Compilation

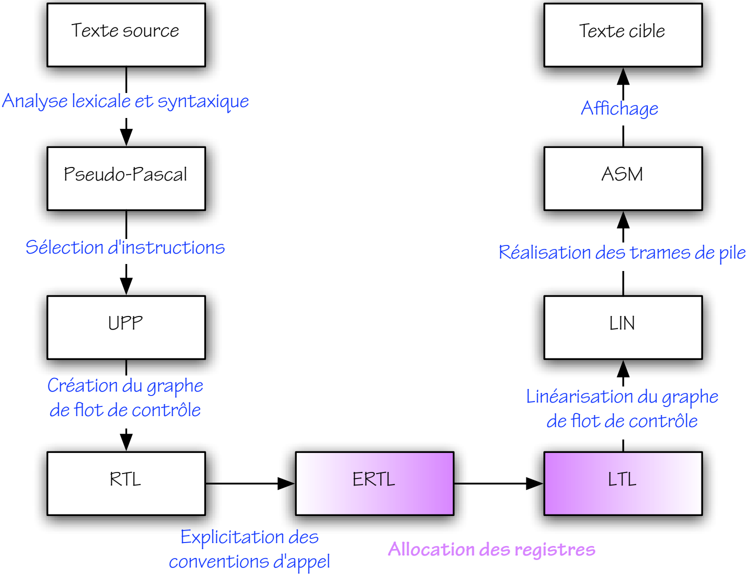
Cours n°6: Analyse de durée de vie
Construction du graphe d'interférences

Sandrine Blazy
(d'après le cours de François Pottier)

ensiie - 2^e année

1^{er} décembre 2008

école nationale supérieure d'informatique pour l'industrie et l'entreprise


Texte source
Analyse lexicale et syntaxique
Pseudo-Pascal
Sélection d'instructions
UPP
Création du graphe
de flot de contrôle
RTL
ERTL
LTL
Explicitation des
conventions d'appel
Allocation des registres
Linéarisation du graphe
de flot de contrôle
LIN
Réalisation des trames de pile
ASM
Affichage
Texte cible

ensiie

La fonction factorielle en ERTL

Comment *réaliser* les pseudo-registres %3, %0, %4, et %1 ?

```
procedure f(1)
var %0, %1, %2, %3, %4, %5, %6
entry f11
f11 : newframe              → f10
f10 : move    %6, $ra       → f9
f9  : move    %5, $s1       → f8
f8  : move    %4, $s0       → f7
f7  : move    %0, $a0       → f6
f6  : li      %1, 0         → f5
f5  : blez    %0            → f4 , f3
f3  : addiu   %3, %0, −1    → f2
f2  : j                     → f20
f20 : move    $a0, %3       → f19
f19 : call    f(1)          → f18
f18 : move    %2, $v0       → f1
f1  : mul     %1, %0, %2    → f0
f0  : j                     → f17
f17 : move    $v0, %1       → f16
f16 : move    $ra , %6      → f15
f15 : move    $s1 , %5      → f14
f14 : move    $s0 , %4      → f13
f13 : delframe              → f12
f12 : jr      $ra
f4  : li      %1, 1         → f0
```

Durée de vie et interférence

Cette réflexion doit avoir illustré trois points :

- il faut savoir en quels points du code chaque variable est *vivante* – contient une valeur susceptible d'être utilisée à l'avenir ;
- deux variables peuvent être réalisées par le même emplacement *si elles n'interfèrent pas* – si l'on n'écrit pas dans l'une tandis que l'autre est vivante.
- les instructions **move** suggèrent des emplacements préférentiels.

J'écris *« variable »* pour *« pseudo-registre ou registre physique »*.

La phase d'allocation de registres doit donc être précédée d'une *analyse de durée de vie*, d'où on déduit un *graphe d'interférences*.

Analyse de la fonction factorielle

Voici les résultats de l'analyse de durée de vie :

```
procedure f(1)
var %0, %1, %2, %3, %4, %5, %6
entry f11
f11: newframe           → f10      $a0, $s0, $s1, $ra
f10: move  %6, $ra      → f9       %6, $a0, $s0, $s1
f9 : move  %5, $s1      → f8       %5, %6, $a0, $s0
f8 : move  %4, $s0      → f7       %4, %5, %6, $a0
f7 : move  %0, $a0      → f6       %0, %4, %5, %6
f6 : li    %1, 0        → f5       %0, %4, %5, %6
f5 : blez  %0           → f4, f3   %0, %4, %5, %6
f3 : addiu %3, %0, −1   → f2       %0, %3, %4, %5, %6
f2 : j                  → f20      %0, %3, %4, %5, %6
f20: move  $a0, %3      → f19      %0, %4, %5, %6, $a0
f19: call  f(1)         → f18      %0, %4, %5, %6, $v0
f18: move  %2, $v0      → f1       %0, %2, %4, %5, %6
f1 : mul   %1, %0, %2   → f0       %1, %4, %5, %6
f0 : j                  → f17      %1, %4, %5, %6
f17: move  $v0, %1      → f16      %4, %5, %6, $v0
f16: move  $ra, %6      → f15      %4, %5, $v0, $ra
f15: move  $s1, %5      → f14      %4, $v0, $s1, $ra
f14: move  $s0, %4      → f13      $v0, $s0, $s1, $ra
f13: delframe           → f12      $v0, $s0, $s1, $ra
f12: jr    $ra
f4 : li    %1, 1        → f0       %1, %4, %5, %6
```

ensiie

1 Analyse de durée de vie

Analyses de flot de données

Graphe d'interférences

Élimination du code mort

Vivants et morts

Voici une définition légèrement informelle :

> *Une variable v est vivante (« live ») au point ℓ s'il existe un chemin menant de ℓ à un point ℓ' où v est utilisée et si v n'est pas définie le long de ce chemin.*

Une variable est *morte* (« dead ») lorsqu'elle n'est pas vivante.

Approximation

L'analyse de durée de vie est *approximative :* on vérifie s'il *existe* un chemin menant à un site d'utilisation, mais on ne se demande pas dans quelles conditions ce chemin est *effectivement emprunté.*

De ce fait, « vivante » signifie *« potentiellement vivante »* et « morte » signifie *« certainement morte »*.

Cette approximation est *sûre*. Au pire, si on suppose toutes les variables vivantes en tous points, on devra attribuer à chacune un emplacement physique distinct – un résultat inefficace mais correct.

« Naissance » d'une variable

Une variable v est *engendrée* par une instruction i si i *utilise* v, c'est-à-dire si i *lit* une valeur dans v.

Dans ce cas, v est *vivante au point qui précède* immédiatement i.

« Mort » d'une variable

Une variable v est *tuée* par une instruction i si i *définit* v, c'est-à-dire si i *écrit* une valeur dans v.

Dans ce cas, v est *morte au point qui suit* immédiatement i.

ensiie

« Vie » d'une variable

Si i n'engendre ni ne tue v, alors v est vivante immédiatement *avant* i si et seulement si elle est vivante immédiatement *après* i.

Une variable est vivante *après* i si et seulement si elle est vivante *avant* l'un quelconque des successeurs de i.

Mise en inéquations

Les assertions précédentes permettent d'exprimer le problème sous forme *d'inéquations ensemblistes*.

À chaque étiquette ℓ du graphe de flot de contrôle, on associe *deux* ensembles de variables :

- $\textit{vivantes}_{\textit{entrée}}(\ell)$ est l'ensemble des variables vivantes immédiatement *avant* l'instruction située au point ℓ ;
- $\textit{vivantes}_{\textit{sortie}}(\ell)$ est l'ensemble des variables vivantes immédiatement *après* l'instruction située au point ℓ.

Inéquations

Les inéquations qui définissent l'analyse sont :

$$\begin{array}{rll} & \text{vivantes}_{\text{entrée}}(\ell') & \\ \overset{?}{\subseteq} & \text{vivantes}_{\text{sortie}}(\ell) & \textit{si } \ell \rightarrow \ell' \end{array}$$

$$\begin{array}{rl} & (\text{vivantes}_{\text{sortie}}(\ell) \setminus \text{tuées}(\ell)) \cup \text{engendrées}(\ell) \\ \overset{?}{\subseteq} & \text{vivantes}_{\text{entrée}}(\ell) \end{array}$$

Toute solution de ces inéquations est sûre. La *plus petite* solution donne les meilleurs résultats.

Pourquoi pas $(\text{vivantes}_{\text{sortie}}(\ell) \cup \text{engendrées}(\ell)) \setminus \text{tuées}(\ell)$?

La recherche de la plus petite solution conduit à une analyse *en arrière :* la vivacité se propage *dans le sens inverse des arêtes* du graphe de flot de contrôle.

Inéquations

Les inéquations qui définissent l'analyse sont :

$$
\begin{array}{ll}
 & \text{vivantes}_{\text{entrée}}(\ell') \\
\subseteq & \text{vivantes}_{\text{sortie}}(\ell) \qquad \textit{si } \ell \rightarrow \ell' \\
 & \\
 & (\text{vivantes}_{\text{sortie}}(\ell) \setminus \text{tuées}(\ell)) \cup \text{engendrées}(\ell) \\
\subseteq & \text{vivantes}_{\text{entrée}}(\ell)
\end{array}
$$

Pourquoi pas $(\text{vivantes}_{\text{sortie}}(\ell) \setminus \text{tuées}(\ell)) \cup \text{engendrées}(\ell)$?

Toute solution de ces inéquations est sûre. La *plus petite* solution donne les meilleurs résultats.

La recherche de la plus petite solution conduit à une analyse *en arrière :* la vivacité se propage *dans le sens inverse des arêtes* du graphe de flot de contrôle.

Inéquations

Les inéquations qui définissent l'analyse sont :

$$\begin{array}{ll} & \text{vivantes}_{\text{entrée}}(\ell') \\ \subseteq & \text{vivantes}_{\text{sortie}}(\ell) \qquad\qquad \textit{si } \ell \to \ell' \end{array}$$

$$\begin{array}{ll} & (\text{vivantes}_{\text{sortie}}(\ell) \setminus \text{tuées}(\ell)) \cup \text{engendrées}(\ell) \\ \subseteq & \text{vivantes}_{\text{entrée}}(\ell) \end{array}$$

Toute solution de ces inéquations est sûre. La *plus petite* solution donne les meilleurs résultats.

La recherche de la plus petite solution conduit à une analyse *en arrière :* la vivacité se propage *dans le sens inverse des arêtes* du graphe de flot de contrôle.

Inéquations

Les inéquations qui définissent l'analyse sont :

$$
\begin{array}{ll}
 & \text{vivantes}_{\text{entrée}}(\ell') \\
\subseteq & \text{vivantes}_{\text{sortie}}(\ell) \qquad\qquad \textit{si } \ell \rightarrow \ell'
\end{array}
$$

$$
\begin{array}{ll}
 & (\text{vivantes}_{\text{sortie}}(\ell) \setminus \text{tuées}(\ell)) \cup \text{engendrées}(\ell) \\
\subseteq & \text{vivantes}_{\text{entrée}}(\ell)
\end{array}
$$

Pourquoi pas $(\text{vivantes}_{\text{sortie}}(\ell) \cup \text{engendrées}(\ell)) \setminus \text{tuées}(\ell)$?

Toute solution de ces inéquations est sûre. La *plus petite* solution donne les meilleurs résultats.

La recherche de la plus petite solution conduit à une analyse *en arrière :* la vivacité se propage *dans le sens inverse des arêtes* du graphe de flot de contrôle.

ensiie

Inéquations

Les inéquations qui définissent l'analyse sont :

$$
\begin{array}{ll}
 & \mathsf{vivantes}_{\mathsf{entrée}}(\ell') \\
\subseteq & \mathsf{vivantes}_{\mathsf{sortie}}(\ell) \qquad\qquad\qquad \textit{si } \ell \rightarrow \ell'
\end{array}
$$

$$
\begin{array}{ll}
 & (\mathsf{vivantes}_{\mathsf{sortie}}(\ell) \setminus \mathsf{tuées}(\ell)) \cup \mathsf{engendrées}(\ell) \\
\subseteq & \mathsf{vivantes}_{\mathsf{entrée}}(\ell)
\end{array}
$$

Toute solution de ces inéquations est sûre. La *plus petite* solution donne les meilleurs résultats.

La recherche de la plus petite solution conduit à une analyse *en arrière :* la vivacité se propage *dans le sens inverse des arêtes* du graphe de flot de contrôle.

Communication entre procédures

La communication entre procédures se fait à travers des registres physiques qui doivent donc être considérés comme *vivants :*

- l'instruction ICall *engendre* les registres physiques dédiés au passage d'arguments – un préfixe de \$a0-\$a3 – et *tue* tous les registres « caller-save » – à savoir \$a0-\$a3, \$v0, \$ra, \$t0-\$t9 ;
- l'instruction IReturn *engendre* non seulement \$ra, mais aussi le registre physique dédié au renvoi de résultat – \$v0 – et tous les registres « callee-save » – à savoir \$s0–\$s7.

Analyse de durée de vie

2 Analyses de flot de données

Graphe d'interférences

Élimination du code mort

ensiie

Analyse de flot de données

L'analyse de durée de vie est membre de la famille des *analyses de flot de données*. Ces analyses associent une *propriété* à chaque point du code.

En général, les propriétés sont *ordonnées ;* un système *d'inéquations* définit un ensemble de solutions *sûres ;* parmi les solutions sûres, la *plus petite* est celle recherchée.

Cette théorie classique date des années 1970. C'est un cas particulier d'une théorie plus générale, datant de la même époque et beaucoup développée depuis, intitulée « interprétation abstraite » (Cousot, 2000).

Exemples

Voici un échantillon des propriétés étudiées dans la littérature :

- quelles variables seront *potentiellement utilisées ?*
- quelles variables ont *une valeur connue* et laquelle ?
- quelles variables ont une valeur *appartenant à un intervalle connu* et lequel ?
- quelles sont les *relations affines connues* entre variables ?
- quelles variables sont *certainement égales ?*
- quelles variables sont *potentiellement égales ?*
- quelles expressions *seront certainement évaluées ?*
- quels points du code *ont certainement été atteints* auparavant ?
- ...

Le treillis des propriétés

L'ensemble $\mathcal{P}$ des propriétés doit être un *sup-demi-treillis :* il doit jouir

- d'un *ordre* $\sqsubseteq$;
- d'un élément *minimum* $\perp$ (« bottom ») ;
- d'une opération de *plus petite borne supérieure* (« join ») $\sqcup$.

On exige de plus que *toute suite croissante converge*, de façon à garantir la terminaison de l'analyse. Pour cela, il est suffisant (mais non nécessaire) que $\mathcal{P}$ soit *de hauteur finie*.

Ces conditions impliquent en fait que $\mathcal{P}$ est un *treillis complet*.

Pour l'analyse de durée de vie, $\mathcal{P} = (2^{\mathcal{V}}, \subseteq, \emptyset, \cup)$.

Valuations

Soit $\mathcal{L}$ l'ensemble des étiquettes du graphe de flot de contrôle.

On manipule des *valuations* qui à chaque étiquette ℓ associent une propriété. Une valuation est donc un élément de $\mathcal{L} \rightarrow \mathcal{P}$.

On se donne une valuation *à l'entrée* et une valuation *à la sortie*, notées $\text{propriété}_{\text{entrée}}(\ell)$ et $\text{propriété}_{\text{sortie}}(\ell)$.

ensiie

Arêtes

On se donne un *graphe de dépendances*, simple sous-ensemble de $\mathcal{L} \times \mathcal{L}$. L'existence d'une arête entre ℓ et ℓ' est notée $\ell \to \ell'$.

On associe à chaque arête une *inéquation :*

$$\text{propriété}_{\text{sortie}}(\ell) \sqsubseteq \text{propriété}_{\text{entrée}}(\ell') \qquad \text{si } \ell \to \ell'$$

Pour l'analyse de durée de vie, une analyse *arrière*, le graphe de dépendances est *l'inverse* du graphe de flot de contrôle. Ainsi, $\text{propriété}_{\text{entrée}}$ est $\text{vivantes}_{\text{sortie}}$ et $\text{propriété}_{\text{sortie}}$ est $\text{vivantes}_{\text{entrée}}$.

Fonctions de transfert

L'effet de chaque instruction est modélisé par une *fonction de transfert* de $\mathcal{P}$ dans $\mathcal{P}$.

Toute fonction de transfert f doit être *monotone*, ce que l'on peut écrire de deux façons équivalentes :

$$p_1 \sqsubseteq p_2 \Rightarrow f(p_1) \sqsubseteq f(p_2)$$
$$f(p_1 \sqcup p_2) \sqsupseteq f(p_1) \sqcup f(p_2)$$

Cette condition signifie qu'une meilleure information à l'entrée d'une instruction doit donner une meilleure information à la sortie.

On n'exige pas la *distributivité :*

$$f(p_1 \sqcup p_2) = f(p_1) \sqcup f(p_2)$$

Fonctions de transfert

On associe à chaque point ℓ une fonction de transfert notée transfert(ℓ), d'où on déduit une *inéquation :*

$$\text{transfert}(\ell)(\text{propriété}_{\text{entrée}}(\ell)) \sqsubseteq \text{propriété}_{\text{sortie}}(\ell)$$

Pour l'analyse de durée de vie, la fonction de transfert est donnée par :

$$\text{transfert}(\ell)(p) = (p \setminus \text{tuées}(\ell)) \cup \text{engendrées}(\ell)$$

Cette fonction est *monotone* et distributive.

Conditions initiales

Pour certaines analyses, il est utile d'imposer des *conditions initiales* en certains points.

On se donne donc une fonction initiale de $\mathcal{L}$ dans $\mathcal{P}$, et on associe à chaque point une nouvelle *inéquation :*

$$\text{initiale}(\ell) \sqsubseteq \text{propriété}_{\text{entrée}}(\ell)$$

Pour l'analyse de durée de vie, on pose $\text{initiale}(\ell) = \emptyset$ pour tout ℓ.

Vers une inéquation unique

On peut transformer le système d'inéquations en *une seule inéquation* exprimée dans le treillis $(\mathcal{L} \to \mathcal{P})^2$:

$$\begin{pmatrix} \ell \mapsto \text{initiale}(\ell) \sqcup \bigsqcup_{\ell' \to \ell} \text{propriété}_{\text{sortie}}(\ell') \\ \ell \mapsto \text{transfert}(\ell)(\text{propriété}_{\text{entrée}}(\ell)) \end{pmatrix} \sqsubseteq \begin{pmatrix} \text{propriété}_{\text{entrée}} \\ \text{propriété}_{\text{sortie}} \end{pmatrix}$$

Cette inéquation est de la forme

$$F(X) \sqsubseteq X$$

où F est monotone.

Vers une équation au point fixe

Théorème (Tarski). Soit F une fonction monotone d'un treillis complet vers lui-même. Alors l'équation $F(X) = X$ admet une plus petite solution, appelée *plus petit point fixe* de F et notée lfp F. De plus, celle-ci coïncide avec la plus petite solution de l'inéquation $F(X) \sqsubseteq X$.

Calcul par approximations successives

Le plus petit point fixe de F peut être calculé par *approximations inférieures successives :*

Théorème. Soit F une fonction monotone d'un treillis complet vers lui-même. On suppose que toute suite croissante converge. Alors,

$$\mathsf{lfp}\ F = \lim_{n \to \infty} F^n(\perp)$$

Complexité

Quelle est la *complexité* de cette approche directe dans le cas de l'analyse de durée de vie ?

La *hauteur* du treillis $(\mathcal{L} \rightarrow 2^{\mathcal{V}})^2$ est $O(\mid \mathcal{L} \mid\mid \mathcal{V} \mid)$ et constitue une borne pour le nombre d'étapes nécessaires avant la convergence.

À chaque étape, le nombre d'opérations du treillis (union, restriction) requises est $O(\mid G \mid)$ où G est le graphe de flot de contrôle, dont on compte sommets et arêtes.

Enfin, chaque opération du treillis exige un temps $O(\mid \mathcal{V} \mid)$ si l'on représente les ensembles de variables par des vecteurs de bits.

Le coût total est donc en principe $O(\mid \mathcal{L} \mid\mid G \mid\mid \mathcal{V} \mid^2)$. Il s'agit d'une complexité très élevée.

Complexité

Quelle est la *complexité* de cette approche directe dans le cas de l'analyse de durée de vie ?

La *hauteur* du treillis $(\mathcal{L} \to 2^{\mathcal{V}})^2$ est $O(\mid \mathcal{L} \mid\mid \mathcal{V} \mid)$ et constitue une borne pour le nombre d'étapes nécessaires avant la convergence.

À chaque étape, le nombre d'opérations du treillis (union, restriction) requises est $O(\mid G \mid)$ où G est le graphe de flot de contrôle, dont on compte sommets et arêtes.

Enfin, chaque opération du treillis exige un temps $O(\mid \mathcal{V} \mid)$ si l'on représente les ensembles de variables par des vecteurs de bits.

Le coût total est donc en principe $O(\mid \mathcal{L} \mid\mid G \mid\mid \mathcal{V} \mid^2)$. Il s'agit d'une complexité très élevée.

Complexité

Quelle est la *complexité* de cette approche directe dans le cas de l'analyse de durée de vie ?

La *hauteur* du treillis $(\mathcal{L} \rightarrow 2^{\mathcal{V}})^2$ est $O(\mid \mathcal{L} \mid\mid \mathcal{V} \mid)$ et constitue une borne pour le nombre d'étapes nécessaires avant la convergence.

À chaque étape, le nombre d'opérations du treillis (union, restriction) requises est $O(\mid G \mid)$ où G est le graphe de flot de contrôle, dont on compte sommets et arêtes.

Enfin, chaque opération du treillis exige un temps $O(\mid \mathcal{V} \mid)$ si l'on représente les ensembles de variables par des vecteurs de bits.

Le coût total est donc en principe $O(\mid \mathcal{L} \mid\mid G \mid\mid \mathcal{V} \mid^2)$. Il s'agit d'une complexité très élevée.

ensiie

Complexité

Quelle est la *complexité* de cette approche directe dans le cas de l'analyse de durée de vie ?

La *hauteur* du treillis $(\mathcal{L} \rightarrow 2^{\mathcal{V}})^2$ est $O(\mid \mathcal{L} \mid\mid \mathcal{V} \mid)$ et constitue une borne pour le nombre d'étapes nécessaires avant la convergence.

À chaque étape, le nombre d'opérations du treillis (union, restriction) requises est $O(\mid G \mid)$ où G est le graphe de flot de contrôle, dont on compte sommets et arêtes.

Enfin, chaque opération du treillis exige un temps $O(\mid \mathcal{V} \mid)$ si l'on représente les ensembles de variables par des vecteurs de bits.

Le coût total est donc en principe $O(\mid \mathcal{L} \mid\mid G \mid\mid \mathcal{V} \mid^2)$. Il s'agit d'une complexité très élevée.

ensiie

Un algorithme à base de « workset »

Nous ne travaillons pas avec un treillis arbitraire, mais avec un treillis de la forme $\mathcal{L} \to \mathcal{P}$. Il n'est utile de réévaluer la propriété au point ℓ que lorsque la propriété associée à l'un des prédécesseurs de ℓ a évolué.

D'où un meilleur algorithme :

insérer tous les points ℓ dans un ensemble de travail
tant que l'ensemble de travail n'est pas vide
  en retirer un point ℓ
  réévaluer la propriété au point ℓ
  si elle a cru,
    ajouter les successeurs de ℓ à l'ensemble de travail

ensiie

Correction de l'algorithme

L'invariant de l'algorithme est : si l'équation au point fixe est *violée* en un certain point ℓ, alors ℓ est *accessible* à travers le graphe depuis un point de l'ensemble de travail.

Initialement, tout point ℓ appartient à l'ensemble de travail, donc l'invariant est satisfait.

Lorsque l'algorithme s'arrête, l'ensemble de travail est *vide*, donc, d'après l'invariant, l'équation au point fixe n'est *violée nulle part*, c'est-à-dire est satisfaite partout.

Ce résultat est *indépendant de l'ordre* dans lequel les points sont retirés de l'ensemble de travail.

Complexité

Dans le cas de l'analyse de durée de vie, le coût dans le cas le pire est $O(\mid G \mid\mid V \mid^2)$, soit un gain vis-à-vis de l'algorithme naïf.

De plus, la complexité *dépend de l'ordre* dans lequel les points sont retirés de l'ensemble de travail. En gros, il vaut mieux traiter les points *dans l'ordre topologique* imposé par le graphe. (Pourquoi ?)

En pratique, la complexité se rapproche de $O(\mid G \mid\mid V \mid)$ lorsqu'on traite les points dans un ordre obtenu par parcours en profondeur du graphe. Voir « Analysis of a simple algorithm for global data flow problems » (Hecht & Ullman, 1973) ou encore « Iterative Data-flow Analysis, Revisited » (Cooper, Harvey & Kennedy, 2002).

La solution « join-over-all-paths »

L'approche étudiée ici *entremêle* applications des fonctions de transfert et calculs de plus petite borne supérieure.

Une approche différente, nommée *« join-over-all-paths »*, définit la propriété recherchée au point ℓ comme la plus petite borne supérieure des propriétés contribuées par *chacun des chemins* qui mènent à ℓ, et où la propriété contribuée par un chemin pris isolément est définie par *composition* des fonctions de transfert le long de ce chemin.

La solution « join-over-all-paths »

L'approche « join-over-all-paths » *extrait* les calculs de plus petite borne supérieure des applications de fonctions de transfert. En gros, on calcule $f(p_1) \sqcup f(p_2)$ plutôt que $f(p_1 \sqcup p_2)$.

De ce fait, elle produit en général un *meilleur* résultat, avec *égalité* si les fonctions de transfert sont *distributives*.

Cependant, en général, la solution « join-over-all-paths » n'est *pas calculable* car elle est définie comme la plus petite borne supérieure d'une famille *infinie*. En effet, en présence de cycles, les chemins menant à ℓ sont en nombre infini.

Voir « Global Data Flow Analysis and Iterative Algorithms » (Kam & Ullman, 1976) pour plus de détails.

Analyse de durée de vie

Analyses de flot de données

3 Graphe d'interférences

Élimination du code mort

ensiie

Interférence

On pourrait poser que deux variables distinctes *interfèrent* si elles sont toutes deux *vivantes en un même point*. Cette définition ne serait *pas correcte*... (Pourquoi ?)

En fait, il faut poser que deux variables distinctes interfèrent si *l'une est vivante à la sortie d'une instruction qui définit l'autre.*

Deux variables qui n'interfèrent pas *peuvent* être réalisées par un *unique* emplacement – registre physique ou emplacement de pile.

Inversement, deux variables qui interfèrent *doivent* être réalisées par deux emplacements distincts.

Une exception

Supposons x vivante à la sortie d'une instruction qui définit y. Si *la valeur reçue par y est certainement celle de x*, alors il n'y a pas lieu de considérer que les deux variables interfèrent.

Cette propriété est en général indécidable, mais il en existe *un cas particulier simple*, celui d'une instruction **move** y, x.

Ce cas particulier est important car, dans ce cas, on *souhaite* justement que x et y soient réalisés par le même emplacement, de façon à supprimer l'instruction **move**.

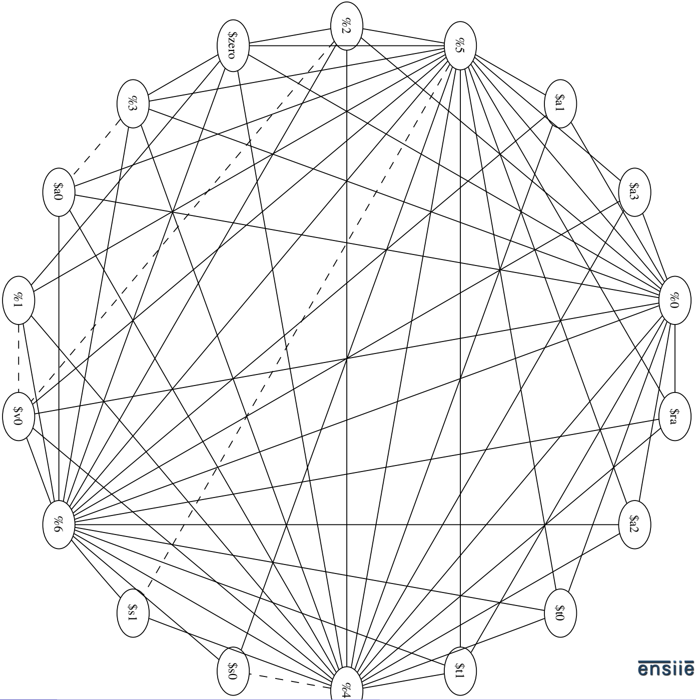
Le graphe d'interférences

On construit un *graphe* dont les sommets sont les variables et dont les arêtes représentent les relations *d'interférence* et de *préférence*.

On crée une arête d'interférence entre deux variables qui interfèrent. On crée une arête de préférence entre deux variables reliées par une instruction **move**.

Ce graphe permet de *spécifier* à l'allocateur de registres les contraintes sous lesquelles il doit travailler.

Voici le graphe d'interférences correspondant à la fonction factorielle.



Les arêtes de préférence sont en pointillés.

Analyse de durée de vie

Analyses de flot de données

Graphe d'interférences

4 Élimination du code mort

Élimination du code mort

Une instruction *pure* dont la variable de destination est *morte* à la sortie de l'instruction est dite *éliminable* et peut être supprimée.

Une instruction est *pure* si elle n'a pas d'effet autre que de modifier sa variable de destination. Par exemple, IConst, IUnOp, IBinOp sont pures ; ICall *n'est pas* pure.

Élimination des initialisations superflues

Lors du passage de PP à UPP, des instructions sont *insérées* au début de chaque procédure pour *initialiser* chaque variable locale à 0, même celle-ci est explicitement initialisée plus loin.

Si une de ces instructions est superflue, son pseudo-registre destination est *mort*. L'instruction est alors *supprimée* après l'analyse de durée de vie, lors du passage de ERTL à LTL.

On obtient ainsi sans difficulté un code *efficace* et qui néanmoins *respecte* la sémantique de PP selon laquelle les variables sont implicitement initialisées à 0.

Raffinement de l'analyse de durée de vie

On peut prendre en compte la notion d'instruction éliminable pour améliorer l'analyse de durée de vie.

Il suffit de considérer qu'une instruction *ne définit ni n'engendre aucune variable* si, *d'après la valuation courante*, elle est éliminable.