Chapitre 6

Bilan et perspectives

Cette thèse a présenté le langage Java, un langage à objets concurrent. Nous avons successivement étudié les aspects objets, la concurrence et ses mécanismes pour en proposer une spécification formelle en utilisant le formalisme Typol. Ce formalisme fait partie du système Centaur, un environnement de développement d'environnement de programmation. Cela a le grand avantage de permettre la définition d'une sémantique exécutable sous Centaur après une simple compilation des spécifications Typol.
Cette exécutabilité nous a conduit tout naturellement à développer un environnement d'exécution particulièrement axé sur la visualisation graphique de la topologie des objets et des mécanismes de synchronisation. Un autre aspect que nous avons étudié est l'utilisation de cette formalisation comme une base formelle pour la vérification de propriétés du langage Java.

Ce chapitre constitue en quelque sorte un bilan de notre étude, et présente ensuite des perspectives multiples, liées tout d'abord à des développements de notre travail, puis s'élargissant vers des domaines connexes.

6.1 Bilan

Cette première section permet de dresser un bilan de notre étude, que ce soit au niveau de la définition d'une spécification formelle d'un large sous-ensemble du langage Java, du développement de l'environnement d'exécution associé, ou de la vérification de propriétés du langage.

6.1.1 Une sémantique formelle d'un large sous-ensemble de Java

L'étude (présentée dans la Section 2.1)des concepts objets et des mécanismes liés à la concurrence nous a permis de mettre en avant des difficultés présentées par l'écriture et la mise au point de programmes Java. Cette complexité croissante des langages est

une des raisons du développement de spécifications formelles définissant une sémantique claire et surtout précise des langages par opposition aux sémantiques informelles des tutoriaux ou d'autres ouvrages de présentation des langages.
Nous avons donc défini une sémantique formelle dynamique du langage Java (étudiée dans le Chapitre 3. Pour cela, nous avons tout d'abord écrit une syntaxe concrète et abstraite du langage, puis étudié la définition de structures sémantiques permettant d'une part de représenter l'état de l'exécution à chaque étape, et d'introduire des continuations utilisées par notre sémantique *small-step*.

Les spécifications sémantiques sont liées directement au source, ce qui signifie que nous ne sommes pas limités à une machine virtuelle particulière (standard, propriétaire ou modifiée). A la place, nous interprétons directement, en utilisant des règles sémantiques liées aux constructeurs du source Java, le source du programme (et non le byte-code). Cette spécification, écrite dans le formalisme Typol, est originale par l'utilisation conjointe du style *big-step* pour définir les aspects de Java liés aux objets (comme leur création, l'héritage, etc), et du style *small-step* pour spécifier les aspects liés à la concurrence (exécution concurrente des threads, synchronisation, etc). Les ASM évitent cette dichotomie entre les aspects objets et les aspects concurrents de la formalisation de Java.
L'exécution concurrente des threads est simulée par l'introduction d'entrelacement au niveau de l'exécution des threads, ce qui permet de rendre compte des mêmes mécanismes.

Une autre originalité de cette spécification est constituée par le fait qu'elle soit exécutable au sein du système Centaur après une simple compilation. La génération automatique de l'interpréteur à partir de ces spécifications ne requiert des changements qu'au sein des spécifications, et non dans le code de l'environnement (par exemple pour passer d'une version du langage Java à une autre).

6.1.2 Un environnement d'exécution innovant

Notre spécification formelle étant exécutable, nous l'avons utilisée comme base d'un environnement d'exécution de programmes Java présenté dans le Chapitre 4.
Les effets de l'exécution du programme sont montrés à la volée, grâce à la visualisation et à l'animation de l'interprétation du programme. L'environnement que nous avons développé propose :

- des vues textuelles permettant la visualisation de :
  - la liste des objets avec pour chacun leur type, leurs attributs, et leur lock
  - la liste des threads avec leur activité
- des vues synthétiques proposant :
  - la liste des objets classés selon leur statut
  - la liste des interactions entre threads et objets

- une vue graphique représentant :
  - la topologie de l'ensemble des objets
  - les statuts des objets grâce à des codes de couleur
  - les synchronisations
- la détection de deadlocks
- un mécanisme d'abstraction permettant le passage à l'échelle.

Chaque vue est obtenue directement par un pretty-printing des structures sémantiques utilisées au sein de notre spécification. La vue graphique demande cependant un traitement plus complexe, avec une traduction des structures sémantiques vers des structures graphiques. Cette vue graphique représente la topologie de l'ensemble des objets et des threads. Grâce à la visualisation immédiate des références, des locks et des synchronisations, elle permet en particulier de détecter facilement tout objet partagé par plusieurs threads, et donc de protéger ses attributs par des méthodes *synchronized*.
Cette visualisation est animée et réalisée de façon incrémentale afin de gagner en efficacité. Elle est rendue possible grâce à un système de notification de le sémantique vers l'environnement l'informant de chaque modification et mise à jour des structures sémantique.

Aucune compilation du programme n'est nécessaire, ce qui rend possible un débuggage interactif. Il est cependant nécessaire de stopper l'exécution avant de modifier le source du programme et de la lancer à nouveau pour visualiser la nouvelle exécution. Aucune instrumentation n'est requise au niveau du source du programme. Comme l'interprétation a lieu au niveau du source, les développeurs acquièrent une meilleure compréhension du comportement de leur programme.

Les principales originalités de cet environnement sont la visualisation animée sous forme graphique de l'exécution du programme et surtout le mécanisme d'abstractions, qui permet de visualiser uniquement un sous-graphe donné ce qui est très utile en présence d'un grand nombre d'objets.

6.1.3 Une sémantique formelle comme base de vérification de propriétés

Un dernier chapitre (Chapitre 5) nous a permis de présenter une des fonctions d'une spécification formelle : servir de base à l'écriture de preuves de propriétés de programmes ou de langage.

Nous avons particulièrement étudié les propriétés du langage Java liées à la concurrence et aux mécanismes de synchronisation. Notre spécification traduit directement une grande partie de ces propriétés, que ce soit à travers la syntaxe abstraite des structures sémantiques que nous avons définies ou directement à travers nos règles Typol. Dans le cas où ce n'est pas directement évident, nous avons étudié comment exprimer ces

propriétés directement en Typol, et intégrer leur vérification à l'interprétation de programmes. Cela nous a permis d'éviter une traduction vers un autre formalisme, source possible d'erreurs.
Enfin, un schéma de la preuve par induction d'une de ces propriété a été présenté.

6.2 Perspectives

6.2.1 Développements de notre travail

Dans un premier temps de nombreuses perspectives existent à partir de notre travail. Ce paragraphe présente des perspectives de développement de notre spécification et de notre environnement. Une dernière perspective est constituée par l'intégration de la vérification d'assertions dans notre spécification.

6.2.1.1 Développements de notre spécification

Nous avons vu dans la section 2.1.4 que certains éléments du langage Java n'étaient pas spécifiés dans notre sémantique.
D'autre part, notre sémantique prend pour hypothèse que le programme Java à analyser est syntaxiquement correct, mais surtout correct au niveau typage statique. Un bon complément de notre travail serait l'écriture d'une sémantique statique du langage qui pourrait ainsi vérifier cette correction des types du programme.

6.2.1.2 Développement des interactions sémantique-environnement

Nous avons vu dans la section 2.1.3.2 que notre ordonnanceur était déterministe. Il peut être intéressant d'introduire de l'indéterminisme à ce niveau afin de coller au plus près au modèle. D'autre part, permettre à l'utilisateur d'influer sur l'ordonnancement (comme cela a été fait en Eiffel// [10], en choisissant le thread devant s'exécuter, ou en bloquant artificiellement un thread, faciliterait la mise au point des programmes en mettant en évidence les différentes exécutions possibles.

6.2.1.3 Intégration d'assertions à notre spécification

Le traitement d'assertion pourrait compléter notre travail sur les propriétés du langage en cherchant à vérifier des propriétés exprimées directement au sein des programmes (sous forme de commentaires Java spécifiques afin de ne pas avoir à supprimer ces assertions lors de l'utilisation du programme avec un compilateur "classique"). L'intégration de la vérification d'assertions à notre spécification serait tout à fait réalisable, il faudrait pour cela définir une syntaxe concrète et abstraite de ces assertions, puis naturellement leur sémantique.

6.2.2 Un nouveau langage de spécification sémantique

L'utilisation du formalisme Typol nous a permis de l'étudier, et d'en découvrir les avantages et les faiblesses. Cette étude ouvre d'autres perspectives à travers la création d'un langage de spécification sémantique innovant.

6.2.2.1 Limites du formalisme Typol

Le formalisme Typol que nous avons utilisé pour définir notre spécification sémantique du langage Java possède comme nous l'avons vu des qualités certaines comme sa syntaxe proche de celle de la déduction naturelle, son exécutabilité et sa modularité. Cependant, il possède aussi des limites :

- Il n'est pas possible d'indiquer qu'une règle doit s'appliquer pour toute variable ne représentant pas un élément d'une liste sans citer explicitement tous les éléments pour lesquels la règle ne doit pas s'appliquer. Prenons l'exemple d'un set contenant des règles traitant successivement plusieurs cas (Cas1, Cas2, ...). Si l'on veut prévoir une règle traitant de tous les autres cas non encore cités, il n'est pas possible d'exprimer une condition du style `pour tous les cas différents des précédents`, il faut dire explicitement `pour les cas différents de Cas1, différents de Cas2, ...`. Ce problème possède une solution en Typol : il s'agit de l'utilisation de la clause `error recovery`, mais cela pollue la nature logique du cadre en supprimant la possibilité de backtrack.
- Les règles d'un set Typol doivent s'exclure mutuellement, et cela entraîne une augmentation de la complexité des règles, ou du moins la multiplication de celles-ci.
- Typol ne permet pas de représenter des ensembles autrement que par des listes : par exemple l'ensemble des objets est modélisé par une liste.
- Typol ne comprend pas les ensembles de types : il n'est pas possible de demander qu'une règle s'applique pour tous les sous-types d'un type donné.
- Le passage d'une sémantique formelle écrite dans le formalisme Typol vers les systèmes de preuves est très délicat et complexe. Ce passage ne peut se faire directement de façon automatique.

Une perspective intéressante peut alors être la création d'un langage de spécification sémantique en palliant ces faiblesses.

6.2.2.2 L'environnement SmartTools

Le système Centaur dans lequel est défini possède lui aussi ses propres faiblesses. En effet, son utilisation requiert l'apprentissage de multiples langages :

- Metal pour définir la syntaxe concrète et la syntaxe abstraite
- Ppml pour définir un pretty-printer. Ce langage permet d'afficher de manière lisible un programme stocké en mémoire sous forme d'arbre de syntaxe abstraite.
- Typol pour spécifier la sémantique
- Prolog : sa connaissance facilite la mise au point des spécifications Typol

– Lisp, langage dans lequel l'environnement d'exécution est développé

Cette multitude de langages au sein du système Centaur constitue un frein à son utilisation, et à sa diffusion en introduisant des problèmes de licences, ou de portabilité suivant les systèmes d'exploitation.

L'environnement SmartTools [11] en cours de développement dans le projet OASIS de l'INRIA Sophia Antipolis évite ces écueils. Il s'agit d'un générateur d'environnements de développement qui propose comme principaux composants un éditeur structuré et des outils sémantiques basé sur l'utilisation des visiteurs (pattern Java). SmartTools est facile à utiliser grâce à son interface utilisateur graphique développée avec les API Swing de Java. Le fait d'être basé sur Java et la technologie web (XML, XSSL, etc.) permet à SmartTools d'être portable, sous Unix comme sous Windows, et de pouvoir être connecté aisément à d'autres systèmes. SmartTools est en effet beaucoup plus ouvert sur l'extérieur en permettant par exemple de prendre en entrée une spécification syntaxique DTD au lieu d'avoir à utiliser le méta-langage AST de SmartTools dédié aux spécifications syntaxiques.

6.2.2.3 Un langage de spécification sémantique avec de multiples orientations

Des recherches sont actuellement en cours afin de définir un nouveau langage de spécification sémantique qui ait les mêmes qualités que le formalisme Typol sans en avoir les inconvénients. Ce langage doit prendre en compte les différentes orientations que l'on peut suivre à partir d'une définition formelle d'un langage : en faire la base d'un environnement de développement ou/et s'en servir comme base pour effectuer des preuves de propriétés du langage ou de programmes.

Une spécification sémantique exécutable

Ce langage de spécification doit être exécutable, permettant ainsi de servir de base à un environnement d'exécution tel que Jitan, mais réalisé avec les outils Java pour le rendre portable et plus aisé à développer. Ce formalisme doit donc permettre une communication avec l'environnement.

Une transition sémantique - preuve en Coq facilitée

Une perspective intéressante serait que ce même formalisme permette d'effectuer des preuves de propriétés de programmes. Nous visons la possibilité d'une traduction directe vers le système de preuves Coq.

Un tel formalisme de spécification sémantique permettrait alors à l'utilisateur d'une part de tester ses programmes grâce à l'environnement, et d'autre part d'effectuer des preuves de propriétés de son programme grâce au système de preuves Coq sans avoir de manipulation à effectuer.

Des travaux portant eux aussi sur des preuves de propriétés basées sur une sémantique formelle d'un langage sont effectués dans le cadre des projets KEY [5, 149, 13] de Karlsruhe et KIV [144, 155] de Ausburg. Dans le cadre de ces projets, les modélisations ASM de Java sont intégrées au sein d'une logique dynamique afin de vérifier les exemples de programmes donnés dans le manuel Java de Sun.

Bibliographie

[1] M. Abadi and L. Cardelli. *A Theory of Objects*. Springer-Verlag, New York, 1996.

[2] G. Agha. *Actors: A model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, 1986.

[3] G. Agha, I. Mason, S. Smith, and C. Talcott. A foundation for actor computation. technical report, UIUC, 1995. To appear in Journal of Functional Programming.

[4] G. Agha, I. A. Mason, S. Smith, and C. Talcott. Towards a Theory of Actor Computation. *CONCUR'92, pages 565-580, Stony Brook, NY, USA, August*, 1992.

[5] W. Ahrendt, T. Baar, B. Beckert, M. Giese, E. Habermalz, R. Hähnle, W. Menzel, and P. H. Schmitt. The KeY approach: Integrating object oriented design and formal verification. In Manuel Ojeda-Aciego, Inma P. de Guzmán, Gerhard Brewka, and Luís Moniz Pereira, editors, *Proc. 8th European Workshop on Logics in AI (JELIA), Malaga, Spain*, volume 1919, pages 21–36, October 2000. URL: `ftp://ftp.cs.chalmers.se/pub/users/reiner/jelia.ps.gz`.

[6] Andrews. *Logic Programming: Operational Semantics and Proof Theory.* Cambridge University Press, Cambridge, UK, 1992.

[7] G. R. Andrews. *Concurrent programming.* Benjamin/Cummings Publishing Comp., Redwood City, 1 edition, 1991.

[8] G. R. Andrews and F. B. Schneider. Concepts and notations for concurrent programming. Technical Report TR82-520, Cornell University, Computer Science Department, September 1982.

[9] I. Attali, D. Caromel, and S. O. Ehmety. An operational semantics for the eiffel language. Technical Report RR-2732, Inria, Institut National de Recherche en Informatique et en Automatique.

[10] I. Attali, D. Caromel, S. O. Ehmety, and S. Lippi. Semantic-based visualization for parallel object-oriented programming. In *Proceedings of OOPSLA'96*, volume 31, 10 of *ACM Sigplan Notices*, San Jose, CA, October 1996. ACM Press.

[11] I. Attali, C. Courbis, P. Degenne, A. Fau, D. Parigot, and C. Pasquier. SmartTools: a generator of interactive environment tools. In *In Reinhard Wilhelm, editor, International Conference on Compiler Construction CC'01*, volume à paraître, Genova, Italy, avril 2001. Springer-Verlag. disponible à l'adresse http://www-sop.inria.fr/oasis/SmartTools/publications/ETAPS01/smartetap01.ps.

[12] K. Beck. Object explorer for visual works. Commercial Product available at <http://c2.com/ppr/about/author/kent.html>, First Class Software, Inc.

[13] B. Beckert. A dynamic logic for the formal verification of Java Card programs. In *Proceedings, Java Card Workshop (JCW), Cannes, France*, LNCS. Springer, 2001. To appear.

[14] G. Berry and G. Gonthier. The ESTEREL synchronous programming language : design, semantics, implementation. Technical Report RR-0842, Inria, Institut National de Recherche en Informatique et en Automatique.

[15] G. Blaschek. Type-safe object-oriented programming with prototypes – the concepts of omega. *Structured Programming*, 12:217–225, 1991.

[16] G. Blaschek. *Object-Oriented Programming with Prototypes*. Springer-Verlag, New York, NY, 1994.

[17] E. Börger. A Logical Operational Semantics for Full Prolog. Part I: Selection Core and Control. In E. Börger, H. Kleine Büning, M. M. Richter, and W. Schönfeld, editors, *CSL'89. 3rd Workshop on Computer Science Logic*, volume 440 of *LNCS*, pages 36–64. Springer, 1990. See Comments to [19].

[18] E. Börger. A Logical Operational Semantics of Full Prolog. Part II: Built-in Predicates for Database Manipulation. In B. Rovan, editor, *Mathematical Foundations of Computer Science*, volume 452 of *LNCS*, pages 1–14. Springer, 1990. See Comments to [19].

[19] E. Börger. A Logical Operational Semantics for Full Prolog. Part III: Built-in Predicates for Files, Terms, Arithmetic and Input-Output. In Y. Moschovakis, editor, *Logic From Computer Science*, volume 21 of *Berkeley Mathematical Sciences Research Institute Publications*, pages 17–50. Springer, 1992.

[20] E. Börger and K. Dässler. Prolog: DIN Papers for Discussion. ISO/IEC JTCI SC22 WG17 Prolog Standardization Document 58, National Physical Laboratory, Middlesex, England, 1990. A version of [17, 18, 19] proposed to the International Prolog Standardization Committee as a complete formal semantics of Prolog. An improved version is in [28].

[21] E. Börger and I. Durdanović. Correctness of compiling Occam to Transputer code. *Computer Journal*, 39(1):52–92, 1996.

[22] E. Börger, I. Durdanović, and D. Rosenzweig. Occam: Specification and Compiler Correctness. Part I: Simple Mathematical Interpreters. In U. Montanari and E. R. Olderog, editors, *Proc. PROCOMET'94 (IFIP Working Conference on Programming Concepts, Methods and Calculi)*, pages 489–508. North-Holland, 1994. A truly concurrent ASM model of Occam is defined as basis for a provably correct, smooth transition to the Transputer Instruction Set architecture. This model is stepwise refined, in a provably correct way, providing: (a) an asynchronous implementation of synchronous channel communication, (b) its optimization for internal channels, (c) the sequential implementation of processors using priority and time–slicing. See [21] for the extension of this work to cover the compilation to Transputer code.

[23] E. Börger and U. Glässer. Modelling and Analysis of Distributed and Reactive Systems using Evolving Algebras. In Y. Gurevich and E. Börger, editors, *Evolving Algebras – Mini-Course, BRICS Technical Report (BRICS-NS-95-4)*, pages 128–153. University of Aarhus, Denmark, July 1995. This is a tutorial introduction into the ASM approach to design and verification of complex computing systems. The salient features of the methodology are explained by showing how one can develop from scratch an easily understandable and transparent ASM model for PVM, the widespread virtual architecture for heterogeneous distributed computing.

[24] E. Börger and E. Riccobene. Logical Operational Semantics of Parlog. Part I: And-Parallelism. In H. Boley and M. M. Richter, editors, *Processing Declarative Knowledge*, volume 567 of *Lecture Notes in Artificial Intelligence*, pages 191–198. Springer, 1991. See comment to [26].

[25] E. Börger and E. Riccobene. Logical Operational Semantics of Parlog. Part II: Or-Parallelism. In A. Voronkov, editor, *Logic Programming*, volume 592 of *Lecture Notes in Artificial Intelligence*, pages 27–34. Springer, 1992. See comment to [26].

[26] E. Börger and E. Riccobene. A Formal Specification of Parlog. In M. Droste and Y. Gurevich, editors, *Semantics of Programming Languages and Model Theory*, pages 1–42. Gordon and Breach, 1993. An ASM formalization of Parlog, a well known parallel version of Prolog. This formalization separates explicitly the two kinds of parallelism occurring in Parlog: AND–parallelism and OR–parallelism. It uses an implementation independent, abstract notion of terms and substitutions. Improved and extended version of [24, 25], obtained combining the concurrent features of the Occam model of [84] with the logic programming model of [28]. Also published as Technical Report TR 1/93 from Dipartmento di Informatica, Università da Pisa, 1993.

[27] E. Börger and D. Rosenzweig. A Formal Specification of Prolog by Tree Algebras. In V. Čeric, V. Dobrić, V. Lužar, and R. Paul, editors, *Information Technology Interfaces*, pages 513–518. University Computing Center, Zagreb, Zagreb, 1991. Prompted by discussion in the international Prolog standardization committee (ISO/IEC JTC1 SC22 WG17), this paper suggests to replace the stack based model of [17] and the stack implementation of the tree based model of [18] by a pure tree model for Prolog. An improved version of the latter is the basis for [28] where also an error in the treatment of the *catch* built-in predicate is corrected.

[28] E. Börger and D. Rosenzweig. A Mathematical Definition of Full Prolog. In *Science of Computer Programming*, volume 24, pages 249–286. North-Holland, 1994. An abstract ASM specification of the semantics of Prolog, rigorously defining the international ISO 1995 Prolog standard by stepwise refinement. Revised and final version of [17, 18, 20, 27]. An abstract of this was issued as *Full Prolog in a Nutshell* in *Logic Programming* (Proceedings of the 10th International Conference on Logic Programming) (D. S. Warren, Ed.), MIT Press 1993. A preliminary version appeared under the title *A Simple Mathematical Model for Full Prolog* as research report TR-33/92, Dipartimento di Informatica, Università di Pisa, 1992.

[29] E. Börger, J. Schmid, and R. Stärk. *Java and the Java Virtual Machine - Definition, Verification, Validation.* Springer-Verlag, 2001.

[30] E. Börger and W. Schulte. Defining the Java Virtual Machine as Platform for Provably Correct Java Compilation. In *23rd International Symposium on Mathematical Foundations of Computer Science*, LNCS. Springer-Verlag, 1998. to appear.

[31] E. Börger and W. Schulte. A Programmer Friendly Modular Definition of the Semantics of Java. In *Formal Syntax and Semantics of Java.* Springer-Verlag, LNCS 1523, 1998.

[32] A. H. Borning. Classes versus prototypes in object-oriented languages. In *Proceedings of the ACM-IEEE Fall Joint Computer Conference, Montvale (NJ), USA*, pages 36–39, 1986.

[33] P. Borras and et al. Centaur: the System. In *SIGSOFT'88 Third Annual Symposium on Software Development Environments*, Boston, 1988.

[34] G. Brat, K. Havelund, S. Park, and W. Visser. Model Checking Programs. In *Proc of The IEEE International Conference on Automated Software Engineering (ASE)*, September 2000.

[35] M. H. Brown. Zeus: a system for algorithm animation and multiview editing. In *Proc. of the IEEE Workshop on Visual Languages*, 1991.

[36] K. B. Bruce, J. Crabtree, and G. Kanapathy. An operational semantics for TOOPLE: A statically-typed object-oriented programming language. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Semantics 9th International Conference, New Orleans, LA, USA, Proceedings*, volume 802 of *Lecture Notes in Computer Science*, pages 603–626. Springer-Verlag, New York, NY, April 1993.

[37] J. Bryans, J. Davies, and S. Schneider. Towards a denotational semantics for ET-LOTOS. *Lecture Notes in Computer Science*, 962, 1995.

[38] *Concurrent Object-Oriented Programming.* Communications of the ACM, 36 (9), 1993. Special issue.

[39] J. R. Carbonell and A. M. Collins. Natural semantics in artificial intelligence. In Nils J. Nilsson, editor, *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, pages 344–351, Standford, CA, August 1973. William Kaufmann.

[40] D. Caromel. *Programmation parallèle asynchrone et impérative: études et propositions.* Ph.D. thesis, Université de Nancy, February 1991.

[41] CenterLine Software, Cambridge, MA. *CodeCenter, The Programming Environment*, 1992.

[42] C. Chambers, D. Ungar, and E. Lee. An efficient implementation of self, a dynamically-typed object-oriented language based on prototypes. *SIGPLAN Notices*, 24(10), October 1989.

[43] C. Choppy et al. A LISP compiler for FP language and its proof via algebraic semantics. In *Lecture Notes in Computer Science 185.* Springer-Verlag, New York, NY.

[44] Koen Claessen, Ton Vullinghs, and Erik Meijer. Structuring graphical paradigms in TkGofer. In *International Conference on Functional Programming*. ACM, June 1997.

[45] C. Colby, L. Jategaonkar Jagadeesan, R. Jagadeesan, K. Läufer, and C. Puchol. Design and Implementation of Triveni: a Process-Algebraic API for Threads + Events. In *Proceedings of the 1998 International Conference on Computer Languages*, pages 58–67. IEEE Computer Society Press, 1998.

[46] C. Colby, L. Jategaonkar Jagadeesan, R. Jagadeesan, K. Läufer, and C. Puchol. Objects and Concurrency in Triveni: a Telecommunication Case Study in Java. In *Proceedings of the 4th Conference on Object-Oriented Technologies and Systems (COOTS-98)*, pages 133–148, Berkeley, April 27–30 1998. USENIX Association.

[47] C. Colby, L. Jategaonkar Jagadeesan, R. Jagadeesan, K. Läufer, and C. Puchol. The Semantics of Triveni: a Process-Algebraic API for Threads + Events. In Rance Cleaveland, Michael Mislove, and Philip Mulry, editors, *Electronic Notes in Theoretical Computer Science*, volume 14. Elsevier Science Publishers, 2000.

[48] Compaq. Visual Threads. Commercial Product available at <http://www5.compaq.com/products/software/visualthreads/>.

[49] M. Consens, A. Mendelzon, and A. Ryman. Visualizing and querying software structures. In *Proc. of the* 14^{th} *International Conference on Software Engineering*, pages 138–156, May 1992.

[50] J. Corbett, M. Dwyer, C. Pasareanu J. Hatcliff, Robby, S. Laubach, and H. Zheng. Bandera : Extracting Finite-state Models from Java Source Code. In *2000 International Conference on Software Engineering*, Februar 2000.

[51] O.-J. Dahl and K. Nygaard. Simula Begin. Technical report, Norsk Regnesentral (Norwegian Computing Center), Oslo/N, 1967.

[52] D. Dean. The security of static typing with dynamic linking. In *Proceedings of the Fourth ACM Conference on Computer and Communications Security*, Zurich, Switzerland, April 1997.

[53] C. Demartini, R. Iosif, and R. Sisto. Modeling and Validation of Java Multithreaded Applications using Spin. In *Proc. of the 4th SPIN Workshop, Paris, France*, November 1998.

[54] P. Deransart, M. Jourdan, and B. Lorho. *Attribute Grammars: Definitions, Systems and Bibliography*, volume 323 of *Lecture Notes in Computer Science*. Springer-Verlag, New York, NY, 1989.

[55] J. Despeyroux. An algebraic specification of a Pascal compiler. *ACM SIGPLAN Notices*, 18(12):34–48, December 1983.

[56] T. Despeyroux. Typol: A Formalism to Implement Natural Semantics. Research Report 94, INRIA, 1988.

[57] A. Di Pierro and C. Palamidessi. A logical denotational semantics for constraint logic programming. *Lecture Notes in Computer Science*, 788, 1994.

[58] C. Dichter. We test the top 6 Java visual IDEs. Product Reviews available at <http://www.javaworld.com/javaworld/jw-04-1998/jw-04-visual-ides.html>, JavaWorld: IDG's magazine for the Java community, 1998.

[59] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Murthy, C. Parent, C. Paulin-Mohring, and B. Werner. The Coq proof assistant user's guide. Rapport Techniques 154, INRIA, Rocquencourt, France, 1993. Version 5.8.

[60] D. G. Drake. Introduction to Java threads. *JavaWorld: IDG's magazine for the Java community*, 1(2), April 1996.

[61] S. Drossopoulou and S. Eisenbach. Is the Java Type System Sound? In *4th Int. Workshop Foundations of Object-Oriented Languages*, 1997.

[62] S. Drossopoulou and S. Eisenbach. Java is Type Safe - Probably. In *ECOOP'97*, LNCS 1241, pages 389–418. Springer Verlag, January 1997.

[63] S. Drossopoulou and S. Eisenbach. Towards an Operational Semantics and Proof of Type Soundness for Java. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java*, volume 1523 of *LNCS*. Springer-Verlag, 1998.

[64] S. Drossopoulou and S. Eisenbach. Describing the semantics of Java and proving type soundness. *Lecture Notes in Computer Science*, 1523, 1999.

[65] S. Drossopoulou, J. Uhl, G. Persch, G. Goos, Dausmann, M., and G. Winterstein. An attribute grammar for Ada. *ACM SIGPLAN Notices*, 17(6):334–339, June 1982.

[66] M. Dwyer and L. A. Clarke. Data Flow Analysis for Verifying Properties of Concurrent Programs. In *Proc, of the Second ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 62–75, December 1994.

[67] D. Dyer. Can Assure save Java from the perils of multithreading? article available at <http://www.javaworld.com/javaworld/jw-10-1998/jw-10-assure.html>, JavaWorld: IDG's magazine for the Java community, 1998.

[68] D. Dyer. Capsule Reviews of JAVA tools. Product Reviews Available at <http://www.andromeda.com/people/ddyer/java/Reviews.html>, Andromeda Software, 1998.

[69] M. Felleisen, R. B. Findler, M. Flatt, and S. Krisnamurthi. The DrScheme project: An overview. *ACM SIGPLAN Notices*, 33(6):17–23, June 1998.

[70] R. B. Findler, C. Flanagan, M. Flatt, S. Krishnamurthi, and M. Felleisen. DrScheme: A pedagogic programming environment for scheme. *Programming Languages: Implementations, Logics, and Program s*, 1292:369–388, September 1997.

[71] M. Flatt, S. Krishnamurthi, and M. Felleisen. Classes and mixins. In *Conference Record of POPL 98: The 25TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, California*, pages 171–183, New York, NY, January 1998. ACM.

[72] M. Flatt, S. Krishnamurthi, and M. Felleisen. A programmer's reduction semantics for classes and mixins. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java*, volume 1523 of *LNCS*. Springer-Verlag, 1998.

[73] K. A. Frenkel. Special issue on parallelism. *Communications of the ACM*, 29(12):1168–1169, December 1986.

[74] G. Friedrich, W. H., C. Stary, and M. Stumptner. Objview: A task-oriented, graphics-based tools for object visualization and arrangement. In *Proc. ECOOP '89*, pages 299–310, Nottingham, July 1989. Cambridge University Press.

[75] U. Goltz. CCS and petri nets. In *Semantics of Systems of Concurrent Processes*, pages 334–357, Berlin - Heidelberg - New York, April 1990. Springer.

[76] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison–Wesley, 1996.

[77] J. E. Grass. Object-oriented design archaeology with CIA++. *Computing Systems*, 5(1):5–67, 1992.

[78] P. Greg. *Using Visual J++*. Que, 1996. ISBN 078970899X.

[79] I. Guessarian. Some applications of algebraic semantics. In J. Winkowski, editor, *Proceedings of the 7th Symposium on Mathematical Foundations of Computer Science*, volume 64 of *LNCS*, pages 257–266, Zakopane, Poland, September 1978. Springer.

[80] I. Guessarian. *Algebraic semantics*, volume 99 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., New York, NY, USA, 1981.

[81] Y. Gurevich. Evolving algebras: An attempt to discover semantics. In G. Rozenberg and A. Salomaa, editors, *Current Trends in Theoretical Computer Science*, pages 266–292. World Scientific, River Edge, NJ, 1993.

[82] Y. Gurevich. Evolving algebras 1993: Lipari Guide. In Egon Börger, editor, *Specification and Validation Methods*, pages 9–37. Oxford University Press, 1994.

[83] Y. Gurevich. The sequential ASM thesis. *BEATCS: Bulletin of the European Association for Theoretical Computer Science*, 67, 1999.

[84] Y. Gurevich and L. Moss. Algebraic Operational Semantics and Occam. In E. Börger, H. Kleine Büning, and M. M. Richter, editors, *CSL'89, 3rd Workshop on Computer Science Logic*, volume 440 of *LNCS*, pages 176–192. Springer, 1990. The first application of ASMs to distributed parallel computing with the challenge of true concurrency. See [22, 21].

[85] Y. Gurevich, W. Schulte, and C. Wallace. Investigating Java Concurrency using Abstract State Machines. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 151–176. Springer-Verlag, 2000. An ASM specification and verification of Java's model of concurrency, including threads and synchronization. Also in TIK-Report 87, Swiss Federal Institute of Technology (ETH) Zurich, March 2000, 227–271, and as University of Delaware Department of Computer & Information Sciences Technical Report 2000-04.

[86] V. Haarslev and R. Möller. A framework for visualizing object-oriented systems. In *Proc. OOPSLA/ECOOP '90, ACM SIGPLAN Notices 25 (10)*, pages 237–244, October 1990.

[87] K. Havelund and T. Pressburger. Model Checking Java Programs using Java PathFinder. In *International Journal on Software Tools for Technology Transfer (STTT)*, 1998.

[88] K. Havelund and J. Skakkebaek. Applying Model Checking in Java Verification. In *Proc. of the 5th International SPIN Workshop*, volume 1680, pages 216–231. Springer, LNCS, September 1999.

[89] L. Holenderski. Lustre. *Lecture Notes in Computer Science*, 891, 1995.

[90] A. Holub. Programming Java threads in the real world: Threading architectures. *JavaWorld: IDG's magazine for the Java community*, 3(9), September 1998.

[91] G. J. Holzmann. The Spin Model Checker. *IEEE Trans. on Software Engineering, Volume 23*, pages 279–295, May 1997.

[92] A. Le Hors. Graph: A directed graph displaying server, GIPE 2 Esprit project, 4th review report, workpackage 4, 1992.

[93] IBM Corporation. Visual Age for Java. Commercial Product available at <http://www.software.ibm.com/ad/vajava/>, 1997.

[94] T. Jensen, D. Le Métayer, and T. Thorn. Security and Dynamic Class Loading in Java: a Formalisation. In *Proceedings of the 1998 IEEE International Conference on Computer Languages*, pages 4–15, May 1998.

[95] D. F. Jerding, J. T. Stasko, and T.Ball. Visualizing interactions in program executions. In *Proc. of the* 17^{th} *International Conference on Software Engineering*, pages 360–370, May 1997.

[96] G. Kahn. Natural semantics. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, volume 247 of *Lecture Notes in Computer Science*, pages 22–39. Springer-Verlag, 1987.

[97] G. Kahn, B. Lang, and B. Mélèse. Metal : A Formalism to Specify Formalisms. *Science of Computer Programming, Volume 3, North-Holland*, 1983.

[98] S. Kent. Constraint diagrams: Visualizing invariants in object-oriented models. In *Proc. OOPSLA'97 (Object-Oriented Programming: Systems, Languages, and Applications)*, volume 32, number 10. ACM Press, Sigplan Notices, Oct 1997.

[99] E. Kraemer and J. T. Stasko. The visualization of parallel systems: an overview. *Journal of Parallel and Distributed Computing, 18*, 1993.

[100] C. Laffra and A. Malhotra. Hotwire – A visual debugger for C++. In *USENIX Sixth C++ Technical Conference*, pages 109–122, Cambridge, MA, April 11-14 1994. USENIX.

[101] W. R. LaLonde and J. R. Pugh. *Inside Smalltalk: Volume I.* Prentice Hall, Englewood Cliffs, NJ, 1990.

[102] W. R. LaLonde and J. R. Pugh. *Inside Smalltalk: Volume II.* Prentice Hall, Englewood Cliffs, NJ, 1990.

[103] L. Lamport. The 'hoare logic' of concurrent programs. In Springer-Verlag, editor, *Acta Informatica 14*, pages 21–37, 1980.

[104] L. Lamport and F. B. Schneider. The "hoare logic" of CSP and all that. *TOPLAS*, 6(2):281–296, April 1984.

[105] J. Launchbury. A natural semantics for lazy evaluation. In *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 144–154, New York, NY, 1993. ACM.

[106] D. Lea. *Concurrent programming in Java: design principles and patterns.* The Java series. Addison-Wesley, Reading, MA, USA, 1996.

[107] D. Lea. *Concurrent Programming Using Java.* Addison-Wesley, 1996.

[108] G. T. Leavens, A. L. Baker, and C. Ruby. JML: a java modeling language. In *Formal Underpinnings of Java Workshop (at OOPSLA '98)*, October 1998. `http://www-dse.doc.ic.ac.uk/~sue/oopsla/cfp.html`.

[109] G. T. Leavens, K. Rustan, M. Leino, E. Poll, C. Ruby, and B. Jacobs. JML: Notations and Tools Supporting Detailed Design in Java. In *OOPSLA 2000 Companion, Minneapolis, Minnesota*, pages 105–106. ACM, October 2000.

[110] Gary T. Leavens, Albert L. Baker, and Clyde Ruby. JML: A notation for detailed design. In Haim Kilov, Bernhard Rumpe, and Ian Simmonds, editors, *Behavioral Specifications of Businesses and Systems*, pages 175–188. Kluwer Academic Publishers, Boston, 1999.

[111] L. Lemay and Charlesworth. *Teach Yourself SunSoft Java Workshop in 21 Days.* SAMS, 1996. ISBN 1575211599.

[112] R. Lencevicius, U. Hölzle, and A. K. Singh. Query-based debugging of object-oriented programs. In *Proc. OOPSLA'97 (Object-Oriented Programming: Systems, Languages, and Applications)*, volume 32, number 10. ACM Press, Sigplan Notices, Oct 1997.

[113] J. J. Lukkien. An operational semantics for the guarded command language. In R. S. Bird, C. C. Morgan, and J. C. P. Woodcock, editors, *Mathematics of Program Construction, Second International Conference, Oxford, U.K.*, volume 669 of *Lecture Notes in Computer Science*, pages 233–249. Springer-Verlag, New York, NY, June/July 1993.

[114] W. Marek. A natural semantics for modal logic over databases. *Theoretical Computer Science*, 56(2):187–209, February 1988.

[115] C. McManis. Java in depth: Synchronizing threads in Java. *JavaWorld: IDG's magazine for the Java community*, 1(2), April 1996.

[116] Metrowerks. Metrowerks CodeWarrior. Commercial Product available at <http://www.metrowerks.com/>, Austin, TX, 1995.

[117] B. Meyer. *Eiffel: The Language.* Object-Oriented Series. Prentice Hall, New York, NY, 1992.

[118] E. Morcos-Chounet and A. Conchon. Ppml: A General Formalism to Specify Pretty-printing. *Proc. of the IFIP Congress Dublin, North-Holland*, 1986.

[119] P. D. Mosses. Denotational semantics. In J. van Leewen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 11, pages 577–631. The MIT Press, New York, NY, 1990.

[120] H. Muller, J. Winckler, S. Grzybek, M. Otte, B. Stoll, F. Equoy, and N. Higelin. PASTIS-program animation using X. In Anonymous, editor, *European X Window System Conference*, pages 104–111, Edinburgh, UK, 1990. CEP Consultants.

[121] B. A. Myers. Incense: A system for displaying data structures. *Computer Graphics*, 17(3):115–125, July 1983.

[122] G. Naumovich, S. Avrunin, and L. A. Clarke. Data Flow Analysis for Checking Properties of Concurrent Java Programs. In *Proc. of the International Conference on Software Engineering*, May 1999.

[123] O. Nierstrasz. Active objects in hybrid. In *Proc. OOPSLA '87, ACM SIGPLAN Notices 22 (12)*, pages 243–253, 1987.

[124] O. Nierstrasz. Two models of concurrent objects. In *ACM SIGPLAN Notices 24 (4), Proc. Workshop on Object-Based Concurrent Programming (San Diego, Sept 1988)*, pages 174–176, April 1989.

[125] O. Nierstrasz and M. Papathomas. Towards a type theory for active objects. In *ACM OOPS Messenger, 2 (2), Proc. OOPSLA/ECOOP 90 Workshop on Object-Based Concurrent Systems*, pages 89–93, April 1991.

[126] T. Nipkow. Invited talk: Embedding programming languages in theorem provers. In Harald Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, volume 1632 of *LNAI*, page 398, Berlin, July 7–10, 1999. Springer-Verlag.

[127] T. Nipkow and D. Von Oheimb. Java Light is Type Safe - Definitely. In *25st ACM Symp. Principles of Programming Languages*, 1998.

[128] T. Nipkow and D. Von Oheimb. Machine-checking the Java specification: Proving type-safety. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java*, volume 1523 of *LNCS*, pages 119+37. SV, 1998.

[129] Nova Software Labs. JProbe Profiler. Commercial Product available at <http://www.nsl.fr/index.html>, Paris, France, 1998.

[130] K. Nygaard and O.-J. Dahl. Simula 67. In R. W. Wexelblat, editor, *History of Programming Languages*. acm press, 1981.

[131] S. Oaks and H. Wong. *Java Threads*. O'Reilly, 1997.

[132] J. Paakki. Attribute grammar paradigms: A high-level methodology in language implementation. *ACM Computing Surveys*, 27(2):196–255, June 1995.

[133] A. Paepcke. *Object-Oriented Programming: The CLOS Perspective*. The MIT Press, 1993.

[134] D. Park, U. Stern, and D. Dill. Java Model Checking. In *First International Workshop on Automated Program Analysis, Testing and Verification, Limerick, Ireland*, June 2000.

[135] W. De Pauw, R. Helm, D. Kimelman, and J. Vlissides. Visualizing the behavior of object-oriented systems. In *Proc. OOPSLA '93, ACM SIGPLAN Notices, 28 (10)*, pages 326–337, October 1993.

[136] W. De Pauw, D. Kimelman, and J. Vlissides. Modeling object-oriented program execution. In *Proc. ECOOP '94*, LNCS 821, pages 163–182, Bologna, Italy, July 1994. Springer-Verlag.

[137] G. D. Plotkin. A Structural Approach to Operational Semantics. Report, DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark, 1981.

[138] A. Poetzsch-Heffter and P. Müller. A programming logic for sequential Java. In S. D. Swierstra, editor, *Programming Languages and Systems (ESOP '99)*, volume 1576 of *Lecture Notes in Computer Science*, pages 162–176. Springer-Verlag, 1999.

[139] E. Poll, J. van den Berg, and B. Jacobs. Formal Specification and Verification of JavaCard's Application Identifier Class. In *Proceedings of the JavaCard Workshop, Cannes, France*, September 2000.

[140] E. Poll, J. van den Berg, and B. Jacobs. Specification of the JavaCard API in JML. In *Fourth Smart Card Research and Advanced Application Conference (IFIP Cardis)*. Kluwer Academic Publishers, 2000.

[141] J. Power and G. Rosolini. A modular approach to denotational semantics. *Lecture Notes in Computer Science*, 1443, 1998.

[142] D. Prawitz. *Natural Deduction*. Almquist and Wiksell, Stockholm, 1965.

[143] Z. Qian. A Formal Specification of the Java Virtual Machine Instructions for Objects, Methods and Subroutines. In *Formal Syntax and Semantics of Java*. Springer-Verlag, LNCS 1523, 1998.

[144] W. Reif. The KIV-approach to Software Verification. In M. Broy and S. Jähnichen, editors, *KORSO: Methods, Languages, and Tools for the Construction of Correct Software – Final Report*, LNCS 1009. Springer, Berlin, 1995.

[145] G.C. Roman and et al. A System for Declarative Visualization of Concurrent Computations. *Journal of Visual Languages and Computing, 3 (2)*, 1992.

[146] M. Roncken and R. Gerth. A denotational semantics for synchronous and asynchronous behavior with multiform time. In Marta Zofia Kwiatkowska, Michael William Shields, and Richard Monro Thomas, editors, *Proceedings of the International BCS-FACS Workshop on Semantics for Concurrency*, Workshops in Computing, pages 21–37, Berlin, July 1990. Springer.

[147] A. W. Roscoe. Denotational semantics for OCCAM. In S. D. Brookes, A. W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, volume 197 of *Lecture Notes on Computer Science*, pages 306–329. Springer-Verlag, Berlin, 1985.

[148] B. J. Ross. Using algebraic semantics for proving prolog program termination and transformation. In *Proceedings of the 1991 UK ALP meeting*, pages 63–70, Edinburgh, April 1991.

[149] T. Sattler and W. Ahrendt. Embedding asms into state transition diagrams. Technical Report 2000/20, University of Karlsruhe, Department of Computer Science, January 2000.

[150] G. Schellhorn and W. Ahrendt. Reasoning about Abstract State Machines: The WAM Case Study. *Journal of Universal Computer Science*, 3(4):377–413, 1997.

[151] N. Shankar. Symbolic analysis of transition systems. In Yuri Gurevich, Phillip W. Kutter, Martin Odersky, and Lothar Thiele, editors, *Abstract State Machines: Theory and Applications (ASM 2000)*, number 1912 in Lecture Notes in Computer Science, pages 287–302, Monte Verità, Switzerland, March 2000. Springer-Verlag.

[152] M. S. Sherman. *Paragon: A Language Using Type Hierarchies for the Specification, Implementation and Selection of Abstract Data Types*. PhD thesis, School of Computer Science, Carnegie Mellon University, August 1983. CMU-CS-83-147, also published by Springer-Verlag, LNCS, Volume 189, 1985.

[153] C. R. Snow. *Concurrent Programming*, volume 26 of *Cambridge Computer Science Texts*. Cambridge University Press, New York, NY, 1992.

[154] J. T. Stasko and E. Kraemer. A methodology for building application-specific visualizations of parallel programs. *Journal of Parallel and Distributed Computing, 18*, 1993.

[155] K. Stenzel. Verification of JavaCard Programs. Technical report 2001-5, Institut für Informatik, Universität Augsburg, Germany, 2001. Available at http://www.Informatik.Uni-Augsburg.DE/swt/fmg/papers/.

[156] Step Ahead Software Pty Ltd. Javelin. Commercial Product available at <http://www.ozemail.com.au/ stepsoft/javelin.htm>, NSW 2077 Australia, 1998.

[157] G. S. Stiles. Safe and Verifiable Design of Multithreaded Java Programs with CSP and FDR. In *Proceedings of Formal Underpinnings of Java, an OOPSLA'98 Workshop*, Vancouver, CA, October 1998. Technical Report, Princeton University.

[158] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, 1986.

[159] Sun. JDB The Java Debugger. Commercial Product available at <http://java.sun.com/products/jdk/1.1/docs/tooldocs/solaris/jdb.html>, Palo Alto, CA, 1997.

[160] Sun. Java Debugger Interface. Commercial Product available at <http://java.sun.com/products/jdk/1.3/docs/guide/jpda/jdi/index.html>, Palo Alto, Ca, 1998.

[161] SuperCede, Inc. SuperCede Java. Commercial Product available at <http://www.supercede.com>, Bellevue, WA, 1998.

[162] D. Syme. Proving Java Type Soundness. Technical report 427, University of Cambridge Computer Laboratory, 1997.

[163] D. Syme. Proving Java type soundness. *Lecture Notes in Computer Science*, 1523, 1999.

[164] A. Taivalssari. Classes versus prototypes: Some philosophical and historical observations. *Journal of Object-Oriented Programming*, 10(7):44–49, November 1997.

[165] Tek-Tools, Inc. Kawa. Commercial Product available at <http://www.tek-tools.com/kawa/>, Dallas TX, 1998.

[166] D. Terrasse. Encoding natural semantics in Coq. In V. S. Alagar, editor, *Proceedings of the Fourth International Conference on Algebraic Methodology and Software Technology*, pages 230–244, Montreal, Canada, July 1995. Springer-Verlag LNCS 936.

[167] D. Terrasse. Vers un environnement de développement de preuves en Sémantique naturelle. Thèse de doctorat, ENPC, 1995.

[168] M. Tienari. *On the Definition of an Attribute Grammar*, volume 94 of *Lecture Notes in Computer Science*, pages 408–414. Springer-Verlag, Heidelberg, FRG, 1980.

[169] C. Tomlinson and M. Scheevel. Concurrent object-oriented programming languages. In Won Kim and Farederick H. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, chapter 5, pages 79–124. Addison-Wesley Publishing Co., Reading, Mass., 1989.

[170] E. Wagner, J. Wright, J. Goguen, and J. Thatcher. Some fundamentals of order-algebraic semantics. In Andrzej Mazurkiewicz, editor, *Proceedings, Fifth Symposium on Mathematical Foundations of Computer Science*, volume 45 of *Lecture Notes in Computer Science*, pages 153–168. Springer, 1976.

[171] E. G. Wagner, S. L. Bloom, and J. W. Thatcher. Why algebraic theories? In M. Nivat and J. C. Reynolds, editors, *Algebraic Semantics*, pages 607–634. Cambridge University press, 1985.

[172] D. A. Watt. An extended attribute grammar for PASCAL. *ACM SIGPLAN Notices*, 14(3):60–74, February 1979.

[173] D. A. Watt. Executable denotational semantics. *Software: Practice and Experience*, 16(1):13–43, 1986.

[174] P. Wegner. Design issues for object-based concurrency. In *Proc. of the ECOOP '91 Workshop on Object-Based Concurrent Computing*, LNCS 612, pages 245–256, 1992.

[175] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction.* Foundations of Computing series. MIT Press, February 1993.

[176] A. Yonezawa and M. Tokoro, editors. *Concurrent Object-Oriented Programming.* The MIT Press, Cambridge, Mass., 1987.