Spécification formelle et génération automatique de programmes : le système Descartes

Jean-Luc Dormoy, Bruno Ginoux, Jean-YvesLucas, Laurent Pierre[1], Claudia Jimenez-Dominguez[2]
Electricité de France
Direction des Etudes et Recherches
1, avenue du Général de Gaulle
92141 CLAMART Cedex
Rene.Descartes@der.edfgdf.fr

Résumé

Descartes est un système développé à la Direction des Etudes et Recherches d'EDF, qui associe un langage formel de spécification et un générateur automatique de programmes.
Le langage Descartes est un langage de type mathématique qui repose sur la théorie des ensembles, la logique des prédicats du premier ordre et la théorie des fonctions récursives. Il permet de définir et de manipuler les objets mathématiques tels que les ensembles, les vecteurs ou les fonctions en utilisant les quantificateurs et les opérateurs mathématiques habituels.
Le système Descartes permet de générer des programmes écrits dans des langages de programmation procéduraux classiques à partir de spécifications de programmes écrites dans le langage Descartes. La première version du système est opérationnelle depuis le mois de juin 1997. Elle représente environ un million de lignes de code C, pour la plupart (80%) générées automatiquement. Une version du système écrite dans son propre langage est actuellement disponible et doit permettre, à terme, de « bootstrapper » le système.
Descartes ne supprime pas le cycle en V du logiciel mais le situe à un niveau d'abstraction supérieur. Ceci a pour effet de rendre les programmes plus faciles à écrire, plus courts, plus clairs, plus fiables et plus évolutifs. Cela permet également d'attaquer des problèmes qui restaient hors de portée des langages classiques du fait de leur trop faible niveau d'abstraction. Mais l'avantage déterminant est sans nul doute la diminution drastique des coûts de développement et de maintenance.
Une des applications générées grâce à Descartes concerne un programme d'aide à la conduite des centrales thermiques classiques. Le programme généré compte 43000 lignes de C pour environ 2000 lignes de spécification. Outre cette application, Descartes a permis de générer automatiquement un programme de gestion de l'arrêt d'urgence dans une centrale nucléaire (projet ESPRIT DARTS), et est aussi appliqué à la validation et la cohérence de données, au transfert de données entre mondes informatiques hétérogènes, et à l'optimisation d'arbres de défaillance et de simulation.

1. Introduction

La programmation a toujours constitué le principal obstacle à l'utilisation de l'ordinateur. En effet, les programmes ont toujours été longs à écrire, difficiles à mettre au point, complexes à valider et « impossibles » à maintenir ou faire évoluer. La principale raison de cet état de fait réside

[1] DER-EDF, IMA/TIEM/GLIP, 1 avenue du Général de Gaulle 92141 CLAMART CEDEX
[2] Société INFORAMA

essentiellement dans le bas niveau d'abstraction des langages qui restent la plupart du temps très proches de la machine, rendant l'activité de programmation très technique.

Automatiser le processus de programmation a donc constitué, depuis que l'informatique existe, une préoccupation permanente des informaticiens. Si l'on résume, peut-être un peu brièvement, l'histoire de cette automatisation, on est amené à identifier deux grandes catégories d'approches. La première que l'on pourrait qualifier d'approche « par le bas » s'attache à remonter le niveau d'abstraction des langages de programmation afin de simplifier le travail du programmeur en le déchargeant des considérations techniques. La deuxième, que l'on qualifierait d'approche « par le haut » tente de partir d'une spécification formelle complètement indépendante de toute implémentation, par exemple une spécification exprimée dans un langage de type mathématique, et d'obtenir automatiquement le programme correspondant grâce à un générateur de code.

Nous présentons dans cet article le système DESCARTES développé à la Direction des Etudes et Recherches d'Electricité de France, qui permet de passer automatiquement d'une spécification formelle exprimée dans le langage mathématique au programme procédural correspondant. Autrement dit, Descartes est basé sur la seconde des deux approches mentionnées ci-dessus, rompant ainsi nettement avec les langages de programmation habituels.

Nous présentons brièvement le système DESCARTES puis nous donnons, afin de fixer les idées, un exemple de spécification DESCARTES que nous commentons. Nous précisons ensuite les principales caractéristiques du langage et du système. Nous concluons sur l'avenir de la programmation automatique et notamment sur les potentialités et les bénéfices très importants qui pourraient être retirés d'un couplage avec les autres méthodes formelles par exemple, les méthodes liées à la preuve de programmes.

1. Qu'est-ce que DESCARTES ?

DESCARTES est à la fois un langage de spécification formelle et un système de génération automatique de code.

Le langage DESCARTES est un langage de type mathématique qui repose sur la théorie des ensembles, la logique des prédicats d'ordre 1 et la théorie des fonctions récursives.
Il permet de définir et manipuler les objets mathématiques usuels tels que les ensembles, les vecteurs ou les fonctions en utilisant les quantificateurs et les opérateurs mathématiques habituels.

La spécification DESCARTES d'un programme est composée de trois parties:

- le schéma conceptuel des données du domaine de l'application. Ce schéma est exprimé dans un modèle de type Entité-Association. Il est commun à toutes les applications concernant ce domaine,

- un ensemble de fonctions définies par l'utilisateur. Chaque fonction exprime une connaissance sur le domaine d'application,

- un ordre « CALCULER » qui spécifie le but du programme, c'est à dire en fait la quantité que le programme doit calculer. Cet ordre « CALCULER » constitue à lui seul finalement la véritable spécification du programme. En effet, le schéma conceptuel des données ainsi que les définitions de fonctions sont exprimés indépendamment de tout programme particulier.

Le système DESCARTES est un générateur automatique de programmes. Il génère des programmes écrits dans des langages classiques à partir de spécifications de programmes écrites dans le langage DESCARTES. La première version du générateur compte environ un million de lignes de code C et Pascal, la plupart étant elles-mêmes générées automatiquement.

2 Exemple de spécification DESCARTES

Soit un ensemble E donné. On veut écrire un programme permettant de calculer un sous-ensemble particulier des éléments de E, à savoir:

$$\{ x \mid x \in E \text{ et } \forall y (y \in f(E,x) \Rightarrow \exists z (z \in E \text{ et } Q(x,y,z))) \}$$

L'ensemble E peut être par exemple un ensemble d'assemblages (barres d'uranium). La fonction f peut renvoyer l'ensemble des types de grappes potentiels pour un assemblage donné et le prédicat Q peut exprimer le fait qu'un assemblage « z » est ou non en mesure d'échanger sa grappe avec celle de l'assemblage « x » compte tenu de l'ensemble des types de grappes potentiels de « x ». On obtient alors un programme utilisable lors du déchargement-rechargement des coeurs de centrales nucléaires.

De même, on peut dire que l'ensemble E est un ensemble de compteurs EDF, de pompes, de clients, de règles d'inférences,....

Pour simplifier, supposons que E est un ensemble d'entiers. Précisons maintenant la fonction « f » et le prédicat « Q » pour obtenir la spécification complète suivante :

```
SOIT f(A,a) : P(ENTIER) , ENTIER -> P(ENTIER) :
   { b | b ∈ A et b < a }

SOIT Q(a,b,c) : ENTIER, ENTIER, ENTIER -> BOOLEEN :
    a = b + c

SOIT h(A) : P(ENTIER) -> P(ENTIER) :
   { x | x ∈ A et ∀ y (y ∈ f(A,x) => ∃ z (z ∈ A et Q(x,y,z))) }

CALCULER h(E)
```

On met en évidence sur ce simple petit exemple de nombreux avantages du langage DESCARTES. En effet, en DESCARTES, la spécification du programme se réduit à la seule définition mathématique de l'ensemble en compréhension. Il n'y a en particulier pas de choix de structures de contrôle et pas de choix de structures de données. L'utilisateur est déchargé de toutes ces considérations de type informatique puisque l'algorithme et le programme correspondants sont générés automatiquement par le système.

Par ailleurs, on constate sur cet exemple que la spécification est extrêmement concise, à savoir, 7 lignes, qu'elle est lisible, plutôt plus que le programme C équivalent écrit à la main, qui fait une petite centaine de lignes, et ce, y compris par des gens ne connaissant pas le langage DESCARTES (ce qui n'est pas le cas pour C) et il paraît raisonnable de supposer que cette spécification est plus

facile à valider formellement et à maintenir que son équivalent C du fait qu'elle est à un niveau d'abstraction beaucoup plus élevé.

En effet, en C, et dans tout autre langage procédural classique, il faut au contraire écrire explicitement un algorithme permettant de calculer l'ensemble et coder cet algorithme dans la syntaxe du langage cible. Pour écrire l'algorithme, il faut, entre autres:

- au niveau structures de contrôle: écrire explicitement trois boucles imbriquées permettant de gérer le parcours de l'ensemble E, calculer pour chaque élément x l'ensemble f(E,x) correspondant, parcourir chacun de ces ensembles f(E,x) pour vérifier si pour chaque élément de ces ensembles il existe un z tel que le prédicat Q est vérifié, ce qui nécessite bien une troisième boucle sur l'ensemble des z potentiels.

- au niveau structures de données: respecter explicitement dans l'algorithme les structures informatiques des ensembles, c'est à dire entre autres écrire les instructions relatives à leurs parcours en fonction de leurs représentations internes (le parcours d'une liste chainée est différent du parcours d'un arbre binaire équilibré, d'un tas, d'un tableau,...).

Sans rentrer dans les détails, montrons les toutes premières transformations que DESCARTES va, avant toute chose, effectuer sur cet exemple particulier:

Dépliage de la fonction « h » dans l'expression à calculer « h(E) » :
On a: $h(E) = \{ x \mid x \in E \text{ et } \forall y (y \in f(E,x) \Rightarrow \exists z (z \in E \text{ et } Q(x,y,z))) \}$

Dépliage de la fonction « f » et du prédicat « Q » :
$h(E) = \{ x \mid x \in E \text{ et } \forall y (y \in \{ b \mid b \in E \text{ et } b < x \} \Rightarrow \exists z (z \in E \text{ et } x = y + z)) \}$

Transformations et simplifications :

1/ $y \in \{ b \mid b \in E \text{ et } b < x \}$ est réécrit : $y \in E \text{ et } y < x$

En effet, si l'élément « y » appartient à l'ensemble défini en compréhension, il doit obligatoirement vérifier la condition d'appartenance

2/ $\exists z (z \in E \text{ et } x = y + z)$ est réécrit : $x - y \in E$

d'ou finalement: $h(E) = \{ x \mid x \in E \text{ et } \forall y (y \in E \text{ et } y < x \Rightarrow x - y \in E) \}$

On constate donc déjà deux améliorations notables. D'une part la fonction f(E,x) n'est pas calculée, du moins pas telle qu'elle est écrite dans la spécification, ce qui a pour effet de ne pas avoir, à chaque itération, à stocker en mémoire et à parcourir l'ensemble correspondant. D'autre part, la troisième boucle sur les ensembles de « z potentiels » n'est plus nécessaire. L'algorithme proposé par DESCARTES ne contient donc que deux boucles et ne crée aucun ensemble intermédiaire.

Nul doute qu'un programmeur humain peut faire au moins aussi bien (dans le cas présent il se trouve qu'il est difficile de faire mieux), mais là n'est pas notre propos. Le programmeur ne se pose pas forcément toujours de questions particulières quant à l'optimisation. Il peut également écrire volontairement le programme d'une façon non optimale dans le but de pouvoir réutiliser des sous-programmes déjà présents en bibliothèque ou de faciliter la future phase de maintenance. En DESCARTES, ces questions ne se posent pas. La spécification est écrite indépendamment de ces considérations, le programme généré étant lui raisonnablement optimisé.

3 Principales caractéristiques du langage et du système DESCARTES

3.1 Le Langage DESCARTES

Récapitulons en quelque sorte les enseignements de l'exemple précédent. DESCARTES est un langage de type mathématique. Cette caractéristique première du langage est à l'origine de ses nombreux points forts.

On peut citer tout d'abord, la déclarativité qui permet d'exprimer un problème à un haut niveau d'abstraction, c'est à dire indépendamment de la machine. En d'autres termes, pour écrire une spécification de programme, il suffit de formuler le « Quoi », c'est à dire le but à atteindre et non le « Comment », c'est à dire la mise en oeuvre informatique permettant d'atteindre le but. Cette description du « Quoi » se fait en utilisant le langage mathématique.

On peut citer également la concision, la lisibilité et la facilité d'apprentissage. Le langage mathématique, au moins en ce qui concerne les notions de base qui sont celles utilisées dans DESCARTES, est en général bien connu. D'une certaine façon, DESCARTES n'est pas un nouveau langage. Il est donc rapidement facile à lire et pour les mêmes raisons il est facile à apprendre.

Mais, l'une des principales qualités du langage DESCARTES est la réutilisabilité. Ecrire une spécification DESCARTES consiste à donner un ensemble de définitions de fonctions, chaque fonction exprimant une connaissance, un concept. Chacune de ces fonctions étant écrite indépendamment de toute utilisation particulière, c'est à dire de tout programme particulier, devient éminemment réutilisable. La réutilisabilité est en effet bien meilleure que dans les langages classiques dans lesquels les connaissances sont toujours exprimées en fonction d'un besoin particulier mais aussi inextricablement mêlées avec une mise en oeuvre particulière. En DESCARTES, le mode d'utilisation d'une connaissance donnée est généré automatiquement par le système lorsqu'a été spécifié, par l'intermédiaire de l'ordre « CALCULER », quel était le résultat désiré. La réutilisabilité est ici celle des spécifications et non plus celle des programmes.

Considérons par exemple la fonction DESCARTES très simple définie comme suit:

```
SOIT Suivante(c): COULEUR -> COULEUR:
  case
    : c = Vert  -> Orange
    : c = Orange  -> Rouge
    : c = Rouge -> Vert
  fin
```

Cette définition n'est pas un programme. Elle peut en revanche être utilisée pour générer plusieurs programmes différents. On peut vouloir calculer par exemple la couleur suivante d'une couleur c0 donnée en entrée :

```
  CALCULER Suivante(c0)
```

ou calculer la couleur c1 dont la couleur suivante est la couleur c0:

```
  CALCULER (c1 | Suivante(c1) = c0)
```

ou encore l'ensemble des couleurs suivantes d'un ensemble de couleurs C0 donné:

```
CALCULER {Suivante(c0) | c0 ∈ C0}
```

ou encore,....

On voit bien que dans le deuxième cas par exemple, c'est en fait la couleur précédente de la couleur c0 qui est demandée. Le générateur va être conduit, par des transformations successives à inverser la fonction pour pouvoir écrire un programme donnant le résultat demandé. On peut noter que le générateur ne comprend pas qu'il faut inverser la fonction et qu'il ne cherche donc pas à le faire. C'est la combinaison d'un certain nombre de transformations élémentaires et générales qui va le conduire dans ce cas précis à inverser la fonction.

Une même connaissance (une même définition de fonction) est donc la plupart du temps utilisée différemment selon le programme à construire. Ce n'est pas du tout le cas dans les langages de programmation (y compris les langages fonctionnels tels que Lisp ou CamL) dans lesquels les fonctions sont « figées » et ne peuvent servir que pour l'utilisation (unique) pour laquelle elles ont été définies. Il aurait donc fallu dans l'exemple précédent des couleurs écrire autant de sous-programmes différents que d'ordre « CALCULER » alors qu'en fait tous ces sous-programmes reposent sur une même et unique connaissance.

C'est justement cette connaissance là qui constitue la totalité de la spécification DESCARTES. En DESCARTES, les définitions de fonctions sont donc bien des connaissances et non des sous-programmes au sens habituel du terme.

DESCARTES n'est pas un langage de programmation car ce qui est écrit n'est pas exécuté. **La spécification n'est pas le programme, c'est un ensemble de connaissances à partir desquelles le générateur construit un programme spécifique permettant de calculer la quantité spécifiée dans la partie « CALCULER » de la spécification**. Dans le programme généré, les fonctions définies par l'utilisateur dans la spécification sont adaptées chaque fois au contexte particulier dans lequel elles sont utilisées, c'est à dire la plupart du temps radicalement transformées, parfois utilisées « à l'envers » (inversées), ou même totalement absentes. La structure du programme généré est complètement imprévisible au vu de la spécification à moins bien sûr de suivre la même démarche que le générateur.

3.2 Le système DESCARTES

Le système DESCARTES est un système général, c'est-à-dire non dédié à un domaine d'application particulier. Il est également complètement automatique, ce qui signifie qu'il ne requiert pas l'intervention d'un utilisateur. Il possède des connaissances qui lui permettent de prendre toutes les décisions seul. Dans la grande majorité des cas, ses connaissances sont suffisantes pour assurer la génération d'un code raisonnablement efficace.

DESCARTES repose sur une méthodologie transformationnelle. Le système applique en fait une longue séquence de transformations, chaque transformation ayant pour effet de « raffiner » l'état courant du programme en formation, c'est à dire de le rendre de plus en plus concret jusqu'à l'obtention du programme final exprimé dans un langage cible. Par ailleurs, nous avons structuré cette séquence de transformations en figeant des niveaux d'abstraction intermédiaires du programme en formation, les trois principaux niveaux étant définis par les langages intermédiaires (internes donc transparents pour l'utilisateur) COGITO, ERGO et SUM. Détailler ces transformations serait beaucoup trop long, et n'est pas le sujet de cet article. Mais mentionnons tout de même que certaines transformations connues des spécialistes de la génération de code, comme

les simplifications locales et contextuelles, la dérécursivation, la résolution de contraintes ou la différentiation formelle, …, sont présentes sous une certaine forme dans le système.

Le système DESCARTES permet de générer des programmes dans des langages cibles a priori quelconques. En effet, la chaîne de transformations est conçue de manière à rester indépendante de tout langage cible le plus longtemps possible. La quasi totalité du système SUM a pour but d'obtenir un programme exprimé dans un langage de programmation procédural générique suffisamment pauvre de manière à n'utiliser que les structures de contrôle et structures de données communes à la plupart des langages habituels.

C'est le tout dernier maillon du système SUM qui se charge de traduire ce programme exprimé dans ce langage générique en un programme écrit dans un langage cible particulier. Cette traduction est quasi triviale puisqu'elle n'est que syntaxique.

4 Les premiers résultats

Nous avons actuellement (octobre 1997) une première version du générateur DESCARTES qui dépasse le million de lignes de code C et Pascal. Cette version a elle même été obtenue par génération automatique à partir de 80 000 lignes de code écrites en SHAL. Le système SHAL est un ensemble de petits générateurs de code qui a été développé spécifiquement pour permettre la mise en oeuvre de la première version opérationnelle de DESCARTES.

Ce prototype fonctionne et a fourni ses premiers programmes. Nous avons notamment généré automatiquement le programme DARTS, de gestion de l'arrêt d'urgence dans une centrale nucléaire. Ce programme servait de support à un projet ESPRIT pour comparer différentes méthodes, langages et outils de spécification et de développement de logiciels. DESCARTES a permis de générer automatiquement à partir de 600 lignes de spécification formelle un programme PASCAL de 12000 lignes qui a passé avec succès les jeux d'essais officiels du projet.

Nous avons également spécifié et généré le programme SACSO qui permet de planifier en temps réel les opérations de conduite d'une centrale thermique, de suivre le comportement des processus physiques et de détecter ou d'anticiper des incidents sur la tranche. La spécification initiale compte 2000 lignes, et le programme généré 43000 lignes de C.

Nous travaillons également sur la spécification d'un programme de vérification de la cohérence des Diagrammes Fonctionnels Logiques et d'un programme de manipulation d'arbres de défaillance.

Mais notre application la plus importante reste la spécification du système DESCARTES lui-même qui représente actuellement environ 50 000 lignes écrites dans le langage DESCARTES et qui doit nous permettre, à terme, de « bootstrapper » le système.

Conclusion

Les résultats obtenus avec DESCARTES montrent que la programmation automatique ne pose plus aujourd'hui de problème insurmontable en ce qui concerne la faisabilité technique, au moins en ce qui concerne la classe des problèmes considérés, à savoir, les programmes éventuellement « gros » mais conceptuellement simples, c'est à dire les programmes dont la complexité provient essentiellement de la taille. Sur cette classe de problèmes, qui par ailleurs est extrêmement vaste, l'efficacité des programmes générés est très satisfaisante.

Il semble bien qu'il faille désormais rompre avec l'idée que la programmation automatique n'est envisageable que dans certains domaines très spécifiques. Nous pensons même que la programmation automatique trouvera dans un proche avenir un champ d'application étendu dans l'industrie. Car si comme toute innovation technologique elle nécessite évidemment un investissement, elle est sans nul doute un terme essentiel de réponse à la « crise du logiciel » que nous vivons actuellement. En particulier, un système comme Descartes peut permettre d'élever le niveau et d'étendre le champ d'application des outils spécifiques existants.

DESCARTES concerne potentiellement tous les acteurs qui interviennent dans le cycle de vie du logiciel. En gros, toutes les personnes qui conçoivent, spécifient, développent, valident, maintiennent sont susceptibles de l'utiliser, pour peu qu'ils acceptent d'avoir confiance en la capacité du générateur à produire rapidement des programmes justes et efficaces. Le système DESCARTES doit leur permettre de faire les mêmes choses qu'avant mais mieux, c'est à dire développer, maintenir et faire évoluer les logiciels plus rapidement et de manière plus fiable. En effet, en situant le cycle en V du logiciel à un niveau d'abstraction supérieur, les programmes deviennent plus faciles à écrire, plus courts, plus clairs, plus fiables et plus évolutifs. Mais DESCARTES doit également leur permettre d'attaquer des problèmes qui restaient hors de portée des langages classiques du fait de leur trop grande complexité « pratique » par rapport au niveau des langages utilisés. Il y a en fait un seuil de complexité pratique au delà duquel il devient démesurément difficile d'écrire un programme qui de toutes façons sera impossible à maintenir.

Il reste que la programmation automatique ne résoud pas tous les problèmes et c'est pour cette raison qu'elle ne constitue qu'un des éléments de réponse à la crise du logiciel. Il serait très intéressant en effet de lier génération de programmes et méthodes de preuve. Souvent la preuve de programme va jusqu'au programme ou à un quasi-programme par une série de « raffinements ». On pourrait imaginer supprimer une grande partie de ces raffinements par la génération de code : dès que la spécification est « complète », c'est-à-dire dès qu'elle définit totalement la quantité que le programme doit calculer, on pourrait arrêter le processus - manuel - de raffinement et utiliser la « machine-outil à raffinement » qu'est le générateur de code. Cela est d'autant plus important qu'une modification du programme implique de refaire une partie des preuves - la preuve d'un programme n'est pas un processus linéaire.

Dans cette mesure, l'éventuel investissement supplémentaire lié à l'écriture de la spécification formelle et à la preuve de propriétés sur la spécification est compensé par le fait que le programme est généré automatiquement et que sa maintenance se résume à la maintenance de la spécification.

Les trois aspects - spécification formelle, preuve et synthèse de programme - sont donc complémentaires, et un outil les regroupant reste à construire. Des outils partiels utilisables existent, et commencent à trouver leur application dans l'industrie. Si leurs domaines d'application restent limités, ils ne pourront que s'étendre dans l'avenir.

Bibliographie

Abrial J-R., 1996. The B-Book. Cambridge University Press.

Backus J.W., 1978. Can programming be liberated from the Von Neumann Style ? : A functional style of programming and its algebra of programs. Aout. ACM 21, 8.

Burstall R.M. et Darlington J., 1977. A transformation system for developing recursive programs. J; ACM 24, 1. Janvier. pp 44, 67.

Feather M-S., 1987. A Survey and Classification of some Program Transformation Approaches and Techniques. Program Specification and Transformation. L.G.L.T. Meertens (Ed.). Elsevier Science Publishers B.V. North-Holland. IFIP.

Ginoux B., 1996. DESCARTES en 32 questions, 32 réponses et 32 pages. Note Technique EDF/DER n° HI29/96/036. Décembre.

Kant E. et Barstow D.R., 1981. The refinement paradigm : The interaction of coding and efficiency knowledge in program synthesis. IEEE Trans. Softw. Eng. 7, 458-471.

Kant E. et Newell A., 1983. An automatic algorithm designer : An initial implementation. In Proceedings of the National Conference on Artificial Intelligence. Août. pp 177, 181.

Manna Z. et Waldinger R., 1978. DEDALUS : the DEDuctive Algorithm Ur-synthesizer. In Proceeding of National Computer Conference. vol 47 AFIPS Press, Reston, Va. pp 683, 690.

Partsch H. 1990. Specification and Transformation of Programs. Springer Verlag. 1990

Rich C. et Waters R-C., 1986. Readings in Artificial Intelligence and Software Engineering. Morgan Kaufmann Publishers, Inc. Los Altos. Californie.

Smith D-R., 1990. KIDS: A Semiautomatic Program Development System. IEEE transactions on software engineering. Vol. 16. No. 9. Septembre.