Static Single Assignment Form

Paul Feautrier

Paul.Feautrier@ens-lyon.fr.

ENS Lyon

Plan

- Concepts de base
- Transformation en SSA
- Utilisation de la SSA
- Elimination des fonctions ϕ
- Perspectives

Bibliographie

- Muchnick : Advanced Compiler Design and Implementation, Chap. 8
- Cytron ?

Concepts de base, I

- Qu’est-ce qu’une variable :
  - en mathématique, c’est une constante dont la valeur est inconnue. La valeur d’une « variable » ne change pas au cours d’un raisonnement.
  - en informatique, c’est la désignation symbolique d’une cellule de mémoire. La valeur d’une variable change à chaque affectation.
- Si l’on veut appliquer le raisonnement mathématique à un programme, il faut s’intéresser aux valeurs contenues dans les cellules de mémoire : on peut dire que ces valeurs sont des fonctions du temps.

Concepts de base, II

- En informatique, le temps est discret. La valeur d'une variable ne change que si on exécute une affectation. On préfère donc représenter une variable par une suite que par une fonction.
- Un programme est en assignation unique dynamique si chaque cellule de mémoire n'est affectée qu'une fois au cours de l'exécution. Il y a alors identité entre cellule de mémoire et valeur. En contrepartie, la complexité spatiale est égale à la complexité temporelle.
- Un programme est en assignation unique statique si chaque cellule de mémoire n'est affectée qu'une fois dans le texte du programme.
- On ne peut identifier variable et valeur que dans des fragments de programme où chaque instruction n'est exécutée qu'une fois.

Concepts de base, III

- Bloc de base : un sous graphe du graphe de contrôle formant une chaîne, et n'ayant qu'un point d'entrée et un point de sortie.
- Superbloc : un sous graphe du graphe de contrôle formant un DAG, et n'ayant qu'un point d'entrée et un point de sortie.
- *Reaching definition* : soit D une instruction qui définit une variable x et U une instruction qui l'utilise. Il y a une dépendance de flot entre D et U si il existe un chemin de D à U sur lequel x n'est pas modifiée.

Transformation en assignation unique

- Les programmes courants sont rarement écrits en assignation unique.
- Comment transformer un programme ordinaire en programme en assignation unique ?
- On ne considère ici que les variables scalaires.
- Deux étapes :
  - On indexe la variable à gauche du signe d'affectation pour obtenir la propriété d'assignation unique.
  - Pour chaque variable x à droite du signe d'affectation, on détermine les *reaching definitions* $x_a, \ldots x_z$. On remplace x par $\phi(x_a, \ldots, x_z)$.

Un exemple

```
/* calcul du pgcd */
 u = a;
 v = b;
L :
 r = u % v;
 if(r== 0) return v;
 u = v;
 v = r;
 goto L;
```

```
 u_1 = a;
 v_1 = b;
L :
 r_1 = u % v;
 if(r == 0) return v;
 u_2 = v;
 v_2 = r;
 goto L;
```

- Quelle est la source de u au moment du calcul du modulo ?
- u peut provenir de l'initialisation (u_1) ou du corps de la boucle à l'itération précédente (u_2).
- On écrit par convention $u = \phi(u_1, u_2)$.

Les fonctions ϕ

- Une fonction ϕ fonctionne comme un multiplexeur.
- Elle sélectionne la « bonne » valeur parmi ses entrées et la retourne sans la modifier.
- La sélection dépend du chemin suivi dans le graphe du programme au moment du calcul de la fonction ϕ.

$u_1 = a$
$v_1 = b$
L : $r_1 = \phi(u_1, u_2) \bmod \phi(v_1, v_2)$
If $r_1 = 0$ return $\phi(v_1, v_2)$
$u_2 = \phi(v_1, v_2)$
$v_2 = r_1$
goto L

Optimisation

On peut toujours optimiser en déplaçant les fonctions ϕ vers les points d'entrée des blocs de base.

$u_1 = a$
$v_1 = b$
L : $u_3 = \phi(u_1, u_2)$
$v_3 = \phi(v_1, v_2)$
$r_1 = u_3 \bmod v_3$
If $r_1 = 0$ return v_3
$u_2 = v_3$
$v_2 = r_1$
goto L

Quel est l'intérêt de la forme SSA ?

Théorème 1 *Au cours d'une exécution d'un superbloc, deux expressions identiques ont la même valeur.*

On suppose que l'exécution du programmee est déterministe, par exemple qu'il n'y a pas de tirage d'un vrai nombre aléatoire. Au cours de l'exécution d'un superbloc, le contrôle suit donc un chemin bien déterminé dans le graphe de contrôle. Soit e_1 et e_2 les expressions distinguées. Soit v une variable de e_1 et e_2. Si v avait été modifiée sur le chemin qui va de e_1 à e_2, le mécanisme de mise en SSA aurait changé son nom, et les deux expressions ne seraient plus identiques. ∎

Attention, la réciproque est fausse. Deux expressions peuvent être égales sans être identiques.

Applications, I

Dans un programme en forme SSA, et à l'intérieur d'un bloc de base ou d'un superbloc, on peut raisonner sur les variables informatiques comme si c'était des variables mathématiques.

- Détection des calculs redondants. Si dans un bloc de base on trouve deux expressions ou sous-expressions identiques, on peut réutiliser la valeur de l'une à la place de l'autre.

```
                                    u = a + b;
x = (a + b) + c;                    x = u + c;
                        ==>
y = (a + b) + d;                    y = u + d;
```

- Propagation des constantes.

```
x = 12.;                            x = 12.;
                        ==>
y = x;                              y = 12.;
```

Application, II

Contrôle des tailles de tableaux.

```
real a(n), b(n);
allocate (a);
a = c(1:n);
allocate (b);
b = a;
```

Ce programme est déja en forme SSA.
Donc la deuxième affectation est valide.

```
real a(n), b(n);
allocate (a);
a = c(1:n);
n = n + 1;
allocate (b);
b = a;
```

==>

```
real a(n), b(n);
allocate (a);    //taille n_1
a = c(1:n_1);
n_2 = n_1 + 1;
allocate (b);    //taille n_2
b = a;
```

Le programme initial n'est pas en SSA. Une fois la transformation réalisée, il est visible que la dernière affectation n'est pas correcte.

Elimination des fonctions ϕ

- La forme SSA est un outil d'optimisation. Il n'est pas question de la compiler et de l'exécuter. Il faut donc éliminer les fonctions ϕ.
- Il est possible de remplacer une fonction ϕ par une expression conditionnelle (*gated SSA*). Exemple du PGCD :

$u_1 = a$
$v_1 = b$
`i = 0;`
L : $u_3 = \text{if } i = 0 \text{ then } u_1 \text{ else } u_2$
$v_3 = \text{if } i = 0 \text{ then } v_1 \text{ else } v_2$
$r_1 = u_3 \bmod v_3$
If $r_1 = 0$ return v_3
$u_2 = v_3$
$v_2 = r_1$
`i++;`
goto L

Elimination des fonctions ϕ, II

- L'autre méthode consiste à déplacer les fonctions ϕ jusqu'en un point où l'ambiguité est levée.
- Sur l'exemple du PGCD, on déplace la fonction ϕ avant le point de jonction.
- Comme le point de jonction a deux prédécesseurs, la fonction doit être dupliquée.
- Dans l'initialisation, $\phi(v_1, v_2) = v_1$.
- A la fin du corps de boucle, $\phi(v_1, v_2) = v_2$.
- On retrouve le programme initial à quelques duplications de variables près.

Remarques finales

- Il est impossible – sauf cas particuliers – de traiter de la même façon tableaux et pointeurs.
- Il existe de nombreux travaux sur le placement optimal des fonctions ϕ.
- La forme SSA est une autre façon d'exprimer l'information contenue dans les *reaching definitions*.
- Des informations analogues – parfois plus précises – peuvent être obtenues par la méthode de l'exécution symbolique.
- Il existe une autre forme (Dynamic Single Assignment Form) plus précise, mais qui ne peut être calculée que pour certains programmes.