorité* inférieure à celle des *signaux* «normaux», et peut être utilisé pour susciter une transition lors de chaque changement de valeur de donnée externe. L'opération implique l'interfonctionnement d'un certain nombre d'*états* et de *signaux.*

En utilisant des *opérations composites*, on peut réduire le nombre d'*états* et de *transitions* dans une *définition de processus.*

2.6 *Le concept d'option dans le LDS*

Lorsque, en utilisant le LDS, on *spécifie* ou on *décrit* plusieurs applications analogues, on peut souvent appliquer la même *définition de processus* à plusieurs applications en ne modifiant celle-ci que très légèrement. Le concept d'*OPTION* offre la possibilité d'avoir des parties optionnelles dans une *définition de processus.*

De même, dans une *spécification*, ce concept permet de représenter des *comportements* également acceptables du point de vue du spécificateur. Le *système* effectif mettra en œuvre l'une de ces solutions.

3 Notes préliminaires à la spécification du langage

La spécification de langage du LDS figure dans les Recommandations Z.101, Z.102, Z.103 et Z.104. On trouvera dans les notes préliminaires ci-après l'explication des méthodes et des stratégies utilisées pour définir le langage ainsi que des directives relatives à l'interprétation des Recommandations.

3.1 *Stratégie utilisée dans la spécification du langage*

Le LDS donne le choix entre deux formes syntaxiques différentes pour présenter des descriptions LDS: une représentation graphique (LDS/GR) et une représentation textuelle (LDS/PR). Comme ces formes sont toutes deux des représentations concrètes de la même sémantique du LDS, elles sont équivalentes du point de vue sémantique. Pour s'assurer qu'elles sont bien équivalentes l'une à l'autre, et donc transformables les unes dans les autres, la définition de la sémantique du LDS est rigoureusement distincte des définitions des différentes syntaxes concrètes. Les relations entre les définitions sémantiques du LDS et les formes syntaxiques concrètes peuvent être représentées de la manière indiquée à la figure 11/Z.100.

La sémantique du LDS se définit par une *syntaxe abstraite* qui n'a pas de représentation concrète, associée à des règles de formation adéquate et d'interprétation. Cette définition est appelée modèle de langage commun. Chacune des formes syntaxiques concrètes possède ensuite une définition de sa propre forme syntaxique et de sa relation à la *syntaxe abstraite* (c'est-à-dire comment la transformer en forme abstraite). Avec cette méthode, il existe une seule définition de la sémantique du LDS; chacune des formes concrètes de représentation héritera de cette sémantique par le biais de sa relation à la *syntaxe abstraite.* Cette méthode garantit aussi que les formes syntaxiques concrètes sont équivalentes. Ainsi, les transformations étant possibles dans les deux sens, toute représentation peut être transposée dans l'autre forme au moyen de la *syntaxe abstraite.*

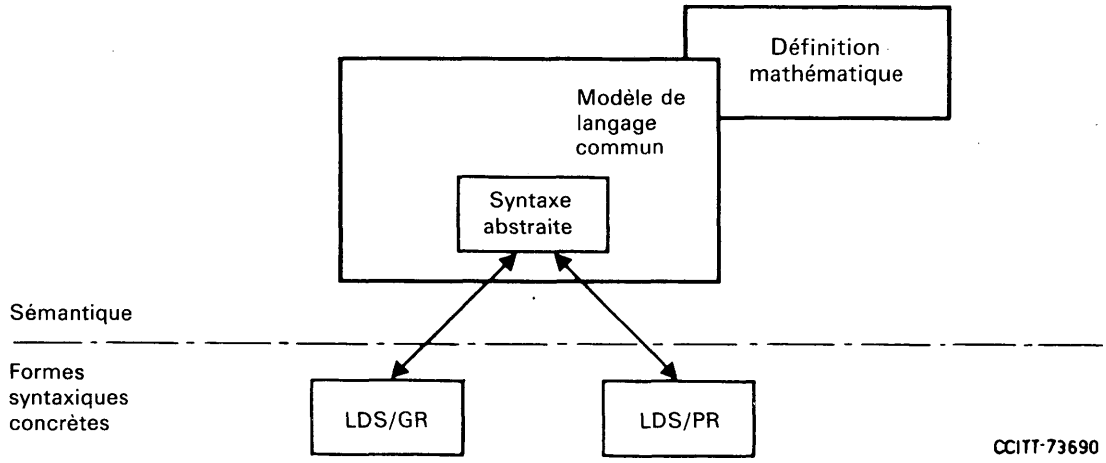


FIGURE 11/Z.100

Structure des spécifications de langage du LDS

Les règles d'interprétation du modèle de langage commun sont définies de façon opérationnelle, c'est-à-dire que la définition décrit comment les instances des concepts du LDS interprètent leur définition comme une «machine abstraite LDS». En outre, il existe une définition mathématique du LDS en sémantique dénotationnelle (mais non dans le cadre des Recommandations). La définition mathématique est étroitement liée à la *syntaxe abstraite* et à la structure du modèle de langage commun.

En application de cette stratégie, les Recommandations comprendront, pour chaque concept LDS, d'abord une définition de la *syntaxe abstraite* et de ses règles, puis des règles d'interprétation du concept. Suivront, enfin, les formes syntaxiques concrètes qui représentent le concept.

3.2 *Terminologie*

Pour chaque concept LDS, le même terme sera utilisé dans toutes les Recommandations. Toutefois, afin de différencier les cas de termes se rapportant à des concepts LDS de ceux dans lesquels les termes ont une portée plus générale, tous les termes de concept LDS seront écrits en *italique.*

On trouvera dans l'annexe A des Recommandations Z.100 à Z.103, un glossaire de tous les termes LDS. Ce glossaire donne une brève explication de chaque terme et un renvoi au paragraphe où se trouve sa définition.

3.3 *Définition du LDS/GR*

Le LDS est défini de la manière suivante:

- en premier lieu, on définit la forme et le contenu des symboles;
- en deuxième lieu, on définit, le cas échéant, les règles de connexion, c'est-à-dire les compositions de symboles autorisées;
- en dernier lieu, figurent les relations entre la *syntaxe abstraite* et le modèle de langage commun.

3.4 *Définition du LDS/PR*

Le *LDS/PR* est défini grâce à des *diagrammes de syntaxe* et à des règles supplémentaires en langage naturel. Les définitions sont établies de la manière suivante:

- en premier lieu, la syntaxe est définie au moyen de diagrammes et de texte;
- cette définition est suivie des relations entre la *syntaxe abstraite* et le modèle de langage commun.

3.4.1 *Diagramme de syntaxe*

Un *diagramme de syntaxe* se compose de symboles terminaux et de symboles non terminaux connectés par des lignes de liaison.

Un symbole terminal contient un caractère ou une suite de caractères. Les règles de production sont les suivantes: lorsque ces caractères suivent un trajet, il doivent apparaître dans le texte *LDS/PR.*

Un symbole non terminal renvoie à un autre *diagramme de syntaxe* dont le nom apparaît dans le symbole. Les règles de production sont les suivantes: lorsqu'un symbole non terminal apparaît dans un diagramme, le trajet conduit au diagramme mentionné et ne quitte pas le symbole non terminal avant d'avoir quitté le diagramme mentionné.

Pour tous les symboles terminaux, les symboles non terminaux et les *diagrammes de syntaxe*, il y a une seule ligne de liaison qui y conduit et une seule ligne de liaison qui en sort.

Les symboles graphiques sont représentés dans la figure 12/Z.100.

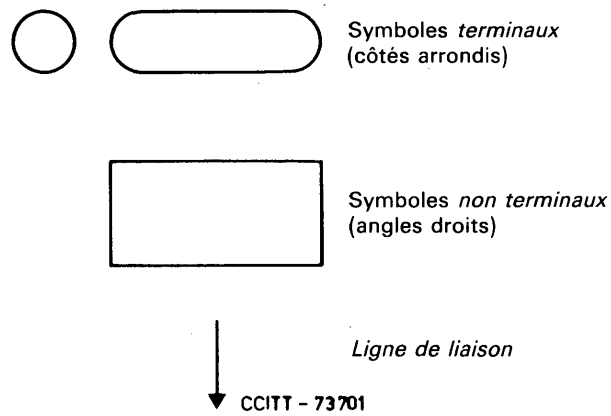


FIGURE 12/Z.100

Symboles utilisés dans les diagrammes de syntaxe

La définition de la syntaxe LDS/PR se compose, au niveau «le plus élevé», d'un diagramme de syntaxe, le SYSTÈME. Ce diagramme renvoie au moyen de non-terminaux, à un ensemble d'autres diagrammes. Tout trajet qui commence avec la ligne de liaison conduisant à ce diagramme et sortant du diagramme produit, sur son parcours, un texte *LDS/PR*. Le texte est syntaxiquement correct si les règles de la syntaxe LDS ont été respectées.

3.5 *Eléments syntaxiques communs pour le LDS/GR et le LDS/PR*

Pour certains éléments syntaxiques, la même syntaxe concrète est utilisée pour le LDS/GR et le LDS/PR.

3.6 *Structure de la Recommandation relative au LDS*

Quatre Recommandations (Z.101 à Z.104) suivent la présente Recommandation. Elles ont pour objet d'aider les usagers à choisir un sous-ensemble de LDS convenant à leurs applications et à leur méthodologie.

Le LDS peut être appliqué de nombreuses manières, et répondre à différents objectifs et méthodes. Le sous-ensemble minimum de LDS qui peut être choisi est indiqué dans la Recommandation Z.101.

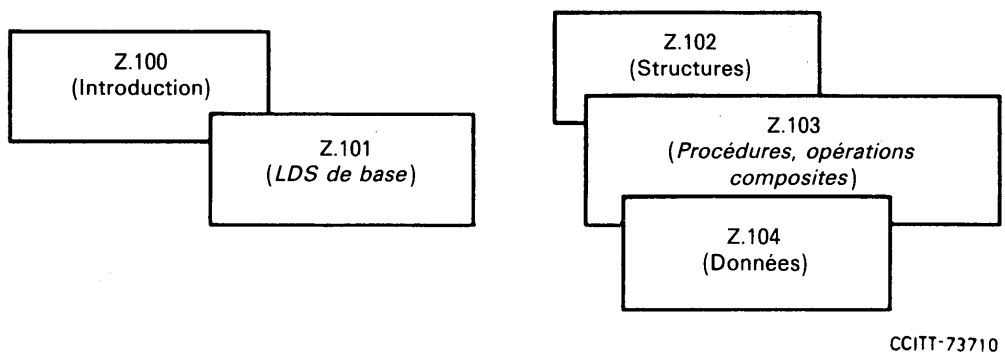


FIGURE 13/Z.100

Structures des Recommandations relatives au LDS

Les Recommandations Z.102 à Z.104 contiennent des extensions au sous-ensemble minimum du LDS qui devraient être appliquées le cas échéant. Les extensions peuvent être appliquées dans une combinaison quelconque.

On trouvera dans les annexes des exemples d'utilisation des concepts définis dans les Recommandations. Des exemples plus détaillés d'utilisation du LDS figurent également dans les Directives pour les usagers du LDS.

On peut résumer brièvement comme suit la teneur des Recommandations ci-après:

Z.101 Cette Recommandation définit les concepts fondamentaux du LDS. Elle forme le sous-ensemble minimum du LDS à appliquer. Ce sous-ensemble du LDS suffit pour décrire le *comportement* des *systèmes.*

Z.102 Cette Recommandation définit des concepts structuraux supplémentaires, utilisés pour décrire des *systèmes* grands et/ou complexes. Ces concepts peuvent servir tant pour décrire la *structure* effective d'un *système* que pour décrire le *système* à plusieurs *niveaux d'abstraction.*

Z.103 Cette Recommandation définit le concept de *procédure*, les *opérations composites*, le concept de *macro*, le concept d'*option* ainsi que les extensions graphiques en fonction des états (*LDS/PE*). Ces concepts sont définis indépendamment les uns des autres et peuvent être appliqués conjointement dans une combinaison quelconque.

Z.104 Certains concepts de données, qui sont considérés comme prédéfinis dans les Recommandations Z.100 à Z.103, sont définis dans la Recommandation Z.104. Il convient de noter que les *données* peuvent être utilisées de façon tout à fait informelle dans le LDS.

La Recommandation Z.104 définit également le concept *de type abstrait de données* pour le LDS.

Outre ces Recommandations, un ensemble de documents auxiliaires est disponible; ils décrivent et expliquent le langage sans avoir pour autant qualité de Recommandation. Certains de ces documents figurent en annexe aux Recommandations. Les documents auxiliaires sont les suivants:

La définition formelle du LDS

Ce document contient la définition mathématique de la sémantique du LDS. La définition est rédigée en sémantique dénotationnelle (VDM, META IV). Elle fera l'objet d'un document distinct.[1)]

Le glossaire du LDS

Ce document contient tous les termes relatifs au LDS. Chaque terme est suivi d'une brève explication et d'un renvoi à sa définition dans le texte des Recommandations. Ce document constitue l'annexe A aux Recommandations, et se trouve dans le Fascicule VI.11 du Livre rouge.

Le résumé de syntaxe abstraite du LDS

Ce document contient un résumé de la *syntaxe abstraite* complète pour le langage. La *syntaxe abstraite* est décrite sous forme abrégée, du type BNF. Le document constitue l'annexe B aux Recommandations, et se trouve dans le Fascicule VI.11 du Livre rouge.

Le résumé de syntaxe concrète du LDS

Ce document contient un résumé de toutes les syntaxes concrètes du LDS, c'est-à-dire la représentation graphique (LDS/GR), l'extension graphique orientée vers les états pour la forme graphique (LDS/PE) et la représentation textuelle (LDS/PR). Ce document constitue l'annexe C aux Recommandations, et se trouve dans le Fascicule VI.11 du Livre rouge.

Les directives pour les usagers du LDS

Ce document donne des exemples d'utilisation du LDS, avec des explications (sans définir le langage). Il contient un certain nombre d'exemples et d'analyses de différentes utilisations du LDS. Certains concepts qui sont définis dans les Recommandations et qu'il convient d'utiliser avec une attention particulière, sont également étudiés dans les directives pour les usagers. Ce document constitue l'annexe D aux Recommandations et se trouve dans le Fascicule VI.11 du Livre rouge.

Le cours LDS

Ce document est conçu comme un manuel d'instruction pour l'utilisation du LDS. Il est disponible en tant que document distinct.[2)]

Enfin à l'intérieur de la dernière page de couverture de ce fascicule, on trouvera deux gabarits utilisables pour dessiner les éléments graphiques du LDS. Ils contiennent tous les symboles graphiques ainsi que leur mode de présentation recommandés.

---

1) La définition formelle du LDS, peut être obtenue auprès du Secrétariat général de l'Union internationale des télécommunications, Section des ventes, place des Nations, CH-1211 Genève 20, (Suisse).

2) Le cours LDS peut être obtenu auprès du Secrétaire général de l'UIT, Système international d'échange de l'UIT en matière de formation professionnelle, Département de la Coopération Technique, Division de formation professionnelle, place des des Nations, CH-1211 Genève 20, (Suisse).

LANGAGE DE SPÉCIFICATION ET DE DESCRIPTION FONCTIONNELLES (LDS) DE BASE DU CCITT

1 Introduction

La présente Recommandation définit le Langage de spécification et de description fonctionnelles (LDS) de base du CCITT. Le LDS est fondé sur la notion de machines à *états finis*, communiquant entre elles, appelées *processus*. Un *système* LDS est un ensemble de *blocs*. Les *blocs* sont connectés les uns aux autres et à l'*environnement* par des canaux. Chaque *bloc* contient un ou plusieurs *processus*. Ces *processus* communiquent entre eux par des signaux et sont censés s'exécuter en parallèle.

Dans la définition du LDS, on a jugé utile de définir tout d'abord un modèle LDS commun. Ce modèle commun constitue une base abstraite pour les *syntaxes concrètes*, et le lien entre ces syntaxes. Les *syntaxes concrètes* ne sont rien d'autre que des variantes de représentation des concepts du LDS. Il existe actuellement deux *syntaxes concrètes*: le LDS/GR et le LDS/PR. Comme ces deux syntaxes représentent les mêmes concepts LDS, il est possible de mettre en correspondance un système utilisant une des formes de la *syntaxe concrète* du LDS avec l'autre *syntaxe concrète*.

Dans la présente Recommandation, pour définir le langage, on commence par définir le modèle commun, puis les syntaxes concrètes LDS/GR et LDS/PR.

2 Modèle commun du langage

2.1 *Introduction au modèle commun*

Dans le LDS, un *système* est un ensemble de *blocs* connectés les uns aux autres et à l'*environnement* du *système*, au moyen de canaux (voir la figure 1/Z.101). Les *canaux* sont unidirectionnels.

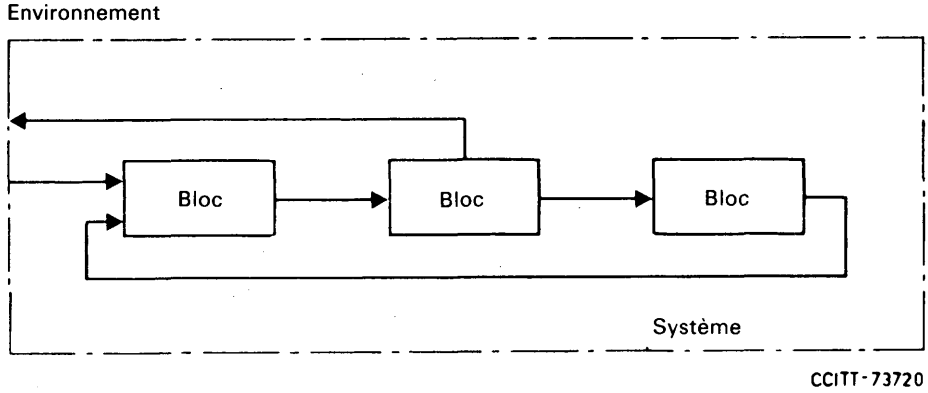


FIGURE 1/Z.101

Modèle du LDS

Le comportement de chaque *bloc* est modélisé par un ou plusieurs *processus*. Un *processus* est défini par une *définition de processus*.

Les *processus* sont en interaction avec d'autres processus, ou avec l'environnement, au moyen de signaux. Un signal est un *flux de données* qui *véhicule* de l'information entre des *processus*. Lorsqu'un *processus* fait sortir un *signal*, celui-ci est transporté jusqu'au *processus* auquel il est destiné. Le mécanisme de transport pour les signaux véhiculés entre des *processus* du même *bloc* est le même que pour des signaux *véhiculés* entre des *processus* faisant partie de blocs différents. Les *canaux* représentent les trajets de transport pour les signaux échangés entre les *blocs*.

Quand un signal arrive au *processus* auquel il est destiné, il est *retenu* à l'extérieur de ce *processus* jusqu'à ce que le *processus* soit prêt à recevoir le *signal d'entrée*. Un *signal* est absorbé quand le *processus* de destination reçoit le *signal*.

Le LDS modélise des *systèmes* ouverts, ce qui signifie que le *système* peut être en interaction avec son *environnement*. Cette interaction a lieu exclusivement au moyen de *signaux véhiculés* dans des *canaux* qui aboutissent à l'*environnement* ou qui en proviennent. On admet que l'*environnement* agit à la manière du LDS, c'est-à-dire qu'on peut considérer que l'*environnement* contient un *processus* qui émet des *signaux* vers les *canaux* conduisant jusqu'au *système* et qui reçoit des *signaux* en provenance des *canaux* sortant du *système*.

Pendant sa durée de vie, un *processus* se trouve soit dans un *état* (attente de la réception d'un signal faisant partie d'un ensemble de *signaux*), soit dans une *transition* (exécution d'une suite d'actions). Quand le *processus* se trouve dans un *état*, il peut recevoir seulement un ensemble spécifié de signaux. Si l'un de ces *signaux d'entrée* est *retenu* à l'extérieur du *processus*, il est reçu par le *processus*. La réception de l'entrée fait que les *données* transportées par le signal sont accessibles pour le *processus*, et *déclenche* une *transition*. Pendant la durée d'une *transition*, les *données* du *processus* peuvent être manipulées et des *signaux* peuvent être émis. La *transition* s'achève quand le processus entre dans un nouvel *état* ou quand il y a une action d'*arrêt*.

Une action d'*arrêt* met fin à l'existence du *processus*.

Le concept de temps absolu est le même pour le *système* et son *environnement*, dont il constitue un paramètre commun.

2.2 *Syntaxe abstraite*

Définition de système

Une *définition de système* contient un *nom de système*, une ou plusieurs *définitions de bloc*, un ensemble de *définitions de canal* et un ensemble de définitions de signal.

Une *définition de système* contient la *définition de signal* pour chaque *nom de signal* contenu dans la *liste de signaux* associée à chaque *définition de canal*.

Définition de bloc

Une *définition de bloc* contient un *nom de bloc*, une ou plusieurs *définitions de processus*, et peut contenir des définitions de signal.

Chaque *définition de bloc* contient la *définition de signal* pour chaque type de *signal* qui est échangé entre les *processus* à l'intérieur du *bloc*.

Définition de canal

Une *définition de canal* contient un *nom de canal*, un *identificateur de définition de bloc* d'origine, un *identificateur de définition de bloc* de destination et une *liste de signaux*. La *liste de signaux* contient les identificateurs de chaque type de signaux pouvant être véhiculés par le *canal*.

L'*identificateur de définition de bloc* d'origine et l'*identificateur de définition de bloc* de destination associés à une *définition de canal* doivent être différents. Chacun d'eux doit être soit l'*identificateur* d'une *définition de bloc* dans la *définition de système*, soit l'*environnement*.

La *liste des signaux* associés à la *définition de canal* contient au moins un *identificateur de signal*.

Définition de signal

Une *définition de signal* contient un *nom* de signal et peut contenir une liste de *noms* de *types* de *données*.

Définition de processus

La *définition de processus* contient un *nom de processus*, une paire de nombres entiers et un *graphe de processus*. Elle peut contenir une liste de *paramètres formels*, des *définitions de variable* et des *définitions de visibilité*.

Graphe de processus

Un *graphe de processus* est un graphe dont les *nœuds* sont connectés entre eux par des *arcs* orientés. Un *arc* qui entre dans un *nœud* est appelé *arc* entrant et un *arc* qui sort d'un nœud, *arc* sortant. Un *nœud* pour lequel un arc est un *arc* entrant suit le *nœud* pour lequel le même *arc* est un *arc* sortant.

On distingue les catégories suivantes de *nœud*:

Nœud d'état
Nœud d'entrée
Nœud de tâche
Nœud de sortie
Nœud de décision
Nœud de départ
Nœud d'arrêt
Nœud de demande de création

Les règles suivantes définissent la connectivité d'un *graphe de processus*:

1) Chaque *graphe de processus* contient un nœud de départ, et un seul. Le nœud de départ est suivi d'une *chaîne de transition.* Le nœud *de départ* ne suit aucun autre nœud.

2) Une *chaîne de transition* peut se présenter sous l'une des formes suivantes:

   a) un zéro suivi d'un *nœud d'état* ou d'un *nœud d'arrêt*;
   b) une chaîne d'action suivie d'une *chaîne de transition*;
   c) un *nœud de décision.*

3) Une *chaîne d'action* peut se présenter sous l'une des formes suivantes:

   a) un *nœud de tâche*;
   b) un *nœud de sortie*;
   c) un *nœud de demande de création.*

4) Un *nœud de décision* est suivi de deux ou plusieurs *arcs de décision.*

5) Un *arc de décision* est un *arc* nommé, suivi d'une *chaîne de transition.*

6) Un *nœud d'état* est suivi d'un ou plusieurs *nœuds d'entrée.*

7) Un *nœud d'entrée* suit un *nœud d'état* et un seul.

8) Un *nœud d'entrée* est suivi d'une *chaîne de transition.*

9) Le *nœud d'arrêt* n'est suivi d'aucun *nœud.*

10) Chaque *graphe* de processus comporte au maximum un *nœud d'arrêt.*

11) Chaque *nœud* peut être atteint à partir du *nœud de départ.*

Nœud d'état

Un *nœud d'état* contient un *nom d'état* et peut contenir un *ensemble de signaux de mise en réserve.* A l'intérieur d'un *graphe de processus,* les *nœuds d'état* ont des noms différents.

Nœud d'entrée

Un *nœud d'entrée* contient un *identificateur de signal* et peut contenir un ensemble ordonné d'*identificateurs* de *variable.*

Ensemble de signaux de mise en réserve

Un *ensemble de signaux de mise en réserve* contient des *identificateurs de signal.*

Nœud de tâche

Un *nœud de tâche* contient soit une *séquence d'instructions*, soit un *texte informel.*

Instruction

Une *instruction* peut être soit une *instruction d'initialisation,* soit une *instruction de réinitialisation,* soit une *instruction d'affectation.*

Instruction d'initialisation

Une *instruction d'initialisation* contient une *expression* de *temps* et un *identificateur de temporisation.*

Instruction de réinitialisation

Une *instruction de réinitialisation* contient un *identificateur de temporisation.*

Instruction d'affectation

Une instruction d'affectation contient un *identificateur de variable*, un *opérateur d'assignation* et une *expression.*

Nœud de sortie

Un *nœud de sortie* contient un *identificateur de signal*, une *expression de destination* et peut contenir *une liste de paramètres effectifs.*

Nœud de décision

Un *nœud de décision* contient une *question* et est associé à deux arcs sortants, au moins. A chaque arc sortant est associé un ensemble constitué par une ou plusieurs réponses à la *question.* Chaque réponse possible à la question doit être associée à un *arc* et à un seul. Une question est une *expression* ou un texte informel. Une réponse est soit un *identificateur de valeur,* soit un *texte informel.*

Nœud de demande de création

Un *nœud de demande de création* contient un *identificateur de définition de processus* et peut contenir *une liste de paramètres effectifs.*

Type de données

Il existe des *types de données* prédéfinis pour les *types* suivants: *entiers naturels, nombres entiers, nombres réels, caractère, chaîne de caractères, booléen, temps, durée, temporisation* et *identificateur d'instance de processus.* Ces différents types prédéterminés sont définis dans la Recommandation Z.104.

Définition de variable

Une définition de variable contient un *nom de variable* et un *identificateur de type de données*; elle peut contenir un *attribut d'apparition.*

Liste de paramètres formels

Une *liste de paramètres formels* est un ensemble donné de *paramètres formels.*

Paramètre formel

Un *paramètre formel* contient un *nom de paramètre formel* et un *identificateur de type.*

Liste de paramètres effectifs

Une liste de *paramètres effectifs* est un ensemble ordonné de *paramètres effectifs.*

Paramètre effectif

Un *paramètre effectif* est une expression.

Définition de visibilité Une définition de *visibilité* contient un *identificateur de variable*, un *identificateur de type de données* et un *identificateur de définition de processus.*

La *variable* doit avoir un attribut d'*apparition* dans la *définition de processus* à laquelle se rapporte l'*identificateur de définition de processus.* La *définition de processus* d'apparition correspondante doit appartenir au même *bloc* que la *définition de visibilité du processus.*

Expression

Une *expression* est soit un *identificateur de valeur*, soit un *identificateur de variable*, soit une *opération.*

Opération

Une *opération* contient soit une *expression de visibilité*, soit un *opérateur* et une liste d'une ou plusieurs *expressions.*

Expression de visualisation

Une *expression de visualisation* se compose d'un *opérateur de visualisation* accompagné d'un *nom de variable* et d'un *identificateur de processus.*

2.3 *Règles d'interprétation*

2.3.1 *Système*

Un *système* est une entité concrète, par exemple un central téléphonique; c'est une forme de réalisation d'un type de système défini par une *définition de système.* Un *système* est séparé de son *environnement* par une *frontière* de *système* et contient un ensemble de *blocs.* La communication entre le *système* et son *environnement,* ou entre les *blocs* intérieurs au *système,* ne peut se faire qu'au moyen de *signaux.* A l'intérieur d'un *système,* ces signaux sont véhiculés dans des *canaux.* Les *canaux* relient les *blocs* entre eux ou à la *frontière du système.*

Le *système* possède une fonction sans paramètre du *type temps* appelée NOW qui donne le *temps* actuel. Le *temps* actuel est immédiatement disponible dans tout le *système* et dans son *environnement.* NOW peut être utilisé dans des *expressions* apparaissant dans tout *processus du système.*

2.3.2 *Canal*

A l'intérieur du *système*, il existe un *canal* pour chaque *définition* de canal incluse dans la *définition de système.* Un canal est un moyen de transport pour des *signaux.* Ce moyen de transport est unidirectionnel. Les points extrêmes du canal sont soit un *bloc,* soit la *frontière du système.* Un des points extrêmes du *canal,* au moins, doit être dans un *bloc.* Si les deux points extrêmes sont dans des *blocs,* ces *blocs* doivent être différents. La *définition de canal* contient la liste de tous les *signaux* pouvant être *véhiculés* sur le *canal.*

Quand un *signal* est *émis* sur le *canal*, il est véhiculé jusqu'au *bloc* de destination. L'ordre de sortie des signaux sur le *canal* est le même que l'ordre des *signaux* reçus du *canal*. Si deux ou plusieurs *signaux* arrivent simultanément sur un *canal*, ils sont ordonnés arbitrairement.

2.3.3 *Bloc*

A l'intérieur du *système*, il existe un *bloc* pour chaque *définition de bloc* incluse dans la *définition de système*. Un *bloc* est un objet qui, par ses dimensions, permet un traitement facile et dans lequel un ou plusieurs *processus* peuvent être interprétés. Il existe deux mécanismes de communication entre les *processus* à l'intérieur d'un même *bloc*: les *signaux* et les *valeurs partagées*.

Quand un *signal* atteint le *bloc* après transmission par un *canal*, le *bloc* livre le *signal* à l'*accès d'entrée* du *processus* désigné par l'*identificateur de processus* du *signal*.

Quand un *signal* sort par un *processus* faisant partie du *bloc*, il est livré au *processus* indiqué par l'*identificateur de processus* contenu dans le *signal*. Si le *processus* indiqué se trouve dans le même *bloc*, le *bloc* délivre le *signal* à son *accès d'entrée*. Si le *processus* indiqué fait partie d'un autre *bloc*, le *bloc* délivre le *signal* au *canal* qui est capable de véhiculer ce *signal*.

Les *valeurs partagées* permettent à un *processus* de *voir* une *variable* apparue dans un autre *processus*. Le *processus* d'apparition est seul autorisé à modifier la *valeur* de la *variable*. Le *processus* de visualisation reçoit la *valeur* actuelle de la *variable* apparue en appliquant un *opérateur de visualisation*.

2.3.4 *Signal*

Un *signal* est une suite de *données* véhiculant une information entre des *processus*; c'est une instantiation d'un *type de signal* défini par une *définition de signal*. Un *signal* peut être émis par l'*environnement du système* ou par un *processus*; il se dirige toujours vers un *processus* ou vers l'*environnement*.

Chaque signal contient un *identificateur de signal* dans la définition de *signal*, un *identificateur* d'*instance* de *processus* d'origine et un *identificateur d'instance de processus* de destination. De plus, des *variables* dans un *signal* peuvent véhiculer d'autres *valeurs*. Dans un *signal*, il y a une *variable* pour chaque *nom* dans la *liste de types de données* dans la *définition de signal*.

2.3.5 *Processus*

Un *processus* est une machine à états finis communicante; c'est une forme de réalisation d'un *type de processus* défini par une *instantiation*. A l'intérieur d'un bloc, il peut y avoir zéro, un ou plusieurs *processus* pour chaque définition de *processus*. Les *processus* peuvent donc exister à partir de l'instant de création du *système*; ils peuvent aussi être créés par des *actions de demande de création*, et cesser d'exister sous l'effet d'*actions d'arrêt*. Un *processus* peut se dérouler indépendamment des autres *processus* du *système* et concurremment avec ceux-ci.

Tous les processus du *système* possèdent quatre *variables* prédéfinies de *type d'identificateur* de processus appelés: SELF (lui-même), ASCENDANT (parent), OFFSPRING (descendant), SENDER (émetteur). Ces *variables* sont des *identificateurs d'instance* de processus pour:

- le *processus* (SELF);
- le *processus* de créateur (PARENT);
- le *processus* créé le plus récemment (OFFSPRING);
- le *processus* en provenance duquel le dernier *signal entrant* a été reçu (SENDER).

Ces *variables* peuvent être utilisées dans des *expressions* mais il n'est pas possible de leur affecter explicitement une *valeur*. Pour tous les *processus* présents au moment de l'*initialisation* du système, PARENT reçoit la même *valeur* distincte, qui est différente de celle attribuée à SELF pour un *processus* quelconque.

Les *signaux* qui se dirigent vers le *processus* sont des *signaux entrants*, les signaux qui partent du *processus* sont des *signaux sortants*. Un *signal entrant* est une entité destinée à activer le *processus* et à lui fournir de l'information. Un *signal sortant* est destiné à activer un autre *processus* et à lui fournir de l'information.

L'ensemble des *identificateurs de signal* qui se présentent attachés aux *nœuds d'entrée* du *graphe de processus* représente l'ensemble des *identificateurs* de *signaux entrants valables* pour la *définition de processus* correspondante. Pour chaque *état*, tous les *identificateurs de signal entrant* sont inclus soit dans un *ensemble de signaux de mise en réserve*, soit dans un *nœud* d'entrée.

La paire de *nombres entiers* contenus dans la *définition de processus* définit le nombre d'*instances* du *processus* qui sont créées lors de la création du *système* et le nombre maximum d'*instances* simultanées du même *type de processus*. Lors de la création d'un *système*, les *processus* initiaux sont créés en ordre aléatoire et aucun des *paramètres effectifs* ne passe dans le *processus*.

Quand un *processus* est créé, il reçoit un *accès d'entrée* vide, et il y a alors création de *variables* ayant des *valeurs indéfinies*. Le *processus commence* ensuite à se dérouler en interprétant le *nœud de départ* dans le *graphe de processus*.

Quand un *signal entrant valable* parvient au *processus*, il est placé dans l'*accès d'entrée* du *processus*. Chaque *processus* contient un seul *accès d'entrée*. L'*accès d'entrée* peut retenir un nombre quelconque de *signaux entrants* de sorte que plusieurs *signaux entrants* sont mis dans une file d'attente pour le *processus*. L'ensemble des *signaux retenus* est ordonné dans la file d'attente, dans l'ordre de leurs instants d'arrivée. Si deux ou plusieurs signaux arrivent simultanément, ils sont ordonnés arbitrairement. A l'*accès d'entrée* est attaché un ensemble de *temporisateurs* éventuellement vide. Chaque temporisateur contient une *valeur du type* de *temps*.

Le *processus* peut être soit mis en attente, tout en occupant un *état*, soit en activité, tout en effectuant une *transition*. Pour chaque *état*, il existe un *ensemble de signaux de mise en réserve*. Dans le cas d'attente dans un *état*, le premier *signal entrant* dont l'*identificateur* ne figure pas dans l'ensemble des *signaux de mise en réserve* est extrait de la file d'attente, et reçu par le *processus*. L'*accès d'entrée* va comparer en permanence la fonction NOW avec le temporisateur – s'il en existe un – ayant la *valeur* la plus basse supérieure à zéro. Lorsque la *valeur* NOW est supérieure ou égale à la *valeur* de ce temporisateur, un *signal* ayant le même *nom* que ce temporisateur est placé dans la file d'attente, puis le temporisateur reçoit la valeur zéro.

Quand un *processus* est en attente, il se trouve toujours dans un *état* unique, désigné par le *nœud d'état* correspondant dans le *graphe de processus*. A la réception d'un *signal entrant*, le *processus* devient actif; il interprète le *nœud d'entrée* qui possède le même *nom* que le *signal d'entrée* et il effectue une *transition* qui conduit à un nouvel *état*.

2.3.6 *Graphe de processus*

Un *nœud de départ* est interprété comme une *action de départ* et celle-ci suscite l'interprétation du *nœud* qui suit le *nœud de départ*.

Un *nœud d'entrée* est interprété comme une *action d'entrée* qui reçoit et absorbe le *signal* donné, après quoi il y a interprétation du *nœud* qui vient à la suite du *nœud d'entrée*. Du fait de l'absorption du *signal*, l'information transportée par celui-ci est mise à la disposition du *processus*. Les *variables* contenues dans le *nœud d'entrée* reçoivent les *valeurs* des *variables* correspondantes *du signal*. Si le *nœud d'entrée* ne contient pas de variable correspondant à une *variable* du signal, la *valeur* de cette *variable* est mise au rebut. La valeur attribuée à SENDER, dans le *processus* de réception, est celle de l'*identificateur d'instance* du *processus d'origine* fournie par le signal.

Un *nœud de tâche* est interprété comme une *action de tâche*. L'*action* de tâche est l'interprétation d'une suite d'instructions ou d'un texte informel. Après l'exécution de cette *action*, il y a interprétation du *nœud* qui vient à la suite du *nœud de tâche*.

Une *instruction d'initialisation* entraîne une instruction de *réinitialisation* sur le *temporisateur* donné dans *l'instruction* et affecte cette *valeur du temps* au *temporisateur*.

Une *instruction de réinitialisation* met à zéro la *valeur* du temps du temporisateur. Si des signaux ayant le même *nom* que le temporisateur sont présents dans la file d'entrée, ils sont éliminés de la file d'entrée et rejetés. Tous les *signaux* qui se trouvent dans la file d'attente et qui ont le même *identificateur* que l'*identificateur de signal* de temporisateur sont éliminés de la file d'attente et mis au rebut.

Une *instruction d'affectation* est interprétée de manière informelle telle que la *valeur* de la *variable* de cette instruction prend la valeur de l'*expression* de cette *instruction d'affectation*.

Un *nœud de sortie* est interprété comme une *action de sortie* qui crée un signal et qui remet ce signal au *bloc*. Cette opération est suivie de l'interprétation du *nœud* qui vient à la suite du *nœud de sortie*. Le *signal de sortie* est une instantiation d'un *type de signal* défini par la *définition du signal*, indiquée par l'*identificateur de signal* dans le *nœud de sortie*. Les *variables* du *signal* reçoivent les *valeurs* des *paramètres effectifs* contenues dans le *nœud de sortie*. Si, dans le *nœud de sortie*, il n'y a pas de *paramètre effectif* correspondant à une *variable* d'un signal, cette *variable* a alors une valeur indéfinie. L'*identificateur d'instance de processus* d'origine véhiculé par le signal reçoit la valeur de la *variable* SELF. L'élément de données de processus de *destination* reçoit la *valeur* de l'*expression* de processus de destination contenue dans le *nœud de sortie*.

Un *nœud de décision* est interprété comme une *action de décision* qui répond à une *question*. L'*arc* qui fait correspondre la *réponse* à la *question* est choisi et cette opération est suivie de l'interprétation du *nœud* qui suit cet *arc*.

Un *nœud d'état* est interprété comme la terminaison d'une *transition*, l'opération consistant à donner au *processus* un nouvel *état*, désigné par le *nom* contenu dans le *nœud*. L'*accès d'entrée* est informé que le *processus* est maintenant en attente dans un *état*, et il reçoit en même temps l'*ensemble des signaux de mise en réserve* attaché au *nœud d'état* qui a été interprété. Le *processus* attend ensuite qu'on lui présente un nouveau *signal d'entrée*.

Un *nœud d'arrêt* est interprété comme la terminaison immédiate du *processus*. Cela signifie que les *signaux entrants* retenus dans l'*accès d'entrée* sont mis au rebut et que les *variables* créées pour le *processus*, l'*accès d'entrée* et le *processus* cessent d'exister.

Un *nœud de demande de création* est interprété comme une *action de demande de création*. L'*action de demande de création* provoque la création d'un *processus* dans le même *bloc*. La *définition* du *processus* se trouve dans le même *bloc* et elle est identifiée par l'*identificateur de processus* dans le *nœud de demande de création*. Dans le cadre de l'*action de demande de création*, la *variable* PARENT du *processus* créé reçoit la *valeur* de la *variable* SELF du *processus* créateur. La *variable* SELF du *processus* créé et la *variable* OFFSPRING du *processus* créateur reçoivent toutes deux la même *valeur* unique d'*identificateur d'instance de processus*. Les *paramètres formels* du *processus* nouvellement créé reçoivent les *valeurs* des *paramètres effectifs* contenus dans le *nœud de demande de création*.

2.3.7 *Type de données*

Les types prédéfinis sont utilisés avec leur sens normal.

2.3.8 *Variable*

Une *valeur* peut être affectée à une *variable*. Le contenu de la *valeur affectée* à une *variable* peut être extrait ultérieurement de la *variable*. Une *variable* possède aussi un *type de données* qui restreint la classe des *valeurs* pouvant être affectées à la variable.

A l'intérieur d'un *processus*, il existe une *variable* pour chaque *définition de variable* dans la *définition de processus*. La *valeur* d'une *variable* peut être modifiée uniquement par le *processus* qui définit la *variable*. La *valeur* d'une *variable* est connue seulement du *processus* qui définit la *variable*, sauf si la *variable* possède l'*attribut d'apparition*. L'*attribut d'apparition* permet à d'autres *processus* du *bloc* de *voir* une *variable*. Les *variables* ont la même durée de vie que le *processus* déclarant (cela signifie qu'elles sont créées en même temps que le *processus* déclaratoire et qu'elles cessent d'exister lorsque ce *processus* cesse lui-même d'exister).

Dans un *signal*, il y a une *variable* anonyme pour chaque apparition de chaque *nom de type de données* dans la *définition du signal*. La *valeur* de la *variable* ne peut être affectée que dans le *nœud de sortie* qui a créé le *signal* et ne peut être connue que dans le *nœud d'entrée* qui reçoit et consomme le *signal*. Ces *variables* ont une durée de vie liée à la durée de vie du *signal*.

2.3.9 *Expression*

Une *expression* est interprétée de manière non formelle comme produisant une *valeur*.

2.3.10 *Définition de visibilité*

Une *définition de visualisation* définit un *nom de variable* qui peut être utilisé uniquement dans une *expression de visibilité*, pour obtenir la *valeur* d'une *variable* appartenant à un autre *processus*.

2.3.11 *Expression de visibilité*

La *valeur* d'une *expression de visibilité* est la *valeur* de la *variable* identifiée par le nom de *variable* et l'*identificateur d'instance de processus* dans l'*expression de visibilité*.

3 LDS/GR

Une *définition du système* est représentée dans la syntaxe LDS/GR par:

- un *diagramme d'interaction de blocs* qui contient le *nom de système* et les *définitions de canal* et qui identifie les *définitions de bloc*. Le *diagramme d'interaction de blocs* identifie également les *définitions de processus* qui modélisent le comportement de chaque *bloc*. Le *diagramme d'interaction de blocs* peut désigner: 1) les listes des *signaux* qui circulent entre les *processus* dans le même *bloc*; 2) les listes des *signaux* véhiculés par les *canaux* entre les *blocs* et 3) la création de nouvelles *instances de processus* par d'autres *instances de processus*;
- des *diagrammes de processus* qui définissent le comportement de chaque *processus* et qui sont les représentations graphiques des *définitions de processus*. Les *diagrammes de processus* peuvent contenir des *définitions de variable*, des *définitions de paramètre formel*, et des *définitions de visibilité*;

– des *listes de signaux* qui nomment les signaux véhiculés par canal, ou d'un *processus* à un autre, à l'intérieur du *bloc*. Cés *signaux* peuvent être introduits dans le *diagramme d'interaction de blocs* au moyen de *symboles* de *listes de signaux*, ou présentés dans les listes distinctes sous toute forme jugée adéquate;
– des *définitions de signal* qui indiquent les *types de données* et l'ordre des *valeurs de données* pouvant être contenues dans un signal, et cela pour chaque *signal* nommé dans la *liste de signaux*. Elles sont spécifiées au moyen du LDS/PR.

Dans une *définition de système*, on peut utiliser des *noms* et des *identificateurs*. Les *noms* sont spécifiés dans la syntaxe du LDS/PR. Les *identificateurs* se composent d'un *nom* complété par des *qualificateurs*. Un *qualificateur* est le *type* d'entité auquel est associé le *nom*; il représente le *niveau* hiérarchique de l'entité identifiée. Un même *identificateur* ne peut pas être associé à deux ou plusieurs entités. Dans le LDS/GR, les qualificateurs des *noms* peuvent être déduits du contexte mais, dans tous les cas où ils sont utilisés, ils sont spécifiés dans la syntaxe du LDS/PR.

3.1 *Diagrammes d'interaction de blocs*

Le *diagramme d'interaction de blocs* pour un *système* contient un *nom de système*, un ensemble de *symboles de bloc*, des *symboles d'environnement*, un ensemble de *symboles de canal*, et des *symboles de liste de signaux*.

3.1.1 *Symboles*

Les *symboles* recommandés sont représentés dans la figure 2/Z.101.

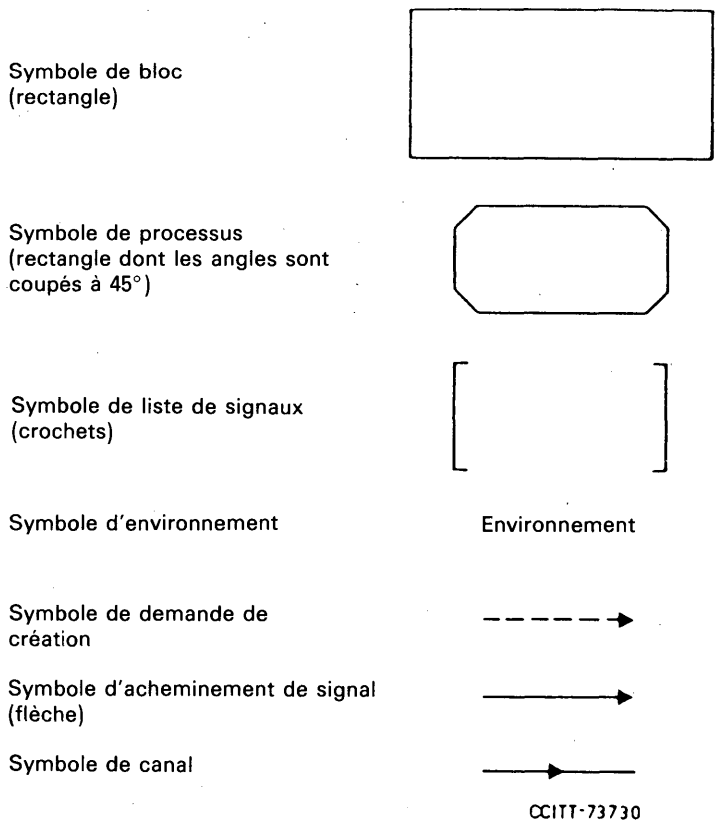


FIGURE 2/Z.101

Symboles des diagrammes d'interaction de blocs

3.1.2 *Relations entre les diagrammes d'interaction des blocs LDS/GR, la syntaxe abstraite et l'utilisation des symboles*

- Le *symbole d'environnement* représente l'*environnement du système* ; il peut apparaître plusieurs fois.
- *Un symbole de bloc* contient un *nom*, un ensemble non vide *de symboles de processus*, des *symboles d'acheminement des signaux* et peut contenir des *symboles de création* et des *symboles* de *liste de signaux.*
- *Un symbole de processus* contient un *nom de processus* et peut contenir un *symbole de liste de paramètres formels*. Le *nom de processus* est le même que le *nom* contenu dans la *définition de processus* qui décrit le comportement du *processus.* Le *symbole de liste de paramètres formels* contient une liste des *noms* des *paramètres formels* qui sont initialisés soit quand le *système* est créé, soit quand *l'instance de processus* est créée dynamiquement.
- Un couple de *valeurs entières* peut être associé à un *symbole de processus* ; la première *valeur* représente le nombre des *instances* du *processus* qui existent au moment où le *système* est créé, la deuxième *valeur* représente le nombre maximum d'*instances* simultanées de ce *type de processus.* Les deux *valeurs* sont placées dans l'angle supérieur droit du *symbole de processus.*

  La valeur par défaut de la première *valeur* est «un». La valeur par défaut de la deuxième *valeur* est l'«infini». Ce couple d'*entiers* est spécifié dans la syntaxe du LDS/PR. Lorsque l'un des *entiers* n'est pas spécifié, c'est la *valeur* par défaut qui est choisie.
- Un *symbole de canal* a un *nom de canal* qui lui est attaché. Le *symbole de canal* a une extrémité d'origine qui est connectée à un *symbole de bloc*, et une extrémité de destination qui est connectée à un autre *symbole de bloc.* A titre de variante, l'extrémité d'origine (ou l'extrémité de destination mais non les deux) peut être connectée à un *symbole d'environnement*, en lieu et place d'un *symbole de bloc.* Un *symbole de liste de signaux* peut être juxtaposé à un *symbole de canal* pour identifier les *signaux* véhiculés par le *canal.*
- Dans un *bloc*, un *symbole d'acheminement de signal* est associé à un *symbole* de *liste de signaux.* Un *symbole d'acheminement de signal* conduit soit d'un *processus* à un autre, soit d'un *processus* jusqu'à l'extrémité d'origine d'un *canal* (à la frontière du *bloc*), soit de l'extrémité de destination d'un *canal* (à la frontière du *bloc*) jusqu'à un *processus.*
- Dans un *diagramme d'interaction de blocs*, une *liste de signaux* est une liste de *noms*, le tout étant contenu dans le *symbole* de *liste de signaux* (c'est-à-dire des crochets). La *liste de signaux* peut elle-même avoir un *nom*, qui est inscrit au-dessus du *symbole.* Les rubriques de la liste (qui sont séparées par des virgules et peuvent être placées dans des colonnes ou des rangées) sont les *noms* des diverses définitions signal et les *noms* d'autres *listes de signaux.* A l'intérieur de la liste, on fait la distinction entre les *noms* de liste de signaux et les *noms* des définitions de signaux individuels en entourant chaque *nom* de *liste de signaux* par deux crochets supplémentaires. Si l'on décide d'inclure les *listes de signaux* dans le *diagramme d'interaction de blocs* en utilisant des *symboles* de *liste de signaux*, toutes les *listes de signaux* doivent être incluses dans le diagramme. La figure 3/Z.101 donne des exemples de *listes de signaux.*

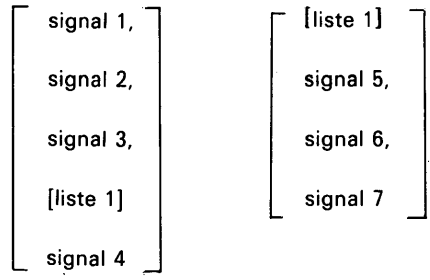


FIGURE 3/Z.101

Exemples de listes de signaux dans les diagrammes d'interaction de blocs

– Les *symboles de création* conduisent d'un *symbole de processus* jusqu'à un autre *symbole de processus*. Le premier *symbole de processus* représente le *processus* «créateur». Le deuxième *symbole de processus* représente le *processus* «créé».

3.1.3 *Règles applicables aux dessins*

– Les *symboles de canal* sont connectés aux frontières des *symboles de bloc* auxquels ils sont associés. Les *symboles de canal* doivent joindre les frontières des *symboles de bloc* en formant des angles de 90°. Si nécessaire, les *symboles de canal* peuvent représenter des coudes à 90°. Un *symbole de canal* contient une flèche qui indique le sens de circulation des *signaux* dans le *canal*.

– Les *symboles d'acheminement de signal* sont connectés aux frontières des *symboles de bloc* ou des *symboles de processus* auxquels ils sont associés. Les *symboles d'acheminement de signal* joignent ces frontières en formant de préférence des angles de 90°. Si nécessaire, les *symboles d'acheminement de signal* peuvent présenter des coudes à 90°. Plusieurs *symboles d'acheminement de signal* peuvent converger à l'extrémité d'origine d'un *canal* (à la frontière du *bloc*). Plusieurs *symboles d'acheminement de signal* peuvent diverger à partir de l'extrémité de destination d'un *canal*, à la frontière du *bloc*. Un *symbole d'acheminement de signal* contient une flèche à une extrémité, pour indiquer le sens de circulation des *signaux*.

– Les *symboles de création* sont connectés aux frontières des *symboles de processus* par des lignes pointillées qui vont jusqu'aux frontières des *symboles de processus*, formant des angles de 90°. Ces lignes de connexion portent des flèches dessinées de telle façon que le *symbole de processus* «créateur» est orienté dans la direction du *symbole du processus* «créé».

– L'orientation préférée des *symboles* est représentée dans la figure 2/Z.101.

– Les dimensions des *symboles* sont déterminées librement, au gré de l'utilisateur.

– Les frontières des *symboles* ne doivent ni se superposer, ni se couper. Font exception à cette règle les *symboles de canal* et les *symboles d'acheminement de signal* qui peuvent se couper. Il n'existe aucune relation de logique entre les *symboles de canal* ou entre les *symboles d'acheminement* de *signal* qui se coupent.

3.2 *Listes de signaux*

L'union de tous les *noms de signal* figurant dans les *listes de signaux* associées à des *symboles d'acheminement de signal* qui sont connectés à une *définition de processus* donnée est égale à l'ensemble des noms de *signal entrant* valables pour le *processus* considéré.

L'union de tous les *noms de signal* figurant dans les *listes de signaux* associées à des *symboles d'acheminement de signal* qui sont connectés à l'extrémité d'origine d'un *canal* est égale à la liste des *noms* figurant dans la *liste de signaux* associée à ce *canal*; elle est égale aussi à l'union de tous les *noms de signal* figurant dans les *listes d'acheminement des signaux* associées aux *symboles de signal* (dans le *bloc* de destination de *canal*) connectés à l'extrémité de destination du *canal* considéré.

3.3 *Diagrammes de processus*

3.3.1 *Symboles*

Le comportement d'un *processus* est représenté graphiquement par un *diagramme de processus*. Le nom du *diagramme de processus* est identique au *nom de processus* dans la *définition de processus* qu'il représente. La *figure 4/Z.101* montre les *symboles* utilisés dans les *diagrammes de processus* de la forme LDS/GR.

3.3.2 *Relations entre les diagrammes de processus LDS/GR et la syntaxe abstraite et les symboles du LDS*

Chaque *symbole de diagramme de processus* mentionné à la *figure 4/Z.101* représente, dans la *syntaxe abstraite*, le *nœud* de même *nom* du *graphe de processus*. Les lignes de liaison qui connectent les *symboles* représentent les *arcs* orientés qui connectent les *nœuds*. La *figure 5/Z.101* représente les connexions admissibles des *symboles* par des lignes de liaison dans un *diagramme de processus* du LDS/GR.

Les *paramètres formels*, l'*ensemble de signaux entrants valables*, les *définitions de variable*, les *définitions de visibilité*, les *expressions* et les *expressions de visibilité* sont spécifiés dans la syntaxe du LDS/PR.

Un *symbole de départ* représente un *nœud de départ* (voir aussi le § 2.2.3.3). Le *symbole de départ* contient le *nom* du *processus* qu'il décrit.

Un *symbole d'arrêt* représente le *nœud d'arrêt*.

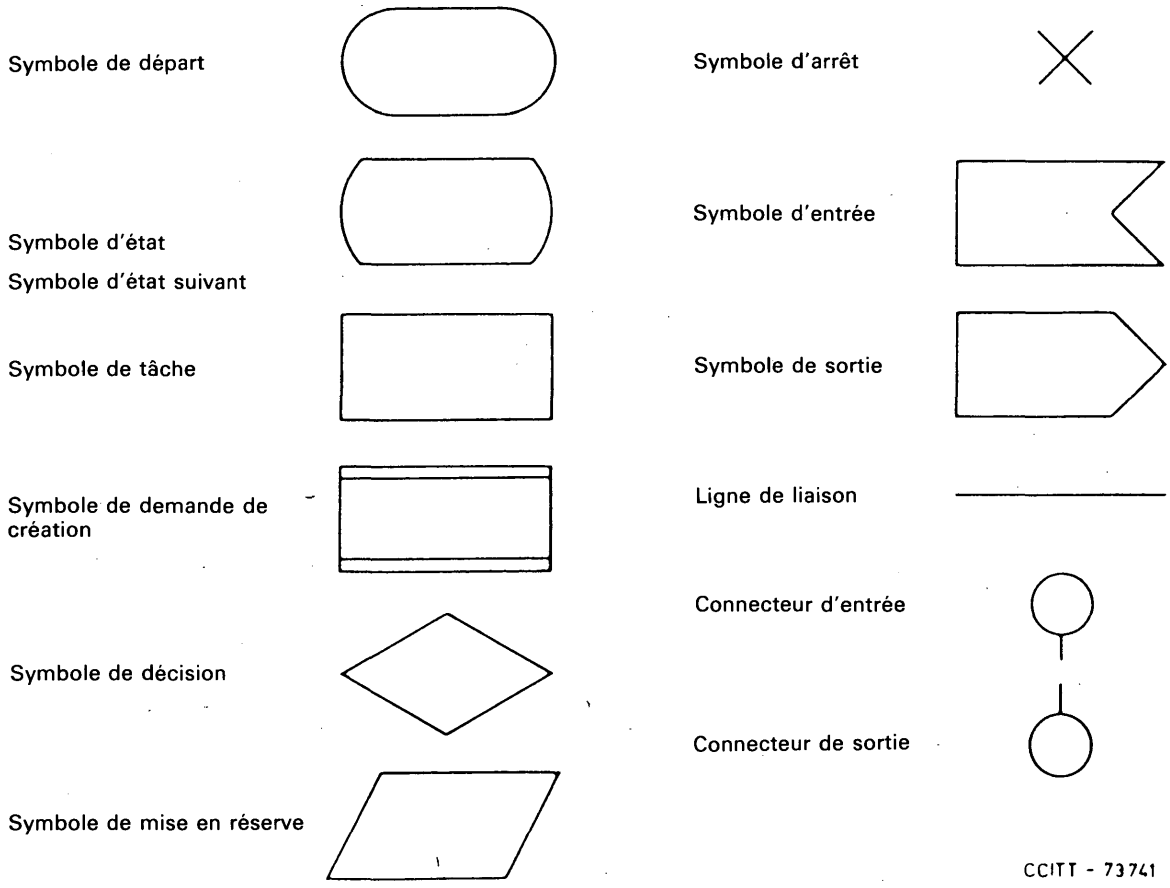


FIGURE 4/Z.101

Symboles des diagrammes de processus dans le LDS/GR

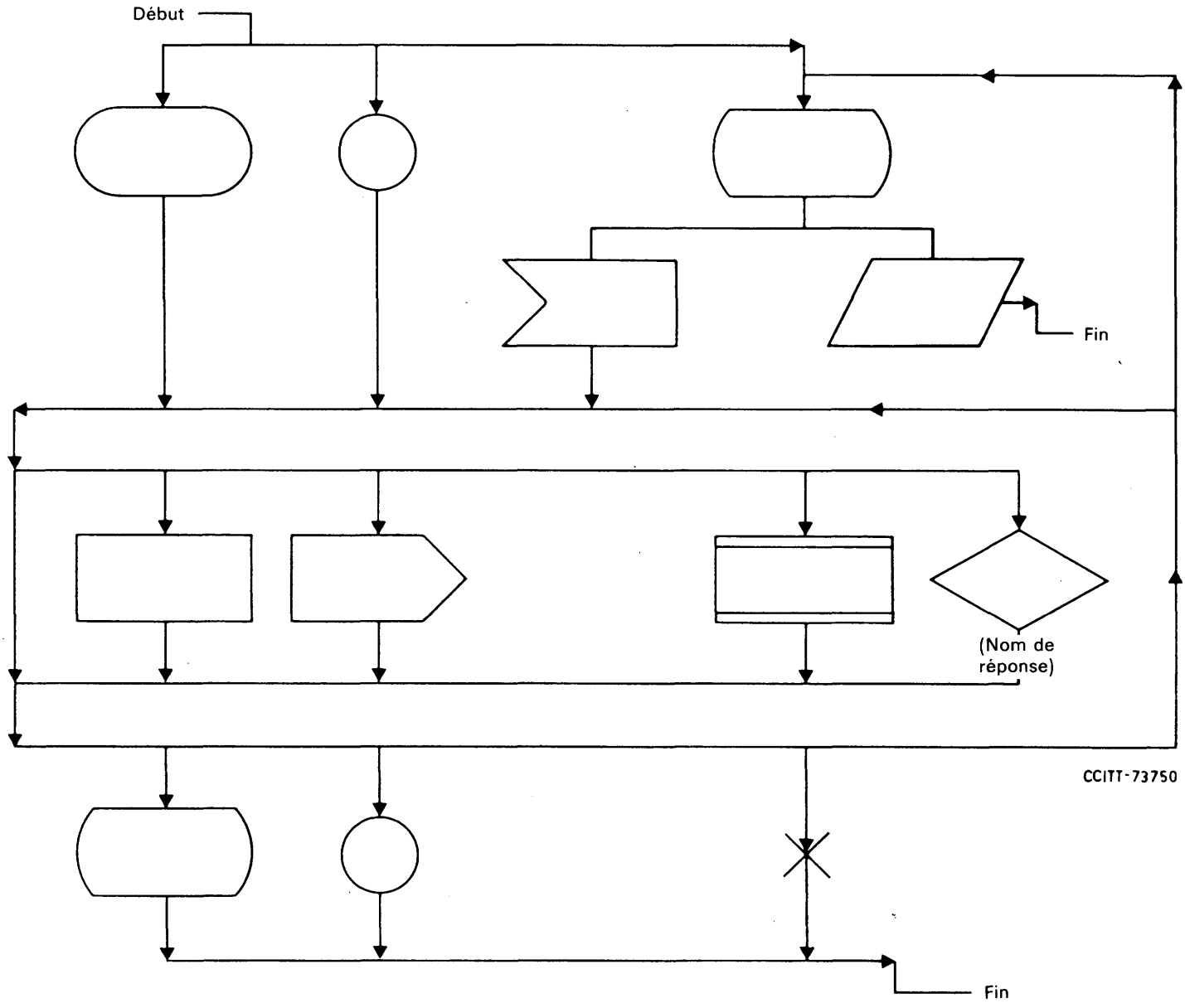


FIGURE 5/Z.101

Connexions admissibles des symboles par des lignes de liaison dans un diagramme de processus du LDS/GR

Un *symbole d'état* représente un ou plusieurs *nœuds d'état* et contient soit un ou plusieurs *noms d'état* séparés par des virgules, soit un astérisque, soit encore un astérisque suivi d'une liste de *noms d'état* placée entre parenthèses.

Un *symbole de mise en réserve* représente l'ensemble de *signaux de mise en réserve* attaché à un *nœud d'état*. Il contient un ou plusieurs *noms de signaux*, séparés par des virgules ou par un astérisque.

Un *symbole* d'entrée représente un ou plusieurs *nœuds d'entrée*. Les *noms de signal* contenus dans le *symbole d'entrée* sont séparés par des virgules. Chacun de ces *noms de signal* donne le *nom* d'un des *nœuds d'entrée* représenté par ce *symbole* d'entrée.

Un *symbole de tâche* représente un *nœud de tâche*. Le *symbole de tâche* contient soit une séquence d'*instructions*, soit des expressions informelles.

Le *symbole de demande de création* représente un *nœud de demande de création*. Il contient une *action de demande de création*, comme cela est spécifié dans la syntaxe LDS/PR.

Un *symbole de décision* représente un *nœud de décision*. Il contient une question et peut contenir un *nom de décision*. Le *symbole de décision* est relié à d'autres *symboles* par deux ou plusieurs *lignes de liaison*; à chacune de ces *lignes de liaison* est attaché son propre *nom* de réponse (inscrit le long de la *ligne* ou dans une interruption de *ligne*). La *réponse* ELSE (autre) implique une *réponse* qui vaut pour toute *réponse* qui n'est couverte par aucun *nom* de *réponse*.

Un *symbole de sortie* représente un ou plusieurs *nœuds de sortie*. Les *noms de signal* contenus dans le *symbole de sortie* sont séparés par des virgules. Chacun de ces *noms* de signal donne le *nom* d'un *nœud de sortie* qui est représenté par ce *symbole de sortie*. L'identificateur d'instance de processus de destination peut recevoir facultativement dans le *symbole de sortie* spécifié dans la syntaxe LDS/PR (c'est-à-dire un mot clé TO suivi d'une *expression* de l'*identificateur d'instance de processus type*. Le mot clé TO vient après la liste des *noms de signaux*). S'il est impossible de définir sans ambiguïté le *processus* de destination *d'un signal* – pour cause d'insuffisance du *nom de signal* ou du contexte – *l'identificateur d'instance de processus* est nécessaire.

Un *symbole d'état suivant* représente un *arc* reliant le dernier *nœud* dans une *chaîne de transition* vers le *nœud d'état* suivant.

Une *ligne de liaison* reliant deux autres *symboles* (représentant des *nœuds*) représente *l'arc* reliant les *nœuds* correspondants.

Un *symbole de commentaire* est utilisé pour attacher un *texte informel* à tout autre *symbole*.

Un *connecteur d'entrée* contient une *étiquette* et représente la continuation d'une *ligne de liaison* à partir d'un *connecteur de sortie* correspondant qui contient la même *étiquette*.

3.3.3 *Conventions graphiques*

3.3.3.1 *Transitions implicites*

La *syntaxe abstraite* du LDS exige qu'une *mise en réserve* ou une *entrée* conduisant à une *transition* soit spécifiée pour chaque *signal entrant* dans l'ensemble des *signaux entrants* valables d'un *processus* pour chaque *état* du *processus*. Le LDS/GR fournit des *transitions* «nulles» implicites dans le cas des signaux pour lesquels il n'est donné ni de *transitions*, ni de *mises en réserve* explicites. Une *transition* nulle équivaut à une connexion conduisant d'un *symbole d'état* jusqu'à un *symbole* d'entrée, qui est ensuite connecté en retour au même *symbole d'état*. Pour un *état* donné, le LDS/GR rejette donc les signaux qui ne sont pas mentionnés explicitement en relation avec cet *état*. De la même façon, les *données* contenues dans ces *signaux* sont rejetées.

Si aucun *symbole de départ* ne figure dans un *diagramme de processus* LDS/GR, on admet la présence d'un *symbole de départ* implicite qui est connecté directement au *symbole d'état* «départ». L'*état* de départ peut être identifié indirectement d'après son *nom*, ou par un *commentaire*.

3.3.3.2 *Lignes de liaison et connecteurs*

Lorsque deux ou plusieurs *symboles* sont suivis d'un seul *symbole*, les *lignes de liaison* aboutissant à ce *symbole* convergent. Cette *convergence* peut se présenter sous la forme d'une *ligne de liaison* en rejoignant une autre, sous la forme de plusieurs *connecteurs de sortie* associés à un seul *connecteur d'entrée*, ou encore sous la forme de *lignes de liaison* séparées entrant dans le même *symbole*.

Lorsqu'un *symbole* est suivi de deux ou plusieurs autres *symboles*, une *ligne de liaison* partant de ce *symbole* peut diverger en deux ou plusieurs *lignes de liaisons*.

Des flèches sont nécessaires chaque fois que deux *lignes de liaison* se rencontrent et chaque fois que ces lignes entrent dans un *connecteur de sortie* ou dans un *symbole d'état*. L'utilisation des flèches est interdite sur les *lignes de liaison* entrant dans les *symboles d'entrée*. Dans tous les autres cas, les flèches sont facultatives.

3.3.3.3 *Apparition multiple*

Chaque fois qu'apparaissent des *symboles d'état* ou des *symboles de connecteurs* ayant le même *nom*, on considère qu'ils représentent la même entité sémantique. Cette entité est censée contenir l'union de toutes les *lignes de liaison* entrantes et sortantes correspondant à toutes ses représentations multiples. Chaque fois qu'un ou plusieurs *symboles d'arrêt* apparaissent sur le même *diagramme de processus*, ils représentent tous le *nœud* d'arrêt.

3.3.3.4 *Notation abrégée*

Une notation abrégée permet de faire référence à tous les autres *signaux* ou *états*, dans les *symboles* d'entrée, de *mise en réserve* ou d'*état*.

Un *symbole* d'entrée attaché à un *état* et contenant un «*» (astérisque) indique que la *transition* suivante s'applique à tous les signaux entrants qui ne figurent pas d'une autre manière dans les *symboles d'entrée* ou de *mise en réserve* attachées à une apparition quelconque de ce *symbole d'état*. Pour un *état* donné, il n'est admis qu'un seul *symbole* d'entrée ou un seul symbole de mise en réserve avec «*».

Un *symbole de mise en réserve* attaché à un *état* et contenant un «*» indique qu'il faut *mettre en réserve* tous les signaux ne figurant pas dans les symboles d'entrée attachés à une apparition quelconque de ce *symbole d'état*.

Un «*» dans un *symbole d'état* se rapporte à tous les *états* dans ce *processus* et indique que les *transitions* ou les *mises en réserve* suivantes doivent être interprétées dans chaque *état*. Un «*» suivi d'une liste de *noms d'état* entre parenthèses indique que les *transitions* ou les *mises en réserve* suivantes doivent être interprétées dans tous les *états*, à l'exception de ceux qui figurent dans la liste. Ces *symboles d'état* ne doivent pas avoir de *lignes de liaison* entrantes.

Un «–» dans un *symbole d'état suivant* signifie que l'*état* qui suit est le même *état*, à partir duquel la *transition* actuelle a commencé. La notation «–» ne doit pas figurer dans un *symbole d'état suivant* qui vient à la suite du *symbole de départ*.

La fusion d'un *symbole d'état suivant* et d'un *symbole d'état* n'est possible que si ces deux symboles représentent un seul et même *symbole d'état*.

3.3.3.5 *Divers*

Dans un diagramme donné, tous les *symboles* du même *type* doivent, de préférence, avoir les mêmes dimensions.

L'orientation des *symboles* doit, de préférence, être horizontale et le rapport entre la longueur et la largeur des *symboles* doit de préférence être 2:1.

Les images «miroir» des *symboles d'entrée* et de *sortie* sont autorisées.

Les *lignes de liaison* sont horizontales ou verticales et elles forment des angles droits.

Les *lignes de liaison* qui se croisent n'ont aucune relation logique.

Le texte associé à un *symbole* doit, dans la mesure du possible, figurer à l'intérieur de celui-ci.

3.3.3.6 *Gabarit LDS*

On trouvera dans le manuel destiné aux usagers du LDS, un gabarit servant à dessiner l'ensemble de base des symboles du LDS.

3.4 *Symbole d'extension de texte*

Un *symbole d'extension de texte* peut être attaché à tous les *symboles* LDS/GR. Le texte contenu dans ce *symbole* doit être considéré comme contenu dans le *symbole* auquel est attaché le *symbole d'extension de texte*. Le *symbole d'extension de texte* est illustré en figure 6/Z.101.

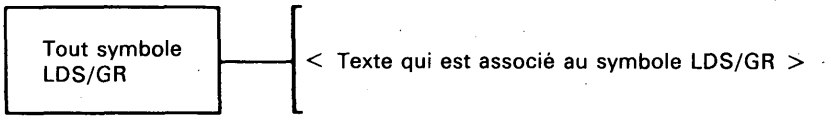


FIGURE 6/Z.101

Symbole d'extension de texte

3.5 *Commentaires dans le LDS/GR*

Dans tous les diagrammes LDS/GR, des *commentaires* peuvent être insérés à la place estimée nécessaire par l'usager. Les *commentaires* peuvent être insérés soit au moyen d'un *symbole de commentaire* LDS/GR (voir la figure 7/Z.101) soit au moyen de la syntaxe LDS/PR pour les *commentaires* (voir le § 4.3.2, règle de lexique n° 7).

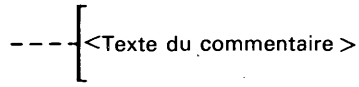


FIGURE 7/Z.101

Symbole de commentaire

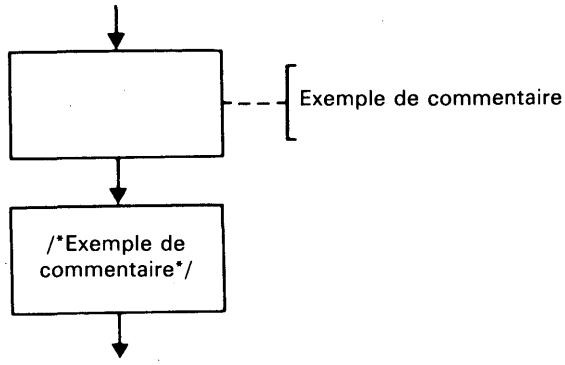


FIGURE 8/Z.101

Exemples de commentaires dans le LDS/GR

4 Syntaxe linéaire

4.1 *Observations générales*

On trouvera ci-après la définition du LDS/PR et la relation de ce dernier avec le modèle commun de langage (voir le § 2).

Dans la syntaxe LDS/PR, une *définition de système* est représentée par une suite d'instructions délimitée par les mots SYSTEM et ENDSYSTEM.

Les règles détaillées de la syntaxe du LDS/PR sont contenues dans les diagrammes de syntaxe du § 4.3.

La référence à une entité nommée extérieure à sa définition est assurée au moyen d'un *identificateur*, lequel est composé d'un *nom* et d'une *partie qualificative* optionnelle (appelée: «partie qualificatif») qui ne doit être utilisée que lorsque le *nom* ne permet pas, à lui seul, de déterminer sans ambiguïté l'objet dont il s'agit.

4.2 *Mots clés*

Le LDS/PR utilise un certain nombre de mots clés pour exprimer les concepts du LDS, tels qu'ils sont définis dans la syntaxe abstraite. Certains de ces mots clés sont utilisés par couples pour refléter le mode de structuration du LDS en LDS/PR.

4.2.1 *Couples de mots clés de structuration se rapportant aux définitions*

Le verbe «englobe» est utilisé avec ces couples de mots pour mettre en évidence leur rôle délimiteur.

SYSTEM ENDSYSTEM	Englobent le concept de *définition de système.* (En LDS/PR, la représentation d'un système commence avec le mot clé SYSTEM et se termine par le mot clé ENDSYSTEM.)
BLOCK ENDBLOCK	Englobent le concept de *définition de bloc.*
PROCESS ENDPROCESS	Englobent le concept de *définition de processus.*

4.2.2 *Mots clés simples se rapportant à des définitions*

Dans le présent paragraphe, les mots clés sont utilisés pour indiquer qu'une définition suit.

DCL	introduit la représentation de la *définition de variable.* Le mot clé «REVEALED» est utilisé dans une instruction DCL pour identifier des *variables* révélées.
VIEWED	introduit la représentation de la *définition* de *visibilité.*
SIGNAL	introduit la représentation de la *définition de signal.*
CHANNEL	introduit la représentation de la *définition de canal.* Le mot clé «FROM» est utilisé dans la définition de CANAL pour indiquer le *bloc* d'origine du *canal* et le mot clé «TO» est utilisé pour indiquer le *bloc* de destination. Le mot clé «ENV» est utilisé comme référence à *l'environnement.* Le mot clé «WITH» est suivi de la liste des *signaux* véhiculés dans les *canaux.*
FPAR	introduit la représentation de la *définition du paramètre formel.*
DURATION TIME TIMER NATURAL INT REAL CHARSTRING CHARACTER BOOLEAN PID	représentent les types de données prédéfinis.
VIEW	introduit la représentation de l'*expression de visibilité.* Ce terme est utilisé dans des *expressions* chaque fois qu'une *variable* déclarée visible est utilisée.
SET	introduit la représentation de l'*instruction* d'*initialisation.*
RESET	introduit la représentation de l'*instruction* de *réinitialisation.*
SYSTEM BLOCK PROCESS	introduit la *partie qualificative* d'un *identificateur*

4.2.3 *Mots clés associés aux nœuds dans un graphe de processus*

Le graphe de processus dans la *syntaxe abstraite* consiste en *nœuds* connectés par des *arcs* orientés.

Les mots clés sont choisis pour correspondre aux différents *nœuds* et les *arcs* qui relient les *nœuds*, dans la *syntaxe abstraite*, sont représentés par l'ordre dans lequel apparaissent les mots clés (voir le § 4.2.5).

START	représente le *nœud de départ*. En l'absence de ce mot clé, le premier mot clé ÉTAT qui suit PROCESSUS, représente l'*état* de départ.
STATE	introduit la représentation d'un ou de plusieurs *nœuds d'état.* L'*ensemble des signaux de mise en réserve* attaché à un *nœud d'état* est représenté par le mot clé SAVE suivi d'un ou plusieurs *identificateurs de signal.*

INPUT — introduit la représentation d'un ou plusieurs *nœuds d'entrée.*

TASK — introduit la représentation d'un *nœud de tâche.*

OUTPUT — introduit la représentation d'un ou de plusieurs *nœuds de sortie.* L'*identificateur d'instance de processus* de destination peut être donné facultativement par le mot clé TO suivi d'une *expression* qui donne une *valeur d'identificateur d'instance de processus.* Lorsque la destination du *signal* ne peut être déterminée d'une manière univoque, le mot clé TO est nécessaire.

DECISION — couvre le concept de *nœud de décision*. Le mot clé ELSE est utilisé pour représenter la *réponse* à tous les autres cas non explicitement désignés.

CREATE — introduit la représentation d'un *nœud de demande de création.*

STOP — représente un *nœud d'arrêt.*

4.2.4 *Mots clés associés avec des arcs*

JOIN — représente un *arc* entre des *nœuds* qui ne sont pas des *nœuds d'état*. Le premier *nœud* est généralement représenté par le mot clé précédant immédiatement le mot clé JOIN; le deuxième *nœud* est toujours identifié par le fait qu'il a le même *identificateur d'étiquette* que le mot clé JOIN. Il y a cependant des exceptions à cette explication générale, dans la mesure où le premier *nœud* est concerné (voir le § 4.2.5).

Si le deuxième *nœud* est un autre JOIN, l'*arc* est relié au *nœud* auquel se réfère ce JOIN.

Si le mot clé ayant l'*étiquette* est NEXTSTATE (ÉTAT SUIVANT), le deuxième *nœud* est l'*état* de même *nom.* (Les règles de NEXTSTATE sont valables.)

NEXTSTATE représente un *arc*. Le premier *nœud* de l'*arc* est représenté par le mot clé précédant immédiatement le mot clé NEXTSTATE, le deuxième *nœud* est l'état de même *nom.*

4.2.5 *Représentation des arcs dans le LDS/PR*

La règle de représentation d'un *arc* dans la forme PR est donnée par l'ordre des mots clés.

Il y a quelques exceptions à cette signification générale, notamment dans le cas des mots clés tels que JOIN et NEXTSTATE, comme cela est indiqué dans le paragraphe précédent.

De plus, lorsqu'un mot clé (associé à un *nœud* ou à un *arc*) suit immédiatement une *réponse*, le premier *nœud* de l'*arc* est la *décision* correspondante qui précède.

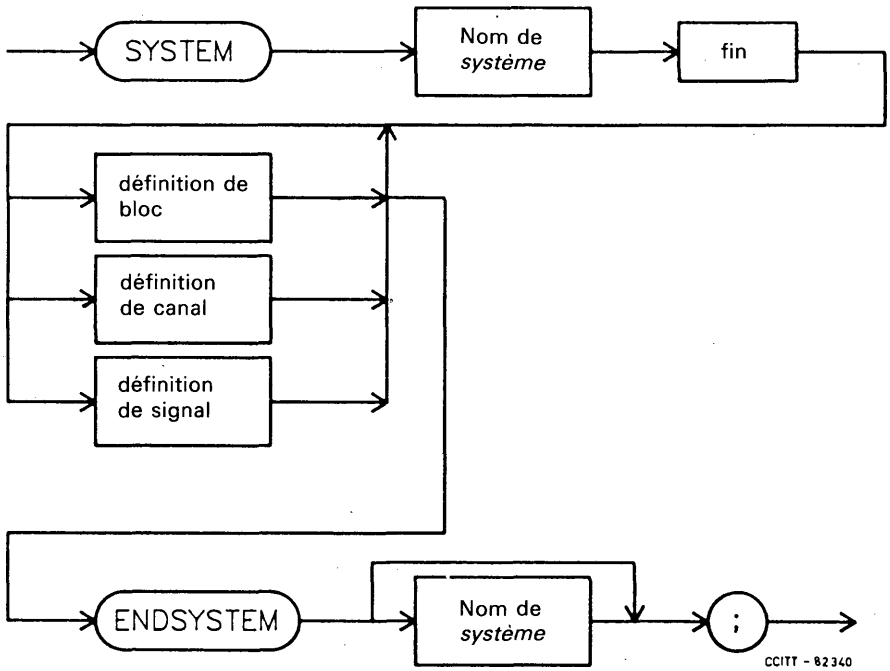
Si le mot clé qui précède directement un mot clé associé à un nœud ou à un arc est ENDDECISION, les premiers *nœuds des arcs* sont représentés par les derniers mots clés dans toutes les *chaînes de transition* de la *décision* dépourvue d'instructions terminales.

Le dernier mot clé d'une branche de *décision* représente un *nœud* qui n'est pas relié au mot clé qui suit le *nom* de résultat suivant mais qui est relié au mot clé faisant suite à ENDDECISION. Il est clair que cette règle n'est pas applicable si le dernier mot clé d'une branche de *décision* est une instruction terminale.

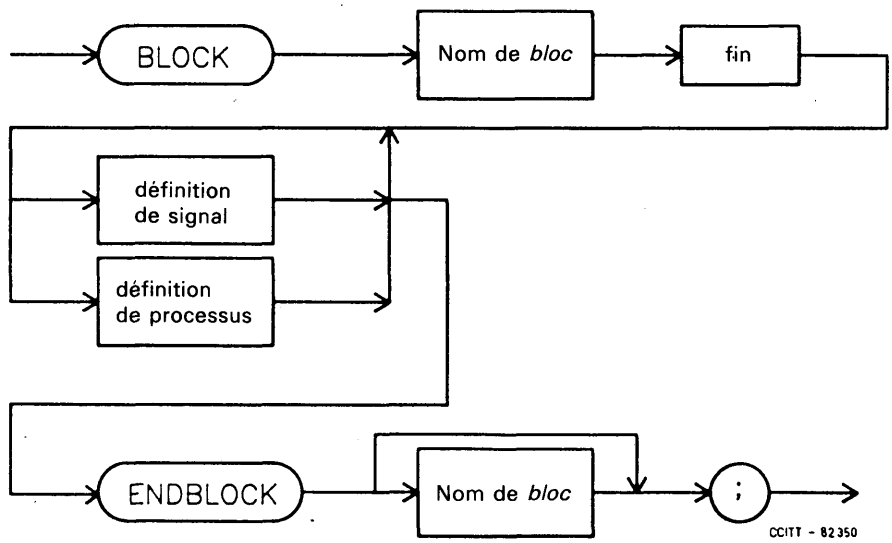
4.3 *Mots réservés dans le LDS/PR*

Certains mots sont réservés dans le LDS/PR et peuvent ne pas être utilisés comme *noms.* La liste des mots réservés figure à l'annexe C.2 aux Recommandations Z.100 à Z.104.

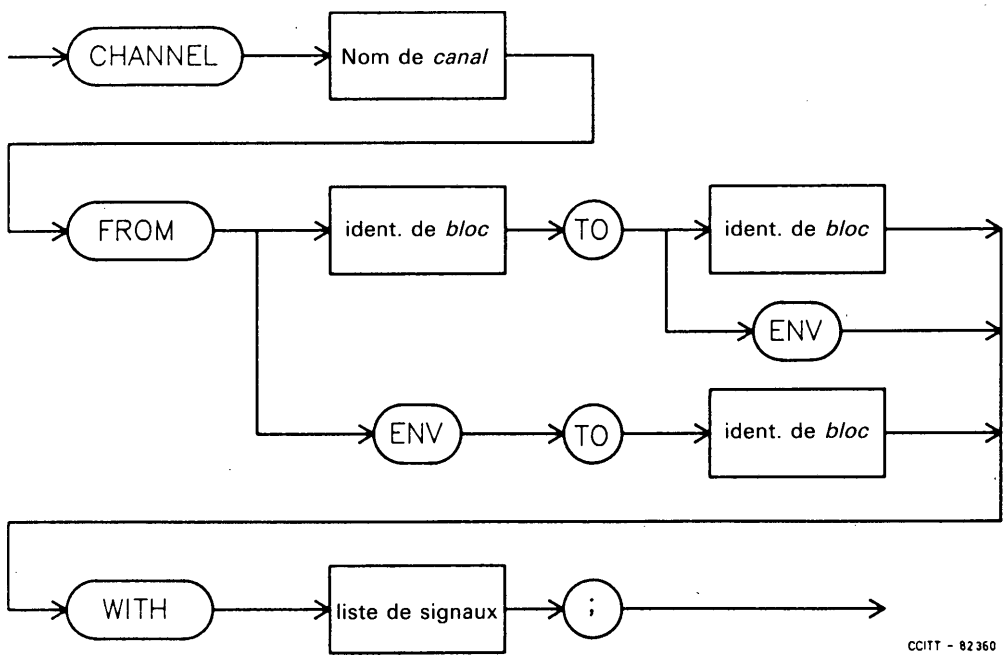
DÉFINITION DE SYSTÈME



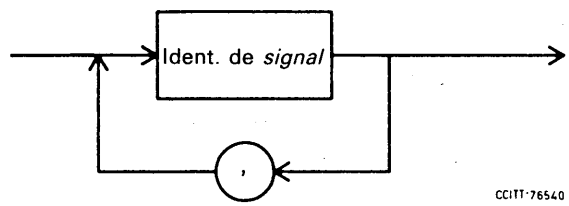
DÉFINITION DE BLOC



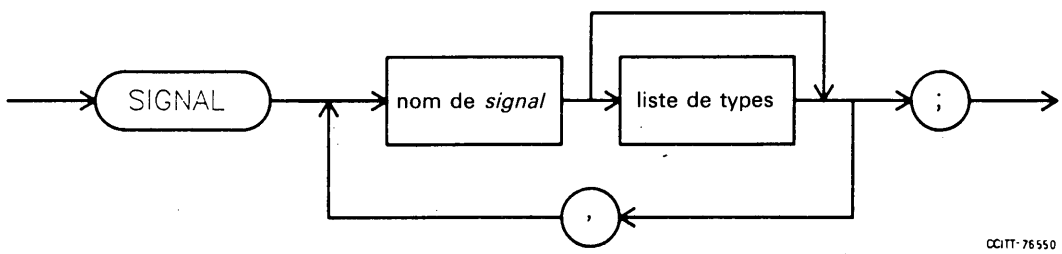
DÉFINITION DE CANAL



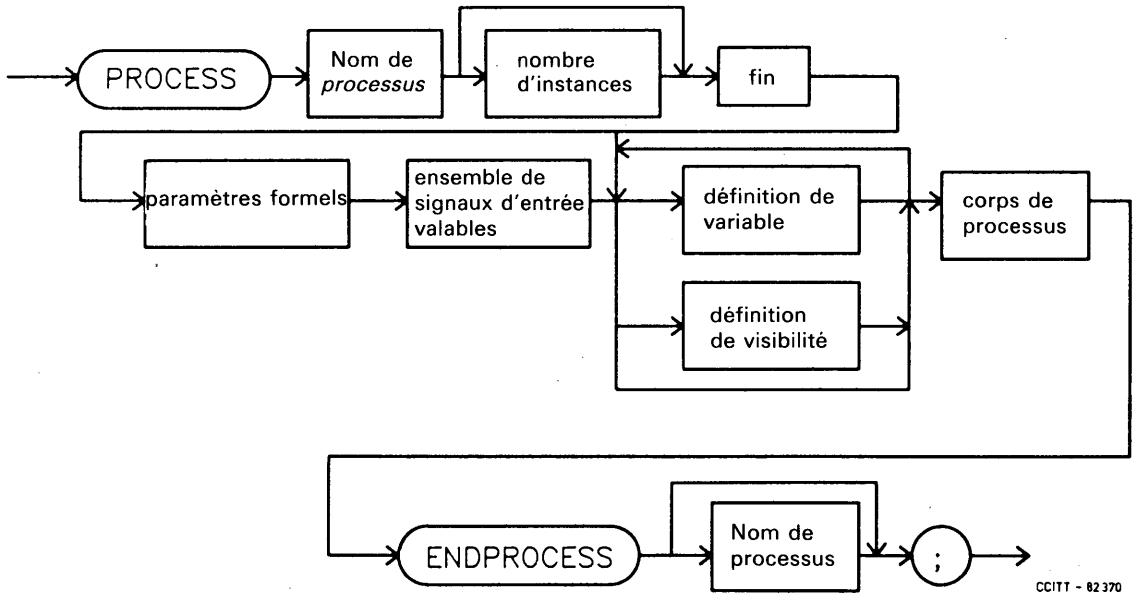
LISTE DE SIGNAUX



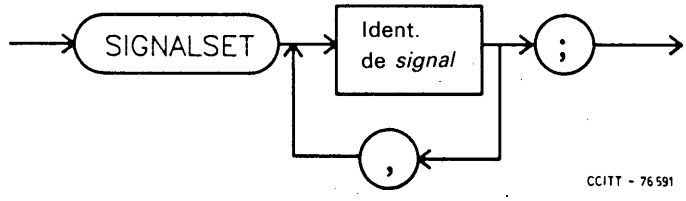
DÉFINITION DE SIGNAL



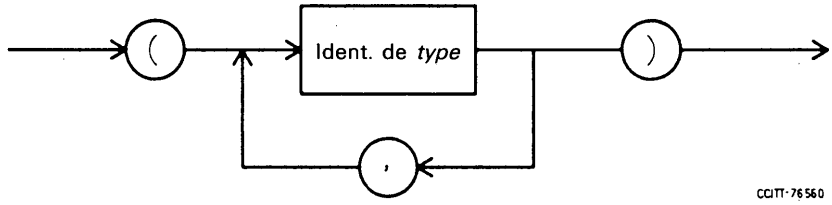
DÉFINITION DE PROCESSUS



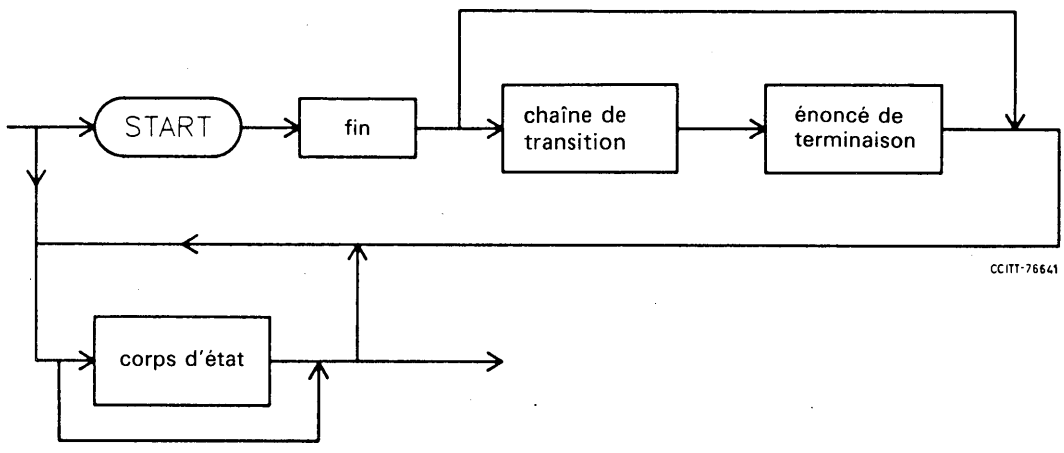
ENSEMBLE DE SIGNAUX D'ENTRÉE VALABLES



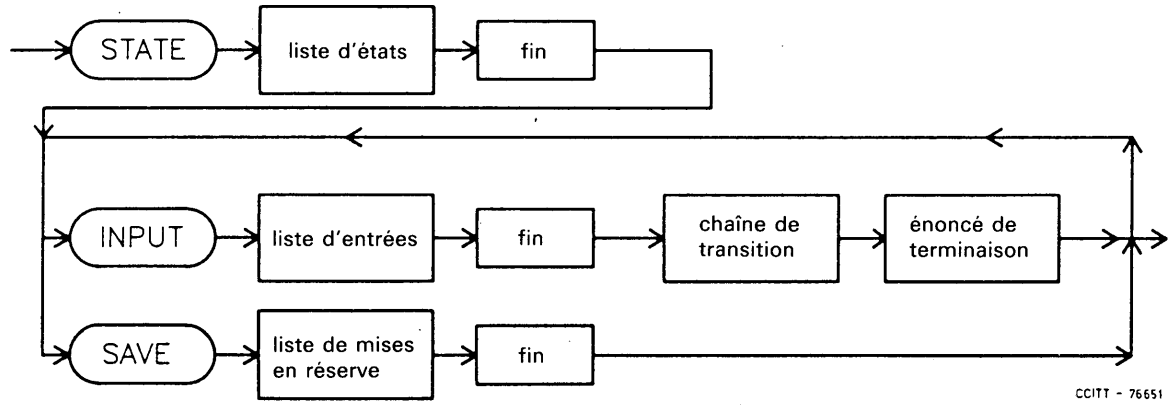
LISTE DE TYPES



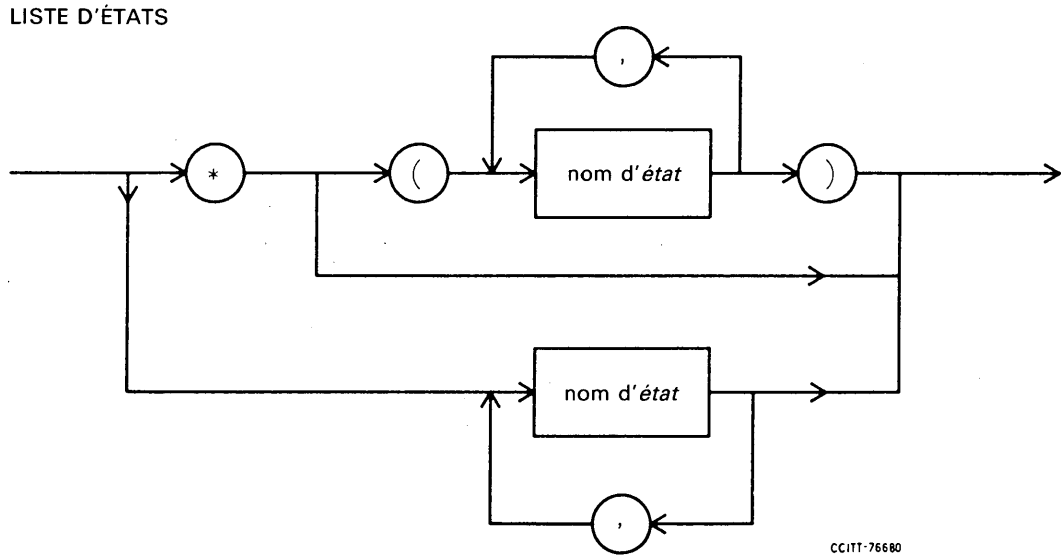
CORPS DE PROCESSUS



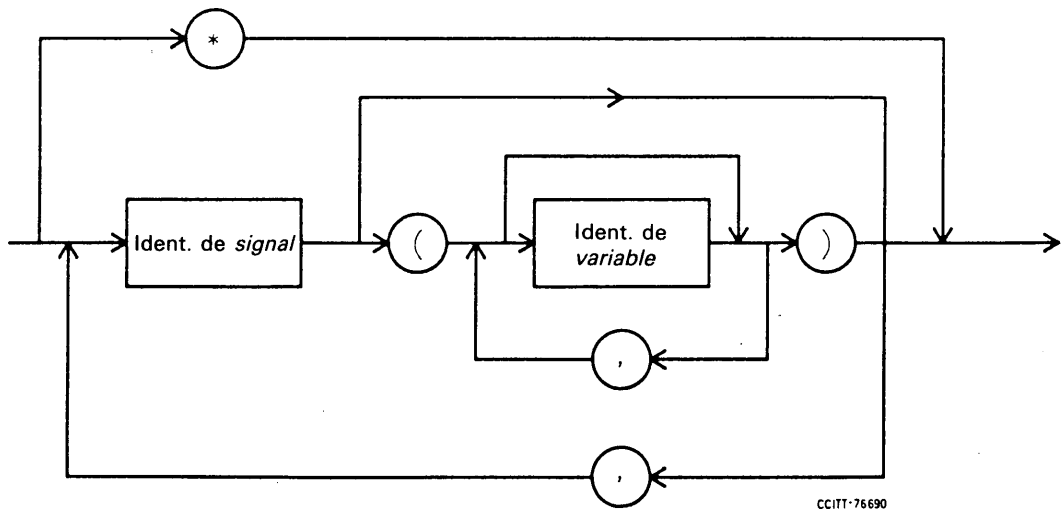
CORPS D'ÉTAT



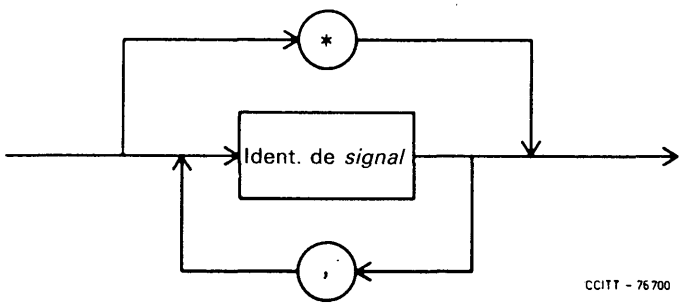
LISTE D'ÉTATS



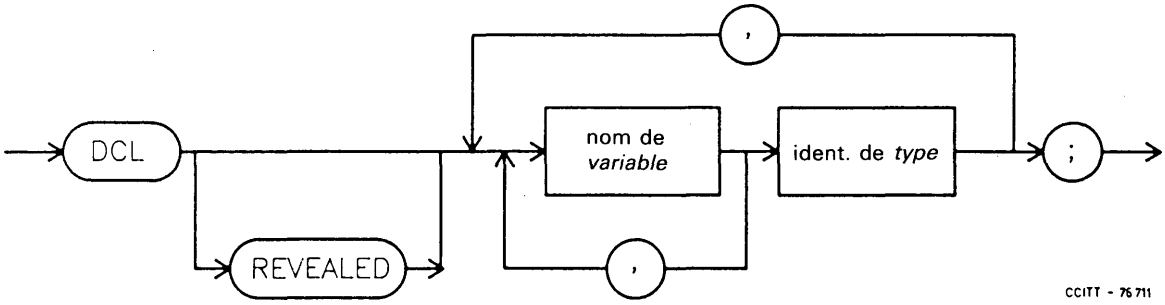
LISTE D'ENTRÉES



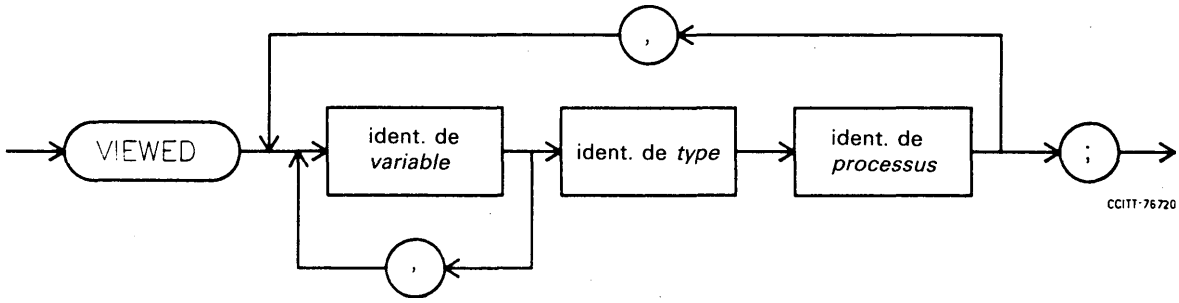
LISTE SAVE



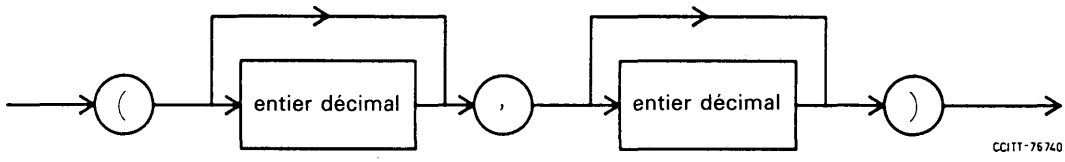
DÉFINITION DE VARIABLE



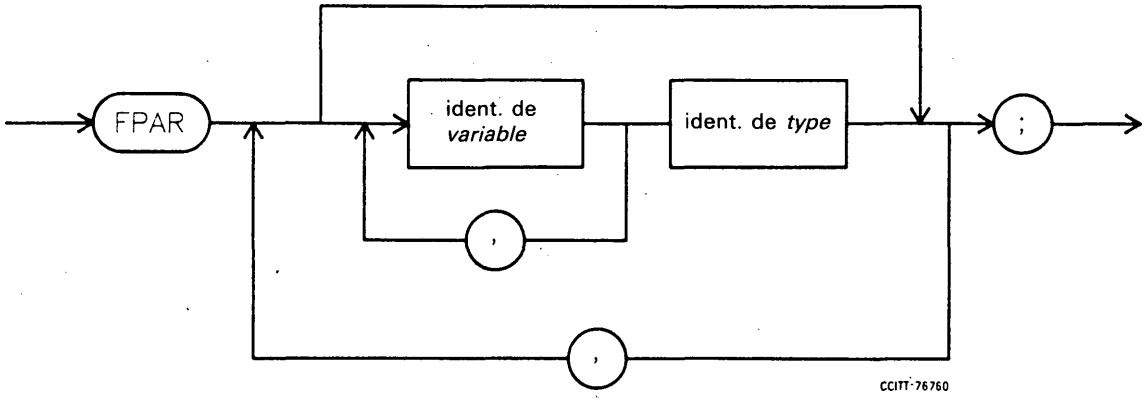
DÉFINITION DE VISIBILITÉ



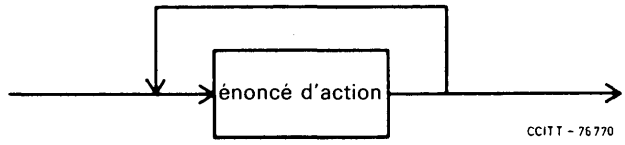
NOMBRE D'INSTANCES



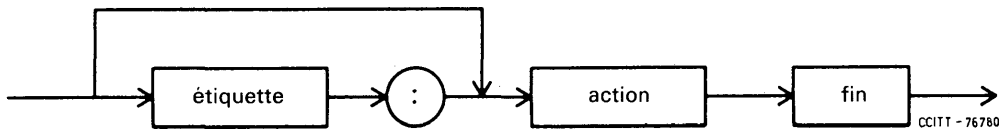
PARAMÈTRES FORMELS



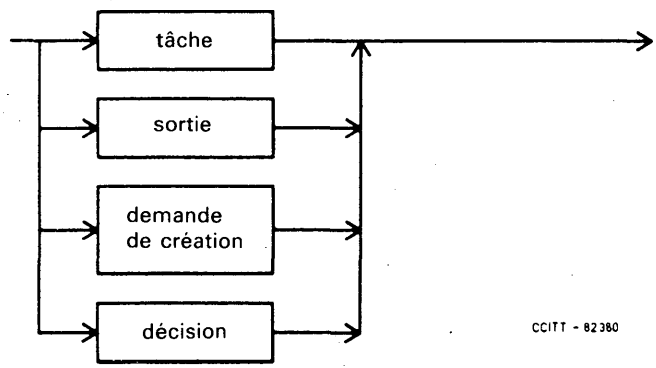
CHAÎNE DE TRANSITION



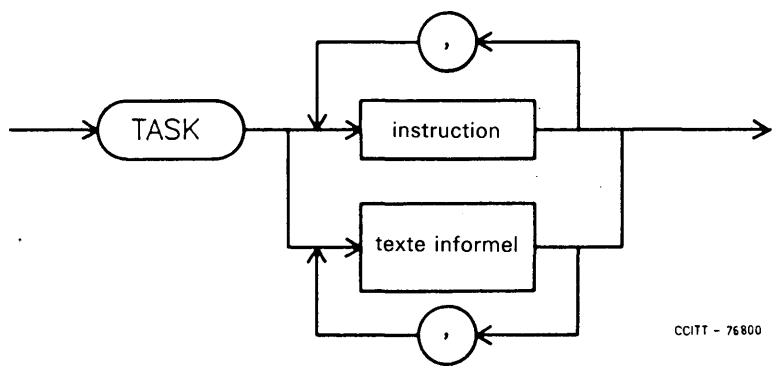
ÉNONCÉ D'ACTION



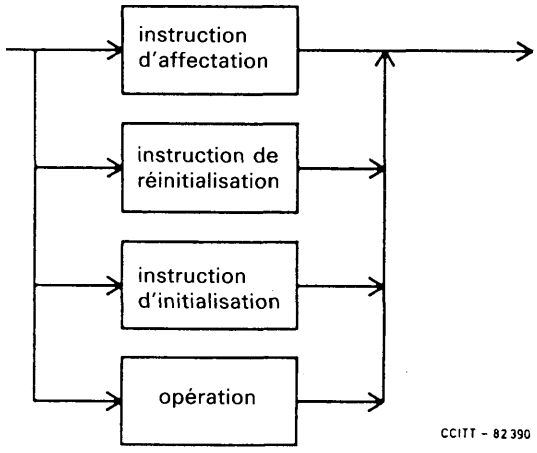
ACTION



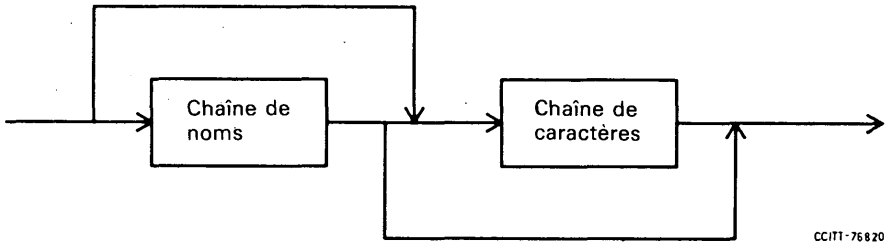
TÂCHE



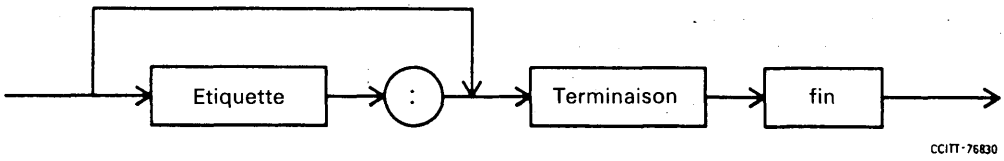
INSTRUCTION



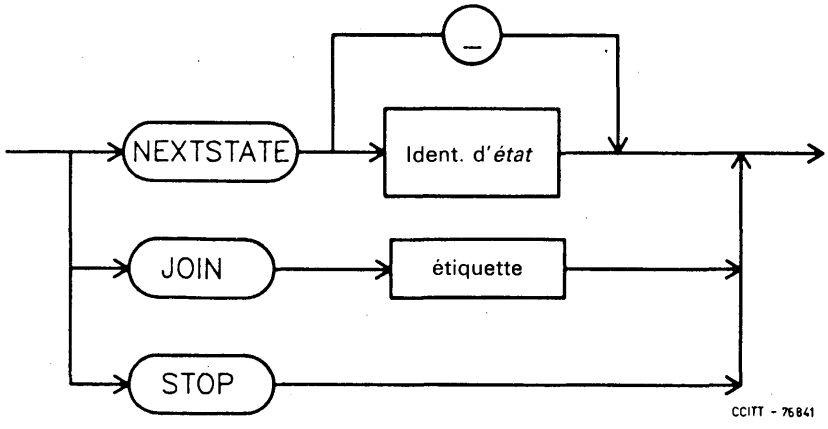
TEXTE INFORMEL



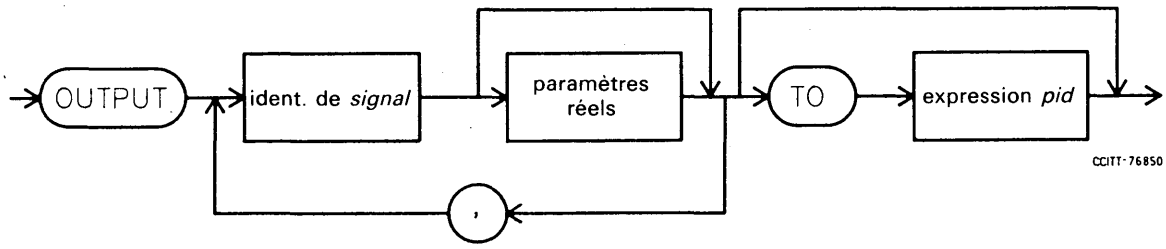
ÉNONCÉ DE TERMINAISON



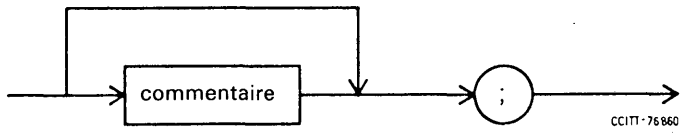
TERMINAISON



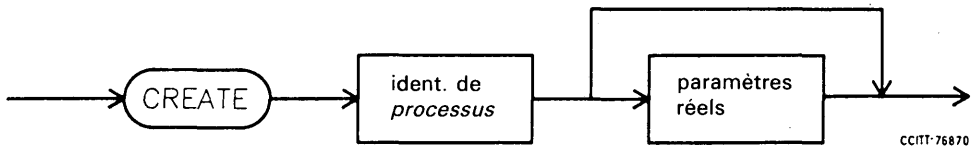
SORTIE



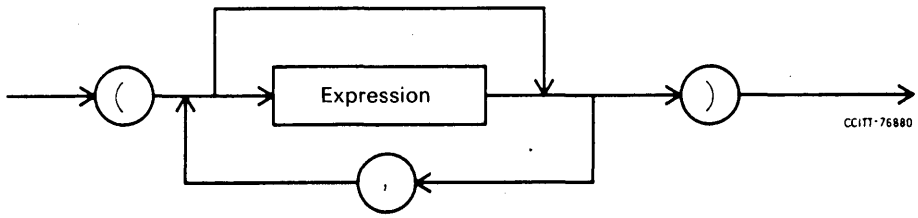
FIN



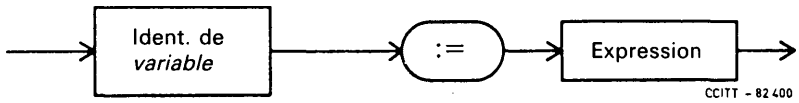
DEMANDE DE CRÉATION



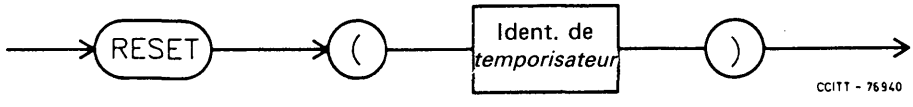
PARAMÈTRES RÉELS



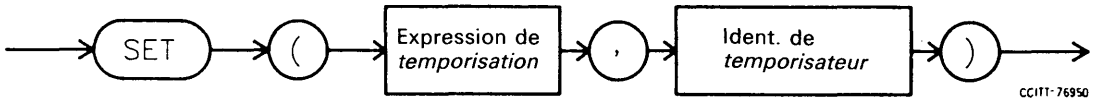
INSTRUCTION D'AFFECTATION



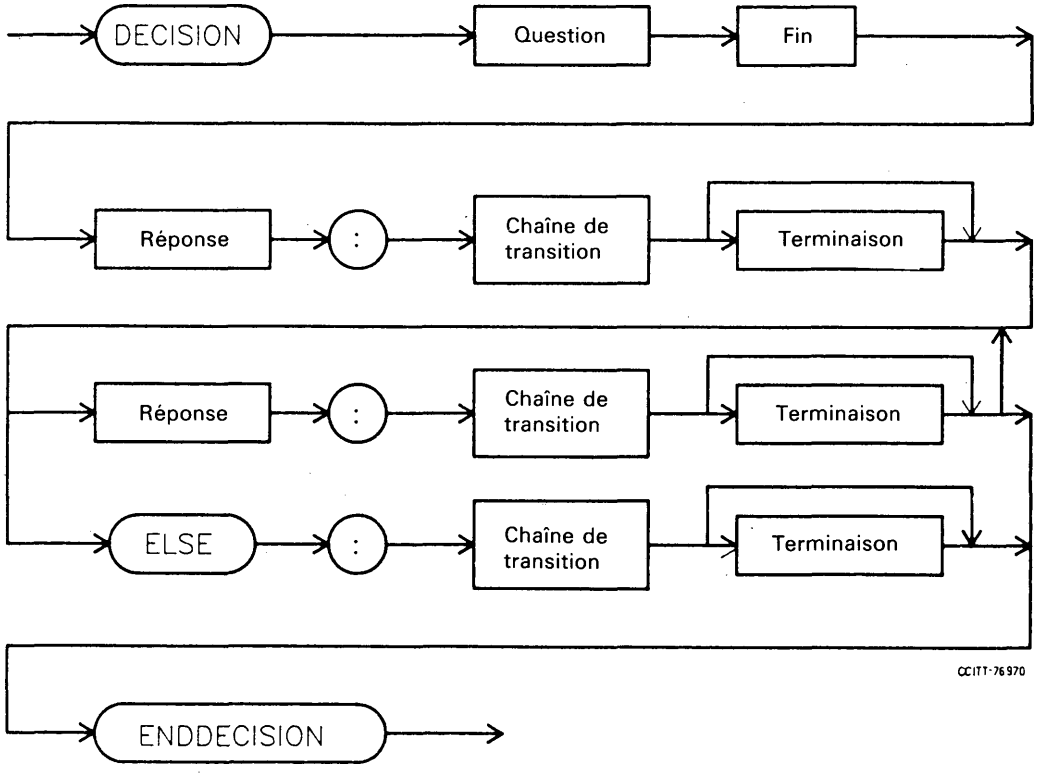
INSTRUCTION DE RÉINITIALISATION



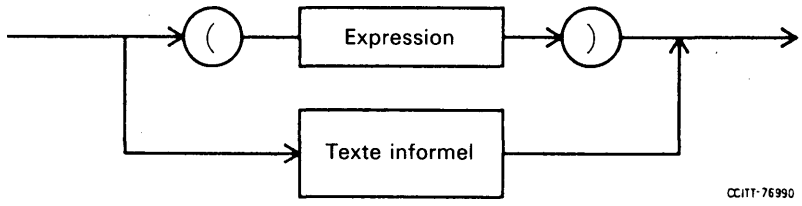
INSTRUCTION
D'INITIALISATION



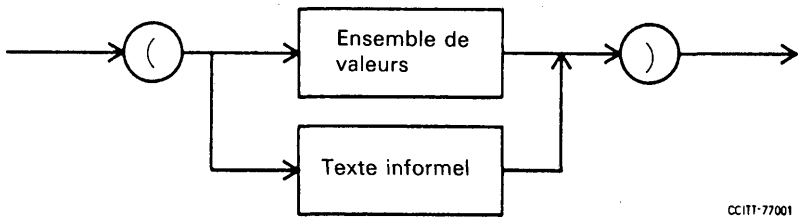
DÉCISION



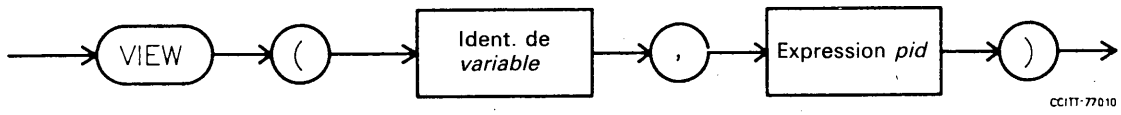
QUESTION



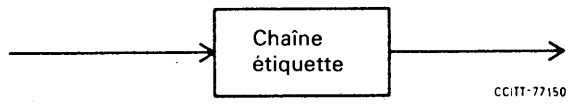
RÉPONSE



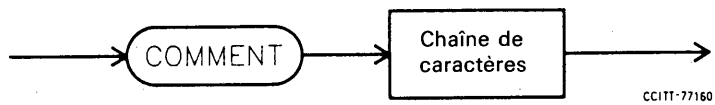
OPÉRATEUR DE VISION



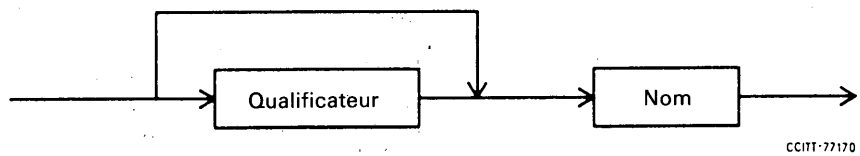
ÉTIQUETTE



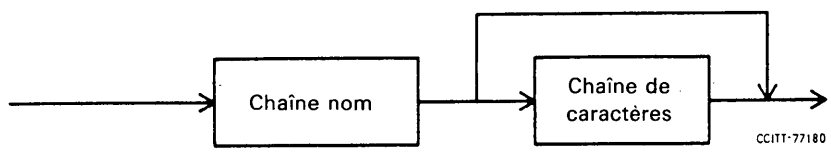
COMMENTAIRE



IDENT.



NOM

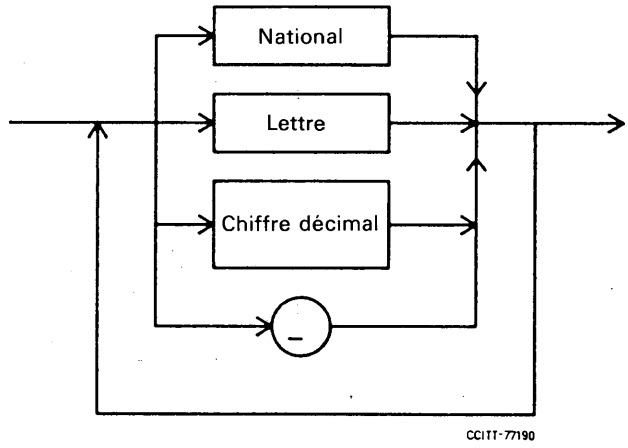


4.4.1 *Unités lexicales*

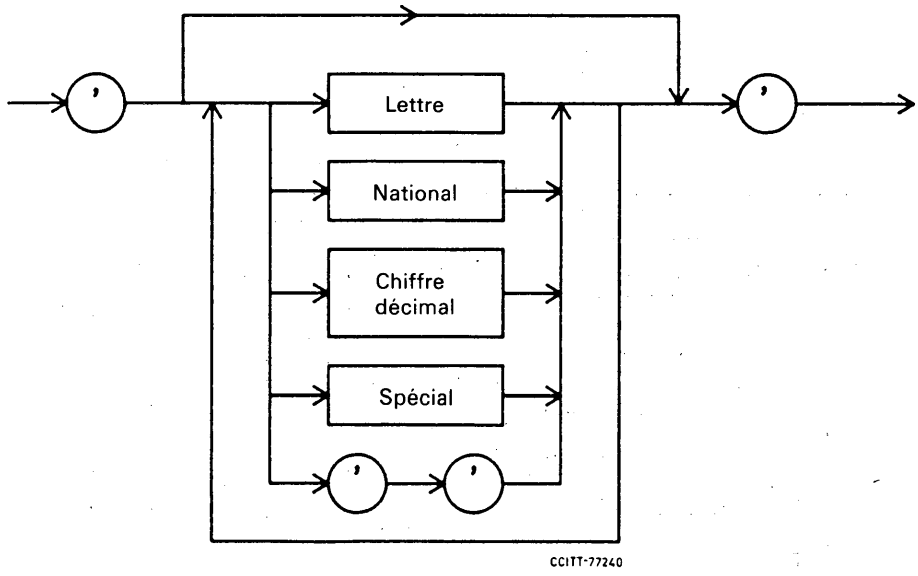
4.4.1.1 *Règles lexicales*

- Tous les signes de ponctuation [par exemple , . ; ' : ! = ( )] et les symboles d'opération (par exemple +, −, *, <,> ...) sont des unités lexicales qui peuvent prendre la place d'espaces.
- Deux unités lexicales doivent être séparées par un ou plusieurs espaces.
- Les mots clés appartiennent à la même catégorie lexicale que la chaîne nom et sont réservés.
- En dehors des unités lexicales, plusieurs espaces ont la même «signification» qu'un seul espace.
- Les caractères de tabulation (VT, HT, CR, BS ...) peuvent être considérés comme des espaces.
- Toutes les lettres et les nationaux sont toujours assimilés à des majuscules, sauf dans une chaîne de caractères.
- Partout où des espaces peuvent apparaître, des commentaires délimités par «/*» et «*/» peuvent être insérés; ces commentaires ont la même signification qu'un espace. Le commentaire ne doit pas contenir la séquence spéciale «*/».

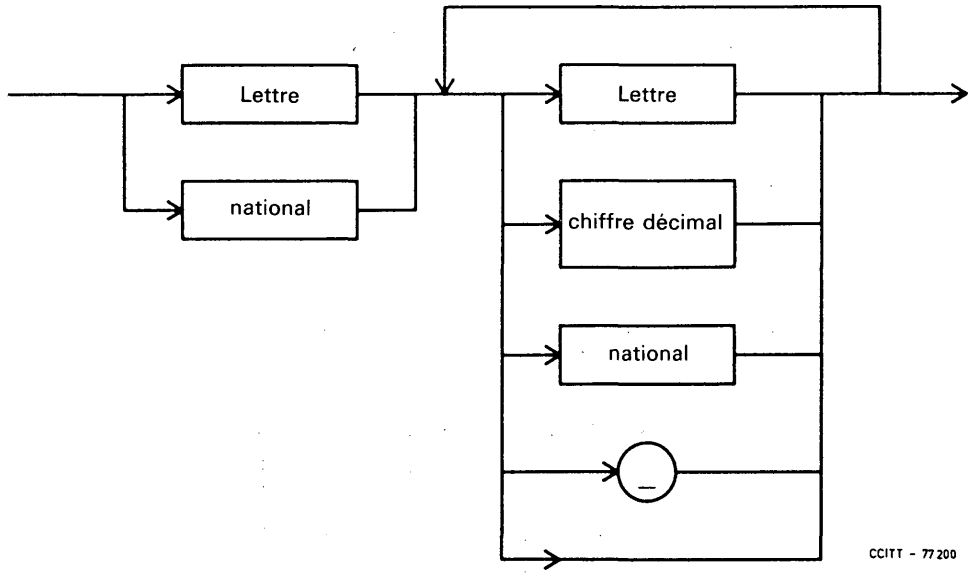
CHAÎNE ÉTIQUETTE



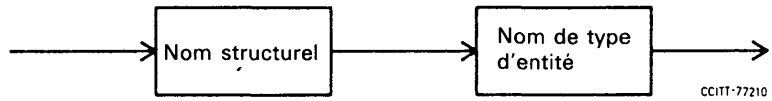
CHAÎNE DE CARACTÈRES



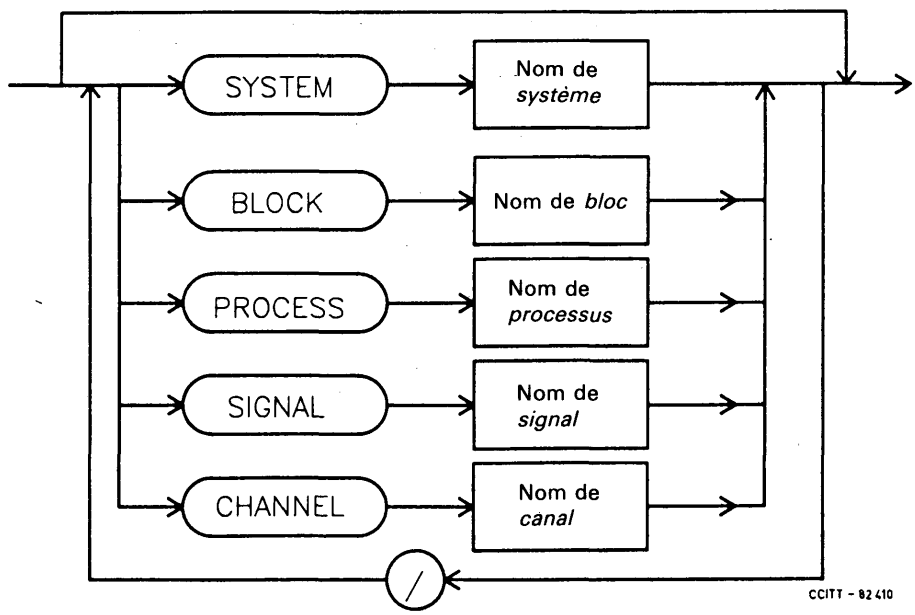
CHAÎNE NOM



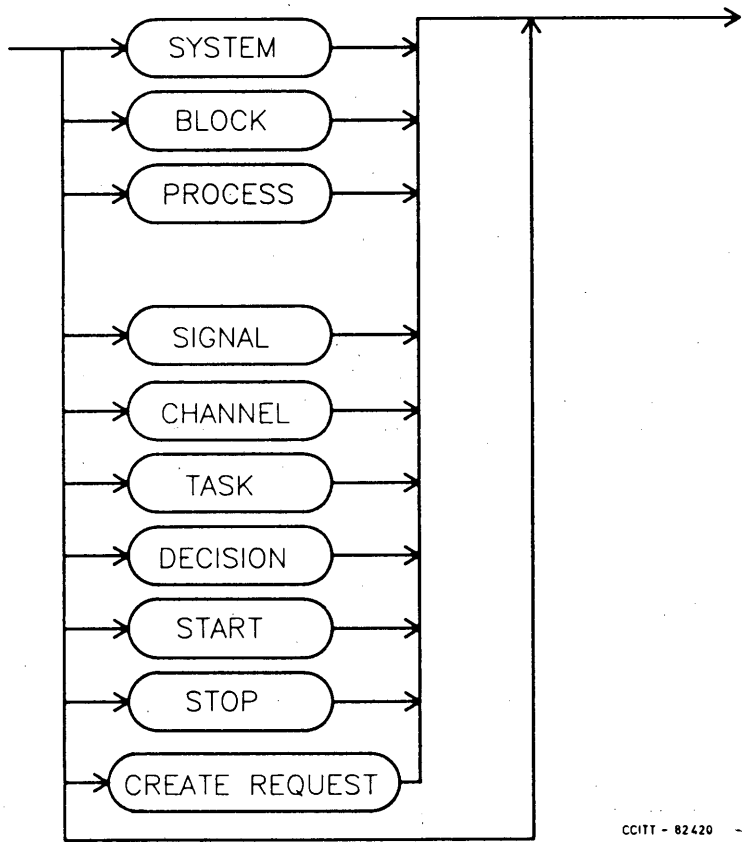
QUALIFICATEUR



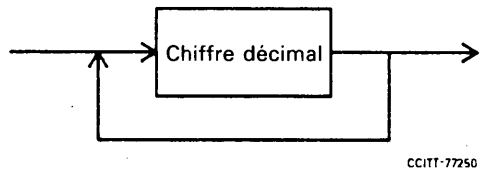
NOM STRUCTUREL



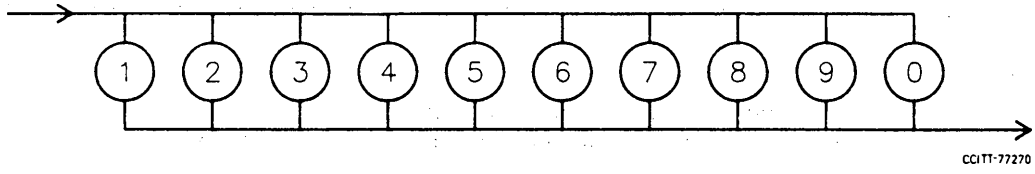
NOM DE TYPE D'ENTITÉ



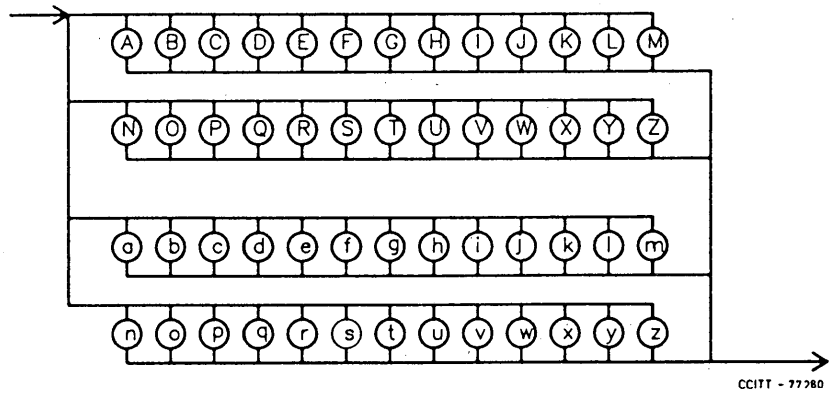
ENTIER DÉCIMAL



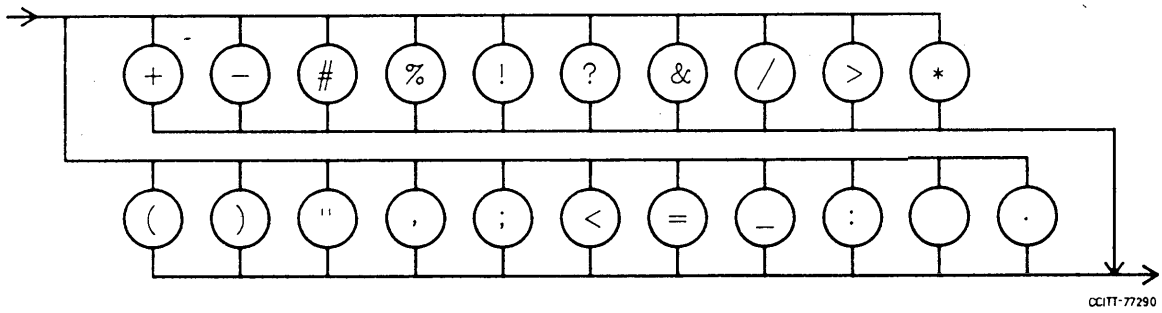
CHIFFRE DÉCIMAL



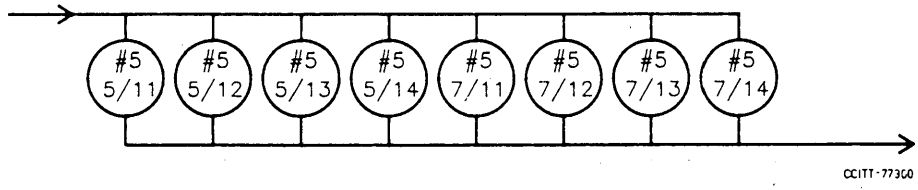
LETTRE



SPÉCIAL

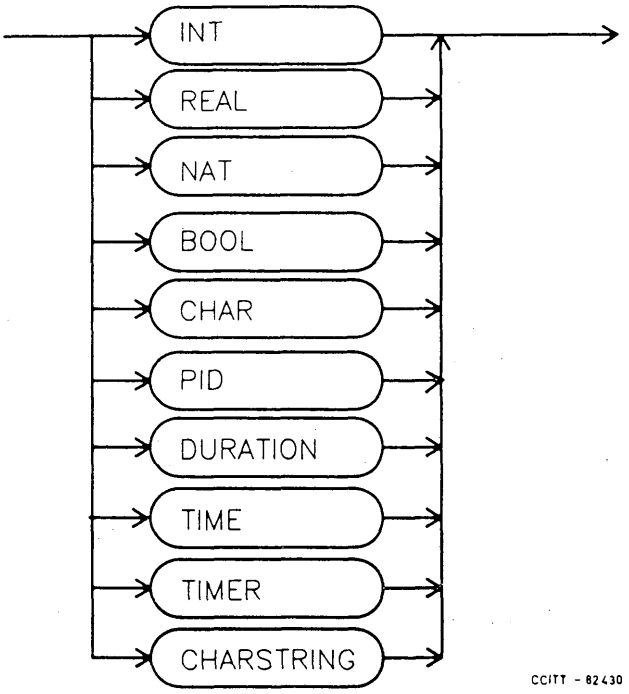


NATIONAL



Les positions représentées ci-dessus sont les positions de l'Alphabet international n° 5 du CCITT réservées pour usage national.

NOM DE TYPES DE DONNÉES PRÉDÉFINIS



Recommandation Z.102

CONCEPTS STRUCTURELS DANS LE LDS

1 Introduction

La présente Recommandation a pour objet de définir un certain nombre de concepts dont on a besoin pour traiter des structures hiérarchiques dans le LDS. La base de ces concepts est le LDS, tel qu'il est défini dans la Recommandation Z.101; ceux qui sont définis ci-après sont de strictes additions à ceux qui sont définis dans la Recommandation Z.101. Il n'y a aucune contradiction entre les définitions de la présente Recommandation et celles des Recommandations Z.103 et Z.104.

Les concepts présentés ci-après ont pour objet de fournir aux utilisateurs du LDS le moyen de décrire des systèmes vastes et/ou complexes. Le LDS, tel que le définit la Recommandation Z.101, convient pour spécifier ou décrire des systèmes relativement petits, que l'on peut comprendre et traiter à un seul niveau de *blocs*. Lorsque l'on est amené à représenter un système plus vaste ou un système complexe, on est forcé d'en subdiviser la *spécification* ou la *description* en unités de gestion que l'on puisse traiter et comprendre indépendamment. Il est souvent indiqué de procéder à cette subdivision en un certain nombre d'étapes, d'où résulte une structure hiérarchique des éléments qui représentent le système.

On est également obligé d'utiliser des concepts structurels afin de spécifier ou de décrire la structure que doit avoir un système ou sa structure réelle.

La présente Recommandation définit des concepts pour la subdivision:

– de *blocs* en *sous-blocs*, en *sous-canaux* et en nouveaux *canaux*,

– de *canaux* en *blocs* et en *canaux*,

– de *processus* en *sous-processus*.

Les concepts sont tels que la structure hiérarchique résultante qui représente le système fournit au lecteur une série de vues d'ensemble lui permettant de se faire un jugement général avant de descendre jusque dans une description plus détaillée. Cela signifie aussi que ces concepts sont à l'origine de techniques de conception visant à un «affinage par étapes» obtenu par adjonction de renseignements plus détaillés en un certain nombre d'étapes.

2 Modèle de langage commun

2.1 *Observations générales*

La Recommandation Z.101 décrit un *système* comme composé d'un ensemble de *blocs* connectés l'un à l'autre et à la *frontière du système* par des *canaux* unidirectionnels. Elle introduit des concepts pour décrire la subdivision des *blocs*, des *canaux* et des *processus* en leurs composantes.

Dans un *système*, chaque *bloc* peut être subdivisé en un ou plusieurs *sous-blocs*. Dans cette *subdivision*, on introduit de nouveaux *canaux* afin de relier les *sous-blocs* les uns aux autres; de plus, les *canaux* qui se terminent au *bloc* subdivisé ou qui y ont leur origine peuvent être scindés en *sous-canaux*.

Le bloc A est subdivisé comme suit:

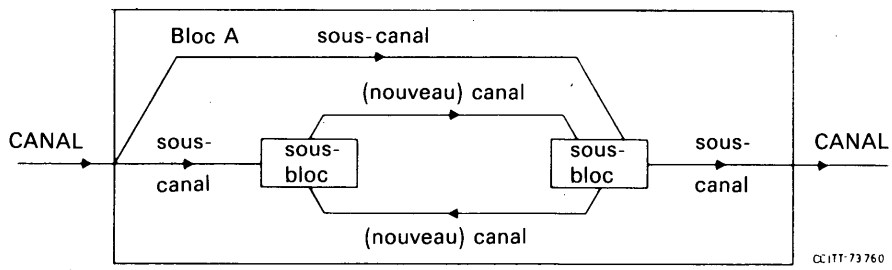


FIGURE 1/Z.102

Subdivision d'un bloc

Un *sous-bloc* est lui-même un *bloc* et on peut le subdiviser. On peut répéter cette *subdivision* autant de fois qu'on le veut et il en résulte une structure hiérarchique de *blocs* et de *sous-blocs*. On dit que les *sous-blocs* d'un *bloc* existent au niveau immédiatement *inférieur* de l'*arbre de blocs*.

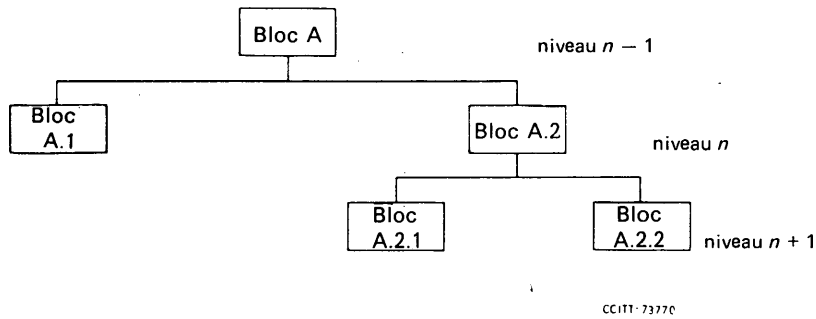


FIGURE 2/Z.102

Arbre de blocs

Un bloc étant subdivisé en *sous-blocs*, la règle de la Recommandation Z.101 qui veut qu'une *définition de bloc* contienne une ou plusieurs *définitions de processus* est assouplie et n'est plus valable que pour un *bloc* qui n'est pas subdivisé davantage; en d'autres termes, le *bloc* «feuille» de l'*arbre des blocs* décrivant le *système* doit contenir des *processus*. Cependant, si une *définition de bloc* contient des *définitions de processus*, les *définitions de sous-bloc* doivent comprendre au moins les *définitions de sous-processus* résultant de la *subdivision* des *processus*.

Les *sous-canaux* résultant de la *subdivision d'un bloc* sont des *canaux*.

On peut aussi subdiviser des *canaux*. De cette opération résultent de nouveaux *blocs*, de nouveaux *canaux*, un *canal d'arrivée* et un *canal de départ*:

$\overrightarrow{C\,[1_1]}$

Le canal C est subdivisé comme suit:

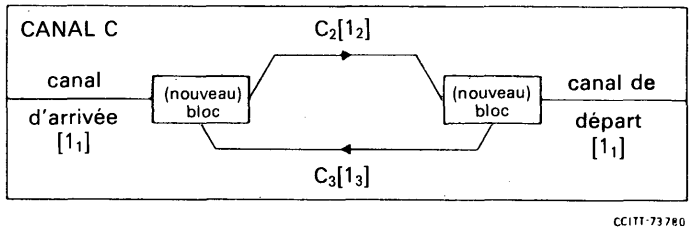


FIGURE 3/Z.102

Subdivision d'un canal

Il est à noter que la *liste des signaux* associée au *canal* C initial est associée aux *canaux d'arrivée* et de *départ*.

Tant la *subdivision* des *blocs* que celle des *canaux* laisse inchangées les interfaces des objets subdivisés.

On peut subdiviser une *définition de processus* en un ensemble de *définitions de sous-processus*. Ces deux descriptions du fonctionnement reviennent au même en ce sens que, lorsque l'on interprète la *définition du système*, on interprète soit l'ensemble des *définitions des sous-processus*, soit la *définition du processus*. La *définition du processus* doit être considérée comme une variante de la description constituée par l'ensemble des *définitions des sous-processus*.

Un *sous-processus* est un *processus* et on peut à son tour le subdiviser en *sous-processus*. La hiérarchie de *processus* qui en résulte est représentée sous la forme d'un *arbre de processus*.

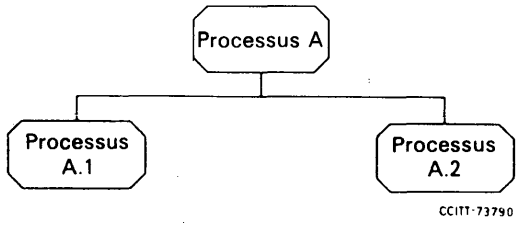


FIGURE 4/Z.102

Relation «processus – sous-processus»

Les *sous-processus* étant des *processus*, le fonctionnement décrit par le *processus* subdivisé est subdivisé en un ensemble de «sous-fonctionnements» simultanés. Il faut des méthodes qui permettent de garantir que cette *subdivision* est correcte, mais elles ne font pas partie du LDS.

La *subdivision* d'un *bloc* en *sous-blocs* et celle de ses *processus* en *sous-processus* peuvent se faire en même temps. Etant donné qu'un *processus* doit être contenu dans un *bloc*, ses *sous-processus* doivent être contenus dans des *sous-blocs* du *bloc* qui contient le *processus* subdivisé. Les *blocs* et *processus* subdivisés peuvent être considérés comme des structures séparées reliées les unes aux autres:

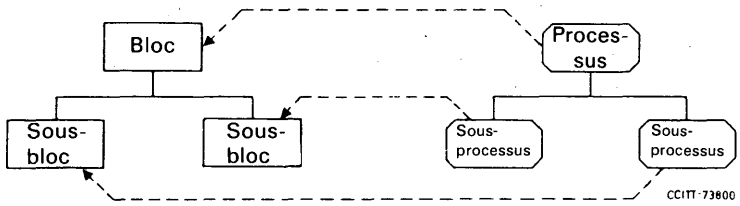


FIGURE 5/Z.102

Relation entre un arbre de blocs et un arbre de processus

Si un *bloc* est subdivisé, tous les *processus* qu'il contient peuvent également l'être, auquel cas tous ses *sous-processus* doivent apparaître dans les *sous-blocs* du *bloc*. Si le *processus* n'est pas subdivisé, il doit apparaître dans l'un des *sous-blocs*. Pour décrire plus avant le comportement des *sous-blocs*, de *nouveaux processus* peuvent aussi être inclus dans les *sous-blocs*, qu'ils aient été ou non des *processus* du *bloc*. Cependant, les *blocs* «feuilles» doivent contenir des *processus*.

Si, dans la représentation totale de la *subdivision* d'un *système*, les *définitions de processus* apparaissent à plus d'un *niveau*, on peut trouver plusieurs sous-ensembles cohérents de cette représentation. Un sous-ensemble cohérent est une sélection des *définitions de bloc* et des *définitions de processus* de la représentation totale répondant aux conditions suivantes:

a) il contient le *niveau* le plus élevé de l'*arbre de blocs*;

b) s'il contient un *bloc*, il doit contenir son parent;

c) s'il contient un *sous-bloc* d'un *bloc*, il doit contenir aussi tous ses autres *sous-blocs*;

d) tous les *blocs* «feuilles» de la structure résultante contiennent des *processus*.

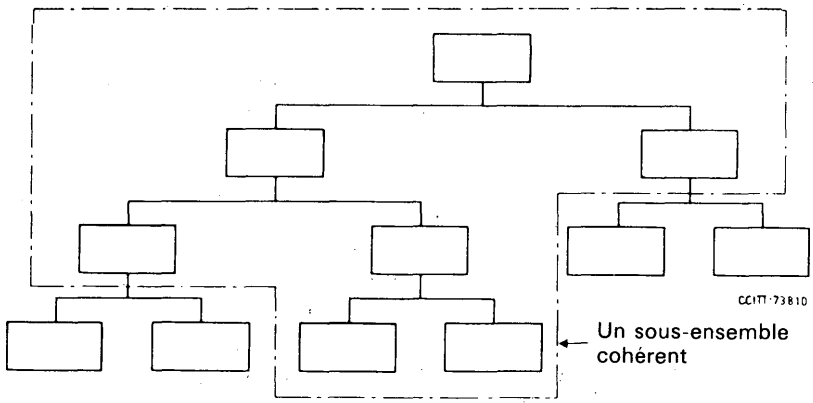


FIGURE 6/Z.102

Sous-ensemble cohérent d'une représentation d'un système

Si des *processus* apparaissent dans une *définition de systèmes* à plus d'un niveau, on peut trouver plusieurs sous-ensembles cohérents des représentations. Ces sous-ensembles représentent des variantes des descriptions du *système*, plus ou moins détaillées. On peut s'en servir pour procurer aux lecteurs à la fois des vues d'ensemble et des descriptions détaillées. On peut les choisir de manière telle qu'ils concrétisent l'interprétation du *système* à divers niveaux d'abstraction au choix. Lorqu'il existe plusieurs variantes de la représentation du fonctionnement, on doit les considérer toutes, à l'exception de la plus détaillée, comme des vues d'ensemble du fonctionnement.

2.2 *Syntaxe abstraite*

La *syntaxe abstraite* ci-dessous est fondée sur celle de la Recommandation Z.101. Seules les adjonctions aux définitions de ladite Recommandation sont données dans ce qui suit:

Définition de bloc

Une *définition de bloc* peut contenir aussi une *définition de bloc de partie interne*; si tel est le cas, il n'est pas nécessaire qu'elle contienne des *définitions de processus.*

Définition de bloc de partie interne

La *définition de bloc de partie interne* contient une seule *définition de sous-structure de bloc* et elle contient une seule définition de *sous-structure de processus* pour chacune des *définitions de processus* contenues dans la *définition de bloc.* Elle peut aussi contenir des *définitions de signal* et des *définitions de type de données.*

Définition de sous-structure de bloc

Une *définition de sous-structure de bloc* peut contenir une ou plusieurs *définitions de sous-bloc* et une ou plusieurs *définitions de canal.*

Pour chacun des points d'extrémité terminaux des *sous-canaux* du *bloc* englobant, il doit y avoir au moins une *définition de sous-canal* ayant ce point d'extrémité comme point d'extrémité d'origine, et l'inverse doit être vrai pour tous les points d'extrémité du *sous-canal* d'origine du *bloc* englobant. L'union des *listes de signaux* des *définitions de sous-canal* ayant le même point d'extrémité en tant que *canal* aboutissant au *bloc* englobant, ou en partant, doit être identique à la *liste des signaux* de ce *bloc*; de plus, les *listes des signaux* des *définitions de sous-canal* au départ d'un point d'extrémité terminal doivent être disjointes.

Toutes les *définitions de canal* contenues dans la définition de *sous-structure de bloc* doivent connecter soit des *sous-blocs* à des points d'extrémité de *canal* du *bloc* englobant, soit des *sous-blocs* entre eux.

Définition de sous-bloc

Une *définition de sous-bloc* est une *définition de bloc.*

Définition de canal

Une *définition de canal* peut aussi contenir une *définition de sous-structure de canal.*

Définition de sous-canal

Une *définition de sous-canal* est une *définition de* canal.

Définition de sous-structure de canal

Une *définition de sous-structure de canal* contient au moins deux *définitions de canal*, au moins une *définition de bloc* et peut contenir des *définitions de signal.*

Toutes les *définitions* de *canal* contenues dans la *définition de sous-structure de bloc* peuvent relier les *sous-blocs* entre eux et toutes les *définitions de sous-canal* peuvent relier les extrémités du *canal* englobant aux sous-blocs.

Définition de sous-structure de processus

Une *définition de sous-structure de processus* est associée à un *nom de processus* et contient un ou plusieurs *noms de processus*, dont chacun est associé à un *nom de sous-bloc.*

Le *nom de processus* associé peut être celui d'une *définition* de *processus* contenue dans la *définition de bloc* englobant. Les *noms de processus* contenus doivent être des *noms* de *définition de processus* contenue dans la *définition de bloc* ayant le *nom de sous-bloc* associé.

Chaque *nom de signal* contenu dans l'ensemble des *signaux d'entrée valables du processus* associé doit apparaître dans un et un seul des ensembles de *signaux d'entrée valables* des *définitions de sous-processus* ayant les *noms de sous-processus* contenus. Chaque *nom de signal* attaché *au* nœud de *sortie* du *processus* associé doit être attaché à au moins un *nœud de sortie* des *définitions de sous-processus* ayant les *noms de sous-processus* contenus.

Définition de sous-processus

Une *définition de sous-processus* est une *définition de processus.*

2.3 *Interprétation*

Les règles d'interprétation ci-dessous sont définies en tant qu'adjonctions à l'ensemble correspondant de règles définies dans la Recommandation Z.101.

Canal

Si une *définition de canal* contient une *définition de sous-structure de canal*, ou bien le *canal* peut être interprété conformément à la définition de la Recommandation Z.101, ou bien la *définition de sous-structure de canal* peut être interprétée.

Si la *définition de sous-structure de canal* est interprétée, tout *signal* fourni au point d'extrémité d'origine du *canal* est donné à la *sous-structure de canal*, et tout *signal* fourni par la *sous-structure de canal* est donné au point d'extrémité terminal du *canal*.

Sous-structure de canal

Un *signal* fourni à la *sous-structure de canal* est donné au *canal d'arrivée*, et un *signal* fourni par le *point d'extrémité terminal* du *canal de départ* est fourni au *canal* englobant.

Bloc

Si une *définition de bloc* contient une ou plusieurs *définitions de processus*, ainsi qu'une *définition de bloc de partie interne*, ou bien le *bloc* peut être interprété, conformément à la définition de la Recommandation Z.101, ou bien le *bloc de partie interne* peut être interprété. Si le *bloc* ne contient pas de *définition de processus*, il faut interpréter le *bloc de partie interne*.

Si le *bloc de partie interne* est interprété, tous les *signaux* fournis au *bloc* seront donnés au *bloc de partie interne*, et tous les *signaux* fournis par le *bloc de partie interne* seront en outre fournis aux *canaux* partant du *bloc* de la même manière que s'ils étaient fournis par un *processus* contenu dans le *bloc*.

Bloc de partie interne

Tout processus du *bloc* englobant est remplacé par une *sous-structure de processus*.

Si un *signal* donné au *bloc de partie interne* provenant du *bloc* englobant est adressé à un *processus*, ce *signal* est donné à la *sous-structure de processus*; sinon, il est donné à la *sous-structure de bloc*.

Un *signal* fourni par la *sous-structure de bloc* sera donné au *bloc* englobant. Si ce *signal* a été envoyé d'un *processus* qui apparaît comme un *sous-processus* d'une *sous-structure de processus*, l'attribut-expéditeur du *signal* sera remplacé par l'*identificateur d'instance de processus* du *processus* remplacé.

Sous-structure de bloc

Une *sous-structure de bloc* contient des *blocs* et des *canaux*, selon la *définition de sous-structure de bloc*; ceux-ci sont interprétés selon les règles définies pour les *blocs* et les *canaux*.

Sous-structure de processus

Une *instance de sous-structure de processus* remplace une *instance de processus* de la *définition de processus* référencée. Elle dénote également un ensemble d'*instances de sous-processus*, à raison d'une par *nom de sous-processus* de la *définition*. Chaque *sous-processus* est alloué au *bloc* avec le *nom de bloc* associé.

Chaque *signal* donné à la *sous-structure de processus* sera réadressé au *sous-processus* qui a le *nom du signal* dans l'ensemble de *signaux d'entrée* valable, et sera donné à la *sous-structure de bloc* du *bloc* englobant.

3 Syntaxe graphique

La syntaxe graphique ci-dessous est une adjonction à celle qui est définie dans la Recommandation Z.101. Ces adjonctions s'étendent à la représentation de la structure et de la *subdivision* d'un *système*.

Le *diagramme d'arbre de blocs* donne une vue d'ensemble de la structure d'un *système*. La *subdivision* des *blocs*, des *processus* et des *canaux* en leurs éléments composants est représentée dans le *diagramme d'interaction de blocs* et dans le *diagramme de sous-structure de canal*.

L'ensemble des documents et des diagrammes qui décrivent le *système* peut être volumineux. Il est essentiel que les documents renvoient de l'un à l'autre à l'aide de références et de titres adéquats; cependant, les moyens syntaxiques pertinents ne font pas partie de la syntaxe graphique du LDS.

3.1 *Diagramme d'arbre de blocs*

Le *diagramme d'arbre de blocs* a pour objet de donner une vue d'ensemble de la struture d'un *système*, c'est-à-dire de la *subdivision* de ce *système* en une structure hiérarchique de *blocs*. Les indications détaillées sur la façon dont les *blocs* sont reliés par des *canaux* sont données par le *diagramme d'interaction des blocs*.

On peut également utiliser une partie du diagramme pour donner une vue d'ensemble de la façon dont un *bloc* est *subdivisé* en *sous-blocs*. En pareil cas, le *bloc* subdivisé est indiqué comme bloc racine.

3.1.1 *Symboles*

Le symbole utilisé dans un *diagramme d'arbre de blocs* est une case qui représente un *système* ou un *bloc*. Le *nom* de l'objet représenté doit figurer à l'intérieur de la case.

Chaque *bloc* (case) est relié vers le bas à ses *sous-blocs* (cases) de manière à former un arbre hiérarchique, comme dans l'exemple de la figure 7/Z.102 ci-dessous.

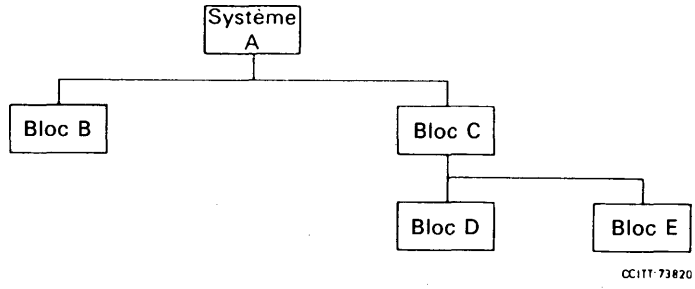


FIGURE 7/Z.102

Exemple d'un diagramme d'arbre de blocs

3.1.2 *Relation avec la syntaxe abstraite du LDS*

La structure dont il s'agit a son équivalent dans la *définition de système* et dans les *définitions de sous-structure* des *blocs* dans le *système.*

3.1.3 *Conventions graphiques*

La meilleure chose à faire est de dessiner l'arbre de telle manière que les *blocs* d'un même *niveau* figurent côte à côte dans la représentation.

Le *diagramme d'arbre de blocs* d'un grand système étant étendu, il peut être bon de le scinder en plusieurs diagrammes. Cette opération devrait être telle que le premier diagramme, ayant le *système* pour racine, soit subdivisé de manière qu'un ensemble de *blocs* subdivisés plus avant apparaisse comme non subdivisé. Dans le diagramme ci-dessous, ces *blocs* apparaissent comme des racines. Par exemple, dans la figure 8/Z.102, le diagramme de la figure 7/Z.102 est scindé en deux.

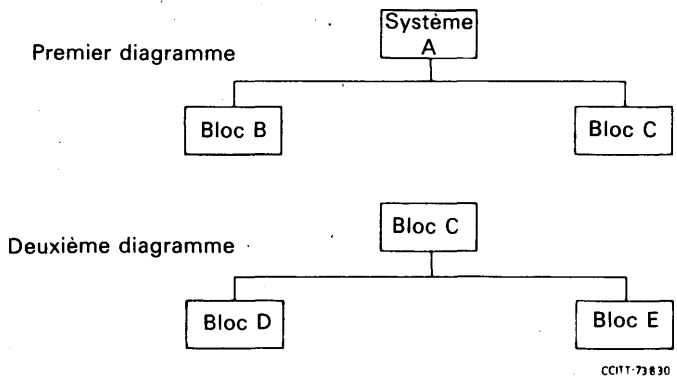


FIGURE 8/Z.102

Exemple montrant comment un diagramme d'arbre de blocs est scindé en plusieurs diagrammes

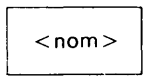
3.2 *Diagramme d'interaction de blocs*

Ce diagramme représente la *subdivision* d'un *bloc* en *sous-blocs*, en *sous-canaux* et en (nouveaux) *canaux*. Sa forme est essentiellement la même que celle du *diagramme d'interaction de blocs*, présenté dans la Recommandation Z.101, qui représente la subdivision d'un *système* en *blocs* et en *canaux*.

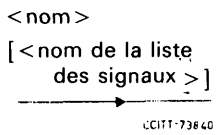
3.2.1 *Symboles*

Les symboles utilisés pour représenter un *diagramme d'interaction de blocs* sont ceux de la figure 9/Z.102.

Cadre
Le cadre entoure le diagramme et représente la frontière du *bloc ou système* subdivisé.



Symbole de sous-bloc
La case représente un *sous-bloc*, dont le *nom* doit figurer à l'intérieur de ce symbole.



Symbole de canal
Ce symbole représente un *canal* ou un *sous-canal*. Le *nom* du *canal* doit figurer à côté du symbole. On peut y associer une *liste de signaux* optionnelle (voir la Recommandation Z.101, § 2.2.1).

FIGURE 9/Z.102

Symboles utilisés dans un diagramme d'interaction de blocs

En plus de ces symboles, peuvent figurer dans le diagramme des *définitions de signal* utilisant la syntaxe LDS/PR.

Les règles de connexion des symboles sont les mêmes que pour le *diagramme d'interaction des blocs* (voir la Recommandation Z.101), à la seule exception près que, dans le titre du diagramme, il doit être précisé qu'il s'agit d'un *diagramme d'interaction de blocs* pour un *bloc*. La figure 10/Z.102 donne un exemple simple de *diagramme d'interaction de blocs*:

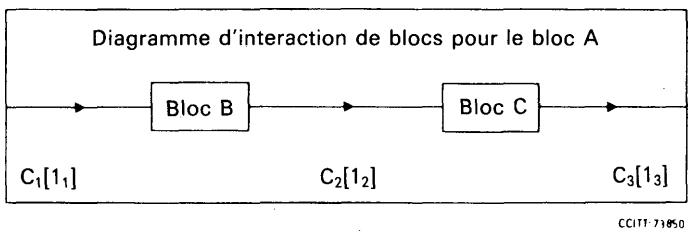


FIGURE 10/Z.102

Exemple de diagramme d'interaction de blocs

3.2.2 *Relation avec la syntaxe abstraite du LDS*

Un *diagramme d'interaction de blocs* représente une *définition de sous-structure de bloc*. Les définitions contenues dans les *définitions de sous-structure de bloc* sont représentées par les symboles de *bloc* et de *canal* ainsi que par des *définitions de signal* de la syntaxe *LDS/PR*.

3.2.3 *Conventions graphiques*

Les conventions graphiques décrites pour le *diagramme d'interaction* défini dans la Recommandation Z.101 s'appliquent au *diagramme d'interaction de blocs.*

En outre, il est souvent utile de décrire plusieurs *niveaux* de *subdivision de bloc* dans un seul et même diagramme. C'est ce que l'on obtient en remplaçant, dans un diagramme, un *symbole de bloc* par le *diagramme d'interaction de blocs* pour ce *bloc.* La figure 11/Z.102 en donne un exemple.

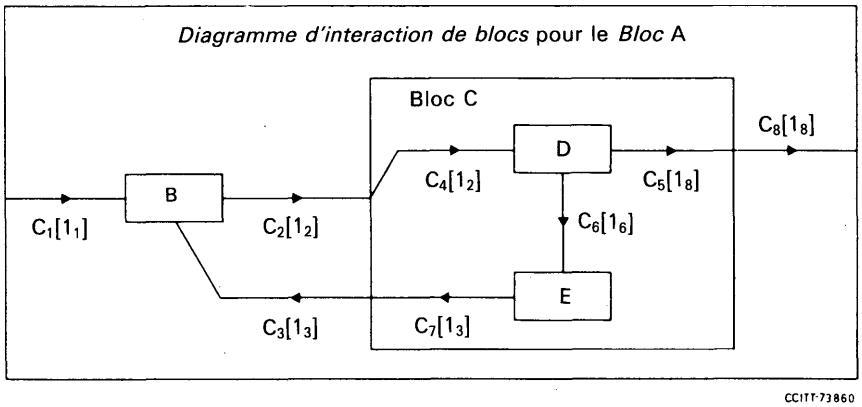


FIGURE 11/Z.102

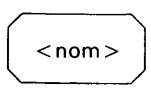
Exemple de diagramme d'interaction de blocs emboîté

3.3 *Diagramme d'arbre de processus*

Un *diagramme d'arbre de processus* décrit la *subdivision* d'un *processus* en *sous-processus* et indique où ces *sous-processus* sont alloués.

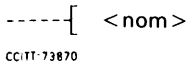
3.3.1 *Symboles*

Les symboles utilisés pour composer un *diagramme d'arbre de processus* sont indiqués sur la figure 12/Z.102 ci-dessous:



Symbole de processus

Ce symbole représente un processus, dont le *nom* doit figurer à l'intérieur du symbole.



CCITT-73870

Symbole d'allocation

L'objet de ce symbole de commentaire est de faire savoir où est alloué le *processus* ou le *sous-processus.* Le *nom* ajouté à titre d'explication doit être celui du *bloc* auquel le *processus* est alloué.

FIGURE 12/Z.102

Symboles utilisés dans un diagramme d'arbre de processus

Chaque *processus* est relié vers le bas à ses *sous-processus* de manière à former un arbre hiérarchique, comme dans l'exemple de la figure 13/Z.102.

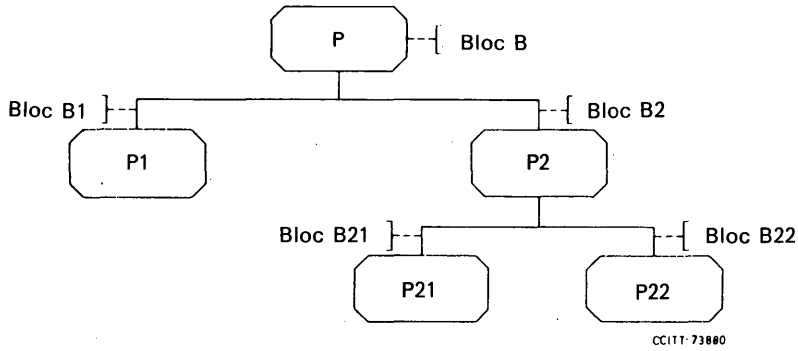


FIGURE 13/Z.102

Exemple d'un diagramme d'arbre de processus

3.3.2 *Relation avec la syntaxe abstraite du LDS*

Le diagramme représente une *définition de sous-structure de processus*. Le *nom* inscrit dans le symbole de la racine est le nom de *processus* associé, et les *noms* inscrits dans les symboles «feuilles» sont les *noms de sous-processus* contenus. Les *noms de bloc* inscrits dans les symboles d'allocation sont les *noms de sous-bloc* associés.

3.3.3 *Conventions graphiques*

La meilleure chose à faire est de dessiner l'arbre de telle manière que les *processus* d'un même *niveau de subdivision* figurent côte à côte dans le diagramme.

Si un *diagramme d'arbre de processus* est étendu, il peut être bon de le subdiviser en plusieurs diagrammes. Cette opération devrait être telle que le premier diagramme soit découpé de manière qu'un ensemble de *processus* subdivisés plus avant apparaisse comme non subdivisé. Dans les diagrammes ci-dessous, ces *processus* apparaissent comme des racines. Par exemple, dans la figure 12/Z.102, le diagramme de la figure 14/Z.102 est scindé en deux.

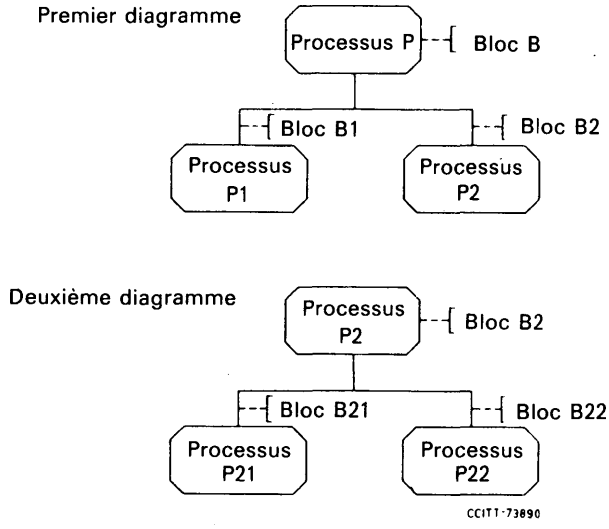


FIGURE 14/Z.102

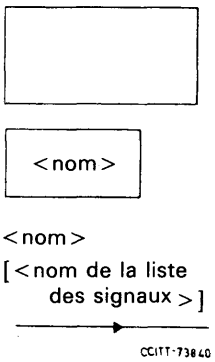
Exemple montrant comment un diagramme d'arbre de processus est scindé en plusieurs diagrammes

3.4 *Diagramme de sous-structure de canal*

Ce diagramme représente la *subdivision* d'un *canal* entre ses composants. Ces derniers étant des *blocs* et des *canaux*, le diagramme ressemble à un *diagramme d'interaction de blocs.*

3.4.1 *Symboles*

Les symboles utilisés pour représenter un diagramme de *sous-structure de canal* sont exposés sur la figure 15/Z.102.



Cadre
Le cadre entoure le diagramme et représente le *canal* subdivisé.

Symbole de bloc
La case représente un *bloc*, dont le *nom* doit figurer à l'intérieur de ce symbole.

Symbole de canal
Ce symbole représente un *sous-canal* ou un nouveau *canal*. Le *nom* du *canal* doit figurer à côté du symbole. Une *liste des signaux* optionnelle peut y être associée (Recommandation Z.101, § 2.2.1).

FIGURE 15/Z.102

Symboles utilisés dans un diagramme de sous-structure de canal

En plus de ces symboles, peuvent figurer dans le diagramme des *définitions de signal* et des *définitions de type de données*, utilisant la syntaxe LDS/PR.

Les règles de connexion de ce diagramme sont les mêmes que pour le *diagramme d'interaction* (voir la Recommandation Z.101), exception faite que le titre du diagramme doit indiquer clairement qu'il s'agit d'un *diagramme de sous-structure de canal.*

La figure 16/Z.102 donne un exemple simple de *diagramme de sous-structure de canal.*

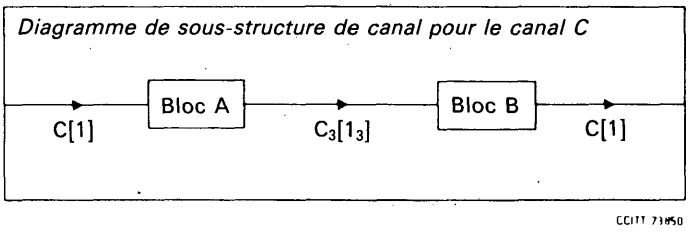


FIGURE 16/Z.102

Exemple de diagramme de sous-structure de canal

3.4.2 *Relation avec la syntaxe abstraite du LDS*

Un *diagramme de sous-structure de canal* représente une *définition de sous-structure de canal.* Les définitions contenues dans la *définition de sous-structure de canal* sont représentées par les symboles de *bloc* et de *canal* ainsi que par les définitions de la syntaxe *LDS/PR.*

Le *canal* partant du cadre et allant vers l'intérieur du diagramme représente le *canal d'arrivée* et celui qui se termine au cadre représente le *canal de départ.*

3.4.3 *Conventions graphiques*

Les conventions graphiques décrites pour le *diagramme d'interaction* défini dans la Recommandation Z.100 s'appliquent au *diagramme de sous-structure de canal.*

4 LDS/PR

Cette syntaxe LDS/PR s'ajoute à la syntaxe définie dans la Recommandation Z.101. Les adjonctions couvrent la représentation de la structure et la *subdivision* d'un *système.*

Il est à noter que, dans les exemples, les mots clés *LDS/PR* figurent en majuscules.

4.1 *Définition de bloc*

Dans le *LDS/PR*, la *définition de bloc* est étendue de manière à inclure, à titre facultatif, le non-terminal «*définition de sous-structure de blocs*».

4.1.1 *Syntaxe*

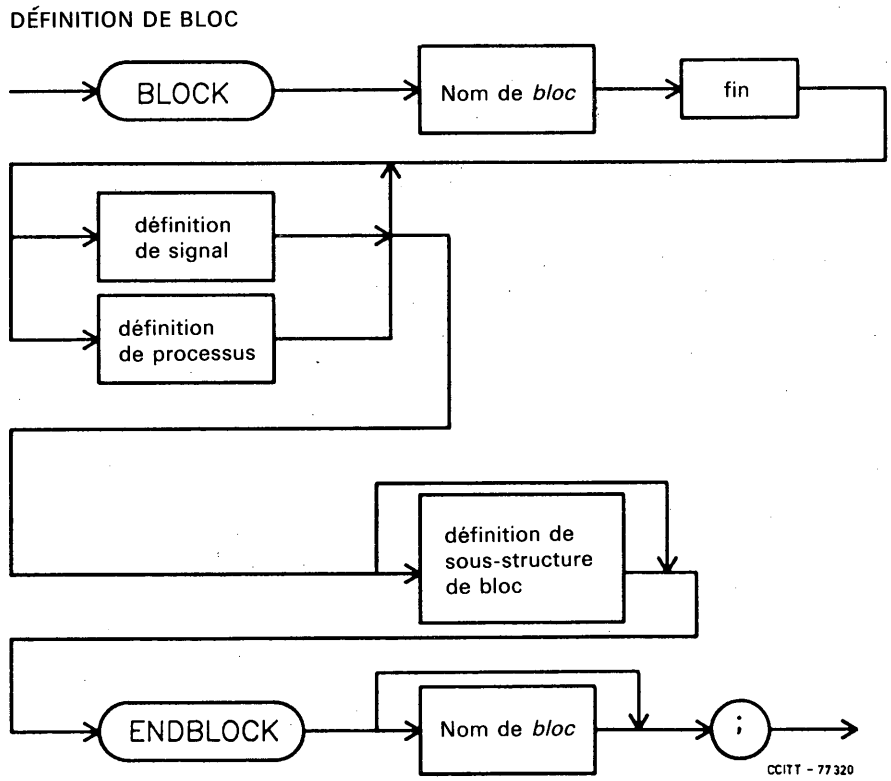


FIGURE 17/Z.102

Diagramme de syntaxe pour bloc

4.2 *Définition de sous-structure de bloc*

La *définition de sous-structure de bloc* représente la *subdivision* d'un *bloc* en *sous-blocs*, *sous-canaux* et nouveaux *canaux.*

4.2.1 *Syntaxe*

DÉFINITION DE SOUS-STRUCTURE DE BLOC

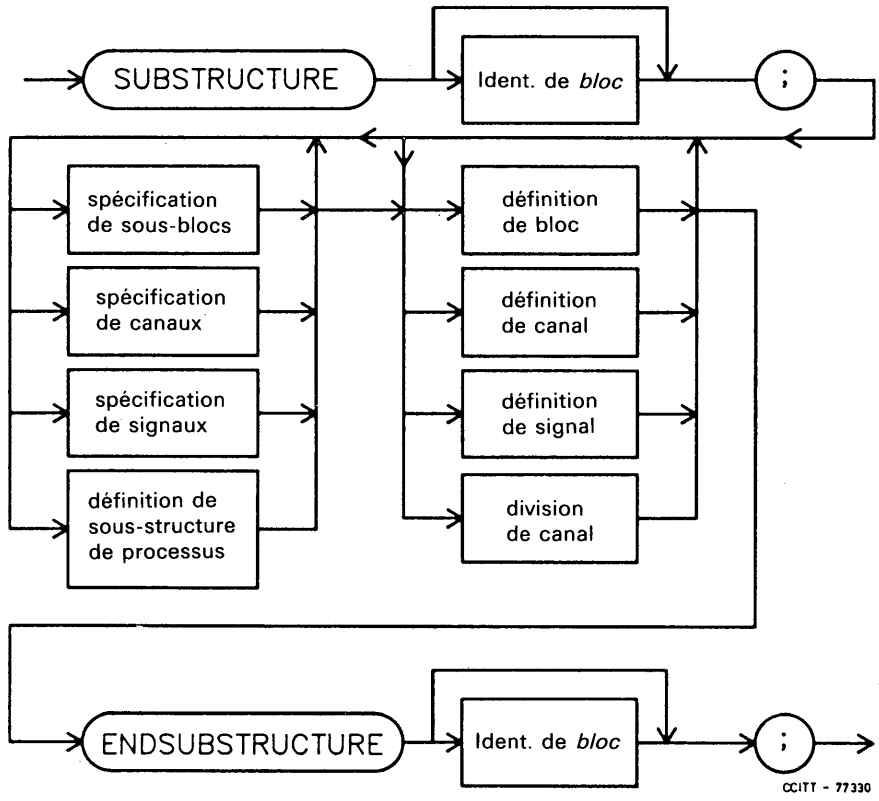
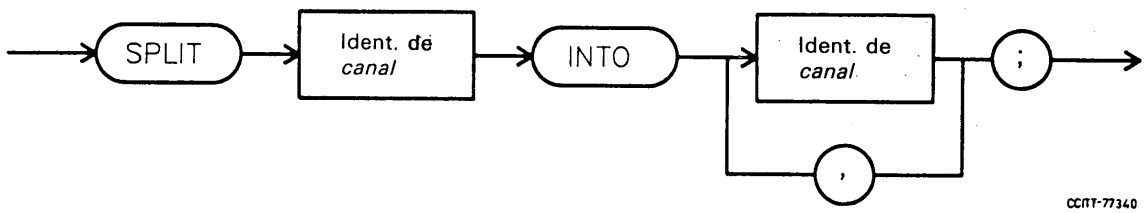


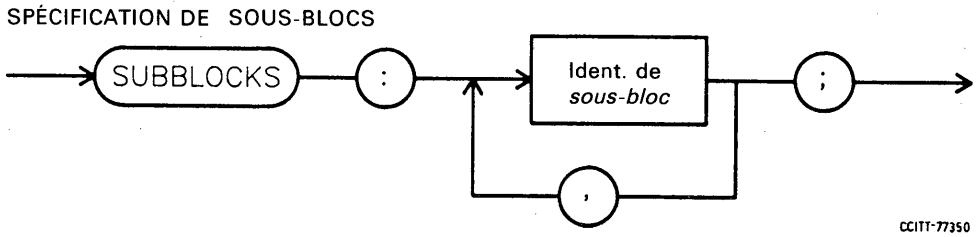
FIGURE 18/Z.102

Diagramme de syntaxe pour définition de sous-structure de bloc

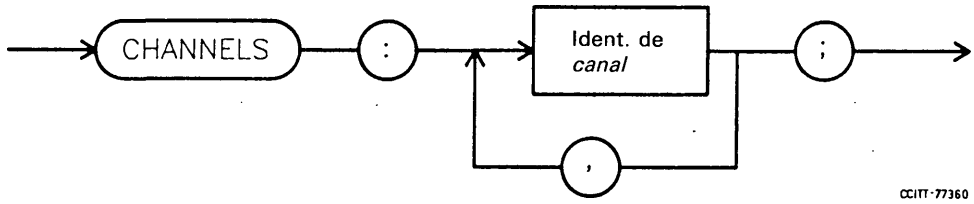
DIVISION DE CANAL



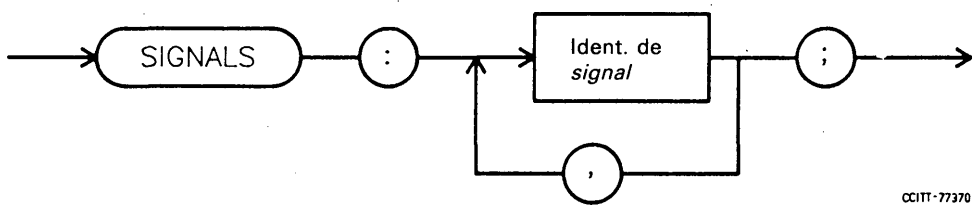
SPÉCIFICATION DE SOUS-BLOCS



SPÉCIFICATION DE CANAUX



SPÉCIFICATION DE SIGNAUX



Exemple:

```
BLOCK b:

    ...
    SUBSTRUCTURE b;
        SUBBLOCKS:  b1,b2,b3;
        CHANNELS:   c1,c2,d1,d2,e;
        SIGNALS:    s1,s2,s3;

    SPLIT c INTO c1,c2;
    SPLIT d INTO d1,d2;

    . . .
        /* Définitions de canaux */
        /* Définitions de blocs */
        /* Définitions de signaux */

    ENDSUBSTRUCTURE;

ENDBLOCK b.
```

4.2.2 *Relation avec la syntaxe abstraite LDS*

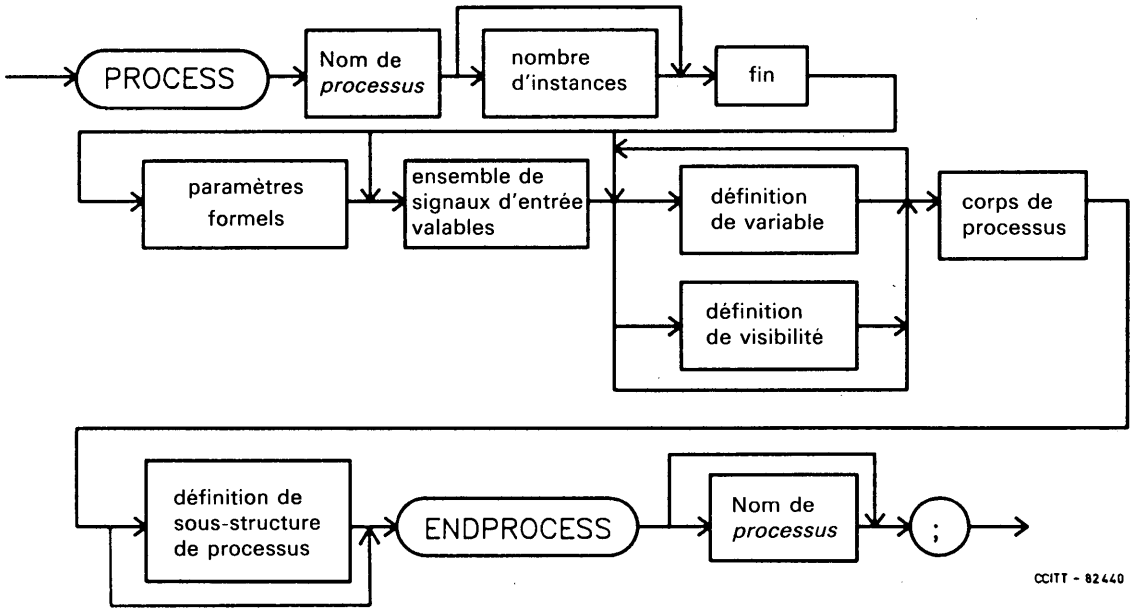
Cette syntaxe présente la *définition de sous-structure de bloc* et la *définition de bloc de partie interne* de la *syntaxe abstraite.* Les *noms* des *blocs,* des *signaux* et des *canaux* sont des références aux *définitions* de bloc, aux *définitions de signal* et aux *définitions de* canal qui y sont contenues.

4.3 *Sous-structure de processus*

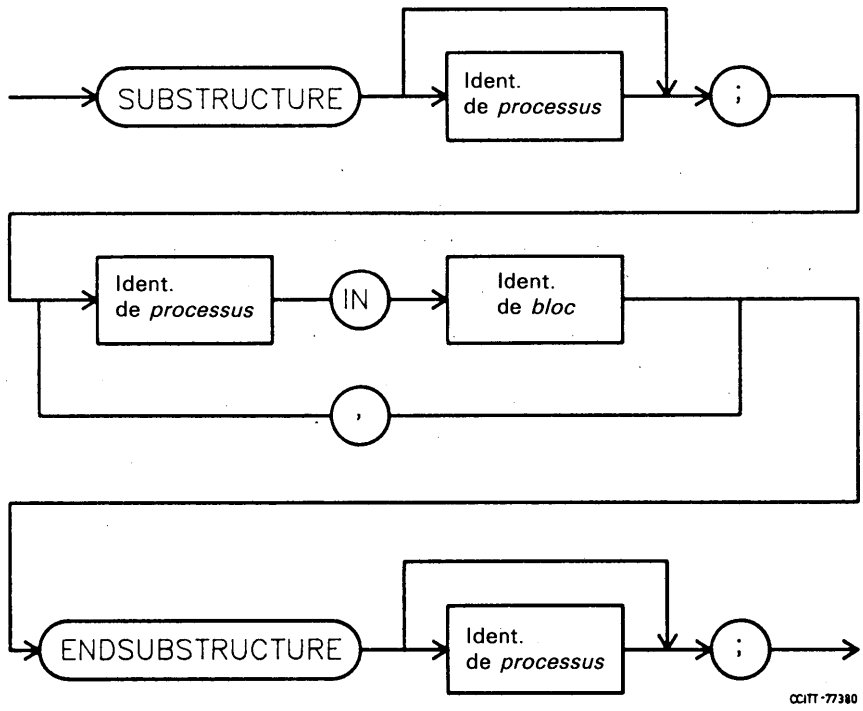
Cette syntaxe représente la *subdivision* d'un *processus* en *sous-processus* et l'allocation de ceux-ci à des *sous-blocs.* La *subdivision* peut être représentée dans la *définition de processus* et la *définition de sous-structure de bloc.*

4.3.1 *Syntaxe*

La DÉFINITION DE PROCESSUS est étendue comme suit:



La DÉFINITION DE SOUS-STRUCTURE DE PROCESSUS est la suivante:



L'exemple ci-dessous représente la *sous-structure de* processus dans une *définition de processus*:

Exemple:

```
...
    SUBSTRUCTURE p1;
        p2 IN b2;
        p3 IN b1;
    ENDSUBSTRUCTURE;
...
ENDPROCESS p1:
```

L'exemple suivant représente la *sous-structure de processus* dans une *définition de sous-structure de bloc*:

Exemple:

```
...
    SUBSTRUCTURE b;
        SUBSTRUCTURE p;
            p1 IN b2;
            p2 IN b1;
        ENDSUBSTRUCTURE
    ...
    ENDSUBSTRUCTURE;
ENDBLOCK b;
```

4.3.2 *Relation avec la syntaxe abstraite LDS*

La construction de cette syntaxe représente la *définition de sous-structure de processus* dans la syntaxe abstraite. Le *nom de processus* associé est le *nom* de la clause de départ et l'ensemble de *noms de processus* qui y est contenu, chacun étant associé à un *nom de bloc*, consiste en les *noms* séparés par le mot clé «IN».

4.4 *Sous-structure de canal*

Cette syntaxe représente la *subdivision* d'un *canal* en un ensemble de *canaux* et de *blocs*.

4.4.1 *Syntaxe*

La syntaxe de *canal* est étendue de manière à inclure, à titre optionnel, le non-terminal «*sous-structure de canal*».

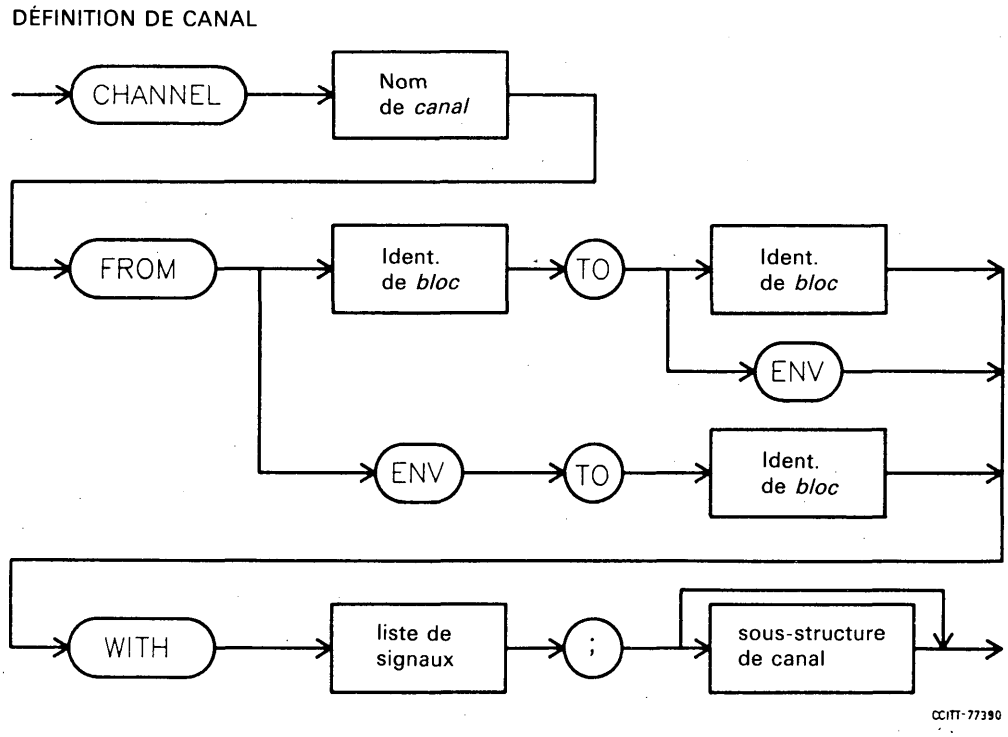


FIGURE 19/Z.102 (feuillet 1 sur 4)

Diagrammes de syntaxe pour sous-structure de canal

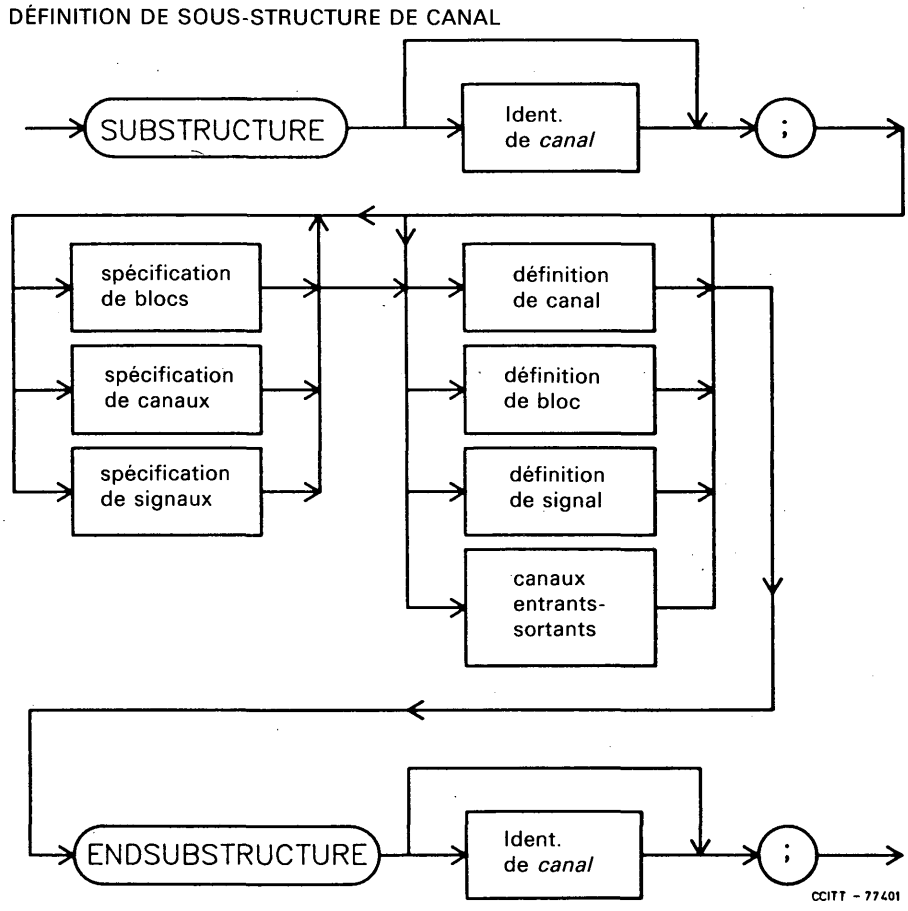


FIGURE 19/Z.102 (feuillet 2 sur 4)

Diagrammes de syntaxe pour sous-structure de canal

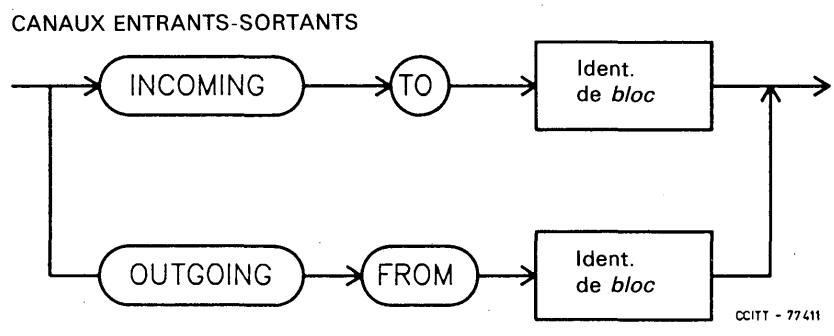


FIGURE 19/Z.102 (feuillet 3 sur 4)

Diagrammes de syntaxe pour sous-structure de canal

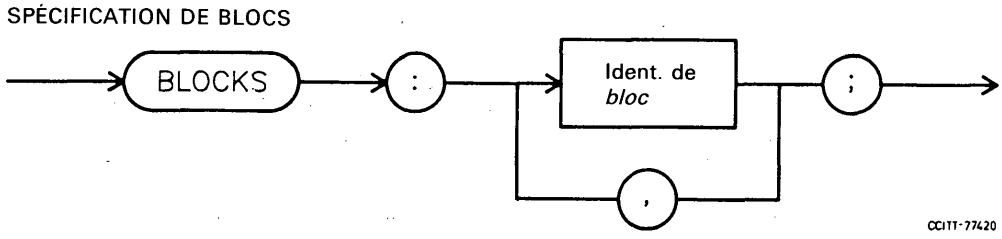


FIGURE 19/Z.102 (feuillet 4 sur 4)

Diagrammes de syntaxe pour sous-structure de canal

La *sous-structure de canal* est un ensemble de références aux composants et aux définitions additionnelles de la sous-structure.

Exemple:

```
CHANNEL c FROM b TO d;
    SUBSTRUCTURE
        INCOMING TO e;
        OUTGOING FROM f;
        BLOCKS: e,f;
        CHANNEL c1 FROM e TO f WITH s1,s2,s3;
        BLOCK e;
                    ...
        ENDBLOCK e;
        BLOCK f;
                    ...
        ENDBLOCK f;
    ENDSUBSTRUCTURE;
```

4.4.2 *Relation avec la syntaxe abstraite LDS*

Cette syntaxe représente la *définition de sous-structure de canal* dans la syntaxe abstraite. Les *identificateurs de bloc, de canal, de signal et de données* donnés dans la syntaxe sont des références aux définitions contenues.

Recommandation Z.103

EXTENSIONS FONCTIONNELLES DU LDS

1 Introduction

La présente Recommandation définit un certain nombre de concepts et abréviations supplémentaires du LDS. Ces adjonctions sont fondées sur le LDS de base, tel que le définit la Recommandation Z.101. Leur but est de fournir aux utilisateurs du LDS des concepts et des abréviations qui leur conviennent.

Les concepts et abréviations ci-dessous sont définis dans la présente Recommandation:

Procédures

Elles procurent un moyen de représenter une portion d'un *graphe de processus* par un seul élément, auquel on peut se référer plusieurs fois. Le *comportement* détaillé de la *procédure* est défini ailleurs, à l'intérieur ou à l'extérieur du *graphe de processus*. On peut utiliser les *procédures* afin de favoriser l'emploi, dans le LDS, de méthodes de conception structurée, cela en permettant de décomposer un *graphe de processus* en une hiérarchie de sections. Ce concept est semblable au concept de procédure qui figure normalement dans les langages de programmation.

Importation et exportation de valeurs

Abréviation pour l'interfonctionnement de *signaux* entre des *instances de processus* lorsque celles-ci doivent partager une *valeur* d'une variable appartenant à l'un des *processus.*

Condition de validation

Abréviation destinée à éviter l'«explosion d'état» lorsque la réception ou la mise en réserve d'un ensemble de *signaux* sont conditionnelles.

Signal continu

Abréviation destinée à représenter l'interfonctionnement de *signaux*, lorsque l'on observe une condition continue, extérieure au *processus.*

Macro

Méthode syntaxique destinée à permettre à l'utilisateur de définir une notation sténographique. Une *macro* est la définition d'un élément de syntaxe composite selon les termes d'autres éléments de syntaxe déjà définis. Une *macro* n'a pas de sémantique propre.

Option

Facilité syntaxique pour représenter dans un diagramme, ou texte linéaire, plusieurs fonctionnements possibles. On doit décider, avant l'interprétation du diagramme ou du texte linéaire, quelle est la variante proposée par une *option* qu'il convient de choisir.

La présente Recommandation définit aussi une extension de la syntaxe graphique du LDS, à savoir l'utilisation d'ÉLÉMENTS GRAPHIQUES dans des symboles d'*état.*

2 Procédures

Une *procédure* est un moyen de donner un nom à un assemblage d'objets et de le représenter par une référence unique. Les règles relatives aux *procédures* imposent une discipline sur la manière dont l'assemblage d'objets est choisi et elles limitent la portée du nom des *éléments de données* et des *signaux.*

Les procédures ont pour but:

a) de permettre de structurer un *graphe de processus* en plusieurs niveaux de détail;

b) de restreindre le volume des spécifications en autorisant un assemblage complexe d'objets que l'on peut considérer isolément de manière à le représenter comme un objet unique;

c) de permettre que des assemblages d'objets d'usage courant soient prédéfinis et utilisés de manière répétée.

Les *procédures* sont définies au moyen de *définitions de procédure*. On fait appel à une procédure au moyen d'un *appel de procédure* faisant référence à la *définition de procédure.* Des paramètres sont associés à un *appel de procédure*: on les emploie à la fois pour fournir des *valeurs* et pour limiter la portée des *données* et des *signaux* à employer pour l'exécution de la *procédure.* C'est du mécanisme de transfert des paramètres que dépendent les *signaux* et les éléments de données affectés par l'interprétation d'une *procédure.*

2.1 *Modèle commun de langage*

Les définitions ci-après doivent être considérées uniquement comme des additions à ce qui est défini dans la Recommandation Z.101.

2.1.1 *Introduction au modèle commun*

Une *procédure* est une section d'un *graphe de processus* que l'on peut considérer isolément. Elle a un point d'entrée unique et un point de sortie unique. Un *appel de procédure* peut apparaître partout où peut apparaître une *tâche* dans un *graphe de processus* ou dans un *graphe de procédure.* Une *procédure* peut contenir des *états.* Tous les *éléments de données* utilisés dans une *procédure* doivent être définis dans le cadre de la *définition de procédure.* Si un *élément de données* est défini comme un *paramètre formel,* il utilise des *valeurs d'éléments de données* dans l'environnement appelant et/ou donne des *valeurs* à des *éléments de données* dans cet environnement; sinon l'objet est un objet local de la *procédure* et il doit être défini dans la *définition de procédure.*

Une *procédure* est interprétée seulement lorsqu'une *instance de processus* l'appelle, et elle est interprétée par cette *instance de processus.* Quand un *appel de procédure* est interprété, le *graphe de procédure* doit être interprété avant que continue l'interprétation de la *transition* dans laquelle apparaît l'*appel.* Cela signifie que, tandis que l'on interprète la *procédure,* on utilise l'*accès d'entrée de l'instance de processus* appelant ainsi que l'ensemble *des signaux d'entrée valides.* Tous les *signaux* référencés dans une *procédure* doivent être *nommés* comme des *paramètres formels* de la *procédure.* Tous les *signaux* de l'ensemble des noms de *signaux d'entrée valides* de l'*environnement* appelant qui ne sont pas référencés dans la *procédure* doivent aussi être *nommés* comme des *paramètres* de la *procédure,* c'est-à-dire comme l'ensemble de mise en réserve supplémentaire. Cela permet à la

procédure de *mettre en réserve* automatiquement tous les autres *signaux* susceptibles d'arriver à l'*accès d'entrée*. Si cela n'était pas fait, l'introduction dans une *transition* d'un *appel* à une *procédure* contenant un *état* aurait masqué les effets de bord de perte des *signaux* (par des «transitions nulles implicites») qui seraient arrivés pendant qu'une *instance de processus* interprétait une *procédure*.

Une *définition de procédure* est donnée par un *graphe de procédure* et peut apparaître partout où peut apparaître une *définition de type de données*.

Un *graphe de procédure* suit les mêmes règles qu'un *graphe de processus*, sous réserve des exceptions suivantes:

- un *nœud de départ* est remplacé par un *nœud de départ de procédure*;
- un *nœud d'arrêt* est remplacé par un *nœud de retour*;
- seuls les *éléments de données*, les *paramètres formels* et les *noms de signal* qui sont déclarés dans la *procédure* sont visibles de l'*instance de processus* pendant que l'on interprète la *procédure*; en d'autres termes, une *procédure* ne peut renvoyer à aucun *élément de donnée* ni à aucun *signal* qui est en dehors d'elle à moins qu'il ne soit déclaré comme *paramètre formel*, (les *signaux* appartiennent nécessairement à l'*environnement d'appel* de la *procédure*, et tous ceux qui sont d'*entrée*, de *réserve* ou de *sortie* dans cette *procédure* doivent être *déclarés*);
- un même *paramètre effectif de nom de signal* ne peut pas être mis en correspondance avec plus d'un *paramètre formel de nom de signal*;
- dans la *syntaxe abstraite*, l'ensemble des *signaux* à *mettre en réserve* est fourni sous forme de *paramètres* de l'*appel de procédure*. En revanche, dans la syntaxe concrète, tous les *nœuds d'état* contenus dans une *procédure* ont un *ensemble implicite de signaux de réserve*, lequel contient tous les *signaux d'entrée* non déclarés comme *paramètres formels* dans la *procédure*.

2.1.2 *Syntaxe abstraite*

A la *syntaxe abstraite* définie dans la Recommandation Z.101, il est ajouté ce qui suit:

Définition de système

Une *définition de système* peut aussi contenir une ou plusieurs *définitions de procédure*.

Définition de bloc

Une *définition de bloc* peut aussi contenir une ou plusieurs *définitions de procédure*.

Définition de processus

Une *définition de processus* peut aussi contenir une ou plusieurs *définitions de procédure*.

Graphe de processus

Un *graphe de processus* peut aussi contenir un *nœud d'appel de procédure*.

Une *chaîne de transition* peut aussi être un *nœud d'appel de procédure*, suivi d'une *chaîne de transition*.

Un *nœud d'appel de procédure* contient un *identificateur de procédure* et une *liste de paramètres effectifs*.

Définition de procédure

La *définition de procédure* est associée à un *nom de procédure* et contient une *liste de paramètres formels*, un *ensemble de réserve* supplémentaire et un *graphe de procédure*; elle peut contenir des *définitions de procédure* et des *définitions de données*.

La *liste des paramètres formels* peut être vide. Chaque *paramètre formel* de cette liste doit recevoir l'un des attributs *IN, IN/OUT ou SIGNAL*.

Les *noms* des *paramètres formels* possédant l'attribut *IN ou IN/OUT* ne peuvent pas être utilisés dans les *définitions de variables* contenues dans la *définition de procédure*; l'ensemble des *noms de paramètres formels* possédant l'attribut *SIGNAL* doit contenir tous les *noms de signaux* référencés dans le *graphe de procédure*.

Graphe de procédure

Un *graphe de procédure* est un graphe dont les *nœuds* sont reliés entre eux par des *arcs* dirigés. Un *arc* qui entre dans un *nœud* est appelé *arc* d'arrivée et un *arc* qui part d'un *nœud* est appelé *arc* de départ. Un *nœud* pour lequel un *arc* est *arc* d'arrivée suit le *nœud* ayant le même *arc* comme *arc* de départ.

Les types de *nœud* suivants sont autorisés dans un *graphe de procédure*:

Nœud d'état
Nœud d'entrée
Nœud de tâche
Nœud de sortie

Nœud de décision
Nœud d'appel de procédure
Nœud de départ de procédure
Nœud de retour
Nœud de demande de création

Chaque *nœud* du graphe a un *nom* et un *type*. Les *nœuds d'état* contenus dans un *graphe de procédure* portent des *noms* différents.

Les règles suivantes définissent la connectivité d'un *graphe de procédure*:

- Chaque *graphe de procédure* contient un *nœud de départ de procédure* et un seul. Le *nœud de départ de procédure* est suivi d'une *chaîne de transition*. Le *nœud de départ* ne suit aucun autre *nœud*.
- Une chaîne de *transition* peut être:
  a) une *chaîne de transition* nulle suivie d'un *nœud d'état* ou d'un *nœud de retour*,
  b) une chaîne d'action suivie d'une *chaîne de transition*,
  c) un *nœud de décision*.
- Une chaîne d'*action* peut être:
  a) un *nœud de tâche*,
  b) un *nœud de sortie*,
  c) un *nœud de demande de création*.
- Un *nœud de décision* est suivi d'au moins deux *arcs de décision*.
- Un *arc de décision* est un *arc nommé* suivi d'une *chaîne de transition*.
- Un *nœud d'état* est suivi d'un ou de plusieurs *nœuds d'entrée*.
- Un *nœud d'entrée* est suivi d'une chaîne de *transition*.
- Le *nœud de retour* n'a aucun *nœud* à sa suite.
- Chaque graphe a au moins un *nœud de retour*.
- Chaque *nœud* est atteignable à partir du *nœud de départ de procédure*.

Un *nœud d'appel* contient un *identificateur de procédure*, une *liste de mise en réserve supplémentaire* et une *liste de paramètres effectifs*. Dans cette liste, doit se trouver au moins un *paramètre effectif* pour chacun des *paramètres formels* de la *définition de procédure* citée. Chaque *paramètre effectif* doit correspondre au *type* du *paramètre formel* correspondant. Aucun *nom de paramètre effectif de signal* ne peut apparaître plus d'une fois dans la *liste de paramètres effectifs*.

Tout *paramètre effectif* dont le *paramètre formel* correspondant a l'attribut *IN/OUT* doit être un *nom de variable*.

L'*ensemble de mise en réserve supplémentaire* représente éventuellement un ensemble vide *d'identificateurs de signaux*.

2.1.3 *Interprétation*

Aux règles d'interprétation données dans la Recommandation Z.101, il est ajouté ce qui suit:

Processus

Un *nœud d'appel* est interprété comme l'interprétation du *nœud de départ de procédure* de la *définition de procédure* désignée par le *nom* du *nœud*; ensuite, le *nœud* qui suit le *nœud d'appel* est interprété.

L'*ensemble de mise en réserve supplémentaire* reçoit la *valeur* de tous les *identificateurs de signaux* de l'*ensemble des signaux d'entrée valables* de la *procédure* appelante, qui n'apparaît pas comme les *paramètres effectifs* du *nœud*.

Procédure

Lorsqu'un *processus* interprète un *nœud d'appel* qui désigne la *définition de procédure* contenant le *graphe de procédure*, le *graphe de procédure* est interprété. Les *nœuds* du *graphe de procédure* sont interprétés de la même manière que les *nœuds* équivalents du *graphe de processus*, sous réserve des exceptions et adjonctions suivantes:

- Chaque *paramètre formel* possédant l'attribut *IN* dénote une *variable* typée. Cette *variable* est locale pour la *procédure*; elle est créée lorsque le *nœud de départ de la procédure* est interprété et elle cesse d'exister lorsque le *nœud de retour* est interprété.
- Chaque *paramètre formel* possédant l'attribut *IN/OUT* dénote un *synonyme* de la *variable* donnée comme *paramètre effectif*. Ce nom de *synonyme* est utilisé d'un bout à l'autre de l'interprétation du *graphe de procédure* lorsqu'on se réfère à la *valeur* de la *variable* ou qu'on lui affecte une nouvelle *valeur*.

Nœud de départ de procédure

L'interprétation d'un *nœud de départ de procédure* est la suivante:

- Une *variable* locale est créée pour tout *paramètre formel* possédant l'attribut *IN* et qui a le *nom* et le *type de données* du *paramètre formel*. A cette *variable* est assignée la *valeur* du *paramètre effectif*, qui peut ne pas être définie.
- Chaque *nom de variable* donné dans les *paramètres formels* possédant l'attribut *IN/OUT* est utilisé comme *nom synonyme* pour la *variable* donnée comme *paramètre effectif*. La *variable* représentée par le *nom synonyme* est évaluée une fois seulement au *nœud de départ de procédure*, et non à chaque utilisation du *paramètre formel* dans la *procédure*.
- Chaque *nom de signal* donné dans les *paramètres formels* possédant l'attribut *SIGNAL* est utilisé comme *nom synonyme* de *l'identificateur de signal* donné comme *paramètre effectif*.
- Le *nœud* qui suit le *nœud de départ de procédure* est interprété.

Nœud d'état

Le *nœud d'état* est interprété de la même manière que dans un *graphe de processus*, exception faite que l'*ensemble des signaux de réserve* présenté à l'*accès d'entrée* est l'union de l'*ensemble de réserve supplémentaire* et de l'*ensemble des signaux de réserve* du processus ou de la procédure dite procédure en cours.

2.2 *LDS/GR*

L'adjonction du concept de *procédure* dans le LDS conduit à ajouter un nouveau symbole dans le *diagramme de processus*: il s'agit du symbole d'*appel*.

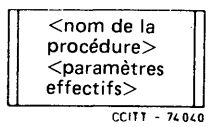
La *définition de la procédure* dans la syntaxe graphique est représentée par un *diagramme de procédure*, lequel est semblable au diagramme de processus, sauf pour ce qui est du symbole de *départ de procédure* et du *symbole de retour*.

Dans les paragraphes qui suivent, seules sont données les adjonctions requises par l'introduction d'une *procédure* à la syntaxe graphique telle que définie dans la Recommandation Z.101.

2.2.1 *Diagramme de processus*

2.2.1.1 *Symboles*

Le symbole suivant sert à représenter l'*appel de procédure*:



Le symbole peut apparaître dans un *diagramme de processus* chaque fois qu'un symbole de *tâche* est autorisé.

Le *nom de procédure* et les *paramètres effectifs* sont donnés à l'aide de la syntaxe LDS/PR.

FIGURE 1/Z.103

Le symbole d'appel de procédure

2.2.1.2 *Relations avec la syntaxe abstraite du LDS*

Le *symbole d'appel* de procédure représente le *nœud d'appel de procédure* dans un *graphe de processus*.

Les *paramètres effectifs* donnés dans les symboles représentent les *paramètres effectifs* attachés au *nœud d'appel de procédure*. Tout *signal d'entrée* valide qui n'est pas donné comme *paramètre effectif* est membre de l'*ensemble de mises en réserve additionnelles* pour le *nœud d'appel de procédure*, indiquant que ce *signal* doit être mis en réserve pendant l'exécution de la *procédure*.

Nœud de retour

L'interprétation du *nœud de retour* est la suivante:

- toutes les *variables* créées par l'interprétation du *nœud de départ de procédure* vont cesser d'exister;
- tous les *noms synonymes* établis par l'interprétation du *nœud de départ de procédure* vont cesser d'exister;
- l'interprétation du *nœud de retour* complète celle du *nœud de départ de procédure.*

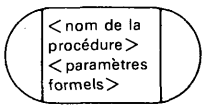
2.2.2 *Diagramme de procédure*

Un *diagramme de procédure* est semblable à un *diagramme de processus*, les seules différences étant que le *symbole de départ* est remplacé par *le symbole de départ de procédure* et que le *symbole d'arrêt* est remplacé par *le symbole de retour.*

Dans ce qui suit, seule la syntaxe supplémentaire est définie.

2.2.2.1 *Symboles*

Dans la figure 2/Z.103 sont définis les deux symboles supplémentaires suivants:



Symbole de départ de procédure
Le *nom de la procédure* et les *paramètres formels* sont donnés par utilisation de la syntaxe *LDS/PR*

Symbole de retour

CCITT-73900

FIGURE 2/Z.103

Symboles supplémentaires pour diagramme de procédure

2.2.2.2 *Relations avec la syntaxe abstraite du LDS*

Le *diagramme de procédure* représente la *définition de procédure* dans la *syntaxe abstraite* du LDS.

Les symboles que l'on trouve aussi dans le *diagramme de processus* ont les mêmes relations avec la *syntaxe abstraite* que lorsqu'ils apparaissent dans un *diagramme de processus.* Le symbole supplémentaire de *départ de procédure* et celui de *retour de procédure* représentent respectivement le *nœud de départ de procédure* et le *nœud de retour.*

Les *paramètres* attachés au *symbole de départ de procédure* représentent les *paramètres formels* attachés au *nœud de départ de procédure.*

2.2.2.3 *Conventions graphiques*

De façon générale, les mêmes conventions graphiques que pour *le diagramme de processus* s'appliquent. Les conventions pour le *symbole de départ de procédure* et pour celui de *retour* sont respectivement les mêmes que pour le *symbole de départ de processus* et pour celui d'*arrêt.*

Chaque fois qu'apparaît un *symbole de retour*, il représente le *nœud de retour* (apparition multiple d'un *symbole de retour*).

2.3 *LDS/PR*

La *syntaxe LDS/PR* applicable à une *définition de procédure* est donnée ci-dessous. La *définition de procédure* peut apparaître chaque fois qu'une *définition de type de données* peut apparaître dans la syntaxe. La syntaxe est semblable à celle de la *définition de processus.*

On trouvera ci-dessous les adjonctions à la syntaxe déjà définie dans la Recommandation Z.101.

2.3.1 *Système*

2.3.1.1 *Syntaxe*

La syntaxe de la *définition de procédure* s'ajoute à celle du *système.*

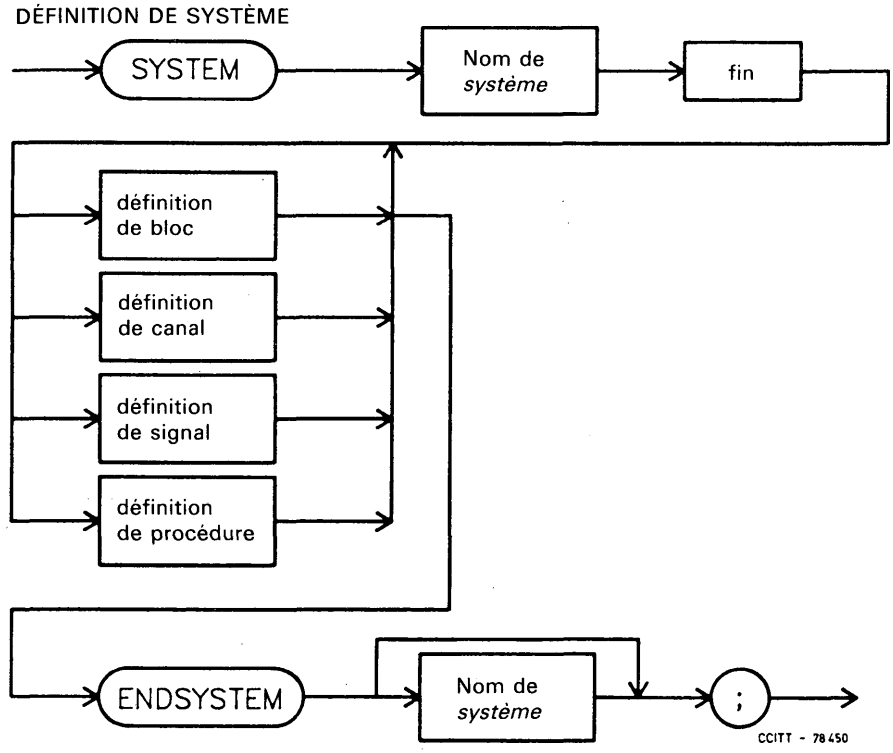


FIGURE 3/Z.103

Diagrammes de syntaxe pour système

2.3.2 *Bloc*

2.3.2.1 *Syntaxe*

La syntaxe de *définition de procédure* s'ajoute à celle du *bloc.*

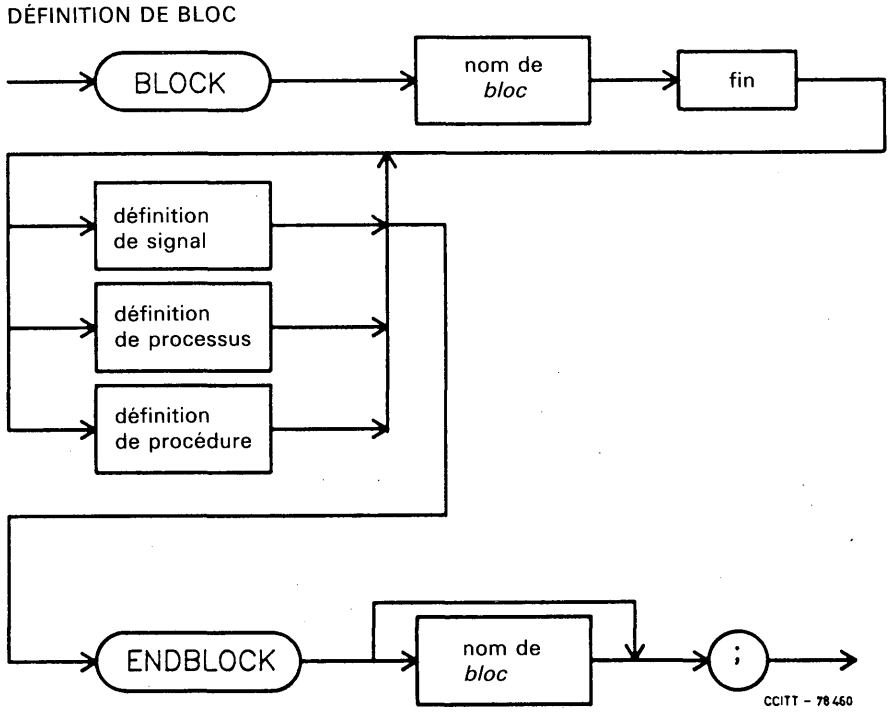


FIGURE 4/Z.103

Diagramme de syntaxe pour bloc

2.3.3 *Processus*

2.3.3.1 *Syntaxe*

La syntaxe de *définition de procédure* s'ajoute à celle du *processus* et celle d'*appel de procédure* s'ajoute à celle d'une *chaîne de transition*:

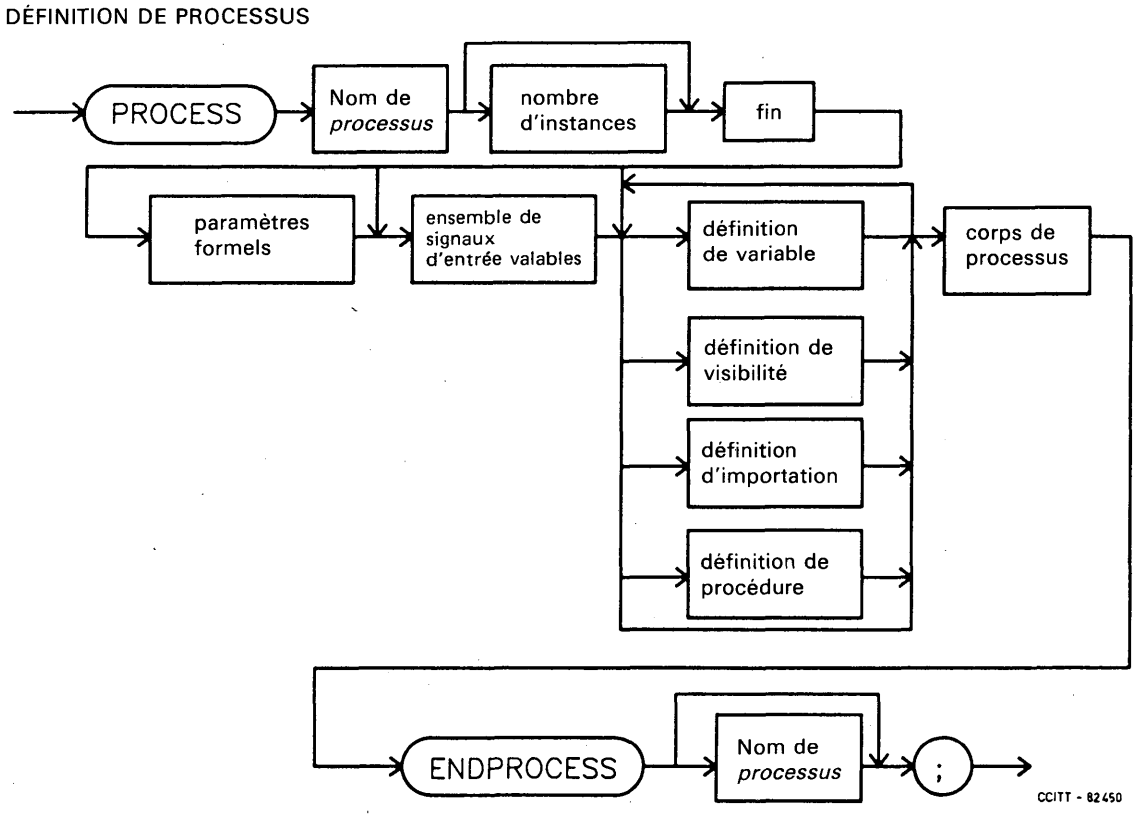


FIGURE 5/Z.103

Diagramme de syntaxe pour processus

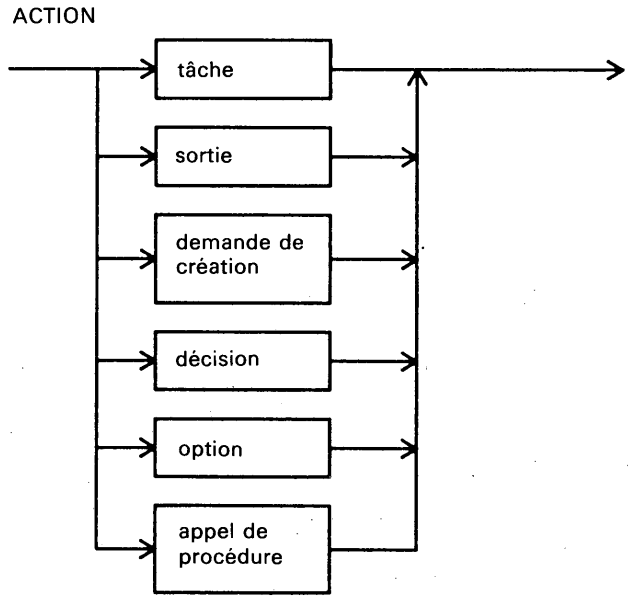


FIGURE 6/Z.103 (feuillet 1 sur 2)

Diagramme de syntaxe pour action

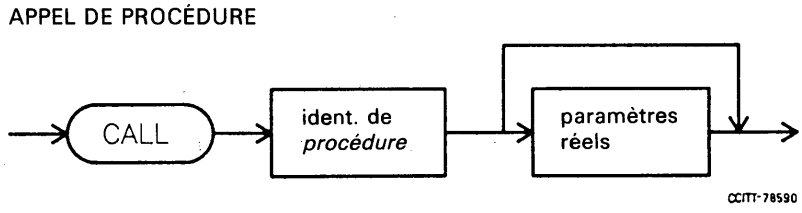


FIGURE 6/Z.103 (feuillet 2 sur 2)

Diagramme de syntaxe pour appel de procédure

2.3.3.2 *Relations avec la syntaxe abstraite du LDS*

L'*appel de procédure*, dans une *transition*, représente le *nœud d'appel* de la *syntaxe abstraite*, et les *paramètres effectifs* représentent les *paramètres effectifs* attachés au *nœud d'appel.*

Chaque *paramètre effectif* donné doit être conforme au *type de données* du *paramètre formel* correspondant de la *définition de procédure* référencée.

2.3.4 *Procédure*

2.3.4.1 *Syntaxe*

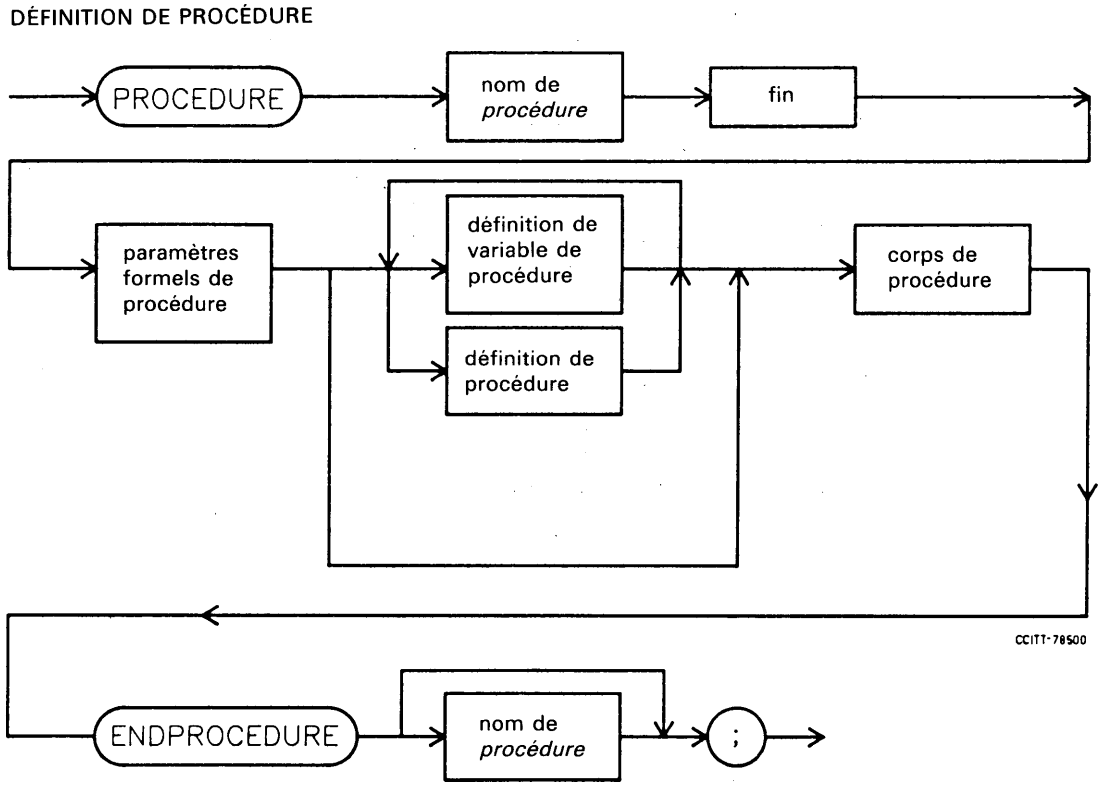


FIGURE 7/Z.103 (feuillet 1 sur 5)

Diagramme de syntaxe pour procédure

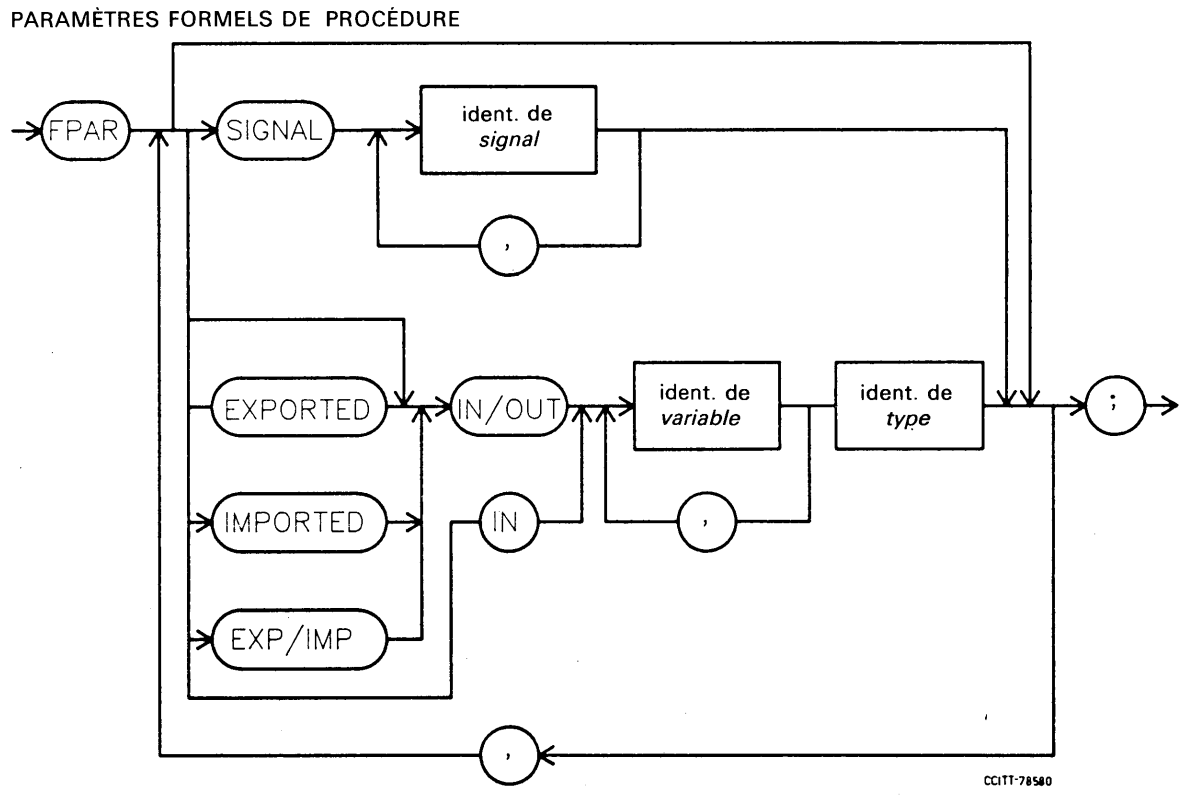


FIGURE 7/Z.103 (feuillet 2 sur 5)

Diagramme de syntaxe pour procédure

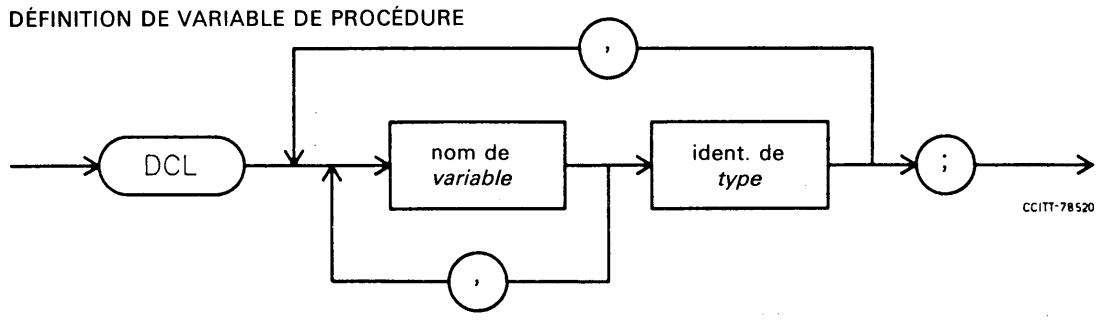


FIGURE 7/Z.103 (feuillet 3 sur 5)

Diagramme de syntaxe pour procédure

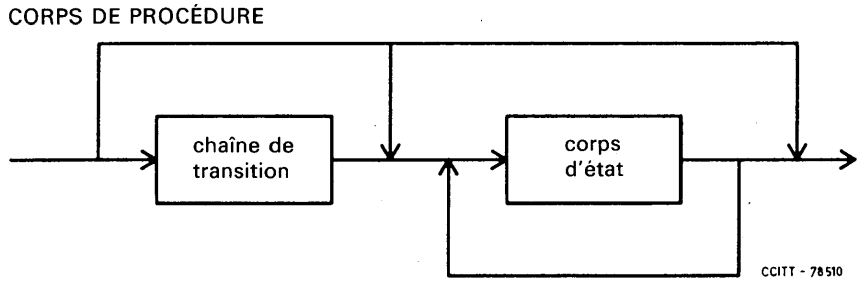


FIGURE 7/Z.103 (feuillet 4 sur 5)

Diagramme de syntaxe pour procédure

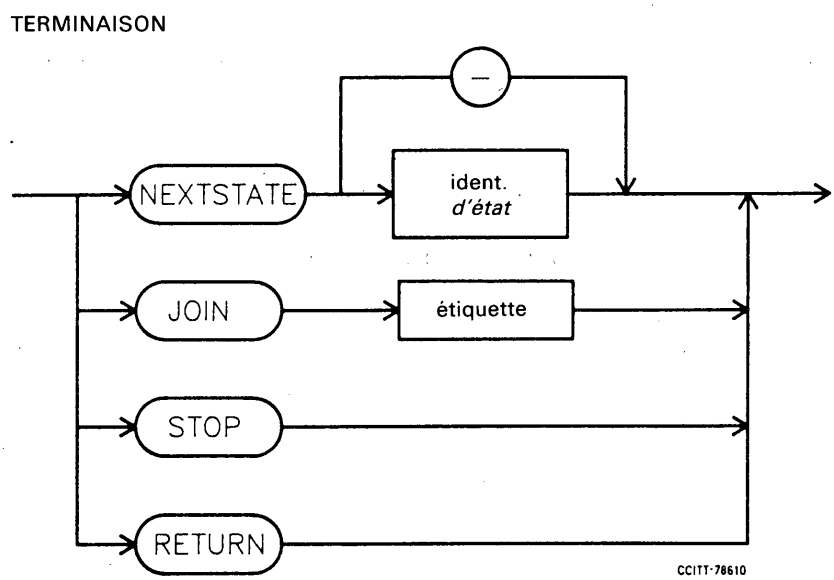


FIGURE 7/Z.103 (feuillet 5 sur 5)

Diagramme de syntaxe pour procédure

2.3.4.2 *Relations avec la syntaxe abstraite du LDS*

Le diagramme de syntaxe de *procédure* représente la *définition de procédure* de la *syntaxe abstraite*. Les *paramètres formels* inclus dans le diagramme de syntaxe représentent les *paramètres formels* attachés au *nœud de départ* de procédure.

Les constructions syntaxiques autorisées à l'intérieur d'une *procédure* et qui sont communes avec celles autorisées dans un *processus* ont les mêmes relations avec la *syntaxe abstraite* que lorsqu'elles apparaissent dans un *processus*. Le *nœud de départ de procédure* est implicite et il est suivi de la première construction *syntaxique* de la *procédure*; les instructions *de retour* représentent le *nœud de retour*.

3 Opérations composites pour communication entre PROCESSUS LDS

Les *opérations composites* sont des abréviations normalisées pour des assemblages de concepts LDS. Elles sont définies à l'aide de concepts qui existent déjà dans le LDS et n'ajoutent rien à la sémantique de ce langage. Elles ne changent donc pas la *syntaxe abstraite* pour ce qui est de son interprétation. Elles sont introduites pour la commodité des utilisateurs du LDS.

Les *opérations composites* définies ici fournissent quelques méthodes variées de communication entre des *instances de processus* du LDS. Bien qu'elles paraissent très différentes de celles du mécanisme normal d'échange des *signaux*, pour l'utilisateur du LDS, elles sont en fait fondées sur la sémantique normale du LDS.

Les opérations composites permettent:

a) le partage de *valeurs* d'*éléments de données* entre des *processus*, même s'ils sont attribués dans des *blocs* différents;

b) la mise temporaire en service ou hors service de la réception de *signaux* particuliers sans que l'on ait à montrer cela avec exactitude au moyen d'*états* séparés;

c) des *signaux continus*, pour lesquels les *transitions* sont dues à un changement d'une *valeur* du *signal*.

A chaque *opération composite*, on donne une syntaxe qui lui est propre; elle est définie à l'aide des concepts LDS «normaux». Les *opérations composites* entraînent donc pour l'utilisateur des *états*, des *signaux* et des *procédures* implicites ou cachés. Elles sont définies aux § 3.1 à 3.3 ci-après, au moyen de noms qui illustrent les symboles LDS normaux. Lorsqu'une *opération composite* quelconque est utilisée dans une représentation du LDS, les symboles LDS qui interviennent ont des noms implicites uniques. Lesdits noms sont choisis de telle manière qu'il n'y ait pas de conflit entre les diverses apparitions de l'*opération composite* ou avec d'autres noms de la représentation LDS. L'utilisateur du LDS est ainsi libre d'employer ces noms à d'autres fins s'il le désire.

Les *opérations composites* étant définies au moyen de termes utilisés ailleurs dans le LDS, leur emploi peut dans certaines situations donner lieu à des effets de bord, lesquels sont examinés dans les Directives pour les usagers du LDS.

3.1 *Valeurs importées et exportées*

Dans le LDS, une *variable* appartient toujours à une *instance de processus*, dont elle est une variable locale. Normalement, elle n'est visible que de l'*instance de processus* à laquelle elle appartient; elle peut cependant être *déclarée* comme *valeur partagée* (voir la Recommandation Z.101), ce qui permet à d'autres *instances de processus* du même *bloc* d'en *visualiser* la *valeur*. Si une *instance de processus* d'un autre *bloc* doit *voir* une *variable*, on a besoin pour cela d'un échange de *signaux* avec l'*instance de processus* à laquelle cette *variable* appartient.

Dans ce qui suit, on trouve la description d'une méthode normalisée et d'une abréviation à cet effet. On parlera de *valeur(s) importée(s)* puisque la communication se fait par des copies de la *valeur* de la *variable*, seule l'*instance de processus* à laquelle elle appartient ayant accès à la *variable* elle-même. On peut aussi utiliser cette technique pour exporter *des valeurs* vers d'autres *instances de processus* du même *bloc*, auquel cas elle constitue une variante de l'utilisation de *valeurs partagées*. Cette autre façon de procéder est nécessaire, soit quand il est probable que le *bloc* sera décomposé de telle manière que les *instances de processus* concernées se trouvent dans des *sous-blocs* différents, soit quand les *valeurs* sont utilisées dans des *conditions de validation* (voir le § 3.3).

L'*instance de processus* à laquelle appartient une *variable* dont les *valeurs* sont mises à la disposition d'autres *instances de processus* est appelée l'*exportateur* de la *variable*. Les autres *instances de processus* qui utilisent cette *variable* en sont les *importateurs*.

L'accès à la *valeur* de la *variable* se fait par un échange de *signaux*. L'*importateur* envoie un *signal* à l'*exportateur* et attend la réponse. En réponse au *signal*, l'*exportateur* envoie à l'*importateur* un *signal* de retour accompagné de la *valeur* de la *variable* au moment de la dernière *opération d'exportation*.

Les *variables* dont les *valeurs* sont *importées* et *exportées* sont définies comme telles dans leurs *définitions de variable*. Elles sont aussi identifiées par des *définitions* dans les *canaux* empruntés pour l'échange implicite des *signaux*.

La définition d'une *variable* comme *exportée* produit une définition implicite d'une copie de cette variable à utiliser dans les opérations d'*import* et d'*export*.

L'*instance de processus* révèle la *valeur* d'une *variable* définie comme *exportée*, au moyen de l'énoncé:

EXPORT(x), x étant le *nom* de la *variable*.

L'*EXPORT*(x) entraîne le stockage de la *valeur* actuelle de x dans la copie implicite et l'envoi de *signaux* aux *instances de processus* dans l'attente d'une *condition de validation* (§ 3.2) ou de *signaux continus* (§ 3.3).

L'opération d'EXPORTATION peut être faite conjointement avec la manipulation de la variable ou indépendamment d'elle, soit:

EXPORT (x):= <expression>;

ou

EXPORT (x).

Cette construction ne peut apparaître que dans une *tâche.*

L'accès à partir d'une autre *instance* ne peut se faire qu'au moyen de la construction syntaxique:

IMPORT(x, pid), x étant le *nom* de la *variable* et pid une référence à l'*instance de processus* à laquelle elle appartient.

Cette construction peut apparaître dans une *tâche* ainsi que dans une *décision.*

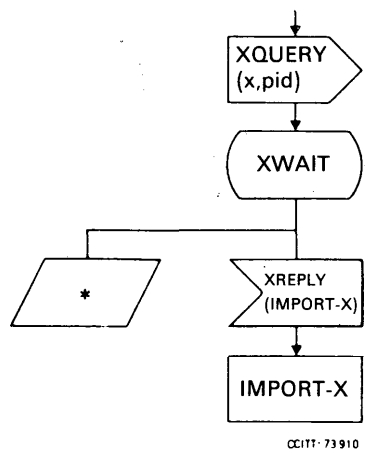
3.1.1 *Définition*

3.1.1.1 *Opération d'importation par l'importateur*

Dans le *graphe de processus* de l'*importateur*, cette opération est modélisée par la séquence de *nœuds* suivante:

- un *nœud de sortie*, appelé XQUERY, comportant une référence du *nom* de *variable* pour laquelle on veut avoir accès à la *valeur*, adressée à l'*instance de processus* à laquelle appartient la *variable* (c'est-à-dire à l'*exportateur*);
- un *nœud d'état*, avec un *ensemble de signaux de réserve* comportant tous les *signaux*, à l'exception de XREPLY;
- un *nœud d'entrée* pour le *signal* XREPLY, qui renvoie la *valeur* de la valeur de la *variable* demandée;
- une *variable* implicite du même *type* que la *variable exportée* reçoit la *valeur* associée au *signal* XREPLY. Le *nom de la variable* implicite remplace l'opération d'*importation.* Un nom de variable implicite est utilisé pour chacune des opérations d'importation.

La séquence des opérations qui définit l'opération d'*importation* est représentée ci-dessous au moyen de la syntaxe graphique du LDS. L'instruction IMPORT(x, pid) correspond à la *chaîne de transition* suivante:



Remarque – Ce diagramme n'est représenté que pour explication. L'utilisateur de l'opération d'*importation* se bornerait à écrire: IMPORT(x, pid).

FIGURE 8/Z.103

Opération d'importation par l'importateur expliquée en LDS/GR

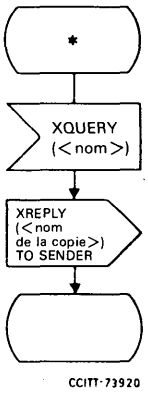
3.1.1.2 *Opération d'importation par l'exportateur*

L'opération d'*importation* impliquant des actions exécutées par l'*exportateur*, la *transition* implicite suivante est ajoutée à chaque *état* de l'*exportateur* (y compris tous ses états implicites):

- un *nœud d'entrée* pour le *signal* XQUERY;
- un *nœud de sortie*, appelé XREPLY, qui renvoie la copie implicite de *valeur* de la *variable* demandée à l'*instance de processus* qui l'a demandée;
- un retour à l'*état* dont est partie la *transition.*

(Il convient de noter que l'*exportateur* est aussi modifié, de même que d'autres *transitions*, si les *valeurs exportées* sont utilisées dans des *conditions de validation* et/ou dans des *signaux continus;* voir les § 3.2 et 3.3.)

On trouvera dans la figure 9/Z.103, la *transition* implicite représentée au moyen de la syntaxe graphique du LDS.



Remarque – Ce diagramme n'est représenté que pour explication. La *transition* étant implicite, l'utilisateur n'a rien à écrire.

FIGURE 9/Z.103

Opération d'importation par l'exportateur expliquée en LDS/GR

3.1.1.3 *Opération d'exportation*

Cette opération est le moyen dont dispose celui à qui appartient une *variable exportée* pour révéler aux importateurs la *valeur* actuelle de la *variable.* A la suite de l'opération d'*exportation*, la copie implicite de la *variable* reçoit la *valeur* actuelle de la *variable.* En l'absence de *conditions de validation* ou de *signaux continus*, l'opération elle-même est modélisée comme un *nœud de tâche* qui affecte la *valeur* à la *variable.* Cependant, cette opération interagit avec les modèles de *condition de validation* et de *signal continu* (voir les § 3.2 et 3.3).

3.1.2 *Règles d'utilisation des valeurs importées*

- Toutes les *valeurs importées* doivent être définies avec l'attribut *IMPORTED* dans l'*importateur* et avec l'attribut *EXPORDED* dans l'*exportateur.* Dans une *définition de variable*, l'attribut d'exportation rend *visible l'identificateur de variable* dans l'ensemble du système.
- On indique l'origine et la destination de l'*opération* d'*importation* en insérant le *nom* de la *valeur importée* dans une *liste de signaux* attachée à un *canal* ou à un trajet de *signal* à l'intérieur d'un *bloc.* L'origine et la destination peuvent se trouver dans des *blocs* différents ou dans le même bloc.
- Une *instance de processus* qui *importe* des *valeurs* peut aussi en *exporter*, mais elle ne doit pas *importer* de *valeurs* de ses propres *variables.*
- Une *valeur importée* <nom> est *importée* par l'utilisation de l'instruction:

  IMPORT(<nom>, <pid>)

  dans une *tâche*, une *décision* ou une *condition de validation.* Le <nom> est celui de l'objet importé et l'expression «pid» est l'*identificateur d'instance de processus* de l'*instance de processus* à laquelle appartient la variable. Dans le *processus*, toutes les autres références au <nom> seront interprétées comme des références à la copie locale de la *valeur* du <nom>.
- La valeur d'une *variable* exportée est révélée au moyen de l'instruction:

  EXPORT(<nom>):= <expression>

  ou

  EXPORT(<nom>)

  contenue dans une *tâche.*

3.1.3 *Syntaxe concrète*

Les *valeurs importées* sont référencées en trois points dans la *définition de processus.* La syntaxe qui s'y rapporte est la même dans le *LDS/PR* et dans le *LDS/GR.*

3.1.3.1 Dans les *définitions de variable* et les *définitions d'importation*, les mots clés *IMPORTED* et *EXPORTED* indiquent l'usage qu'il faut faire de la *variable* déclarée.

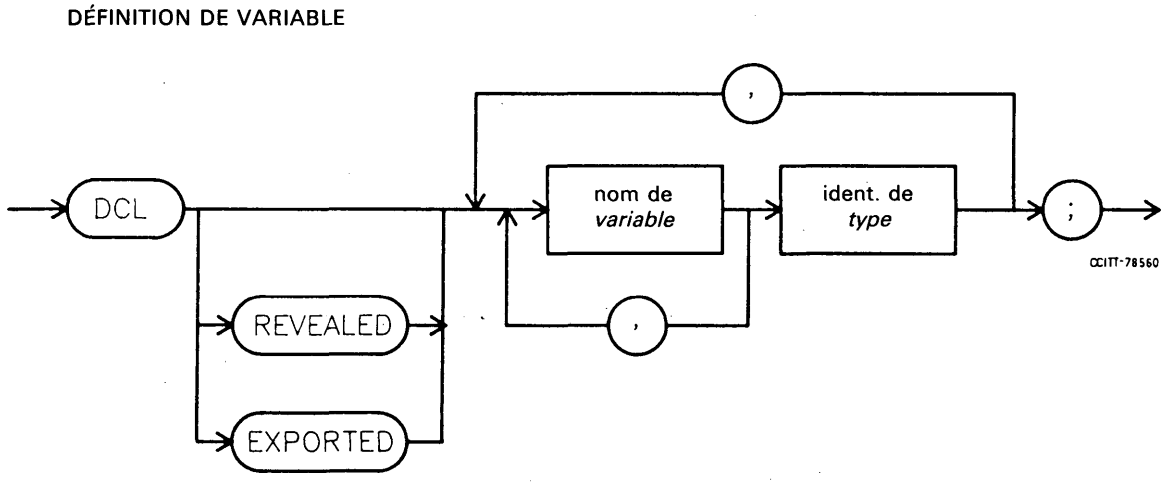


FIGURE 10/Z.103 (feuillet 1 sur 2)

Diagramme de syntaxe pour définition de variable et définition d'importation

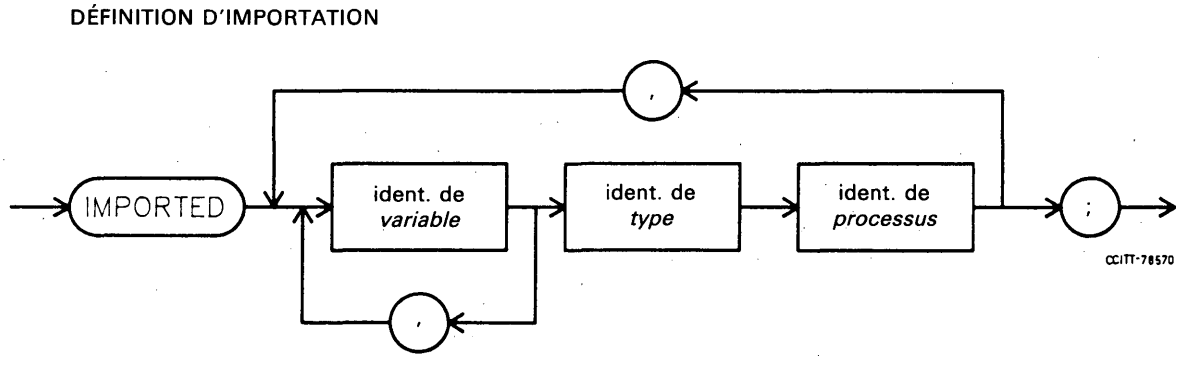


FIGURE 10/Z.103 (feuillet 2 sur 2)

Diagramme de syntaxe pour définition de variable et définition d'importation

3.1.3.2 Dans les *listes de signaux* du *LDS/GR*, une *valeur importée* se distingue d'un *signal* par le fait que son *nom* figure entre parenthèses:

[Nom de signal 1
(Nom de variable)
Nom de signal 2]

La syntaxe dans le *LDS/PR* est la suivante:

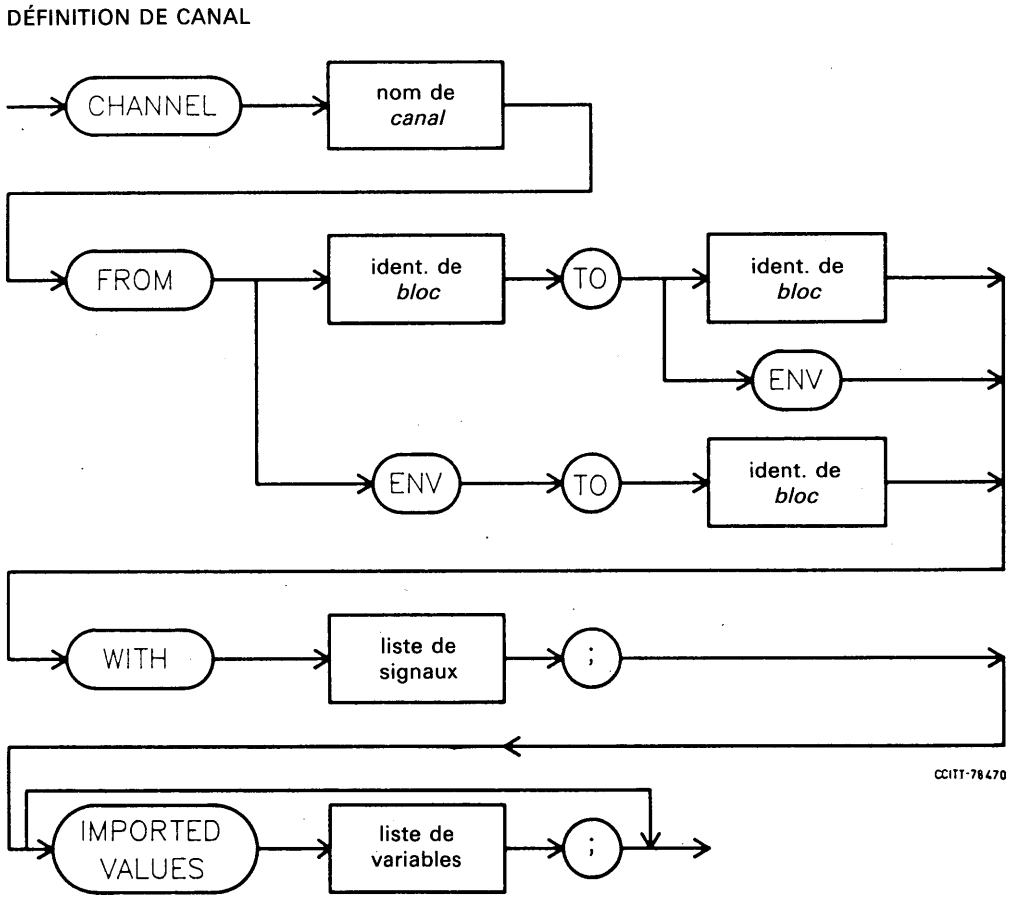


FIGURE 11/Z.103 (feuillet 1 sur 2)

Diagramme de syntaxe pour définition de canal et liste de variables

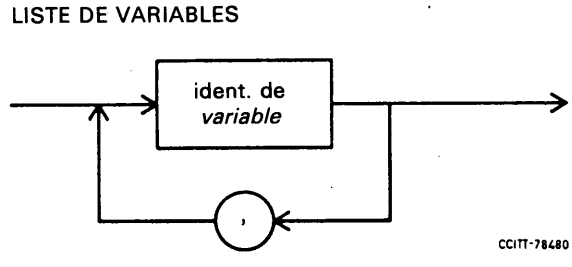


FIGURE 11/Z.103 (feuillet 2 sur 2)

Diagramme de syntaxe pour définition de canal et liste de variables

3.1.3.3 L'instruction *EXPORT*(<nom>) peut être utilisée dans un *nom de tâche ;* de même, l'instruction *IMPORT*(<nom, pid>) peut être utilisée dans un *nom* de *tâche*, de *décision* ou de *condition de validation.*

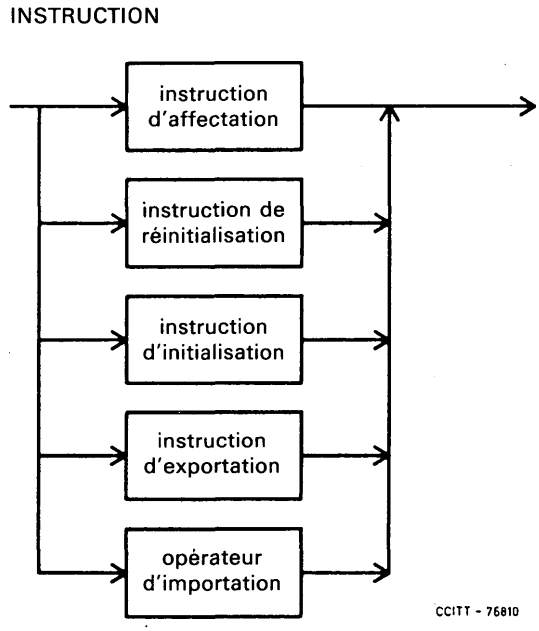


FIGURE 12/Z.103 (feuillet 1 sur 3)

Diagramme de syntaxe pour l'instruction d'exportation

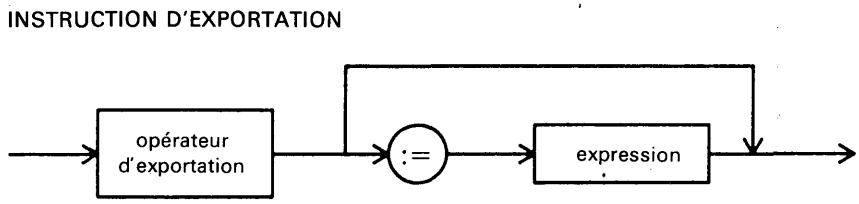


FIGURE 12/Z.103 (feuillet 2 sur 3)

Diagramme de syntaxe pour l'instruction d'exportation

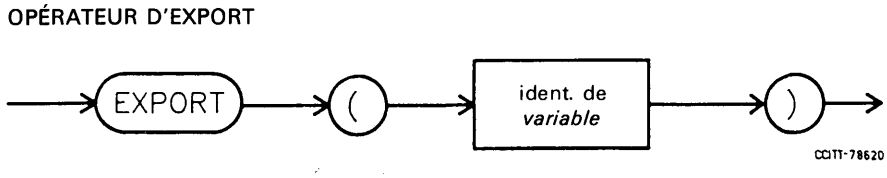


FIGURE 12/Z.103 (feuillet 3 sur 3)

Diagramme de syntaxe pour l'instruction d'exportation

OPÉRATEUR D'IMPORTATION

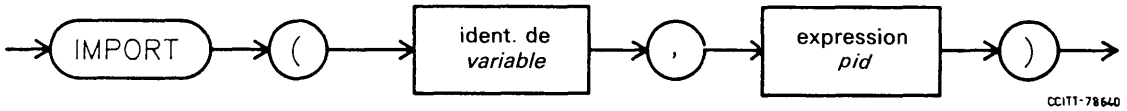


FIGURE 13/Z.103

Diagramme de syntaxe pour l'instruction d'importation

3.2 *Condition de validation*

On parvient souvent à réduire le nombre des *états* d'un *graphe de processus* en faisant en sorte que des conditions soient attachées au début des *transitions*. Ces conditions sont des assertions portant sur une *variable;* elles sont connues sous le nom de *conditions de validation*. Si une condition est vraie et si le *signal d'entrée* associé est retenu, la *transition* se déroule normalement. Si la *condition de validation* n'est pas vraie, la *transition* est rendue impossible et le *signal d'entrée* est alors *mis en réserve*.

Dans le LDS normal, cette exécution conditionnelle des *transitions* devrait être modélisée au moyen d'états *séparés* pour chaque *valeur* de la *condition de validation*. La notation concise fournie par les *conditions de validation* est utile pour simplifier un *graphe de processus*.

Les *conditions de validation* peuvent s'exprimer à l'aide de valeurs locales et/ou de *valeurs importées*, mais jamais de *valeurs partagées*.

La *condition de validation* est décrite au moyen de concepts du LDS de base (voir la Recommandation Z.101). On l'utilise dans un *graphe de processus* et on l'attache à un symbole d'*entrée* ou à une instruction *d'entrée*. C'est une assertion, faisant intervenir des *valeurs de données* qui sont des valeurs locales ou des valeurs *importées*. La condition doit être une expression booléenne, qui donne donc lieu au retour des *valeurs* VRAI ou FAUX. Chaque *état* qui est suivi de *transitions* commandées par des *conditions de validation* représente en fait un ensemble d'*états*, à savoir un pour chaque combinaison possible des *conditions de validation*. On évalue les conditions avant que l'élément approprié de l'ensemble des *états* soit choisi en tant que prochain *état*.

L'interprétation d'une *condition de validation* dépend des *valeurs de données* utilisées dans la condition. Si une *condition de validation* ne contient que des *valeurs* qui sont locales au *processus*, il suffit qu'elle soit évaluée une fois. Si elle contient des valeurs importées (voir le § 3.1), on doit l'évaluer chaque fois que l'une des *valeurs importées* change: on procède à l'évaluation par un échange implicite de *signaux* entre l'*importateur* et le (ou les) *exportateur(s)* des *valeurs*.

Les *conditions de validation* qui utilisent des *valeurs importées* font usage, en plus du signal XQUERY défini au § 3.1, de *signaux* cachés qui sont XATTACH et XDETACH. Tels sont les *signaux* par lesquels on demande que l'*exportateur* ajoute (ou supprime) *l'importateur* à (ou d') une liste des *instances de processus* qui doivent être informées des changements de la *valeur exportée*. L'*exportateur* tient une telle liste implicite (XLIST) pour chacune des *valeurs* qu'il *exporte*. Un autre *signal* (XWAKE) est envoyé par l'*exportateur* à chaque membre de la liste toutes les fois qu'il procède à une opération d'*exportation* sur la *donnée* correspondante.

3.2.1 *Définition*

3.2.1.1 *Condition de validation unique fondée sur des valeurs locales*

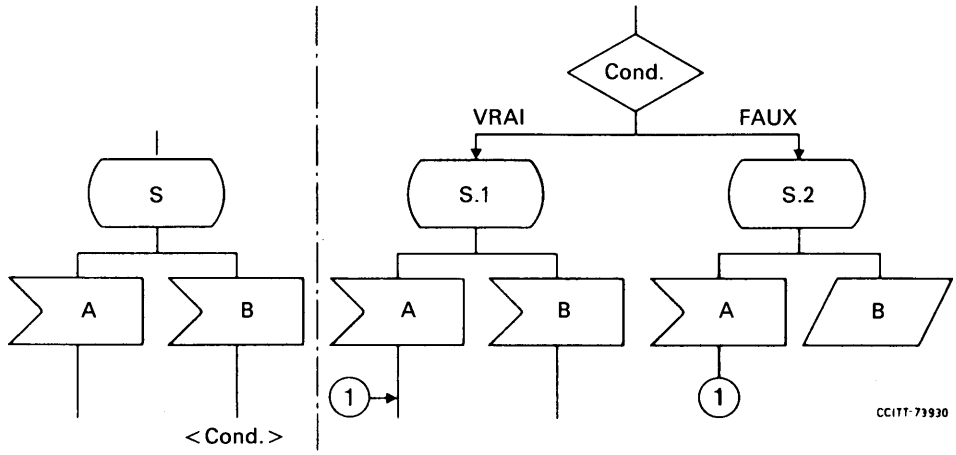
Lorsqu'une seule des *entrées* qui suivent un *état* a une *condition de validation* qui lui est attachée, et si cette condition ne contient que des *valeurs* locales, l'interprétation est la suivante.

L'*état* ainsi que les *entrées* et les *mises en réserve* qui le suivent sont remplacés par:

- un *nœud de décision* qui évalue la *condition de validation ;*
- la branche «VRAI» de la *décision* est suivie d'un *nœud d'état*, avec le même *ensemble de signaux de réserve* que l'*état* original, puis du même ensemble d'*entrée* que l'*état* original (l'*état* auquel une *condition de validation* était attachée apparaît comme un *nœud d'entrée*);
- la branche «FAUX» de la *décision* est suivie d'un *nœud d'état* ayant le même *ensemble de signaux de réserve* que l'*état* original, avec en plus le *nom de signal* de l'*entrée* à laquelle est attachée la *condition de validation ;*
- chacun des *nœuds d'entrée* est suivi des mêmes *transitions* que dans le graphe original.

Dans la figure 14/Z.103 est représentée, à l'aide de la syntaxe graphique du LDS, la définition de la *condition de validation* unique fondée sur des *valeurs* locales.

Remarque – L'utilisateur écrit ceci:



Remarque – Le diagramme est seulement explicatif.

FIGURE 14/Z.103

Condition de validation unique fondée sur des valeurs locales seulement, expliquée en LDS/GR

3.2.1.2 *Condition de validation unique contenant des valeurs importées*

Lorsqu'une seule des *entrées* qui suivent un *état* a une *condition de validation* qui lui est attachée, laquelle contient au moins une *valeur importée*, l'interprétation conduira à un *échange* implicite de *signaux* avec ceux auxquels appartiennent les *valeurs importées.*

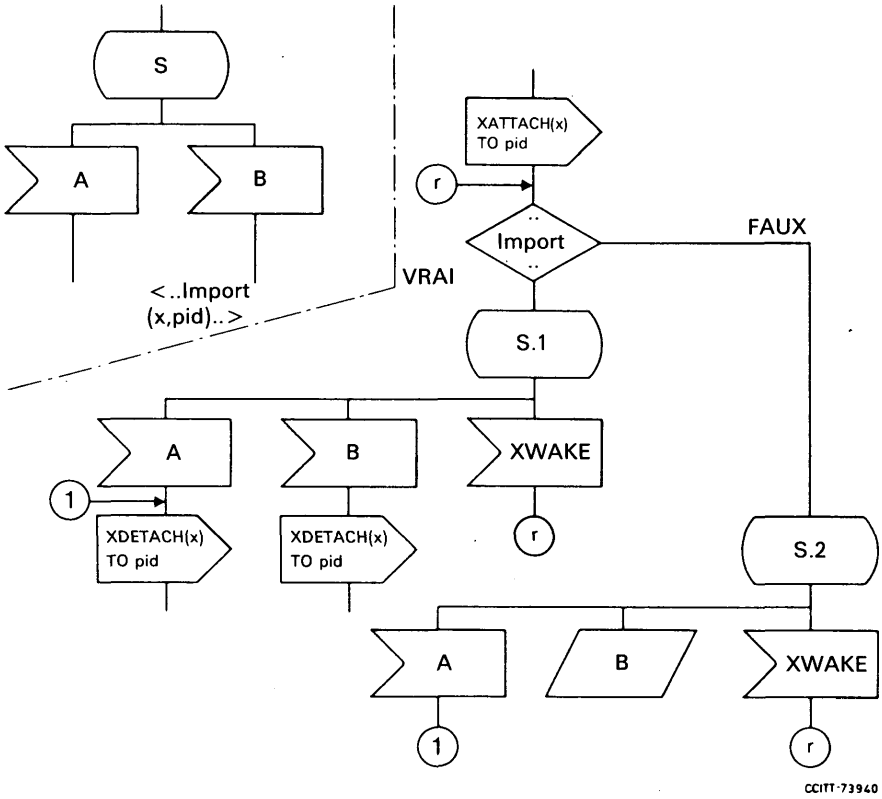
Tandis que les *variables* qui détiennent les *valeurs importées* sont traitées par d'autres *instances de processus*, fonctionnant simultanément, la *valeur* de la *condition de validation* est susceptible de changer pendant que l'*instance de processus* est en attente dans un *état.* Un tel changement devrait conduire à réévaluer la condition. Lorsqu'ils entrent dans une construction de *condition de validation*, ceux à qui appartiennent les *valeurs importées* sont informés de ce que cette *instance* devra être notifiée de tout changement de la *valeur* susceptible de se produire. Par conséquent, lorsqu'une *valeur importée* est changée, un *signal* est envoyé à tous les *processus* qu'il y a lieu d'informer, de manière qu'ils réévaluent les conditions.

La définition est la suivante:

A l'*ensemble de signaux d'entrée valides* du *processus* qui contient la *condition de validation*, on ajoute le *signal* implicite XWAKE. L'*état* et l'ensemble des *entrées* et des *mises en réserve* qui le suivent sont remplacés par:

- une séquence de *nœuds de sortie* pour le *signal* XATTACH (un *nœud* pour chacune des *valeurs importées* utilisées dans la *condition de validation*);
- cette séquence est suivie par une *décision* et par deux *nœuds d'état*, comme au § 3.2.1.1;
- la *décision*, du fait qu'elle contient des *valeurs importées*, est elle-même étendue comme au § 3.1.1.2, de manière à être précédée par des *opérations d'importation* pour chaque *valeur importée* qu'elle utilise;
- en plus de l'extension définie au § 3.2.1.1, les deux *nœuds d'état* sont aussi suivis d'un *nœud d'entrée* pour le *signal* XWAKE, la *transition* suivante ramenant à la *décision* d'évaluation de la condition;
- également en plus de l'extension définie au § 3.2.1.1, chacun des *nœuds d'entrée*, à l'exception du XWAKE, est suivi d'une séquence de *nœuds de sortie* pour le *signal* XDETACH (à savoir un pour chacune des *valeurs importées* utilisées dans la *condition de validation*) après quoi la séquence est suivie de la même *transition* que dans la construction originale.

L'utilisateur écrit ceci:



Remarque – Ce diagramme n'est représenté que pour explication.

FIGURE 15/Z.103

Condition de validation unique contenant des valeurs importées, expliquée en LDS/GR

3.2.1.3 *Mesures prises par l'exportateur lorsque des valeurs importées sont utilisées dans des conditions de validation*

Lorsqu'une *valeur exportée* est utilisée dans une *condition de validation*, les propriétés implicites suivantes s'ajoutent à celles de l'*exportateur* qui sont définies aux § 3.1.1.2 et 3.1.1.3.

Afin de pouvoir procéder à la réévaluation des *conditions de validation* utilisant des *valeurs importées*, l'*exportateur* tient implicitement une liste (XLIST) des *instances de processus* qu'il y a lieu de tenir informées si les *valeurs* changent. Le *signal* XATTACH fait entrer les *instances de processus* dans cette liste et le *signal* XDETACH les en fait sortir.

Chaque fois qu'une nouvelle *valeur* est exportée du fait de l'opération d'*exportation*, toutes les *instances de processus* de la XLIST sont informées de cette nouvelle *valeur* par le *signal* XWAKE.

Les définitions de cette adjonction sont les suivantes:

Chaque *processus* qui *exporte* des *valeurs* est étendu:

a) en ajoutant les *signaux* implicites XATTACH et XDETACH à l'*ensemble des signaux d'entrée valides ;*

b) en définissant une liste XLIST.<nom> pour chaque valeur *exportée.* Les listes contiennent des *identificateurs d'instance de processus.*

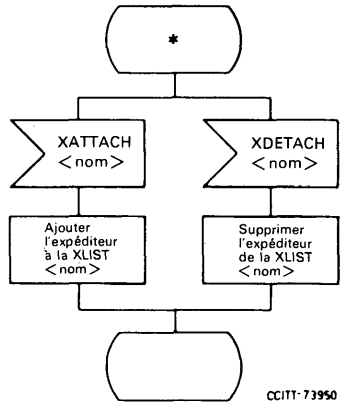
A chaque *état* du *processus*, y compris tous les *états* implicites, sont ajoutées deux *transitions* implicites, en plus de celle qui est définie au § 3.1.1.2. La première de ces *transitions* consiste:

- en un *nœud d'entrée* pour le *signal* XATTACH;
- en une *tâche* qui ajoute l'*identificateur d'instance de processus* de l'expéditeur du *signal* XATTACH à la XLIST correspondant à la *variable* désignée dans le *signal*, dont la *valeur* est *exportée;*
- en un retour à l'*état* dont provenait la *transition.*

La seconde de ces transitions consiste:

- en un *nœud d'entrée* pour le *signal* XDETACH;
- en une *tâche* qui supprime l'*identificateur d'instance de processus* de l'expéditeur du *signal* XDETACH de la XLIST désignée dans le *signal;*
- en un retour à l'*état* dont provenait la *transition.*

Les *transitions* implicites supplémentaires de chaque *état* de *l'exportateur* sont représentées ci-dessous au moyen de la syntaxe graphique du LDS:



Remarque – Ce diagramme n'est représenté que pour explication. Toutes les transitions sont implicites.

FIGURE 16/Z.103

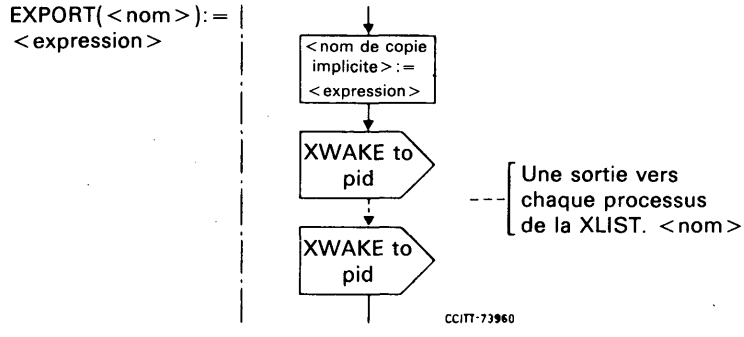
Transitions implicites du processus d'exportation expliquées en LDS/GR

La transition *implicite* suivante est ajoutée à la définition de l'*opération d'exportation* du § 3.1.1.3:

- A la suite d'une *tâche*, comprenant une opération d'*export*, vient une séquence de *nœuds de sortie* pour le *signal* XWAKE qui transporte la *valeur* révélée. Il y aura un *signal* XWAKE envoyé à chaque *instance de processus* comprise à ce moment dans la XLIST pour la variable correspondante à laquelle cette *valeur* était révélée.

L'adjonction à l'*opération d'exportation* est représentée ci-dessous au moyen de la syntaxe graphique du LDS.

L'utilisateur écrit:



Remarque – Ce diagramme n'est représenté que pour explication.

FIGURE 17/Z.103

Actions explicites de l'opération exportation expliquées en LDS/GR

3.2.1.4 *Conditions de validation multiples*

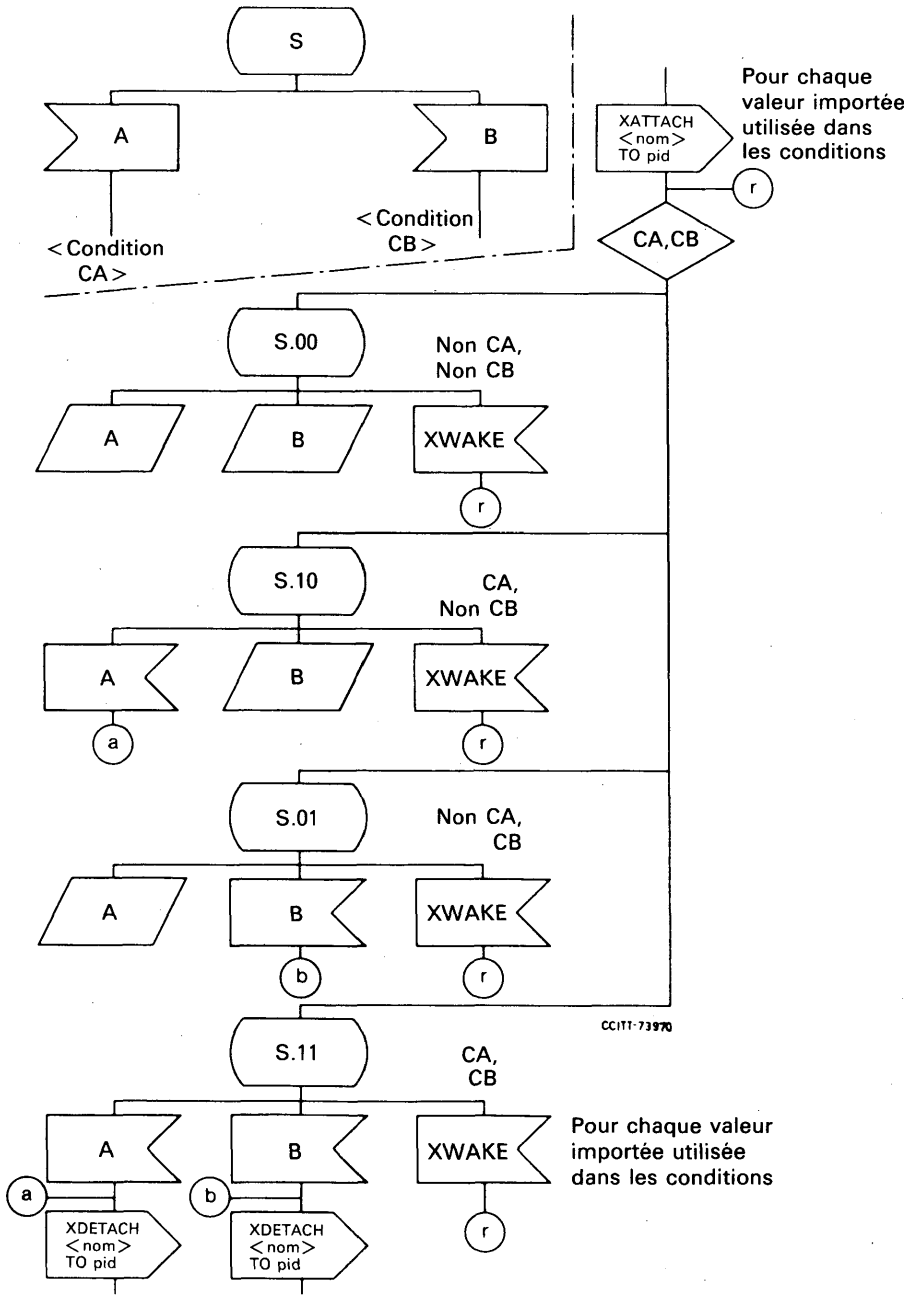
Dans le cas où il y a plusieurs *conditions de validation* qui suivent un *état*, on parle de *conditions de validation* multiples.

Si, parmi les nœuds d'*entrée* qui suivent un *état*, il y en a *n* auxquels sont attachées des *conditions de validation*, les définitions des *conditions de validation uniques* sont étendues comme suit:

a) L'*état* ainsi que les *entrées* et les *mises en réserve* qui le suivent sont représentés par un *nœud de décision* pour une condition complexe, dont la *valeur* est égale à n-multiplets, avec ses éléments, les valeurs des *conditions de validation* élémentaires. Ce *nœud* est suivi de 2**n *nœuds d'état*, dont chacun est suivi du traitement de *transition* approprié *aux valeurs* particulières des *conditions de validation* dont l'arc qui conduit à ce *nœud d'état.*

b) Si l'une quelconque des *conditions de validation* se réfère à des *valeurs importées*, les définitions du § 3.2.1.2 s'appliquent aussi.

L'utilisateur écrit:



Remarque – Ce diagramme n'est représenté que pour explication.

FIGURE 18/Z.103

Conditions de validation multiples expliquées en LDS/GR

3.2.2 *Règles d'utilisation des conditions de validation*

1) Des *conditions de validation* peuvent être attachées à tout *signal d'entrée.*

2) Il ne peut y avoir qu'une *entrée* à la suite d'un *état* contenant un *signal d'entrée* donné, cela indépendamment du fait qu'une *condition de validation* soit ou non attachée à cette *entrée.*

3.2.3 *Syntaxe concrète*

3.2.3.1 *LDS/GR*

Les *conditions de validation* sont représentées dans le *LDS/GR* par un *symbole de condition de validation* venant après un *symbole d'entrée.* Cet élément est constitué par une paire de crochets obliques entre lesquels est indiquée une condition *booléenne,* comme cela est illustré sur la figure 19/Z.103:

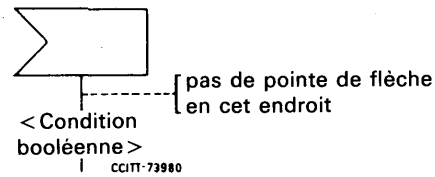


FIGURE 19/Z.103

Syntaxe LDS/GR pour la condition de validation

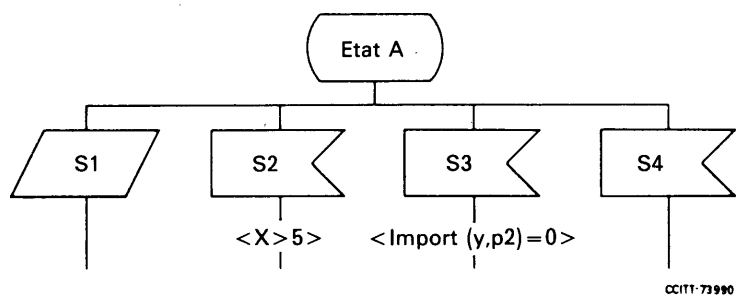


FIGURE 20/Z.103

Exemple d'utilisation de la syntaxe LDS/GR pour la condition de validation

3.2.3.2 *LDS/PR*

Dans le *LDS/PR*, la syntaxe de l'instruction d'*entrée* est modifiée par l'adjonction du mot *PROVIDED*, lequel contient une expression *booléenne*. Le diagramme syntaxique pour l'instruction d'*entrée* modifiée est représenté sur la figure 21/Z.103:

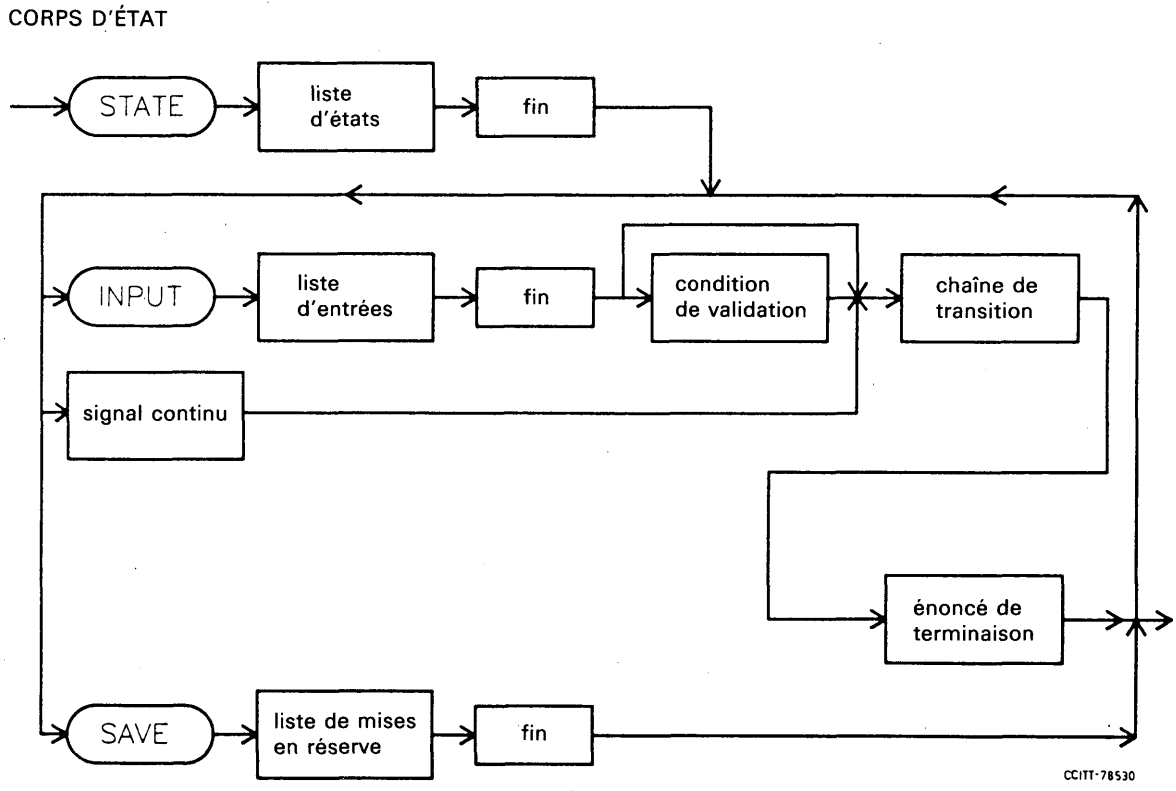


FIGURE 21/Z.103 (feuillet 1 sur 2)

Diagramme de syntaxe pour énoncé d'entrée, comportant le mot PROVIDED

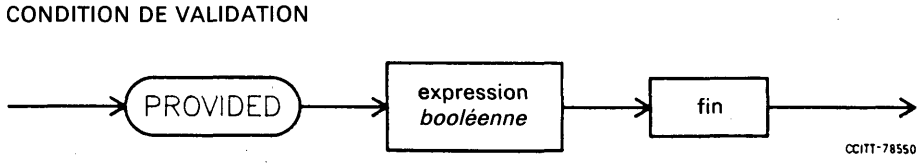


FIGURE 21/Z.103 (feuillet 2 sur 2)

Diagramme de syntaxe pour énoncé d'entrée, comportant le mot PROVIDED

3.3 *Signal continu*

Lorsque l'on décrit un *système* à l'aide du LDS, on se trouve souvent dans un cas où l'usager aimerait montrer une *transition* comme causée par un changement de la *valeur* d'une *variable* extérieure au *processus.* Il pourrait, par exemple, s'agir d'une tension, élevée ou basse, sur une ligne, ou d'un nombre dans un registre d'état. Pour y parvenir dans le LDS, il serait normal de faire en sorte qu'un *signal* soit engendré au moment où le changement de la *valeur* se produit, et de fonder la *transition* sur la réception de ce *signal.* L'obligation de *définir* explicitement, d'engendrer et de recevoir de tels *signaux* risque de compliquer les *graphes de processus.* L'*opération composite* appelée *signal continu* permet qu'un changement de la *valeur* d'une condition déclenche directement une *transition.*

Une *condition de validation* représente une *décision* avant l'entrée dans un *état.* Lorsque l'on utilise des *valeurs importées,* les *conditions de validation* fournissent des moyens de sortir d'un *état* grâce au *signal* implicite XWAKE. On peut également l'utiliser sans *signal d'entrée* associé, auquel cas il devient un *signal continu.* Dans ce cas, la *condition de validation* ne représente pas un état *implicite* supplémentaire, mais au contraire définit une circonstance dans laquelle on peut quitter l'*état* duquel la *condition de validation* découle. Cette circonstance a une *priorité* inférieure à celle des *signaux* retenus.

Plusieurs *signaux continus* peuvent partir du même *état* et peuvent être tels que plusieurs d'entre eux soient vrais simultanément. A chaque *signal continu* est associée une *priorité* qui détermine l'ordre relatif dans lequel les *signaux continus* sont essayés.

Un *signal continu* n'utilisant que des *valeurs* locales fournit un moyen de quitter conditionnellement un *état* si aucun *signal* n'est en attente à l'accès d'entrée au moment de son entrée. Un *signal continu* qui utilise des *valeurs importées* ajoute à cela la capacité de réévaluer la condition lorsqu'il y a un changement dans l'une des *valeurs importées* utilisées dans cette condition.

3.3.1 *Définition*

La définition ci-après est fondée sur celles de l'*importation* et de l'*exportation* des *valeurs* (voir le § 3.1) et des *conditions de validation* (voir le § 3.2).

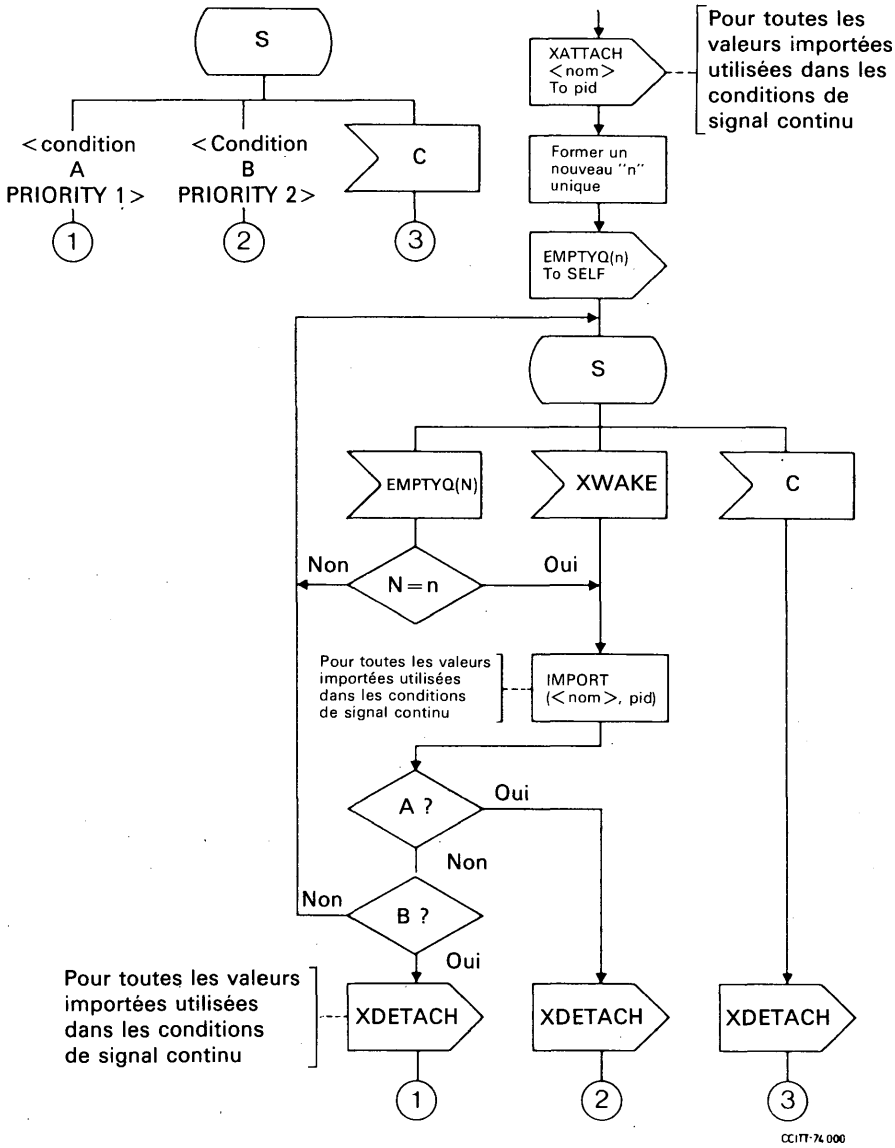
L'interprétation d'un *état* suivi de *signaux continus* est présentée comme un modèle général dans lequel plusieurs *signaux continus* désignent collectivement plusieurs *valeurs importées.* S'il n'est pas fait usage de *valeurs importées,* le modèle est simplifié du fait de l'élimination des *sorties* XATTACH et XDETACH et de l'*entrée* XWAKE.

L'*état* ainsi que l'ensemble des *entrées* et des *mises en réserve* qui le suivent, avec les *signaux continus,* sont remplacés par:

1) une séquence de *nœuds de sortie* pour le *signal* XATTACH, à raison d'un pour chacune des *valeurs importées* utilisées dans la condition de *signal continu;*
2) une *tâche* qui crée une *valeur* unique à utiliser dans le *signal* EMPTYQ utilisé en 3);
3) un *nœud de sortie* pour un *signal* EMPTYQ qui est envoyé à l'*identité d'instance de processus* de l'expéditeur, c'est-à-dire à son propre *accès d'entrée;*
4) un *nœud d'état,* comme dans le *graphe de processus* ou de *procédure* original, suivi par un ensemble de *nœuds d'entrée,* qui comprend l'ensemble original et deux autres *nœuds d'entrée;*
5) chaque *nœud d'entrée* est suivi d'une séquence de *nœuds de sortie* pour le *signal* XDETACH, à raison d'un pour chacune des *valeurs importées* utilisées dans les conditions de *signal continu.* Cette séquence est suivie du traitement de *transition* qui suivait à l'origine le *nœud d'entrée;*
6) le *nœud d'état* de 4) est également suivi par un *nœud d'entrée* pour le *signal* XWAKE, ce qui déclenche la *transition* de 7);
7) une séquence de *nœuds de tâche,* pour *importer* chacune des *valeurs importées* utilisées dans les conditions de *signal continu;*
8) une séquence de *nœuds de décision* pour chacune des *conditions de signal continu,* la première décision évaluée étant celle relative au *signal continu* de plus forte priorité (le nombre le plus bas dans la syntaxe concrète);
9) la branche FAUX de chaque *décision* conduit à un *nœud de décision* pour la condition de *signal continu* de la *priorité* immédiatement inférieure suivante. La branche FAUX de la *décision de signal continu* de la priorité la plus faible ramène à l'*état* de 4);
10) la branche VRAI de chaque *décision* conduit à une séquence de *nœuds de sortie* pour le *signal* XDETACH, à raison d'un pour chacune des *valeurs importées* utilisées dans les conditions de *signal continu,* suivie par le traitement de *transition* correspondant à la condition de *signal continu* essayée dans la *décision;*
11) le *nœud d'état* de 4) est également suivi d'un *nœud d'entrée* pour le *signal* EMPTYQ, à son tour suivi d'un *nœud de décision* qui vérifie si le *signal* transporte bien la *valeur* qui lui a été donnée en 2), c'est-à-dire s'il s'agit du *signal* qui a été envoyé en 3), et non d'un signal EMPTYQ antérieur non traité. La branche VRAI de cette *décision* conduit au traitement de *signal continu* de 7) et sa branche FAUX ramène au *nœud d'état* de 4).

L'ensemble de *signaux d'entrée valides* du *processus* qui contient des *signaux continus* utilisant des *valeurs importées* est étendu par le *signal* implicite XWAKE. Les *ensembles de signaux d'entrée valides* de tout *processus* contenant des *signaux continus* sont étendus par le *signal* implicite EMPTYQ.

L'utilisateur écrit ceci:



Remarque – Ce diagramme n'est représenté que pour explication.

FIGURE 22/Z.103

Signal continu expliqué dans le LDS/GR

3.3.2 *Règles d'utilisation des signaux continus*

1) Un *signal continu* peut suivre n'importe quel *état.*

2) La *condition de signal continu* peut être fondée sur des *valeurs locales* et/ou sur des *valeurs importées.*

3) Deux *signaux continus* qui suivent le même *état* ne peuvent jamais avoir le même numéro de priorité.

3.3.3 *Syntaxe concrète*

3.3.3.1 *LDS/GR*

Dans le *LDS/GR*, un *signal continu* est indiqué par un symbole de *condition de validation* qui suit directement un symbole *d'état*, c'est-à-dire que la *transition* n'est pas précédée par un *symbole d'entrée*. Le symbole contient, outre la *condition de signal continu*, le mot clé PRIORITY suivi d'un numéro de *priorité* (nombre naturel). Plus ce nombre est faible, plus est élevée la *priorité* du *signal continu*.

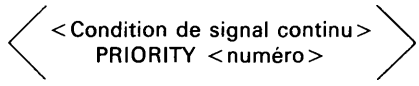


FIGURE 23/Z.103

Symbole LDS/GR pour le signal continu

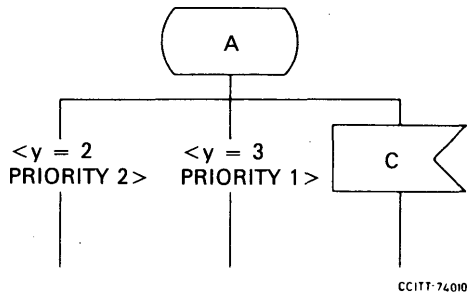


FIGURE 24/Z.103

Exemple de l'utilisation du signal continu LDS/GR

Si un seul *signal continu* suit un *état*, la clause de PRIORITÉ peut être supprimée. Si la clause est supprimée, le numéro de priorité «1» est mis en œuvre.

3.3.3.2 *LDS/PR*

Dans le *LDS/PR*, une instruction PROVIDED, suivie par une chaîne de *transitions*, représente le *signal continu*. Cette instruction contient une clause de PRIORITY; plus le nombre indiqué dans cette clause est faible, plus est élevée la *priorité* du *signal continu*.

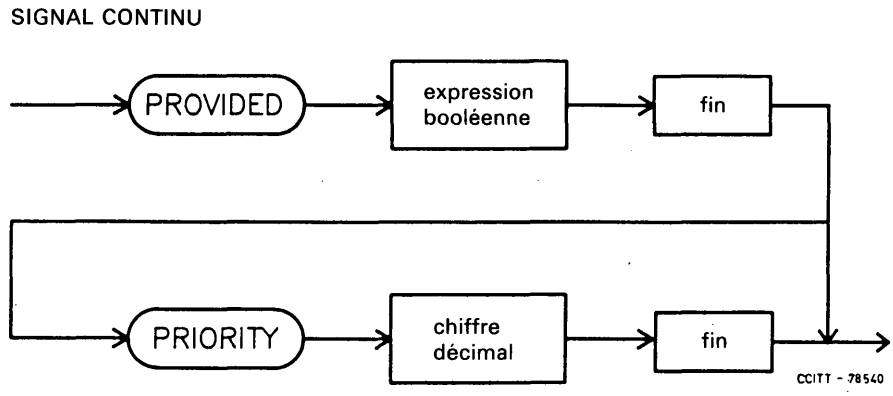


FIGURE 25/Z.103

Diagramme de syntaxe pour signal continu

4 Macros

Une *macro* est une abréviation, définie par l'usager, qui peut être incluse en un ou en plusieurs endroits dans la représentation LDS concrète d'un *système*. Elle représente une référence à une définition qui figure quelque part ailleurs dans un document. Une *macro* n'est qu'une partie de la syntaxe concrète; il faut qu'elle soit remplacée par le corps de sa définition pour que soit interprétée la représentation LDS dans laquelle il apparaît.

4.1 *Définition*

Une **macro** peut représenter n'importe quelle collection d'objets syntaxiques; toutefois, pour des raisons évidentes (expansion infinie!), elle ne peut pas être récursive.

Une *macro* peut avoir zéro, un ou plusieurs *accès entrants* et zéro, un ou plusieurs *accès sortants*. S'il y a plusieurs *accès entrants*, il convient que soit attachée à chacun d'eux une *étiquette* correspondant à l'*étiquette d'accès entrant* de la *définition de la macro;* s'il y a un seul *accès entrant*, l'*étiquette* n'est pas nécessaire. Il en va de même pour les *accès sortants*.

Ni *portée*, ni *visibilité* ne sont associées au concept de *macro* en tant que tel. L'interprétation d'une référence à une *macro* ne peut être obtenue que lorsque la *macro* est remplacée par sa définition.

4.2 *Syntaxe concrète*

4.2.1 *LDS/GR*

4.2.1.1 *Syntaxe*

La référence à une *définition* de *macro* est indiquée sur le symbole de *macro* du *LDS/GR*.

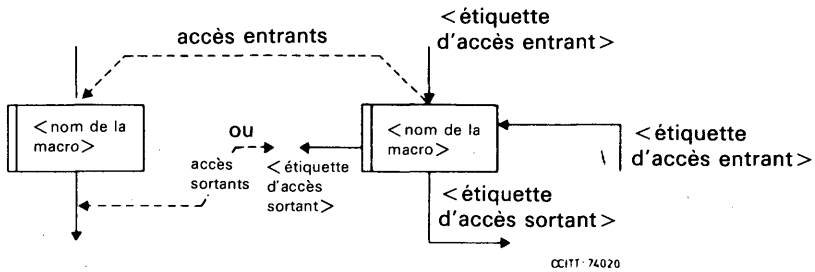


FIGURE 26/Z.103

Symbole de macro dans le LDS/GR

Les accès *entrants* et *sortants* sont représentés par des *lignes de liaison* qui conduisent au symbole ou qui en partent. Des *étiquettes* peuvent facultativement être attachées à ces *lignes de liaison.*

Le symbole de *macro* contient le *nom* de la définition de la *macro* à laquelle il se réfère; un *commentaire* peut être attaché au symbole.

La *définition de macro* se présente comme suit:

<nom> MACRO DÉFINITION

Le <nom> est le *nom* de la *macro*, utilisé dans un symbole de *macro.*

La définition de *macro* contient la représentation graphique qui remplace le symbole de *macro* avant que l'interprétation ait lieu. Les *accès entrants* et *sortants* provenant de la définition sont représentés par des lignes de liaison qui conduisent aux symboles ou qui en partent; aux uns et aux autres peuvent être attachées des *étiquettes.*

4.2.1.2 *Symboles*

Deux symboles supplémentaires sont représentés dans la figure 27/Z.103:

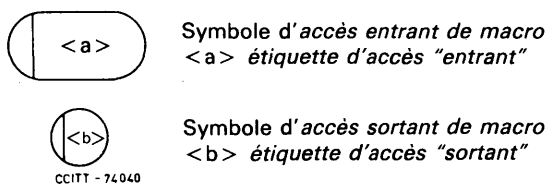


FIGURE 27/Z.103

Symboles supplémentaires de macro

4.2.1.3 *Règles d'utilisation des macros dans le LDS/GR*

Le symbole de *macro* peut être inséré en n'importe quel endroit dans un diagramme et il peut représenter n'importe quelle collection de symboles *LDS/GR.*

Il doit y avoir, dans un *symbole de macro*, autant d'*accès entrants* qu'il y en a dans la *définition de macro* référencée. L'ensemble des *étiquettes* attachées aux *accès entrants* du *symbole de macro* doit être le même que l'ensemble des *étiquettes* attachées aux *accès entrants* de la *définition de macro.* La même règle s'applique aux *accès sortants* et à leurs *étiquettes.*

Les *accès entrants* peuvent aboutir sur n'importe quel côté du symbole et les *accès sortants* peuvent partir de n'importe quel côté du symbole.

La sémantique d'une *macro* s'obtenant par le remplacement de la référence par la collection des symboles de la définition, toutes les conventions graphiques s'appliquent au diagramme dans lequel toutes les apparitions de la macro ont été remplacées. Il peut en résulter des conséquences inattendues, comme, par exemple, l'apparition multiple d'*états.* Cette question est examinée plus en détail dans les Directives pour les utilisateurs.

4.2.2 *LDS/PR*

4.2.2.1 *Syntaxe*

L'appel macro peut être inséré dans tout diagramme utilisant la syntaxe:

MACRO nom de macro;

L'expansion de macro est un morceau d'un programme LDS/PR commençant par:

MACRO EXPANSION nom de macro;

et se terminant par:

ENDMACRO nom de macro;

dans le dernier énoncé, le nom de macro n'est pas obligatoire.

Cette définition doit être placée immédiatement après la construction SYSTEM, BLOCK ou PROCESS respectivement selon ce à quoi la *macro* est rattachée. La *définition de macro* peut contenir n'importe quel caractère et l'on ne peut vérifier qu'elle est correcte qu'après qu'elle a remplacé l'instruction de *macro.*

4.2.2.2 *Règles d'utilisation des macros dans le LDS/PR*

Les règles et considérations valables lorsque les *macros* sont utilisées dans le cadre du *LDS/GR* s'appliquent encore lorsqu'elles sont utilisées dans celui du *LDS/PR.*

5 Options

Lorsque plusieurs applications semblables sont spécifiées ou décrites au moyen du LDS, il est fréquent que l'on puisse utiliser la même *définition de processus* dans les différents *systèmes*, sous réserve d'une légère modification. La facilité *OPTION* introduite dans le LDS donne la possibilité de définir des *processus* valables pour plusieurs applications, cela en introduisant des variantes optionnelles dans les descriptions.

5.1 *Définition*

Une *option* est le choix entre plusieurs variantes d'une *définition de processus* effectué d'après l'évaluation d'une *expression d'option.* Ce choix se fait avant que la *définition de processus* soit interprétée.

La facilité *option* n'est qu'une partie de la syntaxe concrète; on devrait la considérer comme une notation sténographique grâce à laquelle on peut avoir une description générique valable pour plusieurs applications, plutôt qu'autant de descriptions qu'il y a d'applications.

5.2 *Syntaxe concrète*

5.2.1 *LDS/GR*

5.2.1.1 *Symboles*

Dans le *LDS/GR*, l'*option* est représentée par le symbole suivant:

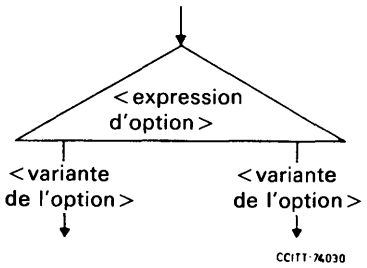


FIGURE 28/Z.103

Le symbole option dans le LDS/GR

5.2.1.2 *Règles d'utilisation des options dans le LDS/GR*

Le symbole d'*option* peut suivre un symbole de *tâche*, un *accès sortant de symbole de décision*, un *symbole de sortie* ou un *symbole de procédure* dans un *diagramme de processus*. Le symbole d'option peut être suivi par un *symbole* d'*état*, un *symbole* de *tâche*, un *symbole* de *décision*, un *symbole* de *sortie* ou un *symbole* de *procédure*.

L'*< expression d'option >* contenue dans le symbole est une expression telle qu'une et une seule des *< variantes >* qui suivent ce symbole soit choisie après l'évaluation, et cela de façon telle qu'on puisse l'évaluer avant d'interpréter le *processus*. Chaque *variante* doit être une *valeur* du même *type* que l'*expression d'option*. Une fois que le *processus* résultant a été interprété, les parties de la *définition de processus* que l'on ne peut pas atteindre devraient être considérées comme supprimées.

5.2.2 *LDS/PR*

5.2.2.1 *Syntaxe*

L'*option* est représentée, dans le *LDS/PR*, par la syntaxe suivante:

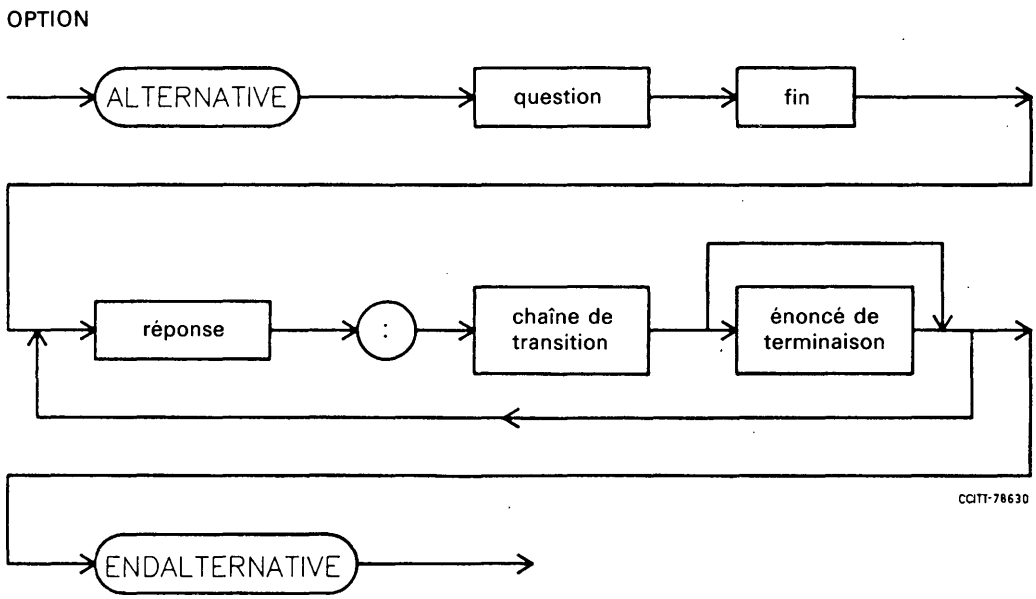


FIGURE 29/Z.103

Diagramme de syntaxe pour option

5.2.2.2 *Règles d'utilisation des options dans le LDS/PR*

L'< *expression d'option* > contenue dans l'instruction est une expression telle qu'une et une seule des < *variantes*> puisse être choisie après l'évaluation, et cela de façon telle qu'on l'évalue avant d'interpréter le *processus.* Chaque *variante* doit être une *valeur* du même *type* que l'*expression d'option.* Une fois que le *processus* résultant a été interprété, les parties de la *définition de processus* que l'on ne peut pas atteindre devraient être considérées comme supprimées.

6 Eléments graphiques dans le LDS/GR

Quand on emploie la syntaxe graphique du LDS pour représenter des *définitions de processus*, l'utilisation d'*éléments graphiques* pour former une *illustration d'état* à l'intérieur d'un symbole *d'état* est une partie facultative du *LDS/GR.*

Ces *illustrations d'état* peuvent présenter des avantages quand on les applique à certaines *définitions de systèmes*, car elles conduisent à des *diagrammes de processus* plus compacts et contenant moins de texte. Une *illustration d'état* décrit, à l'aide d'*éléments graphiques* et d'un *texte d'accompagnement*, l'état réel du *processus* lorsqu'il est dans cet *état.* De même, l'état supposé de l'*environnement* du *processus* peut être décrit dans l'*illustration d'état.* Quand on emploie des *illustrations d'état*, les mesures à prendre lors des *transitions* entre les *états* résultent implicitement des différences entre les états décrits.

Quand on emploie des *éléments graphiques*, la syntaxe et la sémantique décrites dans la Recommandation Z.101 sont applicables. Cependant, cette syntaxe et cette sémantique sont étendues conformément aux définitions ci-après.

6.1 *Sémantique des illustrations d'état*

Quand on emploie des *éléments graphiques*, un *nœud d'état* est représenté par un *symbole d'état.* Ce dernier est identifié par son *nom* et il contient une *illustration d'état* consistant en des *éléments graphiques*, des *valeurs* de variables, *variables d'entrée* et un texte d'accompagnement.

Une *illustration d'état* peut représenter:

- une utilisation d'éléments graphiques et un texte d'accompagnement, les *valeurs* dans cet *état* d'un sous-ensemble choisi de l'ensemble des *variables* associées à ce *processus.* Ledit sous-ensemble peut contenir des *variables* qui servent uniquement de «mandataires» pour les *variables* associées à d'autres *processus.* Ces «mandataires» transportent les *valeurs* des *variables* associées à d'autres *processus.* Ces valeurs sont obtenues soit par *visibilité*, soit par *importation de valeurs;*
- une utilisation des *valeurs* de *variables d'entrée*, des *actions d'entrée*, en ce qui concerne les *signaux d'entrée valides* pour cet *état.*

Le répertoire des *éléments graphiques* est en principe illimité, puisque l'on peut toujours imaginer de nouveaux *éléments graphiques* adaptés à une application nouvelle du LDS. Toutefois, dans les applications à la commutation et à la signalisation dans les télécommunications, il s'est révélé que le répertoire d'*éléments graphiques* suivant est d'une souplesse considérable:

- frontières du processus (à gauche ou à droite),
- équipements terminaux (divers),
- récepteur de signalisation,
- émetteur de signalisation,
- émetteur-récepteur de signalisation,
- surveillance d'un processus par un temporisateur,
- chemin de commutation (connecté, réservé),
- module de commutation,
- taxation en cours,
- élément de contrôle,
- symbole d'incertitude.

Les symboles standards recommandés pour ces *éléments graphiques* sont indiqués au § 6.3 ci-après.

6.2 *Règles d'interprétation*

1) Les *variables d'entrée* sont des *variables booléennes* et chacune d'elles correspond à un et un seul *signal* appartenant *à l'ensemble des signaux d'entrée valables* pour le *processus.* Un changement de la *valeur* d'une *variable d'entrée* entre les illustrations d'état représente toujours l'absorption d'un *signal* par une *opération d'entrée* d'un *processus.* En conséquence, on peut utiliser des *variables d'entrée* pour représenter celles des conditions d'un *processus* qui, si elles subissent un changement, entraîneront une *transition* de la part du *processus.* Les *valeurs* d'une *variable d'entrée* peuvent être associées à un *élément graphique.*

2) La présence d'*éléments graphiques* dans une *illustration d'état* indique des *valeurs* spécifiques pour un sous-ensemble de variables pendant que ce *processus* est dans cet *état*. Les *valeurs* des *variables* additionnelles, notamment de celles qui sont associées à ce sous-ensemble initial, peuvent être indiquées par le texte d'accompagnement des *éléments graphiques*. Ce texte n'est pas une *variable d'entrée*. Une modification de ce texte NE représente PAS l'absorption d'un *signal d'entrée* dans une *entrée* du processus.

3) *Position*

   a) La position d'un *élément graphique* quelconque (autre qu'une *frontière de processus*) par rapport à une *frontière de processus* détermine le caractère, interne ou externe, de cet *élément graphique* par rapport au *processus*. Un *élément graphique* interne représente des *variables* qui appartiennent à ce *processus*. Un *élément graphique* externe représente des *variables* qui appartiennent à un autre processus de telle sorte qu'il faut utiliser le mécanisme de *visibilité* et d'*importation* pour accéder à ces *variables*.

   b) La règle a) s'applique aussi à la distinction entre texte d'accompagnement *interne* et externe. Il suffit de remplacer chaque fois l'expression «texte d'accompagnement» pour les *éléments graphiques* dans cette règle.

4) *Règle essentielle*

   Tout le traitement impliqué dans le passage d'un *état* au suivant est celui qu'il faut effectuer pour faire les changements dans les *illustrations d'état*, auquel s'ajoute le traitement indiqué dans toutes les *décisions*, *sorties* ou *tâches* qui apparaissent dans la *transition* entre les *états*. C'est ainsi que:

   a) Le passage de l'apparition d'un *élément graphique interne* dans un certain *état* à l'absence de cet *élément graphique* dans l'*état* suivant (ou vice versa) correspond à un changement des *valeurs* de certaines variables que l'on peut représenter d'une façon équivalente par l'utilisation d'une *tâche* dans la *transition* entre les *états*.

   b) Le passage de l'apparition d'un *élément graphique externe* dans un *état* à l'absence de cet *élément graphique* dans l'*état* suivant (ou vice versa) correspond à un changement dans des *variables* appartenant à un autre *processus*. Il revient au même de représenter ce changement par un *signal* de *sortie* en direction de cet autre *processus* ou tout simplement par le *signal d'entrée* en provenance de ce *processus*.

   c) Les règles a) et b) s'appliquent aussi à l'apparition ou la disparition de *texte d'accompagnement*. Il suffit de remplacer chaque fois l'expression «texte d'accompagnement» par les *éléments graphiques* dans ces règles.

5) Pour un *diagramme de processus* donné, la position d'un *élément graphique* particulier (ou d'une combinaison particulière d'*éléments graphiques* et d'un texte d'accompagnement) est unique au sein de l'*illustration d'état*, de sorte que l'on peut rapidement déterminer si cet *élément graphique* (ou cette combinaison) est présent dans un symbole d'*état*, ou en est absent, et cela en comparant l'*illustration d'état* avec d'autres *illustrations d'état* dans le *diagramme de processus*.

6) Lorsqu'un *envoyeur de signalisation* apparaît dans une *illustration d'état*, le texte d'accompagnement qui lui est joint identifie un *signal* qui est *sorti* antérieurement à cet état (ou, dans le cas d'un *signal continu* commandé par le *processus*, antérieurement à cet *état* et pendant cet *état*).

6.3 *Symboles recommandés pour les éléments graphiques*

Quand on utilise des *éléments graphiques*, chaque *état* est représenté par un *symbole d'état* contenant une *illustration de l'état*. Le format est celui que représente la figure 30/Z.103:

Un jeu élémentaire d'*éléments graphiques* a été normalisé pour utilisation dans le *LDS/GR* avec application au *système de description* des processus de traitement des appels en télécommunications, y compris les protocoles de signalisation, les services du réseau et les procédés d'interfonctionnement de la signalisation. Beaucoup de ces *éléments graphiques* peuvent être utilisés dans des applications du *LDS/GR* dépassant les processus de traitement des appels et leur application à d'autres processus de télécommunications est encouragée, si elle est pertinente.

L'ensemble des symboles recommandés pour l'ensemble de base des *éléments graphiques* est représenté à la figure 31/Z.103.

Le choix des illustrations pour les *éléments graphiques* est fondé sur les considérations et sur les critères généraux de sélection exposés dans l'annexe C à la présente Recommandation. Il y a lieu de consulter cette annexe avant d'élaborer des symboles supplémentaires d'*éléments graphiques* pour des applications plus larges du *LDS/GR*.

L'annexe B à la présente Recommandation indique les proportions recommandées pour les symboles d'*éléments graphiques*.

Le gabarit qui se trouve à l'intérieur de la couverture arrière de ce fascicule et qui permet de dessiner l'ensemble de base des symboles du *LDS/GR* comporte les symboles d'*éléments graphiques* représentés à la figure 31/Z.103.

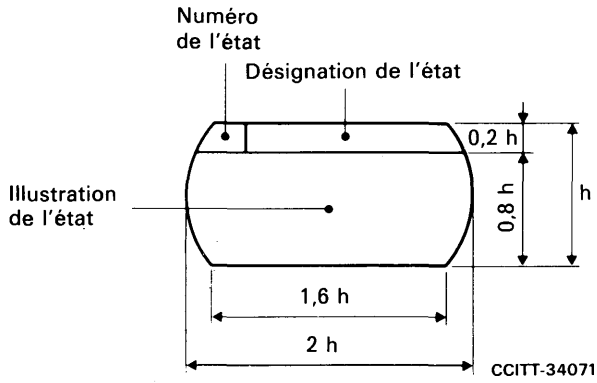


FIGURE 30/Z.103

Format recommandé pour un symbole d'état avec illustration de l'état

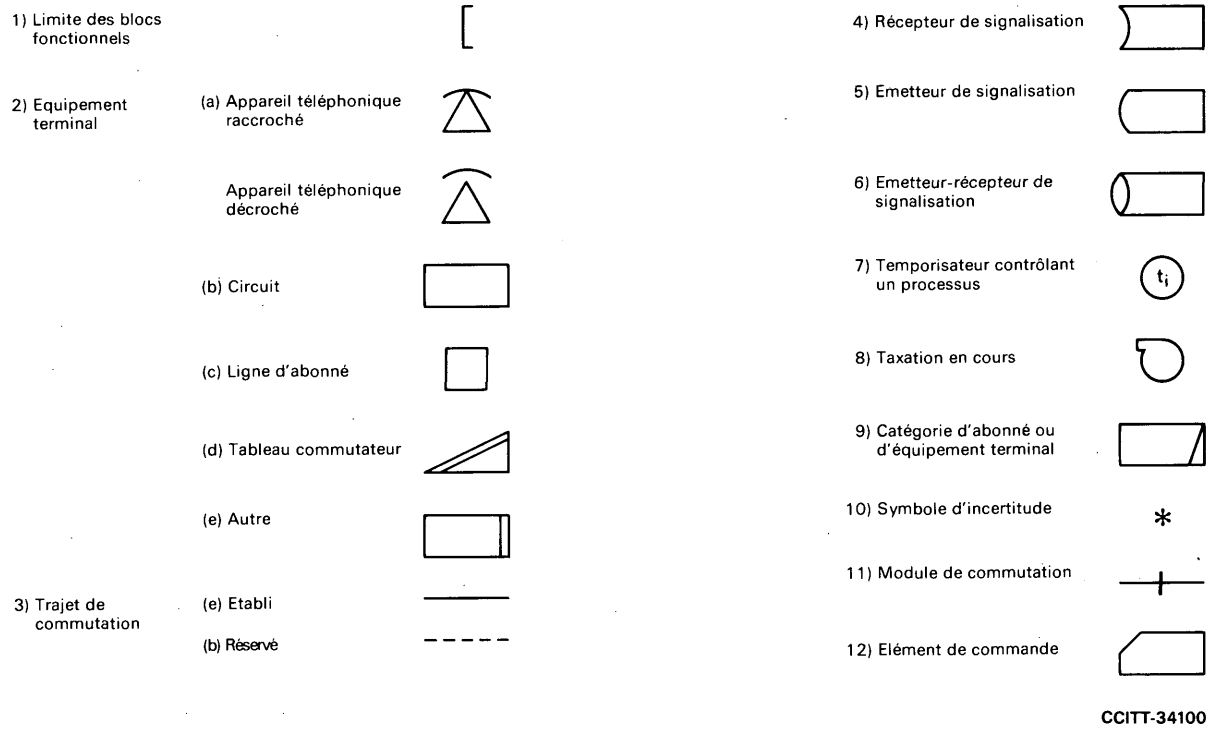


FIGURE 31/Z.103

Symboles recommandés pour l'ensemble de base des concepts d'éléments graphiques

6.4 *Conventions et interprétations spéciales utilisées dans l'extension du LDS/GR dans le sens des états*

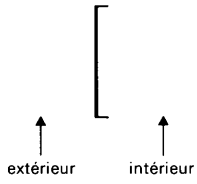
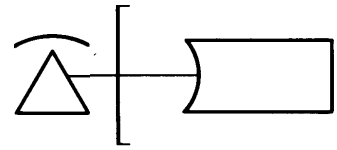
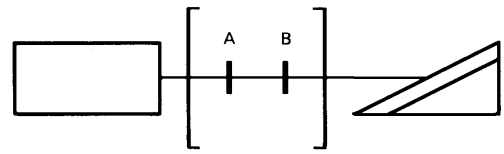
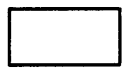
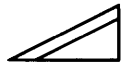
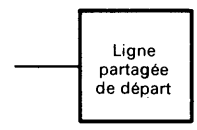
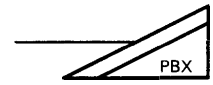
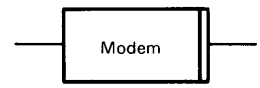
Dans ce paragraphe sont définies un certain nombre de conventions et interprétations spéciales relatives aux extensions du *LDS/GR* dans le sens des états, à savoir:

- L'interprétation spéciale à donner aux *diagrammes de processus* selon la RÈGLE ESSENTIELLE (règle 4 du § 6.2).
- La position unique des *éléments graphiques* (ou des *éléments graphiques* avec texte d'accompagnement) à l'intérieur de l'*illustration de l'état*, qui est une condition requise lorsque l'on utilise des *éléments graphiques* (règle 5 du § 6.2).
- L'interprétation spéciale à donner aux *variables* représentées par des *éléments graphiques externes* et par un texte d'accompagnement externe, en tant que mandataires d'autres *variables* associées à d'autres *processus*.

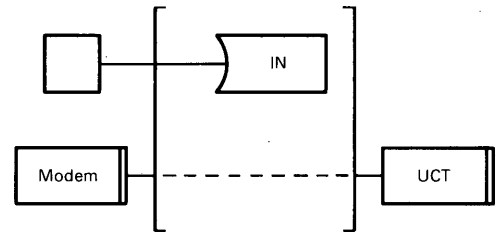
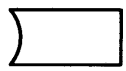
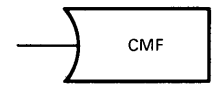
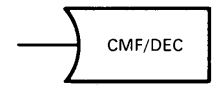
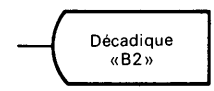
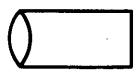
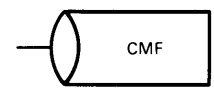
ANNEXE A

(à la Recommandation Z.103)

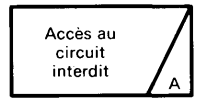
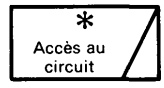
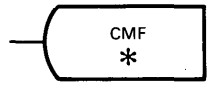
Exemples d'utilisation de l'ensemble de base des éléments graphiques

N°	Elément graphique	Commentaires	Exemples
1.	*Limites du bloc fonctionnel* (BF)  	Sert à distinguer les éléments à l'intérieur et à l'extérieur des limites du bloc fonctionnel. Seul les états des éléments se trouvant à l'intérieur de ces limites peuvent être modifiés directement par ce processus.	1.1 Combiné se trouvant à l'extérieur des limites du bloc fonctionnel, raccordé à un récepteur de chiffres se trouvant à l'intérieur de ces limites.  1.2 Circuit se trouvant à l'extérieur des limites du bloc fonctionnel, connecté par l'intermédiaire d'un dispositif de commutation à deux étages à un commutateur se trouvant à l'extérieur des limites du bloc fonctionnel 
2.	*Equipement terminal* a) Appareil téléphonique raccroché  décroché  b) Circuit  c) Ligne d'abonné [excepté a)]  d) Tableau commutateur manuel  e) Autre élément 	Il peut être utile de représenter l'équipement terminal extérieur aux limites du bloc fonctionnel (par exemple, appareil téléphonique et équipement de tableau commutateur) pour assurer une meilleure compréhension des opérations de traitement.	2.1 A raccroché $\overline{h}$  2.2 B décroché h_B  2.3 Joncteur décadique d'arrivée (d'un centre à commutation spatiale)  2.4 Ligne d'abonné de départ vers une ligne partagée  2.5 Tableau commutateur privé (PBX)  2.6 Modem 

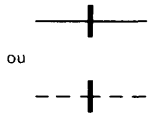
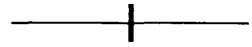
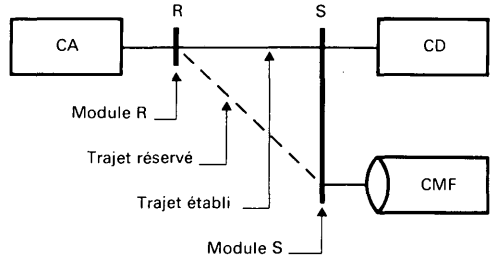
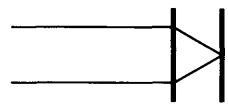
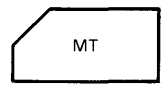
CCITT-20880

N°	Elément graphique	Commentaires	Exemples
3.	*Trajet de commutation* a) établi ——— b) réservé - - - - - -	Pour représenter la connexion des terminaux et/ou des dispositifs de signalisation intervenant dans le processus.	3.1 Ligne d'abonné connectée à un récepteur de chiffres à impulsion de numérotation (IN) et à un modem avec trajet réservé vers une unité centrale de traitement (UCT) 
4.	*Récepteur de signalisation* 	Pour spécifier un processus de réception de signaux et pour indiquer la nature des signaux reçus, en particulier de ceux qui traversent les limites du bloc fonctionnel.	4.1 Récepteur de signalisation de code multifréquence (CMF)  4.2 Récepteur de signalisation CMF/décadique 
5.	*Emetteur de signalisation* 	Pour spécifier un processus d'émission de signaux et pour indiquer la nature des signaux émis, en particulier de ceux qui doivent traverser les limites du bloc fonctionnel.	5.1 Emetteur de signalisation décadique avec envoi d'un signal B2 vers l'arrière 
6.	*Emetteur-récepteur de signalisation* 	Combine les fonctions d'émetteur et de récepteur de signalisation.	6.1 Emetteur-récepteur CMF 
7.	*Temporisateur contrôlant un processus* 	Les temporisateurs influent sur les états suivants du processus. ***Remarque*** **– Le symbole d'entrée correspondant, indiquant l'expiration du temps de fonctionnement du temporisateur, peut être indiqué par $\bar{t}_i$.**	7.1 Temporisateur t_3 en cours  7.2 Temporisateur générique t_s en cours  où s = 1, 2, ... n pour définir les différentes tonalités de service

CCITT-20890

N°	Elément graphique	Commentaires	Exemples
8.	*Taxation en cours (avec indication de l'abonné qui doit payé la taxe)* 	Les principes de taxation intéressent l'Administration, le constructeur et l'usager.	8.1 La taxation de l'abonné A est en cours 
9.	*Catégorie d'abonné ou de terminal (et information d'identité)* 	Tout changement dans la catégorie d'abonné ou de terminal peut, pour chaque abonné participant à une communication conférence, affecter le déroulement du processus.	9.1 L'abonné A n'a pas accès au circuit  9.2 L'abonné C a la catégorie d'origine N° 2 
10.	*Symbole d'incertitude* 	Solution de remplacement pour une information délibérément non définie qui est indiquée sans ambiguïté dans d'autres illustrations d'état. Dans certains cas, on peut effectuer sans risque la fusion de deux ou plusieurs états en un seul, ce qui facilite considérablement la compréhension du diagramme.	10.1 Combiné raccroché ou décroché  10.2 Catégorie de l'abonné: interdiction ou non de l'accès au circuit à cet état du processus  10.3 Un signal CMF non défini est émis dans cet état 

CCITT-20900

N°	Elément graphique	Commentaires	Exemples
11.	*Module de commutation*  	Pour indiquer quels modules de commutation interviennent dans le processus. *Remarque* – La ligne horizontale est l'élément graphique représentant un trajet de commutation qui peut être établi ou réservé. La ligne verticale peut être utilisée pour représenter soit un module de commutation complet (lorsque l'indication de la structure interne du module n'est pas demandée) soit l'un des étages de commutation dans un module de commutation.	11.1 Trajet établi en passant par un module de commutation. RCL = Réseau de connexion de ligne  11.2 Trajets établis et réservés en passant par deux modules  CA – Circuit d'arrivée CD – Circuit de départ CMF – Code multifréquence *Remarque* – Dans cet exemple, CA est connecté à CD mais CA n'est pas connecté à l'émetteur/récepteur CMF 11.3 Trajet établi en passant par un module de commutation à trois étages RSN  11.4 Trajet réservé en passant par un module de commutation à trois étages ABC  11.5 Trajet établi en passant par un réseau plié 
12.	*Elément de commande affecté à un processus* 	Pour indiquer quel équipement de commande intervient dans le processus (notamment des modules qui doivent avoir une dimension). Ce symbole peut servir à indiquer que des éléments logiciels particuliers ont été affectés au processus.	12.1 Mémoire-tampon (MT) d'enregistreur 

CCITT-20910

ANNEXE B

(à la Recommandation Z.103)

Critères de sélection pour les éléments graphiques

B.1 *Considérations générales*

Le choix des symboles pour les éléments graphiques repose sur les considérations et critères généraux de sélection ci-après, auxquels il convient de se reporter avant d'élaborer de nouveaux symboles pour de plus vastes applications du LDS.

B.2 *Lecteurs typiques*

On peut prévoir que les diagrammes LDS utilisant des éléments graphiques seront lus aussi bien par des spécialistes que par des non-spécialistes des domaines suivants: commercialisation de nouveaux services; spécification de nouveaux services; mise au point de matériels et de logiciels à partir d'une spécification; gestion de projets; exploitation et maintenance de centraux; ingénierie du trafic; formation et perfectionnement professionnels en téléphonie. On peut prévoir aussi que les diagrammes LDS constitueront une documentation qui servira à la fois:

a) aux Administrations et aux constructeurs;
b) aux différents départements de ces entités;
c) de documentation sur les centraux téléphoniques;
d) pour l'élaboration de manuels de formation.

Il n'est pas prévu que les diagrammes LDS soient lus directement (en tant que diagrammes) par des machines. Il est prévu, en revanche, que la forme LDS/PR du LDS (comprenant des éléments graphiques) sera lue par des machines qui dessineront des diagrammes [voir b) et c) du § B.3].

B.3 *Méthodes de dessin typique*

Il est prévu que les diagrammes LDS utilisant des éléments graphiques seront en principe dessinés par des techniciens, notamment par des dessinateurs, selon un des procédés suivants:

a) à la main, à l'aide d'un gabarit, et/ou
b) par la présentation électronique du diagramme sur un terminal à affichage optique d'éléments graphiques, et/ou
c) par l'utilisation d'un traceur à commande électronique.

B.4 *Méthodes de reproduction*

Les méthodes de reproduction typiques seront, pense-t-on, les suivantes:

a) des lignes colorées ou du tirage bleu en papier au ferroprussiate, comme dans le dessin industriel classique;
b) la photocopie par des machines de bureau, y compris la photo-réduction;
c) la photo-impression en général.

B.5 *Facilité de reproduction*

Afin de permettre une reproduction commode des diagrammes LDS à l'aide des méthodes de reproduction fondées sur les lignes colorées ou le tirage bleu en papier au ferroprussiate, sur la photocopie ou la photoimpression, les symboles des éléments graphiques doivent être constitués de lignes nettes, sans dégradation.

B.6 *Facilité de dessin*

Les critères suivants sont fondés sur l'hypothèse que le dessin initial sera établi à la main à l'aide d'un gabarit et que la deuxième étape consistera à présenter le diagramme sur un écran de visualisation électronique et à dessiner le diagramme à l'aide d'un stylet à commande électronique:

a) chaque symbole d'élément graphique doit être facile à dessiner à l'aide d'un stylet ou d'un crayon, soit directement à la main, soit au moyen d'un gabarit;
b) tous les dessins des symboles d'éléments graphiques doivent être exécutés avec une épaisseur de ligne identique;
c) pour créer les symboles des éléments graphiques, il faut faire une synthèse de lignes et de courbes géométriques très simples afin de permettre une production aisée des symboles par un procédé électronique.

B.7 *Intelligibilité*

Ce critère est le plus important; en effet, la documentation LDS a pour particularité que les lecteurs sont beaucoup plus nombreux que les dessinateurs (ou que les auteurs). Ce critère se subdivise en plusieurs sous-critères concernant les divers symboles d'éléments graphiques:

a) *Pertinence* – La forme de chaque symbole doit être appropriée au concept que ce symbole représente.

b) *Caractère distinctif* – Lors du choix d'un ensemble de symboles de base, il convient de faire en sorte que chaque symbole puisse être aisément identifié par rapport aux autres symboles de cet ensemble.

c) *Affinité* – Les formes des éléments graphiques représentant des fonctions différentes mais connexes, par exemple, les récepteurs et les émetteurs, doivent être reliées les unes aux autres d'une manière évidente.

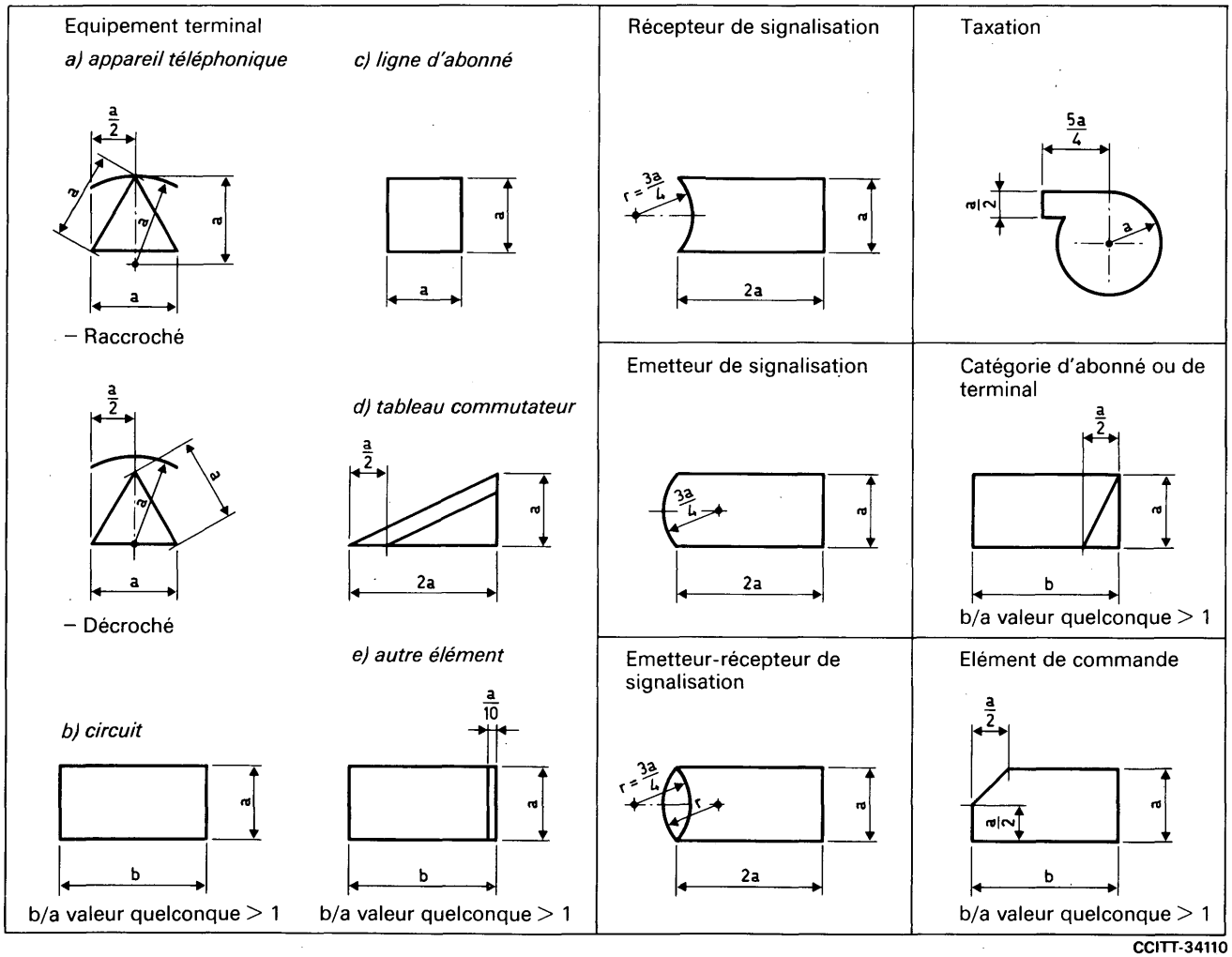
d) *Texte abrégé associé aux symboles* – Dans certains cas, il est prévu qu'un texte abrégé sera associé à un élément graphique, afin d'indiquer la catégorie de cet élément; par exemple, les lettres CMF pourront accompagner le symbole correspondant à un récepteur pour indiquer que celui-ci peut recevoir des signaux du code multifréquence. En pareil cas, les éléments graphiques doivent comporter un espace fermé dans lequel pourront être inscrits un très petit nombre de caractères alphanumériques.

e) *Limitation du nombre des symboles* – Le nombre total de symboles doit être réduit à un minimum afin que la méthode de représentation graphique puisse être aisément apprise.

ANNEXE C

(à la Recommandation Z.103)

Proportions recommandées pour l'ensemble de base des éléments graphiques



DONNÉES DANS LE LDS

1 Introduction

Cette Recommandation définit le concept de *données* dans le LDS; la terminologie des *données* LDS, l'utilisation des *types de données prédéfinis* et le moyen de définir de *nouveaux types de données.*

Les principales apparitions de *données* dans le LDS sont représentées par les *définitions de types de données*, les *expressions*, l'application d'*opérateurs*, de *variables*, de *valeurs*, de *constantes* et de *littéraux.*

Définitions de types de données

Dans le LDS, les *données* se rapportent principalement à des *types de données.* Un *type de données* définit un ensemble de *valeurs*, un ensemble d'*opérateurs* qui peuvent être appliqués à ces *valeurs*, et un ensemble d'*axiomes* définissant le comportement au moment où ces *opérateurs* sont appliqués aux *valeurs.* Les *valeurs*, *opérateurs* et *axiomes* définissent collectivement les *propriétés* des *types de données.* Ces *propriétés* sont définies par des *définitions* de *types de données.*

Le LDS permet de définir tout *type de données* nécessaire, y compris les mécanismes de *structuration* (types composites), à condition seulement que cette définition soit formellement spécifiée, contrairement aux langages de programmation dans lesquels les considérations relatives à l'implémentation exigent que l'ensemble des *types de données* disponibles et, en particulier, l'ensemble des mécanismes de structuration fournis (*tableau*, *structure*, etc.) soient limités.

Expressions et opérateurs

Les *expressions* permettent de manipuler les *valeurs* (en appliquant des *opérateurs* appropriés), pour obtenir de nouvelles *valeurs.*

Variables

Les *variables* sont des objets qui peuvent être associés à une *valeur* au moyen d'une *affectation explicite ou implicite.* Lorsque la *variable* est *accédée*, la *valeur* est retournée.

Valeurs, constantes et littéraux

Tous les *types de données* ont au moins une *valeur.* Pour la plupart des *types de données*, il existe des formes *littérales* (syntaxiques) servant à indiquer les *valeurs constantes* du *type de données* (par exemple, *les entiers*). On dit que les *types de données* pour lesquels il existe des *littéraux* servant à indiquer des *valeurs* sont des *constantes dénotables.* Plusieurs *littéraux* peuvent indiquer la même *valeur constante* (par exemple, 12 et H'C indiquent tous les deux la même constante *entière*), et la même indication littérale peut être utilisée pour plusieurs types de données. Certains types n'ont pas de constantes dénotables. Par exemple, les valeurs du type de données pile ne peuvent être générées que par l'application d'*opérateurs* qui donnent des *valeurs* de pile.

Dans un langage de spécification, il faut pouvoir décrire formellement les *types de données* du point de vue de leur comportement, plutôt que de les composer à partir de primitives fournies, comme dans un langage de programmation. Cette dernière méthode implique invariablement une implémentation particulière du *type de données* et restreint donc la liberté du réalisateur au niveau du choix des représentations appropriées du *type de données.* La méthode du LDS permet n'importe quelle implémentation à condition qu'elle soit possible et correcte du point de vue du LDS.

Dans le LDS, tous les *types de données* sont des *types abstraits.* On trouvera au § 5 les exemples de ces types dans lesquels sont définis les *types de données prédéfinis* du langage.

Bien que tous les *types de données* soient abstraits et que les *types de données prédéfinis* puissent être redéfinis par l'utilisateur, on s'est efforcé de fournir dans le LDS des *types de données prédéfinis* qui soient courants tant du point de vue de leur comportement que de leur syntaxe. Les noms des types de données en question sont les suivants:

Tableau (Array); Booléen (Boolean); Caractère (Character); chaîne de caractères (Charsting); Durée (Duration); Entier (Integer); Entier naturel (Natural); Mode ensembliste (Powerset); PId (PId); Réel (Real); Chaîne (String); Temps (Time) et Temporisateur (Timer).

Des types composites peuvent être formés en utilisant des *types de données structurés (Struct).*

1.1 *Formalisme de données du LDS*

Dans le LDS, les *données* sont modelées par une *algèbre de type*. Une *algèbre de type* est un ensemble de *domaines*, un *domaine désigné* et un ensemble de fonctions établissant des correspondances entre les *domaines*. Chaque *domaine* regroupe toutes les *valeurs* possibles pour un *type de données*. Le *domaine désigné* est le *type de données* que l'on décrit actuellement. Les fonctions représentent les *opérations du type de données*.

Le *domaine* et les *opérations*, ainsi que le comportement spécifié par les *axiomes* du *type de données*, constituent les *propriétés* du *type de données*.

L'introduction d'un *syntype* crée un sous-ensemble des *valeurs* d'un *type* déjà défini. L'introduction d'un *newtype* crée un *nouveau type de données distinct*, avec des *propriétés héritées de l'ascendant*, mais avec différents *identificateurs* pour ces *propriétés*. Dans la syntaxe concrète, il n'est pas nécessaire que ces *noms* soient distincts, et cette ambiguïté doit être éliminée par le contexte.

Un *générateur de type* est une *description de type* incomplète; avant qu'il suppose l'état d'un *type de données*, il doit être *instancié* en fournissant cette information manquante.

Les fonctions qui établissent une correspondance entre les domaines d'un type de données sont divisées normalement en deux catégories. Les fonctions génératrices, qui établissent une correspondance avec le *domaine désigné* et produisent ainsi les *valeurs* (éventuellement nouvelles) du *type de données*. Toutes les autres fonctions sont des fonctions sémantiques et attribuent une signification au *type* en établissant une correspondance avec d'autres *types* définis. Les fonctions sémantiques comprennent les prédicats qui établissent une correspondance avec le *domaine Booléen*.

2 Modèle de langage commun

2.1 *Généralités*

Dans la Recommandation Z.104, on utilise les concepts de *variables*, des *types de données prédéfinis*, des *valeurs*, des *expressions* et des *expressions* informelles. Cette Recommandation définit rigoureusement des *variables*, des *types de données prédéfinis*, des *valeurs* et des *expressions* ainsi que des adjonctions aux Recommandations Z.101, Z.102 et Z.103 qui permettent l'introduction de nouveaux *types de données*.

2.2 *Syntaxe abstraite*

Cette syntaxe abstraite élargit celle définie par les Recommandations Z.101, Z.102 et Z.103.

Définition de données

Une *définition de données* est une *définition de type de données* ou une *définition de synonyme*.

Définition de système

Une *définition de système* peut contenir des *définitions de données*.

Définition de bloc

Une *définition de bloc* peut contenir des *définitions de données*.

Définition de bloc de partie interne

Une *définition de bloc de partie interne* peut contenir des *définitions de type de données*.

Définition de sous-structure de canal

Une *définition de sous-structure de canal* peut contenir des *définitions de données*.

Définition de processus

Une *définition de processus* peut contenir des *définitions de données* et des *définitions de variables*.

Définition de procédure

Une *définition de procédure* peut contenir des *définitions de données* et des *définitions de variables*.

Nœud d'entrée

Une *variable* mentionnée dans un *nœud d'entrée* doit être définie dans une *définition de variable* et doit avoir le même *type de données* que le *type de données* correspondant dans la *définition de signal*.

Définition de type de données

Une *définition de type de données* contient un *nom de type* et une description de *newtype* ou une description de *syntype.*

Description de newtype

Une description de *newtype* contient un ensemble éventuellement vide de *noms de valeur*, un ensemble d'une ou de plusieurs introductions *d'opérateur* et un ensemble éventuellement vide d'*axiomes de type de données.*

Tous les *noms* de valeur de la *description de type de données* doivent être uniques à l'intérieur du *type de données.*

Tous les *noms d'opérateur* de la *description de type de données* doivent être mutuellement exclusifs.

Tous les *noms de type de données* dans le même contexte doivent être uniques.

Introduction d'opérateur

Une introduction d'*opérateur* introduit soit un des *opérateurs universels* qu'il est permis d'introduire avec un *type de données* quelconque, soit un *nom d'opérateur* avec le *typage d'opérateur.*

Opérateur

Un *opérateur* est soit un *opérateur universel* avec un *identificateur de type de données* soit un *identificateur d'opérateur* défini par l'utilisateur avec un *identificateur de type de données.* Dans les deux cas, l'*identificateur de type de données* permet de donner la *définition de type de données* qui définit l'*opérateur* à établir.

Typage d'opérateur

Un *typage d'opérateur* contient la liste des *identificateurs de type de données* des *paramètres* de l'*opérateur* et *l'identificateur de type de données* du *résultat* de l'application de l'*opérateur.*

Au moins une des *identités de type de données* de la liste, ou le résultat, doit être celle du *type de données* que l'on définit.

Le *type* de résultat ne peut être un *syntype.*

Axiomes

Des *axiomes* sont des énoncés de vérité à utiliser, dans toutes les situations, pour le *type* à définir, et spécifient donc le comportement des types.

Instruction d'affectation

Une *instruction d'affectation* contient un *opérateur d'affectation*, une *identité de variable* et une *expression.* L'*opérateur d'affectation* est soit l'*opérateur universel d'affectation qualifié* par l'*identité de type de données* du *type de données* de la *variable* soit l'*opérateur insérer défini par l'utilisateur.*

Description de syntype

Une description de *syntype* contient un *nom de syntype, l'identité de type de données de l'ascendant* et l'ensemble des *identités de valeur* du *type de données de l'ascendant* qui sont valables pour le *type de données* synonyme.

Expression

Une *expression* est soit un *primaire* soit une *opération.*

Opération

Une *opération* contient un *opérateur* et une liste d'une ou de plusieurs *expressions.* Il y a autant d'*expressions* dans l'*opération* que de *types de données* définis dans les *paramètres* de l'*opérateur.*

Primaire

Un primaire est un des éléments suivants:

- une *identité de synonyme,*
- une *identité de valeur,*
- une *identité de variable,* ou
- une *expression conditionnelle.*

Expression conditionnelle

Une *expression conditionnelle* est une *expression booléenne* et une liste de deux *expressions* du même *type de données.*

Définition de variable

Une *définition de variable* se compose d'une liste de *noms de variable* et d'un *identificateur de type de données.*

Opération universelle

Un *opérateur universel* est soit un *opérateur de variable* soit un *comparateur.*

Un *opérateur de variable* est *déclaration, affectation* ou *accès.* On utilise la *déclaration* pour *déclarer* les *variables,* l'*affectation* pour *affecter* les *variables* et l'*accès* chaque fois qu'une *identité de variable* est interprétée comme une *valeur.*

Un *comparateur* est soit un des *opérateurs d'ordre* soit un *opérateur d'égalité.* Un *opérateur d'ordre* est plus petit que ou plus grand que.

Tous les *opérateurs universels* comprennent le *type de données* auquel ils se rapportent comme un *qualificateur* de sorte que différents *types de données* introduisent différents opérateurs. Par exemple, les deux *types de données* carré et cube introduisent deux *identités d'opérateurs* pour l'*affectation*: carré!assign et cube!assign.

Pour un *type de données* D, le *typage de l'opérateur* pour les comparateurs est la suivante:

D, D $->$ Boolean

Pour un *type de données* D, l'*opérateur d'affectation* a besoin d'une *identité de variable* de *type de données* D et d'une *valeur de type de données* D.

Pour un *type de données* D, l'*opérateur d'accès* a besoin d'un *identificateur* de *variable* de *type de données* D et donne une *valeur de type de données* D.

Pour un *type de données* D, l'*opérateur de déclaration* a besoin d'un *identificateur de variable.*

Définition de synonyme

Une définition de *synonyme* contient un *nom de synonyme* et une *expression constante.*

Expression constante

Une *expression constante* est soit une *valeur constante* soit une *opération* dont tous les paramètres sont des *expressions constantes.*

Valeur constante

Une *valeur constante* est soit un *identificateur de valeur* soit un *identificateur de synonyme.* Un *synonyme* ne peut pas être défini de façon récursive.

2.3 *Règles d'interprétation*

2.3.1 *Processus*

L'*instantiation d'un processus* a lieu avant que le *nœud de départ* soit *interprété* et fait qu'une *opération de déclaration* est appliquée à chaque *nom de variable* qui apparaît dans une *définition de variable* dans la *définition de processus.* L'*opérateur déclare* a pour effet de transformer les *variables* déclarées en association entre les *identités de variable* et la *valeur initiale* (qui sera la *valeur indéfinie,* à moins qu'elle n'ait été spécifiée dans les *axiomes* pour les *types de données* des *variables*). Les *identités de variable* déclarées lors de l'instantiation d'un *processus* contiennent le *nom de variable,* l'*identité d'instance de processus* et *l'identité de type de données* de la *définition de variable.*

2.3.2 *Nœud de départ de procédure*

Demander une *procédure* fait que les *variables* définies à l'intérieur de la *procédure* sont *créées* pour une *procédure* de la même façon que l'*instantiation* d'un *processus.*

2.3.3 *Graphe de processus*

L'*interprétation* d'un *nœud de sortie* fait que chacune des *expressions* du *nœud de sortie* est *interprétée* dans la séquence spécifiée et que les valeurs résultantes sont *affectées* à des *variables implicites anonymes* qui sont associées au *signal.* On considère que ces *variables* sont *déclarées* lorsque le *nœud de sortie* est *interprété.* Chacune de ces *variables* a le *type de données* ou le *type de données synonyme* associé à la position correspondante dans les *définitions de signal.* Les *valeurs* qui sont *affectées* à ces *variables* sont les *valeurs* transmises par le *signal.*

L'*interprétation* d'un *nœud de décision* fait que l'on interprète l'*expression* contenue dans le *nœud de décision* suivie du choix de l'*arc* qui est associé à la *valeur* donnée par l'*expression.*

Demande de création

L'*interprétation* d'un *nœud de demande de création* fait que chacune des *expressions* du *nœud de demande de création* est *interprétée* dans la séquence spécifiée.

L'*instantiation* du *processus* qui se *crée* a ensuite lieu avec la *déclaration* du *paramètre formel* du *processus* et l'*affectation* de la *valeur* résultante correspondante de chaque *expression* dans le *nœud de demande de création* à chaque *paramètre formel.* Le *processus créé* se déroule alors séparément des autres *processus* mais en même temps.

Nœud d'appel

Un *nœud d'appel* fait que les *expressions* utilisées comme *paramètres formels* attribués avec *IN* sont *interprétées* avant l'*interprétation* du *nœud de départ* de la *procédure.* Chacune des *expressions affecte* sa valeur au *paramètre réel* correspondant.

2.3.4 *Procédure*

Un *paramètre formel* ayant l'attribut *IN/OUT* est *interprété* comme l'*identificateur de variable* du *paramètre réel* correspondant dans le contexte de l'*appel de procédure.* Un *paramètre formel* ayant l'attribut *SIGNAL* est *interprété* comme l'*identificateur de signal* du *paramètre réel* correspondant dans le cadre de l'*appel de procédure.*

2.3.5 *Instruction d'affectation*

L'*instruction d'affectation* est *interprétée* comme la combinaison de l'ancienne *valeur* de la *variable* avec la *valeur* d'une *expression* et comme le lien de l'*identité* de *variable* à cette nouvelle *valeur.*

L'*opérateur d'affectation* détermine les règles permettant de combiner l'ancienne *valeur* de la *variable* avec la *valeur* de l'*expression.* Ces règles sont déterminées par l'utilisation de l'*opérateur d'affectation* dans les *axiomes de type de données* du *type de données.* Si l'*opérateur d'affectation* est l'*opérateur universel* pour l'*affectation,* alors la *variable* est *liée* à la *valeur* de l'*expression.*

La *valeur* de l'*expression* doit être une des *valeurs* du *type de données* de la *variable* de l'*opérateur d'affectation.* Pour l'*opérateur universel* d'*affectation,* le *type de données* du *paramètre* est celui de la *variable.*

2.3.6 *Expression et primaire*

Une *expression* est *interprétée* comme le *primaire* qui constitue l'*expression.* Le *primaire* est soit une *opération,* un *synonyme,* un *identificateur de valeur* ou un *identificateur de variable,* et il est ainsi interprété.

2.3.6.1 *Opération*

Une *opération* est *interprétée* comme une application de l'*opérateur* à la liste des *valeurs* obtenues en *interprétant* la liste des *expressions.* L'*interprétation* de l'*opération* est déterminée par l'utilisation de l'*opérateur* dans les *axiomes de type de données* du *type de données.*

2.3.6.2 *Identificateur de synonyme*

Un *identificateur de synonyme* est *interprété* comme l'*expression constante* définie dans la *définition de synonyme.* L'*expression constante* est *interprétée* de la même façon qu'une *expression.*

2.3.6.3 *Identificateur de valeur*

Un *identificateur de valeur* est *interprété* comme la *valeur* qu'il dénote.

La sémantique de la *valeur* qu'un *identificateur de valeur* dénote est déterminée par l'utilisation de l'*identificateur de valeur* dans les *axiomes de type de données* du *type de données.*

2.3.6.4 *Identificateur de variable*

Un *identificateur de variable* est *interprété* d'une des deux façons suivantes selon le contexte. Dans une *expression,* un *identificateur de variable* est *interprété* comme un *accès.* Dans le contexte d'une *instruction d'affectation,* un *identificateur de variable* est *interprété* comme une *variable,* qui est le *lien* entre l'*identificateur de variable* et une *valeur.* L'*accès* à une *variable* est *interprété* comme la *valeur* à laquelle la *variable* est *liée,* sauf que l'*accès* à la *valeur non définie* est *interprété* comme une *erreur.*

2.3.6.5 *Expression conditionnelle*

Une *expression conditionnelle* est *interprétée* comme la première ou la deuxième *expression* de la liste, selon que l'interprétation de l'*expression booléenne* donne *vrai* ou *faux.*

2.3.7 *Définitions de type de données*

Elles ne sont pas *interprétées.*

3 LDS/GR

La spécification normalisée du LDS du comportement des *tâches*, des *décisions*, etc. (c'est-à-dire la structure interne de ces *nœuds*) est la forme du LDS/PR. En conséquence, il n'y a pas de *syntaxe graphique* spécifique pour les données.

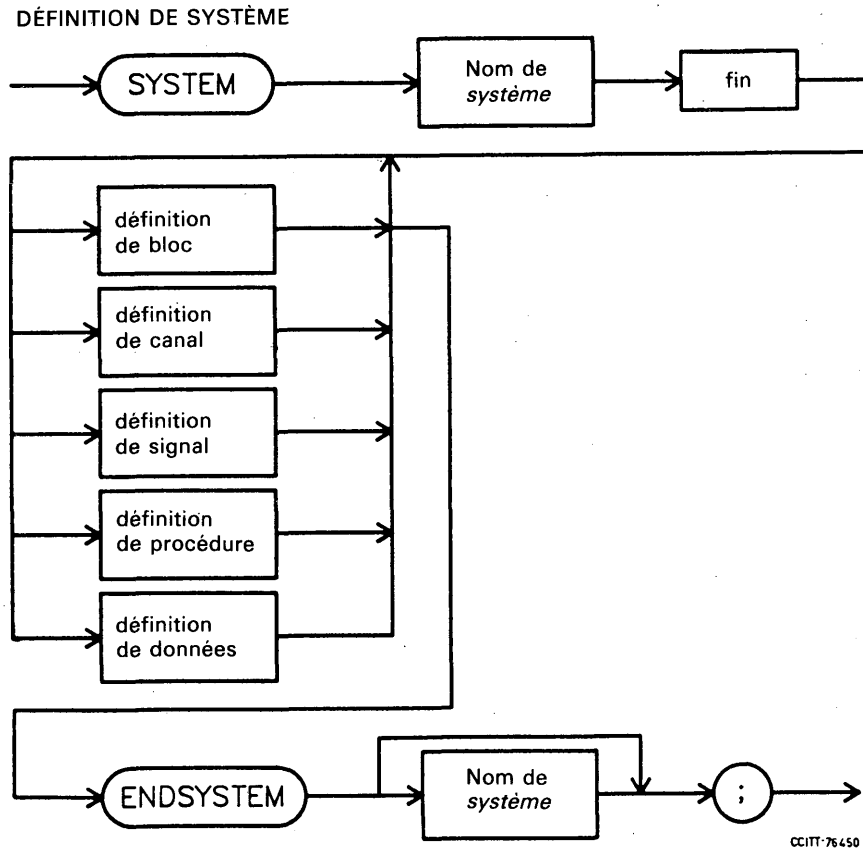
Lorsque les *définitions de données* sont incluses (pour des *types de données*, des *variables* ou des *synonymes*), elles doivent être définies ou référencées dans la partie supérieure du diagramme qui les définit.

4 LDS/PR

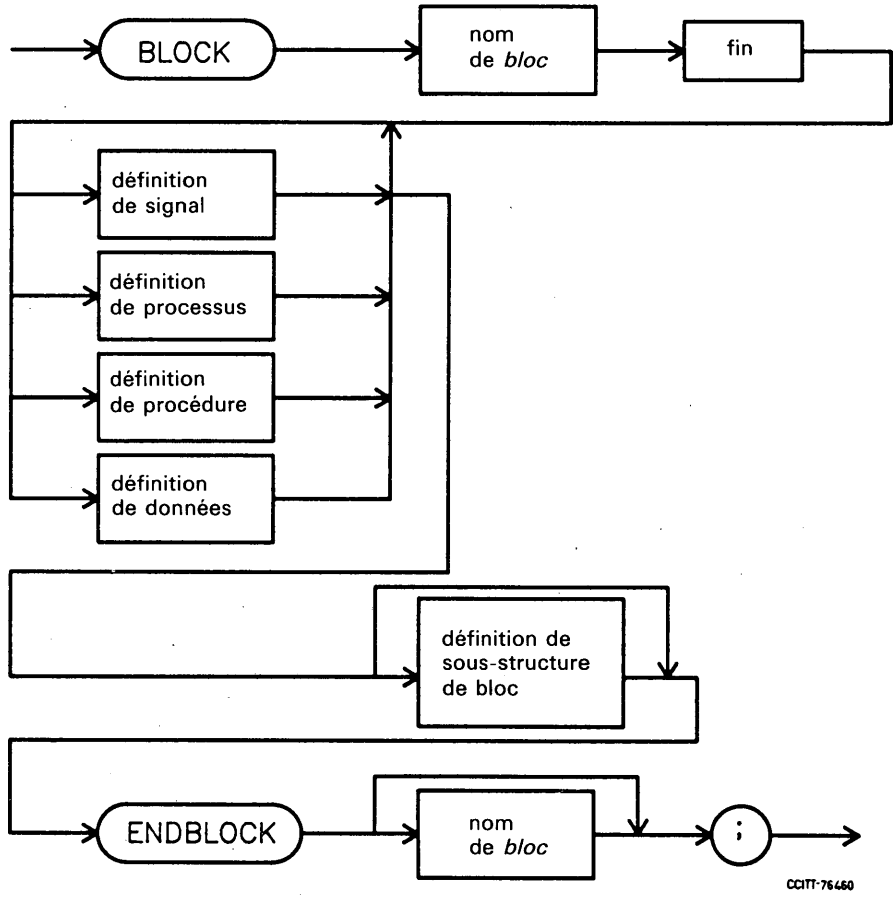
4.1 *Adjonction à la syntaxe*

Les définitions de données sont ajoutées à la syntaxe comme défini dans les Recommandations Z.101, Z.102 et Z.103.

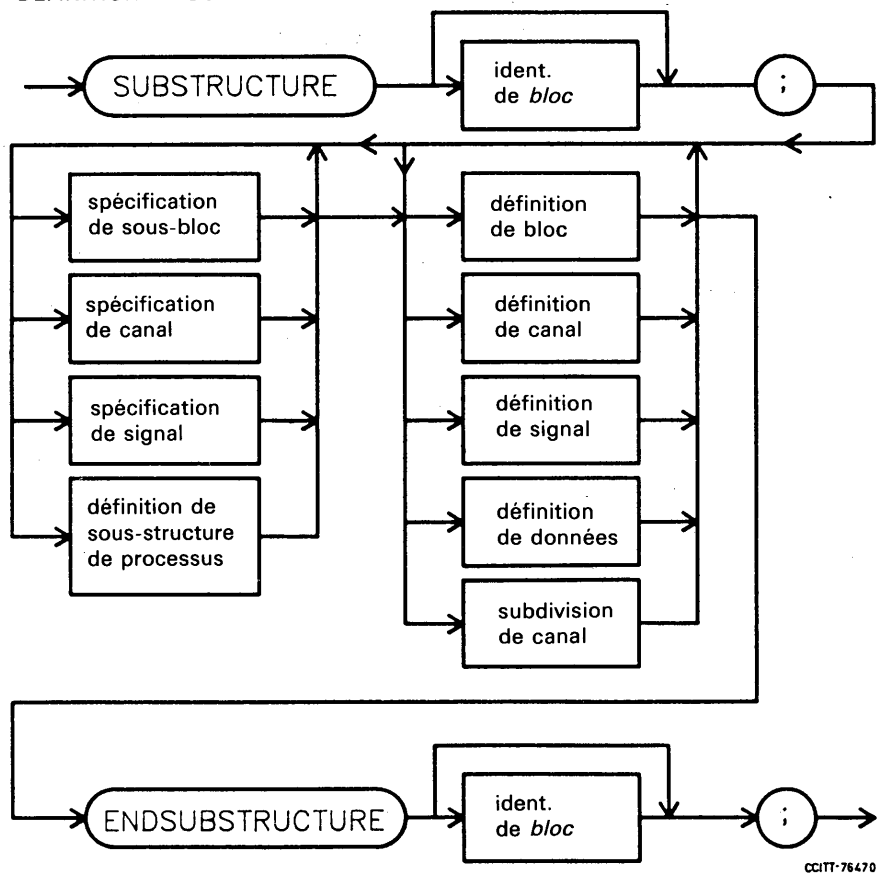
Les *définitions de données* peuvent apparaître lorsqu'une *définition de signal* apparaît dans une *définition de système*, une *définition de bloc*, une *définition de sous-structure de bloc* ou une *définition de sous-structure de canal.* Une *définition de données* peut aussi apparaître lorsqu'une *définition de variable* peut apparaître dans une *définition de processus* ou lorsqu'une *définition de variable* de *procédure* peut apparaître dans une *définition de procédure.*



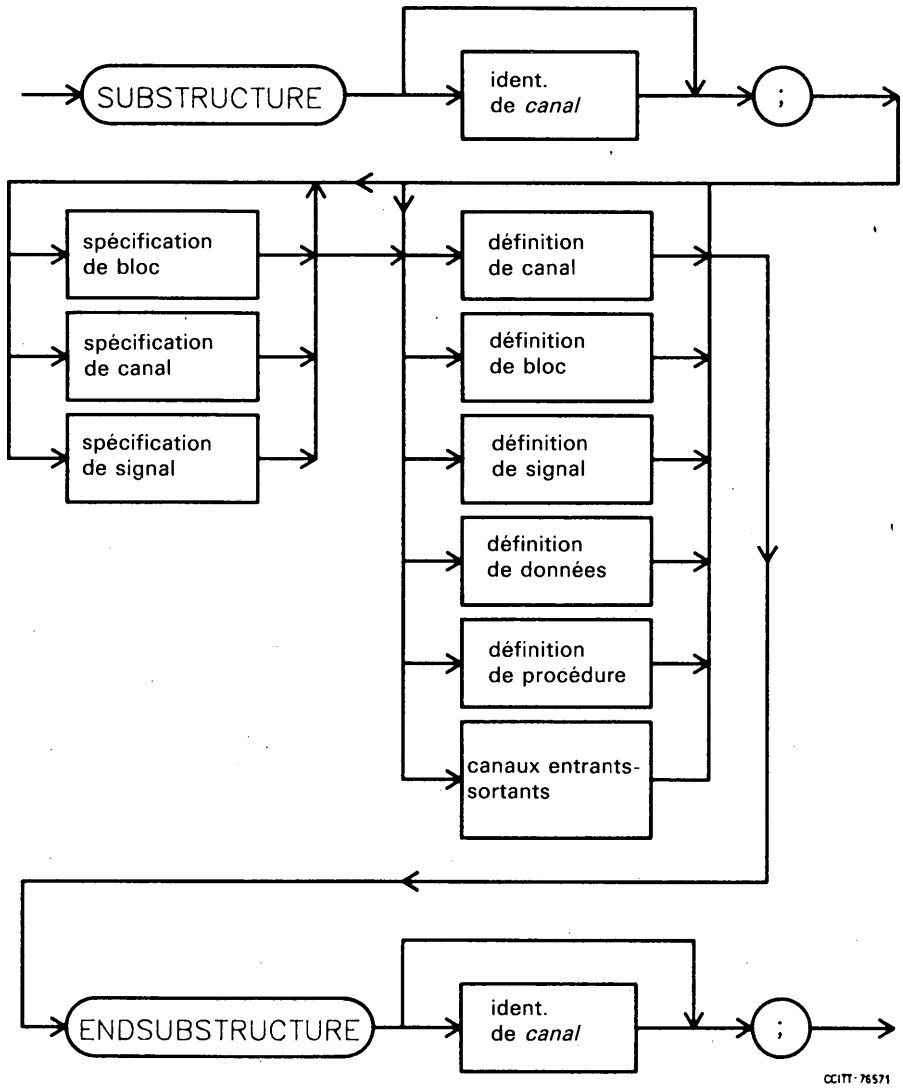
DÉFINITION DE BLOC

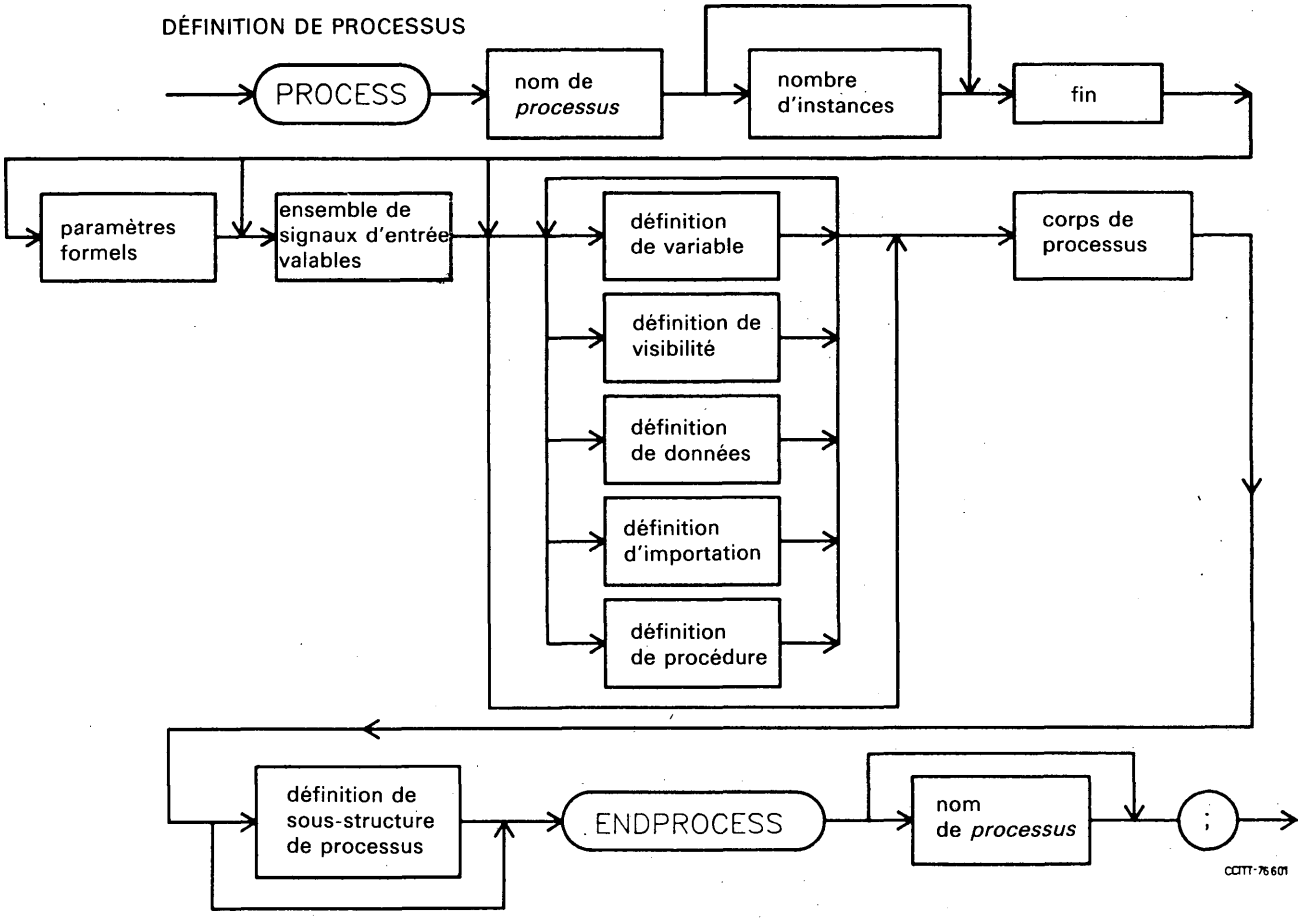


DÉFINITION DE SOUS-STRUCTURE DU BLOC



DÉFINITION DE SOUS-STRUCTURE DE CANAL



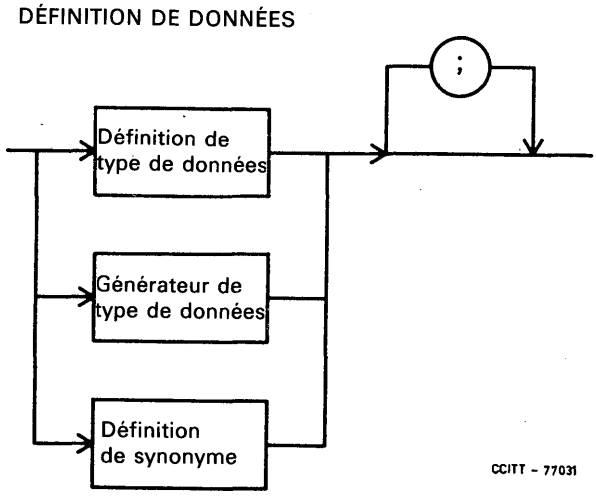


La syntaxe de *type de données* est étendue pour inclure les *identificateurs* de *type de données* définis par l'utilisateur.

La syntaxe pour les *instructions d'affectation* et les *expressions* est donnée. (Il convient de noter que les expressions et les instructions d'affectation sont mentionnées dans la Recommandation Z.101, mais elles ne sont pas définies.)

4.2 *Définition de données*

4.2.1 *Syntaxe*



4.2.2 *Sémantique*

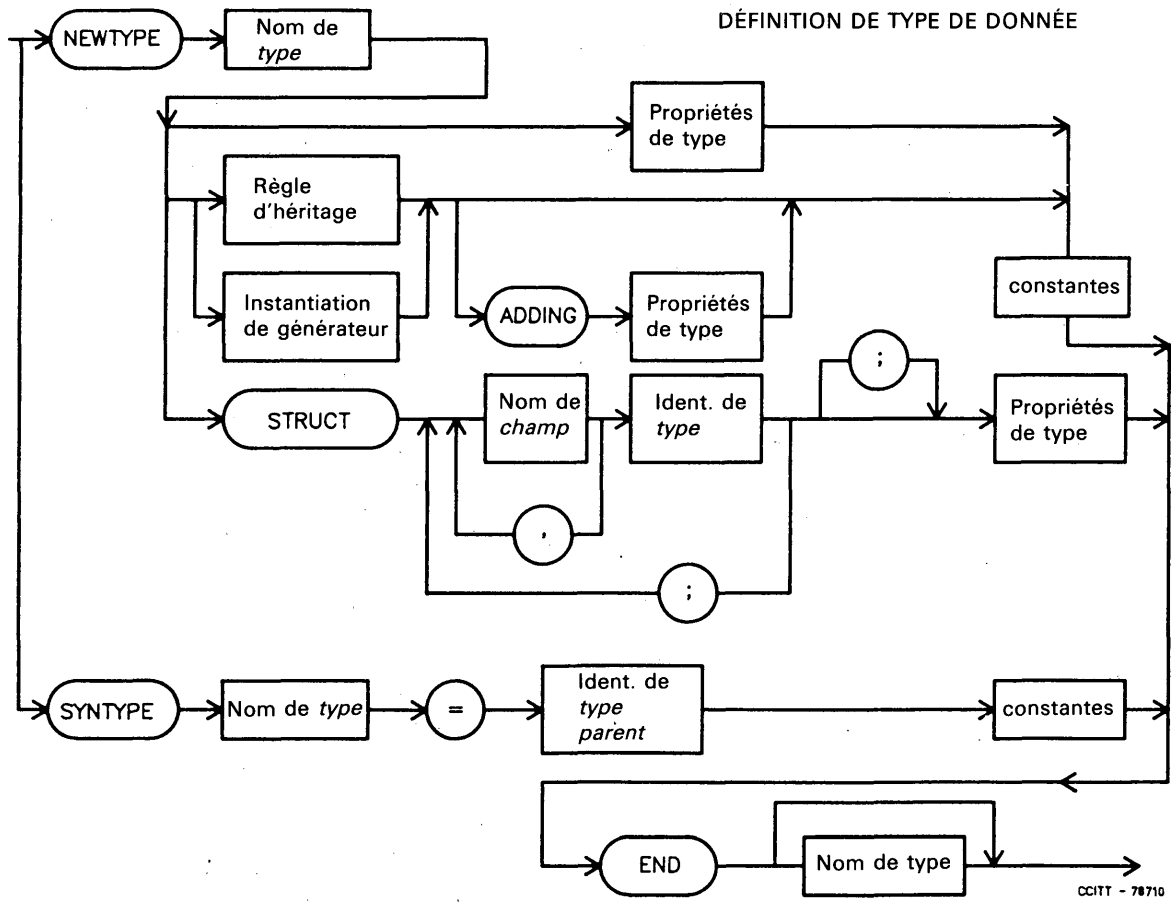
Une *définition de données* sert à introduire les *noms* et les *propriétés* des *types de données*, des *générateurs de type de données* ou des *synonymes*.

4.2.3 *Relation avec la syntaxe abstraite du LDS*

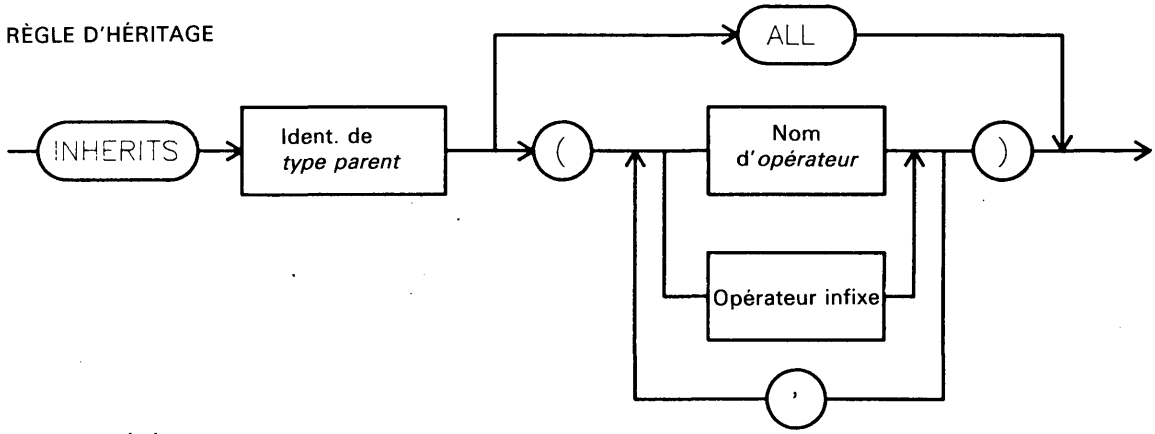
Une *définition de données* dans la LDS/PR représente une *définition de données* dans la *syntaxe abstraite*. Si la *définition de données* est un *générateur* de *type de données*, alors il n'y a pas de correspondance directe avec la *syntaxe abstraite*, car le *générateur de type de données* sert uniquement à définir un texte qui est considéré comme textuellement expansé par l'*instantiation du générateur*.

4.3 *Définition de type de données*

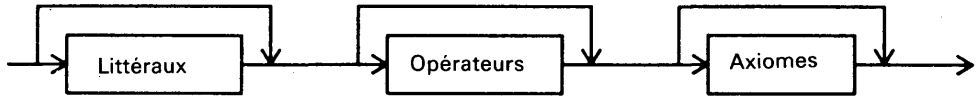
4.3.1 *Syntaxe*



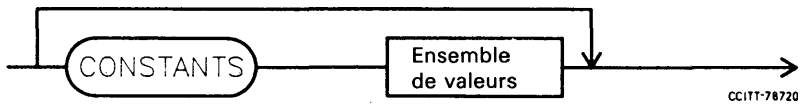
RÈGLE D'HÉRITAGE



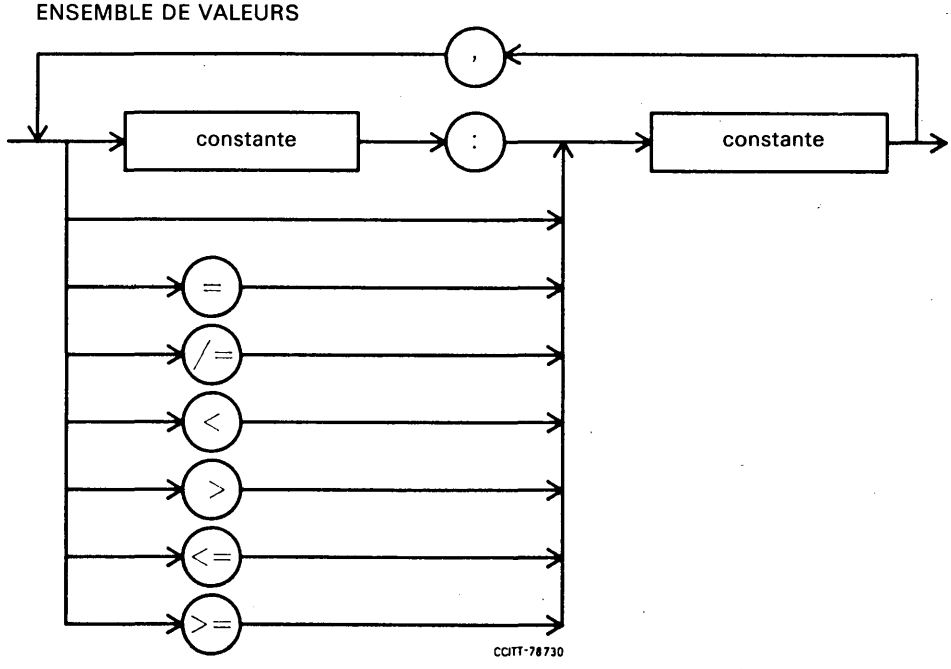
PROPRIÉTÉS DE TYPE

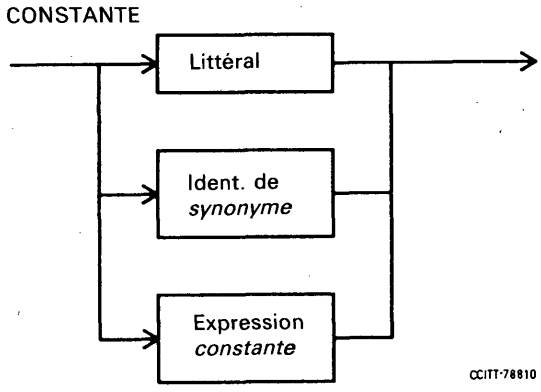


CONSTANTES



ENSEMBLE DE VALEURS





4.3.2 *Sémantique*

Le nom donné dans une *définition de type de données* est un *nom de type de données.*

4.3.2.1 *NEWTYPE*

Une *définition de type de données NEWTYPE* introduit un *nouveau type de données.*

Si aucune *règle d'héritage* n'est spécifiée, le *nouveau type de données* n'est fondé sur aucun autre *type.* Les *propriétés du type* servent à introduire tout *littéral* pour ce *type,* les *opérateurs* applicables au *type* et (à titre facultatif) les *propriétés* du *type* en indiquant les *axiomes* qui sont vrais pour ce *type.*

Un *nouveau type* peut être fondé sur un autre *type* en utilisant *NEWTYPE* en combinaison avec des *règles d'héritage.* Dans ce cas, *l'ensemble de valeurs* du *nouveau type de données* est séparé de l'*ensemble de valeurs du type ascendant.* Bien que les *valeurs* et les *opérateurs* du *nouveau type de données* soient différents de ceux du type de données ascendant, les *littéraux* et les *noms* des *opérateurs* pour le *nouveau type* seront surchargés, c'est-à-dire que ce seront les mêmes *littéraux* et les mêmes *noms* que ceux du *type ascendant* et il faudra décider si un *littéral* ou un *nom* est approprié pour le *nouveau type* ou le *type de données ascendant,* par *qualification* ou d'après le *contexte.* Si le *lien* d'un *littéral* ou d'un *nom d'opérateur* à un *type* ne peut être déterminé, la spécification du LDS est *ambiguë* et donc *non valide.* Lorsque *ALL* est donné comme *règle d'héritage,* tous les *noms d'opérateur* sont *surchargés* pour le *nouveau type de données.* En revanche, les *noms d'opérateur* spécifiés dans la *règle d'héritage* doivent être des *noms d'opérateur* du *type ascendant* et ces *noms* sont définis pour le *nouveau type.* On *hérite* de l'*ensemble* d'*axiomes* et de l'*ensemble* de *littéraux* du *type* de *données ascendant.*

De même que les *littéraux,* les *noms d'opérateur* et les *axiomes hérités,* un *nouveau type de données* peut avoir des *littéraux,* des *opérateurs* et des *axiomes* supplémentaires spécifiés comme *propriétés de type* après le mot clé *ADDING.* Ces *littéraux, noms d'opérateur* et *axiomes* ne doivent pas être en contradiction avec ceux *hérités.*

Une *définition de données* de la forme:

```
NEWTYPE X /* détails */
    CONSTANTS /* liste de constantes */
    END X;
```

est équivalente à

```
NEWTYPE Anon /* détails */
    END Anon;
```

suivi de

```
SYNTYPE X = Anon
    CONSTANTS /* liste de constantes */
    END X;
```

L'utilisation d'une *restriction constante* dans un *NEWTYPE déclare implicitement* un *NEWTYPE anonyme* (Anon ci-dessus) sans cette restriction, qui est donc utilisé comme l'*ascendant* d'un *SYNTYPE* avec la *restriction constante.* Pour assurer l'*anonymat,* il est indiqué que le *nom ascendant* doit être différent de tous les autres *noms* indiqués dans la spécification du LDS en dehors de la *déclaration implicite* particulière.

4.3.2.2 *SYNTYPE*

Un *type de données* peut aussi être défini de façon à avoir un sous-ensemble des *valeurs* du *type de données ascendant* en utilisant *SYNTYPE*. Dans ce cas, *l'ensemble des valeurs* est spécifié après le *mot clé CONSTANTS*, ou bien toutes les *valeurs* du *type de données ascendant* ont des *valeurs* correspondantes dans le *SYNTYPE*. Seules les *valeurs* spécifiées peuvent être *affectées* aux *variables déclarées* avec un *SYNTYPE*.

L'*accès* à une *variable* avec un *SYNTYPE* donne une *valeur* du *type de données ascendant*. Ces *opérations* de *déclaration*, *affectation* et *accès* sont les seules opérations permises pour les SYNTYPES.

Les *valeurs* spécifiées pour un *type de données SYNTYPE* doivent toutes être des *valeurs* du *type de données ascendant*. L'*ascendant* d'un *SYNTYPE* est le deuxième type nommé dans la définition de *Syntype* à condition que ce *type* soit un *Newtype*. Dans le cas contraire, l'*ascendant* est l'*ascendant* du *type* nommé.

4.3.2.3 *Instantiation de générateur*

Une *instantiation* de *générateur* est équivalente au texte du *générateur* avec les *paramètres formels* textuellement remplacés par les *paramètres réels*. Chaque fois que le *nom* de *générateur* est utilisé dans le texte du *générateur*, il est remplacé par le *nom* du *type de données* ou du *générateur* appelant l'*instantiation du générateur*. Le texte équivalent doit compléter une *définition* valide de *type de données NEWTYPE* . Cette *définition* de *type de données* formée par l'expansion textuelle définit alors les *propriétés* du *nom* du *type de données*.

4.3.2.4 *Struct*

Une *définition* de *type de données* qui comprend un *Struct* rend implicites des *types de données* pour chaque *nom* de *champ*. Pour un *type de structure* donné S, pour chaque *nom de champ* Fi et l'*identificateur de type* correspondant Ti, les *axiomes* suivants sont introduits implicitement (moyennant *renforcement*; voir ci-dessous):

```
l'axiome unique      :  extract! (insert! (S, Fi, I), Fi) = I;
l'ensemble d'axiomes :  extract! (insert! (S, Fi, I), Fj)
                     =  Extract (S, Fj);
                        /* pour tous les Fi, Fj distincts */
```

Lorsqu'il y a N *noms de champ* dans une *structure*, N * N *axiomes* de cette forme seront *introduits implicitement*. Pour garantir la suppression de l'*ambiguïté*, le LDS exige que les *noms de champ* à l'intérieur d'une *structure* donnée soient uniques.

Un ensemble de *types* est associé à la *structure* S, un pour chaque *champ*, avec un *nom de type* S!Fi, le *littéral* unique Fi, et aucune autre *propriété*. Ce *type* devient le *porteur* pour le *nom de champ* utilisé dans les opérations *insert!* (insérer) et *extract!* (extraire).

L'effet d'une définition de *Struct* est de créer une structure ou un enregistrement (comme dans les langages de programmation), bien que la définition puisse inclure des *axiomes* supplémentaires pour *renforcer* le *comportement* du *type*. Lorsque des *axiomes* supplémentaires introduits explicitement par l'utilisateur sont en contradiction avec l'*ensemble d'axiomes* implicites pour ce *Struct*, on résout l'incompatibilité en détruisant les *axiomes* implicites. L'introduction d'*axiomes* explicites dans un *Struct* demande beaucoup de soin.

4.3.3 *Relation avec la syntaxe abstraite*

Une *définition de type de données* représente une *définition de type de données* dans la *syntaxe abstraite*. Le *nom de type* représente le *nom de type* dans la *syntaxe abstraite*.

4.3.3.1 *NEWTYPE*

Les *mots clés NEWTYPE* et *END* englobent le concept de *syntaxe abstraite* d'une *description de type de données*. L'*ensemble de noms de valeur*, les *introductions d'opérateur* et l'*ensemble d'axiomes de type* dans la *syntaxe abstraite* sont représentés de la façon suivante:

a) *Ensemble de noms de valeur*

L'ensemble de *littéraux* donné par les *littéraux* dans les *propriétés de type* combiné avec l'ensemble de *littéraux* pour le *type de données ascendant* si *INHERITS* est spécifié.

b) *Ensemble d'introduction d'opérateur*

L'ensemble d'*opérateurs* donné par les *opérateurs* dans les *propriétés* de *type* combiné avec l'ensemble d'*opérateurs du type de données ascendant* si *INHERITS* est spécifié.

c) *Ensemble d'axiomes de type*

L'ensemble d'*axiomes* (s'il y en a) donné par les *axiomes* dans les *propriétés de type de données* combiné avec l'ensemble d'*axiomes du type de données ascendant* si *INHERITS* est spécifié. Si l'*ensemble d'axiomes* est omis, ou incomplet, quelques opérations au moins peuvent seulement être *interprétées informellement*.

4.3.3.2 *SYNTYPE*

Les *mots clés SYNTYPE* et *END* englobent le concept de *syntaxe abstraite* d'une *description* de *SYNTYPE*. L'*identificateur de type ascendant* représente l'*identificateur de type de données ascendant* dans la *syntaxe abstraite*. L'*ensemble de valeurs* représente l'*ensemble de valeurs* dans la *syntaxe abstraite*.

4.3.3.3 *Instantiation de générateur*

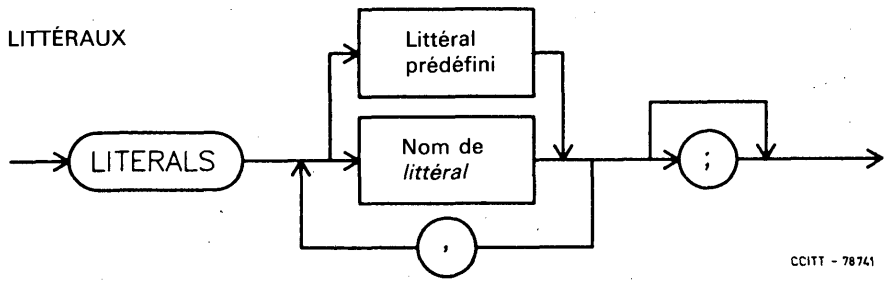
Une *instantiation de générateur* indique le texte qui serait obtenu en développant textuellement le *générateur*, comme au § 4.3.2.3, de façon qu'il ait la même relation avec la *syntaxe abstraite* que le texte équivalent.

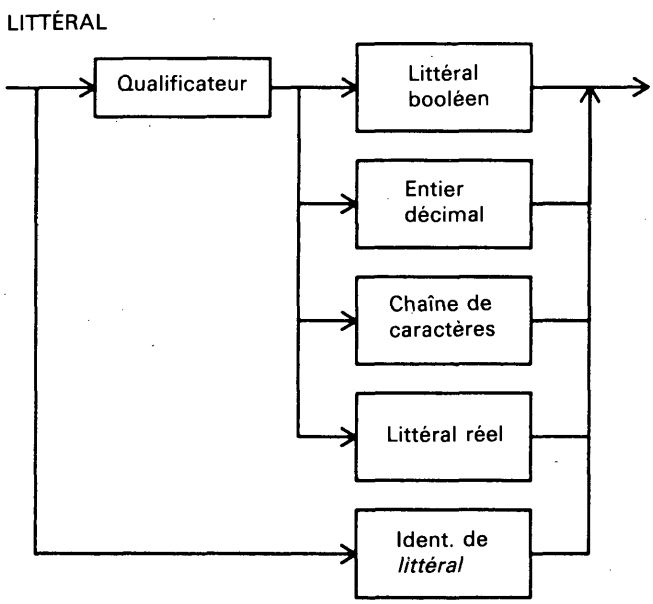
4.3.3.4 *STRUCT*

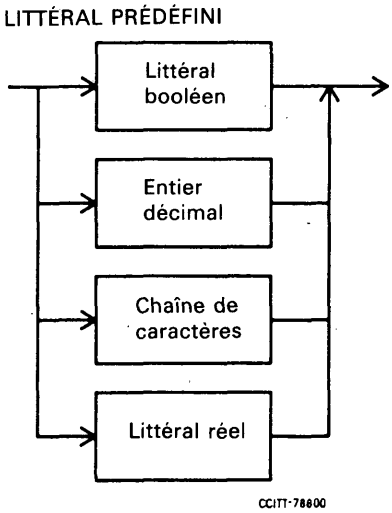
Un *Struct* indique le texte obtenu en nommant explicitement toutes les *propriétés* appropriées et, en conséquence, (comme pour un générateur), il a la même relation avec la *syntaxe abstraite* que le texte ainsi indiqué.

4.4 *LITTÉRAUX*

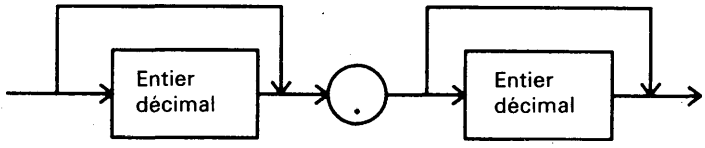
4.4.1 *Syntaxe*



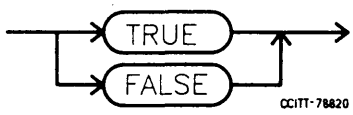




LITTÉRAL RÉEL



LITTÉRAL BOOLEEN



4.4.2 *Sémantique*

Les *littéraux* qui sont utilisés pour indiquer les *valeurs* d'un *type* de *données* sont soit *prédéfinis* (pour des *types* de *données prédéfinis* ou des *types* de *données* fondés sur des *types* de *données prédéfinis*), soit introduits par la liste d'indications pour les *littéraux* d'un *type* de *données* après le *mot clé LITERALS*. Lorsqu'un *type* comprend les *opérations Ordering! (classement)*, les *littéraux* doivent être nommés conventionnellement dans l'ordre ascendant.

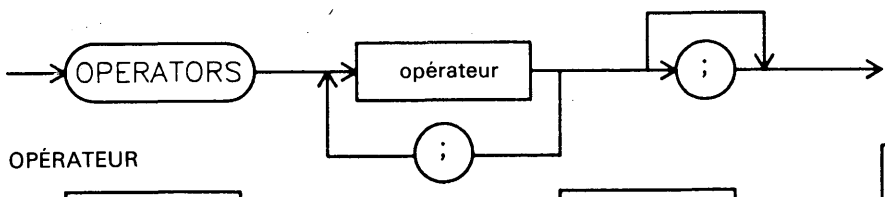
4.4.3 *Relation avec la syntaxe abstraite*

Les *noms* de *littéral* introduits par la partie *littéraux* des *propriétés* de *type* de *données* représentent les *noms* de *valeur* d'une *description* de *type* de *données* dans la *syntaxe abstraite.*

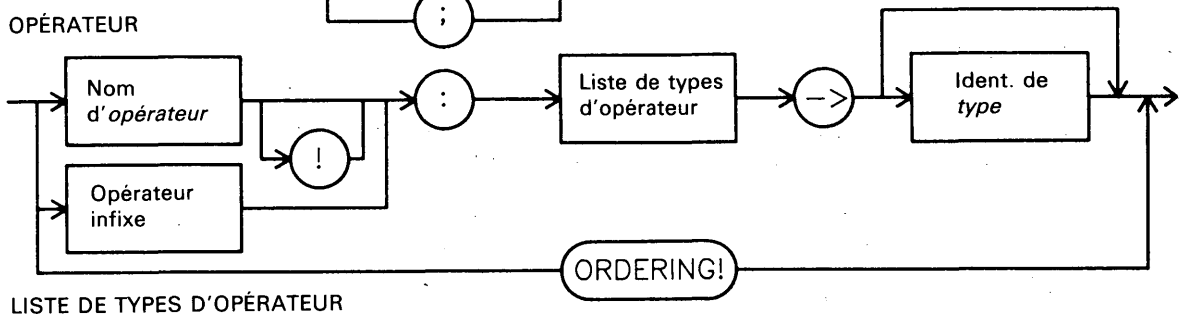
4.5 *Opérateurs*

4.5.1 *Syntaxe*

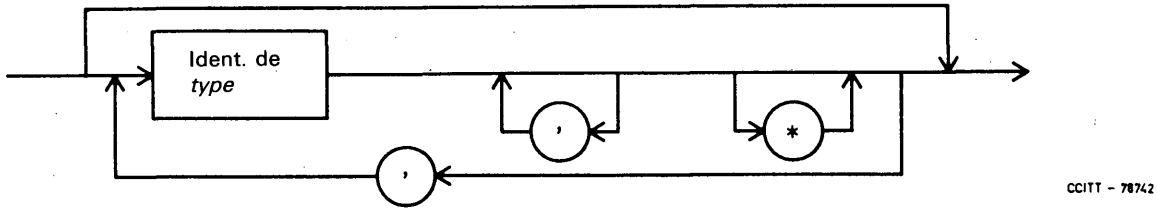
OPÉRATEURS



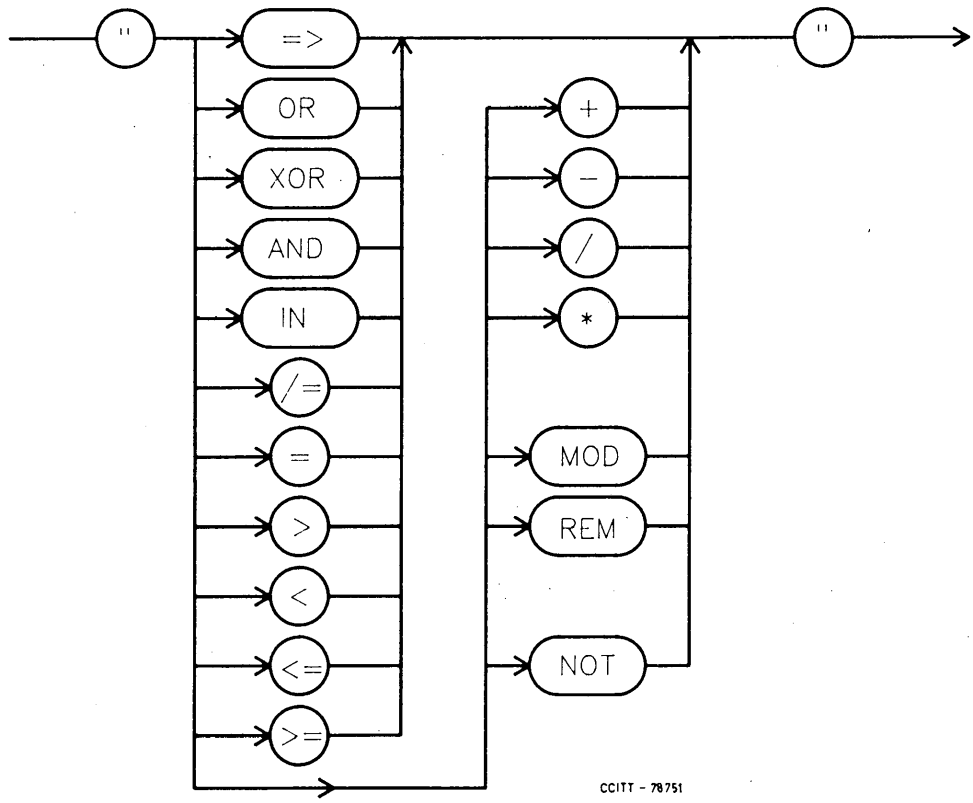
OPÉRATEUR



LISTE DE TYPES D'OPÉRATEUR



OPÉRATEUR INFIXE



4.5.2 *Sémantique*

Les *opérateurs* des *propriétés* des *types* de *données* introduisent les *noms* et la *paramétrisation* des *opérateurs*. La *paramétrisation* détermine le nombre de *paramètres* requis et le *type* de *données* de chaque *paramètre* ainsi que le *type* de *données* de toute *valeur* résultante.

4.5.2.1 *Noms des opérateurs*

Les *opérateurs d'ordre* sont spécifiés par l'inclusion de *Ordering!* dans les *opérateurs*. Il s'agit d'une notation abrégée pour l'introduction des *opérateurs* ci-après pour un type de données D.

"<" : D, D – > Boolean
">" : D, D – > Boolean
"< =" : D, D – > Boolean
"> =" : D, D – > Boolean

Les noms d'*opérateurs infixes* tels que +, AND et OR sont placés entre guillemets dans les *opérateurs*. Ils peuvent être utilisés comme *opérateurs préfixés* par mise entre guillemets de la forme suivante:

"+" (a,2)

qui équivaut à a + 2.

Un *nom* d'*opérateur* peut être suivi, au besoin, par un point d'exclamation qui signifie que l'on ne peut se référer directement à l'*identité* de l'*opérateur* que dans des *définitions* de *types* de *données*. Ce point d'exclamation fait partie du *nom* de l'*opérateur* et, dès lors, doit toujours être donné lorsque cet *opérateur* est utilisé.

Les *noms Assign!* (affecter), *Declare!* (déclarer), *Access!* (accéder), "=" (égal) et "/=" (différent de) sont implicitement définis comme des *opérateurs* pour tous les *types* de *données* avec le *classement implicite* du *type* pour un *type* de *données* D.

```
Assign!  : D'*, D->;
Declare! : D'* D->;
Access!  : D' D->;
"="      : D, D -> Boolean;
"/="     : D', D -> Boolean;
```

L'opérateur «*Assign*» est l'opérateur infixe ":=". Il n'y a pas pas d'opérateur «*Déclare*» vu que son utilisation est rendue implicite par des *déclarations*. L'opérateur «*Access*» est rendu implicite quand une *variable* est mentionnée dans un *contexte* qui nécessite une *valeur*. Les *opérateurs* «égal» et «différent de» sont respectivement les *opérateurs infixes* "=" et "/=".

4.5.2.2 *Typage des opérateurs*

La liste des *identificateurs* de *types* de *données* suivant les deux points et précédant le symbole (->) est appelée la *liste* des *types* d'*opérateurs*. Cette *liste* de *types* d'*opérateurs* spécifie les *types* de *données* des *valeurs* que requiert l'*opérateur*. Si un ou plusieurs *identificateurs* de *type* de *données* ont un *attribut principal*, l'*opérateur* est un *opérateur actif*; à défaut, c'est un *opérateur passif*. Un *opérateur actif* peut changer les *valeurs* associées aux *variables*, alors qu'un *opérateur passif* est purement fonctionnel et ne peut pas changer les *valeurs* associées aux *variables*.

Pour un *opérateur passif*, toutes les *identités* de *type* dans la *liste* des *types* spécifient que les *paramètres réels* de l'*opérateur* doivent être interprétés comme des *expressions*. Chacune de ces *expressions* doit rendre une *valeur*, qui est un membre d'un *ensemble* de *valeurs* du *type* de *données* de la position correspondante dans la *liste* des *types* de *données*.

Pour un *opérateur passif*, il faut un *identificateur* de *type* de *données* après le symbole (->). Cet *identificateur* de *type* de *données* spécifie que l'*opérateur*, lorsqu'il est utilisé, rendra une valeur de ce *type* de *données*. Pour un *opérateur actif*, certaines des *identités* de *types* de *données* dans la *liste* des *types* d'*opérateurs* sont suivies de *primes* (').

Une *prime* spécifie que l'*opérateur* requiert une *variable* du *type* de *données* en question en tant que *paramètre*, et la *valeur* associée à la *variable* peut être changée. Deux *identificateurs de types de données* ne peuvent être suivis du même nombre de *primes* dans la *liste* d'introduction des *types* de *données*.

Le nombre de *primes* distingue un *paramètre primé* d'un autre lorsque l'on utilise une *opération active* avec des *primes* pour désigner la *valeur* associée à une *variable* donnée comme un *paramètre* (ceci n'est permis que dans les *axiomes*). Par exemple:

```
OPERATORS SwapAndAdd: int', int'' -> int
        /* fragment de définition de type de données */
AXIOMS SwapAndAdd (a,b)' = b;
        /* axiome définissant le premier paramètre qui reçoit la valeur du second paramètre */
    SwapAndAdd (a,b)'' = a;
        /* axiome pour le second paramètre */
    SwapAndAdd (a,b) = a + b;
        /* axiome pour le résultat */
```

Si un *paramètre* à *prime(s)* est suivi d'un astérisque, on n'accède pas à la *valeur initiale* de la *variable* avec l'*opération* utilisée (il faut noter que les *axiomes* doivent être conformes à cela, faute de quoi le LDS est *incorrect*).

Si l'*identificateur* de *type* de *données* suivant le symbole (->) est omis en tant qu'*opérateur actif*, l'*opérateur* ne peut être utilisé à l'intérieur d'une *expression*. Il peut y avoir à la fois des *paramètres à prime(s)* et des *paramètres sans prime* dans une *liste* d'introduction de *type* de *données*.

Il y a une ambiguïté de syntaxe entre un *nom* de *type* de *données* suivi d'une chaîne de caractères entre guillemets dans la chaîne d'un nom, et un *identificateur* de *type* de *données* à prime(s) dans une *liste* de *types* de *données*. Ces ambiguïtés surviennent lorsqu'une prime suivant un *nom* de *type* dans une *liste* de *types* est suivie d'une *prime*, d'une virgule ou d'un signe moins (qui fait partie de ->). La *prime* est toujours considérée comme une *prime* de l'*identificateur* de *type* de *données* et non comme le départ d'une chaîne de caractères entre guillemets.

4.5.2.3 *Opérateurs Insert! et Extract!*

Afin de permettre la *définition axiomatique* des *tableaux* et des *structures*, il existe deux *noms* d'*opérateur préalablement définis* qui ont une signification spéciale différente de celle qu'ils ont lors de la *définition* de *type de données*. Ces *opérateurs* sont *Insert!* et *Extract!*. *Insert!* est un *opérateur actif* qui doit être défini avec les *identités* de *type de données* de telle manière que le premier *type* de *données* comporte une (des) *prime(s)* et que les autres *types* n'en comportent pas.

Pour utiliser *Insert!* autrement que pour la *définition de type de données*, on écrit le premier *paramètre* (qui doit être une *variable*), suivi d'une ouverture de parenthèse, de tous les *paramètres* restants sauf le dernier, puis d'une fermeture de parenthèse et de ":=", et enfin du dernier paramètre.

Ainsi, avec A, une *variable* dont le *type de données* a un *Insert!* défini, et i1, i2 et e comme *expressions* appropriées pour *Insert!* pour ce *type de données.*

Insert! (A, i1, i2, e)

est écrit sous la forme de

A(i1, i2) := e

Etant donné qu'*Insert!* peut être défini par plusieurs *types* de *données*, l'utilisation de *Insert!* est déterminée par le *type* de *variable. Insert!* doit être défini avec deux *paramètres* au moins et doit rendre le même *type* de *données* que le premier *paramètre.*

Extract! est un *opérateur passif* qui nécessite une *variable* comme premier *paramètre*, pour des raisons sémantiques. Pour utiliser *Extract!*, on écrit le premier *paramètre* suivi de tous les autres *paramètres* entre parenthèses.

Ainsi

Extract! (A, i1, i2)

s'écrit

A(i1, i2)

L'utilisation de *Extract!* est déterminée par le *type* de *variable.*

4.5.3 *Relation par rapport à la syntaxe abstraite*

Pour un *opérateur passif*, un *nom* d'*opérateur* ou un *opérateur infixe* représente un *nom* d'*opérateur* dans la *syntaxe abstraite.* Pour un *opérateur passif*, la *liste* d'introduction de *types* représente la liste d'*identités* de *types* de *paramètres* dans la *syntaxe abstraite.* L'*identificateur* de *type* de *données* suivant le symbole (– >) représente l'*identité* de *type* de *données* du résultat de l'application de l'opérateur.

Un *opérateur actif* est lié à la *syntaxe abstraite* par une ré-écriture en *opérateurs passifs.* Cela permet également de définir le fonctionnement harmonieux de l'*opération.* Pour chaque *paramètre à prime(s)*, il y a un *opérateur passif* implicite qui renvoie la *valeur* requise par le *jeu d'axiomes.* Pour chaque *paramètre à prime(s)* dépourvu d'astérisque, il y a une *variable implicite* dans chaque *instance de processus* utilisant l'*opérateur* avec le même *type de données* que le *paramètre* qui reçoit la *valeur initiale* du *paramètre.*

La liste des *identités* de *type* de *données* des *paramètres* dans la *syntaxe abstraite* pour chaque *opérateur passif implicite* est représentée par la *liste* d'introduction de *types de données* en ignorant tout paramètre comportant un des astérisques. Il y a autant d'*opérateurs passifs implicites* qu'il y a de *paramètres à prime(s)* dans la *liste* d'introduction de *types de données* et chaque *type de données* de *paramètre à prime(s)* est utilisé comme l'*identité de type de données* du résultat de l'utilisation de l'un de ces *opérateurs.* Par exemple:

```
OPERATORS
    complex: int', int'', int, int''' * – > Bool
        /* permute les deux premiers paramètres, place la somme des trois premiers paramètres dans le
        quatrième paramètre et retourne la valeur true si les deuxième et troisième paramètres étaient
        égaux */
AXIOMS
    complex (a,b,c,d)' = b;
    complex (a,b,c,d)'' = a;
    complex (a,b,c,d)''' = a+b+c;
    complex (a,b,c,d) = (b=c);
        /* voir le § 4.9.2 pour l'utilisation des primes dans les axiomes */.
```

Les opérateurs implicites sont:

```
implied1! : int, int, int – > int
implied2! : int, int, int – > int
implied3! : int, int, int – > int
implied4! : int, int, int – > Boolean
```

Si les *variables implicites* sont V1 et V2, l'utilisation de «complex» dans une déclaration est équivalente à:

V1 := a; V2 := b;

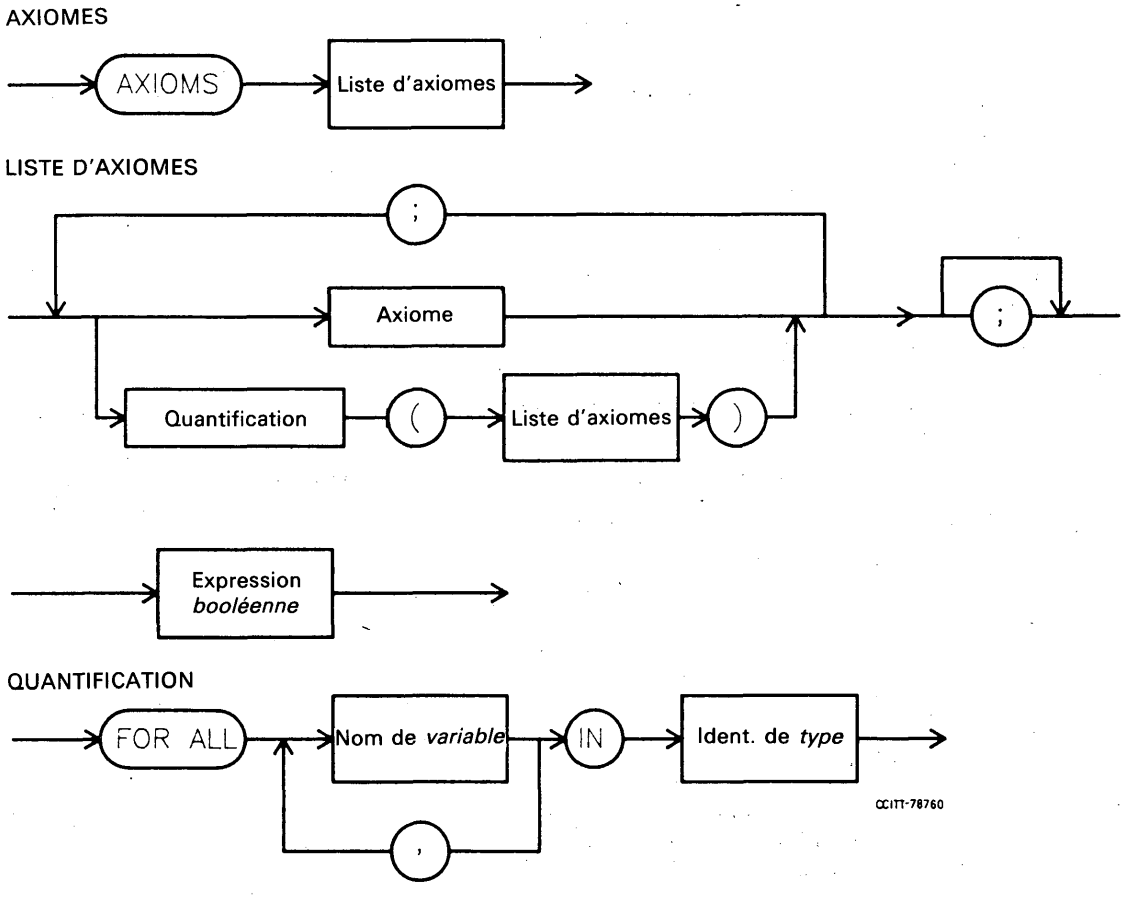
suivi de l'instruction mais avec V1 remplaçant a, V2 remplaçant b et implied4! remplaçant complex, suivi de:

a:=implied1! (V1,V2,c);
b:=implied2! (V1,V2,c);
d:=implied3! (V1,V2,c);

Lorsqu'une instruction comporte plus d'un *opérateur actif*, ces opérateurs sont substitués dans l'ordre dans lequel ils seraient *interprétés*.

4.6 *Axiomes*

4.6.1 *Syntaxe*



4.6.2 *Sémantique*

Les *axiomes* sont formés d'un ensemble d'*expressions booléennes* qui sont vérifiées pour toutes les *valeurs* des *variables* dans les *axiomes*.

A l'intérieur d'un *axiome*, une «*variable*» n'est jamais le nom d'une *variable* de *processus* ou de *procédure*, à laquelle est associée une *valeur*, mais qui est utilisée pour signifier que toutes les *valeurs* d'un *type* spécifique peuvent être substituées à la *variable*, l'*axiome* restant vérifié. Lorsque l'on considère cette substitution, un *nom* de *variable* donné représente toujours la même *valeur* dans un même *axiome*. Par exemple dans:

```
OPERATORS even: int -> Boolean
AXIOMS even (0) = True;                  /* axiome 1 */
       even (i) = NOT even (i+1); /* axiome 2 */
```

L'*axiome* 2 doit être vérifié pour i=2, i=3, i=4, etc., c'est-à-dire que:

```
even (2) = NOT even (2+1)
even (3) = NOT even (3+1)
even (4) = NOT even (4+1)
```

Généralement, le *type* de *données* d'une *variable* dans un *axiome* peut être déterminé par le *contexte*, c'est-à-dire que dans l'exemple ci-dessus, il faut qu'il soit du *type* de *données* signifié par la *syntaxe* de l'*opérateur*.

Parfois, en raison d'une *surcharge* de *noms* et des symboles (tels que "+") dans le LDS, il n'est pas possible de déterminer le *type* de *données* d'une *variable* d'*axiome* par le *contexte*, de telle sorte que la *quantification* supplémentaire est nécessaire. La *quantification* force une *variable* à être d'un *type* particulier. Si l'emploi d'une *variable* à l'intérieur d'un *axiome* est *ambigu* ou *incohérent*, le LDS/PR est *incorrect*.

La *quantification* permet également à un *nom* de *variable* de représenter les mêmes substitutions dans plus d'un *axiome*.

Les *noms* choisis pour les *variables* dans les *axiomes* doivent être différents des *littéraux* appropriés au *contexte* dans lequel la *variable* est utilisée.

Dès que les *opérateurs* "=" et "/=" sont rendus *implicites* pour tous les *types* de *données*, les *axiomes* suivants sont toujours *implicites*:

"/=" (a,b) = NOT ("=" (a,b))
"=" (a,a);
"=" (a,b) AND "=" (b,c) => "=" (a,c);
"=" (a,b) => "=" (b,a);

Lorsque *Ordering!* est spécifié, les *axiomes* suivants sont *implicites*:

"<" (a,b) => NOT ">" (a,b);
">" (a,b) => NOT "<" (a,b);
"<" (a,b) AND "<" (b,c) => "<" (a,c);
NOT "<" (a,a);
"<=" (a,b) => "<" (a,b) OR "=" (a,b);
">=" (a,b) => ">" (a,b) OR "=" (a,b)

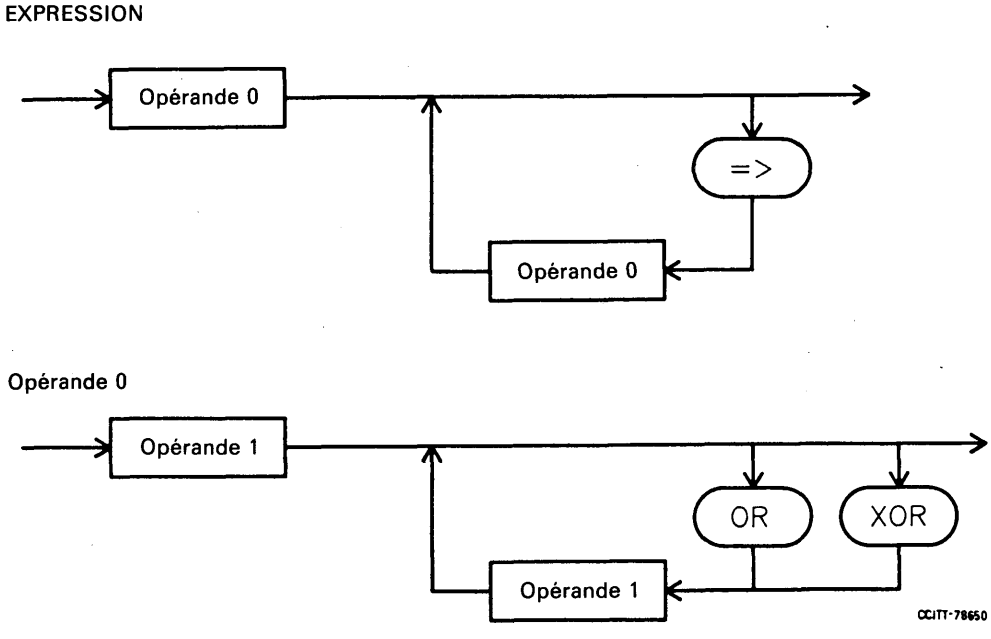
4.6.3 *Relation par rapport à la syntaxe abstraite*

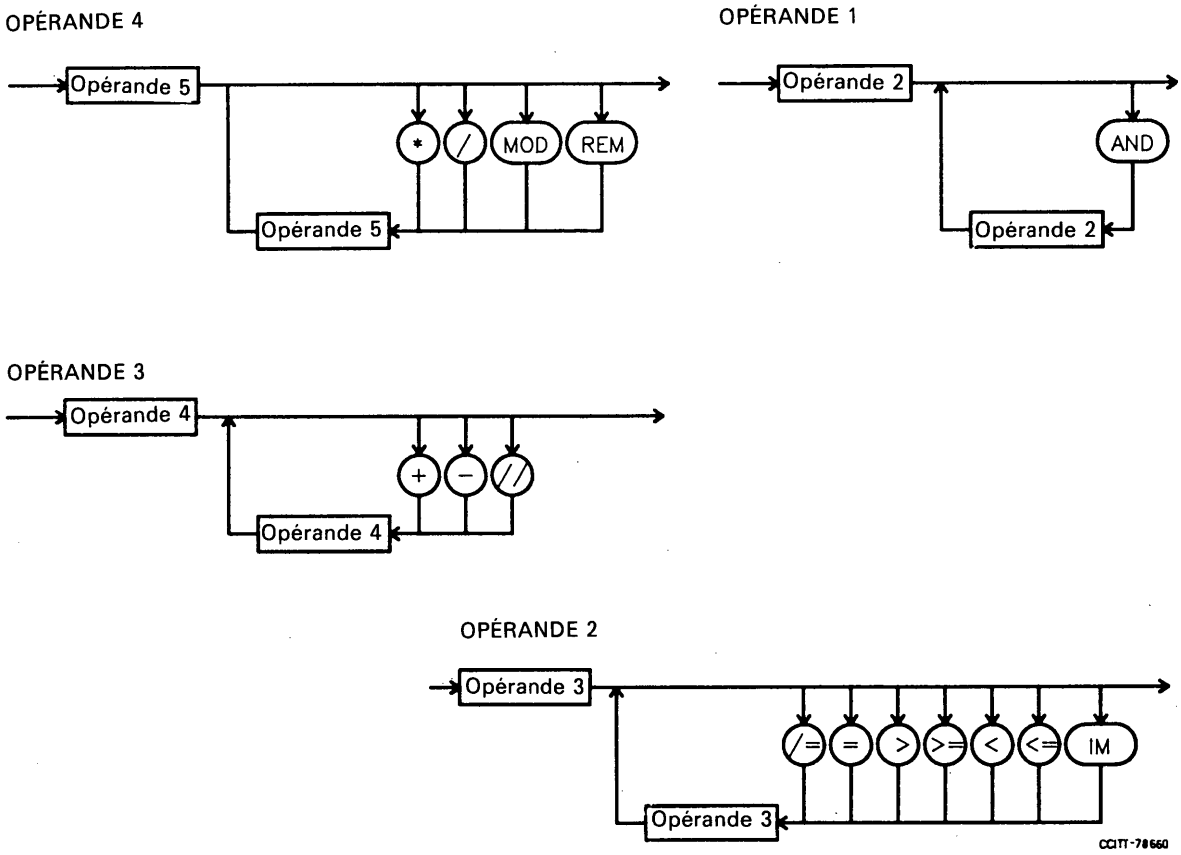
Dans la *syntaxe abstraite*, les *axiomes* représentent les *axiomes* de *type* de *données*, et chaque *axiome* représente un *axiome*.

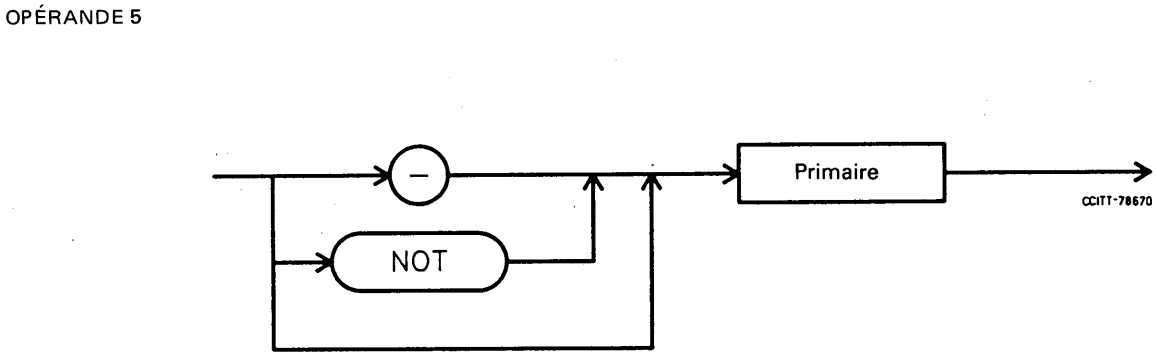
Dans la *syntaxe abstraite*, la *quantification* représente la *quantification*, sauf s'il y a une *quantification implicite* dans la *syntaxe concrète* pour toutes les *variables* de l'*axiome* dont le *type* de *données* est déterminé par le *contexte*.

4.7 *Expressions*

4.7.1 *Syntaxe*







4.7.2 *Sémantique*

Une *expression* est soit un *primaire*, soit l'utilisation d'un nombre d'opérateurs *infixes*.

L'ordre d'utilisation des *opérateurs* est déterminé par leur apparition dans la *syntaxe*, d'une manière similaire à celle des langages de programmation tels que le CHILL (voir la Recommandation Z.200). Cependant, le LDS permet également l'*opérateur* d'*implication booléen* (= >), qui a une *priorité* plus faible que tout autre *opérateur*. Il a la valeur *FALSE* (faux) pour les *opérandes booléens* si l'*opérande* côté gauche est *TRUE* (vrai) et que l'*opérande* côté droit est *FALSE* (faux). Dans les autres cas d'*opérandes booléens*, l'*opération implicite* a la valeur *TRUE*.

Normalement, tous les *opérateurs* auront les mêmes *propriétés* et la même validité telle que celle définie dans les langages de programmation, mais il faut noter que dans le LDS, l'utilisateur peut définir de nouvelles significations pour ces *opérateurs* par leur inclusion dans les *définitions* de *type* de *données*. Cependant, la *priorité* des *opérateurs infixes* ne peut être changée.

La *valeur* rendue par une *opération* valide est déterminée par les *axiomes* dans les *définitions* de *type* de *données*.

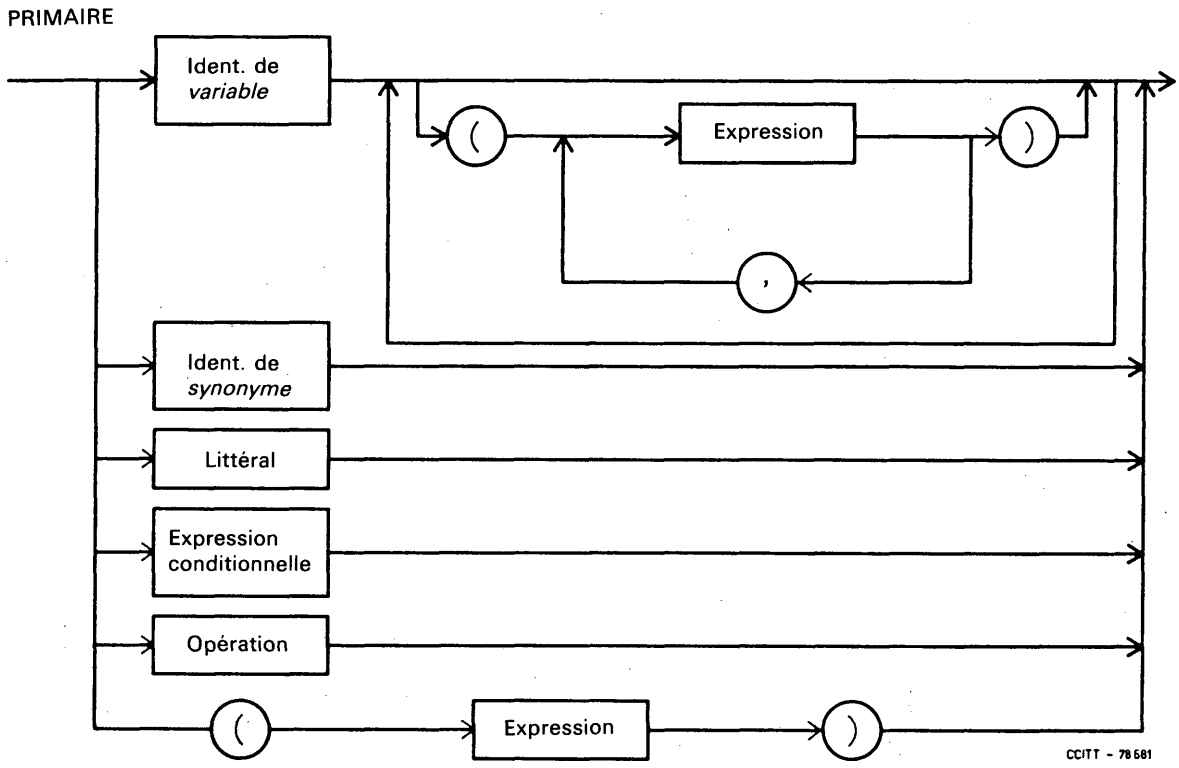
4.7.3 *Relation par rapport à la syntaxe abstraite*

Dans la *syntaxe abstraite*, une *expression* représente une *expression.*

Un *opérateur infixe* peut être *surchargé* et peut représenter un nombre quelconque d'*opérations.* Dans la *syntaxe abstraite*, la *surcharge* peut être résolue de deux manières: soit par le nombre et le *type* de *paramètres* ou, au cas où les *paramètres* eux-mêmes sont *surchargés*, par le *type* de *données* requis par le *contexte* dans lequel l'*opération* est utilisée.

4.8 *Primaire*

4.8.1 *Syntaxe*



4.8.2 *Sémantique*

Un *primaire* est une *identité de variable*, un *littéral*, une *identité de synonyme*, une *expression conditionnelle*, une *opération* ou une *expression entre parenthèses.*

4.8.3 *Relation par rapport à la syntaxe abstraite*

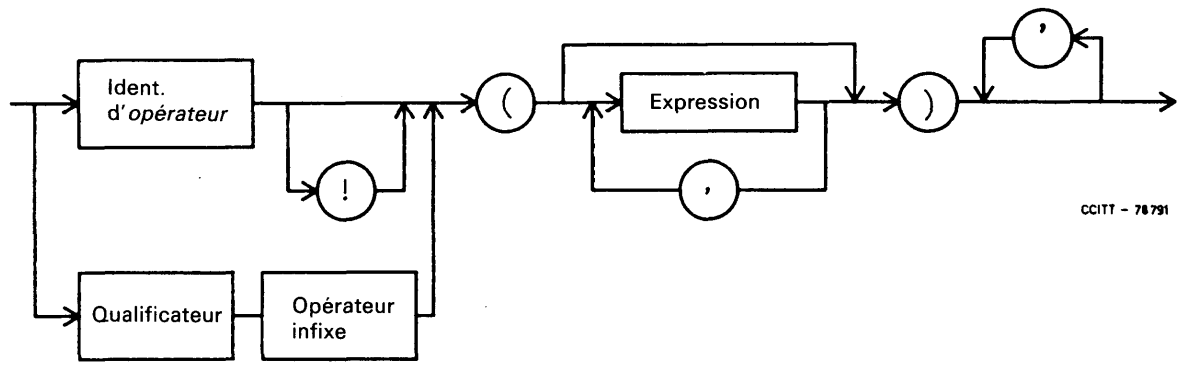
Une *identité variable*, un *littéral*, une *identité synonyme*, une *expression conditionnelle* ou une *expression entre parenthèses* représentent, dans la *syntaxe abstraite*, un *accès d'identité de variable*, une *identité de valeur*, une *identité de synonyme*, une *expression conditionnelle* ou une *expression*, respectivement.

Lorsqu'on se réfère à une *identité de variable* dans un *axiome*, elle représente une *variable axiomatique* plutôt qu'une référence à un *accès* à une *variable* déclarée pour un *processus* ou une *procédure.* Le *type de données* d'une telle *variable axiomatique* est déterminé par le *contexte.*

4.9 *Opération*

4.9.1 *Syntaxe*

OPÉRATION



4.9.2 *Sémantique*

Une *opération* est l'utilisation d'un *opérateur* défini dans une *définition* de *type*. Le nombre et le *type* des *expressions* utilisées comme *paramètres réels* doivent être conformes à la définition de l'*identité* de l'*opérateur*. Ces *paramètres* peuvent être utilisés pour déterminer quel *opérateur* est appliqué si le *nom* de l'*opérateur* est *surchargé*.

Si l'*opérateur* est appliqué dans un *axiome*, il faut, si le *nom* est défini avec un point d'exclamation, que ce point d'exclamation soit répété dans les *axiomes*.

Un *opérateur* défini avec un point d'exclamation ne peut être utilisé autrement que dans une *définition* de *type*.

Les *primes* après la fermeture de parenthèses de l'*opérateur* ne peuvent être utilisées que dans un *axiome* et elles signifient que l'*opération* a la *valeur* résultante du *paramètre* défini avec ce nombre de *primes*. Par exemple:

```
OPERATORS
    example: ti', t2'' - >

AXIOMS
    example: (v1, v2)'' = v1
    /* valeur de v1 introduite dans v2 */
```

Lorsque le *typage d'un paramètre* d'un *opérateur* est spécifié par des *primes*, le paramètre en question doit être une *variable*, sauf dans le *contexte* d'un *axiome*.

4.9.3 *Relation par rapport à la syntaxe abstraite*

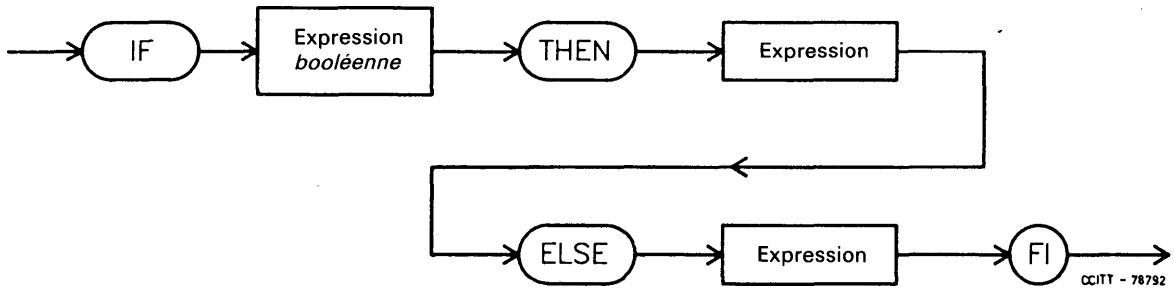
Dans la *syntaxe abstraite*, un *identificateur d'opérateur passif* représente un *opérateur*. Pour une *opération passive*, la liste d'*expressions* représente, dans la *syntaxe abstraite*, la liste d'*expressions* pour cette *opération*.

Une *opération active* représente l'*affectation* à des *variables implicites* qui sont ensuite utilisées comme des arguments pour *impliciter des opérations passives* dont les *valeurs* sont affectées en retour à des *variables* données comme *paramètres réels* (voir le § 4.5.3). A l'intérieur d'un *axiome*, une *opération active* représente une application de l'*opération passive implicite* appropriée déterminée par le nombre de *primes* ajoutées à l'*opération*.

4.10 *Expression conditionnelle*

4.10.1 *Syntaxe*

EXPRESSION CONDITIONNELLE



4.10.2 *Sémantique*

L'*expression conditionnelle* est *interprétée* en *interprétant* l'*expression* suivant le *THEN* si l'*expression booléenne* est *vérifiée* et, si non, elle rend la *valeur* de l'*expression* qui suit le *ELSE.*

A l'intérieur d'un *axiome*, chaque branche de l'*expression conditionnelle* ne doit être valable que pour la condition sous laquelle elle est choisie. Par exemple, étant donné que log(x) n'est pas défini pour des nombres négatifs, dans

IF r > 0 THEN log(r) ELSE 0.0 FI

il n'est pas important que pour r < = 0, log(r) ne soit pas défini.

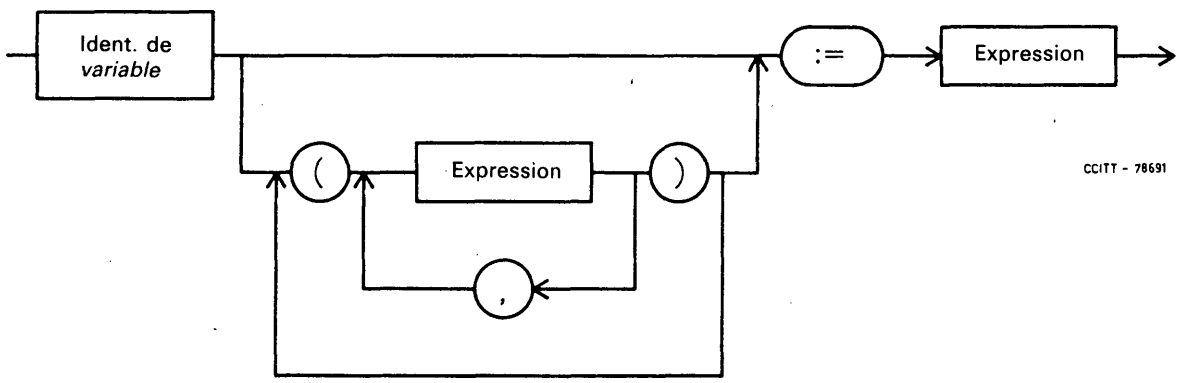
4.10.3 *Relations par rapport à la syntaxe abstraite*

Dans la *syntaxe abstraite*, une *expression conditionnelle* représente une *expression conditionnelle*

4.11 *Instruction d'affectation*

4.11.1 *Syntaxe*

INSTRUCTION D'AFFECTATION



4.11.2 *Sémantique*

Une *instruction d'affectation* permet à une *valeur* d'être associée à une *variable.*

La *valeur* rendue par l'*expression* côté droit est *affectée* à la *variable* ou à un élément de la *variable* côté gauche.

4.11.3 *Relation par rapport à la syntaxe abstraite*

Une *instruction d'affectation* représente l'utilisation d'un opérateur *Assign!* ou d'un opérateur *Insert!*.

Il y a une correspondance entre la *syntaxe* d'une *affectation* et l'application de l'*opérateur* approprié.

Lorsqu'on n'utilise pas de parenthèses du côté gauche d'une *affectation*, cette *affectation* représente l'utilisation de *Assign!*, de telle manière que

V := e représente Assign! (V,e)

Lorsqu'on utilise une seule paire de parenthèses, une telle affectation représente *Insert!* de telle manière que

a(i) := e représente Insert! (a,i,e)

et que

a(i,j) := e représente Insert! (a,i,j,e)

Lorsqu'on utilise plusieurs parenthèses, l'*affectation* est représentée par substitution récursive de telle manière que

a(i)(j) := e

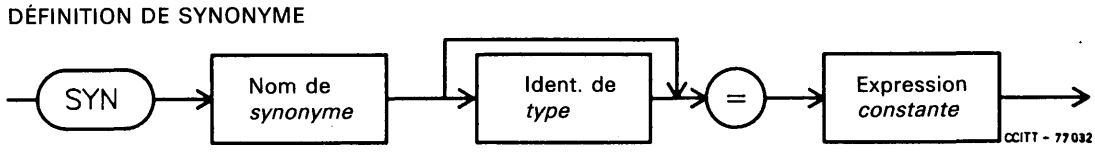
représente

```
vi := a(i);
Insert! (vi,j,e);
a(i) := vi;
```

où vi est une *variable implicite* de même *type* que a(i). Les opérations *Insert!* implicites sont des *opérations actives* qui sont représentées dans la *syntaxe abstraite* normalement pour des *opérateurs actifs* (voir le § 4.9.3).

4.12 *Définition de synonyme*

4.12.1 *Syntaxe*



4.12.2 *Sémantique*

Le *synonyme* est équivalent à l'*expression constante*. Si le *type* de l'*expression* ne peut être déterminé ni par la *constante*, ni par le *contexte* de la *définition* de *synonyme*, il est nécessaire de spécifier un *type*; dans le cas contraire, la *valeur* et le *type* de l'*expression constante* (et dès lors de la valeur du *synonyme*) sont déterminés par le *contexte* dans lequel apparaît la *définition* de *synonyme*.

4.12.3 *Relation par rapport à la syntaxe abstraite*

Dans la *syntaxe abstraite*, un *synonyme* représente une *définition* de *synonyme*. Si le *type de données* est supprimé, alors il est rendu implicite par le *contexte*.

4.13 *Générateur de type de données*

4.13.1 *Syntaxe*

4.13.2 *Sémantique*

Le *nom* donné dans le *générateur de type de données* est un *nom de générateur de type de données*. Les *propriétés* fournies dans un *générateur* forment une spécification partielle d'un *type de données*. Lorsqu'un *générateur* est utilisé dans une *définition de type* ou un autre *générateur de type de données*, les *paramètres* vers le *générateur* sont textuellement substitués dans la *définition du générateur* et (en même temps que toute propriété

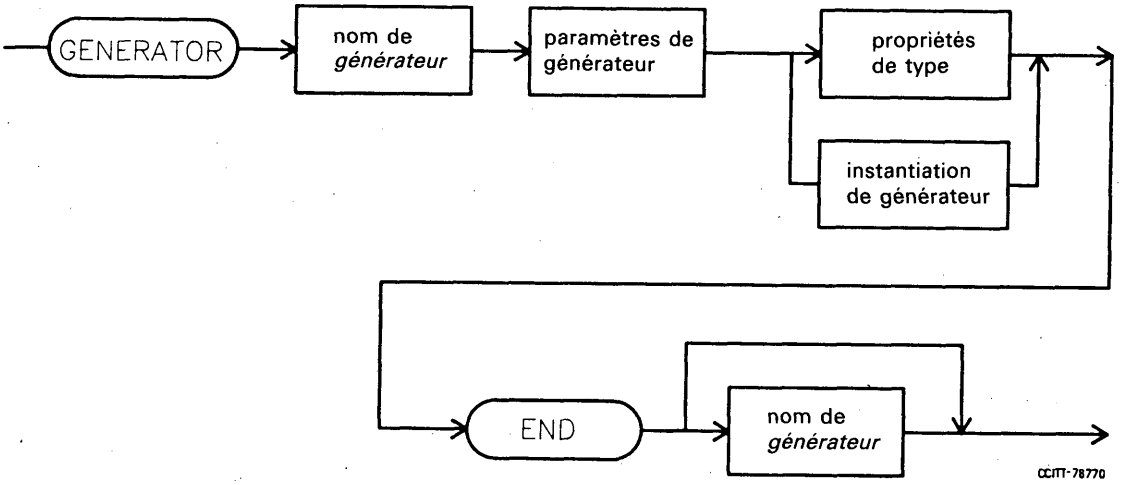
ajoutée dans une *définition de type*) doivent ensuite former une *définition de type* complète ou un autre *générateur de type de données.* Lorsqu'un *générateur de type de données* est défini en termes d'une *instantiation de générateur,* les *paramètres des générateurs* peuvent avoir les mêmes *noms* que ceux fournis à l'*instantiation.* Un tel *générateur* est une *instantiation partielle.* Par exemple:

```
GENERATOR Stack (TYPE Component, CONSTANT Maxsize)
    /* Détails */
    END Stack;
GENERATOR IStack(CONSTANT Max) Stack(Integer, Max)
    END IStack;
    /* Une pile de nombres entiers dont la taille maximum n'est pas spécifiée */
```

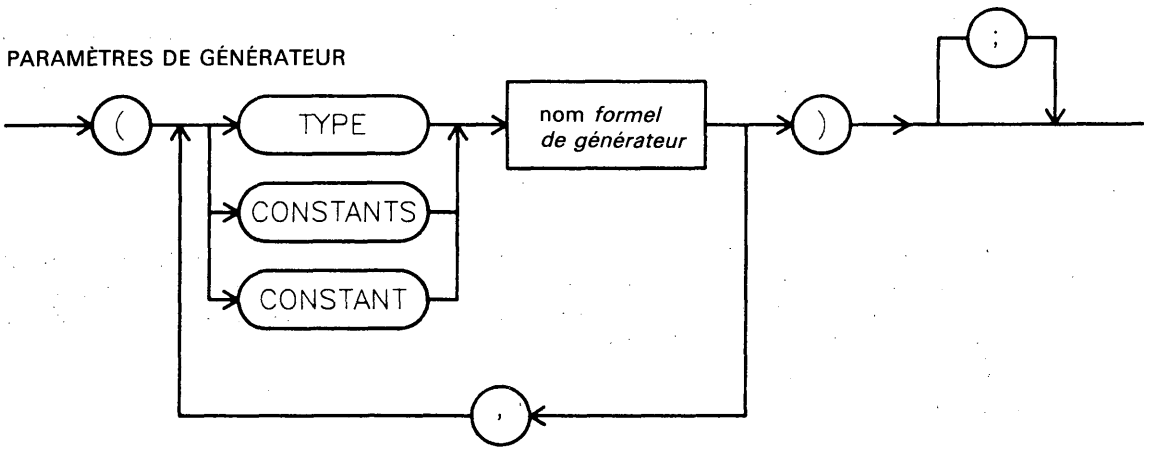
4.13.3 *Relation par rapport à la syntaxe abstraite*

Le générateur de type de données n'a pas d'homologue dans la syntaxe abstraite. L'utilisation d'une instantiation de générateur dans une définition de type de données dénotera le texte formé par la substitution paramétrique.

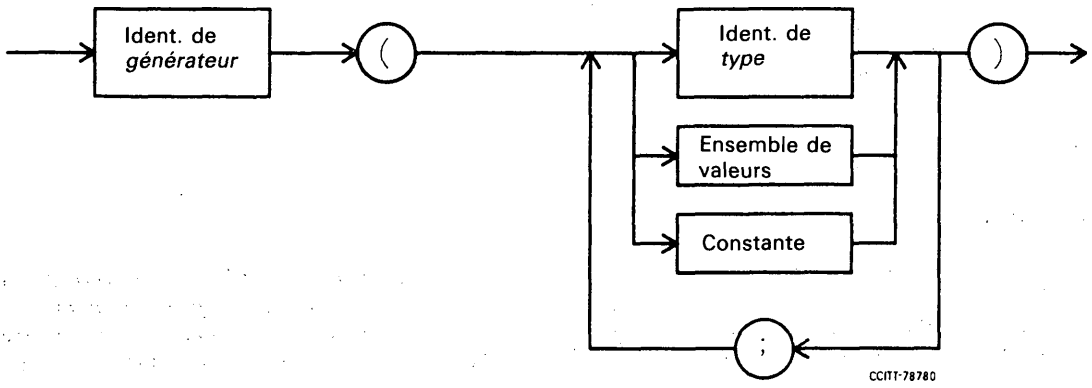
GÉNÉRATEUR DE TYPE DE DONNÉES



PARAMÈTRES DE GÉNÉRATEUR



INSTANTIATION DE GÉNÉRATEUR



5 Types de données prédéfinis

5.1 *Entier*

```
NEWTYPE Integer
        /* Littéraux selon la syntaxe littérale des entiers */
    OPERATORS
        "+": Integer, Integer -> Integer;
        "-": Integer, Integer -> Integer;
        "*": Integer, Integer -> Integer;
        "/": Integer, Integer -> Integer;
        Float: Integer -> Real;
        Fix: Real -> Integer;
    AXIOMS
        /* Hérités d'entiers mathématiques, en ajoutant: */
        Fix(Float(r)) = r;
        r - 1 < Float(Fix(r)) <= r;
        i/j = Fix(Float(i)/Float(j));
    END Integer;
```

5.2 *Réel*

```
NEWTYPE Real
        /* Littéraux de la forme spécifiée dans «littéral réel» */
    OPERATORS
        "-": Real, Real -> Real;
        "+": Real, Real -> Real;
        "*": Real, Real -> Real;
        "/": Real, Real -> Real;
        "-": Real -> Real;

        /* Les axiomes sont hérités des réels mathématiques; ce travail nécessite un complément d'étude
        afin d'être spécifié ici */

    END Real;
```

5.3 *Tableau*

```
GENERATOR Array (TYPE Index, TYPE Item);
    OPERATORS
        Insert!: Array', Index, Item ->;
        Extract!: Array, Index -> Item;
    AXIOMS
        Extract! (Declare!(v)', ]) = Error!;
        Extract!(Insert!(A, IPos, It), EPos) =
          If Epos = Ipos Then It Else Extract(A, EPos) FI;
    End Array;
```

5.4 *Booléen*

```
NEWTYPE Boolean
    LITERALS True, False;
    OPERATORS
        "Not": Boolean -' Boolean;
        "And": Boolean, Boolean -> Boolean;
        "Or": Boolean, Boolean -> Boolean;
        "=>": Boolean, Boolean -> Boolean;
```

```
AXIOMS
    "Not"(True) = False;
    "Not"(False) = True;
    "And"(A, B) = If A = False Then False Else B;
    "Or"(A, B) = If A Then True Else B;
    "=>"(A, B) = If A = True And B = False Then False Else True;

END Boolean;
```

5.5 *Caractère*

```
NEWTYPE character

    LITERALS /* Chaîne de caractères de longueur 1, où les caractères sont ceux de l'alphabet du
     CCITT */;

    OPERATORS
        Ordering!;

    END character;
```

5.6 *Naturel*

```
SYNTYPE Natural Integer Constants >= 0 END Natural;
```

5.7 *Mode ensembliste*

```
GENERATOR Powerset (TYPE Item);

    LITERALS Empty;

    OPERATORS
        "IN" : Item, Powerset -> Boolean;
        Incl. : Item, Powerset' ->;
        Del. : Item, Powerset' ->;
        Ordering!;

    AXIOMS;
        Declare(v)' = Empty;
        I IN Empty = False;
        I IN Incl(I2, S) = IF I = I2 THEN True ELSE I IN S FI;
        Del(I, Incl(I2, S)) = IF I = I2 THEN Del(I, S) ELSE Incl(I2, Del(I, S)) FI;
        For all S1, S2 in powerset (for all I in Item (S1 < S2 => (I IN S1 => I IN S2)));
        Not (I in S) => Del(I, S) = S;

    END powerset;
```

5.8 *PId*

```
NEWTYPE PId

    LITERALS Null;

    OPERATORS
        Create! : -> PId;

    AXIOMS;
        Declare!(v)' = Null;
        Create! /= Create!;
        /* Une manière faible d'établir que tous les Create! produisent des valeurs uniques */
        Create! /= Null;

        /* Les valeurs PId sont renvoyées par l'interprétation d'un nœud de demande de création (elles
        sont désignées comme étant générées par la fonction de création ci-dessus). Chaque interprétation
        de demande de création engendre une valeur PId unique qui n'est pas égale à zéro. Chaque cas
        de traitement déclare implicitement trois variables PId, nommées "Parent", "Self" et "Offspring".

        L'interprétation d'un nœud de demande de création engendre une valeur PId nouvelle et unique,
        et l'affecte au "Offspring" pour la tâche de création. L'identificateur "Self" de la tâche créée se
        verra attribuer cette même valeur alors que "Parent" de la tâche créée se verra attribuer la valeur
        de "Self" du "créateur". La longueur de "Offspring" dans la tâche créée est mise à zéro.

    */ END PId;
```

5.9 *Chaîne*

```
GENERATOR String (TYPE Item);
        /* Littéraux spécifiés par les constantes de chaîne de caractères, */
        /* uniquement si le générateur est instancié par caractère */

    OPERATORS
        Declare!(v)' : -> String;
        "//"         : String, String -> String;
        Length       : String -> Natural;
        First        : String -> Item;
        Last         : String -> Item;
        Extract!     : String, Natural -> Item;
        Insert       : String', Natural, String ->;
        Insert!      : String', Natural, Item ->;

    AXIOMS
        Length(Declare!(v)') = 0;
        Length (S1 // S2) = Length(S1) + Length(S2);
        First(S1) = Extract!(S1, 1);
        Last(S1) = Extract!(S1, Length(S1));
        Extract!(Insert(S1,I,S2),j) =
          IF j< 1 THEN Error! ELSE
            IF j<= I THEN Extract!(S1,j) ELSE
              IF j<= Length(S2) + I THEN Extract!(S2,j-I)) ELSE
                IF j<= Length (S1) + Length(S2) THEN
                        Extract!(S1,J-Length(S2)) ELSE Error!
                FI
              FI
            FI
          FI;
        S1 // S2 = Insert(S1, Length(S1), S2);
        Extract!(Insert!(S1, I, It)', J) =
          IF I = J Then It Else Extract!(S1, J) FI;

    END String;
```

5.10 *Temps*

```
NEWTYPE Time Inherits Real

ADDING
    Operators
        Now: -> Time /* the 'real' time */;

END Time;
```

5.11 *Durée*

```
NEWTYPE Duration Inherits Real ("+", "-", "*", "/")

ADDING
    Operators
        "+": Time, Duration -> Time;
        "+": Duration, Time -> Time;
        "-": Time, Duration -> Time;

CONSTANTS >= 0.0

END Duration;
```

5.12 *Temporisateur*

```
NEWTYPE TIMER

    OPERATORS
        Set: Time, Timer'* - >;
        Reset: Timer' - >;
        Active: Timer - > Boolean;

    AXIOMS
        Active( Set(Tm, Tmr)' ) = Tm > Now ();
        Active( Reset(Tmr) ) = False;
        /* Il est à noter qu'un temporisateur actif envoie un signal au processus lorsqu'il devient inactif.
        Ce signal reçoit le même nom que la variable du temporisateur dans "Set call" */

    END Timer;
```

5.13 *Chaîne de caractères*

```
NEWTYPE Charstring String (Character)
    Adding LITERALS /* character string literals */
    END Charstring;
```

Imprimé en Suisse — ISBN 92-61-02232-4