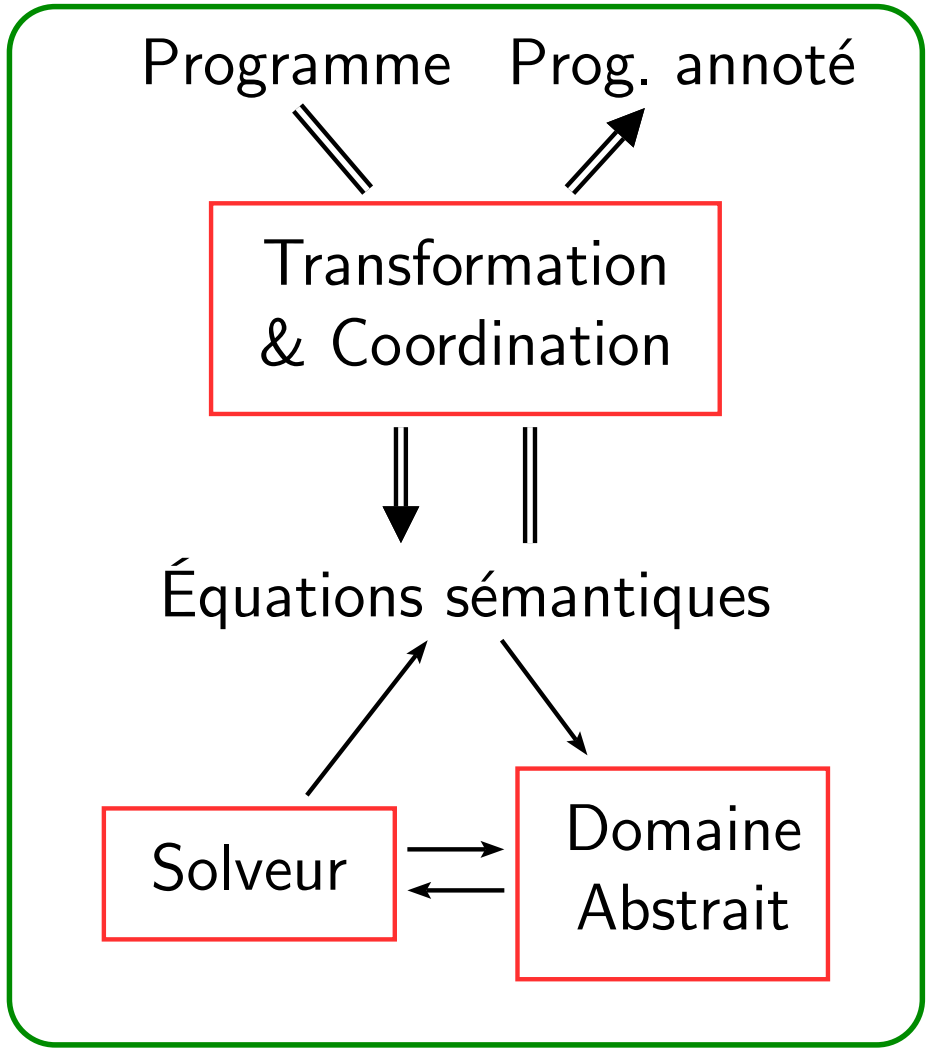
La bibliothèque FIXPOINT

Bertrand Jeannet

INRIA Rhône-Alpes

7 juin 2010

Structure d'un analyseur statique



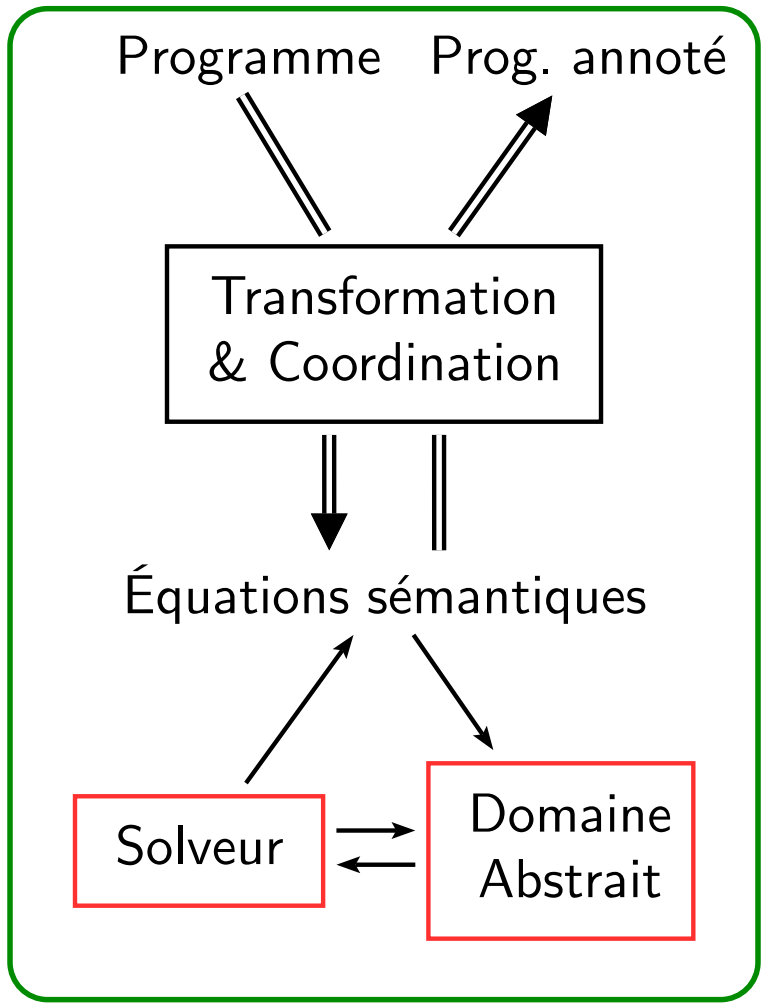
Modularité d'un analyseur statique

Objectif idéal

Chaque composant : librairie indépendante

- API bien identifiée
- Pas de dépendances implicites

$\Longrightarrow$ Composants substituables



En pratique :

Dans quelle mesure est-on robuste à

- Nouveau langage d'entrée ?
- Nouveau type de donnée ?
- Nouveau domaine abstrait ?

"Un composant est réutilisable quand il a été réutilisé au moins deux/trois fois" Anonyme, XX$^{\text{ième}}$ siècle

Solveur

Domaine abstrait partitionné

$$A = A_1 \times \ldots \times A_n$$

Équation de point fixe

$$X_k = F_k^1(\vec{X}) \sqcup F_k^2(\vec{X}) \sqcup F_k^n(\vec{X}) \ldots \ , \ X_k \in A_k$$

But

Calculer une (plus) petite solution $X = (X_1, \ldots, X_n)$ en temps fini.

Solveur : contextes d'utilisation I

Intraprocédural

- Système d'équations = graphe (isomorphe au CFG)
- Fonctions ne dépendent que d'une variables :
  $X_k = F_k^1(X_1) \sqcup F_k^2(X_2) \sqcup F_k^n(X_n) \ldots$
- Graphe plus ou moins structuré :
  - \+ Programme impératif sans goto
  - – Programme synchrone (LUSTRE partitionné

Solveur : contextes d'utilisation II

Interprocédural

- Fonctions pouvant dépendre de plusieurs variables
  - Retour de procédure $X^{\text{caller}}_{\text{ret}} = F(X^{\text{caller}}_{\text{call}}, X^{\text{callee}}_{\text{exit}})$
- Système d'équations = hypergraphe
- (Hyper)graphe moins structuré
  - Cycles : induits par boucles et récursion
  - Appel récursif dans une boucle $\Longrightarrow$ plat de spagettis

Solveur : contextes d'utilisation III

Concurrent

- Sémantique par entrelacement
- CFG global : produit cartésien des CFGs de chaque thread
- Synchronisations : seule une (petite) partie du CFG est accessible
- Nécessité d'explorer "à la volée" le CFG global

Pourquoi le solveur Fixpoint ?

Limitations des solveurs existants :

1. Souvent intégrés dans un analyseur, ne sont pas conçu en tant que librairie indépendante.
2. Souvent orientés analyse flôt de donnée, type bitvector :
   - $\boldsymbol{A}$ est un treillis fini simple, $\boldsymbol{F}$ est de faible complexité
   - Oriente les choix algorithmiques
3. Ne traitent pas le cas où $\boldsymbol{A}$ ne satisfait pas la condition de chaine ascendante finie, cas qui requiert
   - Des opérateurs d'accélération ou d'élargissement
   - Des stratégies d'itération spécifiques
4. Souvent restreints à des équations intraprocédurales (bornes supérieures de fonctions à un argument)
   - Insuffisant pour certaines techniques d'analyse interprocédurales

Fixpoint en bref

1. Librairie OCaml
   - Exploitant polymorphisme et ordre supérieure pour "interpréter" un système d'équation en fonction de $\boldsymbol{A}$ et $\boldsymbol{F}$
   - 1600 LOC (+1000 LOC pour le module hypergraphe)
2. Choix algorithmiques adaptés plutôt à des domaines complexes
   (ex : combinaison de stratégies à la Bourdoncle avec techniques paresseuses type working set)
3. Stratégies d'itération avec élargissement (paramétrable par l'analyseur client)
4. Équations générales
   $\boldsymbol{X_k = F_k^1(X_1, \ldots, X_n) \sqcup F_k^2(X_1, \ldots, X_n) \sqcup \ldots}$
   (bornes supérieures de fonctions à plusieurs arguments)
   Les fonctions sont supposées strictes
   $\boldsymbol{(X_i = \bot \implies F_k(\ldots X_i \ldots) = \bot)}$

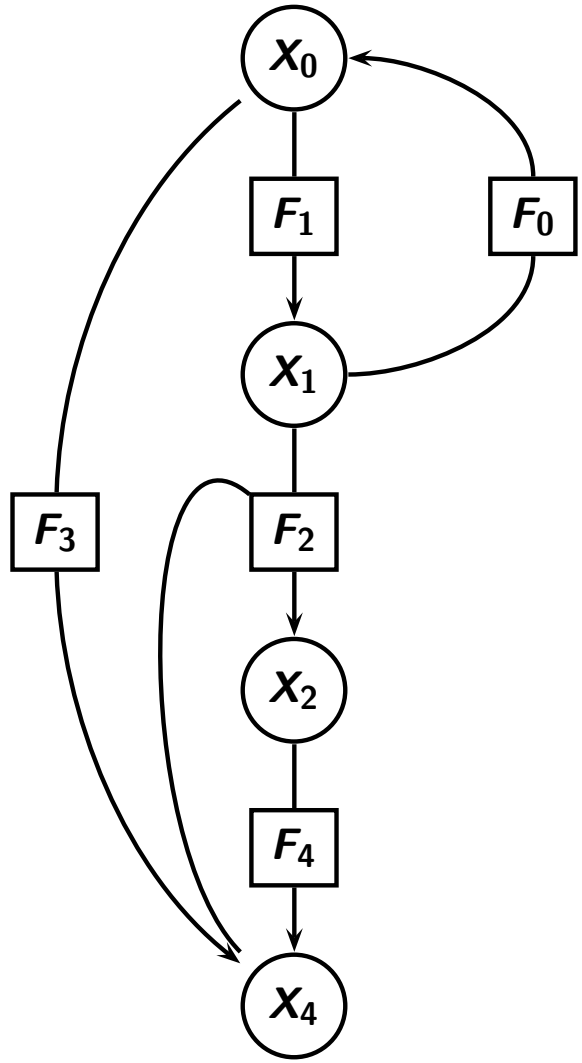
Outline

Système d'équations

$$
\begin{aligned}
X_0 &= X_1^0 \sqcup F_0(X_1) \\
X_1 &= F_1(X_0) \\
X_2 &= F_2(X_1, X_3) \\
X_3 &= F_3(X_0) \sqcup F_4(X_2)
\end{aligned}
$$

Représenté par un *hyper-graphe* :

- Sommets : les inconnues X_0, X_1, X_2, X_3
- Hyperarcs : fonctions F_0, F_1, F_2, F_3, F_4



Système d'équation

- Hypergraphe avec identificateurs de variables (X_i) et de fonctions (F_k) OU
  Fonction d'exploration

  ```
  'vertex -> ('hedge, 'vertex array * 'vertex) list
  ```

  qui associe à un sommet une liste d'éléments (hyperarc sortant, ens. des sommets origines, sommet destination)
- Une structure polymorphe pour les opérations sur A :

  ```
  type ('vertex,'abs) ops = {
    is_bottom : 'vertex -> 'abs -> bool
    is_leq : 'vertex -> 'abs -> 'abs -> bool
    join : 'vertex -> 'abs -> 'abs -> 'abs ...
  }
  ```

- Une fonction apply pour les fonctions F_i :

  ```
  type ('hedge,'abs,'arc) apply =
    'hedge -> 'abs array -> 'abs * 'arc
  ```

Stratégie d'itération à la Bourdoncle

Liste imbriquée `(x y (z w) ((a b) c) d)` dont les éléments ont le type

```
type ('vertex,'hedge) strategy_vertex = {
  vertex : 'vertex;      (* sommet à calculer *)
 hedges : 'hedge list; (* liste d'hyperarcs entrants *)
  widen : bool;          (* élargissement ? *)
}
```

Fonction paramétrée pour générer automatiquement une stratégie d'itération à partir d'un (hyper)graphe

```
val make_strategy_default :
  ?depth:int ->
  ?priority:'hedge PSHGraph.priority ->
  ('vertex, 'hedge, 'e, 'f, 'g) PSHGraph.t -> 'vertex PSette
  ('vertex, 'hedge) strategy
```

Outline

Module standard

- Prend en entrée un hypergraphe
- Calcule un post-point-fixe
- Puis un nombre spécifié d'itérations descendantes

Aspects algorithmiques

- Associe à chaque variable une valeur abstraite
  - Pas d'optimisation en espace pour les gros systèmes creux
- Itération de post-point-fixe guidée par une stratégie à la Bourdoncle
- Combinée à une technique de working set pour éviter les recalculs inutiles
- Idem pour la séquence descendate (avec une stratégie aplatie)

Module “guidé”

Implante “Guided Static Analysis” de Gopan et Reps
Alterne séquence d’itérations ascendantes et descendantes

1. On propage les valeurs initiales (une itération)
2. On restreint le graphe aux hyperarcs actifs (retournant une valeur non vide)
3. On procède à une analyse standard sur ce sous-graphe (séquence ascendante puis descendante)

Et on recommence
cf demo

Module “dynamique”

Équations décrites sous forme implicite par une fonction d'exploration

```
'vertex -> ('hedge, 'vertex array * 'vertex) list
```

qui associe à un sommet une liste d'éléments
(hyperarc sortant, ens. des sommets origines, sommet destination)

Motivation

Analyse de programme concurrents :

- On calcule le produit de processus à la volée
- On n'explore que quand les prédécesseurs d'une variable sont non vide.

Plus généralement : quand on sait que seule une petite fraction des variables sera non vide

Module “dynamique” : algorithme

1. On propage les valeurs initiales en explorant les variables successeurs des variables non vides
   (1 itération)
   On obtient un sous-graphe
2. Deux options :
   - Option guidée : on retire les variables et hyperarcs inactifs ;
   - Option standard : on les garde
3. On procède à une analyse du sous-graphe
4. Otion guidée : itérations ascendantes puis descendantes
5. Option standard : pas d’itération descendante

Et on recommence.

Fixpoint et généricité I

- Systèmes d'équations généraux, sur des domaines pouvant être équipés d'élargissement
- Représentation des équations : explicite ou implicite
- Résolution par la méthode de Kleene
  - Guidé par une stratégie d'itération à la Bourdoncle (potentiellement récursif)
  - Stratégie paramétrable par l'utilisateur
- Convient à quantité d'analyseurs : NBac, Interproc, ConcurInterproc, analyseurs de copains (3 thésards de Nicolas Halbwachs, 2 thésards à moi)

Généricité plus facile à obtenir que pour le domaine abstrait
(invariance versus sémantique concrète (et domaine abstrait))

Fixpoint et généricité II

Mais

- Pas de notion d'entrée/sortie de boucle (dans le solveur)
  Utile pour
  - Dépliage dynamique de boucles (meilleure précision)
  - Gestion mémoire adaptée aux gros systèmes
- Restreint aux techniques de résolutions par itérations

Techniques alternatives explorées dans le projet ASOPT I

Analyse Statique et OPTimisation : INRIA (S. Gaubert et moi), Verilag (D. Monniaux et N. Halbwachs), CEA LMeASI (Goubault, Putot, Vedrine), EADS (trX. Allamigeon)

Approche directe par optimisation

On résoud $\boldsymbol{X = \mathrm{lfp}(F)}$ directement en transformant en problème d'optimisation

- Plus d'approximations dues à l'élargissement
- Envisageable pour les intervalles ou les octogones (analyse infère un nb fini de bornes)
- Expressivité de $\boldsymbol{F}$ limitée
- Domaine abstrait : générateur d'équations dans $\mathbb{R}^{\boldsymbol{n}}$

Itération sur les politiques

Techniques alternatives explorées dans le projet ASOPT II

Idée :

1. On simplifie $\boldsymbol{X = F(X)}$ par une politique π
2. On résoud $\boldsymbol{X = F_\pi(X)}$
   A. par optimisation ($\boldsymbol{F_\pi}$ plus simple que $\boldsymbol{GF}$)
   B. par itération de Kleene avec élargissement
3. Si $\boldsymbol{X_\pi = F(X_\pi)}$, on est content ($\boldsymbol{X_\pi \sqsupseteq F(X_\pi)}$)
4. Sinon ; on modifie π et on itère