IRIA
laboria

Institut de Recherche
d'Informatique
et d'Automatique

de Voluceau
Rocquencourt
78150 - Le Chesnay
Tél. 954.90.20

laboratoire de recherche
en informatique
et automatique

DELTA.
Système de DEscription de Langages et de Traducteurs par Attributs

Laurent Blaizot

Rapport de Recherche n° 20

juin 1973

DELTA

Système de DEscription de Langages et de Traducteurs par Attributs

Laurent Blaizot
(présenté par Bernard Lorho)

Résumé :

Nous développons ici quelques idées sur l'utilisation dans un système complet de métacompilation, d'une méthode de description sémantique due à Knuth.
Cette méthode de définition descriptive de la sémantique d'un langage devrait permettre d'assouplir énormément l'utilisation d'un tel système. Le projet DELTA (DEscription des Langages et de leurs Traducteurs par Attributs) que nous nous proposons de réaliser devrait d'une part mettre l'accent sur cette facilité d'utilisation et d'autre part accorder une place importante aux techniques d'optimisation, qu'elles soient décrites par l'utilisateur ou fournies par le système. La technique de définition par attributs de Knuth s'applique en effet parfaitement à la description par l'utilisateur des techniques d'optimisation qu'il souhaite éventuellement incorporer à son processeur. Enfin, le problème de la portabilité des traducteurs produits est sans aucun doute d'une grande importance et donnera lieu à une étude approfondie.

Abstract :

Some ideas are developped about the application to a complete metacompilation system of a semantics description method first exposed by Knuth. This declarative method of language's semantics definition should allow one to use such a system very easily.
Our purpose is to implement a system, DELTA (DEfinition of Languages and Translators by Attributes), in which a great importance will be attached to optimization techniques described by the user, as well as those supplied by the system, and to the problem of portability, so as to allow most people to use it conveniently.

D E L T A

Système de Description de Langages et de Traducteurs par attributs

---

étude préliminaire

Ce papier concerne le démarrage au LABORIA d'un projet d'étude d'un TWS *. Nous ne parlerons ici que des aspects sémantiques. Les problèmes syntaxiques feront l'objet d'un autre papier. [14]

1. "Translator-oriented" ou "Interpreter-oriented" ?

Le problème de la conception de systèmes d'écriture de traducteurs a été abordé dès 1964. Les premières réalisations étaient basées sur la notion habituelle de compilateur, c'est-à-dire que l'utilisateur décrivait son langage en termes relatifs à un autre langage supposé "connu", en d'autres termes compilable ou assemblable. Ceci a pu faire dire depuis [12] que cette solution n'était pas élégante car elle ramenait le problème posé à un problème déjà connu, et ne s'intéressait pas à la véritable analyse du problème, qui aurait dû être faite relativement à des bases universelles.
Une seconde approche s'est donc dessinée. La base universelle choisie était l'ordinateur, ou plutôt la configuration de sa mémoire.
L'utilisateur, pour définir un langage, décrit l'effet de chaque instruction de son langage sur la mémoire de la machine, c'est-à-dire sur les données de son programme, en fonction de l'état antérieur de ces données. Cette méthode permet évidemment une définition beaucoup plus élégante, car la base de définition est la même pour tous.

* TWS : Translator Writing System : système d'écriture de traducteurs

Le formalisme le plus élaboré dans ce domaine est l'oeuvre du groupe IBM de Vienne et est appelé VDL (Vienna Definition Language).
L'inconvénient de ce mode de description apparait dès que l'on se pose le problème de réaliser un T.W.S. qui utilise ce formalisme.
En effet, de par la nature même de la description, on ne peut en déduire qu'un interpréteur. De là vient le qualificatif de "interpreter-oriented" appliqué à cette approche. La première approche, dont nous avons déjà parlé, est qualifiée de "compiler-oriented", bien que "translator-oriented" paraisse plus approprié.
Il est bien certain qu'il existe deux catégories d'utilisateurs intéressés par la formalisation de la sémantique : d'une part les linguistes et les logiciens, qui s'intéressent à la formalisation elle-même, sans préoccupation d'implantation ; ces utilisateurs souhaitent appliquer cette formalisation à l'étude de schémas de programmes et à la démonstration automatique de théorèmes. Les travaux de Lucas et Walk [3,2], montrent que l'approche de Vienne convient parfaitement à cette utilisation.
Mais il y a d'autre part un ensemble d'utilisateurs spécialistes d'écriture de software, qui ont une grande habitude des traducteurs en général, et qui souhaitent être déchargés de toute une partie fastidieuse de leur travail, à savoir la mise en place du squelette d'un traducteur, analyseur syntaxique, analyseur lexicographique, diverses mises en tables etc...
Ces gens-là sont plus intéressés par l'efficacité et la commodité d'utilisation de l'outil qu'on leur propose que par l'esthétique de l'approche.
En fait, ramener un problème posé à un problème connu est pour eux une habitude, et l'approche "translator-oriented" leur est plus familière que l'autre.
Certains auteurs [13] n'hésitent pas à dire que devant un problème précis d'application à résoudre, il est plus rentable d'utiliser un langage spécialisé dont on écrit le compilateur, que de se servir d'une petite partie d'un langage de haut niveau existant. Cela est d'ailleurs particulièrement vrai pour les petits ordinateurs, sur lesquels on ne dispose pas toujours du compilateur d'un langage de niveau suffisant pour couvrir l'application en question.

Cela montre qu'il serait très intéressant de disposer d'un TWS suffisamment maniable et efficace pour inciter les informaticiens à l'utiliser. C'est à quoi nous avons décidé de nous intéresser. Un autre aspect très intéressant serait évidemment la portabilité de ce TWS d'une machine sur une autre. Nous en reparlerons plus loin.

2. Etude de l'approche "translator-oriented".

Nous avons étudié particulièrement :

- les système META [1]
- le Compiler - Compiler [4]
- le FSL de Feldman [5]
- le "Syntax directed Compiler" de IRONS [6]
- la formalisation de KNUTH [7,8]

Cette dernière, la plus élégante, est inspirée du travail d'IRONS. Elle consiste à exprimer la sémantique d'un non-terminal de la grammaire en fonction de celle des "ascendants" et "descendants" immédiats de ce non-terminal.
L'examen de ces diverses réalisations nous a permis de dégager les points délicats sur lesquels les solutions divergent.

2.1 Le méta-langage sémantique

Un méta-langage adéquat devrait être "universel", en ce sens que son rôle est de permettre de décrire n'importe quel concept. Le premier auquel on pense est évidemment le langage-machine, puisqu'il utilise une base commune à tous les utilisateurs d'une machine, l'ensemble des commandes câblées.
Malheureusement, l'utilisation du langage machine a deux inconvénients :

- tout d'abord, un programme écrit en LM n'est pas facilement compréhensible s'il n'est pas abondamment documenté, ce qui est particulièrement regrettable dans notre cas, où la description d'un langage doit pouvoir se communiquer d'un utilisateur à un autre,
- d'autre part, le fait que le TWS accepte une description dépendant étroitement de la machine rend complexe sa transférabilité, et n'atteint donc que partiellement les objectifs visés.

Le LM n'est donc pas adapté. Le problème est de déterminer à quel niveau de complexité se situe le langage sémantique idéal.
Des réalisations comme FSL ou META 5 utilisent une formulation proche du FORTRAN, avec en plus de nombreuses primitives de mise en table, génération et empilement. Malheureusement, ces primitives ne sont pas à notre connaissance modifiables par l'utilisateur, ce qui rend le système trop rigide.
En outre, on ne trouve pas dans ces systèmes, de primitives de description de données suffisamment évoluées, ce qui limite considérablement l'utilisateur quant à ses structures de données. D'un autre côté, une notation comme celle de Knuth se prête très facilement à l'adjonction de primitives et a l'avantage d'être très élégante. Nous pensons donc que le mieux serait d'adopter cette notation et d'y ajouter toute une bibliothèque de primitives dont le choix reste à faire.
Les primitives devraient permettre :

- de fournir à l'utilisateur des outils commodes de mise en table, génération, empilement etc...
- de mettre à sa disposition une base de données suffisante pour décrire toute structure
- enfin, de lui fournir également une base de description du contrôle de son programme (rupture de séquence, appels de procédure, tasking etc...).

C'est sur ces deux derniers points que l'accent devra être mis, car les réalisations existantes les ont toutes laissés dans l'ombre. Un des points sur lesquels cette réalisation doit se distinguer des autres sera également la clarté du formalisme de description sémantique. La notation de Knuth doit permettre d'atteindre cet objectif.

2.2 Le résultat de la phase d'analyse

C'est un autre aspect du problème, tout aussi délicat que le premier. Le but premier d'un T.W.S. est que l'utilisateur puisse en faire ce qu'il désire : un interpréteur, un compilateur ou un simple traducteur d'un langage dans un autre (compilation par couches, passage d'une implémentation à une autre etc...). Un système comme FSL ne permet que de réaliser un compilateur, puisque les seules sorties possibles sont des sorties par

génération de binaire. Par contre, les systèmes META ont été définis en vue de la traduction d'un langage dans un autre sous forme de chaînes de caractères. Mais ils sont mal adaptés à l'écriture de compilateurs ou d'interpréteur. Ces restrictions sont malcommodes. Un T.W.S. adéquat doit permettre de réaliser n'importe quel processeur.
On voit que selon que l'utilisateur aura décrit un traducteur, un compilateur ou un interpréteur, le résultat de l'exécution de ses routines sémantiques sera :

- un autre programme à passer dans un traducteur existant
- un binaire à charger et exécuter
- un ensemble de résultats.

Un point important est la façon dont sont exécutées ces routines sémantiques.
Nous pensons que, plutôt que de les exécuter une à une au fur et à mesure de la reconnaissance de l'entité syntaxique associée, comme cela est fait par exemple dans le Compiler-Compiler, il est préférable que l'analyseur syntaxique produise un véritable programme d'évaluation sémantique, qui est ensuite fourni au compilateur du méta-langage sémantique.
Cela doit améliorer considérablement l'efficacité du compilateur produit.
Un gros avantage est que, disposant d'un véritable programme, on peut éventuellement lui appliquer une phase d'optimisation.
Nous verrons plus loin que la méthode de Knuth se prête à une autre optimisation, moyennant une légère modification de l'algorithme de construction du programme d'évaluation sémantique.

3. Etude de la formalisation de Knuth

Dans un premier temps, il est nécessaire de réaliser un outil simple dont nous puissions nous servir sur quelques exemples de façon à évaluer les besoins, notamment en primitives. Nous avons donc choisi un méta-langage sémantique dont nous disposions du compilateur : le PL/1. Ce langage a l'avantage d'être assez vaste pour couvrir de nombreuses applications.
Nous utilisons ce méta-langage sémantique dans le cadre de la formulation de Knuth. D'autre part, les outils syntaxiques dont nous disposons sont :