

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR ES SCIENCES

Informatique

par

Michel CAPLAIN



LANGAGE de SPECIFICATIONS



Thèse soutenue le 20 décembre 1978 devant la Commission d'Examen :

Président : B. VAUQUOIS

Examineurs : J.R. ABRIAL

L. BOLLIET

A. COLMERAUER

P. DESVERGNES

A. GUILLON

P. JORRAND

RESUME

- Le problème qui est à l'origine de cette thèse est de fournir aux industriels les moyens d'obtenir des programmes certifiés conformes à des spécifications.

L'insuffisance des techniques classiques de vérification a posteriori (chapitre 0) me conduit à proposer un système de Programmation Assistée par Ordinateur garantissant la production de programmes corrects par une méthode de transformations de spécifications: il s'agit de passer de la spécification au programme par une suite de transformations dont chacune est prouvée valide. Donc le système oblige et aide le concepteur à documenter et justifier chacun de ses choix d'implémentation.

Le principal ingrédient de ce système est un langage permettant d'exprimer à la fois: la spécification initiale (Intention), les spécifications intermédiaires (Documentation), le programme final et les raisonnements justifiant chaque transformation.

Ce langage de spécifications doit satisfaire à quatre exigences apparemment contradictoires:

- . être clair, pour que le texte de la garantie soit lisible, crédible et convaincant ;
- . tolérer les ambiguïtés, puisqu'on ne peut jamais les exclure complètement (chapitre I) ;
- . être défini de façon suffisamment simple et rigoureuse pour se prêter à des preuves formelles ;
- . être d'un emploi facile pour l'utilisateur.

Un tel langage peut évidemment aussi servir à la Conception Assistée de produits autres que des programmes, et à un vaste sous-ensemble de l' "Intelligence Artificielle".

On trouvera ici une présentation aussi intuitive que possible du langage de spécifications que je propose.

- Syntaxe du langage: Pour accommoder l'infinie diversité des applications et la simplicité d'utilisation, le langage doit être "dynamiquement extensible" afin d'épouser progressivement la forme d'expression choisie par l'utilisateur.

La grammaire est donc évolutive ; elle comporte à tout instant un ensemble d'opérateurs dits "généralisés" parce que leurs arguments et résultat sont délimités par des "types" qui - au lieu d'être déterminés - peuvent être seulement liés par une relation (et chaque opérateur a une syntaxe propre).

L'ensemble de ces types est lui-même évolutif et défini à tout instant par des types de base et des constructeurs de types. Cet ensemble est de plus, muni d'un ordre partiel correspondant à l'inclusion (chapitre I).

Chaque identifieur ou expression apparaissant dans le langage a un type qui, s'il n'est pas exactement connu, peut être approché par des bornes inférieure et supérieure (ambiguïté de types).

L'analyse syntaxique (chapitre II) a pour but d'établir pour chaque phrase de l'utilisateur un arbre syntaxique conforme à la syntaxe propre de chaque opérateur. S'il y a zéro ou plusieurs arbres, le système pose des questions dont les réponses permettront de déduire les extensions nécessaires (apprentissage, par le système, du langage de l'utilisateur).

L'utilisateur est sensé ne rien connaître de la grammaire et des types ; son seul rôle est de s'exprimer en respectant quelques conventions, de répondre aux questions, et de changer certains symboles si le système le lui demande.

- Signification du langage

L'utilisation du langage suppose qu'on puisse prouver qu'une spécification est conforme à (c'est-à-dire implique) une autre. Pour ce faire, on définit par des "règles d'inférence" une sémantique déductive du langage (chapitre III) qui, à tout ensemble de phrases (même ambiguës) associe les phrases qui en sont des conséquences certaines.

Un démonstrateur de théorèmes est donc nécessaire pour au moins approximer cette sémantique.

Parmi les divers critères de qualité applicables à tout démonstrateur, on montre que le plus important est l' "autonomie" (difficulté des théorèmes qu'il peut prouver en un temps donné) ; en effet, en améliorant l'autonomie, on diminue d'autant l'intervention requise de l'utilisateur. Malheureusement, l'autonomie ne sera jamais suffisante pour que le démonstrateur puisse être entièrement automatique: il sera nécessairement interactif.

Le démonstrateur proposé est fondé pour l'essentiel sur une technique de "pattern matching" très efficace qui pourrait être appliquée aussi à d'autres domaines: l'information est codée sous forme de "spectres", et la déduction est dirigée par la ressemblance (résonance) des spectres des diverses hypothèses.

Avenir du langage

Seule une expérimentation en vraie grandeur permettra de décider du bien-fondé de cette approche, même s'il existe déjà des indices favorables, notamment:

- . simulation à la main ;
- . fondements semblables à ceux du langage LCF de Milner, dont le succès est indiscutable ;
- . contournement de difficultés classiques de l'Intelligence Artificielle.

La principale difficulté d'implémentation sera de déterminer l'état initial du langage (types, opérateurs, axiomes et théorèmes initiaux) permettant aux utilisateurs de s'en servir facilement.

Enfin, l'intérêt et l'étendue des applications du langage justifient largement le coût estimé (relativement élevé) de son implémentation.

SOMMAIRE

7 CHAPITRE 0: INTRODUCTION: PREUVES DE PROGRAMMES ET "INTELLIGENCE" ARTIFICIELLE

I - VALIDATION DES PROGRAMMES

1. Intérêt des preuves par rapport aux tests de programmes
Preuves par récurrence
- 11 2. Difficultés des preuves de programmes: un diagnostic
1er défaut: inaptitude à utiliser des principes de récurrence
2ème défaut: communication avec un démonstrateur interactif
3ème défaut: principes de démonstration.
- 13 3. Remède: construction et preuve par transformations de spécifications
3.1. Spécification de base S
3.2. Spécifications intermédiaires Si et finale P
3.3. Transformations
3.4. Comparaison avec d'autres méthodes de programmation
α/ transformations de programmes
β/ programmation en logique du 1er ordre
γ/ situation par rapport à d'autres techniques de preuves de programmes
- 19 II - "INTELLIGENCE" ARTIFICIELLE

21 CHAPITRE I - TYPES

0 - INTRODUCTION

22 I - NOTION DE TYPE

1. Types et domaines
2. Types et classes d'expressions
3. Relations entre types
4. Types et opérateurs

25 II - DEFINITIONS DES TYPES

1. Types de base
2. Constructeurs acceptables
3. Exemples de constructeurs

- 26 a/ produit cartésien
 b/ application
 c/ itération
 d/ partie
 e/ polymorphisme
 e'/ polymorphisme dual
 f/ restriction
 g/ union
 h/ intersection
 i/ différence
 j/ nomination (récursive ou non)
 k/ qualification
- 32 4. Redondance des constructions, égalités de types
- 35 5. Tableau récapitulatif
- 37 III- CONNAISSANCE DES TYPES D'UNE EXPRESSION
 Détermination par contexte du type [minimum] d'une expression
 Cas général
- 38 CONCLUSION
- 40 CHAPITRE II - SYNTAXE
- 0 - INTRODUCTION
- I - ETAT DU LANGAGE
1. L'ensemble des types
2. Ensemble des "règles opératoires"
3. Ensemble des identificateurs typés
- 44 II - ANALYSE SYNTAXIQUE
1. La fonction de l'analyseur syntaxique
 Arbre interprété
- 46 2. Phase A: Analyse structurelle
 Arbre non interprété
 Résultats de l'analyse structurelle
- 51 3. Phase B: Interprétation
- a/ caractérisation de l'ensemble des interprétations valides
 b/ algorithme d'interprétation complète

- 55 c/ exemple
- 59 d/ gestion des identifiieurs
 - α / règles d'élision des quantifieurs
 - β / variables globales: inférence de types
 - γ / variables locales: marquage des arbres
- 62 4. Résultats de l'analyse syntaxique
- 65 III - EXTENSIONS, INFERENCE, APPRENTISSAGE
 - 1. Localisation des difficultés syntaxiques
 - a/ absence d'arbre syntaxique
 - b/ ambiguïté fondamentale
 - c/ absence d'interprétation
 - 70 2. Questions-réponses: principe de discrimination
 - a/ ambiguïté fondamentale
 - b/ autres difficultés
 - 72 3. Mécanismes de l'extension.
 - 1. extension des types
 - introduction d'un type de base
 - introduction d'un ordre sur les types de base
 - constructeurs "interactifs"
 - 75 2. extension des règles opératoires
 - 1. résolution de l'absence d'arbre syntaxique
 - 2. résolution de l'ambiguïté fondamentale
 - 3. résolution de l'absence d'interprétation
 - a/ abus de langage et figures de rhétorique
 - b/ règle manquante
 - 4. résolution des difficultés de déduction
 - 5. conversion de séparateur en identifieur de constante
 - 80 3. extension des identifiieurs typés
 - a/ introduction d'identifieur
 - b/ modification par déclaration
 - c/ modification par inférence syntaxique
 - d/ modification par inférence déductive
 - 82 4. Conclusion sur l'extension.

84 IV - PERFORMANCES DE L'ANALYSE SYNTAXIQUE

1. Conséquences de l'extensibilité
2. Conséquences de la tolérance aux ambiguïtés
3. Améliorations possibles

88 CHAPITRE III - SEMANTIQUE DEDUCTIVE

89 I - NOTION DE SEMANTIQUE

1. Rappel
2. Hétéro-sémantique
3. Auto-sémantique

92 II - PRINCIPE DE LA SEMANTIQUE DEDUCTIVE

1. Organisation de la base de spécifications
 - a/ éléments de la base
 - b/ opérations sur les bases
 - c/ représentation de la base

- 95 2. Règle de déduction
- a/ convention de notation et signification
 - b/ substitutions
 - c/ règles d'inférence
 - d/ intervention de l'égalité
 - e/ existence d'une substitution entre deux phrases
 - f/ effet de l'ambiguïté d'interprétation
 - α / déductions abusives
 - β / déductions impossibles
 - γ / apprentissage

- 104 3. Utilisation par un démonstrateur de théorèmes
- a/ mode exploratoire
 - b/ mode inverse
 - c/ réduction du "domaine d'incertitude", preuve par parties
 - d/ recherche de substitutions

- 110 4. Exemple de déduction

117

III - SYSTEME DE DEDUCTION

1. Démonstrateurs de théorèmes: critères de qualité

- a/ critères qualitatifs
 - α / facilité d'accès et d'utilisation
 - β / précision de la réponse
 - γ / résistance au "bruit"
 - δ / portée
- b/ critères quantitatifs
 - α / occupation mémoire
 - β / rapidité
 - γ / autonomie
 - δ / fiabilité
- c/ compromis entre ces critères
 - α / portée ou rapidité
 - β / rapidité ou fiabilité

122

2. Interaction

124

3. Amélioration de l'autonomie (I)

- a/ rappel sur les substitutions
- b/ classe réduite de substitutions
- c/ reconnaissance d'arbres
- d/ spectre exact d'un arbre
- e/ spectre approché d'un arbre
- f/ ϵ -démonstrateur
- g/ "pattern matching" amélioré par les codes homomorphes
- h/ fonctions de codage dans un corps Z/N
 - α / fonctions linéaires non dégénérées
 - β / fonctions linéaires dépendant d'un seul paramètre
 - γ / les "bons" nombres premiers
 - δ / exemple de codage utilisant un "très bon" nombre premier
 - ϵ / autres fonctions dans Z/N
- [ξ / fonctions dans d'autres corps finis]

148

4. Amélioration de l'autonomie (II)

148

5. Amélioration de l'autonomie (III): heuristiques

6. Conclusion

151

CHAPITRE IV - UTILISATION - PROLONGEMENTS

I - MISE EN OEUVRE DU LANGAGE

1. Problèmes de performance

- a/ effet de maquette
- b/ état initial et divergence
- c/ autonomie du démonstrateur

154

2. Coût de l'implémentation

154 II - APPLICATIONS

1. Programmation assistée par ordinateur
2. Conception assistée par ordinateur
3. Relations avec l'Intelligence artificielle
4. Calcul par résonance

157 III - CONCLUSION

159 ANNEXE I

Estimation de Fiabilité:

Système sans vieillissement n'ayant jamais subi de panne.

161 ANNEXE II

Complexité des démonstrateurs de théorèmes

164 ANNEXE III

Probabilité de collision

165 ANNEXE IV

Exemple d'application à la programmation

174 REFERENCES