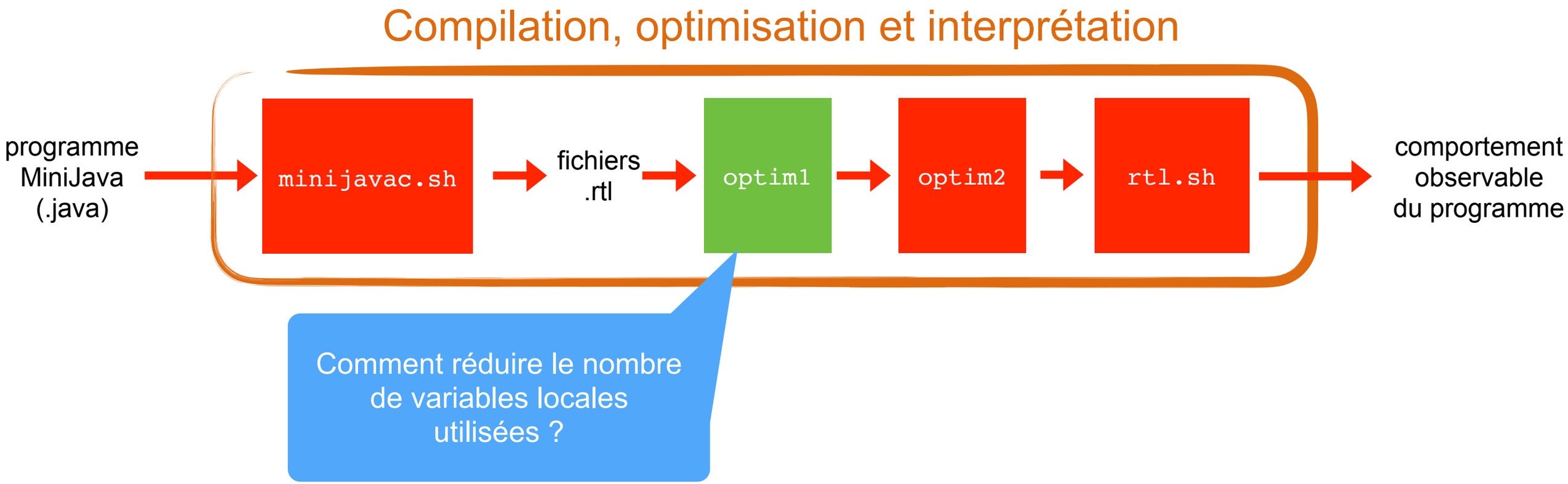
AST

Analyse statique pour l'optimisation de programme

17 janvier 2020

David Pichardie

Architecture globale du projet AST



Durée de vie et interférence

- On souhaite renommer certaines variables pour diminuer le nombre de variables d'une fonction
- C'est une technique indispensable pour réussir *l'allocation de registre* (pour attribuer un emplacement physique à chaque registre).
- Deux variables peuvent partager le même nom si elles n'*interfèrent* pas.
- Pour calculer les interférences, nous allons nous appuyer sur une analyse statique qui calcule les *durées de vie* des variables (*liveness*).

Exemple

Proposer un renommage réduisant le nombre de variables

```
func F(a b)
  entry:
    t1 = Add(a 1)
    t2 = Add(b 1)
    t3 = Add(t1 t2)
    ret t3
```

```
func F(a b)
  entry:
    t1 = Add(a 1)
    t3 = t1
    t2 = Add(b 1)
    t3 = Add(t3 t2)
    ret t3
```

Durée de vie

lu pour exécuter une instruction

Une variable v est vivante au point n
1.s'il existe un chemin menant de n à un point n' où v est utilisée
2.et si v n'est pas re-définie le long de ce chemin

écrit pour exécuter une instruction

chemin dans le graphe de flot de contrôle

Une variable v est morte au point n, s'il est n'est pas vivante en n

Exemple

Indiquez les variables vivantes à chaque point

```
func F(a b)
  entry:
    t1 = Add(a 1)
    t2 = Add(b 1)
    t3 = Add(t1 t2)
    ret t3
```

```
func F(a b)
  entry:
    t1 = Add(a 1)
    t3 = t1
    t2 = Add(b 1)
    t3 = Add(t3 t2)
    ret t3
```

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

notée v ∈ gen(n) dans la suite

- Une variable v est engendrée au point n si l’instruction en n utilise v, c’est-à-dire si elle lit une valeur dans v.
- Dans ce cas, v est vivante en entrée de n.

Mort d’une variable

notée v ∈ kill(n) dans la suite

- Une variable v est tuée au point n si l’instruction en n définit v, c’est-à-dire si elle écrit une valeur dans v.
- Dans ce cas, v est morte en entrée de n (à préciser).

Vie d’une variable

- Si le point n n’engendre ni ne tue v, alors v est vivante immédiatement avant n si et seulement si elle est vivante immédiatement après n.
- Une variable et vivante après n si et seulement si elle est vivante avant l’un quelconque des successeurs de n.

Mise en équations

- Les assertions précédentes permettent d'exprimer le problème sous forme d'équations ensemblistes.

- À chaque étiquette n du graphe de flot de contrôle, on associe deux ensembles de variables:

- $L_{in}(n)$ est l'ensemble des variables vivantes immédiatement avant l'instruction située au point n;

- $L_{out}(n)$ est l'ensemble des variables vivantes immédiatement après l'instruction située au point n.

Équations

- Les équations / inéquations qui définissent l'analyse sont :

$$L_{out}(n) \supseteq L_{in}(n') \qquad \text{si } n \rightarrow n'$$

$$L_{in}(n) = (L_{out}(n) - kill(n)) \cup gen(n)$$

- Pourquoi pas $L_{in}(n) = (L_{out}(n) \cup gen(n)) - kill(n)$?
- Toute solution de ce système est sûre, mais la plus petite solution donne le résultat le plus informatif.

Équations

- La recherche de la plus petite solution demande de réunir toute les inéquations

$$L_{out}(n) \supseteq L_{in}(n') \qquad \text{si } n \rightarrow n'$$

en une équation

$$L_{out}(n) = \bigcup_{n \rightarrow n'} L_{in}(n')$$

- En particulier : si n est un point sans successeurs $L_{out}(n) = \varnothing$

Équations

- Au final le système d'équation prend la forme suivante

  pour tout point n

$$L_{in}(n) = (L_{out}(n) - kill(n)) \cup gen(n)$$

$$L_{out}(n) = \bigcup_{n \to n'} L_{in}(n')$$

- Remarque : dans cette analyse kill(n)=def(n) et gen(n)=use(n)

Algorithme

- Les équations peuvent être résolues avec l'algorithme suivant

pour tout point n
$L_{out}(n) := \varnothing$
$L_{in}(n) := \varnothing$

répéter
pour tout point n
$L'_{out}(n) := L_{out}(n)$
$L'_{in}(n) := L_{in}(n)$
$L_{out}(n) := \bigcup_{n \rightarrow n'} L_{in}(n')$
$L_{in}(n) := (L_{out}(n) - kill(n)) \cup gen(n)$

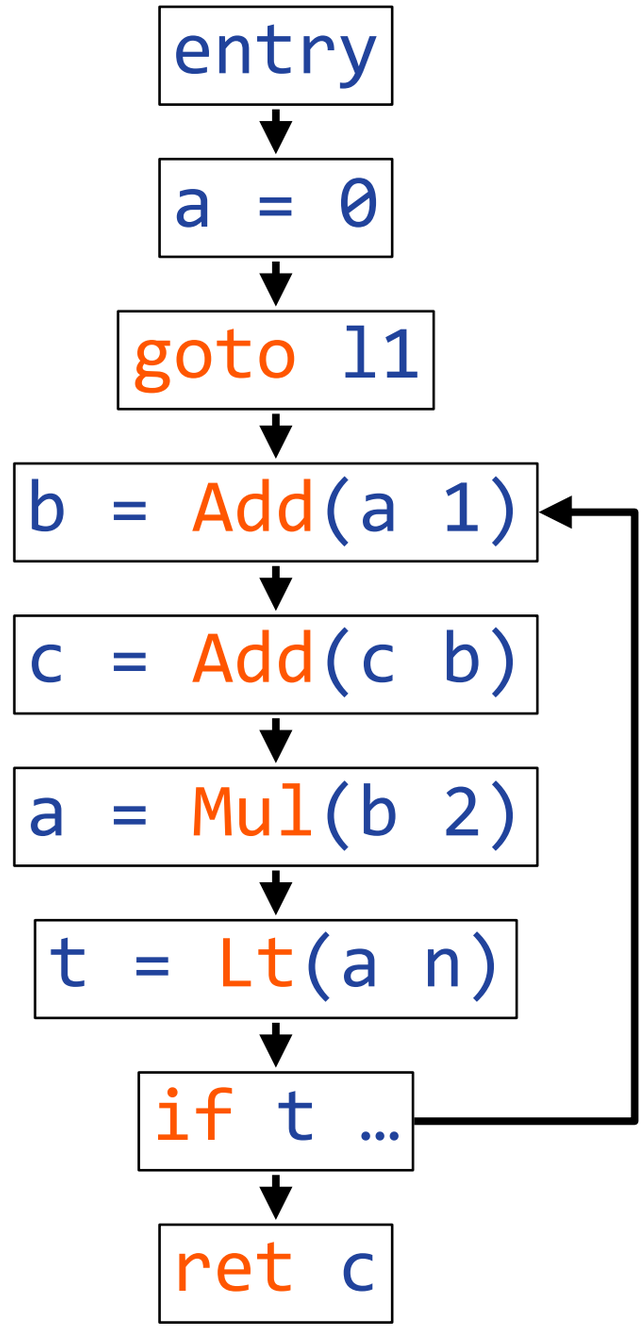
jusqu'à $L'_{in}(n) = L_{in}(n)$ et $L'_{out}(n) = L_{out}(n)$ pour tout point n

Exemple

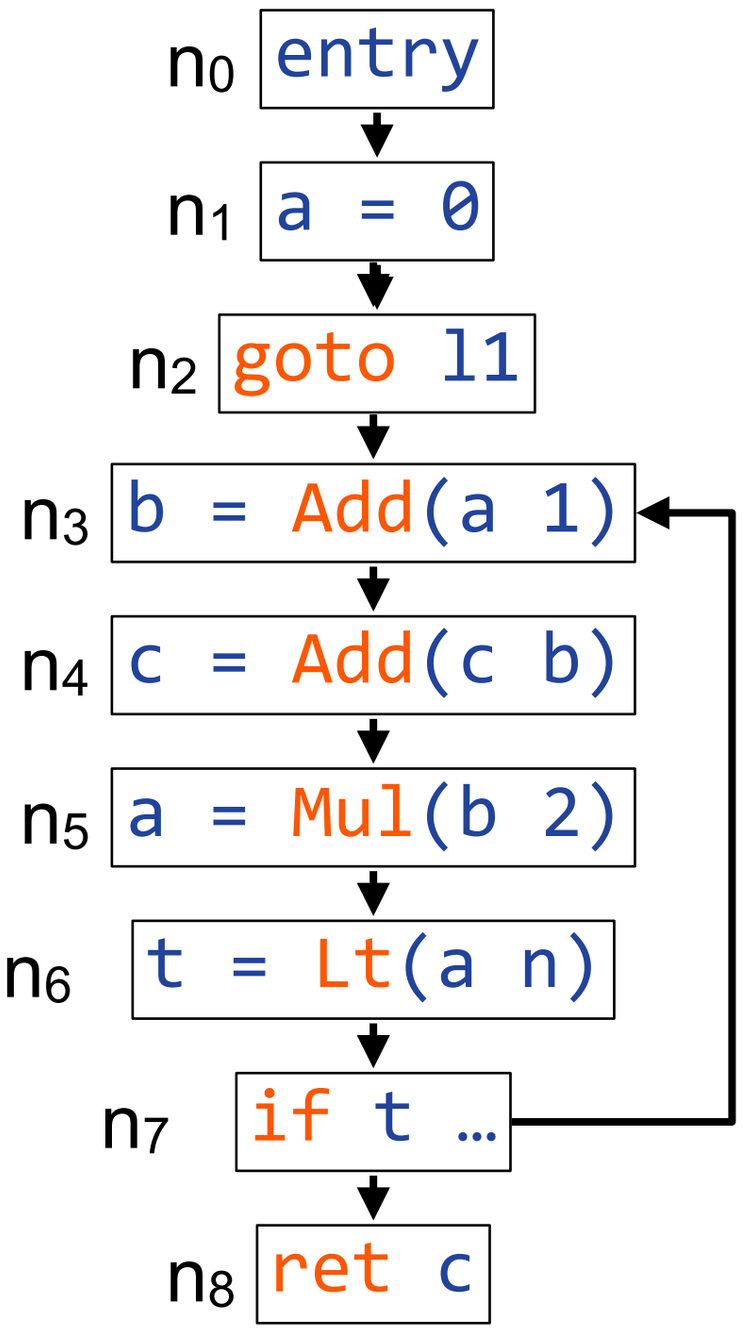
```
func Exemple2(n c)
  entry:
    a = 0
    goto l1
  l1:
    b = Add(a 1)
    c = Add(c b)
    a = Mul(b 2)
    t.0 = Lt(a n)
    if t.0 goto l1 else if0_end
if0_end:
  ret c
```

Exemple

```
func Exemple2(n c)
  entry:
    a = 0
    goto l1
  l1:
    b = Add(a 1)
    c = Add(c b)
    a = Mul(b 2)
    t = Lt(a n)
    if t goto l1 else if0_end
if0_end:
  ret c
```



Exemple



$gen(n_0)=\varnothing$ $kill(n_0)=\{c,n\}$

$gen(n_1)=\varnothing$ $kill(n_1)=\{a\}$

$gen(n_2)=\varnothing$ $kill(n_2)=\varnothing$

$gen(n_3)=\{a\}$ $kill(n_3)=\{b\}$

$gen(n_4)=\{b,c\}$ $kill(n_4)=\{c\}$

$gen(n_5)=\{b\}$ $kill(n_5)=\{a\}$

$gen(n_6)=\{a,n\}$ $kill(n_6)=\{t\}$

$gen(n_7)=\{t\}$ $kill(n_7)=\varnothing$

$gen(n_8)=\{c\}$ $kill(n_8)=\varnothing$

$L_{in}(n_0)=L_{out}(n_0)-\{c,n\}$
$L_{out}(n_0)=L_{in}(n_1)$
$L_{in}(n_1)=L_{out}(n_1)-\{a\}$
$L_{out}(n_1)=L_{in}(n_2)$
$L_{in}(n_2)=L_{out}(n_2)$
$L_{out}(n_2)=L_{in}(n_3)$
$L_{in}(n_3)=(L_{out}(n_3)-\{b\})U\{a\}$
$L_{out}(n_3)=L_{in}(n_4)$
$L_{in}(n_4)=(L_{out}(n_4)-\{b,c\})U\{c\}$
$L_{out}(n_4)=L_{in}(n_5)$
$L_{in}(n_5)=(L_{out}(n_5)-\{a\})U\{b\}$
$L_{out}(n_5)=L_{in}(n_6)$
$L_{in}(n_6)=(L_{out}(n_6)-\{t\})U\{a,n\}$
$L_{out}(n_6)=L_{in}(n_7)$
$L_{in}(n_7)=L_{out}(n_7)U\{t\}$
$L_{out}(n_7)=L_{in}(n_8)UL_{in}(n_3)$
$L_{in}(n_8)=L_{out}(n_8)U\{c\}$
$L_{out}(n_8)=\varnothing$

Astuces (pour cette exemple)

- Astuce #1: nous pouvons déjà calculer $L_{in}(n_8)$ et $L_{out}(n_8)$

$L_{out}(n_8)=\varnothing$ $L_{in}(n_8)=\varnothing U\{c\}=\{c\}$

- Astuce #2 : nous pouvons nous contenter d'appliquer l'algorithme itératif sur $L_{out}(n_7)$ et $L_{in}(n_3)$ en simplifiant les équations

$L_{in}(n_3)=(L_{out}(n_3)-\{b\})U\{a\}$

$=(((L_{out}(n_4)-\{b\})U\{c\})-\{b\})U\{a\}=(L_{out}(n_4)-\{b\})U\{a,c\}$

$=(((L_{out}(n_5)-\{a\})U\{b\})-\{b\})U\{a,c\}=(L_{out}(n_5)-\{b\})U\{a,c\}$

$=((L_{out}(n_6)-\{t\})U\{a,n\})-\{b\})U\{a,b\}$ $=(L_{out}(n_6)-\{t\})U\{a,b,n\}$

$=((L_{out}(n_7)U\{t\})-\{t\})U\{a,b,n\}=L_{out}(n_7)U\{a,b,n\}$

Exercice

- Calculer les informations $L_{in}(n)$ et $L_{out}(n)$ pour chaque point du programme suivant.

```
func ComputeFac(n)
  entry:
    t.0 = Lt(n 1)
    if t.0 goto if0_then else if0_else
  if0_then:
    n0 = 1
    goto if0_end
if0_else:
    t.2 = Sub(n 1)
    t.1 = call ComputeFac(t.2)
    n0 = Mul(num t.1)
    goto if0_end
if0_end:
    ret n0
```

Améliorations de l'algorithme

- L'ordre de la boucle interne « pour tout point » influence la vitesse de convergence
  - Amélioration #1 : utiliser un tri topologique inverse (faible) sur le graphe de flot de contrôle
  - Amélioration #2 : algorithme de *workset* (voir prochain cours)
  - Amélioration #3 : calculer les informations blocs par blocs plutôt que point par point

Application de l’analyse de durée de vie

- Deux variables ne peuvent pas être réalisées par un même registre si elles interfèrent.
- Interférence : chevauchement des durées de vie de deux variables.
- Les interférences d'un programme sont représentées par un graphe, le graphe d'interférences.

Interférence

- Deux variables distinctes interfèrent si elles sont toutes les deux vivantes en un même point.

- Pour calculer les paires de variables interférentes on regarde les point de définition :

pour tout point n
  pour tout x ∈ def(n)
    pour tout y ∈ L_{out}(n)
      si x ≠ y ajouter (x,y) à INTERFERENCES

- Peut-on faire un peu mieux ?

Exercices

- Quel est le graphe d'interférence de la fonction suivante ?

```
func F()
  entry:
    j = 0
    k = 0
    g = Add(j 12)
    h = Sub(k 1)
    f = Mul(g h)
    e = Add(j 8)
    m = Add(j 16)
    b = Add(f 6)
    c = Add(e 8)
    d = c
    k = Add(m 4)
    j = b
    y = Add(d j)
    z = Add(y k)
    ret z
```

Coloriage du graphe d’interférence

- Un graphe non-orienté est coloriable si chaque couleur attribué à chaque sommet satisfait la contrainte suivante : deux sommets reliés par une arête ne doivent pas avoir la même couleur
- Pour réduire le nombre de variables d’une fonction, il suffit d’attribuer le même nom à chaque variable possédant la même couleur

Autre application de l’analyse de durée de vie

- 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, Add et Sub sont pures; call n'est pas pure.

Pour la prochaine séance

- Lire le chapitre 10 *Liveness analysis*
- Lire le chapitre 11 *Register allocation*