Etude et réalisation d'une traduction automatique d'Algol 60 en Algol 68

Alain Landelle

N° d'ordre

THESE

présentée à

LA FACULTE DES SCIENCES DE L'UNIVERSITE DE GRENOBLE

pour obtenir

LE GRADE DE DOCTEUR DE TROISIEME CYCLE

"Informatique"

par

Alain LANDELLE

Etude et réalisation d'une traduction automatique d'Algol 60 en Algol 68

Thèse soutenue le 14 Mai 1971 devant la commission d'examen :

MM	N. GASTINEL	Président
	M. BERTHAUD	
	J.C. BOUSSARD	Examinateurs
	M. GRIFFITHS	

FACULTE DES SCIENCES
DE GRENOBLE

LISTE DES PROFESSEURS

<> <> <>

PROFESSEURS TITULAIRES

MM.	NEEL Louis	Physique expérimentale
	KRAVTCHENKO Julien	Mécanique rationnelle
	CHABAUTY Claude	Calcul différentiel et intégral
	BENOIT Jean	Radioélectricité
	CHENE Marcel	Chimie papetière
	FELICI Noël	Electrostatique
	KUNTZMANN Jean	Mathématiques Appliquées
	BARBIER Reynold	Géologie appliquée
	SANTON Lucien	Mécanique des fluides
	OZENDA Paul	Botanique
	KOSZUL Jean Louis	Mathématiques pures
	GALVANI Octave	Mathématiques pures
	MOUSSA André	Chimie nucléaire et radioactivité
	TRAYNARD Philippe	Chimie générale
	SOUTIF Michel	Physique générale
	CRAYA Antoine	Hydrodynamique
	REULOS René	Théorie des Champs
	BESSON Jean	Chimie minérale
	AYANT Yves	Physique approfondie
	GALLISSOT François	Mathématiques pures
Mle.	LUTZ Elisabeth	Mathématiques pures
MM.	BLAMBERT Maurice	Mathématiques pures
	BOUCHEZ Robert	Physique nucléaire
	LLIBOUTRY Louis	Géophysique
	MICHEL Robert	Minéralogie et pétrographie
	BONNIER Etienne	Electrochimie-Electrométallurgie
	DESSAUX Georges	Physiologie animale
	PILLET Emile	Physique industrielle
	YOCCOZ Jean	Physique nucléaire théorique
	DEBELMAS Jacques	Géologie générale
	GERBER Robert	Mathématiques pures
	PAUTHENET René	Electrotechnique
	MALGRANGE Bernard	Mathématiques pures
	VAUQUOIS Bernard	Calcul électronique
	BARJON Robert	Physique nucléaire
	BARBIER Jean Claude	Physique expérimentale
	SILBER Robert	Mécanique des fluides
	BUYLE-BODIN Maurice	Electronique

	DREYFUS Bernard	Thermodynamique
	KLEIN Joseph	Mathématiques pures
	VAILLANT François	Zoologie
	ARNAUD Paul	Chimie
	SENGEL Philippe	Zoologie
	BARNOUD Fernand	Biosynthèse de la cellulose
	BRISSONNEAU Pierre	Physique du solide
	GAGNAIRE Didier	Chimie physique
Mme	KOFLER Lucie	Botanique et physiologie végétale
MM.	DEGRANGE Charles	Zoologie
	PEBAY-PEROULA Jean Claude	Physique
	RASSAT André	Chimie systématique
	DUCROS Pierre	Cristallographie
	DODU Jacques	Mécanique appliquée- I.U.T."A"
	ANGLES D'AURIAC Paul	Mécanique des fluides
	LACAZE Albert	Thermodynamique
	GASTINEL Noël	Analyse numérique
	GIRAUD Pierre	Géologie
	PERRET René	Servomécanisme
	PAYAN Jean Jacques	Mathématiques pures
	CAUQUIS Georges	Chimie organique
	RENARD Michel	Thermodynamique
	BONNET Georges	Electrotechnique
	BOLLIET Louis	Informatique- I.U.T."B"

PROFESSEURS ASSOCIES

MM.	RADHAKRISHNA	Thermodynamique
	BULLEMER Bernhard	Physique

PROFESSEURS SANS CHAIRE

M.	GIDON Paul	Géologie et minéralogie- C.S.U.
Mme	BARBIER Marie Jeanne	Electrochimie
Mme	SOUTIF Jeanne	Physique générale
MM.	COHEN Joseph	Electrotechnique
	DEPASSEL Roger	Mécanique des fluides
	GLENAT René	Chimie organique
	BARRA Jean	Mathématiques appliquées
	COUMES André	Electronique- E.I.E.
	PERRIAUX Jacques	Géologie et minéralogie
	ROBERT André	Chimie papetière
	BIAREZ Jean Pierre	Mécanique
	BONNETAIN Lucien	Chimie minérale
	HACQUES Gérard	Calcul numérique
	POULOUJADOFF Michel	Electrotechnique
Mme	KAHANE Josette	Physique
Mme	BONNIER Jane	Chimie générale
MM.	VALENTIN Jacques	Physique nucléaire
	REBECQ Jacques	Biologie- C.S.U.Chambery
	DEPORTES Charles	Chimie minérale

MM.	SARROT-REYNAULD Jean	Géologie
	BERTRANDIAS Jean Paul	Mathématiques appliquées
	AUBERT Guy	Physique
	GAUTHIER Yves	Sciences biologiques- C.S.U.
	DOLIQUE Jean Michel	Physique des plasmas
	DESRE Pierre	Métallurgie
	LAURENT Pierre	Mathématiques appliquées
	CARLIER Georges	Biologie végétale
	SIBILLE Robert	Construction mécanique-I.U.T."A"

MAITRES DE CONFERENCES

MM.	LANCIA Roland	Physique automatique
Mme	BOUCHE Liane	Mathématiques. C.S.U.Chambery
MM	KAHANE André	Physique générale
	BRIERE Georges	Physique expérimentale
M.	LAJZEROWICZ Joseph	Physique
Mme	BERTRANDIAS Françoise	Mathématiques pures
MM.	LONGEQUEUE Jean Pierre	Physique nucléaire
	ZADWORNY François	Electronique
	DURAND Francis	Métallurgie
	PFISTER Jean Claude	Physique générale
	CHIBON Pierre	Biologie animale
	IDELMAN Simon	Physiologie animale
	BLOCH Daniel	Electrotechnique- I.P.
	MARTIN-BOYER Michel	Chimie-C.S.U.Chambery
	BRUGEL Lucien	Energétique- I.U.T."A"
	BOUVARD Maurice	Mécanique des fluides
	ARMAND Yves	Chimie-I.U.T."A"
	KUHN Gérard	Physique- I.U.T."A"
	RICHARD Lucien	Botanique
	PELMONT Jean	Physiologie animale
	BOUSSARD Jean Claude	Mathématiques appliquées.I.P.
	MOREAU René	Hydraulique. I.P.
	GERMAIN Jean Pierre	Mécanique
	JOLY Jean René	Mathématiques pures
Mle	PIERY Yvette	Biologie animale
MM	CONTE René	Physique I.U.T."A"
	PEFFEN René	Métallurgie I.U.T. "A"
	LE JUNTER Noël	Electronique I.U.T. "A"
	VIALON Pierre	Géologie
	BENZAKEN Claude	Mathématiques appliquées
	MAYNARD Roger	Physique
	BLIMAN Samuel	Electronique E.I.E.
	DUSSAUD René	Mathématiques C.S.U. Chambery
	BELORIZKY Elie	Physique
Mme	LAJZEROWICZ Jeannine	Physique
M.	JULLIEN Pierre	Mathématiques pures
Mme	RINAUDO Marguerite	Chimie macromoléculaire
MM	LEROY Philippe	Mathématiques I.U.T. "A"
	BRODEAU François	Mathématiques I.U.T. "B"

MM	ROMIER Guy	Mathématiques I.U.T. "B"
	NEGRE Robert	Mécanique I.U.T. "A"
	BEGUIN Claude	Chimie organique
	BUISSON Roger	Physique I.U.T."A"
	IVANES Marcel	Electricité I.U.T. "A"
	GIDON Maurice	Géologie
	COHEN ADDAD	Spectrométrie physique
	VAN CUTSEM Bernard	Mathématiques appliquées
	GRIFFITHS Michael	Informatique
	MACHE Regis	Physiologie végétale
	GENSAC Pierre	Sciences biologiques
	JOUBERT Jean Claude	Physique du solide I.P
	VEILLON Gerard	Mathématiques appliquées I.P.
	MARECHAL Jean	Physique mécanique I.U.T. "A"
	PECCOUD François	Analyse I.U.T. "B"
	CHIAVERINA Jean	Biologie appliquée E.F.P.

MAITRES DE CONFERENCES ASSOCIES

MM.	CHEEKE John	Thermodynamique
	BOUDOURIS Georges	Radioélectricité
	BENENSON Walter	Physique nucléaire
	GOLDSCHMIDT H.	Mathématiques

Je tiens à remercier

Monsieur le Professeur N. GASTINEL, *qui a bien voulu me faire l'honneur de présider le jury,*

Monsieur M. BERTHAUD, *Ingénieur au Centre Scientifique IBM-France, auteur d'un traducteur automatique d'Algol 60 en PL/I, et dont les conseils m'ont été précieux, tant pour le démarrage, que pour la mise au point de ce projet,*

Monsieur J.C. BOUSSARD, *Maître de Conférences à l'Institut Polytechnique de Grenoble, pour la confiance qu'il m'a témoignée, et l'intérêt qu'il a su me communiquer pour Algol 68, à l'époque où il n'était encore qu'Algol X,*

Monsieur M. GRIFFITHS, *Maître de Conférences à la Faculté des Sciences de Grenoble, sans lequel ce travail n'eut peut être jamais été mené à bien.*

Je souhaite également remercier tous mes collègues de Laboratoire, en particulier Messieurs P.Y. CUNIN *et* J. PLEYBER, *pour leur active et amicale collaboration à la réalisation de ma thèse.*

Je ne saurais, non plus, oublier Mademoiselle J. QUACCHIA, *pour son inlassable gentillesse et son impeccable dactylographie;*

ainsi que Messieurs C. ANGUILLE, C. LABORIE, *et* P. MOUNET, *pour leur contribution à la réalisation matérielle de cet ouvrage.*

TABLE DES MATIERES

TABLE DES MATIERES

INTRODUCTION

INTRODUCTION

La mise en oeuvre de ce projet remonte à octobre 1968, date à laquelle le langage Algol X commençait à arriver enfin à une certaine stabilisation. L'adoption du rapport final devait bientôt suivre, et ce fut : X = 68.

Ce projet fut donc l'un des premiers, relatifs à Algol 68, à être défini à la Faculté des Sciences de Grenoble. Comme eux, il avait pour but de constituer une approche du nouveau langage.

En effet, Algol 68 [VANW1] présente par rapport à Algol 60 [NAUR] dont il est destiné à être le successeur, des particularités - sur lesquelles nous aurons l'occasion de revenir -, telles que son implémentation ne peut être envisagée par des méthodes analogues à celles qui avaient été utilisées pour Algol 60 [BRAN1, BRAN2, MAIL]. C'est pour cette raison que nous décidâmes, dans une première étape, de ne nous intéresser qu'à un sous-ensemble du langage - le plus petit sous-ensemble d'Algol 68 de puissance supérieure à celle d'Algol 60.

Nous verrons qu'en fait, la définition n'en est pas minimale. Nous avons en effet cru bon d'y inclure des possibilités que n'offrait pas Algol 60, et qui restent simples à implémenter - par exemple l'initialisation des déclarations. C'est-à-dire, que pour un concept d'Algol 60 donné, nous avons recherché un concept d'Algol 68 l'exprimant au mieux, mais simple à implémenter - en l'étendant donc quelquefois pour augmenter le pouvoir d'expression du sous-ensemble, quand l'extension restait simple à implémenter. Les concepts d'Algol 68 utilisés n'ont donc pas été poussés au maximum, de telle sorte que la définition du sous-ensemble n'est pas orthogonale au sens de Wijngaarden. Mais la question se posait, et se pose d'ailleurs toujours, de savoir ce que nous devions ou non y inclure.

Le but de cette démarche était, avant d'entreprendre l'étude du compilateur du langage complet, l'obtention rapide d'un noyau d'Algol 68, destiné à remplacer Algol 60 dans les divers enseignements et à servir de langage véhiculaire pour la communication des algorithmes.

Parallèlement à ce projet d'implémentation d'un sous-ensemble d'Algol 68 compilable de façon classique, fut envisagée l'étude d'un traducteur Algol 60 - Algol 68. L'écriture de ce traducteur répondait à deux motivations :

1) une étude comparative des deux langages, pour voir comment les concepts d'Algol 60 se retrouvaient - ou sinon pouvaient éventuellement être exprimés - en Algol 68.

2) la réalisation effective d'un traducteur automatique en raison de l'abondante bibliothèque de sous-programmes en Algol 60 de l'I.M.A.G.

De plus, ce traducteur définissait - indirectement - le sous-ensemble d'Algol 68 que nous cherchions. De cette façon, les deux projets - sous-ensemble et traducteur - eurent une base commune : la définition des règles de traduction.

Mais on ne peut cependant considérer le sous-ensemble comme un sous-produit du traducteur, car, en fait, les règles de traduction ont été choisies en fonction des séquences en Algol 68 qu'elles généraient. Nous avons donc été amenés à restreindre le nombre des concepts d'Algol 68 à utiliser; ceci, bien évidemment, au détriment de la qualité de la traduction.

Le document de travail relatif aux règles de traduction fut publié [LAND] et fit l'objet d'une communication au Congrès Algol 68 de Vancouver en août 1969 [VANC].

Cette définition préalable ayant été faite en commun, les deux projets divergèrent, J. Pleyber prenant en charge l'écriture du compilateur du sous-ensemble.

Les termes techniques utilisés dans cette rédaction pourront sembler être du mauvais "franglais", et nous nous en excusons. Mais, dans le but d'éviter toute ambiguïté, nous avons préféré utiliser les termes anglais du rapport Algol 68 - bien que leur traduction en français soit à l'ordre du jour du groupe Algol 68 de l'AFCET. Nous espérons aussi que le mélange inévitable des termes techniques propres aux deux langages Algol 60 et Algol 68 n'apportera pas de confusion.

La suite de cet exposé est divisée en deux parties. On trouvera dans les différents chapitres de la première, une présentation rapide d'Algol 68

et de ses concepts - surtout par rapport à Algol 60, et dans le contexte de la traduction -, une présentation du langage Algol 60 source - c'est-à-dire les suppositions faites sur le programme source -, les caractéristiques principales du sous-ensemble d'Algol 68 - ceci de façon relativement sommaire; on se reportera pour une définition complète à la thèse de J. Pleyber [PLEY] -, enfin, le schéma de l'organisation du traducteur. La deuxième partie sera consacrée à l'étude des règles de traduction qui ont été choisies - sous les deux aspects algorithmique et programmation.

PREMIERE PARTIE

PRINCIPES GENERAUX

CHAPITRE I

PRESENTATION D'ALGOL 68

CHAPITRE I

PRESENTATION D'ALGOL 68

Ce chapitre n'a pas pour but une description complète d'Algol 68 et de ses concepts, mais plutôt une étude comparative avec Algol 60, et dans le contexte qui nous intéresse, celui de la traduction.

Cette présentation sera donc assez sommaire, et nous supposerons même que le lecteur a, des concepts d'Algol 68, des idées plus précises que celles que nous allons essayer de dégager. Il en trouvera, en effet, dans [VAND] et [BOUS] une introduction non formelle meilleure que celle que nous pourrions donner.

Si Algol 68 est un successeur d'Algol 60, il ne peut cependant être considéré comme en étant une extension. Les concepts d'Algol 60 qui étaient apparus, à l'usage, comme étant les plus intéressants, ont été repris, mais plutôt comme cas particuliers de constructions plus générales. C'est l'"orthogonalité" de Van Wijngaarden [VANW2] - orthogonalité qui réside dans la minimisation du nombre des concepts et en leur expansion maximum.

Le langage Algol 68 est défini en trois niveaux : le langage strict, le langage étendu, et le langage de représentation.

Le langage strict est celui décrit par la syntaxe. Il en était de même du "langage de référence" en Algol 60. Mais si la description syntaxique d'Algol 60 était faite en notation de Backus [BACK], celle d'Algol 68 utilise une grammaire à deux niveaux, c'est-à-dire le mécanisme d'une méta-grammaire permettant de générer les règles de la grammaire proprement dite.

Le langage étendu est obtenu à partir du langage strict par les "extensions". Les extensions n'augmentent pas la puissance du langage. Elles facilitent l'écriture des programmes, et quelquefois aussi, l'implémentation. C'est ainsi, par exemple, que l'instruction pour n'est introduite en Algol 68 qu'au niveau du langage étendu.

Le langage de représentation est le seul à dépendre des représentations externes utilisées pour les terminaux ("symboles") de la grammaire. Ce langage est celui utilisé pour l'implémentation sur une machine donnée.

Algol 68 possède quatre types primitifs : entier, réel, booleen et caractère. On y trouvera donc des possibilités de manipulation de chaînes qui n'existaient pas dans la définition d'Algol 60. A partir de ces types primitifs, il est possible de définir de nouveaux types de valeurs. En fait, la notion de type d'Algol 60 a été généralisée en Algol 68, en celle de "mode" dont les productions sont en nombre infini [KOST].

Les modes figurent, en Algol 68, dans la syntaxe. Il en est de même des contraintes de mode, dites en Algol 68, "coercions", qui ne sont plus exprimées en paragraphes sémantiques comme en Algol 60, mais directement dans la syntaxe. Nous aurons l'occasion de revenir sur l'absence de coercion automatique réel vers entier. Cette absence est justifiée par la perte d'exactitude de l'information. Il en résulte de multiples façons de définir l'équivalent entier d'un nombre réel : troncature, arrondis, ... entre lesquelles on a refusé de faire un choix.

Algol 68 a le mérite d'introduire clairement le concept de "nom"; c'est-à-dire de définir des modes correspondant aux notions usuelles de "constante", "variable", "pointeur", ...

La notion de déclaration d'Algol 60 se retrouve généralisée dans le concept de "déclaration d'identité" par laquelle l'identificateur d'un paramètre formel est fait "posseder" la valeur du paramètre effectif correspondant : constante, nom, ... Il est ainsi possible, en particulier, de déclarer en Algol 68 des constantes ou des pointeurs alors que l'on ne pouvait en Algol 60, déclarer que des variables.

La portée des déclarations est aussi en Algol 68 une portée de bloc. Mais le concept de bloc y a été étendu. Par rapport à Algol 60, tout ce qui se trouve entre begin et end, if et then, then et else, else et fi est devenu un bloc. Ceci implique qu'en particulier, la distinction bloc-instruction composée a disparu, et qu'il n'est plus possible de se renvoyer par une instruction goto à l'intérieur d'une instruction composée ou conditionnelle.

L'ensemble des déclarations standard constitue en Algol 68, le "standard-prélude" - qui s'incorpore automatiquement à tout programme particulier. Il contient non seulement des fonctions d'analyse, mais aussi des procédures standard d'entrée-sortie (ce dont on ne disposait pas en Algol 60) et des déclarations d'opérateurs et de leurs priorités. Ces priorités qui étaient implicites et fixes en Algol 60, sont au contraire dynamiquement modifiables par le programmeur en Algol 68.

Les concepts d'Algol 60, d'instruction et d'expression, ont été unifiés en celui de "clause", qui délivrera ou non une valeur.

L'exécution - en Algol 68, on dit : "élaboration" - peut être sérielle (comme c'était le cas en Algol 60) ou collatéralle. Nous interpreterons cette notion d'exécution collatéralle, au niveau de la traduction et du sous-ensemble, comme un refus du rapport de préciser un ordre d'exécution - ce qui facilite les recherches d'optimisation de l'implémentation, et revient à laisser les effets de bord non définis.

L'instruction pour se retrouve en Algol 68 plus concise et plus efficace. Elle ne comporte qu'un seul "élément de pour", et est de la forme suivante :

for I from A by B to C while D do E

Valeurs initiale et finale, ainsi que le pas, sont de mode entier et calculées une fois pour toutes à l'entrée de la boucle - donc non modifiables en cours d'exécution. Elle est donc très efficace puisque les valeurs des différents composants ne sont pas recalculées après chaque exécution de l'instruction contrôlée, contrairement à ce qui se passait en Algol 60. Son utilisation est donc réservée à des constructions pour lesquelles il est possible de générer un code objet efficace - ce qui est l'un des objectifs d'Algol 68. L'utilisateur des instructions pour d'Algol 60, s'il désire retrouver les mêmes effets, devra recourir à une suite d'instructions sémantiquement équivalente.

Expressions et instructions conditionnelles sont devenues des "clauses conditionnelles", parenthésées syntaxiquement par le couplage if - fi. Ceci a permis de résoudre les ambiguïtés qui étaient apparues à l'origine d'Algol 60 et qui amenèrent la restriction que l'on sait sur l'instruction de la partie then. Cette possibilité de choix entre deux clauses est étendue dans la "clause de choix" (case clause) où le choix peut être fait parmi un nombre non déterminé de clauses unitaires, selon la valeur d'une clause unitaire entière.

La manipulation de tableaux est beaucoup plus complète en Algol 68 qu'elle ne l'était en Algol 60. Il est possible d'y manipuler, par exemple, des constantes aussi bien que des variables du mode tableau. A cet effet a été introduit le concept de "valeur multiple", ainsi que le mode "rangée de MODE" - les affectations de ce mode réalisant une affectation globale de la

valeur multiple. D'une valeur multiple donnée, il est possible d'extraire une sous-valeur à laquelle peut se référer une "tranche" (slice) du tableau initial. Les tableaux pourront être d'encombrement variable - à chaque affectation - lors d'une même activation, s'ils sont déclarés à "bornes flexibles". Ceci implique pour le compilateur d'Algol 68 une gestion de la mémoire différente de celle qui était faite en Algol 60.

Il est aussi possible en Algol 68 de manipuler un ensemble de valeurs, non toutes du même type simple : les "valeurs structurées"; la sélection d'un élément se faisant, non par indexation, mais à l'aide d'un sélecteur de champ.

Les concepts de rémanence (own) et d'étiquette numérique ont disparu, ainsi que les expressions de désignation et les aiguillages. Nous verrons dans la seconde partie comment il est éventuellement possible d'exprimer ces concepts en Algol 68.

CHAPITRE II

PRESENTATION DU LANGAGE ALGOL 60 SOURCE

CHAPITRE II

PRESENTATION DU LANGAGE ALGOL 60 SOURCE

Nous définissons ici le niveau du langage Algol 60 considéré; c'est-à-dire, les conditions auxquelles doit satisfaire le programme en Algol 60, source du traducteur.

Les premières restrictions apportées au langage appartiennent à celles du compilateur d'Algol 60 actuellement utilisé à l'I.M.A.G.

- pas d'étiquette numérique
- spécification obligatoire de tous les paramètres formels
- déclaration d'aiguillage appartenant à la portée de toute quantité apparaissant dans la liste.

La traduction de la notion de rémanence, pour les tableaux, nous a amené à utiliser le concept d'Algol 68, de tableau à bornes flexibles. Ce concept n'a pas été introduit dans le sous-ensemble, de telle sorte que, tout au moins pour l'instant, c'est gratuitement que l'étude de l'expression de la rémanence des tableaux en Algol 68 a été faite.

La définition d'Algol 60 ne comprend pas de procédures standard d'entrée-sortie. Nous verrons ultérieurement le traitement de celles que nous avons choisies, et les restrictions qui les concernent : restrictions sur les data-sets manipulés, et sur les valeurs lues ou écrites. Ces dernières restrictions consisteront à ne pas permettre de tolérance de correspondance de type, en entrée ou en sortie.

Nous verrons aussi, lors du traitement des expressions arithmétiques, les restrictions relatives aux opérandes de l'opérateur d'exponentiation. Le domaine de définition de cet opérateur est en effet plus restreint en Algol 68 qu'il ne l'était en Algol 60.

La traduction des déclarations de procédure et de leurs appels nous amènera aussi à imposer quelques restrictions au programme en Algol 60 :

En effet, la partie spécification ne fournit pas le mode Algol 68 complet d'un paramètre formel de type tableau ou procédure - d'où de nombreuses manipulations pour rechercher l'information manquante. Cette information pourra venir de plusieurs sources différentes qui devront toutes fournir le même mode Algol 68. Nous verrons que ceci n'est nullement évident, ni incompatible avec le fait de supposer le programme en Algol 60 correct. Le problème se compliquera pour un paramètre formel de type procédure, en raison des tolérances de correspondance de type arithmétique entre les paramètres formels et les paramètres effectifs en Algol 60.

Les restrictions concernant les appels de procédure, tiennent au mode d'appel des paramètres. En effet, Algol 68 ne dispose que d'un seul mode d'appel des paramètres - que nous décrirons - à l'aide duquel nous devrons exprimer les appels par nom et par valeur d'Algol 60. Nous verrons dans quelle mesure nous y parvenons, et jusqu'à quel point nous pouvons restituer dans le programme en Algol 68 les possibilités qui étaient offertes en Algol 60 par les correspondances de type arithmétique dans les appels par nom et les appels par valeur. Nous justifierons aussi la non prise en charge de la traduction de l'appel par valeur d'un paramètre de type étiquette.

Le programme Algol 60 en entrée est supposé correct - c'est-à-dire qu'il a déjà pu être compilé et exécuté avec succès. En effet, soumettre au traducteur un programme en Algol 60 dont la mise au point n'est pas terminée, n'aurait qu'un intérêt assez limité. Aussi, nous ne ferons aucune vérification, si bien qu'un programme erroné pourra peut-être fournir une traduction, mais évidemment non garantie.

Les effets de bord seront ignorés. Ils sont en effet fonction de l'implémentation, et n'appartiennent pas au langage de référence. Ils pourront, dans la traduction, en générer d'autres, pas nécessairement les mêmes, d'ailleurs. Cependant, nous nous sommes efforcés d'éviter d'en générer d'autres nous-mêmes - en particulier lors de la traduction de l'instruction pour d'Algol 60.

Nous supposerons que les corps des déclarations de procédure d'Algol 60 ne sont pas en code - c'est-à-dire qu'ils ne peuvent être exprimés dans un langage autre qu'Algol 60. En effet, un tel code est fonction de la représentation machine, et dépend du compilateur où il est destiné à être inséré.

Les identificateurs du programme seront restitués dans la traduction, systématiquement tronqués à 8 caractères - en raison de l'utilisation d'un dictionnaire d'identificateurs.

Enfin et plus généralement, nous devrons trouver dans le programme en Algol 60 source toute l'information nécessaire à la traduction. Cette recherche posera quelquefois de nombreuses difficultés, et il pourra même arriver que nous soyions dans l'impossibilité de traduire. La traduction serait alors abandonnée après emission d'un message d'erreur approprié.

CHAPITRE III

CARACTERISTIQUES DU SOUS-ENSEMBLE D'ALGOL 68

CHAPITRE III

CARACTERISTIQUES DU SOUS-ENSEMBLE D'ALGOL 68

Notre intention n'est pas ici une description exhaustive des concepts du sous-ensemble; nous ne ferions que redire ce que nous avons dit au chapitre de présentation d'Algol 68. Nous nous bornerons à une présentation rapide des caractéristiques les plus marquantes. Pour le reste, disons simplement qu'étant d'une puissance supérieure à celle d'Algol 60, il est possible d'y exprimer tout ce que l'on pouvait exprimer en Algol 60. Nous terminerons ce chapitre par la description du langage de représentation utilisé pour la sortie du traducteur.

Les variables simples pourront être déclarées avec initialisation. Mais l'expression d'initialisation ne pourra dépendre que de quantités non locales au bloc contenant la déclaration.

Des affectations globales de tableaux pourront être effectuées. Mais l'expression partie droite ne pourra être qu'un identificateur de tableau, car les constantes valeurs multiples n'ont pas été introduites dans le sous-ensemble. La présence éventuelle du symbole flex dans une borne de tableau en partie gauche d'affectation signifie seulement que la borne est fixée par cette initialisation, et non qu'elle est susceptible de pouvoir varier par la suite.

Dans un programme écrit sur le sous-ensemble, les modes des paramètres formels sont complètement décrits. Ceci permet une compilation séparée des procédures dans la mesure où il n'y a pas de quantités non locales.

Les concepts d'instruction et d'expression y sont unifiés en celui de "clause". La valeur éventuellement délivrée sera, ou non, prise en considération, selon le contexte.

Les clauses conditionnelles sont parenthésées par le couplage if - fi. Il n'y a donc pas dans le sous-ensemble les restrictions qu'il y avait en Algol 60. Les parties then et else, ainsi que la condition, pourront être des "clauses sérielles"; c'est-à-dire un assemblage de déclarations et d'instructions, où toutes les déclarations doivent précéder la première instruction étiquetée.

L'instruction pour du sous-ensemble est d'utilisation plus restreinte que celle d'Algol 60, mais elle gagne en efficacité.

Les concepts d'aiguillage et d'expression de désignation n'existent pas dans le sous-ensemble, mais nous verrons dans l'étude de leurs traductions qu'il est possible de les exprimer à l'aide d'autres concepts d'Algol 68.

L'intérêt du sous-ensemble est de pouvoir être implémenté de façon classique et de permettre la génération d'un code objet efficace, en raison de la simplicité de ses concepts.

La représentation externe que nous donnons maintenant est celle qui correspond à la sortie du traducteur.

En sortie cartes perforées, le programme symbolique en Algol 68, obtenu en sortie du traducteur, sera perforé sur les colonnes 1 à 72 d'une carte, selon les conventions usuelles - les colonnes 73 à 80 étant habituellement réservées à une information externe d'identification.

La représentation externe qui a été choisie, est basée sur le chapitre 3 du rapport Algol 68, de préférence à [LIND]. Cependant des différences pourront apparaître, dues à la non disponibilité des représentations proposées. C'est ainsi que :

- le symbole d'étiquette sera représenté par ".."
- la lettre E sera utilisée pour le facteur de cadrage des nombres réels
- le caractère espace dans une chaîne sera restitué en un blanc
- nous avons adopté pour les crochets de tableaux la représentation d'Algol 60
- les mots-clés n'étant pas non plus en Algol 68 des identificateurs réservés, nous avons choisi pour les distinguer, la méthode d'Algol 60 : les enfermer entre apostrophes. Ce symbole est typographiquement différent du guillemet de chaîne, et ne provoque donc aucune ambiguïté.

Il est possible que cette représentation externe doive ultérieurement être revue, en fonction de ce que seront les contraintes du compilateur du sous-ensemble, ou celles du compilateur du langage Algol 68 complet.

CHAPITRE IV

ORGANISATION DU TRADUCTEUR

CHAPITRE IV

ORGANISATION DU TRADUCTEUR

4.1 INTRODUCTION

Sur la donnée d'un programme source en Algol 60 satisfaisant aux restrictions du chapitre 2, et dans le cadre des limitations de concepts que nous nous sommes imposées à cause du sous-ensemble, nous avons cherché à obtenir un programme en Algol 68 qui utilise au mieux les possibilités de ce langage.

En effet, un traducteur est un programme de traitement qui, en principe, n'est utilisé pour un programme donné, qu'une seule fois. L'objectif qui doit présider à son écriture, n'est donc pas tant sa rapidité, que l'efficacité du code qu'il génère.

Cependant, on constatera rapidement, qu'en n'imposant au programme source en Algol 60 que le minimum de restrictions, nous nous sommes surtout intéressés à l'étude comparative des deux langages. Nous nous sommes, en effet, efforcés de restituer en Algol 68, le maximum d'un concept donné d'Algol 60, en considérant peut-être trop souvent le cas général et sans tirer parti des cas simples où la traduction est beaucoup plus immédiate, en particulier pour les tableaux rémanents et les instructions pour. Il ne faudrait pourtant pas en déduire que le rôle même du traducteur n'est que secondaire, même si le programme résultant en Algol 68 risque d'être quelquefois moins performant que le programme source en Algol 60.

Ce traducteur procède en trois passages, qu'il eut peut-être été possible de restreindre à deux, mais au prix de multiples complications. Nous avons choisi la solution de facilité - mais aussi de simplicité - en fonctionnant en trois passages.

Deux passages sont indispensables comme le montre l'exemple suivant :

```
begin
     procédure p;
          begin
          :
            a := x;
          :
          end;
     real x; integer a;
     :
end
```

La traduction de l'affectation est impossible au premier passage, où x et a sont encore inconnus à ce niveau.

Nous allons maintenant donner deux exemples - le premier, d'une configuration que nous traitons en trois passages, mais qu'il serait possible de traiter en deux - le second, celui d'une configuration justifiant le choix d'un fonctionnement en trois passages - : la recherche du mode Algol 68 d'un paramètre formel de type tableau et d'un paramètre formel de type procédure (c'est-à-dire la détermination de la dimension et du mode des paramètres) à partir d'une occurrence d'application du paramètre.

```
begin
     procédure p(A, f); real array A; procédure f;
          begin
          :
          :  A[1, 1] := 3.14;
          :  f(x, y);
          :
          end;
     real x, y;
     :
end
```

La détermination du mode Algol 68 du paramètre formel de type tableau pourrait être effectuée, dès le premier passage, de la façon suivante :

Au premier passage est constitué le dictionnaire d'identificateurs - on en trouvera une description dans un paragraphe suivant et en [GRIF1]. Il est possible d'acquérir dès ce stade le mode Algol 68 du paramètre formel de type tableau, grâce à la technique décrite en [GRIF2]. Cette technique consiste à relier une occurrence d'application d'un identificateur, à son occurrence de définition. Elle revient ici à repousser au niveau de la tête de procédure, à la sortie du corps de procédure, l'information suivante : "le tableau A apparaît avec 2 indices".

Cette même technique est plus difficilement utilisable pour le paramètre formel de type procédure, en raison de la nature de l'information qui est cette fois à passer. Aussi, nous renvoyons au deuxième passage la détermination des modes des paramètres de f. Et ce n'est donc qu'au troisième passage qu'est complétée la déclaration de la procédure p. Ce sera aussi la façon dont nous procéderons pour le paramètre formel de type tableau.

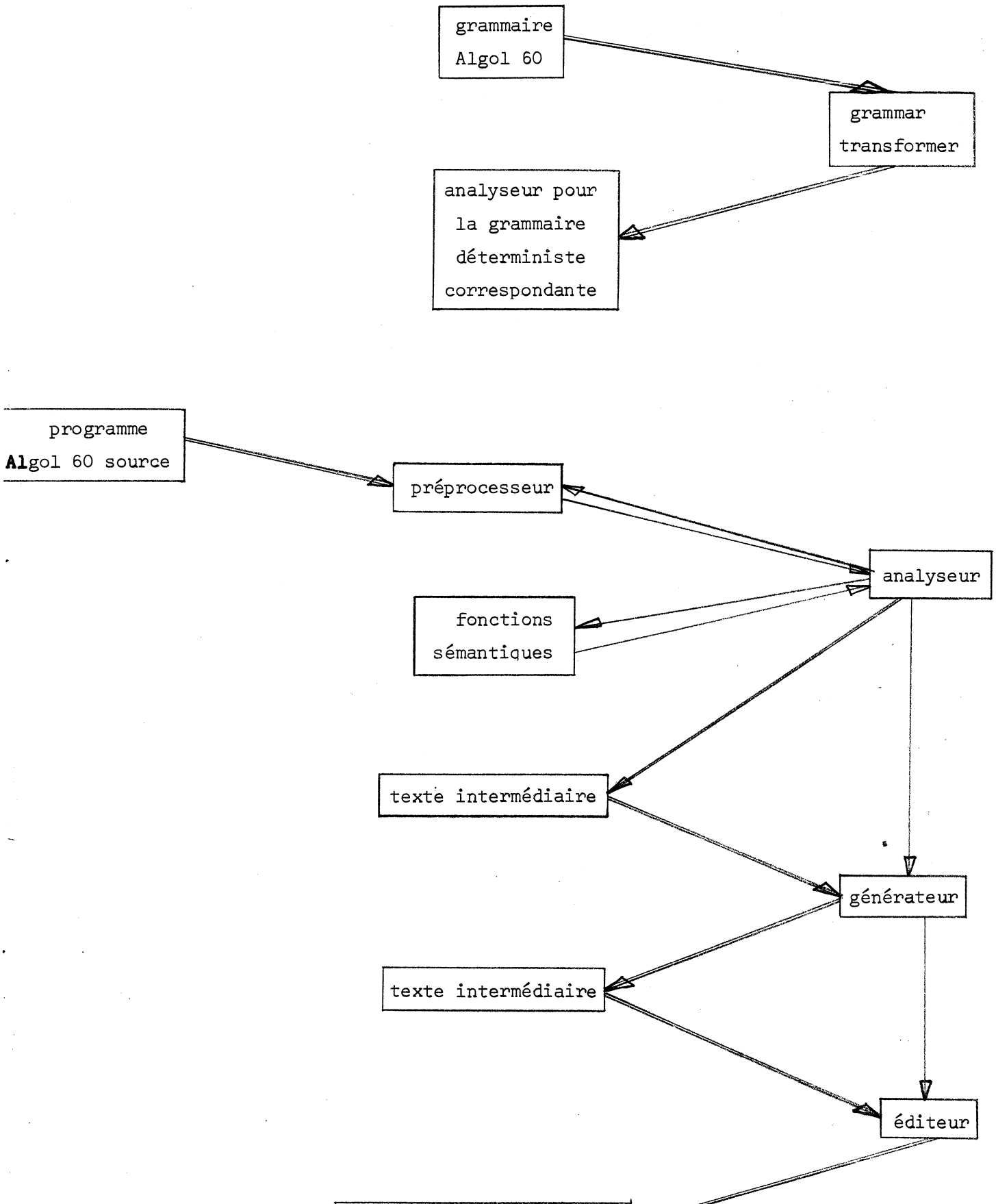
Ce fonctionnement en trois passages étant acquis, nous avons essayé d'organiser en conséquence la traduction, c'est-à-dire d'utiliser ces trois passages de la façon la plus cohérente possible; surtout en ce qui concerne les deux premiers qui sont en fait les deux passages de traduction, le troisième ne comportant qu'un éditeur et se chargeant de récupérer l'information dans diverses chaînes et tables.

Dans le découpage que nous avons choisi, le rôle du premier passage est de retenir toute l'information nécessaire au second pour la traduction, c'est-à-dire un rôle de constitution de tables et de délimitation de l'information. Mais on effectuera dès le premier passage toute traduction purement lexicographique - par exemple : insertion d'un symbole fi en fin de clause conditionnelle ou remplacement de l'instruction vide d'Algol 60 par un symbole skip.

Le deuxième passage récupérera dans ces diverses tables l'information nécessaire à la traduction. Nous repoussons, en particulier, au deuxième passage toute traduction qui nécessite une identification de mode de l'information, c'est-à-dire une recherche en dictionnaire d'identificateurs.

4.2 STRUCTURE DU TRADUCTEUR

La structure du traducteur peut, très schématiquement, être repré-


grammaire
Algol 60
grammar
transformer
analyseur pour
la grammaire
déterministe
correspondante
programme
Algol 60 source
préprocesseur
analyseur
fonctions
sémantiques
texte intermédiaire
générateur
texte intermédiaire
éditeur

On trouvera en annexe la liste des principales routines, ainsi que la grammaire Algol 60 d'entrée. Nous utilisons la version du grammar transformer mise au point par J. Bordier [BORD] avec les facilités de mise en page et d'insertions de commentaires qu'elle offre.

Les différentes routines sont écrites dans un langage de macros [GRIF4] compatible avec l'analyseur produit par le grammar transformer [GRIF3, GRIF5], analyseur auquel peuvent être incorporées des fonctions sémantiques.

Ces fonctions sémantiques sont des routines qui, dans un compilateur, généreraient le pseudo-code et vérifieraient les propriétés du langage que l'on n'aurait pu ou voulu mettre dans la grammaire. Elles réaliseront ici le premier passage du traducteur.

Le préprocesseur a simplement pour rôle de transformer la chaîne programme en Algol 60, en une suite de symboles terminaux de la grammaire. Il prend donc en charge la codification de l'information en une représentation interne, fonction de l'analyseur. En particulier, il attribuera à chaque identificateur, un numéro sur 2 octets; numéro que l'on trouvera souvent mentionné, par la suite, en entrée du dictionnaire d'identificateurs.

Le code intermédiaire produit par le premier passage est ensuite repris par un ensemble de routines - constituant le générateur - pour produire un nouveau code intermédiaire édité au troisième passage.

Nous ne donnerons pas de description des codes intermédiaires, ceux-ci n'ayant aucune structure particulière. Le passage d'Algol 60 à Algol 68 se fait progressivement à chaque passage, en fonction de l'information disponible.

4.3 GESTION DE LA MEMOIRE

4.3.1 Dictionnaire d'identificateurs

Le dictionnaire d'identificateurs est créé au premier passage; chaque déclaration provoquant l'acquisition d'un nouvel enregistrement où sont rangés le numéro de l'identificateur et un nombre variable d'attributs, selon le type de l'identificateur.

Il est géré par le bloc. Au cours du premier passage est constitué le dictionnaire de chaque bloc du programme. C'est la méthode classique en Algol 60.

Les déclarations sont, au fur et à mesure de leurs occurrences, accumulées dans une pile qui est chaînée par blocs. A la fin d'un bloc, le dictionnaire associé est entièrement acquis et se trouve en sommet de pile. Il est alors sauvegardé, et la place qu'il occupait dans la pile récupérée. Cette sauvegarde peut être faite, soit sur mémoire externe (disque par exemple), soit dans une autre partie de la mémoire centrale. Nous avons choisi la deuxième solution, mais sans tirer parti de cette possibilité d'accès aux différents dictionnaires, indépendamment de la structure de bloc; de façon à pouvoir éventuellement utiliser le traducteur sur une machine de plus faible capacité mémoire. L'adresse de sauvegarde est enregistrée dans un tableau, à une position correspondant au numéro lexicographique du bloc.

Au deuxième passage, un tableau contiendra les numéros lexicographiques des blocs entrés - d'où l'accès aux dictionnaires correspondants, et à un enregistrement donné, par un numéro d'identificateur.

4.3.2 Chaînes de code

Plusieurs chaînes de code sont manipulées au cours de la traduction. Elles résident toutes de façon permanente en mémoire centrale, mais on pourrait opérer par sauvegarde sur disque s'il s'avérait nécessaire d'en limiter l'encombrement.

Nous avons déjà vu l'existence des chaînes de texte intermédiaire, générées en sortie du premier et du second passage. Elles seront dites, respectivement, dans la suite : "chaîne code premier passage" et "chaîne code deuxième passage".

La traduction nous amènera souvent, au cours des deux premiers passages, à créer de nouveaux identificateurs. Leurs déclarations seront accumulées dans une chaîne, dite quelquefois "chaîne adjonction de début de programme", et insérées en tête de programme au troisième passage.

Nous verrons apparaître, dans le programme, d'autres points d'accumulation d'insertions que le début du programme. Nous aurons donc aussi besoin de stocker les éléments d'adjonctions qui leur sont relatifs, et qui ne sont pas, eux non plus, immédiatement insérables en chaîne de code. Ces éléments seront rangés dans une chaîne supplémentaire d'adjonctions, dite "chaîne adjonction"; ceux relatifs à un même point d'accumulation d'insertions, chaînés entre eux. A chaque point d'accumulation d'insertions sera associé un enregistrement dictionnaire de contrôle permanent, dont un des champs sera l'adresse

en chaîne adjonction du premier élément d'adjonctions correspondant. Ces différents éléments seront récupérés au troisième passage et insérés aux points correspondants du programme. Cette chaîne adjonction pourra aussi quelquefois servir de simple zone de travail, c'est-à-dire de zone de sauvegarde d'une information non immédiatement utilisable : par exemple, la constitution du mode Algol 68 d'une procédure.

4.3.3 Dictionnaire de contrôle

La traduction utilise en outre deux dictionnaires de contrôle, l'un temporaire, l'autre permanent. Ces dictionnaires ont pour rôle de permettre la sauvegarde et la récupération d'une information non encore utilisable, voire non encore acquise, au moment de la création de l'enregistrement. Ils diffèrent par le domaine de validité de cette information et la méthode utilisée pour sa récupération - qui détermine le mode de gestion du dictionnaire.

4.3.3.1 Dictionnaire de contrôle temporaire

Il est géré en pile. Le domaine d'intérêt d'un tel enregistrement est purement local et sa place toujours récupérée. S'il y a lieu, l'information acquise est insérée en chaîne de code, précédée d'un marqueur spécifique, pour interprétation au passage suivant.

Exemple :

Nous verrons lors du traitement des expressions booléennes que l'opérateur d'implication n'existe pas dans le standard prélude d'Algol 68. Nous l'exprimerons par une clause conditionnelle équivalente.

C'est ainsi que a ⊃ b deviendra dans la traduction :

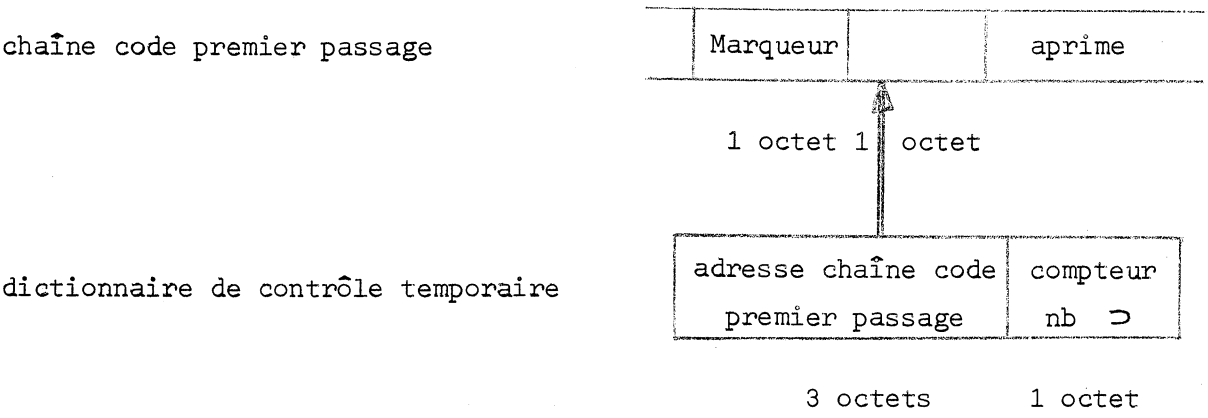
```
if aprime then bprime else true fi
```

et a ⊃ b ⊃ c :

```
if if aprime then bprime else true fi then cprime else true fi
```

où aprime, bprime, cprime désignent les traductions de a, b, c.

Seulement, à l'occurrence de l'opérateur d'implication, il est trop tard pour insérer le symbole if de tête. La traduction devra donc prévoir, dès l'occurrence du a, la possibilité d'un opérateur d'implication à suivre. Pour cela, elle opèrera de la façon suivante :



- insertion d'un marqueur et réservation d'un octet en chaîne code premier passage
- création d'un enregistrement dictionnaire de contrôle temporaire
- comptage du nombre d'opérateurs d'implication
- rangement du nombre d'opérateurs d'implication en chaîne de code et récupération de la place de l'enregistrement dictionnaire de contrôle temporaire.

Au deuxième passage, à l'occurrence du marqueur, est inséré en chaîne code deuxième passage, un nombre de symboles if égal au nombre d'opérateurs d'implication.

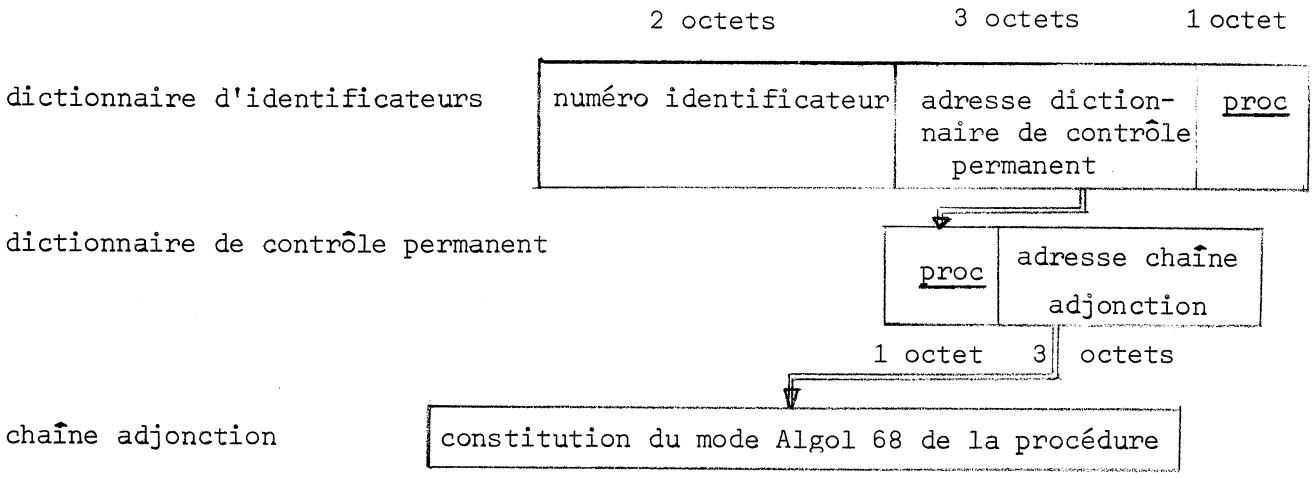
C'est un problème que nous retrouverons souvent dans la suite. Le dictionnaire de contrôle temporaire permettra de sauvegarder l'adresse en chaîne de code d'une éventuelle insertion dont le moment de la prise de décision intervient lexicographiquement trop tard.

4.3.3.2 Dictionnaire de contrôle permanent

L'information qui est d'encombrement variable, est stockée en chaîne adjonction et son adresse rangée dans l'enregistrement. Le domaine d'intérêt de l'enregistrement est inconnu, sinon global. La place ne peut donc en être récupérée. L'accès à un tel enregistrement, pour la récupération de l'infor-

Exemple :

Nous verrons que la traduction d'une déclaration de procédure provoque la création des enregistrements suivants :



et l'insertion d'un marqueur en chaîne de code.

L'enregistrement dictionnaire de contrôle permanent permettra la récupération au troisième passage - stade auquel le dictionnaire d'identificateurs n'existera plus - du mode Algol 68 complet de la procédure, pour la traduction de la déclaration.

DEUXIEME PARTIE

DESCRIPTION DES REGLES DE TRADUCTION

INTRODUCTION

INTRODUCTION

On trouvera, dans cette deuxième partie, l'étude des règles de traduction, qui sont la base de l'écriture du traducteur.

La structure de bloc étant conservée, nous avons pensé que le mieux était d'étudier tour à tour les différents éléments constitutifs du bloc. C'est la raison du découpage de cette partie. Cependant, vu l'importance des déclarations de procédures - non tellement en tant que concept, mais plutôt en raison du travail qu'elles nécessitent - nous avons préféré en faire un chapitre séparé.

D'une façon générale, pour un concept d'Algol 60 donné, dont la traduction n'est pas immédiate, on trouvera :

- la description syntaxique d'Algol 60
- celle de la traduction proposée

ces descriptions étant faites à l'aide de la forme normale de Backus

- la méthode utilisée pour l'implémentation de la traduction, en distinguant la part de chaque passage.

Peut-être aurait-il été préférable de séparer la partie règle, de la partie algorithmique. Nous ne l'avons pas fait, estimant que la solution que nous avons adoptée, facilitait d'avantage l'exposé.

On constatera que cette traduction pourra quelquefois se borner à une simple manipulation lexicographique de l'information ou à une restructuration. Cependant, cette manipulation devra souvent être plus poussée pour rechercher l'information manquante.

CHAPITRE I

DECLARATIONS

CHAPITRE I

DECLARATIONS

1.1 DECLARATION DE TYPE

Une telle déclaration est entièrement traitée au premier passage, et le concept d'Algol 60 intégralement restitué dans la traduction. Les identificateurs dédarés sont enregistrés dans le dictionnaire d'identificateurs, au niveau du bloc courant, selon un enregistrement de la forme suivante.

numéro identificateur	type
2 octets	1 octet

Elle est retransmise sans modification à moins qu'elle soit rémanente.

Algol 60 :

<déclaration de type rémanent> ::= own <type> <liste de type>
<liste de type> ::= <identificateur> | <identificateur>, <liste de type>
<type> ::= real | integer | boolean

traduction :

<traduction déclaration de type rémanent> ::= ref <traduction type> < traduction liste de type>
<traduction liste de type> ::= <identificateur> = <identificateur 1> | < identificateur> = <identificateur 1>, <traduction liste de type>
<traduction type> ::= real | int | bool

Et sont insérées, en début de programme, les déclarations des nouveaux identificateurs créés par la traduction :

<suite de déclarations identificateur 1> ::= <traduction type> <identificateur 1>
<traduction type> <identificateur 1>; <suite de déclarations identificateur 1>

Ces identificateurs ont une représentation externe de la forme IDn où n peut varier de 1 à 65535. L'utilisateur veillera à ce qu'ils n'entrent pas en conflit avec ses propres identificateurs.

Exemple :

Algol 60 : own real x, y
Algol 68 : ref real x = xprime, y = yprime

Et sont sauvegardées en chaîne adjonction de début de programme, pour insertion au troisième passage, les déclarations :

real xprime; real yprime;

On ne pouvait procéder par simple report des déclarations de x et de y en tête de programme, car cela aurait introduit des conflits avec d'autres déclarations de ces identificateurs, qu'il aurait fallu modifier. On le voit aisément sur l'exemple suivant.

```
begin
    begin
        real x, y;
        :
        begin
            own real x, y;        x et y ne seraient plus accessibles
            :                     à ce niveau si leurs déclarations
        end;                      étaient reportées en tête de progra
        :                         me.
    end;
end
```

Remarque à propos de la notion de rémanence

Pour exprimer en Algol 68 le concept d'Algol 60 de rémanence, la première idée qui vient à l'esprit est d'utiliser les générateurs globaux, c'est-à-dire de créer des noms, de portée tout le programme.

En fait, cette démarche est inutilisable, comme le montre l'exemple suivant :

Algol 60

```
(1) begin
    (2) begin
          own real x;
          :
        end;
    end
```

Algol 68

```
(1) begin
    (2) begin
          heap real x;
          :
        end;
    end
```

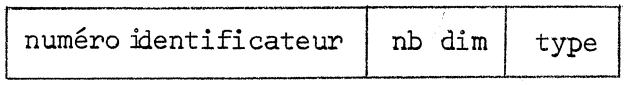
En Algol 60, x n'a pour portée que le bloc (2); le nom qu'il possède est inaccessible à l'extérieur de ce bloc. D'autre part, un nom n'est créé qu'à la première entrée dans le bloc (2). Aux entrées suivantes, on récupère ce nom, avec la valeur qu'il referait à la dernière sortie précédente de ce bloc.

Au contraire, en Algol 68, l'élaboration de la déclaration provoque la création d'un nom, de portée tout le programme, que possède l'identificateur x à l'intérieur du bloc (2). A l'extérieur du bloc (2), cette relation "posseder" cesse d'exister, mais le nom, s'il n'est plus accessible par l'intermédiaire de x, reste cependant parfaitement accessible. De plus, chaque fois que l'on repasse sur la déclaration, un nouveau nom - de portée l'ensemble du programme - est créé. Et x, dans la portée du bloc (2), possède le dernier nom créé, dont la valeur référée est présentement indéfinie.

En conclusion, deux différences fondamentales qui font que ces concepts ne sont pas comparables : un problème d'accès, un problème de valeur référée.

1.2 DECLARATION DE TABLEAU

Les identificateurs sont enregistrés en dictionnaire d'identificateurs selon un enregistrement de la forme suivante.



On rappelle que la dimension d'un tableau fait partie du mode Algol 68 de ce tableau, au même titre que le mode des éléments.

La traduction restitue entièrement le concept d'Algol 60. La déclaration est d'abord restructurée au premier passage; la traduction des expressions arithmétiques qui constituent les bornes n'est effectuée qu'au second passage.

1.2.1 Déclaration non rémanente

Algol 60 :

<déclaration de tableau> ::= array <liste de tableau> | <type> array <liste de tableau>
<liste de tableau> ::= <section de tableau> | <liste de tableau>, <section de tableau>
<section de tableau> ::= <identificateur> [<liste de paires de bornes>] | <identificateur>, <section de tableau>
<liste de paires de bornes> ::= <paire de bornes> | <liste de paires de bornes>, <paire de bornes>
<paire de bornes> ::= <expression arithmétique> : <expression arithmétique>

traduction :

<traduction déclaration de tableau> ::= <traduction section de tableau> | <traduction déclaration de tableau>; <traduction section de tableau>
<traduction section de tableau> ::= [<traduction liste de paires de bornes>] <traduction type> <identificateur> | <traduction section de tableau>, <identificateur>

<traduction liste de paires de bornes> ::= <traduction paire de bornes> |
<traduction liste de paires de bornes>, <traduction paire de bornes>
<traduction paire de bornes> ::= <traduction expression arithmétique> : <traduction expression arithmétique>

Si <type> est omis dans la déclaration d'Algol 60, real est implicite

Exemple :

Algol 60 : array A, B[1 : 10], C[1 : 5]
Algol 68 : [1 : 10] real A, B; [1 : 5] real C

1.2.2 Déclaration rémanente

Le concept d'Algol 60 de tableau rémanent ne fut que peu souvent implémenté dans sa totalité. En effet, les bornes d'un tableau rémanent sont théoriquement à recalculer à chaque nouvelle entrée dans le bloc contenant la déclaration. Les restrictions les plus fréquentes consistèrent à ne calculer ces bornes qu'à la première entrée, ou à imposer qu'elles ne comportent que des constantes.

Le concept lui-même n'existe pas en tant que tel en Algol 68, mais il est possible de l'exprimer à l'aide de celui de tableau à bornes flexibles - concept qui ne possède pas le sous-ensemble. Cependant, en raison de la complexité de sa traduction, nous avons prévu de traduire - séparément - le cas particulier très simple où les bornes sont des constantes, et qui, lui, rentre dans le cadre du sous-ensemble. Quant au cas général, bien qu'il soit pris en charge par le traducteur, le code généré n'est pas accepté par le compilateur du sous-ensemble. C'est donc uniquement dans la perspective du compilateur d'Algol 68 complet, que son étude a été faite.

Algol 60 :

<déclaration de tableau rémanent> ::= own <type> array <liste de tableau>

traduction du cas particulier où les bornes ne comportent que des constantes :

<traduction déclaration de tableau rémanent> ::= <traduction section de tableau rémanent> | <traduction déclaration de tableau rémanent>; <traduction section de tableau rémanent>
<liste de paires de bornes formelles> ::= <vide> | <vide>, <liste de paires de

<traduction section de tableau rémanent> ::= ref [<liste de paires de bornes formelles>] <traduction type> <identificateur> = <identificateur 1> | <traduction section de tableau rémanent>, <identificateur> = <identificateur 1>

Et sont insérées en début de programme les déclarations des nouveaux identificateurs créés par la traduction :

<déclaration identificateur 1> ::= [<traduction liste de paires de bornes>] <traduction type> <liste d'identificateur 1>;
<liste d'identificateur 1> ::= <identificateur 1> | <identificateur 1>, <liste d'identificateur 1>

Exemple :

Algol 60 : own integer array A, B [1 : 10]
Algol 68 : ref [ ] int A = Aprime, B = Bprime

Et sont sauvegardées en chaîne adjonction de début de programme, pour insertion au troisième passage, les déclarations :

[1 : 10] int Aprime, Bprime;

traduction du cas général :

<traduction déclaration de tableau rémanent> ::= <traduction section de tableau rémanent> | <traduction déclaration de tableau rémanent> ; <traduction section de tableau rémanent>
<traduction section de tableau rémanent> ::= [<traduction liste de paires de bornes>] <traduction type> <identificateur 2> | <traduction section de tableau rémanent>, <identificateur 2>; <suite traduction de tableau rémanent>
<suite traduction section de tableau rémanent> ::= <suite d'acquisitions éléments et bornes>; <suite de déclarations identificateur>
<suite d'acquisitions éléments et bornes> ::= <acquisition éléments et bornes> | <acquisition élements et bornes>; <suite d'acquisitions éléments et bornes>

<acquisition élements et bornes> ::= <identificateur 2> [<liste de paires de bornes d'acquisition>] := <identificateur 1> [<liste de paires de bornes d'acquisition>]; <identificateur 1> /= <identificateur 2>

<liste de paires de bornes d'acquisition> ::= <paire de bornes d'acquisition> | <liste de paires de bornes d'acquisition>, <paire de bornes d'acquisition>

<i ème paire de bornes d'acquisition> ::= if i lwb <identificateur 1> <i lwb <identificateur 2> then i lwb <identificateur 2> else i lwb <identificateur 1> fi : if i upb <identificateur 1> <i upb <identificateur 2> then i upb <identificateur 1> else i upb <identificateur 2> fi

où i est un entier prenant les valeurs 1 à n, si n est la dimension du tableau

<suite de déclarations identificateur> ::= <déclaration identificateur> | <déclaration identificateur>; <suite de déclarations identificateur>

<déclaration identificateur> ::= ref [<liste de paires de bornes de déclaration identificateur>] <traduction type> <identificateur> = <identificateur 1>

<liste de paires de bornes de déclaration identificateur> ::= <paire de bornes de déclaration identificateur> | <liste de paires de bornes de déclaration identificateur>, <paire de bornes de déclaration identificateur>

<paire de bornes de déclaration identificateur> ::= flex : flex

avec en début de programme les déclarations des identificateurs de travail, identificateur 1

<déclarations identificateur 1> ::= [<liste de paires de bornes de déclarations identificateur 1>] <traduction type> <liste d'identificateur 1>;

<liste de paires de bornes de déclarations identificateur 1> ::= <paire de bornes de déclarations identificateur 1> | <liste de paires de bornes de déclarations identificateur 1>, <paire de bornes de déclarations identificateur 1>

<paire de bornes de déclarations identificateur 1> ::= skip flex : skip flex

Exemple :

```
Algol 60 : own integer array A, B [m : n]
Algol 68 : [m : n] int Aseconde, Bseconde;
           Aseconde [if 1 lwb Aprime <1 lwb Aseconde then 1 lwb
           Aseconde else 1 lwb Aprime fi : if 1 upb Aprime <1 upb
           Aseconde then 1 upb Aprime else 1 upb Aseconde fi] :=
           Aprime [comment mêmes bornes que la partie gauche comment];
           Aprime := Aseconde; comment instructions analogues pour
           Bprime et Bseconde comment ref [flex : flex] int A =
           Aprime; ref [flex : flex] int B = Bprime
```

et sont sauvegardées en chaîne adjonction de début de programme pour insertion au troisième passage, les déclarations :

```
[skip flex : skip flex] int Aprime, Bprime;
```

commentaires

La traduction de la déclaration d'un tableau rémanent utilise deux tableaux de travail - d'où la création de deux nouveaux identificateurs : identificateur 1 et identificateur 2.

lwb et upb sont deux opérateurs d'Algol 68 fournissant respectivement la borne inférieure et la borne supérieure de la dimension dont le numéro est fourni par le premier opérande, dans le tableau spécifié par le deuxième opérande.

Si en Algol 60 un tableau A est déclaré rémanent, cela s'applique à ses termes, mais non à ses bornes. C'est-à-dire que les bornes d'un tableau déclaré rémanent sont recalculées à chaque nouvelle entrée dans le bloc contenant la déclaration. C'est le rôle du tableau Aprime - déclaré à bornes flexibles - de supporter ces variations des bornes dues au recalcul.

A l'élaboration de la déclaration, le symbole skip sera remplacé par une valeur entière quelconque (fixée par l'implémentation).

L'élaboration de la déclaration de Aseconde provoque l'évaluation des bornes; et Aseconde est créé avec leurs nouvelles valeurs. L'affectation suivante sauvegarde dans Aseconde les éléments de Aprime encore accessibles. La seconde affectation permet à Aprime d'acquérir ces nouvelles bornes et de récupérer les éléments sauvegardés.

En raison de l'utilisation de A dans la suite du programme, il ne reste plus, par une déclaration d'identité, qu'à faire posséder par A la valeur que possède maintenant Aprime.

Nous arrivons ainsi à restituer entièrement la notion de tableau rémanent d'Algol 60.

Au deuxième passage, les expressions arithmétiques qui constituent les bornes seront traduites. Elles devront de plus être analysées pour être éventuellement amenées du mode réel au mode entier, ce transfert n'étant plus automatique en Algol 68. Nous reviendrons ultérieurement sur ce problème lors du traitement des expressions arithmétiques.

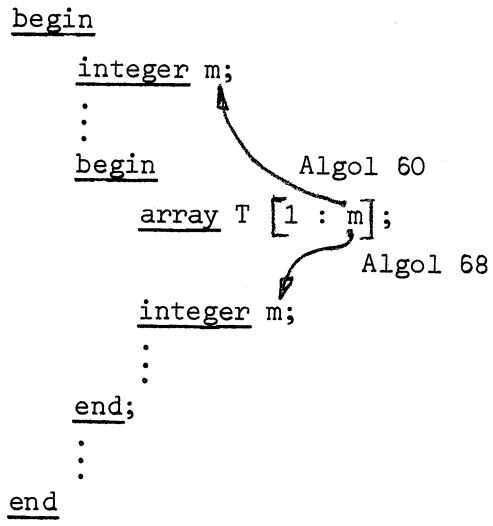
1.2.3 Problèmes posés par les identificateurs figurant dans les bornes

Le problème que nous allons évoquer résulte d'une interprétation particulière qui a été faite du paragraphe du rapport Algol 60 relatif aux expressions arithmétiques qui constituent les bornes des déclarations de tableaux.

Ce paragraphe précise que de telles expressions ne peuvent dépendre que de quantités non locales au bloc contenant la déclaration de tableau. En raison du processus normal d'identification, occurrence d'application - occurrence de définition, il semblerait logique d'en conclure qu'une quantité ayant même représentation qu'un identificateur figurant dans les bornes, ne peut être déclarée au même niveau que le tableau lui-même.

Cependant, certains implémenteurs ont mis à profit la recherche de l'occurrence de définition à partir du bloc englobant pour autoriser l'utilisation à d'autres fins du même identificateur au niveau du tableau.

Les expressions en bornes de déclarations de tableaux ne faisant l'objet dans le rapport Algol 68 d'aucune spécification particulière, les identifications risquent alors de ne plus être en Algol 68 ce qu'elles étaient en Algol 60. C'est ce que montre l'exemple suivant :



Le programme doit donc être restructuré, de façon que cette occurrence d'application identifie en Algol 68 l'occurrence de définition qu'elle identifiait en Algol 60.

Ce problème peut difficilement être traité au premier passage en raison de l'indétermination des positions relatives des déclarations de l'identificateur et de celle du tableau. Il sera résolu au deuxième passage où, au moment du traitement des bornes, les deux occurrences de définition de l'identificateur - si elles existent - seront disponibles en dictionnaire d'identificateurs.

Le traitement consistera tout d'abord à rechercher si une configuration telle que celle ci-dessus se présente; c'est-à-dire à rechercher s'il existe dans les bornes un identificateur déclaré au même niveau que le tableau lui-même. Si oui, cette occurrence de définition serait celle qui serait identifiée en Algol 68. La recherche de l'occurrence de définition identifiée en Algol 60 se poursuit dans un bloc englobant.

Alors, le traitement consistera en :

1) remplacer m par un nouvel identificateur mprime dans la borne du tableau

2) insérer dans le bloc du m identifié en Algol 60, la déclaration suivante

ref <traduction type> mprime = m

qui fait posséder par mprime le nom que possède l'identificateur m identifié en Algol 60 - réalisant ainsi une déclaration de synonymie.

La traduction de l'exemple précédent sera :

```
begin
    int m;
    ref int mprime = m;
    .
    .
    begin
        [1 : mprime] real T;
        int m;
        .
        .
    end;
    .
    .
end
```

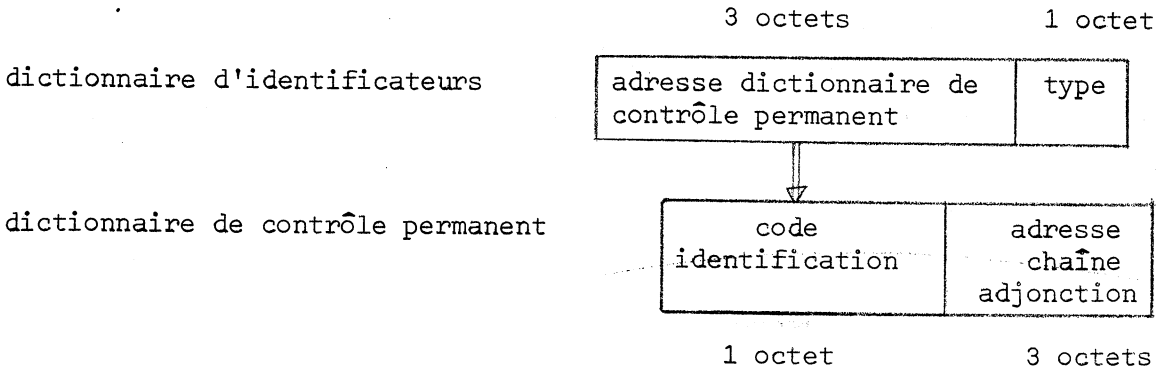
La substitution d'identificateurs est faite pendant le traitement des bornes. Mais il est trop tard pour l'insertion de la déclaration de mprime. Aussi, elle est sauvegardée en chaîne adjonction et n'est insérée qu'au troisième passage.

Cet identificateur peut, en particulier, être un identificateur de tableau ou un identificateur de procédure; c'est-à-dire une quantité de description de mode relativement compliquée. Il peut même être un paramètre formel; un niveau de bloc supplémentaire, englobant le corps de procédure, doit alors être introduit.

Nous arrivons ainsi, par l'utilisation d'un nouvel identificateur, - mais sans encombrement mémoire supplémentaire -, à restituer en Algol 68 l'identification qui était faite en Algol 60.

programmation

A chaque bloc est associé un enregistrement dictionnaire d'identificateurs :



Le dictionnaire d'identificateurs est géré par bloc. Cet enregistrement sert, d'abord, lors de la recherche de l'occurrence de définition d'un identificateur, de marqueur de fin de bloc courant; donc d'indicateur de recherche à poursuivre dans le bloc englobant. Il permet donc de tester les remontées d'un bloc au bloc englobant, c'est-à-dire, dans le cas particulier qui nous intéresse ici, de tester si un identificateur figurant dans les bornes d'une déclaration de tableau est, ou non, déclaré au même niveau que le tableau lui-même, et nécessite donc, ou non, le traitement précédent.

L'enregistrement dictionnaire de contrôle permanent permet la récupération au troisième passage, sur marqueur, des adjonctions - chaînées entre elles - en chaîne adjonction. Ces adjonctions correspondent aux insertions des déclarations des nouveaux identificateurs tels que mprime.

1.3 DECLARATION D'ETIQUETTE

Une déclaration d'étiquette est une déclaration par contexte, aussi bien en Algol 60 qu'en Algol 68. Elle est retransmise sans modification dans le programme en Algol 68, sortie du traducteur.

L'enregistrement dictionnaire d'identificateurs créé est de la forme suivante :

numéro identificateur	niveau définition	niveau accès	indicateur	type
2 octets	3 octets	3 octets	2 octets	1 octet

Nous verrons, lors du traitement des expressions de désignation, l'intérêt des 2°, 3° et 4° champs de l'enregistrement. Ces informations seront utilisées pour le traitement de la possibilité - qui existait en Algol 60, mais qui n'existe plus en Algol 68 -, pour une instruction goto, de se renvoyer à l'intérieur d'une instruction composée ou conditionnelle.

Le branchement direct n'étant plus légal, nous le simulerons, dans la traduction, par un branchement indirect à l'aide d'un indicateur booléen.

Nous définirons aussi une structure de "niveau d'étiquette", qui permettra de tester si un tel conflit de portée, en Algol 68, entre une occurrence d'application d'une étiquette et son occurrence de définition en Algol 60, se présente, et, si oui, de mettre en place le chaînage correspondant au branchement indirect.

1.4 DECLARATION D'AIGUILLAGE

La notion d'aiguillage n'existe plus en Algol 68.

Ce concept revient à grouper un certain nombre d'expressions de désignation auxquelles on peut ultérieurement accéder pour exécution, sur la donnée d'un entier qui est le rang de l'expression dans la liste.

Chaque expression de désignation de la liste représente en fait un branchement potentiel à une certaine étiquette. Nous aurons donc les mêmes problèmes de portée que ceux que nous avons signalés au paragraphe précédent, et qui seront traités au paragraphe des expressions de désignation.

Nous traduirons une déclaration d'aiguillage par une déclaration de procédure à un paramètre entier; ce paramètre étant le rang dans la liste de l'expression de désignation que l'on désire sélecter. De là découlent les contraintes que nous avons imposées sur les portées des quantités apparaissant dans les expressions d'une liste d'aiguillage.

En effet, on considère parfois qu'il suffit que les identificateurs contenus dans une expression de désignation de la liste soient tous déclarés, seulement au moment de l'utilisation de l'expression, et non obligatoirement au moment de la déclaration de l'aiguillage.

La traduction par une déclaration de procédure nous oblige à imposer que les quantités apparaissant dans ces expressions - donc quantités non locales au corps de procédure - soient toutes déclarées en tête du bloc, ou d'un bloc englobant, la déclaration de la procédure, c'est-à-dire de l'aiguillage.

L'enregistrement dictionnaire d'identificateurs créé est de la forme suivante :

2 octets 1 octet

Algol 60 :

<déclaration d'aiguillage> ::= switch <identificateur> := <liste d'aiguillage>
<liste d'aiguillage> ::= <expression de désignation> | <liste d'aiguillage>,
<expression de désignation>

traduction :

<traduction déclaration d'aiguillage> ::= proc <identificateur> = (int <identificateur 1>) : case <identificateur 1> in <traduction liste d'aiguillage> out skip esac

<traduction liste d'aiguillage> ::= <traduction expression de désignation> |
<traduction liste d'aiguillage>, <traduction expression de désignation>

Exemple :

Algol 60 : switch aig := r, s, t

Algol 68 : proc aig = (int aigprime) : case aigprime in r, s, t out skip esac

La sélection du symbole skip correspond à un indicateur d'aiguillage d'Algol 60 non défini. L'instruction goto correspondante est alors équivalente à une instruction vide, comme le spécifie le rapport Algol 60.

Nous arrivons donc de cette façon à retransmettre en Algol 68 le concept d'Algol 60 d'aiguillage. Le cadre de la traduction de la déclaration d'aiguillage en déclaration de procédure est mis en place au premier passage; les expressions de désignation de la liste ne sont traduites qu'au second.

CHAPITRE II

DECLARATION DE PROCEDURE

CHAPITRE II

DECLARATION DE PROCEDURE

2.1 INTRODUCTION

Par "procédure", nous entendons aussi bien "fonction procédure" que "procédure propre", telles que les distingue Algol 60. Nous ne ferons la distinction que lorsque cela sera nécessaire pour la traduction.

Une déclaration de procédure d'Algol 60 est traduite en une nouvelle déclaration de procédure dans laquelle les modes Algol 68 des paramètres sont complètement décrits.

Bien que nous imposions que dans la déclaration d'Algol 60 figurent les spécifications de tous les paramètres, - contrainte qui est une contrainte imposée par de nombreux compilateurs d'Algol 60 -, cette information n'est pas suffisante pour les paramètres formels de type tableau et procédure. Car, de la même façon que le mode Algol 68 d'un tableau comprend le nombre de paires de bornes, celui d'une procédure comprend les modes Algol 68 de ses paramètres. Dans ces deux cas, nous devrons trouver l'information manquante. Cette recherche pourra poser de nombreuses difficultés. Elle pourra même ne pas aboutir - en un sens que nous définirons. Nous serions alors dans l'impossibilité de traduire.

De plus, Algol 68 ne dispose que d'un seul mode d'appel des paramètres, à l'aide duquel nous devrons exprimer les appels d'Algol 60, par nom et par valeur.

Il est évident que le concept d'Algol 60 ne peut être restitué dans son intégralité par la traduction, en raison de la description beaucoup plus précise des paramètres qui est faite dans la déclaration d'Algol 68. Les correspondances paramètre formel - paramètre effectif permises à l'appel de la procédure seront donc beaucoup plus strictes en Algol 68 qu'elles ne l'étaient en Algol 60.

Nous verrons pour chaque type de paramètre comment est effectuée sa traduction, selon le mode d'appel, et les limitations de celle-ci, ainsi que celles qui en résultent sur les paramètres effectifs qui peuvent lui correspondre.

2.2 ORGANISATION DE LA TRADUCTION

a) Déclaration de procédure avec paramètre

Algol 60

<déclaration de procédure avec paramètre> ::= <type procédure> procédure
<tête de procédure> <instruction>
<type procédure> ::= <type> | <vide>
<tête de procédure> ::= <identificateur> (<liste de paramètres formels>); <partie valeur> <partie spécification>
<liste de paramètres formels> ::= <paramètre formel> | <liste de paramètres formels> <délimiteur de paramètre> <paramètre formel>
<délimiteur de paramètre> ::= , |) <chaîne de lettres> : (

traduction

<traduction déclaration de procédure avec paramètre> ::= proc <identificateur> = (<traduction liste de paramètres formels>) <traduction type procédure> : <traduction instruction>
<traduction liste de paramètres formels> ::= <traduction paramètre formel> | <traduction liste de paramètres formels> <traduction délimiteur de paramètre> <traduction paramètre formel>
<traduction délimiteur de paramètre> ::= , | comment <chaîne de lettres> comment,
<traduction type procédure> ::= <traduction type> | <vide>

b) Déclaration de procédure sans paramètre

Algol 60

<déclaration de procédure sans paramètre> ::= <type procédure> procédure
<identificateur>; <instruction>

traduction

<traduction déclaration de procédure sans paramètre> ::= proc <traduction type procédure> <identificateur> = <traduction type procédure> : <traduction instruction>

On notera la différence de traduction, le cas (b) ne se déduisant pas simplement du cas (a) par l'absence de la partie paramètres formels. La différence réside dans le fait que, dans le cas (a), le paramètre effectif de la déclaration d'identité est une dénotation de routine. Il est alors possible, dans le langage étendu, d'opérer des simplifications sur l'écriture du mode du paramètre formel partie gauche. Au contraire, dans le cas (b), le paramètre effectif n'est pas une dénotation de routine (qui a toujours des paramètres). Il n'est pas "a priori" d'un mode commençant par proc, mais il est amené à ce mode "a posteriori" par une contrainte dite de "proceduring". C'est ce qui explique l'absence d'extension dans ce cas (b), alors que, dans le cas précédent, le mode de l'identificateur de procédure peut être - lexicographiquement - déduit de celui de la dénotation de routine paramètre effectif.

On remarquera aussi que, dans la traduction, on ne fait aucune différence entre une fonction procédure et une procédure propre. En Algol 68, une valeur pourra être délivrée. Cette valeur, définie dynamiquement, sera, ou non, prise en considération, selon le contexte où figure l'appel.

Au premier passage est mis en place le cadre de la déclaration en Algol 68, tel qu'il vient d'être défini. Les premier et deuxième passages consistent en l'acquisition du mode Algol 68 de la procédure, c'est-à-dire de ceux de ses paramètres. Ce n'est qu'au troisième passage, lorsque toute l'information nécessaire a été acquise, qu'est complété le cadre de la déclaration.

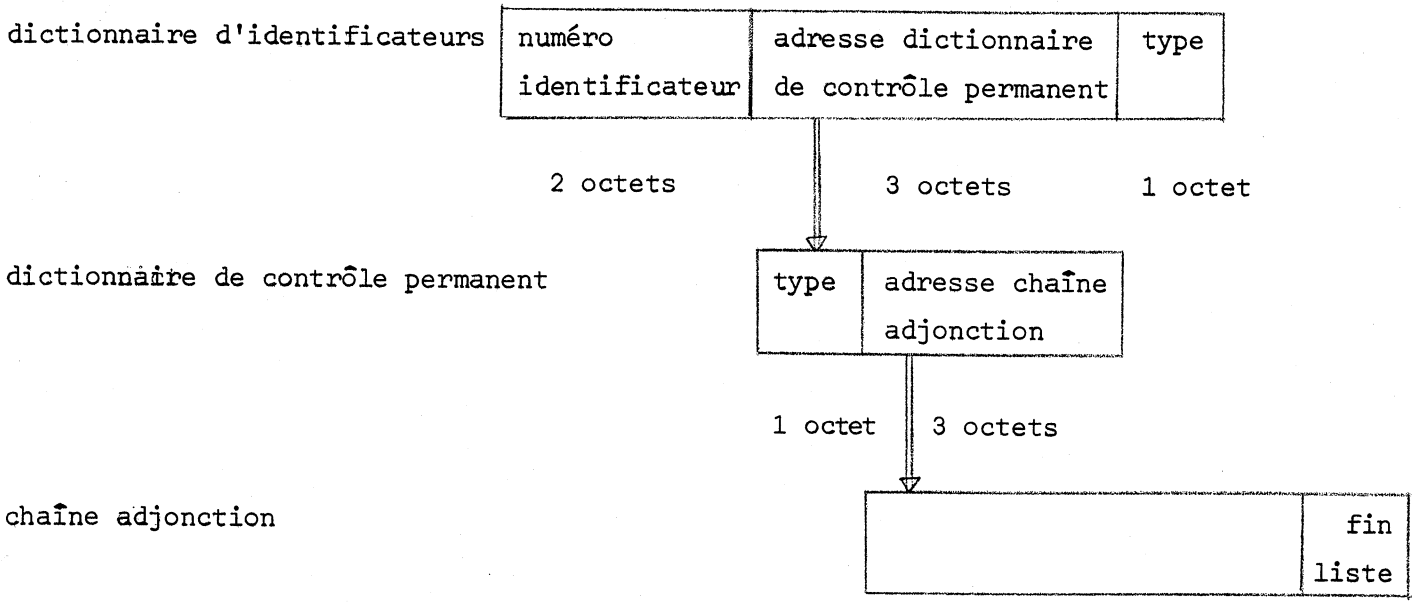
2.3 TRADUCTION D'UN DELIMITEUR DE PARAMETRE

La notion de délimiteur de paramètre offre en Algol 60 la possibilité d'insérer des commentaires entre les paramètres, aussi bien formels qu'effectifs. Ce commentaire est retransmis, comme tous ceux qui figurent dans le programme en Algol 60 source.

On notera dès à présent qu'en Algol 68, un commentaire est syntaxiquement parenthésé par deux symboles comment, ce qui, par rapport aux conventions d'Algol 60, facilite sa délimitation dans un programme source.

2.4 ENREGISTREMENTS DICTIONNAIRE CREES

a) Procédure propre

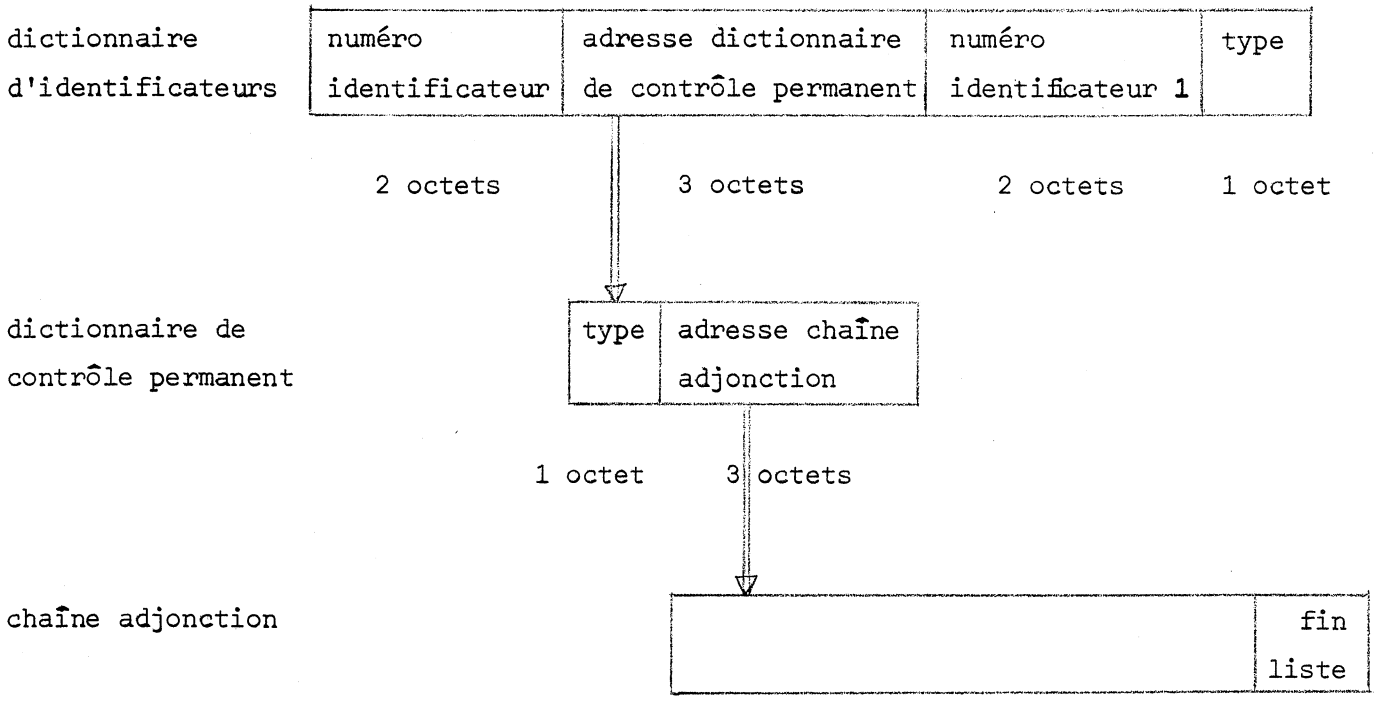


Pour une procédure propre avec paramètre est aussi créé un enregistrement dictionnaire de contrôle temporaire de la forme suivante :

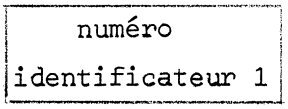
1 octet

Cet enregistrement est utilisé pour introduire un niveau de bloc englobant le corps de procédure, si celui-ci n'est ni un bloc, ni une instruction composée et que des paramètres sont appelés par valeur.

b) Fonction procédure



Et aussi créé l'enregistrement dictionnaire de contrôle temporaire suivant :



2 octets

Cet enregistrement n'existe qu'au premier passage. Il duplique un champ de l'enregistrement dictionnaire d'identificateurs, qui est le numéro d'une variable de travail que nous associerons à l'identificateur de fonction procédure. Mais il a néanmoins été créé pour des raisons de facilités d'accès : le dictionnaire de contrôle temporaire étant géré en pile, l'information courante est immédiatement disponible et cela évite une recherche dans le dictionnaire d'identificateurs - en cours de construction à ce premier passage.

Les enregistrements en chaîne adjonction sont destinés à fournir les modes Algol 68 des paramètres. Nous verrons ce que sont ces enregistrements, au fur et à mesure, pour chaque type de paramètre.

On remarquera que l'enregistrement dictionnaire de contrôle permanent fournit alors le mode Algol 68 de l'identificateur de procédure : mode de la valeur délivrée, et ceux des paramètres. Cet enregistrement est récupéré, sur marqueur, au troisième passage et permet alors, si les diverses recherches ont été concluantes, de compléter la déclaration en Algol 68 de la procédure.

2.5 PROBLEMES PROPRES AUX FONCTIONS PROCEDURE

Une fonction procédure est,normalement,destinée à définir la valeur d'un indicateur de fonction, qui pourra ultérieurement être utilisé dans les expressions. Pour que l'indicateur de fonction soit effectivement défini, doit apparaître dans le corps de procédure, au moins une affectation à l'identificateur de procédure - la valeur de l'indicateur étant la dernière valeur dynamiquement affectée.

Cette façon de procéder n'existe plus en Algol 68 où, les notions d'instruction et d'expression ayant été unifiées, le corps de procédure délivre ou non, de lui-même, une valeur. La traduction consistera donc à faire en sorte que les valeurs délivrées en Algol 60 et en Algol 68 soient les mêmes. Ces deux valeurs sont définies dynamiquement. La solution adoptée utilise une variable de travail, du mode spécifié par le déclarateur de type qui est le premier symbole de la déclaration de procédure en Algol 60, pour mémoriser les différentes valeurs potentielles; la dernière étant passée comme "valeur délivrée par l'élaboration de l'indicateur de fonction".

Le procédé de traduction peut être exprimé par le paragraphe suivant :

1°) La déclaration de la variable de travail est insérée en tête du corps de procédure. Si ce corps de procédure n'est ni un bloc, ni une instruction composée, un nouveau niveau de bloc est créé.

2°) Dans le corps de procédure, les affectations à l'identificateur de procédure sont remplacées par des affectations à la variable de travail.

3°) Une occurrence d'application de cette variable, précédée d'un point-virgule, est insérée avant le end (créé ou déjà existant) de fin du corps de procédure.

Exemple :

Algol 60 :

```
real procédure f(x, y); real x, y;
  begin
    .
    .
    .   f := x + 3.14;
    .
    .
    .
  end
```

Algol 68 :

```
----------------------------------
  begin
    real fprime;
    .
    .
    .  fprime := x + 3.14;
    .
    .
    .
    .         ;
    .
    fprime
  end
```

Les insertions de la déclaration en tête et d'une occurrence d'application en fin du corps de procédure, de cette variable de travail, sont faites au premier passage. Les substitutions des parties gauches des affectations internes ne sont effectuées qu'au deuxième passage.

Cette variable de travail associée à l'identificateur de fonction procédure est sauvegardée - pour le second passage - dans l'enregistrement dictionnaire d'identificateurs. Elle est aussi, comme nous l'avons vu, mémorisée en dictionnaire de contrôle temporaire.

Justification

Le fait que la déclaration de procédure d'Algol 60 commence par un déclarateur de type et non directement par le symbole procédure n'implique absolument pas que le corps de procédure doive contenir des affectations à l'identificateur de procédure, ni que la procédure ne puisse être utilisée qu'en indicateur de fonction.

Seulement, pour des raisons de simplicité, la traduction est systématique. Elle ne fait en cela que refléter ce qui se passe de façon sous-jacente dans le compilateur d'Algol 60 : la création d'un accumulateur résultat. Cette création, systématique, est le fait de la déclaration et non des utilisations ou de la présence d'affectations internes. De telle sorte que la valeur référée restera indéfinie en l'absence d'affectations internes et qu'elle sera ignorée pour une utilisation en instruction procédure.

2.6 TRADUCTION D'UNE OCCURRENCE DE DEFINITION DE PARAMETRE FORMEL

2.6.1 Remarques sur les modes d'appel des paramètres

2.6.1.1 Présentation de l'appel d'Algol 68

Algol 68 ne dispose que d'un seul mode d'appel des paramètres, à l'aide duquel nous devons exprimer les appels par nom et par valeur d'Algol 60. L'interprétation de ce mode d'appel est la suivante :

Un paramètre formel est équivalent à une partie gauche de déclaration d'identité; l'appel fournissant, par le paramètre effectif, la partie droite correspondante. C'est-à-dire que, en Algol 68, l'appel d'une procédure provoque, préalablement à l'élaboration du corps de procédure lui-même, celle de déclarations d'identité de la forme :

paramètre formel = paramètre effectif.

par lesquelles l'identificateur d'un paramètre formel est fait posséder la valeur du paramètre effectif correspondant.

Comme dans toute déclaration d'identité, le paramètre effectif est en position "strong" - ce qui détermine les tolérances permises par le langage dans la correspondance paramètre formel - paramètre effectif.

Exemple :

```
proc P = (ref real x) : begin ..... end
```

Le paramètre formel x étant destiné à posséder un nom, il peut figurer en partie gauche d'affectation dans le corps de procédure. Le paramètre effectif dont il possèdera la valeur doit être du mode a posteriori ref real; ceci implique qu'il ne peut être une expression (au sens Algol 60). Ce paramètre effectif ne sera évalué qu'une fois : à l'entrée du corps de procédure, le nom correspondant est fait possédé par le paramètre formel - toutes les références ultérieures se faisant à cette adresse. On a ainsi, par rapport aux quantités manipulées en Algol 60, une sorte d'appel par nom, mais sans effet de bord, de variable - c'est-à-dire, un "appel par référence".

Exemple :

```
proc P = (real x) : begin ..... end
```

Le paramètre effectif, de mode a posteriori real, est élaboré à l'entrée du corps de procédure. La valeur réelle niveau zéro obtenue est faite possédée par le paramètre formel x, de telle sorte que les références ultérieures se feront uniquement à cette valeur. On a ainsi une sorte d'appel par valeur, mais sans possibilité de figurer en partie gauche d'affectation.

2.6.1.2 Simulation des modes d'appel d'Algol 60

Par la diversité possible du mode du paramètre formel qui détermine le mode a posteriori du paramètre effectif, le mode d'appel d'Algol 68 est plus puissant que ceux dont on disposait en Algol 60. Mais il n'est l'expression immédiate d'aucun. Nous pourrons cependant restituer les modes d'appels d'Algol 60, dans la traduction, et ceci de la façon suivante :

Dans un appel par nom d'Algol 60, le paramètre effectif est réévalué à chaque occurrence du paramètre formel dans le corps de procédure. Pour éviter l'élaboration dès l'entrée du corps de procédure, la traduction consistera en un "proceduring" du paramètre effectif; c'est-à-dire que le paramètre formel sera fait posséder la routine (et non la valeur) qu'est le paramètre effectif. Ainsi, chaque occurrence, et uniquement une occurrence, d'application du paramètre formel dans le corps de procédure provoquera le "deproceduring", c'est-à-dire l'élaboration de la routine qu'il possède. Nous examinerons pour chaque type de paramètre le problème du mode de la valeur qui doit alors être délivrée.

Au contraire, l'appel par valeur revient à la création et à la seule manipulation d'une variable locale au corps de procédure. On provoquera donc d'emblée l'élaboration du paramètre effectif, et on fera référer cette valeur par une variable locale au corps de procédure, de même nom que le paramètre formel de la déclaration d'Algol 60.

On dispose de trois niveaux pour différencier les appels par nom et par valeur :

(1) le mode du paramètre formel
(2) le corps de procédure
(3) l'appel de la procédure.

Considérons l'exemple suivant :

```
begin
    real a;
    procédure f(p); procédure p;
       begin
        :
        :  p(a);
       end;
    procédure g(x); value x; real x;
       begin
        :
        :
       end;
    procédure h(y); real y;
       begin
        :
        :
       end;
    :
    :
     f(g);
    :
    :
     f(h);
    :
    :
end
```

Les modes d'appel des paramètres d'un paramètre formel de type procédure, tel que p, ne peuvent être déterminés, car ils sont fonctions de ceux des paramètres du paramètre effectif correspondant à p dans un appel de f. Et nous ne saurions imposer dans une correspondance de paramètres de type procédure, celle des modes d'appel de leurs paramètres.

La différenciation des appels par nom et par valeur ne peut donc être faite au niveau du mode du paramètre formel - sinon elle introduirait des conflits au niveau de la correspondance formel - paramètre effectif, pour un paramètre de type procédure. Elle ne peut pas l'être non plus au niveau de l'appel de la procédure, en raison, cette fois, des appels, des paramètres formels de type procédure. Cependant, pour un appel d'une procédure non paramètre formel, nous utiliserons le fait qu'un paramètre est appelé par valeur, pour permettre le maximum de latitude dans la correspondance paramètre formel - paramètre effectif.

La différenciation ne peut donc être faite qu'au niveau du corps de procédure. Le choix qui a été fait correspond à un appel par nom, systématiquement, au niveau du mode du paramètre formel - quitte à revenir à un appel par valeur à l'intérieur du corps de procédure.

Dans ce mode, on trouvera, souvent, au moins l'un des éléments fondamentaux suivants :

proc, pour n'évaluer le paramètre effectif qu'à l'occurrence - chaque occurrence - du paramètre formel dans le corps de procédure, et ainsi retransmettre dans la traduction les effets de bord possibles
ref, pour l'apparition possible en partie gauche d'affectation dans le corps de la procédure en Algol 68.

Nous verrons lors du traitement des appels de procédure, les contraintes que le choix du mode du paramètre formel implique sur les paramètres effectifs correspondants - notamment en fonction dans tolérances d'Algol 60 dans ce domaine.

2.6.2 Paramètre formel de type réel, entier ou booleen

Les parties valeur et spécification fournissent toute l'information nécessaire à la traduction.

Au premier passage, l'identificateur - paramètre formel est retransmis dans la liste des paramètres formels, tandis que sont créés les enregistrements suivants :

dictionnaire d'identificateurs

numéro identificateur	type
2 octets	1 octet

enregistrement en chaîne adjonction dans le mode de la procédure

type
1 octet

Au troisième passage est effectuée la traduction du paramètre formel, selon la syntaxe suivante :

2.6.2.1 Appel par nom

Algol 60 :

<partie spécification> ::= < type> <liste d'identificateurs>;
<type> ::= real | integer | boolean
<liste d'identificateurs> ::= <identificateur> | <liste d'identificateurs>,
<identificateur>

traduction :

<traduction paramètre formel> ::= proc ref <traduction type> <identificateur>
<traduction type> ::= real | int | bool

Exemple :

Algol 60 : procédure P(x); real x; .....;
Algol 68 : proc P = (proc ref real x) : .....;

A l'appel de la procédure, le paramètre effectif, qui sera de mode a priori ref real, - nous verrons au paragraphe des appels de procédure, le traitement des paramètres effectifs - sera procéduré. Le paramètre formel x possèdera la routine obtenue, routine qui sera élaborée à chaque occurrence du paramètre formel dans le corps de procédure, fournissant le nom du paramètre effectif. Nous restituons donc bien ainsi l'appel par nom d'Algol 60.

2.6.2.2 Appel par valeur

Algol 60 :

<partie valeur> ::= value <liste d'identificateurs>;
<partie spécification> ::= <type> <liste d'identificateurs>;

traduction :

<traduction paramètre formel> ::= proc ref <traduction type> <identificateur 1>

Est, de plus, insérée, au début du corps de procédure, la déclaration avec initialisation suivante :

<traduction type> <identificateur> := <identificateur 1>;

Une nouvelle structure de bloc est créée si le corps de procédure (déjà fait pour une fonction procédure) n'est ni un bloc, ni une instruction composée.

Exemple :

Algol 60 : procédure P(x); value x; real x; begin ..... end
Algol 68 : proc P = (proc ref real xprime) : begin real x := xprime; ..... end

On a vu que l'on ne peut différencier les appels par nom et par valeur au niveau du mode du paramètre formel, et que c'est l'appel par nom qui détermine le choix du mode du paramètre formel.

Le paramètre effectif - de mode a priori ref real -, qui constitue la routine possédée par la variable de travail xprime, sera élaboré à la première occurrence de xprime dans le corps de procédure, initialisant ainsi x - paramètre formel de la déclaration d'Algol 60, et qui est de mode ref real - à la valeur qu'il réfère. Nous restituons donc bien aussi l'appel par valeur d'Algol 60.

2.6.3 Paramètre formel de type étiquette

En Algol 60, les deux modes d'appel, par nom et par valeur, peuvent être utilisés - l'appel par valeur consistant simplement, dans le contexte d'un tel paramètre, à éviter les risques d'effets de bord. Nous avons vu que nous ne pouvions, dans la traduction, différencier les deux modes d'appel, qu'au niveau du corps de procédure. La technique utilisée, pour simuler, dans la traduction, l'appel par valeur d'Algol 60, consistait à provoquer l'élaboration du paramètre effectif dès l'entrée dans le corps de procédure, et à faire référer la valeur obtenue par une variable locale au corps de procédure, de même nom que le paramètre formel de la déclaration d'Algol 60. Cette démarche, ici, n'a pas de sens, car, en Algol 68, un identificateur d'étiquette ne possède pas de valeur.

En conséquence, nous considérerons qu'un paramètre de type étiquette est toujours appelé par nom. Nous verrons lors du traitement des appels de procédure, quelles en sont les conséquences pour le paramètre effectif, et comment il serait cependant possible de réaliser un appel par valeur, partiel.

Le traitement est identique à celui d'un paramètre formel de type réel, entier ou booléen, et les enregistrements créés, analogues. Quant à la traduction, elle s'effectue selon la syntaxe suivante.

Algol 60 :

<partie valeur> ::= <vide> | value <liste d'identificateurs>;
<partie spécification> ::= label <liste d'identificateurs>;

traduction :

<traduction paramètre formel> ::= proc <identificateur>

Exemple :

Algol 60 : procédure P(e); label e; ..... ;
Algol 68 : proc P = (proc e) : ..... ;

Un paramètre formel de type étiquette est ainsi traduit en une procédure void sans paramètre. Toute occurrence d'application du paramètre formel dans le corps de procédure correspond à l'appel de cette procédure, provoquant donc l'élaboration de l'expression désignation - paramètre effectif, et le branchement à une certaine étiquette.

2.6.4 Paramètre formel de type aiguillage

Nous avions traduit une déclaration d'aiguillage par une déclaration de procédure void à un paramètre entier. La traduction d'un paramètre formel de type aiguillage s'en déduit immédiatement.

Le traitement et les enregistrements créés sont analogues à ceux des paramètres traités dans les paragraphes précédents. Seul l'appel par nom est possible, et la traduction est faite selon la syntaxe suivante :

Algol 60 :

<partie spécification> ::= switch <liste d'identificateurs>;

traduction :

<traduction paramètre formel> ::= proc (int) <identificateur>

Exemple :

Algol 60 : procédure P(aig); switch aig; ..... ;
Algol 68 : proc P = (proc (int) aig) : ..... ;

2.6.5 Paramètre formel de type chaîne

Il n'existe pas en Algol 60 de mode de base caractère. Aucune véritable manipulation de chaîne n'est donc possible. Plus exactement, une chaîne ne peut apparaître dans un programme, qu'en paramètre effectif de procédure.

Nous reviendrons dans le chapitre traitant des expressions, sur la syntaxe Algol 60 des chaînes, et l'expression de ce concept en Algol 68.

Le traitement de ce paramètre est identique à ceux des paramètres des paragraphes précédents. Quant à la traduction, elle est identique à la spécification d'Algol 60.

Exemple :

Algol 60 : procédure P (chain); string chain; ..... ;
Algol 68 : proc P = (string chain) : ..... ;

Nous verrons qu'il existe, en Algol 68, des dénotations distinctes, de caractère et de chaîne, correspondant à des valeurs de modes respectifs, char et string, - une dénotation de chaîne étant définie comme un tableau unidimensionel de caractères.

C'est une distinction dont nous n'aurons pas à nous préoccuper dans la traduction. En effet, une chaîne - au sens Algol 60 du terme -, sera, en tant que paramètre effectif - donc partie droite de déclaration d'identité - toujours en contexte strong. De telle sorte que si le paramètre effectif de type chaîne d'Algol 60 correspond en fait à une dénotation de caractère - donc à une valeur de mode Algol 68 char -, il sera automatiquement converti au mode string par une coercion dite de "rowing" (qui le transformera en une valeur multiple caractère, de dimension 1, bornes inférieure et supérieure 1).

2.6.6 Paramètre formel de type tableau

La partie spécification ne fournit que le mode des éléments. Nous devons donc, sans peine de ne pouvoir traduire, déterminer la dimension d'un paramètre formel de type tableau - information que n'apporte pas la tête de procédure.

Cette recherche est effectuée au deuxième passage - passage où peuvent être faites, grâce au dictionnaire d'identificateurs, toutes les identifications nécessaires. Le processus d'identification des paramètres formels est identique à celui des autres quantités : un paramètre formel a comme portée, le corps de procédure (à moins de redéclarations internes, totale ou partielles).

Au premier passage, le paramètre formel est recopié dans la liste de paramètres formels. Si la dimension a pu être acquise au deuxième passage, sa traduction est complétée au troisième, selon une syntaxe que nous définirons.

2.6.6.1 Acquisition de la dimension

Cette dimension peut, a priori, être déterminée à partir de deux types de source différents :

- à partir des occurrences d'application du paramètre formel dans le corps de procédure
- à partir des occurrences d'appel de la procédure, par examen du paramètre effectif correspondant.

La politique que nous avons adoptée est la suivante. Nous estimons que la dimension doit pouvoir être déterminée à partir du corps de procédure; les occurrences d'appel de la procédure ne devant être utilisées qu'à des fins de vérifications. Cependant, si en raison de la façon dont nous procéderons, la dimension ne s'avérait pas déterminable par le corps de procédure, elle pourrait l'être par les appels - à condition qu'ils soient compatibles (au sens que nous définirons). Cela revient à dire que, en fait, la détermination par le corps de procédure ne donne qu'un "mode préférentiel".

2.6.6.2 Points d'acquisition ou de vérification

Les occurrences d'application du paramètre formel dans le corps de procédure sont de deux sortes.

1) variable indicée

La dimension du paramètre formel de type tableau s'obtient alors par simple comptage des indices.

2) paramètre effectif d'un appel de procédure

a) procédure non paramètre formel

Soit f(t) cet appel. Le traitement consiste en l'examen du paramètre de f correspondant à t : les deux paramètres formels de type tableau ont même dimension. C'est-à-dire que l'on est ramené à la recherche du mode d'un paramètre de f. On remarquera qu'une telle occurrence est aussi une occurrence d'appel de f, donc de vérification pour le paramètre formel correspondant.

b) paramètre formel de type procédure

Dans ce cas, la dimension du paramètre de type tableau est potentiellement variable à chaque appel de la procédure, car elle est fonction du paramètre effectif correspondant au paramètre formel de type procédure. Une telle occurrence ne sera pas considérée comme d'apport d'information, ni pour le paramètre formel de type tableau, ni pour celui de type procédure. Si cette occurrence était la seule occurrence d'application du paramètre formel de type tableau dans le corps de procédure, la dimension ne pourrait être déterminée qu'à partir des appels - sous réserve de leur compatibilité.

Quant aux occurrences d'appel de la procédure, le paramètre effectif correspondant au paramètre formel de type tableau peut être :

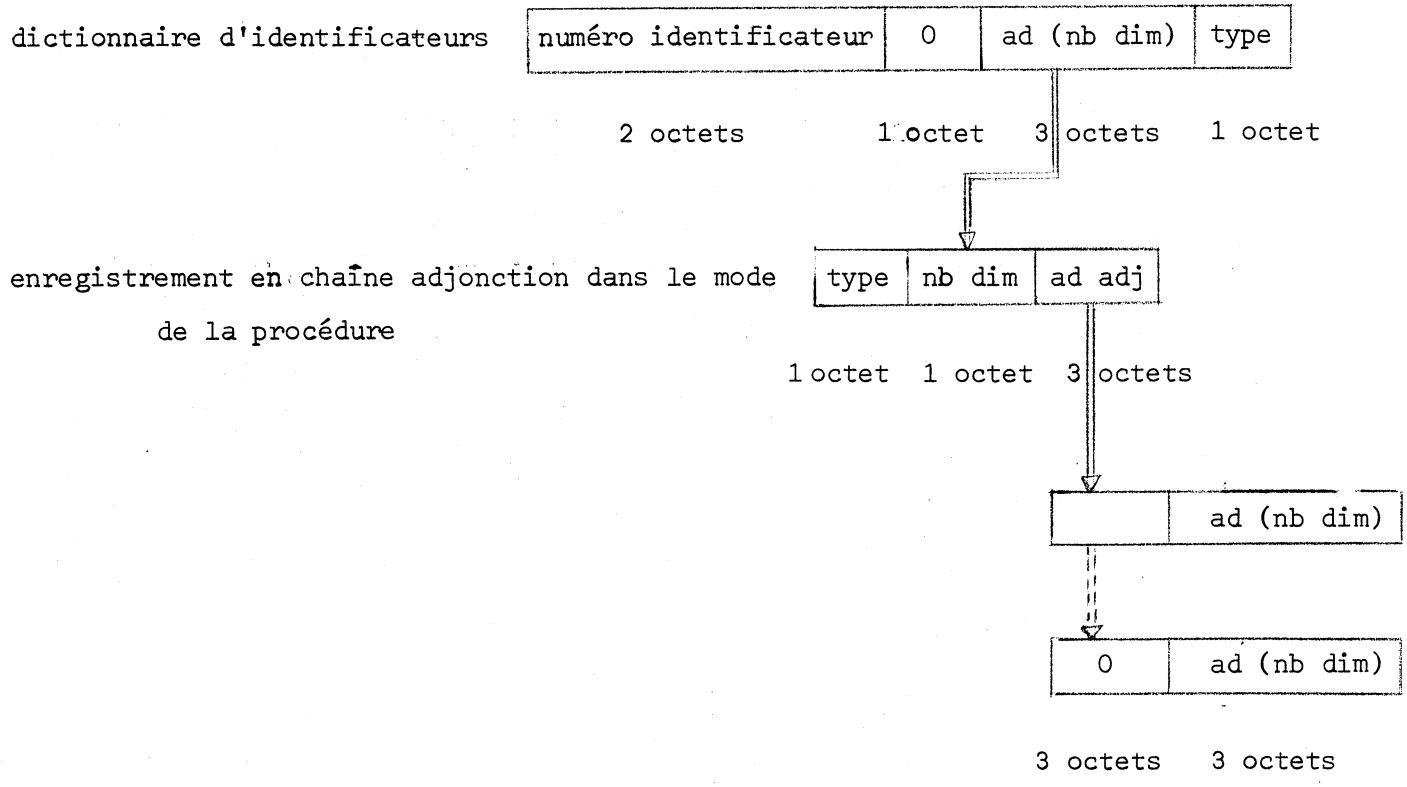
1) un tableau non paramètre formel, donc de dimension déterminable à partir de sa déclaration.

2) un paramètre formel de type tableau, donc de dimension acquise ou à acquérir selon les implantations programme relatives des déclarations des procédures correspondantes.

2.6.6.3 Enregistrements créés

L'information enregistrée, par suite d'une acquisition ou à des fins de vérification ultérieure, sera :

1) une dimension, en clair sur un octet
2) une liste d'adresses se référant au champ contenant la dimension d'un paramètre formel de type tableau.



Une seule position est réservée pour la dimension. Cette position est remplie par la première et testée par toute autre acquisition ou vérification, fournissant une dimension. Quant aux adresses chaînées - en chaîne adjonction -, ce sont celles de cette même position, pour les paramètres formels de type tableau, impliqués dans une acquisition ou une vérification, dont la dimension n'est pas encore connue.

2.6.6.4 Processus d'acquisition ou de vérification

1) Occurrences d'application du paramètre formel

a) Variable indicée

Le nombre d'indices est calculé, fournissant la dimension. Si la position "nb dim" de l'enregistrement est vide, cette information y est enregistrée. Sinon on en vérifie la compatibilité avec celle qui y avait été précedemment rangée.

b) Paramètre effectif d'un appel de procédure non paramètre formel

Soit f(t) cet appel. Le paramètre de f correspondant à t est examiné. Il a pu, ou non, être déjà traité, selon les positions relatives des déclarations des deux procédures.

Si la position "nb dim" de ce paramètre est vide, l'adresse en est enregistrée et chaînée aux précédentes de t. L'acquisition ou la vérification correspondante ne pourra être faite qu'au troisième passage, lors de la traduction de la tête de la déclaration de procédure. Sinon, l'information qu'elle contient est recopiée dans la position correspondante de t, ou comparée avec son contenu, selon celui-ci.

2) Occurrences d'appel de la procédure

a) Paramètre effectif tableau non paramètre formel

La dimension en est déterminable directement à partir de la déclaration. La vérification sur le paramètre formel correspondant est effectuée selon le cas (a) précédent.

b) Paramètre effectif paramètre formel de type tableau

Soit g(u), cet appel. La vérification du mode du paramètre formel t de g, correspondant à u, est effectuée selon le cas (b) ci-dessus.

2.6.6.5 Justification

L'algorithme précédent se résume :

1) en une première acquisition d'une dimension en clair sur un octet, et une vérification immédiate de compatibilité pour toute autre information ultérieure de ce type

2) en un enregistrement de l'adresse de la position "nb dim" lorsque l'acquisition ou la vérification de la dimension n'est pas encore possible. Elle est alors repoussée au troisième passage.

On doit vérifier que toutes les sources d'information fournissant pour dimension du paramètre formel de type tableau, toujours le même nombre. Le contraire n'est nullement incompatible avec le fait d'avoir supposé le programme en Algol 60 correct. C'est ainsi que la procédure suivante est syntaxiquement et sémantiquement correcte en Algol 60 :

```
procédure P(A, I); array A; integer I;
      if I = 1 then A[1] := 3.14 else A[1, 1] := 6.28;
```

avec toute la récurrence que l'on peut imaginer ...

Il importe seulement, en Algol 60, qu'il y ait compatibilité dynamique entre le paramètre formel et le paramètre effectif.

Il n'en est pas de même en Algol 68, car la dimension fait partie du mode du paramètre formel de type tableau. Les processus d'identification des paramètres formels étant en tout point semblables à ceux des autres quantités, un cas tel que le précédent est exclu par les "conditions de contexte".

Si les recherches précédentes ne fournissaient aucun résultat, ou des résultats différents, la traduction serait abandonnée après émission d'un message d'erreur.

Les acquisitions - vérifications que nous faisons, sont incomplètes, pour deux raisons :

1) Nous ne considérons pas une occurrence de paramètre formel de type tableau, en paramètre effectif d'un appel de paramètre formel de type procédure, malgré tout génératrice d'informations, ou source d'erreurs.

2) Au troisième passage, dans la recherche de l'information à travers les différentes adresses enregistrées, toutes les recherches sont limitées au premier niveau de chaque pointeur, c'est-à-dire à la position "nb dim". Cette restriction nous évite d'introduire des tests d'arrêt : il est en effet possible de boucler indéfiniment - ne serait-ce qu'en raison des diverses récursivités possibles.

2.6.6.6 Traduction d'un paramètre formel de type tableau

Elle est faite au troisième passage, selon la syntaxe suivante :

2.6.6.6.1 Appel par nom

Algol 60 :

<partie spécification> ::= array <liste d'identificateurs>; | <type> array <liste d'identificateurs>;

traduction :

<traduction paramètre formel> ::= ref [<liste de paires de bornes formelles>] <traduction type> <identificateur>
<liste de paires de bornes formelles> ::= <vide> | <vide>, <liste de paires de bornes formelles>

Comme dans les déclarations de tableau, une spécification de tableau à <type> omis rend real implicite.

On remarquera la différence dans le choix du mode de ce paramètre formel, par rapport à un paramètre formel de type réel, entier ou booléen. En effet, on a vu que le proceduring n'avait pour rôle que la retransmisssion dans la traduction, des effets de bord possibles sur l'évaluation du paramètre effectif. Une telle démarche était ici inutile, car le paramètre effectif ne peut être qu'un identificateur de tableau.

Exemple :

Algol 60 : procédure P(A); integer array A; ..... ;
Algol 68 : proc P = (ref [,] int A) : ..... ;

2.6.6.6.2 Appel par valeur

Algol 60 :

<partie valeur> ::= value <liste d'identificateurs>;
<partie spécification> ::= array <liste d'identificateurs>; | <type> array
<liste d'identificateurs>;

traduction :

<traduction paramètre formel> ::= ref [<liste de paires de bornes formelles>]
<traduction type> <identificateur 1>

et est insérée au début du corps de procédure, la déclaration suivante :

[<liste de paires de bornes à fixer>] <traduction type> <identificateur> :=
<identificateur 1>;
<liste de paires de bornes à fixer> ::= <paire de bornes à fixer> | <liste
de paires de bornes à fixer>, <paire de bornes à fixer>
<paire de bornes à fixer> ::= skip flex : skip flex

avec éventuellement un niveau de bloc supplémentaire.

Contrairement à ce qui se passait pour la traduction de la rémanence, il ne s'agit pas ici d'un tableau à bornes variables, mais d'un tableau dont les bornes seront fixées par cette déclaration avec initialisation. Une fois acquises, elles resteront inchangées pendant toute une activation de la procédure.

Exemple :

Algol 60 : procédure P(A); value A; integer array A; begin ..... end;
Algol 68 : proc P = (ref [,] int Aprime) : begin [skip flex : skip flex,
skip flex : skip flex] int A := Aprime ; ..... end;

2.6.7 Paramètre formel de type procédure

Le traitement est analogue à celui d'un paramètre formel de type tableau. La déclaration d'Algol 60 ne fournit que le mode du résultat de l'appel d'un paramètre formel de type procédure : vide ou non selon qu'il s'agit d'une procédure propre ou d'une fonction procédure. Il faut donc déterminer le nombre et les modes des paramètres que ce paramètre formel de type procédure est susceptible d'admettre.

Si le paramètre formel de type procédure figure en partie valeur, c'est qu'il ne possède pas de paramètre, et le problème est résolu. Sinon la recherche de leur nombre et de leurs modes est faite au deuxième passage.

Mais, que la partie valeur correspondante soit vide ou non, le paramètre formel est simplement, au premier passage, recopié dans la liste de paramètres formels. Si, au troisième passage, son mode Algol 68 est complètement déterminé, la déclaration Algol 68 de la procédure est complétée, selon une syntaxe que nous définirons.

2.6.7.1 Détermination du mode Algol 68

La politique adoptée est celle qui l'avait été pour les paramètres formels de type tableau.

Les occurrences d'application du paramètre formel dans le corps de procédure sont cette fois :

1) **occurrence d'appel du paramètre formel de type procédure**

La recherche s'orientera vers la détermination du nombre et des modes des paramètres effectifs.

Cependant, cette recherche présente des difficultés et des risques particuliers. En effet, elle signifie que le mode d'un paramètre formel est déterminé à partir de celui d'un paramètre effectif, alors que, dans toute correspondance paramètre formel - paramètre effectif, c'est le paramètre formel qui impose son mode. D'autre part, il n'existe pas, en Algol 68, de coercion sur les modes des paramètres, à l'intérieur du mode d'une procédure. Ceci signifie, qu'étant données les déclarations des deux procédures f et g suivantes :

```
proc f = (proc(real) h) : begin ..... end
proc g = (int i) : begin ..... end
```

un appel tel que f(g) serait incorrect.

Aucun transfert de type n'est donc possible, et nous devons imposer une correspondance stricte pour les paramètres d'un paramètre formel de type procédure.

Pour les mêmes raisons que celles énoncées à propos des paramètres formels de type tableau, une telle occurrence ne serait pas prise en compte si un paramètre effectif était un paramètre formel de type tableau ou procédure.

2) paramètre effectif d'un appel de procédure

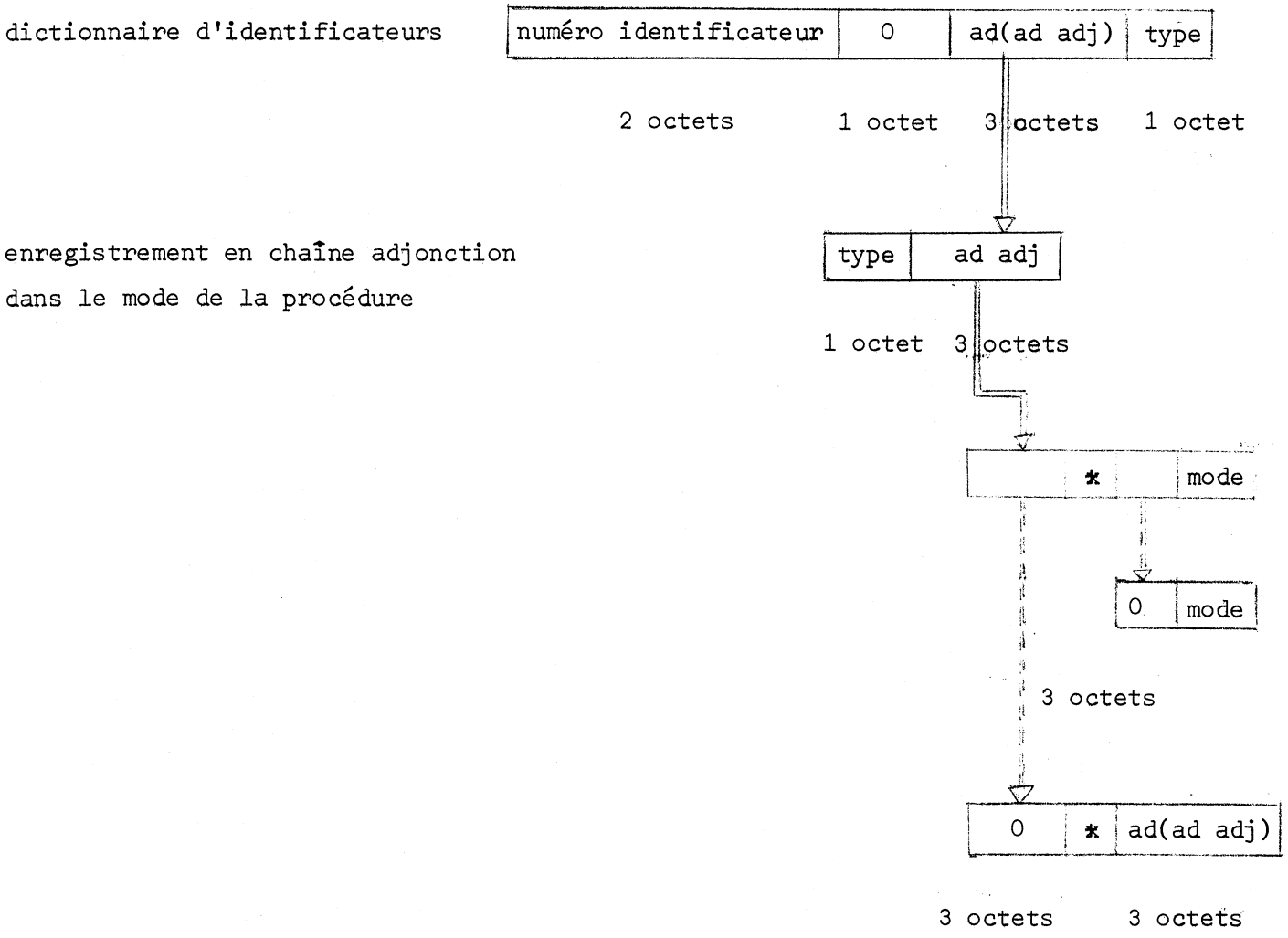
En raison de ce que nous venons de dire au paragraphe précédent, cette procédure n'est pas un paramètre formel. Soit g(f) cet appel. On se ramène alors à la recherche du mode d'un paramètre de g.

Quant aux occurrences d'appel de la procédure, le paramètre effectif correspondant peut, de la même façon, être une procédure non paramètre formel, ou un paramètre formel de type procédure.

2.6.7.2 Enregistrements créés

L'information enregistrée, par suite d'une acquisition, ou à des fins de vérifications, sera cette fois :

1) une liste de modes, fournie par un appel du paramètre formel de type procédure
2) une liste d'adresses se référant au mode Algol 68 d'un paramètre formel de type procédure, ou d'une procédure non paramètre formel



où * symbolise un marqueur d'identification de l'information enregistrée.

2.6.7.3 Processus d'acquisition ou de vérification

Contrairement à ce qui se passait pour un paramètre formel de type tableau, il ne peut pas y avoir, pratiquement, de vérifications immédiates. Il est en effet possible de trouver en paramètre effectif d'un appel de paramètre formel de type procédure, une procédure non paramètre formel dont le mode n'est pas encore complètement déterminé à ce stade du deuxième passage. De la même façon, un appel de la procédure admettant ce paramètre formel de type procédure, peut avoir en paramètre effectif, un paramètre formel de type procédure ou tableau, ou une procédure non paramètre formel. On se contentera alors, pour un tel paramètre effectif, d'enregistrer une adresse de localisation de l'information.

On se contentera donc, au deuxième passage, d'enregistrer toute l'information disponible; les vérifications ne seront faites qu'au troisième passage.

Nous verrons lors du traitement des appels des paramètres formels de type procédure, le processus de détermination des modes des paramètres effectifs, et quelle est l'information enregistrée dans la liste de modes résultant de cet appel.

En raison de l'absence de coercion sur les modes des paramètres d'un paramètre formel de type procédure, toutes les occurrences d'acquisition ou de vérification doivent fournir, pour tous les paramètres du paramètre formel de type procédure, exactement les mêmes modes. S'il n'en était pas ainsi, la traduction serait abandonnée, et un message d'erreur emis.

Comme pour les paramètres formels de type tableau, les acquisitions et vérifications sont incomplètes. Et, lors du troisième passage, pour les mêmes raisons, nous nous limitons au premier niveau de chaque pointeur.

2.6.7.4 Traduction d'un paramètre formel de type procédure

2.6.7.4.1 Appel par nom

Algol 60 :

<partie spécification> ::= <type procédure> procédure <liste d'identificateurs>;
<type procédure> ::= <type> | <vide>

traduction :

<traduction paramètre formel> ::= proc <partie paramètres virtuels> <traduction type procédure> <identificateur>
<traduction type procédure> ::= <traduction type> | <vide>
<partie paramètres virtuels> ::= <vide> | (<liste de paramètres virtuels>)
<liste de paramètres virtuels> ::= <paramètre virtuel> | <paramètre virtuel>, <liste de paramètres virtuels>
<paramètre virtuel> ::= <déclarateur virtuel de mode>

Exemple :

Algol 60 : procédure P(f); real procédure f; ..... ;
Algol 68 : proc P = (proc (proc ref real, proc ref [,] int) real f) : ..... ;

Si on a reconnu que f possédait :

-) un paramètre formel de type réel
-) un paramètre formel de type tableau entier de dimension 2

2.6.7.4.2 Appel par valeur

Il ne peut s'agir alors que d'une fonction procédure sans paramètre, car, étant évaluée dès l'entrée du corps de procédure, sans apport d'information, c'est qu'elle constitue déjà en elle-même une expression.

Algol 60 :

<partie valeur> ::= value <liste d'identificateurs>;
<partie spécification> ::= <type> procédure <liste d'identificateurs>;

traduction :

<traduction paramètre formel> ::= proc <traduction type> <identificateur 1>

et est insérée au début du corps de procédure, avec éventuellement un niveau de bloc supplémentaire, la déclaration suivante :

<traduction type> <identificateur> = <identificateur 1>;

Exemple :

Algol 60 : procédure P(f); value f; real procédure f; begin ..... end
Algol 68 : proc P = (proc real fprime) : begin real f = fprime; ..... end

On remarquera que f est de mode reàl et non ref real.
Comme il ne risquait pas de figurer en partie gauche d'affectation dans le corps de procédure, il était inutile de lui attribuer un nom.

CHAPITRE III

INSTRUCTIONS

CHAPITRE III

INSTRUCTIONS

3.1 INSTRUCTION COMPOSEE ET BLOC

Le terme Algol 60 de "bloc" a l'inconvénient de ne pas faire la distinction entre l'entité syntaxique et ce qu'elle représente au point de vue portée d'identificateur.

La portée d'un identificateur était en Algol 60 une portée de bloc. Il en est de même en Algol 68. Plus exactement, le concept Algol 60 de "bloc" a été étendu en Algol 68, en celui de "clause sérielle" qui est syntaxiquement parenthésée par le couplage de deux délimiteurs : non seulement begin - end, mais aussi if - then, then - else, et else - fi. De telle sorte que, si dans un contexte Algol 68, il nous arrive de parler de "bloc", c'est uniquement pour ce que ce terme représente au point de vue portée d'identificateur.

Une clause sérielle peut ne pas comporter de déclaration. Ces déclarations, lorsqu'elles existent, n'y précèdent pas obligatoirement toutes les instructions, il suffit qu'elles précèdent la première étiquette. Nous avons déjà, dans ce qui précède, utilisé implicitement cette facilité pour la traduction.

Il en résulte donc que la distinction bloc et instruction composée d'Algol 60 a disparu. Ces deux concepts sont unifiés en celui de "clause fermée" (closed clause) qui est une clause sérielle entre les deux délimiteurs begin et end.

Les parties then et else d'une instruction conditionnelle étant devenues, en Algol 68, des blocs, nous avons pensé qu'il était intéressant d'en faire disparaître dans la traduction un éventuel parenthésage begin - end, maintenant redondant.

La traduction d'un bloc ou d'une instruction composée est celle de ses différents constituants.

3.2 COMMENTAIRES

Bien que figurant dans ce chapitre, les commentaires ne peuvent être considérés comme étant des ordres d'un langage, puisqu'ils sont éliminés du programme dès la première phase de la compilation. Ils ne sont qu'une information purement externe, destinée à en faciliter la lecture.

Contrairement à ce qui se passait en Algol 60, un commentaire peut, en Algol 68, être inséré entre deux symboles quelconques du programme. Il y est parenthésé par le couplage de deux symboles comment et peut comporter n'importe quel caractère disponible dans l'implémentation. Par rapport aux conventions d'Algol 60, ce couplage facilite sa reconnaissance dans un programme source.

Puisque notre but est d'obtenir un programme source en Algol 68, un commentaire figurant dans le programme en Algol 60 est traduit et intégralement retransmis.

Nous avons déjà vu la traduction d'un commentaire de procédure.

1) Commentaire introduit par comment

Algol 60 :

comment <toute suite ne comportant pas ;> ;

traduction :

comment <toute suite ne comportant pas ;> comment

C'est-à-dire que le point-virgule, uniquement délimiteur de fin de commentaire en Algol 60, est remplacé par un symbole comment.

2) Commentaire après end

Algol 60 :

end <toute suite ne comportant pas end ou ; ou else>

traduction :

end comment <toute suite ne comportant pas end ou ; ou else> comment

Cette traduction, ne nécessitant qu'une délimitation de l'information, est effectuée dès le premier passage.

Aucun test n'est fait sur le commentaire lui-même. On remarquera qu'il ne peut comporter une suite de caractères qui serait une représentation du symbole comment.

3.3 INSTRUCTION VIDE

Une telle instruction est sans effet. On l'utilisait souvent, en Algol 60, pour se renvoyer à la fin d'un bloc ou d'une instruction composée :

```
begin
:       ;
:
:  goto ℓ;
:
:
ℓ: end
```

Cette possibilité est directement offerte, en Algol 68, par le "compléter" :

```
begin
:       .
:
:  end
```

Ce concept n'a pas été introduit dans le sous-ensemble.

La traduction de l'instruction vide est effectuée dès le premier passage, par insertion d'un symbole skip entre les délimiteurs qui la définissent. Ce symbole n'a, en Algol 68, qu'un rôle de remplissage syntaxique. On peut l'utiliser pour remplacer toute quantité en contexte strong dont la valeur possédée est sans intérêt. Il représente, non seulement le concept Algol 60 d'instruction vide, mais aussi celui d'expression vide.

Par exemple, la traduction de l'exemple précédent sera :

```
begin
:       ;
:
:  goto ℓ;
:
:  ℓ : skip
:
end
```

3.4 INSTRUCTION CONDITIONNELLE

Une instruction conditionnelle est, en Algol 68, complètement parenthésée par un couplage if - fi. Ce couplage permet d'éviter les restrictions d'Algol 60 sur la partie then; restrictions qui étaient dues à des risques d'ambiguïté qui ont maintenant disparu. Les parties then et else étant devenues des blocs, nous avons déjà dit qu'il n'était plus possible, en Algol 68, de s'y renvoyer directement par une instruction goto. Cette possibilité qu'offrait Algol 60 est néanmoins retransmise dans la traduction par la mise en place d'un chaînage que nous verrons lors de la traduction des expressions de désignation, et qui simule ce branchement direct maintenant impossible.

En Algol 68, une instruction conditionnelle possède toujours, dans le langage strict, une partie else. Cependant, dans le langage étendu, la construction "else skip fi" peut être remplacée par "fi", fournissant ainsi l'équivalent de l'instruction si d'Algol 60.

La traduction d'une instruction conditionnelle est effectuée de la façon suivante :

1) insertion, dès le premier passage, d'un symbole fi, délimiteur de fin
2) traduction de l'expression booléenne et des différents constituants des parties then et else.

3.5 INSTRUCTION D'AFFECTATION

Une instruction d'affectation peut aussi, en Algol 68, être multiple. Sa structure sera donc respectée dans la traduction, et sa traduction sera celle de ses différents constituants.

C'est la syntaxe qui implique en Algol 68 que les affectations constituantes soient effectuées de droite à gauche. On rappelle qu'en Algol 68, les modes et les processus de coercion sont exprimés directement dans la syntaxe. Il suffit donc que les affectations constituantes soient toutes syntaxiquement possibles, contrairement à ce qui se passait en Algol 60 où les parties gauches devaient toutes être du même type.

Une partie gauche peut être en Algol 60, une variable, simple ou indicée, paramètre formel ou non, ou un identificateur de procédure. Ce dernier cas a déjà été traité à propos des fonctions procédure. Le traitement de la variable sera étudié au chapitre suivant, traitant des expressions.

Le traitement de l'expression partie droite consiste tout d'abord en sa propre traduction, c'est-à-dire indépendamment de ce contexte d'affectation où elle figure. En effet, un problème particulier surgit du fait, que nous avons déjà signalé et justifié, qu'il n'existe pas en Algol 68 de coercion automatique réel vers entier. De telle sorte qu'une affectation telle que

i := x avec i entier et x réel

est incorrecte. C'est au programmeur de spécifier la conversion à effectuer. Pour retransmettre dans la traduction l'arrondi automatique d'Algol 60, nous utiliserons l'opérateur d'Algol 68 round, dont l'effet peut être défini par rapport à la fonction standard ENTIER d'Algol 60, par la relation :

round x = ENTIER(x + 0.5)

Si les parties gauches sont de mode entier, le traitement de l'expression partie droite comportera donc aussi la recherche de son mode, de façon à éventuellement provoquer, dans la traduction, ce transfert explicite. Nous reviendrons sur ce problème lors du traitement des expressions arithmétiques.

3.6 APPEL DE PROCEDURE

Par appel de procédure, nous entendons aussi bien "instruction procédure" qu' "indicateur de fonction", tels que les distingue Algol 60 selon le contexte d'apparition : instruction ou expression. Les processus d'élaboration sont identiques. Ces deux appels ne diffèrent que sur la prise en considération ou non d'une valeur éventuellement délivrée.

Une instruction procédure peut correspondre aussi bien à l'appel d'une fonction procédure que d'une procédure propre. Si une valeur est délivrée, elle est ignorée. Au contraire, un indicateur de fonction ne peut correspondre qu'à l'appel d'une fonction procédure. Une valeur est toujours délivrée, qui est ensuite reprise pour continuer l'évaluation de l'expression où l'appel apparaît.

Nous avons vu que la distinction fonction procédure - procédure propre disparaît en Algol 68. Un appel de procédure y délivre, ou non, une valeur, prise en considération ou ignorée, selon le contexte.

Dans le contexte de la traduction, une valeur ne peut être délivrée que si l'appel d'Algol 60 délivre lui-même une valeur. Et nous avons vu, lors de la traduction des fonctions procédure, que la traduction délivrera la même valeur.

Le problème des valeurs délivrées étant ainsi résolu, nous ne ferons plus dans la suite de ce paragraphe, la distinction instruction procédure - indicateur de fonction.

Un appel de procédure a pour rôle de fournir les paramètres effectifs correspondant aux paramètres formels de la tête de procédure de la déclaration de procédure. Plus exactement, en termes d'Algol 68, il a pour rôle de fournir la partie droite de la déclaration d'identité dont le paramètre formel est la partie gauche. La traduction d'un paramètre effectif va donc être influencée par celle qui a été choisie pour le paramètre formel correspondant. Nous verrons pour chaque type de paramètre les contraintes et les limitations de traduction qui résultent du choix qui a été fait du mode Algol 68 du paramètre formel.

Le traitement des paramètres effectifs n'est effectué qu'à partir du deuxième passage, en raison des identifications nécessaires. Ces identifications ne posent pas de problème particulier, car, si une procédure est élaborée dans le contexte de la déclaration, un paramètre effectif l'est toujours dans celui de l'appel.

La traduction diffère sensiblement selon qu'il s'agit de l'appel d'un paramètre formel de type procédure ou d'une procédure non paramètre formel. C'est une conséquence de ce que nous avons dit dans le chapitre des déclarations de procédure, à propos des paramètres formels de type procédure.

En effet, la traduction de l'appel d'un paramètre formel de type procédure est mise à profit pour rechercher le nombre et les modes de ses paramètres, à partir de ceux des paramètres effectifs. Nous avons signalé l'obligation où nous nous trouvons, d'imposer que le mode de la quantité figurant dans une position donnée de paramètre effectif, ne varie pas d'un appel à l'autre du paramètre formel de type procédure.

D'autre part, le mode d'appel des paramètres est inconnu. Le choix qui a été fait correspond à un appel par nom, systématique, au niveau de la traduction du paramètre effectif.

Au contraire, la traduction de l'appel d'une procédure non paramètre formel ne répond pas à la même motivation. Et nous mettrons à profit la connaissance du mode d'appel des paramètres pour permettre le maximum de tolérance dans la correspondance paramètre formel - paramètre effectif. Nous verrons jusqu'à quel point nous pouvons restituer dans la traduction les tolérances qui étaient offertes par Algol 60.

3.6.1 Appel d'une procédure non paramètre formel

3.6.1.1 Paramètre effectif correspondant à un paramètre formel de type réel, entier ou booléen

Ce paramètre effectif est, en Algol 60, une expression arithmétique ou booléenne, et est donc d'abord traduit comme tel. S'il correspond à un paramètre appelé par nom, il doit aussi être analysé - la traduction dépendant de son niveau : valeur ou variable. Son mode doit, lui aussi, être déterminé pour prendre éventuellement en charge un transfert explicite de mode, réel vers entier.

3.6.1.1.1 Traduction d'un paramètre effectif correspondant à un paramètre appelé par nom

1) le paramètre effectif est une variable

<traduction paramètre effectif> ::= <traduction expression>

La déclaration d'identité dont le paramètre formel est la partie gauche, est la suivante :

proc ref <traduction type> <identificateur> = <traduction expression>

par exemple :

proc ref real x = t [j]

où t serait déclaré dans le programme en Algol 60 : real array t [1 : 5]

Nous avons vu que nous retransmettions bien ainsi l'appel par nom d'Algol 60 : le paramètre effectif t [j] est réévalué à chaque occurrence d'application du paramètre formel x dans le corps de procédure, fournissant une valeur de mode ref real, le nom qu'il possède.

Nous avons dit que le paramètre effectif était, en Algol 68, en position syntaxique strong - la position syntaxique déterminant les coercions possibles. Il en résulte que le paramètre effectif doit - impérativement - être du mode a priori ref <traduction type> (par exemple ref real; c'est-à-dire posséder un nom susceptible de référer à des valeurs de mode real). En termes d'Algol 60, le paramètre effectif doit être du type du paramètre formel. Aucune tolérance n'est possible, sinon cela conduirait à des déclarations d'identité telles que :

proc ref real x = i avec i de mode a priori ref int

Une telle déclaration d'identité est incorrecte. En effet, le processus de coercion qui serait tenté pour passer du mode a priori ref int du paramètre effectif, au mode a posteriori proc ref real requis par le contexte serait le suivant :

ref int → int → real ✗ ref real → proc ref real

et passer de real à ref real est impossible, car aucune coercion ne peut ajouter un niveau de référence.

Nous sommes donc, pour un paramètre arithmétique, plus restrictifs qu'Algol 60, où (ce que nous ne testons pas) dans la mesure où le paramètre formel n'apparaît pas en partie gauche d'affectation dans le corps de procédure - c'est-à-dire que l'on n'utilisera jamais que la valeur référée par le paramètre effectif -, cette valeur référée par le paramètre effectif est convertie automatiquement, à chaque occurrence, au type du paramètre formel.

2) le paramètre effectif n'est pas une variable

<traduction paramètre effectif> ::= loc <traduction type> := <traduction expression>

La déclaration d'identité dont le paramètre formel est la partie gauche, est cette fois :

proc ref <traduction type> <identificateur> = loc <traduction type> := <traduction expression>

par exemple :

proc ref real x = loc real := y + 3

De la même façon qu'au (1) précédent, l'élaboration de cette déclaration d'identité fait posséder au paramètre formel x, de mode proc ref real, la routine qui est la séquence de caractères : "loc real := y + 3". Et chaque occurrence d'application de x dans le corps de procédure provoque l'élaboration de cette routine, fournissant une valeur de mode ref real : un nom local, créé par l'élaboration du générateur, qui réfère à la valeur réelle que possède le paramètre effectif.

En effet, le paramètre effectif n'est pas de mode référence. Cependant, dans la traduction, en raison du choix du mode du paramètre formel, et en fonction des coercions disponibles, il doit, à priori, être d'un tel mode. C'est la raison pour laquelle la traduction est obligée de prendre en charge la création d'un nom local qui est fait se référer à la valeur possédée par le paramètre effectif. Un nouveau nom est ainsi créé à chaque élaboration de la routine possédée par le paramètre formel. Le programme en Algol 60 a été supposé correct, mais il résulte aussi de ce choix que si le paramètre formel correspondant à un tel paramètre effectif apparaissait en partie gauche d'affectation dans le corps de procédure, aucune erreur ne serait détectée à l'exécution du programme en Algol 68 résultant.

Puisque, dans la traduction d'un tel paramètre effectif, on ne manipule qu'une valeur de niveau zéro, nous pouvons restituer les tolérances de correspondance de type arithmétique qu'offrait Algol 60. Ces tolérances se répercutent sur l'affectation au nom local.

3.6.1.1.2 Traduction d'un paramètre effectif correspondant à un paramètre appelé par valeur

<traduction paramètre effectif> ::= loc <traduction type> := <traduction expression>

La déclaration d'identité dont le paramètre formel est la partie gauche, est cette fois encore :

proc ref <traduction type> <identificateur> = loc <traduction type> := <traduction expression>

par exemple :

proc ref real x = loc real := i

avec i de mode a priori ref int

Nous avons vu, au chapitre des déclarations de procédure, comment était retransmis en Algol 68 l'appel par valeur d'Algol 60. La routine qui est la même séquence de symboles que la traduction du paramètre effectif est élaborée dès l'entrée du corps de procédure, fournissant une valeur de mode ref real (un nom local créé par l'élaboration du générateur). La valeur référée par ce nom local - c'est-à-dire la valeur niveau zéro référée ou possédée par le paramètre effectif d'Algol 60 et convertie au type du paramètre formel - est alors affectée à une variable locale au corps de procédure, de même représentation que le paramètre formel. Il est donc clair que nous pouvons là aussi permettre et restituer les tolérances de type arithmétique d'Algol 60.

3.6.1.2 Paramètre effectif correspondant à un paramètre formel de type étiquette

Le paramètre effectif est une expression de désignation, et est d'abord traduit comme tel.

Nous avons dit que nous considérerons toujours qu'un tel paramètre est appelé par nom, quelque soit donc, en fait, son mode d'appel dans le programme

<traduction paramètre effectif> ::= : <traduction expression de désignation>

En effet, une difficulté surgit pour une expression de désignation conditionnelle. Une telle expression n'est pas ce que l'on appelle en Algol 68, un "coercend" - c'est-à-dire qu'elle n'est pas susceptible de coercion. L'expression booléenne serait évaluée, et, selon la valeur délivrée, le proceduring effectué uniquement sur la partie then ou la partie else.

Par exemple, supposons que la déclaration d'identité dont le paramètre formel est la partie gauche, soit la suivante :

proc e = if a > 0 then goto s else goto t fi

et non proc e = : if a > 0 then goto s else goto t fi

Alors, selon la valeur de a, le paramètre formel e posséderait la routine constituée de la séquence de symboles "goto s" ou bien "goto t".

Nous aurions donc, en fait, un appel par valeur, mais partiel seulement en raison des possibilités de variation, d'une élaboration à l'autre, de l'indice d'un indicateur d'aiguillage figurant dans la partie procédurée.

Pour retransmettre l'appel par nom d'Algol 60; c'est-à-dire pour que l'expression soit procédurée en bloc, il suffit de la transformer en "void cast pack". Compte tenu des extensions, cela revient à la faire précéder d'un symbole ":".

Nous verrons que ce problème se pose aussi lors du traitement d'un branchement intérieur à une instruction composée ou conditionnelle. Aussi, pour des raisons de facilité de traduction, l'insertion du symbole : est systématique, quelque soit le paramètre effectif. Cette insertion, systématique, est justifiée par le fait qu'elle ne pénalise pas l'exécution quand elle est inutile.

3.6.1.3 Paramètre effectif correspondant à un paramètre formel de type aiguillage

Ce paramètre effectif ne peut être qu'un identificateur d'aiguillage. Sa traduction consiste en sa simple retransmission.

3.6.1.4 Paramètre effectif correspondant à un paramètre formel de type chaîne

Ce paramètre effectif ne peut être qu'un paramètre formel de type chaîne, ou une chaîne. Il est simplement retransmis dans la traduction.

3.6.1.5 Paramètre effectif correspondant à un paramètre formel de type tablea

Le paramètre effectif est un identificateur de tableau - paramètre formel de type tableau ou tableau non paramètre formel.

La traduction consiste en la retransmission de cet identificateur, et en la vérification que le paramètre effectif a même dimension et des éléments du même type que le paramètre formel correspondant. Cette correspondance stricte est impérative, en raison de l'absence de toute coercion possible.

Nous avons déjà vu, à propos de la recherche de la dimension d'un paramètre formel de type tableau, le rôle d'apport d'information ou de vérification d'un tel appel, en fonction du paramètre effectif :

1) tableau non paramètre formel

La dimension en est rangée, ou vérifiée avec celle précédemment rangée, dans l'enregistrement en chaîne adjonction du paramètre formel dans le mode de la procédure.

2) paramètre formel de type tableau

Le principe est celui d'une acquisition ou vérification mutuelle de la dimension des deux paramètres formels; éventuellement d'un rangement des adresses de leurs enregistrements en chaîne adjonction dans les modes des procédures.

Par exemple, soit g(u), cet appel

- avec t le paramètre formel de type tableau de g correspondant au paramètre effectif u
- et u, le paramètre effectif, qui est un paramètre formel de type tableau d'une procédure f.

Si la dimension du paramètre formel de type tableau u est connue, l'acquisition ou la vérification pour t est effectuée selon le cas (1) ci-dessus. Sinon, l'adresse de cette position dans l'enregistrement en chaîne adjonction de u dans le mode de f est rangée dans l'enregistrement en chaîne adjonction de t dans le mode de g. Même chose pour u, selon t.

3.6.1.6 Paramètre effectif correspondant à un paramètre formel de type procédure

Le paramètre effectif est un identificateur de procédure - paramètre formel de type procédure ou procédure non paramètre formel.

Il est d'abord retransmis dans la traduction. Nous ferons, pour cette traduction, la distinction, selon que le paramètre effectif correspond à un paramètre appelé par nom ou par valeur.

3.6.1.6.1 Traduction d'un paramètre effectif correspondant à un paramètre appelé par nom

Le traitement est analogue à celui d'un paramètre effectif correspondant à un paramètre formel de type tableau. Nous avons vu le rôle du paramètre effectif dans la détermination du mode du paramètre formel de type procédure correspondant, et la nécessité d'imposer que leurs modes Algol 68 soient les mêmes.

1) paramètre effectif - procédure non paramètre formel

L'adresse de l'enregistrement en chaîne adjonction contenant le mode Algol 68 de la procédure paramètre effectif est rangée dans l'enregistrement en chaîne adjonction du paramètre formel correspondant.

2) paramètre effectif - paramètre formel de type procédure

Le principe en est, de la même façon, un rangement des adresses des enregistrements en chaîne adjonction des paramètres formels, dans les modes des procédures.

Par exemple, soit g(q), cet appel.

- avec p le paramètre formel de type procédure de g correspondant au paramètre effectif q
- et q, le paramètre effectif, qui est un paramètre formel de type procédure d'une procédure f.

Alors, l'adresse de l'enregistrement en chaîne adjonction de q dans le mode de f est rangée dans l'enregistrement en chaîne adjonction de p dans le mode de g. De même pour p.

Si le paramètre effectif est un paramètre formel de type procédure appelé par valeur, on peut immédiatement en déduire que le paramètre formel correspondant n'a pas de paramètre.

3.6.1.6.2 Traduction d'un paramètre effectif correspondant à un paramètre appelé par valeur

Nous avons vu que le mode Algol 68 du paramètre formel est d'emblée déterminé, puisque le paramètre effectif ne peut être qu'une fonction procédure sans paramètre. Cette information est à enregistrer pour un paramètre effectif qui est un paramètre formel de type procédure appelé par nom.

Il est possible, pour un tel paramètre, de permettre des transferts de type arithmétique. En effet, nous avons vu, au chapitre des déclarations de procédure, comment est retransmis en Algol 68, cet appel par valeur :

Par exemple, supposons que la déclaration d'identité dont le paramètre formel est la partie gauche, soit la suivante :

proc real fprime = p avec p , de mode a priori proc int

Le processus de coercion effectué sur p, pour passer de son mode a priori proc int au mode a posteriori proc real, est le suivant :

proc int → int → real → proc real

c'est-à-dire que le paramètre effectif est élaboré, la valeur entière délivrée, convertie au mode real et procédurée.

Et, à l'entrée du corps de procédure, cette valeur réelle est faite possédée par une quantité de mode real, de même représentation que le paramètre formel.

3.6.2 Appel d'un paramètre formel de type procédure

Par rapport à l'appel d'une procédure non paramètre formel, le principe du traitement réside ici en la recherche du mode d'un paramètre formel à partir de celui d'un paramètre effectif, et non en la vérification d'une correspondance.

Le résultat de cette recherche est une liste d'informations de mode dont l'adresse est insérée dans l'enregistrement en chaîne adjonction du paramètre formel de type procédure. Nous rappelons qu'un appel comportant en paramètre effectif un paramètre formel de type tableau ou procédure, est ignoré au point de vue acquisition de mode.

Un enregistrement de cette liste est un mode, suivi de la dimension pour un paramètre effectif tableau, de l'adresse de l'enregistrement en chaîne adjonction contenant le mode Algol 68 d'un paramètre effectif procédure, non paramètres formels. Cette information s'obtient par analyse d'un paramètre effectif, c'est-à-dire le plus souvent d'une expression, sur la base des délimiteurs et des identificateurs qui y figurent.

Signalons aussi l'ambiguïté qui résulte d'un paramètre effectif réduit à l'identificateur d'une fonction procédure (non paramètre formel) sans paramètre : on ne peut savoir a priori si le paramètre formel correspondant est de type : <type> ou <type> procédure. En l'absence de toute autre information, <type> procédure serait choisi.

La traduction d'un paramètre effectif correspond, par rapport à ce qui a été dit pour l'appel d'une procédure non paramètre formel, à toujours considérer un appel par nom. On a vu que le seul intérêt de cette distinction - ici irréalisable - était de permettre des tolérances de type dans la correspondance paramètre formel - paramètre effectif; ce qui est ici sans objet puisque le mode du paramètre formel est choisi être celui du paramètre effectif de l'appel.

3.6.3 Procédures de bibliothèque

De telles procédures ne sont pas déclarées : elles sont directement disponibles au programmeur sans déclaration explicite. Elles doivent donc être considérées comme déclarées en tête d'un bloc fictif contenant le programme.

Leur liste n'est pas limitative, et est fonction de l'implémentation. Nous allons voir le traitement de celles que nous avons choisies.

Nous supposerons qu'une procédure de bibliothèque ne peut pas figurer en paramètre effectif d'un appel de procédure, correspondant à un paramètre formel de type procédure. Ceci pour les deux raisons suivantes :

1) le correspondant, en Algol 68, d'une fonction standard n'est pas toujours une procédure, mais quelquefois un opérateur.

2) lorsque le correspondant d'une procédure de bibliothèque est une procédure, du standard-prélude d'Algol 68, les modes de ses paramètres ne sont pas ceux que nous avons choisis pour la traduction. Une solution, ici, existe : elle consisterait à redéclarer une telle procédure en tête du programme en Algol 68.

3.6.3.1 Fonctions standard

Ce sont des fonctions procédure, à un paramètre arithmétique, d'analyse ou de transfert de type. Celles que nous prenons en charge figurent dans les paragraphes suivants.

1) SQRT, SIN, COS, ARCTAN, LN, EXP

L'argument peut être indifféremment de type real ou integer, le résultat est toujours de type real. Il leur correspond des procédures d'Algol 68, de même représentation, pré-déclarées - c'est-à-dire dont les déclarations figurent dans le standard prélude. Ce sont des procédures à un paramètre de mode real. Il n'y a donc pas de problème de correspondance de type paramètre formel - paramètre effectif. Il suffit juste de traduire le paramètre effectif. La sémantique reste évidemment la même, et le résultat délivré est aussi de mode real.

2) ENTIER, SIGN

La valeur délivrée est de type integer, l'argument étant de type real pour ENTIER, real ou integer pour SIGN. Leur correspondant est un opérateur du standard-prélude d'Algol 68. La différence est importante, pour deux raisons :

- cette construction devient une "formule" en Algol 68, alors qu'un appel de procédure est une "base".

- les tolérances de correspondance de type ne sont pas les mêmes : un paramètre effectif est en contexte "strong" alors qu'un opérande est en contexte "firm". Par exemple, le "widening" (c'est-à-dire la coercion automatique d'entier vers réel) est impossible en contexte firm, pour des raisons d'identification des opérateurs.

Dans la traduction, l'identificateur - représentation de l'opérateur - reste inchangé, mais devenant un mot-clé, il est sorti par l'éditeur entre apostrophes. Il suffit cette fois encore de simplement traduire le paramètre effectif - opérande. A noter qu'il existe en fait deux opérateurs sign en Algol 68 : real → int et int → int.

3) ABS

L'argument est indifféremment de type real ou integer. Le résultat est de type real. Le correspondant, en Algol 68, est un opérateur - donc à sortir entre apostrophes - du standard-prélude. Seulement, il existe en fait deux opérateurs abs en Algol 68 : int → int et real → real. C'est-à-dire qu'en Algol 68, le résultat est du mode de l'opérande. Cet opérande doit donc, en plus de sa traduction propre, être analysé pour déterminer son mode. S'il est de mode int, il sera amené au mode real par l'utilisation d'un "cast", de façon que le mode du résultat soit aussi, en Algol 68, real

Exemple :

```
abs(real : i + 3)
```

L'intérêt d'un cast est, dans une position syntaxique qui n'est pas strong, de fournir une position syntaxique strong à la clause unitaire constituante. Il permet ici le widening.

3.6.3.2 Procédures d'entrée-sortie

La définition d'Algol 60 ne comprend pas de procédures standard d'entrée-sortie. Celles que nous étudierons ne sont que des procédures de bibliothèque propres à une certaine implémentation [IBM, PETI].

Ce sont des procédures à format implicite, et, pour leur traduction, nous nous limitons à l'utilisation des procédures standard d'entrée-sortie à format implicite d'Algol 68 : READ et PRINT. Nous nous contenterons simplement d'étudier les contraintes dues à ce choix sur l'information en entrée, le problème du format du résultat et de sa signification n'étant pas de notre ressort.

Les procédures d'Algol 60 permettent de préciser en paramètre un numéro de data-set. Nous considérerons qu'il s'agit toujours des data-sets 0 et 1, correspondant respectivement aux fichiers standard de lecture et d'écriture "stand in" et "stand out" d'Algol 68 - que spécifient implicitement READ et PRINT. Nous verrions, éventuellement, en fonction de ce que sera l'implémentation des entrée-sortie d'Algol 68, à modifier ultérieurement cette restriction; c'est-à-dire à utiliser les procédures d'Algol 68 GET et PUT, en spécifiant alors un nom de fichier. Il découle immédiatement de ce qui précède, que ne sont pas envisagées les traductions des procédures d'Algol 60 : GET, PUT, SYSACT.

Les procédures d'Algol 60 INSYMBOL et OUTSYMBOL, que nous allons étudier maintenant, ne sont pas non plus prises en charge par le traducteur. En effet, leurs traductions nécessitent des concepts que ne possède pas le sous-ensemble d'Algol 68; surtout, elles justifieraient l'emploi des procédures d'entrée-sortie à format explicite d'Algol 68.

INSYMBOL :

Cette procédure réalise les opérations suivantes : lecture d'un caractère en entrée, test de la présence de ce caractère dans une chaîne de référence qui est un paramètre de l'appel, affectation à une variable entière - qui figure aussi en paramètre - d'une valeur qui est le rang du premier caractère de la chaîne identique au caractère lu, ou zéro s'il n'y figure pas.

Ce processus se traduit très simplement à l'aide d'un "pattern entier de choix" (integral choice pattern) dont les éléments de la liste seraient les caractères de la chaîne. Par rapport à ce qui "est laissé indéfini dans le rapport", il suffirait de prévoir à l'implémentation, l'affectation de la valeur zéro à la variable lorsque le caractère lu ne figure pas dans la chaîne de référence.

OUTSYMBOL :

Le principe en est le même, mais en sortie, cette fois : impression d'un blanc ou d'un caractère d'une chaîne de référence, selon la valeur d'une variable entière : zéro en le rang du caractère dans la chaîne.

Un pattern entier de choix résoudrait de la même façon le problème, en prévoyant, à l'implémentation, l'impression d'un blanc pour la valeur zéro de la variable.

Cependant, l'utilisation des formats n'est pas absolument indispensable. Nous allons présenter sur un exemple, une traduction qui ne les utilise pas. Mais elle est beaucoup moins élégante, et nécessite aussi des concepts que ne possède pas le sous-ensemble d'Algol 68.

INSYMBOL

Algol 60 :
```
insymbol(0, '(' abc')',V );
```

Algol 68 :
```
begin
    char c;
    read(c);
    if ¬ char in string(c, V, "abc") then V := 0 fi.
end;
```

où la procédure char in string est ainsi déclarée dans le standard-prélude d'Algol 68 :

```
proc char in string = (char c, ref int i, string s) bool :
        begin for k to upb s do if c = s[k] then i := k; ℓ fi;
              false . ℓ : true
        end;
```

Elle permet donc d'abord, par la valeur booléenne qu'elle délivre, de tester la présence d'un caractère dans une chaîne de référence, et ensuite, éventuellement, de stocker le rang du premier caractère de la chaîne de référence identique à ce caractère.

OUTSYMBOL

Algol 60 : `outsymbol(1, '(' abc')', V);`

Algol 68 :
```
begin
    char c;
    c := if V = 0 then "  " else "abc" [V] fi;
    print(c)
end;
```

Les autres procédures d'Algol 60 sont prises en charge, et leurs traductions consistent en :

1) remplacer l'identificateur de procédure par READ ou PRINT, selon qu'il commence par IN ou OUT.

2) éliminer le numéro de data-set - premier paramètre effectif de l'appel d'Algol 60. Le nom du fichier est implicite dans READ et PRINT.

3) traduire (au deuxième passage) le deuxième paramètre effectif de l'appel d'Algol 60, qui devient le paramètre de READ ou de PRINT.

Le processus de substitution des identificateurs de procédure nous amène à remarquer qu'il n'existe pas en Algol 68 de procédure spécifique d'entrée ou de sortie d'une valeur réelle ou d'une valeur entière. Avec READ et PRINT, le transfert est conforme au mode du paramètre effectif. La seule tolérance est la possibilité de lire un entier pour un réel.

Ces possibilités qu'offrait Algol 60 - de lire ou imprimer un réel pour un entier, ou un entier pour un réel - n'ont qu'un intérêt restreint. Aussi, nous supposons que la correspondance stricte est satisfaite. Il est d'ailleurs impossible de la vérifier pour une procédure d'entrée.

3.7 INSTRUCTION GOTO

L'utilisation d'une instruction goto a pour effet de rompre le déroulement séquentiel d'un programme en définissant explicitement le successeur d'une instruction donnée.
La traduction est effectuée selon la syntaxe suivante :

Algol 60 :

<instruction goto> ::= goto <expression de désignation>

traduction :

<traduction instruction goto> ::= <traduction expression de désignation>

Exemple :

Algol 60 : goto if a > 0 then s else t
Algol 68 : if a > 0 then goto s else goto t fi

La syntaxe de l'instruction goto d'Algol 68 est la suivante :

<instruction goto> ::= goto <identificateur d'étiquette> | <identificateur d'étiquette>

Nous avons choisi, dans la traduction, de conserver le symbole goto. C'est pourquoi nous le réintroduisons devant les identificateurs d'étiquette non paramètres formels. Nous reviendrons sur ce problème lors de la traduction des expressions de désignation.

Si l'expression de désignation est un indicateur d'aiguillage non défini, l'instruction goto est équivalente, en Algol 60, à une instruction vide. Nous avons vu, lors de la traduction d'une déclaration d'aiguillage, que nous le respections dans la traduction.

3.8 INSTRUCTION POUR

Une instruction pour n'augmente pas la puissance d'un langage. Elle a pour but de faciliter la programmation des processus itératifs. Aussi, comme nous l'avons déjà signalé, elle n'apparaît, en Algol 68, qu'au niveau du langage étendu, c'est-à-dire que comme une extension du langage.

Elle est loin d'offrir, en Algol 68, les facilités qu'elle offrait en Algol 60. A première vue, on pourrait considérer que cela constitue une regression par rapport à Algol 60. En fait, son utilisation est réservée aux seules séquences pour lesquelles la génération d'un code objet efficace est possible.

La syntaxe de l'instruction pour d'Algol 68 est la suivante

for I from A by B to C while D do E

où :

1) A, B, C sont des clauses unitaires de mode entier, calculées une fois pour toutes à l'entrée de la boucle - donc non modifiables durant son exécution.

2) I est une variable de mode entier, non modifiable par l'exécution de D et E, et implicitement déclarée par le contexte.

Sont permises les suppressions suivantes :

for I	si I n'apparaît ni dans D, ni dans E
from A	si A = 1
by B	si B = 1
to C	si la valeur limite est inutile
while D	si D = true

Au contraire, la syntaxe de l'instruction pour d'Algol 60 est la suivante :

<instruction pour> ::= for <variable> := <liste de pour> do <instruction>
<liste de pour> ::= <élément de pour> | <élément de pour>, <liste de pour>
<élément de pour> ::= <élément de pour 1ère forme> | <élément de pour 2ème forme> | <élément de pour 3ème forme>

L'instruction pour d'Algol 68 se caractèrise donc, par rapport à celle d'Algol 60, par :

- un seul élément de pour
- pas, valeurs initiale et finale, obligatoirement de mode entier, et non réévalués à chaque pas de la boucle
- variable contrôlée implicitement déclarée par le contexte, et de portée, seulement, l'instruction pour
- non modification de la variable contrôlée par l'exécution de la condition ou de l'instruction contrôlée.

On pouvait même, en Algol 60, changer de variable contrôlée, quand celle-ci était une variable indicée, par effets de bord sur les indices.

Nous diviserons l'étude de la traduction de l'instruction pour d'Algol 60, selon :

1) elle comporte que des éléments de pour de la 1ère forme
2) elle comporte un seul élément de pour, qui est de la 2ème forme
3) elle comporte un seul élément de pour, qui est de la 3ème forme
4) cas général d'une configuration ne rentrant pas dans les catégories précédentes.

Cette classification est justifiée par la complexité de la traduction du cas général, alors qu'une instruction pour d'Algol 60, donnée, rentre le plus souvent dans l'une des trois premières catégories.

Le rôle du premier passage est de déterminer la nature de chaque élément de la liste de pour, de façon à pouvoir choisir au deuxième passage, la traduction la mieux appropriée.

Le concept d'Algol 60 est intégralement retransmis, mais de façon peut être pas toujours très heureuse. Cependant, le danger le plus grave réside dans l'introduction possible d'effets de bord parasites; ce que nous nous sommes efforcés de limiter au maximum.

3.8.1 Element de pour - 1ère forme

Algol 60 :

<instruction pour> ::= for <variable> ::= <liste d'éléments de pour 1ère forme>
do <instruction>
<liste d'éléments de pour 1ère forme> ::= <élément de pour 1ère forme> | <liste d'éléments de pour 1ère forme>, <élément de pour 1ère forme>
<élément de pour 1ère forme> ::= <expression arithmétique>

traduction :

<traduction instruction pour> ::= for <identificateur> to <nombre> do begin
<traduction variable> := case <identificateur> in <traduction liste d'éléments de pour 1ère forme> esac; <traduction instruction> end
<traduction liste d'éléments de pour 1ère forme> ::= <traduction élément de pour 1ère forme> | <traduction liste d'éléments de pour 1ère forme>, <traduction élément de pour 1ère forme>
<traduction élément de pour 1ère forme> ::= <traduction expression arithmétique>

Dans cette traduction :

- <identificateur> désigne un nouvel identificateur - donc à créer - qui est la variable contrôlée de l'instruction pour d'Algol 68
- <nombre> est une dénotation d'entier qui a pour valeur le nombre d'éléments de pour 1ère forme de la liste
- la compatibilité de type de l'affectation interne doit aussi être traitée.

Exemple :

```
Algol 60 :   for I := 2, 4, 6 do A[I] := B[I] / I
Algol 68 :   for I1 to 3 do
                 begin
                     I := case I1 in 2, 4, 6 esac;
                     A[I] := B[I] / I
                 end
```

3.8.2 Elément de pour - 2ème forme

La traduction de cette configuration est immédiate, lorsqu'elle satisfait aux restrictions d'Algol 68 précédemment énoncées. Mais, en raison du problème de la variable contrôlée (non modification et portée), nous la redécomposons, systématiquement, en une suite d'instructions sémantiquement équivalente. Dans cette transformation, nous nous efforçons de ne pas générer d'effets de bord parasites, en évitant d'évaluer deux fois une expression donnée, dans un même pas de la boucle.

Algol 60 :

<instruction pour> ::= for <variable> := <élément de pour 2ème forme> do
<instruction>
<élément de pour 2ème forme> ::= <expression arithmétique 1> step <expression arithmétique 2> until <expression arithmétique 3>

traduction :

<traduction instruction pour> ::= begin <type> <identificateur 1> := <traduction variable> := <traduction expression arithmétique 1>; real <identificateur 2> := <traduction expression arithmétique 2>;
while sign <identificateur 2> * (<identificateur 1> - <traduction expression arithmétique 3>) < = 0 do
begin <traduction instruction>; <identificateur 2> := <traduction expression arithmétique 2>; <identificateur 1> := <traduction variable> := <traduction variable> + <identificateur 2>
end
end

où :

- <type> désigne un déclarateur de mode, celui de <variable>
- <identificateur 1> et <identificateur 2> sont deux nouveaux identificateurs, créés pour éviter la génération d'effets de bord
- le niveau de bloc begin - end est introduit en raison des déclarations de ⟨identificateur 1⟩ et ⟨identificateur 2⟩
- traitement des compatibilités de type des affectations internes.

Exemple :

```
Algol 60 :   for I := 1 step 2 until 6 do A[I] :=  B[I] / I
Algol 68 :   begin
               int I1 := I := 1; real I2 := 2;
               while sign I2 * (I1 - 6) < = 0 do
                   begin
                       A[I] := B[I] / I;
                       I2 := 2; I1 := I := I + I2
                   end
               end
```

3.8.3 Elément de pour - 3ème forme

Algol 60 :

<instruction pour> ::= for <variable> := <élément de pour 3ème forme> do <instruction>

<élément de pour 3ème forme> ::= <expression arithmétique> while <expression booléenne>

traduction :

<traduction instruction pour> ::= while <traduction variable> := <traduction expression arithmétique>; <traduction expression booléenne> do <traduction instruction>

Exemple :

```
Algol 60 :   for I := I + 1 while I < 6 do A[I] := B[I] / I
Algol 68 :   while I := I + 1; I < 6 do A[I] := B[I] / I
```

Nous trouvons ici une configuration particulière, extrêmement concise, de l'instruction pour d'Algol 68.

3.8.4 Cas général

L'instruction pour d'Algol 60 est décomposée en une forme conditionnelle équivalente. Nous définirons, dans un premier temps, le cadre de la traduction, puis la traduction de chaque élément de la liste de pour, selon la forme à laquelle il appartient.

Algol 60 :

<instruction pour> ::= for <variable> := <liste de pour> do <instruction>
<liste de pour> ::= <élément de pour> | <élément de pour>, <liste de pour>
<élément de pour> ::= <élément de pour 1ère forme> | <élément de pour 2ème forme> | <élément de pour 3ème forme>
<élément de pour 1ère forme> ::= <expression arithmétique>
<élément de pour 2ème forme> ::= <expression arithmétique 1> step <expression arithmétique 2> until <expression arithmétique 3>
<élément de pour 3ème forme> ::= <expression arithmétique> while <expression booléenne>

traduction :

<traduction instruction pour> ::= <identificateur 1> : begin ref <type> <identificateur 2> = <traduction variable>; <identificateur 3> := <identificateur 3> + 1; case <identificateur 3> in <traduction liste de pour>, goto <identificateur 4> esac; <traduction instruction>; goto <identificateur 1>; <identificateur 4> : skip end

avec :

- en début de programme, la déclaration suivante : int <identificateur 3> := 0;
- <type> désignant un déclarateur de mode, celui de <variable>
- <identificateur -> désignant toujours un nouvel identificateur, créé pour les besoins de la traduction, et différent de tous ceux figurant dans le programme.

<traduction liste de pour> ::= <traduction élément de pour> | <traduction élément de pour>, <traduction liste de pour>

<traduction élément de pour> ::= <traduction élément de pour 1ère forme> | <traduction élément de pour 2ème forme> | <traduction élément de pour 3ème forme>

<traduction élément de pour 1ère forme> ::= <identificateur 2> := <traduction expression arithmétique>

<traduction élément de pour 2ème forme> ::= begin <identificateur 2> := <traduction expression arithmétique 1>; if sign (<traduction expression arithmétique 2>) * (<traduction expression arithmétique 3> - <identificateur 2>) < 0 then <identificateur 3> := <identificateur 3> + 1; goto <identificateur 1> fi end, begin real <identificateur 5> := <traduction expression arithmétique 2>; <identificateur 2> := <identificateur 2> + <identificateur 5>; if sign <identificateur 5> * (<traduction expression arithmétique 3> - <identificateur 2>) < 0 then goto <identificateur 1> else <identificateur 3> := <identificateur 3> - 1 fi end

<traduction élément de pour 3ème forme> ::= begin <identificateur 2> := <traduction expression arithmétique>; if <traduction expression booléenne> then <identificateur 3> := <identificateur 3> - 1 else goto <identificateur 1> fi end

On remarquera que la traduction d'un élément de pour 2ème forme génère deux éléments de la liste de case - le premier élément correspondant à l'initialisation du processus itératif. On remarquera aussi, comme nous l'avons signalé, qu'un constituant donné : variable contrôlée, expression arithmétique ou booléenne, est évalué une fois et une seule à chaque pas de la boucle.

Exemple :

Algol 60 : for I := 1, 3 step 2 until 8, I + 1 while I < 10 do A[I] := B[I] / I

Algol 68 :

```
ITER : begin
          ref int I1 = I; I2 := I2 + 1;
          case I2 in
             I1 := 1,
             begin
                  I1 := 3; if sign(2) * (8 - I1) < 0 then
                  I2 := I2 + 1; goto ITER fi
             end,
             begin
                  real I3 := 2; I1 := I1 + I3;
                  if sign I3 * (8 - I1) < 0 then goto ITER
                  else I2 := I2 - 1 fi
             end,
             begin
                  I1 := I + 1;
                  if I < 10 then I2 := I2 - 1 else goto ITER fi
             end,
             goto SUITE
          esac;
          A[I] := B[I] / I; goto ITER;
          SUITE : skip
       end
```

avec en début de programme, la déclaration : int I2 := 0;

CHAPITRE IV

EXPRESSIONS

CHAPITRE IV

EXPRESSIONS

4.1 INTRODUCTION

Il existe, en Algol 60, trois types d'expressions :

- expressions arithmétiques
- expressions booléennes
- expressions de désignation

Ces expressions peuvent être conditionnelles ou simples. Si elles sont conditionnelles, à la différence des instructions conditionnelles, elles comportent obligatoirement une partie else. En raison de l'unification, en Algol 68, des concepts d'instruction et d'expression, le cadre de leur traitement sera celui précédemment énoncé à propos des instructions conditionnelles :

- insertion d'un délimiteur fi au premier passage
- traduction de l'expression booléenne
- traduction des parties then et else

Ce processus est évidemment récursif, une partie then ou else pouvant à son tour comporter une expression conditionnelle.

Nous n'envisagerons alors dans ce qui suit - sauf mention - que le traitement des expressions simples.

Une expression est une règle de calcul pour l'obtention d'une valeur. Notre étude portera donc, à la fois, sur les opérandes et les opérateurs, et les valeurs délivrées.

4.2 ELEMENTS DE BASE DU LANGAGE

4.2.1 Nombre

Le mode d'Algol 68 permet aussi de fixer la précision du calcul; le nombre de symboles long précédant le déclarateur real ou int caractérisant le degré de discrimination avec lequel une valeur donnée est conservée en mémoire. Nous ne prenons pas en charge, dans la traduction, cette notion de "précision de calcul"; le mode Algol 68 des quantités manipulées sera toujours real ou int. Des différences pourront apparaître, dues aux options des compilateurs d'Algol 60 et d'Algol 68 sur les encombrements mémoire correspondants.

En Algol 68, un nombre est toujours un nombre sans signe. Ce qui signifie que le signe ne fait pas partie de la dénotation, mais est un opérateur, monadique.

La seule différence concernant l'écriture des nombres (sans signe) est qu'en Algol 68, un nombre ne peut pas être réduit à un facteur de cadrage. Ce problème sera résolu, dès le premier passage, en faisant précéder du chiffre 1 un tel nombre d'Algol 60.

4.2.2 Chaîne

Algol 60 :

<chaîne> ::= '(' <chaîne ouverte> ')'
<chaîne ouverte> ::= <chaîne propre> | '(' <chaîne ouverte> ')' |
        <chaîne ouverte> <chaîne ouverte>
<chaîne propre> ::= <vide> | <toute séquence ne contenant ni '(' ni ')'>

La grammaire de <chaîne ouverte> est visiblement ambiguë, mais ce n'est pas très gênant car elle ne sert pas, en pratique, à la reconnaissance : l'analyseur ne descend pas au-dessous du niveau de <chaîne>. En effet, on remarquera que des représentations différentes ont été choisies pour les guillemets d'ouverture et de fermeture. Ces guillemets ne peuvent apparaître non couplés. De telle sorte que la reconnaissance de la chaîne peut être effectuée par le préprocesseur qui, à l'aide d'un compteur, détermine les deux guillemets les plus extérieurs.

traduction :

<traduction chaîne> ::= " <traduction chaîne ouverte> "
<traduction chaîne ouverte> ::= <chaîne propre> | "" <traduction chaîne ouverte> "" | <traduction chaîne ouverte> <traduction chaîne ouverte>

La traduction des guillemets de chaîne internes est réalisée dès le niveau du préprocesseur. Une chaîne est retransmise dans la traduction, comme succession des caractères qui la composent.

Au contraire, en Algol 68, la même représentation est utilisée pour les deux guillemets d'ouverture et de fermeture de chaîne. Ce délimiteur de chaîne est doublé lorsqu'il est utilisé comme caractère dans la chaîne. Cette façon de procéder ne crée aucun risque d'ambiguïté puisque jamais, syntaxiquement, deux chaînes ne peuvent se suivre.

Nous avons vu, lors de la traduction des déclarations de procédure, qu'il existe en fait, en Algol 68, des dénotations distinctes de caractère et de chaîne. Une dénotation de caractère comporte un et un seul caractère, celle de chaîne, zéro ou au moins deux; une dénotation de chaîne étant définie comme un tableau unidimensionnel de caractères. Nous avons aussi vu, alors, qu'étant donnés les contextes d'apparition possible des chaînes, dans le programme en Algol 60, nous n'avions jamais, pour la traduction, à nous en préoccuper.

4.2.3 Variable indicée

Algol 60 :

<variable indicée> ::= <identificateur> [<liste d'indices>]
<liste d'indices> ::= <expression arithmétique> | <liste d'indices>, <expression arithmétique>

traduction :

<traduction variable indicée> ::= <identificateur> [<traduction liste d'indices>]
<traduction liste d'indices> ::= <traduction expression arithmétique> | <traduction liste d'indices>, <traduction expression arithmétique>

La traduction d'une variable indicée se ramène donc à celle des expressions arithmétiques qui constituent sa liste d'indices.

Un indice, en Algol 68, doit toujours être de mode entier. La traduction de l'expression arithmétique comprend donc aussi la recherche du type de la valeur délivrée en Algol 60, de façon à prendre éventuellement en charge un transfert de type réel vers entier. Nous reviendrons sur ce problème au paragraphe suivant.

4.3 EXPRESSION ARITHMETIQUE

L'étude de la traduction des opérandes ayant été faite, ce paragraphe sera concerné uniquement par celle des opérateurs arithmétiques. Le schéma d'une telle étude est le suivant :

- existence en Algol 68
- domaine d'existence
- priorité
- mode des opérandes
- mode du résultat
- sémantique.

Ce même schéma sera utilisé au paragraphe suivant pour l'étude des opérateurs booléens. Nous ne signalerons, en fait, que les concepts d'Algol 60 modifiés dans la traduction.

En Algol 60, les priorités - déterminant l'ordre d'évaluation - sont définies par la syntaxe. Elles sont donc implicites et fixes. Au contraire, en Algol 68, les priorités sont dynamiques. Tout opérateur a une priorité implicite (figurant dans le standard-prélude) que le programmeur peut, par une "déclaration de priorité", modifier pour la durée de la totalité, ou d'une partie, de son programme.

4.3.1 Opérateur moins unaire

Cet opérateur a, en Algol 68, une priorité supérieure à celle de tout opérateur binaire, contrairement à ce qui se passait en Algol 60 :

<expression arithmétique simple> ::= <terme> | <opérateur additif> <terme> |
<expression arithmétique simple> <opérateur additif> <terme>
<opérateur additif> ::= + | -

de telle sorte que, par exemple :

- a ↑ 3 signifie - (a ↑ 3) en Algol 60
mais (- a) ↑ 3 en Algol 68

Le traitement consiste en un parenthèsage approprié, pour forcer, dans le programme en Algol 68 résultant, l'ordre d'évaluation à ce qu'il était dans le programme en Algol 60.

Cette situation nous amène à remarquer, comme [BOWL], que l'on n'a pas en Algol 68 :

$$0 - x = - x \quad \forall x$$

par exemple :

$$0 - a \uparrow 2 \neq - a \uparrow 2$$

les ordres d'évaluation de ces deux expressions étant respectivement en Algol 68 :

(0 - (a ↑ 2)) et ((- a) ↑ 2)

ce qui est contraire aux règles normalement admises. En fait, la confusion vient de ce que, dans la notation courante, on utilise la même représentation pour deux opérateurs différents.

4.3.2 Opérateur d'exponentiation

Dans ce qui suit,

- i, i1, i2 désignent des quantités de mode entier
- r désigne des quantités de mode réel
- a désigne des quantités de mode entier ou réel

En Algol 60, le mode du résultat - lorsqu'il est défini - est donné par les règles suivantes :

a ↑ i

i > 0	type de a
i = 0, a ≠ 0	type de a
i < 0, a ≠ 0	réel

a ↑ r

a > 0	réel
a = 0, r > 0	réel

Au contraire, en Algol 68, l'opération d'exponentiation n'est défini que dans les configurations suivantes :

i1 ↑ i2	avec	i2 ⩾ 0	résultat de mode entier
		i2 < 0	skip
r ↑ i			résultat de mode réel

C'est-à-dire, par rapport à ce que l'on peut trouver en opérande dans le programme en Algol 60 :

1) l'exposant doit être de mode entier. Sa traduction en comporte donc la vérification, et un message d'erreur serait émis si cette contrainte n'était pas satisfaite.

2) pour une base entière, l'exposant ne peut pas être négatif. C'es une restriction que nous ne pouvons pas vérifier; nous le supposerons donc réalisée.

3) le mode du résultat est celui de la base, comme dans les configurations correspondantes d'Algol 60.

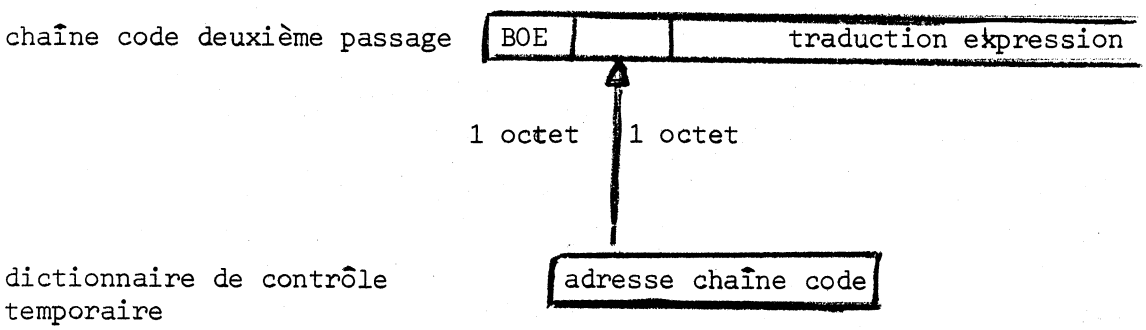
4.3.3 Contrainte de mode réel vers entier

Nous avons déjà signalé et justifié l'absence d'une telle contrainte automatique en Algol 68. C'est au programmeur de la prendre en charge. Nous devons donc la provoquer partout où, dans le programme en Algol 60, elle est implicite :

- borne de déclaration de tableau
- indice de variable indicée
- indice d'indicateur d'aiguillage
- affectation de partie gauche de mode entier
- correspondance de mode paramètre formel - paramètre effectif.

A cet effet, pour retransmettre dans la traduction l'arrondi automatique d'Algol 60, nous utilisons l'opérateur de transfert du standard-prélude : round.

La traduction de l'expression arithmétique comprend donc aussi - dans le contexte d'une telle contrainte - la recherche du mode de la valeur qu'elle délivre. Cette recherche, toujours pour les mêmes raisons d'accès dictionnaire d'identificateurs, n'est entreprise qu'au deuxième passage. Elle utilise un enregistrement dictionnaire de contrôle temporaire de la forme suivante :



Le résultat de cette recherche est inséré en chaîne code, précédé d'un marqueur d'identification, pour récupération au troisième passage.

Une détection de mode réel est faite à l'occurrence de l'un des symboles suivants :

- "." caractéristique d'un nombre réel, point décimal
- "10" d° , facteur de cadrage
- "/" une division, fournissant toujours un résultat de mode réel
- identificateur de mode de base real.

Ce processus d'analyse de mode (non évidemment celui de la traduction) est arrêté quand le résultat "mode réel" est acquis; c'est-à-dire, éventuellement, avant la fin lexicographique de l'expression arithmétique.

En Algol 68, pour qu'une expression arithmétique conditionnelle soit de mode reel, il suffit que l'une des parties then ou else soit de mode réel, l'autre pouvant être de mode entier. C'est une conséquence du "balancing de mode", qui consiste à avoir l'une des deux parties en contexte strong, pour une clause conditionnelle en contexte non strong. Nous en tenons compte, dans la traduction, pour éviter des analyses de mode inutiles.

4.4 EXPRESSION BOOLEENNE

4.4.1 Opérateur de négation

Comme l'opérateur moins unaire des expressions arithmétiques, l'opérateur de négation a, en Algol 68, une priorité supérieure à celle de tout opérateur binaire, contrairement à ce qui se passait en Algol 60 :

<secondaire booléen> ::= <primaire booléen> | ¬ <primaire booléen>
<primaire booléen> ::= <valeur logique> | <variable> | <indicateur de fonction> | <relation> | (<expression booléenne>)
<relation> ::= <expression arithmétique simple> <opérateur de relation> <expression arithmétique simple>

de telle sorte que, par exemple :

¬ a ≮ b signifie ¬ (a < b) en Algol 60
mais (¬ a) < b en Algol 68

Comme pour l'opérateur moins unaire, le traitement réside en un parenthèsage de l'opérande, pour imposer en Algol 68 l'ordre d'évaluation d'Algol 60.

4.4.2 Opérateur d'équivalence

Algol 60 :

<expression booléenne simple> ::= <implication> | <expression booléenne simple> ≡ <implication>

L'opérateur d'équivalence n'a plus, en Algol 68, la même représentation : il y est défini comme un opérateur d'égalité entre quantités booléennes. Son remplacement par un opérateur "égal" est effectué au premier passage.

Sa priorité est, elle aussi, modifiée. Le problème sera résolu par un simple parenthèsage du triplet : opérande - opérateur - opérande.

4.4.3 Opérateur d'implication

Il n'existe pas d'opérateur d'implication dans le standard-prélude d'Algol 68. La solution la plus simple était de le supposer figurer dans le "library prelude" d'une implémentation particulière. Nous avons préféré l'exprimer par une clause booléenne conditionnelle équivalente.

Algol 60 :

<implication> ::= <terme booléen> | <implication> ⊃ <terme booléen>

traduction :

<traduction implication> ::= <traduction terme booléen> | if <traduction implication> then <traduction terme booléen> else true fi

On notera la récursivité de cette règle de traduction : au deuxième passage est inséré un nombre de symboles if égal au nombre d'opérateurs d'implication.

4.5 EXPRESSION DE DESIGNATION

La traduction d'une expression de désignation est celle de ses constituants - soit, étiquette ou indicateur d'aiguillage, puisque nous avons déjà traité le cas d'une expression conditionnelle.

Une expression de désignation peut apparaître :

- en paramètre effectif d'un appel de procédure, correspondant à un paramètre formel de type étiquette

- en élément de la liste d'aiguillage d'une déclaration d'aiguillage
- en opérande d'une instruction goto.

Une expression de désignation, quelque soit le contexte où elle apparaît, représente un branchement potentiel à une certaine étiquette. Etant donnée l'extension, en Algol 68, de la notion de bloc, nous devons traiter, dans la traduction, le problème d'un branchement qui était licite en Algol 60, mais qui ne l'est plus en Algol 68. En effet, comme nous l'avons déjà signalé, il n'est plus possible, en Algol 68, de se renvoyer à l'intérieur d'une instruction composée ou d'une partie then ou else d'instruction conditionnelle - celles-ci étant devenues des clauses sérielles, c'est-à-dire délimitrices de portée. De telle sorte que, par rapport à ce qui était licite en Algol 60, une occurrence d'application d'étiquette peut, dans la traduction, ne plus appartenir à la portée de son occurrence de définition.

La traduction d'une expression de désignation doit donc comporter des tests de compatibilité de portée, pour éventuellement simuler dans le programme en Algol 68, un branchement direct maintenant impossible, et ainsi restituer dans son intégralité le concept d'Algol 60.

Le concept d'aiguillage n'existant plus en Algol 68, nous avons traduit une déclaration d'aiguillage par une déclaration de procédure void à un paramètre entier. Il sera donc tout naturel de traduire un indicateur d'aiguillage comme l'appel de cette procédure - de telle sorte que l'élaboration de la traduction d'un indicateur d'aiguillage est celle du corps de procédure correspondant. Ceci signifie que la traduction d'un indicateur d'aiguillage ne comporte pas de test de portée.

De la même façon, un paramètre formel de type étiquette a été traduit en procédure void, le paramètre effectif fournissant le corps de procédure correspondant. De telle sorte qu'une occurrence d'application d'un paramètre formel de type étiquette représente l'appel de cette procédure void, c'est-à-dire l'élaboration du paramètre effectif. La traduction d'un paramètre formel de type étiquette ne comporte donc pas, non plus, de test de portée.

Il n'y a ainsi de test de portée que pour les étiquettes non paramètres formels, elles seules provoquant un branchement explicite.

4.5.1 Indicateur d'aiguillage

La traduction s'effectuera selon la syntaxe suivante :

Algol 60 :
<indicateur d'aiguillage> ::= <identificateur> [<expression arithmétique>]

traduction :

<traduction indicateur d'aiguillage> ::= <identificateur> (<traduction expression arithmétique>)

La traduction de l'expression arithmétique, paramètre effectif de l'appel de procédure en lequel est traduit l'indicateur d'aiguillage, doit aussi comporter le test de son mode, de façon à l'amener, si besoin est, au mode entier, par application de l'opérateur round - selon ce que nous avons vu au paragraphe des expressions arithmétiques.

4.5.2 Paramètre formel de type étiquette

Une occurrence d'application de paramètre formel de type étiquette n'est pas modifiée dans la traduction. Nous avons vu que, dans le programme en Algol 68 résultant, elle a maintenant pour sémantique l'appel d'une procédure void sans paramètre.

4.5.3 Etiquette non paramètre formel

Comme nous l'avons vu, seule, une étiquette non paramètre formel, provoque un branchement explicite. De telle sorte que, selon ce que nous avons vu être la traduction d'une instruction goto, la traduction d'une étiquette non paramètre formel consiste d'abord à la faire précéder d'un symbole goto. Doit ensuite être effectué le test de portée correspondant.

En cas de branchement à l'intérieur d'une instruction composée ou conditionnelle, nous simulerons le branchement direct maintenant illicite, par un branchement indirect à l'aide d'un indicateur booléen.

Exemple :

Algol 60 :

```
begin
    real y;
    :
    begin
        :
         e : y := 5;
        :
    end;
    :
      goto e;
    :
end
```

Algol 68 :

```
begin
    real y;
    bool id := false;
    :
  e : begin
        if id then id := false; goto e fi;
        :
        e : y := 5;
        :
    end;
    :
     begin
        id := true; goto e
     end;
     :
end
```

Cette même solution est utilisée pour une instruction conditionnelle, mais elle se complique par le fait qu'il est impossible d'accéder directement aux parties then ou else sans passer par la condition.

4.5.4 Branchement intérieur à une instruction composée ou conditionnelle

Le but recherché est que, pour une construction légale en Algol 60, toute occurrence d'application d'une étiquette appartienne, dans le programme en Algol 68 résultant, à la portée d'une occurrence de définition correspondante, par la mise en place éventuelle d'un chaînage effectuant en Algol 68 le même branchement que celui qui était directement réalisable en Algol 60.

Une déclaration d'étiquette - fournissant l'occurrence de définition correspondante - est, en Algol 60 contextuelle. Elle a pour portée le plus petit bloc la contenant.

Pour pouvoir effectuer les tests de portée nécessaires, le premier problème à résoudre est la définition et la mise en place d'une structure de "niveaux d'étiquette", capable de refléter la portée Algol 68, mais plus particulièrement adaptée aux besoins de la traduction. Nous appelerons "niveau d'étiquette" une construction susceptible de nécessiter un branchement indirect, c'est-à-dire la mise en place d'un chaînage intermédiaire.

4.5.4.1 Définition des niveaux d'étiquette

Un niveau d'étiquette est délimité par :

- begin ..... end si le begin n'est précédé ni de then, ni de else - puisque un tel parenthèsage, maintenant inutile, est supprimé dans le programme en Algol 68
- if ..... ;
- then ..... else ou ; s'il n'y a pas de partie else
- else ..... ;

où ";" symbolise la fin lexicographique de l'instruction conditionnelle. Le niveau if, bien que n'étant pas un véritable niveau, est indispensable pour pouvoir accéder à la partie then ou else.

A chaque niveau d'étiquette est associé un enregistrement dictionnaire de contrôle permanent, de la forme suivante :

id	chaînage arrière	supp	ad adj
1 octet	3 octets	1 octet	3 octets

où :

- id est une information d'identification de la nature du niveau d'étiquette : begin, if, then, ou else.
- le chaînage arrière, destiné à refléter l'imbrication des niveaux d'étiquette, contient l'adresse de l'enregistrement dictionnaire de contrôle permanent associé au niveau d'étiquette englobant
- supp a, comme nous le verrons, pour rôle d'éviter la mise en place de chaînages partiels inutiles
- ad adj contiendra l'adresse en chaîne adjonction d'une éventuelle information de chaînage relative à ce niveau d'étiquette.

L'occurrence de définition ne précédant pas nécessairement, en Algol 60, toute occurrence d'application de l'étiquette, la structure de niveaux d'étiquette est mise en place au premier passage, et les tests de portée ne sont effectués qu'au second. Aussi, pour en éviter la regestion au deuxième passage, l'adresse de l'enregistrement dictionnaire de contrôle permanent associé au niveau d'étiquette courant est, au premier passage :

- insérée en chaîne code premier passage dans un contexte de :

  + appel de procédure avec paramètre
  + déclaration d'aiguillage
  + instruction goto

correspondant donc à une occurrence d'application d'étiquette

- insérée dans l'enregistrement dictionnaire d'identificateurs créé lors du traitement d'une déclaration d'étiquette.

4.5.4.2 Test de portée et chaînage

Le traitement - au deuxième passage - d'une occurrence d'application d'étiquette non paramètre formel suit le schéma suivant :

L'adresse de l'enregistrement dictionnaire de contrôle permanent associé au niveau d'étiquette alors courant - dit "niveau d'application", est récupérée en chaîne code.

L'adresse de l'enregistrement dictionnaire de contrôle permanent associé au niveau contenant l'occurrence de définition de l'étiquette - dit "niveau de définition", est récupérée dans l'enregistrement dictionnaire d'identificateurs de l'étiquette.

Un test de compatibilité est fait entre ces deux niveaux. Ceci signifie qu'à partir d'un niveau donné, on peut toujours remonter dans la structure, mais jamais y descendre. Trois cas sont possibles :

1) Les deux niveaux sont les mêmes.
L'occurrence d'application d'étiquette est alors traduite sans difficulté par un "goto e".

2) Le niveau d'application est inférieur au niveau de définition. Il y a donc renvoi à l'intérieur d'une instruction composée ou conditionnelle, et le chaînage correspondant doit être mis en place.

3) Le niveau d'application est supérieur au niveau de définition. On remonte alors la structure, à partir du niveau d'application, à l'aide de l'information de chaînage arrière figurant dans l'enregistrement dictionnaire de contrôle permanent associé à chaque niveau, tant que le niveau reste supérieur au niveau de définition.

Si on arrive ainsi au niveau de définition de l'étiquette, c'est que les portées Algol 60/Algol 68 sont compatibles. L'occurrence d'application de l'étiquette est alors aussi traduite par un "goto e".

Sinon, c'est que l'on finit par arriver à un niveau strictement inférieur à celui de définition et qu'il y a renvoi à l'intérieur d'une instruction composée ou conditionnelle. Et le chaînage correspondant doit être mis en place.

Pour la mise en place d'un chaînage requis, il faut déterminer le plus petit niveau englobant commun aux deux occurrences d'application et de définition de l'étiquette, à partir duquel se fera l'accés à l'occurrence de définition.

Cet accés, comme nous l'avons dit, se fait de façon indirecte, par la mise en place de chaînages intermédiaires correspondant au branchement indirect. Cependant, il faut prendre garde qu'un élément de chaînage, correspondant à une étiquette donnée, ne peut être mis en place qu'une seule fois, sous peine, comme nous le verrons, de l'insertion de multiples déclarations d'une étiquette.

C'est pour cette raison qu'est définie la notion de "niveau d'accés" qui est le niveau le plus extérieur, pour une étiquette donnée, pour lequel le chaînage a déjà été mis en place. L'adresse de l'enregistrement dictionnaire de contrôle permanent qui lui est associé, est tenue à jour et rangée dans l'enregistrement dictionnaire d'identificateurs de l'étiquette.

De telle sorte que la recherche du plus petit niveau englobant est faite entre le niveau d'application et le niveau d'accès, pour tester si en fait le chaînage correspondant n'aurait pas déjà été mis en place. Sinon, on ne fait que le compléter, et mettre à jour le niveau d'accès.

4.5.4.3 Eléments de chaînage

Les informations correspondant aux divers éléments de chaînage sont rangées en chaîne adjonction. Pour un niveau donné, il peut y avoir plusieurs telles informations, correspondant à la mise en place de plusieurs structures de chaînage dues au traitement de plusieurs renvois à l'intérieur d'instructions composées ou conditionnelles. Aussi, les différents éléments d'information sont chaînés entre eux et l'adresse du premier est rangée dans l'enregistrement dictionnaire de contrôle permanent associé à ce niveau. Ils sont récupérés au cours du troisième passage et insérés dans le programme en Algol 68.

L'élément de chaînage est fonction de la nature du niveau. Le premier élément, c'est-à-dire celui correspondant au niveau de définition de l'étiquette est particulier

begin :

<premier élément> ::= <étiquette> : begin if <indicateur> then <indicateur> := false; goto <étiquette> fi;
<autre élément> ::= <étiquette> : begin if <indicateur> then goto <étiquette> fi;

then ou else :

<premier élément> ::= |then / else| if <indicateur> then <indicateur> := false;
goto <étiquette> fi;
<autre élément> ::= |then / else| if <indicateur> then goto <étiquette> fi;

if :

<élément> ::= <étiquette> : if if <indicateur> then |true / false| else <condition> fi

<étiquette> ::= <identificateur>
<indicateur> ::= <identificateur>

Est de plus insérée au début du programme la déclaration avec initialisation suivante :

bool <indicateur> := false;

Quant à la traduction de l'occurrence d'application de l'étiquette non paramètre formel, elle sera :

begin <indicateur> := true; goto <étiquette> end

Le parenthèsage begin - end de la traduction de l'occurrence d'application de l'étiquette est justifié par le fait qu'une instruction goto est, en Algol 60, une instruction de base. Il est donc plus prudent de la traduire par une clause unitaire, en Algol 68.

On remarquera que, dans tous les chaînages intermédiaires, correspondant à un branchement indirect donné, il est toujours possible d'utiliser comme identificateur d'étiquette, celui de l'occurrence en cours de traitement. Cela ne provoque aucun conflit de portée dans le programme en Algol 68.

D'autre part, tous les chaînages relatifs à une même structure utilisent le même indicateur booléen. Il doit donc être, en permanence, disponible. Cet indicateur est créé, et sauvegardé en enregistrement dictionnaire d'identificateurs de l'étiquette, par la création du premier chaînage de la structure relative à cette étiquette. Et sa déclaration est insérée en chaîne adjonction de début de programme.

Il est clair qu'un niveau if ne peut être niveau de définition de l'étiquette. Son seul intérêt vient de ce que l'on ne peut accéder directement à une partie then ou else. Mais, pour le chaînage, il faut mémoriser la nature du niveau de transfert; d'où le true ou le false qui, selon le cas, figurera dans l'information de chaînage relative au niveau if. Quant à la <condition>, elle sera constituée au troisième passage, à partir de :

- la traduction de l'expression booléenne de l'instruction conditionnelle d'Algol 60
- les informations de chaînage, mises en place à ce niveau, relatives aux différentes structures de chaînage.

4.5.4.4 Eléments de chaînages partiellement inutiles

Un élément de chaînage, relatif à un niveau donné, consiste en un branchement conditionnel à un niveau d'étiquette englobé, et permet donc d'éviter l'exécution d'une certaine séquence d'instructions. Ce branchement est inutile quand les deux niveaux sont immédiatement imbriqués; c'est-à-dire pour :

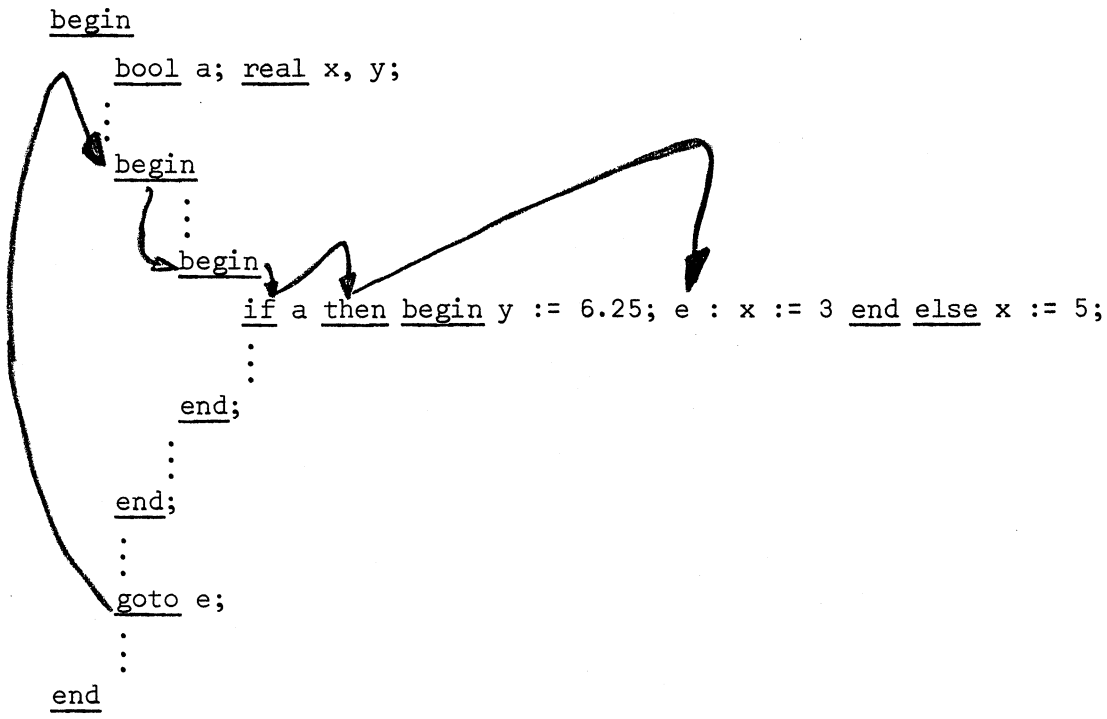
- begin immédiatement suivi de begin ou if
- then ou else immédiatement suivi de if

C'est le rôle de l'information "supp" de l'enregistrement dictionnaire de contrôle permanent : refléter que le niveau précédent est suppressible. Pour cela, au premier passage, à l'ouverture d'un nouveau niveau, on teste si l'ouverture du niveau englobant précède lexicographiquement.

De l'information de chaînage correspondant à ce niveau n'est mise en place que l'étiquette - l'étiquette, en raison de la possibilité de création de chaînages internes ultérieurs.

4.5.4.5 Exemple récapitulatif

Algol 60 :



Algol 68 :

```
begin
    bool id := false;
    bool a; real x, y;
    :
  e : begin
        if id then goto e fi;
        :
      e : begin
            e : if if id then true else a fi then
                    if id then id := false; goto e fi; y := 6.25;
                        e : x := 3 else x := 5 fi;
                :
          end;
        :
    end;
    :
    begin id := true; goto e end;
    :
```

C O N C L U S I O N

CONCLUSION

Une traduction automatique d'Algol 60 en Algol 68 vient d'être proposée. Une première question, fondamentale, que l'on doit se poser, est de savoir si elle est correcte; c'est-à-dire de savoir si le programme obtenu en traduction est de l'Algol 68 correct, et si ce programme fournit bien les mêmes résultats que le programme en Algol 60 source.

Etant donné que nous ne disposons encore d'aucun compilateur d'Algol 68, nous n'avons pour l'instant aucune confirmation de ce que nous avons avancé. Il est donc possible que certaines règles de traduction soient à revoir lorsqu'un compilateur d'Algol 68 - ou même simplement celui du sous-ensemble - sera disponible. Cependant, en raison de leur modularité, de telles modifications devraient a priori n'être que purement locales. D'ailleurs, les difficultés les plus grandes sont venues, non tant du choix de la règle de traduction, que de la localisation et de l'identification de l'information, - démarches qui ne devraient pas être soumises à révision.

La définition d'une traduction doit être faite sur la base d'un rapport, c'est-à-dire d'une description formelle du langage. Mais un tel rapport, n'étant pas toujours suffisamment clair ni précis, est quelquefois source à diverses interprétations. Aussi avons-nous essayé de tenir compte des diverses implémentations qui avaient pu être faites d'Algol 60, par suite d'interprétations différentes du même concept.

Mais, comme nous venons de le dire, cette traduction n'est, et ne peut d'ailleurs être, qu'une traduction de concepts. Nous voulons dire par là que ce qui se passe au niveau de l'évaluation ne nous est pas accessible. Et, comme, dans la définition d'un langage, la formalisation de la partie sémantique est toujours moins poussée que celle de la partie syntaxique, il en résulte, d'une implémentation à l'autre, des variations, dues à des recherches d'optimisation et d'efficacité du code généré, fonctions du matériel utilisé. Il est loin d'être assuré que ces effets seront restitués dans la traduction. Mais ce sont des problèmes d'implémentation Algol 60/Algol 68, et qui n'ont rien à voir avec les langages de référence.

Quant aux restrictions que nous avons apportées à Algol 60, elles viennent de ce qu'un concept d'Algol 60 n'a pas toujours de correspondant immédiat en Algol 68; mais elles restent, le plus souvent, dans le domaine des restrictions "normales" d'un compilateur d'Algol 60. On constatera qu'il n'y a que peu de cas dans lesquels la traduction ne peut pas aboutir, et que l'on peut être pratiquement assuré que ce qu'il est convenu d'appeler un programme en Algol 60 "bien écrit", pourra toujours être traduit et que sa traduction fournira les mêmes résultats, à la précision du calcul près.

Nous avons justifié le fait que la traduction soit directement liée au sous-ensemble d'Algol 68. Cette dépendance pénalise la traduction en raison de la limitation du nombre de concepts d'Algol 68 à utiliser, que nous nous sommes imposée. Peut-être serait-il intéressant de revoir le traducteur dans la perspective d'un compilateur d'Algol 68 complet. La disponibilité de la totalité des concepts d'Algol 68 - notamment des unions de mode -, devrait faciliter la traduction, et peut-être accroître l'efficacité du code généré.

Pour ce qui est enfin de l'intérêt d'un tel traducteur, il est certain que son avenir est d'abord lié à celui d'Algol 68. Mais, même dans la mesure où Algol 68 sera effectivement utilisé, son emploi ne sera que temporaire : il se limitera à la traduction des bibliothèques de sous-programmes en Algol 60. Nous espérons aussi qu'il pourra servir, aux utilisateurs d'Algol 60, de premiere approche à Algol 68. C'est la raison pour laquelle nous nous sommes efforcés d'insister sur l'étude comparative des concepts des deux langages; étude non orientée vers l'implémentation, contrairement à celle qui est faite pour l'écriture d'un compilateur.

A N N E X E

```
INPUT SYNTAX=

 1     PROGRAM          LABLIST BLOC

 2     LABLIST          'ID' M37 'DPT' M1 LABLIST ,

 3     BLOC             M54 'BEGIN' BLOCBODY 'END' M56 M6

 4     BLOCBODY         COMMT M55 DECLIST STATLIST

 5     COMMT            'COMMENT' M5 'PTVIRG' COMMT ,

 6     DECLIST          DEC 'PTVIRG' M1 COMMT DECLIST ,

 7     STATLIST         STAT 'PTVIRG' M1 COMMT STATLIST ,
                        STAT

                        ************************************************
                        *
                        *           DECLARATIONS
                        *
                        ************************************************



 8     DEC              DECTYPE M23 DECITEM ,
                        ARRAYDEC ,
                        PROCDEC ,
                        SWITCHDEC ,
                        OWNDEC

 9     DECTYPE          'REAL' ,
                        'INTEGER' ,
                        'BOOLEAN'

 10    DECITEM          IDLIST ,
                        FUNCTDEC ,
                        'ARRAY' ARRITEM

 11    IDLIST           'ID' M24 'VIRG' M1 IDLIST ,
                        'ID' M24

 12    ARRAYDEC         'ARRAY' M27 ARRITEM

 13    ARRITEM          ARLIST ARRDIM M30 ARRITEMCON

 14    ARRDIM           'CROCG' M1 BCRNLIST 'CROCD'

 15    ARLIST           'ID' M29 'VIRG' ARLIST ,
                        'ID' M29

 16    ARRITEMCON       'VIRG' M31 ARRITEM ,
```

```
17   BORNLIST        BORNITEM M28 'VIRG' M1 BORNLIST ,
                     BORNITEM M28

18   BORNITEM        EXPRA M66 'DPT' M1 EXPRA M66

19   FUNCTDEC        'PROCEDURE' M1 FUNCTITEM

20   FUNCTITEM       'ID' M38 FUNCTITEMCON

21   FUNCTITEMCON    FUNCTHEAD1 COMMT VASP M48 BODY M51 ,
                     FUNCTHEAD2 COMMT M48 BODY

22   FUNCTHEAD1      PARMFORM M39

23   FUNCTHEAD2      'PTVIRG' M11 M39

24   PARMFORM        M11 'PARG' M1 IDFORMLIST 'PTVIRG'

25   VASP            VALPART SPECPART M47

26   BODY            'BEGIN' FUNCTBODY 'END' M50 M6 ,
                     NOBLOC M50

27   FUNCTBODY       COMMT M49 DECLIST STATLIST

28   PROCDEC         'PROCEDURE' M1 PROCITEM

29   PROCITEM        'ID' M40 PROCITEMCON

30   PROCITEMCON     PROCHEAD1 COMMT VASP M52 STAT M53 ,
                     PROCHEAD2 COMMT STAT

31   PROCHEAD1       PARMFORM M12

32   PROCHEAD2       'PTVIRG' M11 M12

33   IDFORMLIST      'ID' M41 'PARD' M1 ,
                     'ID' M41 'PARD' M8 'DPPARG' IDFORMLIST ,
                     'ID' M41 'VIRG' M1 IDFORMLIST

34   VALPART         'VALUE' M42 VALIDLIST 'PTVIRG' COMMT ,

35   VALIDLIST       'ID' M43 'VIRG' VALIDLIST ,
                     'ID' M43

36   SPECPART        SPEC 'PTVIRG' COMMT SPECPART ,
                     SPEC 'PTVIRG' COMMT

37   SPEC            SPECTYPE SPECIDLIST

38   SPECTYPE        DECTYPE M44 DECTYPECON ,
                     DECNOTYPE M45

39   DECTYPECON      'ARRAY' ,
```

```
40   DECNOTYPE       'ARRAY' ,
                     'PROCEDURE' ,
                     'SWITCH' ,
                     'LABEL' ,
                     'STRING'

41   SPECIDLIST      'ID' M46 'VIRG' SPECIDLIST ,
                     'ID' M46

42   SWITCHDEC       'SWITCH' M33 'ID' M24 SWITCHBODY

43   SWITCHBODY      'DPTEG' M34 DESIGNEXLIST M35

44   DESIGNEXLIST    DESIGNEX 'VIRG' M1 DESIGNEXLIST ,
                     DESIGNEX

45   OWNDEC          'OWN' OWNDECITEM

46   OWNDECITEM      DECTYPE M25 OWNIDLIST ,
                     DECTYPE M23 'ARRAY' OWNARRITEM ,
                     OWNARRAYDEC

47   OWNIDLIST       'ID' M24 M26 'VIRG' M1 OWNIDLIST ,
                     'ID' M24 M26

48   OWNARRAYDEC     'ARRAY' M27 OWNARRITEM

49   OWNARRITEM      ARLIST ARRDIM M32 ARRITEMCON1

50   ARRITEMCON1     'VIRG' M31 OWNARRITEM ,


                     ************************************************
                     *
                     *          INSTRUCTIONS
                     *
                     ************************************************


51   STAT            BLOC ,
                     NOBLOC

52   NOBLOC          GOTOST ,
                     IDST ,
                     CONDST ,
                     FORST ,
                     M3

53   GOTOST          'GOTO' M67 DESIGNEX

54   IDST            'ID' M37 'DPT' M1 STAT ,
                     'ID' M22 'PARG' M67 PARLISTPARD ,
                     'ID' M22 X 'DPTEG' M1 AFF ,
```

```
                    'ID' M22

55   AFF            M13 EXPR M21 'DPTEG' M1 AFF ,
                    M13 EXPR M21

56   CCNDST         'IF' M57 EXPR M66 CHOICEST

57   CHOICEST       'THEN' M58 IFST M36 M2

58   IFST           BLOC ELSEPART ,
                    GOTOST ELSEPART ,
                    'ID' M22 ELSEPART ,
                    'ID' M22 'PARG' M67 PARLISTPARD ELSEPART ,
                    'ID' M22 X AFFST ELSEPART ,
                    'ID' M37 'DPT' M1 IFST ,
                    FORST M36 ,
                    M3 ELSEPART

59   AFFST          'DPTEG' M1 AFF

60   ELSEPART       M36 'ELSE' M59 STAT M36 ,
                    M36

61   FORST          'FOR' M60 'ID' M22 X FORSTBODY

62   FORSTBODY      'DPTEG' M1 FORLIST 'DO' FORSTAT

63   FORSTAT        M64 STAT M65

64   FORLIST        FORELEM 'VIRG' M1 FORLIST ,
                    FORELEM

65   FORELEM        EXPRA M66 M61 ,
                    EXPRA M66 STEPEXPRA UNTILEXPRA ,
                    EXPRA M66 'WHILE' M63 EXPR M66

66   STEPEXPRA      'STEP' M62 EXPRA M66

67   UNTILEXPRA     'UNTIL' M1 EXPRA M66

                    ************************************************************
                    *
                    *           EXPRESSIONS
                    *
                    ************************************************************



68   DESIGNEX       UNCONDESEX ,
                    CONDESEX

69   CCNDESEX       'IF' M1 EXPR M66 THENDESEX ELSEDESEX

70   THENDESEX      'THEN' M1 UNCONDESEX
```

```
72  UNCONDESEX      'ID' M22 ,
                    'ID' M22 AIGINDX ,
                    'PARG' DESIGNEX 'PARD'

73  AIGINDX         'CROCG' M9 EXPRA M66 'CROCD' M10

74  EXPR            CONDEXPR ,
                    UNCONDEXPR

75  CONDEXPR        'IF' M1 EXPR M66 THENEXPR ELSEXPR

76  THENEXPR        'THEN' M1 UNCONDEXPR

77  ELSEXPR         'ELSE' M1 EXPR M2

78  UNCONDEXPR      M13 IMPLIC M14 'EQUIV' M11 UNCONDEXCON ,
                    M13 IMPLIC M14

79  UNCONDEXCON     M9 IMPLIC M10 'EQUIV' M11 UNCONDEXCON ,
                    M9 IMPLIC M10

80  IMPLIC          M15 TERMBOOL 'IMPLI' M17 IMPLICON ,
                    M15 TERMBOOL M16

81  IMPLICON        M18 TERMBOOL M19 'IMPLI' M17 IMPLICON ,
                    M18 TERMBOOL M19 M16

82  TERMBOOL        FACTBOOL 'UNION' M1 TERMBOOL ,
                    FACTBOOL

83  FACTBOOL        SECBOOL 'INTER' M1 FACTBOOL ,
                    SECBOOL

84  SECBOOL         'NON' M1 M9 EXPRB M10 ,
                    EXPRB

85  EXPRB           BOOL M1 ,
                    UNCONDEXPRA

86  EXPRA           CONDEXPRA ,
                    UNCONDEXPRA

87  CONDEXPRA       'IF' M1 EXPR M66 THENEXPRA ELSEXPRA

88  THENEXPRA       'THEN' M1 UNCONDEXPRA

89  ELSEXPRA        'ELSE' M1 EXPRA M2

90  UNCONDEXPRA     EXAFTEROP ,
                    'PLUS' M1 EXAFTEROP ,
                    'MOINS' M1 M13 PRIMARY M20 ,
                    'MOINS' M1 M13 PRIMARY M20 NONEXPPART ,
                    'MOINS' M1 M13 PRIMARY M20 EXPPART

91  EXAFTEROP       PRIMARY Y

92  NONEXPPART      NONEXP M1 EXAFTEROP
```

```
93   EXPPART        'EXP' M1 PRIMARY M10 M66 Y

94   BOOL           'TRUE' ,
                    'FALSE'

95   PRIMARY        'ID' M22 'PARG' M67 PARLISTPARD ,
                    'ID' M22 X ,
                    'PARG' M1 EXPR 'PARD' M1 ,
                    NUMBER

96   X              'CROCG' M1 EXLIST 'CROCD' M1 ,

97   EXLIST         EXPRA M66 'VIRG' M1 EXLIST ,
                    EXPRA M66

98   PARLISTPARD    PARM M66 'PARD' M1 ,
                    PARM M66 'PARD' M8 'DPPARG' PARLISTPARD ,
                    PARM M66 'VIRG' M1 PARLISTPARD

99   PARM           CHAINE ,
                    EXPR

100  CHAINE         'OPEN' M7 'CLOSE'

101  Y              BINOP M1 EXAFTEROP ,

102  BINOP          'EXP' ,
                    NONEXP

103  NONEXP         'PLUS' ,
                    'MOINS' ,
                    'MULT' ,
                    'DIV' ,
                    'INTDIV' ,
                    'INF' ,
                    'INFEG' ,
                    'SUP' ,
                    'SUPEG' ,
                    'EGAL' ,
                    'NONEG'

104  NUMBER         ENTIER UNINTCON ,
                    DECPART EXPONENT ,
                    FACTCAD

105  UNINTCON       DECPART EXPONENT ,
                    EXPONENT

106  FACTCAD        'DIX' M4 SIGNINT

107  DECPART        'POINT' M1 ENTIER

108  EXPONENT       'DIX' M1 SIGNINT ,
```

```
109   SIGNINT          UNOP M1 ENTIER ,
                       ENTIER

110   UNOP             'PLUS' ,
                       'MOINS'

111   ENTIER           DIGIT M1 ENTIER ,
                       DIGIT M1

112   DIGIT            'ZERO' ,
                       'ONE' ,
                       'TWO' ,
                       'THREE' ,
                       'FOUR' ,
                       'FIVE' ,
                       'SIX' ,
                       'SEVEN' ,
                       'EIGHT' ,
                       'NINE'

113    /// END OF ULD GRAMMAR

ULD GRAMMAR:LAST CLASSNAME= 430

BNF GRAMMAR : LAST CLASSNAME= 430
```

LISTE DES PRINCIPALES ROUTINES DE TRAITEMENT

PREMIER PASSAGE

INITIAL
ROUTINE D'INITIALISATION DU TRADUCTEUR
INITIALISE LES DIFFERENTES VARIABLES DE TRAVAIL - EN PARTICULIER,
CREE LA TABLE DE HASH-CODING DES MOTS CLES ET INITIALISE CELLE
DES IDENTIFICATEURS AVEC LES PROCEDURES DE BIBLIOTHEQUE D'ALGOL 60
PASSE LE CONTROLE A L'ANALYSEUR APRES CODAGE DU PREMIER SYMBOLE DU
PROGRAMME SOURCE

CHSCR
ROUTINE DE VERIFICATION DE L'EXACTITUDE SYNTAXIQUE DU PROGRAMME SOURCE
EST APPELEE PAR L'ANALYSEUR POUR COMPARER LE TERMINAL COURANT DE LA
GRAMMAIRE AVEC LE SYMBOLE COURANT DU PROGRAMME SOURCE
ASSURE DEJA LE CODAGE DU SYMBOLE SUIVANT DU PROGRAMME PAR APPEL DU
PREPROCESSEUR

PPROC
PREPROCESSEUR
PROVOQUE L'ACQUISITION PAR LA ROUTINE UNCAR DU SYMBOLE COURANT DU
PROGRAMME SOURCE ET DELIVRE LE CODE INTERNE CORRESPONDANT
IDENTIFICATEURS TRONQUES A 8 CARACTERES - CODE INTERNE SUR 3 OCTETS :
CODE IDENTIFICATEUR ET NUMERO D'IDENTIFICATEUR SUR 2 OCTETS
D'OU UNE CAPACITE DE 65.535 IDENTIFICATEURS DIFFERENTS

UNCAR
ROUTINE DE LECTURE DU CARACTERE COURANT DU PROGRAMME SOURCE
PASSE LE CONTROLE AU GENERATEUR (SECOND PASSAGE) A LA DETECTION DE LA FIN DU PROGRAMME SOURCE

FONCTIONS SEMANTIQUES
CONSTITUENT LE PREMIER PASSAGE DU TRADUCTEUR
APPELS INSERES DANS LA GRAMMAIRE ET EFFECTUES PAR L'ANALYSEUR
CONSTRUCTION DE LA CHAINE CODE PREMIER PASSAGE

M1
RETRANSMISSION EN CHAINE CODE PREMIER PASSAGE DU SYMBOLE COURANT

M2
PARENTHESAGE D'UNE INSTRUCTION CONDITIONNELLE PAR INSERTION D'UN SYMBOLE FI

M3
TRADUCTION D'UNE INSTRUCTION VIDE D'ALGOL 60 PAR INSERTION D'UN SYMBOLE SKIP

M4
FAIT PRECEDER DU CHIFFRE 1, UN NOMBRE D'ALGOL 60 REDUIT A UN FACTEUR DE CADRAGE

M5 M6 M8
RETRANSMISSION EN CHAINE CODE PREMIER PASSAGE D'UN COMMENTAIRE

M7
RETRANSMISSION D'UNE CHAINE

M13 M14
TRADUCTION D'UNE EQUIVALENCE
INSERTION, PAR M14, D'UNE PARENTHESE GAUCHE, A L'ADRESSE CHAINE CODE PREMIER PASSAGE MEMORISEE PAR M13 EN DICTIONNAIRE DE CONTROLE TEMPORAIRE

M15 M16 M17 M18 M19
TRADUCTION D'UNE IMPLICATION
INSERTION EN CHAINE CODE PREMIER PASSAGE, PAR M15, A LA SUITE D'UN MARQUEUR D'IDENTIFICATION, DU NOMBRE D'OPERATEURS D'IMPLICATIONS (COMPTE PAR M17), CORRESPONDANT AU NOMBRE DE SYMBOLES IF QUI SERONT INSERES AU SECOND PASSAGE, A L'ADRESSE CHAINE CODE PREMIER PASSAGE MEMORISEE

M13 M20
TRAITEMENT OPERATEUR MOINS UNAIRE PAR PARENTHESAGE OPERANDE
MEME PRINCIPE QUE L'EQUIVALENCE

M22
RETRANSMISSION D'UN IDENTIFICATEUR
SORTIE EN CHAINE CODE PREMIER PASSAGE DU CODE IDENTIFICATEJR ET DU NJ4ERO D'IDENTIFICATEUR

M23
TRAITEMENT DU TYPE D'UNE DECLARATION
CALCUL DU TYPE INTERNE EN DICTIONNAIRE D'IDENTIFICATEURS

M24
TRAITEMENT D'UNE DECLARATION DE VARIABLE SIMPLE JU D'UN AIGJILLAGE
SORTIE DE L'IDENTIFICATEUR EN CHAINE CODE PREMIER PASSAGE PAR APPEL M22
ENREGISTREMENT EN DICTIONNAIRE D'IDENTIFICATEJRS

NEWID
ROUTINE DE CREATION D'UN NOUVEL IDENTIFICATEJR
REPRESENTATION EXTERNE ATTRIBUEE : ID1 A ID65535
ENREGISTREMENT EN TABLE DE HASH-CODING DES IDENTIFICATEJRS

M26
TRADUCTION DE LA DECLARATION D'UNE VARIABLE SIMPLE REMANENTE
SAUVEGARDE EN CHAINE ADJONCTION DE DEBUT DE PROGRAMME DES DECLARATIONS DE NOUVEAUX IDENTIFICATEURS CREES

M28
DECLARATION DE TABLEAU
CALCUL DE LA DIMENSION

M29 M30
DECLARATION DE TABLEAU
ENREGISTREMENT EN DICTIONNAIRE D'IDENTIFICATEURS
RESTRUCTURATION DE LA DECLARATION D'ALGOL 60

M29 M32
TRADUCTION DECLARATION DE TABLEAU REMANENT
ENREGISTREMENT EN DICTIONNAIRE D'IDENTIFICATEURS
SAUVEGARDE EN CHAINE ADJONCTION DE DEBUT PROGRAMME DES DECLARATIONS DES NOUVEAUX IDENTIFICATEURS CREES
SAUVEGARDE DES ELEMENTS ENCORE ACCESSIBLES
ACQUISITION DES NOUVELLES BORNES ET DES ELEMENTS SAUVEGARDES

M33 M34 M35
DECLARATION D'AIGUILLAGE
INSERTION EN CHAINE CODE PREMIER PASSAGE DU NIVEAU D'ETIQUETTE COURANT
RESTRUCTURATION EN UNE DECLARATION DE PROCEDURE

NEWNIV
OUVERTURE D'UN NOUVEAU NIVEAU D'ETIQUETTE
CREATION DE L'ENREGISTREMENT DICTIONNAIRE DE CONTROLE PERMANENT
CORRESPONDANT

M36
FIN DU NIVEAU D'ETIQUETTE COURANT

M37
TRAITEMENT D'UNE OCCURRENCE DE DEFINITION D'ETIQUETTE
SORTIE DE L'IDENTIFICATEUR EN CHAINE CODE PREMIER PASSAGE
ENREGISTREMENT EN DICTIONNAIRE D'IDENTIFICATEURS

M38
DECLARATION DE FONCTION PROCEDURE
SORTIE DE L'IDENTIFICATEUR EN CHAINE CODE PREMIER PASSAGE
CREATION DES ENREGISTREMENTS DICTIONNAIRE D'IDENTIFICATEURS ET
DICTIONNAIRE DE CONTROLE PERMANENT
ENREGISTREMENT EN DICTIONNAIRE DE CONTROLE TEMPORAIRE DE LA VARIABLE
DE TRAVAIL - ACCUMULATEUR

M40
DECLARATION DE PROCEDURE PROPRE
SORTIE DE L'IDENTIFICATEUR EN CHAINE CODE PREMIER PASSAGE
CREATION DES ENREGISTREMENTS DICTIONNAIRE D'IDENTIFICATEURS ET
DICTIONNAIRE DE CONTROLE PERMANENT

M41 M43 M44 M45 M46
RECOPIE DE LA LISTE DES PARAMETRES FORMELS EN CHAINE CODE PREMIER PASSA
CALCUL DU TYPE INTERNE D'UN PARAMETRE FORMEL
PRISE EN COMPTE DU MODE D'APPEL

NEWBLOC
OUVERTURE D'UN NOUVEAU NIVEAU DE BLOC
OUVERTURE D'UN NOUVEAU NIVEAU EN DICTIONNAIRE D'IDENTIFICATEURS
CREATION EN DICTIONNAIRE D'IDENTIFICATEURS DE L'ENREGISTREMENT
DE DEBUT DE BLOC

NDBLOC
FIN DU NIVEAU DE BLOC COURANT
RANGEMENT DU DICTIONNAIRE D'IDENTIFICATEURS DU BLOC TERMINE ET
SAUVEGARDE DE L'ADRESSE DE RANGEMENT

47
CREATION DU NIVEAU DE BLOC VIRTUEL DE LA PARTIE SPECIFICATION
ENREGISTREMENT EN DICTIONNAIRE D'IDENTIFICATEURS DES SPECIFICATIONS
CONSTITUTION EN CHAINE ADJONCTION DU MODE ALGOL 53 DE LA PROCEDURE PROPRE
OU DE LA FONCTION PROCEDURE

48 M49 M50 M51
TRAITEMENT DU CORPS D'UNE FONCTION PROCEDURE
RETRANSMISSION DES NIVEAUX DE BLOC ET D'ETIQUETTE EXISTANTS, OU, SINON,
CREATION
GESTION DE LA VARIABLE DE TRAVAIL DESTINEE A RECEVOIR LA VALEUR
DE L'INDICATEUR DE FONCTION
FIN DU NIVEAU DE BLOC VIRTUEL DE LA PARTIE SPECIFICATION

52 M53
TRAITEMENT DU CORPS D'UNE PROCEDURE PROPRE AVEC PARAMETRE
RETRANSMISSION DES NIVEAUX DE BLOC ET D'ETIQUETTE EXISTANTS, OU, SINON,
CREATION SI IL EXISTE UNE PARTIE VALEUR
FIN DU NIVEAU DE BLOC VIRTUEL DE LA PARTIE SPECIFICATION

54 M55 M56
TRAITEMENT D'UN PARENTHESAGE DE BLOC OU D'INSTRUCTION COMPOSEE
SUPPRESSION D'UN TEL PARENTHESAGE DE PARTIE THEN OU ELSE D'INSTRUCTION
CONDITIONNELLE
CREATION EVENTUELLE D'UN NOUVEAU NIVEAU DE BLOC ET D'ETIQUETTE

57 M58 M59
INSTRUCTION CONDITIONNELLE
CREATION DES NIVEAUX D'ETIQUETTE ASSOCIES

60 M61 M62 M63 M64 M65
INSTRUCTION POUR
DETERMINATION ET SAUVEGARDE DE LA FORME DES DIFFERENTS ELEMENTS DE POUR

66
INSERTION EN CHAINE CODE PREMIER PASSAGE D'UN DELIMITEUR DE FIN D'EXPRESSION

67
INSERTION EN CHAINE CODE PREMIER PASSAGE DU NIVEAU D'ETIQUETTE COURANT
INSTRUCTION POUR ET APPEL DE PROCEDURE

SECOND PASSAGE

CONSTRUCTION DE LA CHAINE CODE SECOND PASSAGE

GENERE
ROUTINE DE CONTROLE ET D'ENCHAINEMENT DU SECOND PASSAGE
PASSE LE CONTROLE A L'EDITEUR (TROISIEME PASSAGE) A LA DETECTION DE LA FIN DE LA CHAINE CODE PREMIER PASSAGE
RENVOI A UNE ROUTINE SPECIFIQUE DE TRAITEMENT SELON LA NATURE DE L'ELEMENT D'INFORMATION COURANT

GBEGNDBL
TRAITEMENT DE LA STRUCTURE DE BLOC
GESTION D'UN DISPLAY POUR LA RECHERCHE DICTIONNAIRE DANS LES BLOCS ENTRES

GBEGNDIC
GESTION DU PARENTHESAGE D'UNE INSTRUCTION COMPOSEE

GDECVS
RETRANSMISSION D'UNE DECLARATION DE VARIABLE SIMPLE

GDECVSR
RETRANSMISSION D'UNE DECLARATION DE VARIABLE SIMPLE REMANENTE

GCOMMT
RETRANSMISSION D'UN COMMENTAIRE

GIF
TRAITEMENT D'UNE PROPOSITION SI

GFOR
TRAITEMENT D'UNE INSTRUCTION POUR
CHOIX DE LA TRADUCTION SELON LA CONFIGURATION DE LA LISTE DE POUR, DETERMINEE AU PREMIER PASSAGE

GARRAY
TRAITEMENT D'UNE DECLARATION DE TABLEAU REMANENT OU NON
TRAITEMENT DES BORNES

'ROC
RETRANSMISSION EN CHAINE CODE SECOND PASSAGE DE LA TETE D'UNE DECLARATION DI
PROCEDURE

LIG
TRAITEMENT D'UNE DECLARATION D'AIGUILLAGE
TRADUCTION DES EXPRESSIONS DE DESIGNATION DE LA LISTE D'AIGUILLAGE

GOTO
TRADUCTION DE L'EXPRESSION DE DESIGNATION - OPERANDE DU GOTO

ESITEM
TRADUCTION D'UNE EXPRESSION DE DESIGNATION :
INDICATEUR D'AIGUILLAGE
TEST DE PORTEE D'ETIQUETTE

ID
TRAITEMENT D'UNE INSTRUCTION COMMENCANT PAR ID
RENVOI A UNE ROUTINE SPECIFIQUE :
ETIQUETTE
AFFECTATION
APPEL DE PROCEDURE
PROCEDURE D'ENTREE-SORTIE

ROC
TRAITEMENT D'UN APPEL DE PROCEDURE NON FORMELLE
TRADUCTION D'UN PARAMETRE EFFECTIF SELON LE TYPE DU PARAMETRE FORMEL
CORRESPONDANT, ET SON MODE D'APPEL
ACQUISITION DU MODE D'UN PARAMETRE FORMEL DE TYPE TABLEAU DU PROCEDURE

ROCFORM
ANALOGUE A PROC, MAIS RECHERCHE DES MODES DES PARAMETRES FORMELS
CORRESPONDANTS

FF
TRAITEMENT D'UNE AFFECTATION EVENTUELLEMENT MULTIPLE
TRADUCTION DE LA LISTE DE PARTIES GAUCHES
TRADUCTION DE L'EXPRESSION PARTIE DROITE

DX
TRADUCTION DE LA LISTE D'INDICES D'UNE VARIABLE INDICEE NON FORMELLE

DXFORM
TRAITEMENT D'UNE VARIABLE INDICEE FORMELLE

EXPR
TRADUCTION D'UNE EXPRESSION ARITHMETIQUE OU BOOLENNE - OPERATEURS ET OPERANDES
APPEL D'UNE FONCTION PROCEDURE FORMELLE OU NON FORMELLE
VARIABLE INDICEE FORMELLE OU NON FORMELLE
FONCTION STANDARD

EXPRINT
IDEM EXPR, MAIS RECHERCHE DU TYPE D'UNE EXPRESSION ARITHMETIQUE
CONTROLE REPASSE A EXPR DES IDENTIFICATION TYPE REEL

SEARCHID
RECHERCHE D'UN IDENTIFICATEUR EN DICTIONNAIRE D'IDENTIFICATEURS, SELON LA STRUCTURE DE BLOC COURANTE
TRAITEMENT DU PROBLEME D'IDENTIFICATION DES IDENTIFICATEURS DANS LES BORNE DES TABLEAUX

TROISIEME PASSAGE

EDITION
TRAITEMENT DE LA CHAINE CODE SECOND PASSAGE
RETOUR A UNE REPRESENTATION EXTERNE DE L'INFORMATION
PRISE EN CHARGE D'UN CADRAGE DE VISUALISATION DE LA STRUCTURE D'IMBRICATION DES BLOCS
SORTIE EN ENREGISTREMENTS LOGIQUES DE 72 OCTETS
MOTS CLES RECUPERES EN TABLE DE HASH-CODING PAR CODE PREPROCESSEUR, ET SORTIS ENTRE QUOTES
IDENTIFICATEURS RECUPERES EN TABLE DE HASH-CODING PAR NUMERO INTERNE
INSERTION DU PARENTHESAGE ALGOL 68 DES CHAINES
RECUPERATION DE L'INFORMATION SAUVEGARDEE EN CHAINE ADJONCTION DE DEBUT DE PROGRAMME
RECUPERATION PAR LES ENREGISTREMENTS EN DICTIONNAIRE DE CONTROLE PERMANENT DE L'INFORMATION SAUVEGARDEE EN CHAINE ADJONCTION :
TRADUCTION DE LA TETE D'UNE DECLARATION DE PROCEDURE, CAD DES MODES ALGOL 68 DES PARAMETRES FORMELS, SELON LE MODE D'APPEL
TRAITEMENT DES IDENTIFICATEURS DANS LES BORNES DES TABLEAUX
ELEMENTS DE CHAINAGE DUS A LA TRADUCTION D'UNE EXPRESSION DE DESIGNATION

B I B L I O G R A P H I E

B I B L I O G R A P H I E

{BACK} J.W. BACKUS, the syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference - IFIP Paris, june 1959.

{BERT} M. BERTHAUD, Algol-to-PL/I language conversion program for IBM System/360 operating system (SRL)-form C33-2000.
Algol-to-PL/I language conversion program for IBM System/360 operating system (PLM)-form Y35-7006.

{BORD} J. BORDIER, Méthodes pour la mise au point de grammaires LL(1). Thèse 3° Cycle. I.M.A.G. janvier 1971.

{BOUS} J.C. BOUSSARD, J.J. DUBY, (ed.). Rapport d'évaluation Algol 68. I.M.A.G. - C.S. IBM-France. Grenoble. juillet 1970.

{BOWL} H.J. BOWLDEN, Algol 68 - comments and recommendations. Algol bulletin AB31.3.3.

{BRAN1} P. BRANQUART, J. LEWI, A sheme of storage allocation and garbage collection for Algol 68. M.B.L.E. Bruxelles. Report R 133. april 1970.

{BRAN2} P. BRANQUART, J. LEWI, General principles of an Algol 68 garbage collector. M.B.L.E. Bruxelles. technical note N60. january 1970.

{GRIF1} M. GRIFFITHS, Langages algorithmiques et compilateurs - section C4 Maîtrise Informatique. I.M.A.G. octobre 1969. pages 56 à 58.

{GRIF2} M. GRIFFITHS, One pass type checking in PL/1 - SEAS Grenoble. september 1969.

{GRIF3} M. GRIFFITHS, Analyse déterministe et compilateurs. Thèse I.M.A.G. octobre 1969.

{GRIF4} M. GRIFFITHS, M. PELTIER, A macro-generable language for the 360 computer - computer bulletin. november 1969.

{GRIF5} M. GRIFFITHS, M. PELTIER, Grammar transformation as an aid to compiler production. University of Grenoble. february 1968.

{IBM} IBM System/360 Operating system : Algol language - form C28 - 6615-1.

{KOST} C.H.A. KOSTER, On infinite modes.Algol bulletin AB30.3.3.

{LAND} A. LANDELLE, J. PLEYBER, A definition of the translation of Algol 60 to Algol 68. University of Grenoble. Technical note. october 1969.

{LIND} C.H. LINDSEY, An ISO-code representation for Algol 68. Algol bulletin AB31.3.6.

{MAIL} B. MAILLOUX, On the implementation of Algol 68. Thèse.Amsterdam. june 1968.

{NAUR} P. NAUR, (ed.). et al. Report on the algorithmic language Algol 60. 1960.

and

P. NAUR, (ed.). et al. Revised report on the algorithmic language Algol 60. 1962.

{PETI} M. PETIT, Cours d'Algol sur les machines IBM de la série 360 sous système OS. I.M.A.G. Mars 1968.

{PLEY} J. PLEYBER, Thèse 3° Cycle Informatique. A paraître, I.M.A.G.

{VANC} Proceedings of an informal conference on Algol 68 implementation 29-30 august 1969. Department of computer science, university of british Columbia,VANCOUVER.

{VAND} S.G. Van Der MEULEN, and C.H. LINDSEY, Informal introduction to Algol 68. 4 th draft - Mathematish Centrum. Amsterdam.august 1969.

{VANW1} A. Van WIJNGAARDEN, (ed.). et al. Report on the algorithmic language Algol 68. february 1969.

{VANW2} A. Van WIJNGAARDEN, Orthogonal design and description of a formal language. Mathematish Centrum. Amsterdam,MR76. october 1965.

VU

Grenoble, le

Le Président de la Thèse

VU

Grenoble, le

Le Doyen de la Faculté des Sciences

VU, et permis d'imprimer
Le Recteur de l'Académie de GRENOBLE