Perspectives pour l'interprétation abstraite

Patrick COUSOT
Département d'informatique
École normale supérieure
45 rue d'Ulm, 75230 Paris cedex 05, France

mailto:Patrick.Cousot@ens.fr
http://www.di.ens.fr/~cousot

Séminaire « Où mène l'interprétation abstraite? »
École normale supérieure, lundi 13 mars 2000.

1. Rappel historique de quelques éléments d'interprétation abstraite

Origine : analyse statique de programmes [1]

In high level languages, compile time type verifications are usualy incomplete, and dynamic coherence checks must be inserted in object code.

We present here a general algorithm allowing most of these certifications to be done at compile time. The stetic analysis of programs we do consists of an abstract evaluation of these programs.

The essential idea is that, when doing abstract evaluation of a program, "abstract" values are associated with variables instead of the "concrete" values used while actually executing. The basic operations of the language are interpreted accordingly and the abstract interpretation then consists in a transitive closure mechanism.

Référence

[1] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, pages 106–130. Dunod, Paris, 1976.

Exemple

```
lwb := 1 ; upb := 100 ;
{(lwb, [1,1]), (upb, [100, 100])}
L : {(lwb, [1,101]), (upb, [0,100]), (m, [1,100])}
if upb < lwb then
      {(lwb, [1,101]), (upb, [0,100]), (m, [1,100])}
        unsuccessfull search ;
fi ;
{(lwb, [1,100]), (upb, [1,100]), (m, [1,100])}
m := (upb + lwb) ÷ 2 ;
{(lwb, [1,100]), (upb, [1,100]), (m, [1,100])}
if K = R(m) then
        successfull search ;
elsif K < R(m) then
        upb := m - 1 ;
       {(lwb, [1,100]), (upb, [0,99]), (m, [1,100])}
else
        lwb := m + 1 ;
        {(lwb, [2, 101]), (upb, [1,100]), (m, [1,100])}
fi ;
{(lwb, [1,101]), (upb, [0,100]), (m, [1,100])}
go to L ;
```

Idées principales

– utilisation de *correspondances de Galois* pour établir le lien entre propriétés [1] concrètes et abstraites avec extension naturelle aux opérations du langage ;

– utilisation de *domaines abstraits infinis* [2] (intervalles) ;

– accélération de la convergence par *élargissement* et *rétrécissement*.

1. "propriétés" et non simplement "valeurs".

2. ne satisfaisant pas la condition de chaîne croissante.

L'interprétation abstraite est un modèle unificateur et général de l'approximation de sémantiques [1]

It is our feeling that most program analysis techniques may be understood as abstract interpretations of programs.

Référence

[1] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, Californie, 1977. ACM Press, New York, NY, USA.

Idées principales

– la sémantique standard décrit une exécution des programmes du langage ;
– la sémantique collectrice décrit les propriétés concrètes des programmes du langage ;
– une sémantique abstraite est une approximation de la sémantique collectrice ;
– les sémantiques/propriétés de programmes s'expriment sous forme de points fixes ;
– les méthodes de preuve de propriétés des programmes sont des sémantiques abstraites ;

Idées principales (suite)

– les sémantiques abstraites s'organisent en un treillis des interprétations abstraites ;

– la définition itérative chaotique ou asynchrone des points fixes conduit à des algorithmes d'analyse statique calculant ou approchant des sémantiques abstraites ;

– l'essentiel est de disposer de méthodes constructives d'approximation effective de points fixes.

Modèles des programmes, spécification de leurs propriétés [1 , 2]

le problème de l'analyse sémantique des programmes est étudié, indépendamment des problèmes de définition de langage, dans le cadre très général de l'étude du comportement d'un système dynamique discret.

...

Nous montrons que ces conditions s'obtiennent comme solutions d'équations de points fixes ou de systèmes d'équations, quand l'ensemble des états du système dynamique discret est partitionné.

Référence

[1] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, 21 mars 1978.

[2] P. Cousot. Semantic foundations of program analysis. In S.S. Muchnick and N.D. Jones, éditeurs, *Program Flow Analysis: Theory and Applications*, chapitre 10, pages 303–342. Prentice-Hall, Inc., Englewood Cliffs, NJ, USA, 1981.

Idées principales

– L'interprétation abstraite peut s'étudier indépendamment d'un langage de programmation particulier (systèmes de transition) ;
– Les propriétés des programmes s'expriment comme :
  – une combinaison booléenne de plus petits ou plus grands points fixes d'opérateurs croissants sur des treillis complets ;
  – les opérateurs de base sont des transformateurs de prédicats [3].

3. pre , post , etc.

La conception de sémantiques abstraites [1 , 2]

Some interesting consequences of the existence of a best predicate transformer are examined. One is that we have in hand a formal specification of the programs which have to be written in order to implement a program analysis framework once a representation of the space of approximate assertions has been chosen.

Référence

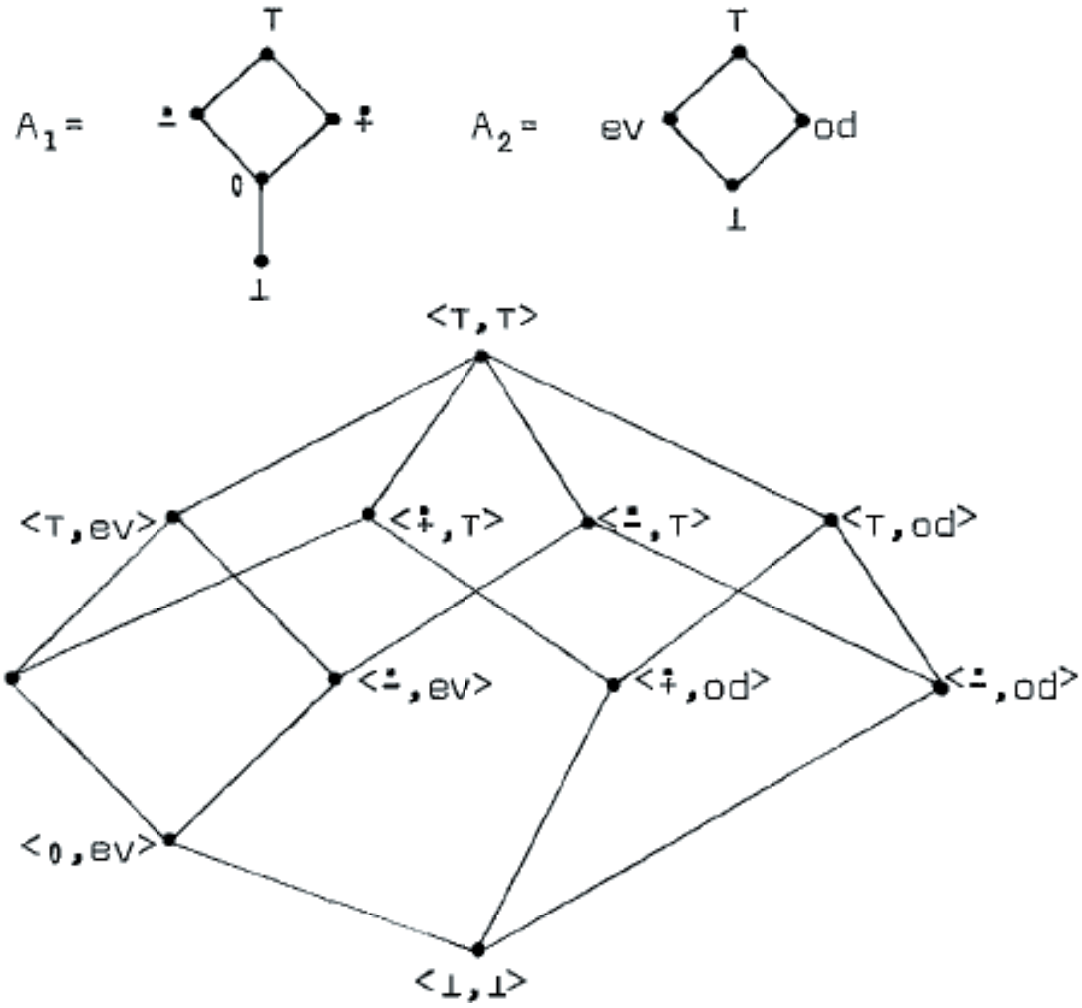
[1] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, 21 mars 1978.

[2] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, NY, USA.

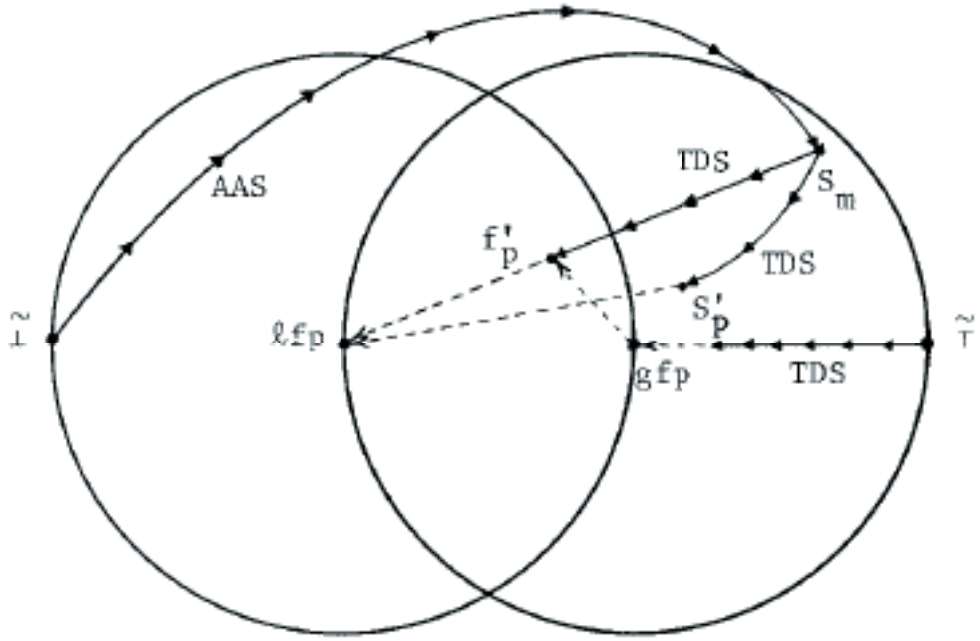
Idées principales

– Une sémantique abstraite est entièrement déterminée par :
  – une sémantique standard ,
  – la spécification des propriétés à analyser ou vérifier,
  – l'abstraction de ces propriétés (correspondance de Galois vers des domaines abstraits),
  – l'extrapolation de ces propriétés (élargissement/rétrécissement).
– L'abstraction peut être conçue de manière compositionnelle (bases de la théorie des domaines abstraits).

Exemple (produit réduit)



Approximation effective de points fixes [1]



Référence

[1] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, Californie, 1977. ACM Press, New York, NY, USA.

Idées principales

– approximations statiques [1] :

D'un point de vue algébrique, cette approximation se formalise par une fermeture sur le domaine des équations à résoudre, fermeture que nous définirons de façon équivalente au moyen de parties de Moore , de familles d'idéaux principaux, de relations de congruence ou de paires de fonctions adjointes.

Référence

[1] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, 21 mars 1978.

Idées principales (suite)

– approximations dynamiques [1] :

Pour accélérer la convergence d'une itération qui ne se stabilise par naturellement en un nombre fini de pas, nous proposons d'extrapoler en cours de calcul les termes de la suite des itérés pour obtenir, en un nombre fini de pas, une approximation de sa limite.

– l'approximation dynamique est plus générale et puissante [2].

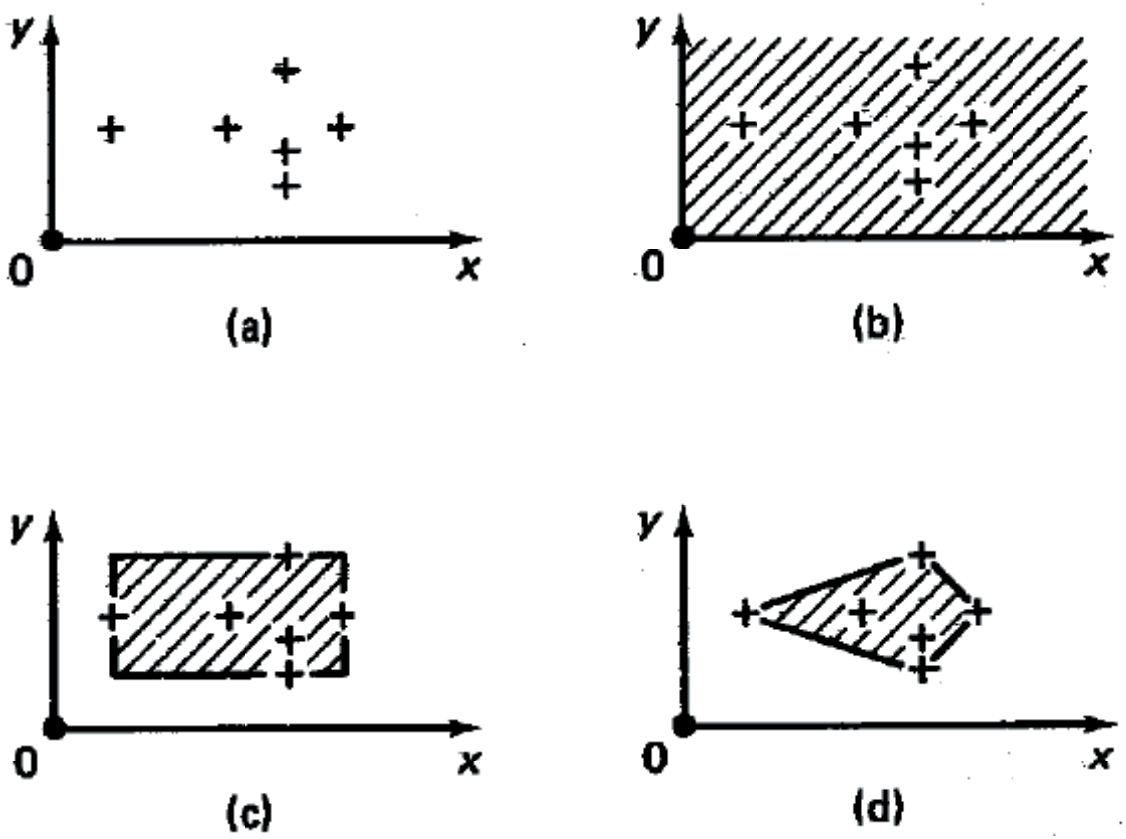
Références

[1] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, 21 mars 1978.

[2] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In M. Bruynooghe and M. Wirsing, éditeurs, *Proceedings of the International Workshop Programming Language Implementation and Logic Programming, PLILP '92*, Louvain, Belgique, 13–17 août 1992, Lecture Notes in Computer Science 631, pages 269–295. Springer-Verlag, 1992.

Approximation des structures de données

I — Données numériques



Référence

[1] P. Cousot. Semantic foundations of program analysis. In S.S. Muchnick and N.D. Jones, éditeurs, *Program Flow Analysis: Theory and Applications*, chapitre 10, pages 303–342. Prentice-Hall, Inc., Englewood Cliffs, NJ, USA, 1981.

Exemple : invariants linéaires [1]

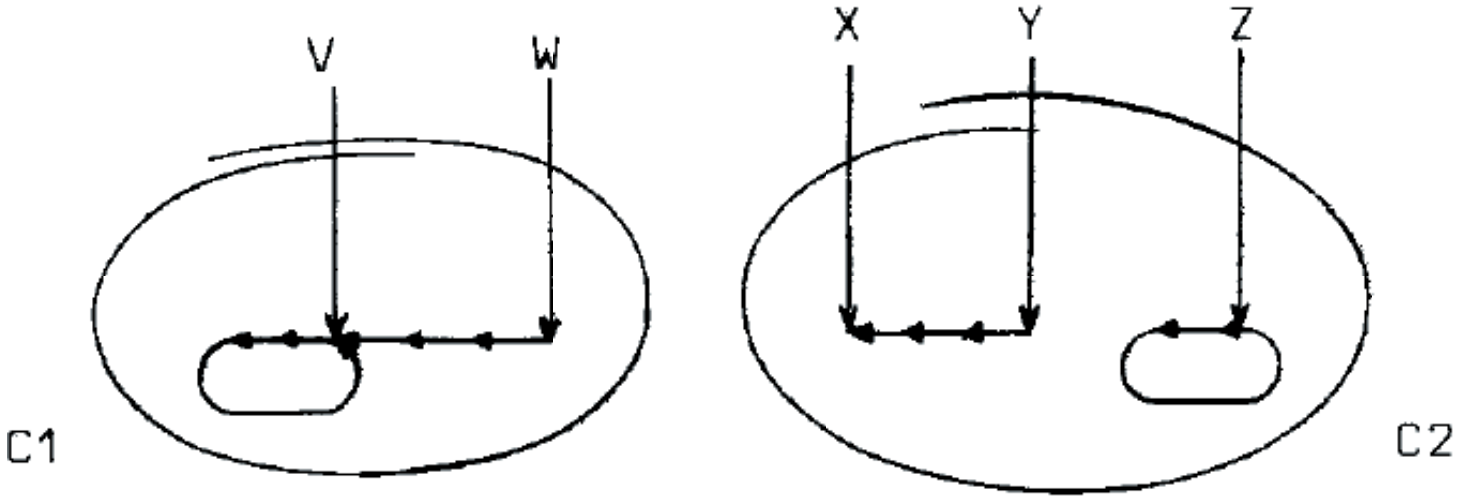
```
      procedure BUBBLESORT(integer value N; integer array[1:N] K);
      begin integer B,J,T;
       B:=N;
{1}    while B≥1 do
{2}     J:=1; T:=0;
{3}     while J≤(B-1) do
{4}      if K[J]>K[J+1] then
{5}       EXCHANGE(J,J+1); {no side effects on N,B,J,
{6}       T:=J,
{7}      fi;
{8}      J:=J+1;
{9}     od;
{10}    if T=0 then return fi;
{11}    B:=T;
{12}   od;
{13}  end;
```

{1}	: B=N
{2}	: 1≤B≤N
{3}	: 1≤B≤N, J=1, T=0
{4},{5},{6}	: B≤N, T≥0, T+1≤J, J+1≤B
{7}	: B≤N, J≥1, J+1≤B, J=T
{8}	: B≤N, J+1≤B, J≥1, T≥0, T≤J
{9}	: B≤N, J≤B, J≥2, T≥0, T+1≤J
{10},{11}	: B≤N, J≤B, T≥0, T+1≤J, B≤J+1
{12}	: J≤N, T≥0, T+1≤J, T=B
{13}	: B≤N, B≤1

Référence

[1] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY, USA.

Approximation des structures de données
II — Données symboliques [1]



Référence

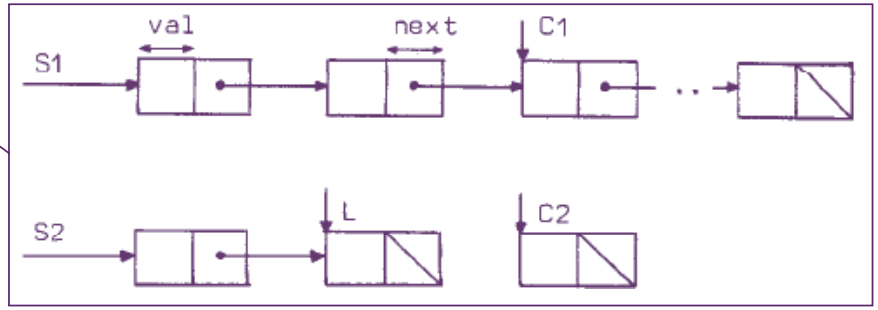
[1] P. Cousot and R. Cousot. Static determination of dynamic properties of generalized type unions. In *ACM Symposium on Language Design for Reliable Software*, Raleigh, NC, USA, ACM SIGPLAN Notices 12(3):77–94, 1977.

Exemple

```
procedure copy (S1 : list; var S2 : list);
   var C1, C2, L : list;
begin
   {S1,S2 / C1,C2,L}
   C1 := S1; S2 := nil; L := nil;
   {S1,C1/S2/C2/L}
   while C1 <> nil do
     begin
       {P2}
       new(C2); C2↑.val := C1↑.val; C2↑.next := nil;
       {S1,C1/S2,L/C2}
       if L = nil then
          {P4}
          S2 := C2
          {S1,C1/S2,C2/L}
       else
          {S1,C1/S2,L/C2}
          L↑.next := C2  {P7};
       {S1,C1/S2,L,C2}
       L := C2; C1 := C1.next;
       {S1,C1/S2,C2,L}
     end
   {C1/S1/S2,C2,L}
```

Pr condition : pas de partage entre param tres et variables locales

```
P1 = extract(L,extract(S2,extract(C1,P0)
P2 = P1 ⊔̄ P9                          ⊔̄ {C1,S1}))
P3 = extract(C2,P2)
P4 = extract(L,P3)
P5 = extract(S2, P4) ⊔̄ {S2, C2}
P6 = P3
P7 = P6 ⊔̄ {L,C2}
P8 = P5 ⊔̄ P7
P9 = extract(L,P8) ⊔̄ {L,C2}
P10 = extract(C1,P1 ⊔̄ P9)
```



Postcondition : pas de partage entre param tres

Approximation des structures de contrôle

– Procédures récursives [1] ;
– Programmes parallèles, processus statiques partageant des données communes [2] ;
– Programmes distribués, processus statiques communiquant par messages (synchrones) [3].

Références

[1] P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. In E.J. Neuhold, éditeur, *IFIP Conference on Formal Description of Programming Concepts, St-Andrews, N.B., Canada*, pages 237–277. North-Holland Pub. Co., Amsterdam, Pays-Bas, 1977.

[2] P. Cousot and R. Cousot. Invariance proof methods and analysis techniques for parallel programs. In A.W. Biermann, G. Guiho, and Y. Kodratoff, éditeurs, *Automatic Program Construction Techniques*, chapitre 12, pages 243–271. Macmillan, New York, NY, USA, 1984.

[3] P. Cousot and R. Cousot. Semantic analysis of communicating sequential processes. In J.W. de Bakker and J. van Leeuwen, éditeurs, *Seventh International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 85, pages 119–133. Springer-Verlag, juillet 1980.

Approximation des structures de contrôle (suite)

– Programmes fonctionnels [4] ;

– Programmes logiques [5] ;

Références

[4] P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages), invited paper. In *Proceedings of the 1994 International Conference on Computer Languages*, pages 95–112, Toulouse, 16–19 May 1994. IEEE Computer Society Press, Los Alamitos, CA, USA.

[5] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992. (L'éditeur du « Journal of Logic Programming » a publié par erreur les épreuves non relues et illisibles. Pour une version correcte de ce papier, voir http://www.di.ens.fr/~cousot.).

Les méthodes d'analyse de programmes sont des interprétations abstraites

– Contraintes [1] ;
– Typage [2] ;
– Model checking [3] ;
– etc.

Références

[1] P. Cousot and R. Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *Proceedings of the Seventh ACM Conference on Functional Programming Languages and Computer Architecture*, pages 170–181, La Jolla, Californie, 25–28 June 1995. ACM Press, New York, NY, USA.

[2] P. Cousot. Types as abstract interpretations, invited paper. In *Conference Record of the Twentyfourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 316–331, Paris, janvier 1997. ACM Press, New York, NY, USA.

[3] P. Cousot and R. Cousot. Parallel combination of abstract interpretation and model-based automatic analysis of software. In R. Cleaveland and D. Jackson, éditeurs, *Proceedings of the First ACM SIGPLAN Workshop on Automatic Analysis of Software, AAS '97*, pages 91–98, Paris, janvier 1997. ACM Press, New York, NY, USA.

Les sémantiques sont des interprétations abstraites

– Toute sémantique décrit une approximation du comportement des programmes dans tous les environnements d'exécution possible [1 , 2] ;

– La seule différence est le niveau d'abstraction et la méthode de présentation [3].

Références

[1] P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In *Conference Record of the Ninthteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 83–94, Albuquerque, New Mexico, 1992. ACM Press, New York, NY, USA.

[2] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. À paraître dans *Theoretical Computer Science*.

[3] P. Cousot and R. Cousot. Compositional and inductive semantic definitions in fixpoint, equational, constraint, closure-condition, rule-based and game-theoretic form, invited paper. In P. Wolper, éditeur, *Proceedings of the Seventh International Conference on Computer Aided Verification, CAV '95*, Liège, Belgique, Lecture Notes in Computer Science 939, pages 293–308. Springer-Verlag, 3–5 juillet 1995.

Exemple

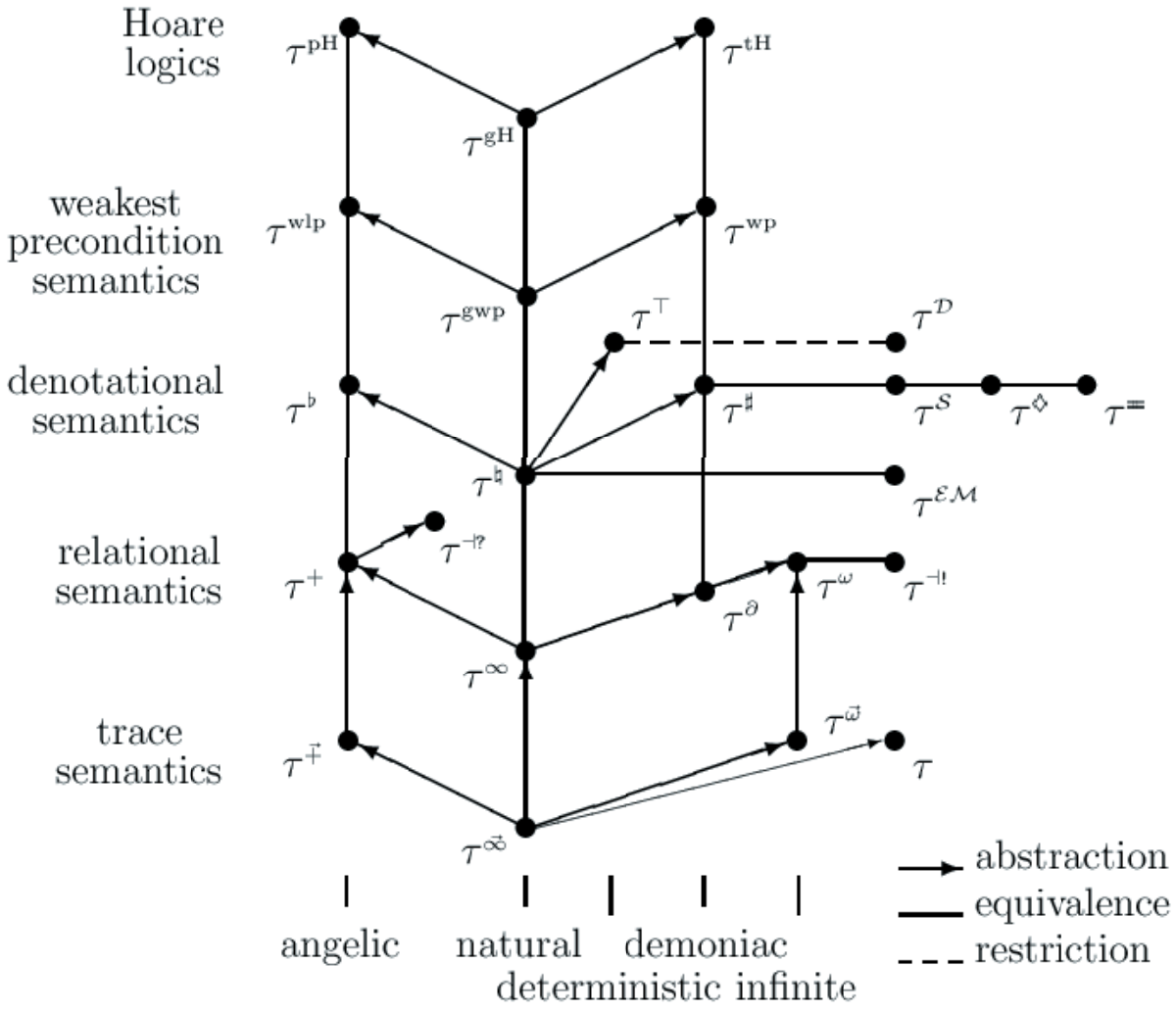


Figure 5. The hierarchy of semantics

Le point de vue unificateur de l'interprétation abstraite

– Il semble utile de comprendre que les centaines de publications sur l'analyse statique de programmes se ramènent à quelques principes simples [1] ;
– Ce point de vue n'est pas toujours compris (une théorie générale ne peut résoudre à priori tous les problèmes pratiques) ;
– Cependant, une structuration est nécessaire pour éviter l'oubli et la redécouverte, quitte à faire évoluer les fondements [2].

Références

[1] P. Cousot. Program analysis: The abstract interpretation perspective. *ACM Computing Surveys*, 28A(4es):165–es, décembre 1996.
[2] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, août 1992.

2. Défis pour l'interprétation abstraite

Passage à l'échelle

– programmes embarqués de commande de vol d'un avion : 100 000 lignes ;
– éditeur de texte grand public : 1 400 000 lignes ;
– système d'exploitation professionnel : 30 000 000 lignes [4] ;

$\Longrightarrow$ La sûreté de fonctionnement des logiciels critiques va devenir un enjeu majeur pour la société ;

$\Longrightarrow$ La contribution de l'analyse statique doit être importante.

4. et 63 000 erreurs (connues) !

Domaines abstraits numériques
— Du linéaire au non-linéaire

– L'analyse de polyèdres convexes [1], bien que complexe, a obtenu de nombreux succès (notamment pour le « model-checking » de systèmes infinis) ;

⟹ Les domaines abstraits numériques non linéaires restent pour l'instant largement inexplorés.

Référence

[1] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY, USA.

Domaines abstraits symboliques
— Vers des domaines d'usage général

– pour les propriétés numériques, on dispose d'une hiérarchie de domaines abstraits d'usage général, qui est bien pratique pour approcher des ensembles de vecteurs de nombres ;

$\Longrightarrow$ pour les propriétés symboliques, les domaines abstraits généraux sont peu développés.

Quelles propriétés analyser ?

– Les premières propriétés considérées pour l'analyse statique étaient des propriétés d'invariance (voire d'inévitabilité, mais simples [1]) ;

$\Longrightarrow$ Il faut pouvoir considérer des propriétés temporelles complexes [2], dans un cadre général de systèmes infinis ;

– Les méthodes automatiques et approchées sont plus élaborées que les techniques de preuve manuelles ou énumératives.

Références

[1] P. Cousot. Semantic foundations of program analysis. In S.S. Muchnick and N.D. Jones, éditeurs, *Program Flow Analysis: Theory and Applications*, chapitre 10, pages 303–342. Prentice-Hall, Inc., Englewood Cliffs, NJ, USA, 1981.

[2] P. Cousot and R. Cousot. Temporal abstract interpretation. In *Conference Record of the Twentyseventh Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Boston, Massachusetts, janvier 2000. ACM Press, New York, NY, USA.

Automatisation de l'abstraction

– La conception d'analyseurs statiques est entièrement formalisée par l'interprétation abstraite ;
– Il doit donc être possible :
  – de vérifier que la sémantique abstraite est une abstraction correcte de la sémantique concrète,
  – d'extraire l'analyseur statique de cette preuve de correction,
  – de composer automatiquement des abstractions différentes,
  – d'améliorer automatiquement la précision de l'abstraction si nécessaire.

Quels modèles de programmes ?

– l'usage de systèmes de transitions pour modéliser l'analyse statique a conduit à des applications immédiates pour lesquelles elle n'était pas initialement prévue (par exemple, en programmation logique) ;

– les systèmes de transitions ne sont pas assez expressifs (récursivité, parallélisme, etc.) ;

$\Longrightarrow$ un modèle mathématique expressif et général serait utile (ordre supérieur, modularité, processus dynamiques, distribution, etc.).

Portée de l'analyse statique

– L'analyse statique est essentiellement limitée à un programme (de grande taille) écrit dans un langage de programmation de haut niveau ;

– Le sûreté du logiciel est un problème global. La portée de l'analyse statique doit être élargie :

$\Longrightarrow$ du langage de spécification au langage machine ;

$\Longrightarrow$ de l'environnement du système d'exploitation à Internet.

Conception des langages informatiques [1]

Il est peut-être plutôt préférable, d'envisager de nouveaux langages ou traits de langages orientés vers la résolution des problèmes d'analyse sémantique exacte ou approchée des programmes.

$\Longrightarrow$ Mis à part le typage, peu de progrès ont été réalisés dans la conception de langages.

Référence

[1] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, 21 mars 1978.

Une théorie complexité/précision

– la théorie de la complexité rend compte de la difficulté de faire réaliser une tâche par un ordinateur ;

– dans le cadre de méthodes approchées, il faut également mesurer l' intérêt de l'approximation ;

– on s'intéresse au rapport coût/bénéfice, pas au coût uniquement ;

– par exemple, il faut expliquer pourquoi certaines analyses peuvent être à la fois plus précises et moins coûteuses.

3. Perspectives pour l'interprétation abstraite

De la difficulté des prévisions [1]

- Les techniques de mise au point encore largement utilisées dans l'industrie informatique du logiciel peuvent être en partie évitées (du moins pour les fautes de programmation si ce n'est pour les fautes de conception), en utilisant nos propositions d'analyse sémantique automatique des programmes, et ce, sans attendre les dix ans (ou plus) qui seront nécessaires pour que les techniques de vérification des programmes utilisant des démonstrateurs de théorèmes soient opérationnelles. Il est d'ailleurs certain que les méthodes que nous proposons sont complémentaires et offrent pour certains types d'analyses un rapport coût/bénéfice très rentable.

Référence

[1] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, 21 mars 1978.

Situation actuelle

– début d'industrialisation de l'analyse statique par interprétation abstraite [1 , 2] ;

– début d'utilisation industrielle de l'analyse statique lors du développement de logiciels critiques stratégiques ;

– la situation va être rapidement très concurrentielle ;

$\Longrightarrow$ la recherche aura un rôle essentiel pour devancer la concurrence.

Référence

[1] AbsInt Angewandte Informatik GmbH, http://www.absint.de/

[2] PolySpace Technologies, http://www.polyspace.com/

La recherche universitaire

La recherche universitaire française/européenne sera-t-elle à la hauteur du défi ?

- évolution très lente des moyens (bourses, postes) ;
- peu de capacité de formation (quelques thèses par an) ;
- grande dispersion (géographique, thématique) ;
- court terme (contrats, publications) ;
- très forte compétition internationale.

En conclusion ...

– L'industrialisation doit être accompagnée par une forte action de recherche à moyen et long terme ;

– L'objectif est d'anticiper les problèmes de fiabilité résultant du formidable développement des logiciels critiques dans les prochaines années.