Optimisations et analyses statiques

Opportunités d'optimisation

- Élimination des tests de sûreté dans les appels et dans les extractions des champs des paires.

Pour $(e_1\ e_2)$:

```
fct2call = /* code de e1 */;
arg1 = /* code de e2 */;
if (pred_clos(fct2call))
  goto do_invoke;
else
  goto do_call_error;
```

Pour $(\mathrm{car}\ e_1)$:

```
tmp1 = /* code de e1 */;
if (pred_pair(tmp1))
  tmp = get_pair_car(tmp1);
else
  goto do_car_error;
```

Pour $(\mathrm{cdr}\ e_1)$:

```
tmp1 = /* code de e1 */;
if (pred_pair(tmp1))
  tmp = get_pair_cdr(tmp1);
else
  goto do_cdr_error;
```

- Abandonner le test d'une conditionnelle qui choisit toujours la même branche.

Pour $(\mathrm{if}\ e_1\ e_2\ e_3)$ où e_1 est toujours faux.

```
tmp1 = /* code de e1 */;
if (get_bool_inC(tmp1))
  tmp2 = /* code de e2 */;
else
  tmp2 = /* code de e3 */;
```

Opportunités d'optimisation

- Simplifier la mécanique générale d'appel.

  Pour $(e_1\ e_2)$.

```
fct2call = /* code de e1 */;
arg1 = /* code de e2 */;
goto get_clos_ptr(fct2call);
```

- Abandonner du code mort.

  Pour $(\lambda \mathrm{x.\ f\ y})$ qui n'est jamais appelée.

```
tmp = make_clos_2(&&LAM42, f, y);
```

- Etc.

Applicabilité des optimisations

Pour pouvoir appliquer des optimisations de façon sécuritaire, il faut analyser statiquement le code.

C'est-à-dire, il faut avoir une idée fiable de se qui va (ou peut) se passer à l'exécution.

Moyens?

- Exécution réelle.
- Profilage.
- Heuristiques.
- Interprétation abstraite.

Applicabilité des optimisations

Pour pouvoir appliquer des optimisations de façon sécuritaire, il faut analyser statiquement le code.

C'est-à-dire, il faut avoir une idée fiable de se qui va (ou peut) se passer à l'exécution.

Moyens?

- Exécution réelle. NON!
- Profilage. NON!
- Heuristiques. NON!
- Interprétation abstraite. Oui.

Langage employé

Syntaxe des expressions:

e	$::=$	$\#f$	constante “faux”
	$\mid$	x	référence
	$\mid$	$(\lambda \mathrm{x}.\ e)$	λ-expression
	$\mid$	$(e\ e)$	appel de fonction
	$\mid$	$(\mathrm{if}\ e\ e\ e)$	conditionnelle
	$\mid$	$(\mu \mathrm{x}.\ e)$	point fixe
	$\mid$	$(\mathrm{cons}\ e\ e)$	création de paire
	$\mid$	$(\mathrm{car}\ e)$	extraction du CAR
	$\mid$	$(\mathrm{cdr}\ e)$	extraction du CDR
	$\mid$	$(\mathrm{pair?}\ e)$	test de “parité”

Syntaxe des valeurs:

v	$::=$	$\#f$	constante “faux”
	$\mid$	$(\lambda \mathrm{x}.\ e)$	fonction
	$\mid$	$(\mathrm{cons}\ v\ v)$	paire

Langage employé

Sémantique: α-réductions

Contexte des α-réductions:

$$
\begin{array}{rcl}
C^\alpha & ::= & (\lambda \mathrm{x}.\ C^\alpha) \\
& | & (C^\alpha\ e) \\
& | & (e\ C^\alpha) \\
& | & (\mathrm{if}\ C^\alpha\ e\ e) \\
& | & (\mathrm{if}\ e\ C^\alpha\ e) \\
& | & (\mathrm{if}\ e\ e\ C^\alpha) \\
& | & (\mu \mathrm{x}.\ C^\alpha) \\
& | & (\mathrm{cons}\ C^\alpha\ e) \\
& | & (\mathrm{cons}\ e\ C^\alpha) \\
& | & (\mathrm{car}\ C^\alpha) \\
& | & (\mathrm{cdr}\ C^\alpha) \\
& | & (\mathrm{pair?}\ C^\alpha) \\
& | & [\![\cdot]\!]
\end{array}
$$

Règles d'α-réduction:

$$
\begin{array}{rcl}
C^\alpha[\![\lambda \mathrm{x}.\ e]\!] & \overset{\alpha}{\mapsto} & C^\alpha[\![\lambda \mathrm{y}.\ (e[\mathrm{x} \mapsto \mathrm{y}])]\!] \\
C^\alpha[\![\mu \mathrm{x}.\ e]\!] & \overset{\alpha}{\mapsto} & C^\alpha[\![\mu \mathrm{y}.\ (e[\mathrm{x} \mapsto \mathrm{y}])]\!]
\end{array}
$$

Langage employé

Sémantique: β-réductions

Contexte des β-réductions:

$$
\begin{array}{rcl}
C^\beta & ::= & (C^\beta\ e) \\
& | & (v\ C^\beta) \\
& | & (\text{if } C^\beta\ e\ e) \\
& | & (\text{cons } C^\beta\ e) \\
& | & (\text{cons } v\ C^\beta) \\
& | & (\text{car } C^\beta) \\
& | & (\text{cdr } C^\beta) \\
& | & (\text{pair? } C^\beta) \\
& | & [\![\cdot]\!]
\end{array}
$$

Règles de β-réduction (d'évaluation):

$$
\begin{array}{rcl}
C^\beta[\![(\lambda \text{x. } e)\ v]\!] & \overset{\beta}{\mapsto} & C^\beta[\![e[\text{x} \mapsto v]]\!] \\
C^\beta[\![\text{if } \#f\ e_2\ e_3]\!] & \overset{\beta}{\mapsto} & C^\beta[\![e_3]\!] \\
C^\beta[\![\text{if } (\lambda \text{x. } e_1)\ e_2\ e_3]\!] & \overset{\beta}{\mapsto} & C^\beta[\![e_2]\!] \\
C^\beta[\![\text{if } (\text{cons } v_1\ v_2)\ e_2\ e_3]\!] & \overset{\beta}{\mapsto} & C^\beta[\![e_2]\!] \\
C^\beta[\![\mu \text{x. } e]\!] & \overset{\beta}{\mapsto} & C^\beta[\![e[\text{x} \mapsto (\mu \text{x. } e)]]\!] \\
C^\beta[\![\text{car } (\text{cons } v_1\ v_2)]\!] & \overset{\beta}{\mapsto} & C^\beta[\![v_1]\!] \\
C^\beta[\![\text{cdr } (\text{cons } v_1\ v_2)]\!] & \overset{\beta}{\mapsto} & C^\beta[\![v_2]\!] \\
C^\beta[\![\text{pair? } \#f]\!] & \overset{\beta}{\mapsto} & C^\beta[\![\#f]\!] \\
C^\beta[\![\text{pair? } (\lambda \text{x. } e)]\!] & \overset{\beta}{\mapsto} & C^\beta[\![\#f]\!] \\
C^\beta[\![\text{pair? } (\text{cons } v_1\ v_2)]\!] & \overset{\beta}{\mapsto} & C^\beta[\![\text{cons } v_1\ v_2]\!]
\end{array}
$$

Langage employé

Exemple de programme:

Fonction de concaténation.

```
μA.
   (λm.
      (λn.
         (if m
             (cons (car m) ((A (cdr m)) n))
             n)))
```

Langage employé

Hypothèses de travail:

Un programme source E

- est α-converti, i.e. que chaque variable a un nom distinct;
- est étiqueté, i.e. que chaque expression est étiquetée et cette étiquette est différente de celle des autres expressions;
- n'a pas de variables libres.

Conservatisme

Toute analyse que nous allons étudier se doit d'être conservatrice.

Définition: Une analyse *conservatrice* se doit de

- déclarer comme "possible" la production d'une valeur v par une expression e_l si cette expression peut s'évaluer à cette valeur à l'exécution et
- déclarer comme "possible" la liaison d'une variable x à une valeur v si cette variable peut être liée à cette valeur à l'exécution.

Formellement,

- si $E \mapsto^* C^\beta[\![e_l]\!]$ et $e_l \mapsto^* v$, alors l'analyse doit mentionner que e_l peut s'évaluer à v, entre autres;
- si $E \mapsto^* C^\beta[\![(\lambda \mathrm{x}.\ e)\ v]\!]$, alors l'analyse doit mentionner que x peut être lié à v, entre autres;
- si $E \mapsto^* C^\beta[\![\mu \mathrm{x}.\ e]\!]$ et $\mu \mathrm{x}.\ e \mapsto^* v$, alors l'analyse doit mentionner que x peut être lié à v, entre autres.

Conservatisme versus précision

L'obligation d'être conservatrice force éventuellement l'analyse à surestimer la réalité.

Exemple 1:

Peut-on éliminer le test de sûreté dans le programme suivant?

$$((\lambda \mathrm{x}.\ e_1)\ e_2)$$

Exemple 2:

Peut-on éliminer les tests de sûreté dans le programme suivant?

$$((\lambda \mathrm{f}.\ (\mathrm{f}\ e_1))\ (\lambda \mathrm{y}.\ e_2))$$

Exemple 3:

Peut-on éliminer les tests de sûreté dans le programme suivant?

$$((\lambda \mathrm{p}.\ (\mathrm{if}\ \mathrm{p}\ ((\mathrm{car}\ \mathrm{p})\ \#f)\ \#f))\\ (\mathrm{cons}\ (\lambda \mathrm{y}.\ e_1)\ \#f))$$

Analyses statiques conventionnelles

Dans les analyses statiques destinées aux langages conventionnels, l'estimation du flot de contrôle est simple:

- on se préoccupe surtout du flot de contrôle *intra-procédural*;
- le flot de contrôle intra-procédural est simple à obtenir;
- le flot de contrôle intra-procédural est habituellement suffisant à cause que les procédures ont tendance à être longues, contrairement à celles présentes dans les langages fonctionnels;
- aussi, au niveau *inter-procédural*, le flot de contrôle est souvent relativement facile à obtenir à cause que les procédures appelées sont (le plus souvent) clairement identifiées.

Analyses statiques conventionnelles

Dans les analyses statiques destinées aux langages fonctionnels, l'estimation du flot de contrôle est plus complexe et les analyses conventionnelles ne permettent pas d'obtenir une estimation suffisamment précise:

- il y a de nombreuses petites fonctions;
- celles-ci ne sont pas assez longues pour offrir de véritables opportunités d'optimisation;
- donc, il faut utiliser des analyses inter-procédurales pour obtenir une estimation valable;
- or, le flot de contrôle inter-procédurale dépend de la propagation des fonctions dans le programme, i.e. du flot de données;
- conventionnellement, on calcule une estimation du flot de données une fois que l'on possède une estimation du flot de contrôle.

On a donc une analyse de flot de contrôle et une analyse de flot de données qui sont mutuellement dépendantes.

Analyses statiques conventionnelles

Les langages orientés-objet ont un problème similaire à celui des langages fonctionnels.

- Il y a une prolifération de petites procédures, les méthodes.
- Le flot de contrôle dépend fortement du flot de données à cause des objets qui transportent des méthodes.

Première analyse

On appelle *analyses de flot de contrôle* les analyses calculant à la fois une estimation du flot de contrôle et une estimation du flot de données pour un langage fonctionnel.

Voici ce que notre première analyse devrait faire:

- Recevoir un programme E étiqueté.
- Retourner
  - pour chaque expression e_l, l'ensemble α_l des fonctions qui peuvent être le résultat de l'évaluation de e_l et
  - pour chaque variable x, l'ensemble α_{x} des fonctions auxquelles x peut être liée;
  - ces ensembles de fonctions sont représentés comme des ensembles d'étiquettes de λ-expressions.

Première analyse

Exemple:

$$
\begin{array}{l}
({}_1({}_2(\lambda_3 \mathrm{f}.\ (\lambda_4 \mathrm{x}.\ ({}_5\mathrm{f}_6\ ({}_7\mathrm{f}_8\ \mathrm{x}_9)))) \\
\qquad (\lambda_{10} \mathrm{y}.\ (\mathrm{cons}_{11}\ \#f_{12}\ \mathrm{y}_{13}))) \\
\quad (\mathrm{cons}_{14}\ \#f_{15}\ \#f_{16}))
\end{array}
$$

Première analyse

Exemple:

$$
\begin{array}{l}
({}_1({}_2(\lambda_3 \mathrm{f}.\ (\lambda_4 \mathrm{x}.\ ({}_5\mathrm{f}_6\ ({}_7\mathrm{f}_8\ \mathrm{x}_9)))) \\
\qquad (\lambda_{10} \mathrm{y}.\ (\mathrm{cons}_{11}\ \#f_{12}\ \mathrm{y}_{13}))) \\
\quad (\mathrm{cons}_{14}\ \#f_{15}\ \#f_{16}))
\end{array}
$$

$$
\begin{array}{llll}
\alpha_1 = \emptyset & \alpha_6 = \{10\} & \alpha_{11} = \emptyset & \alpha_{16} = \emptyset \\
\alpha_2 = \{4\} & \alpha_7 = \emptyset & \alpha_{12} = \emptyset & \alpha_{\mathrm{f}} = \{10\} \\
\alpha_3 = \{3\} & \alpha_8 = \{10\} & \alpha_{13} = \emptyset & \alpha_{\mathrm{x}} = \emptyset \\
\alpha_4 = \{4\} & \alpha_9 = \emptyset & \alpha_{14} = \emptyset & \alpha_{\mathrm{y}} = \emptyset \\
\alpha_5 = \emptyset & \alpha_{10} = \{10\} & \alpha_{15} = \emptyset &
\end{array}
$$

Première analyse

Remarque:

Grâce à ces résultats, on a une estimation du flot de contrôle et il est possible d'appliquer une analyse de flot de données conventionnelle.

Question:

Comment obtient-on les ensembles $\alpha_?$ automatiquement?

En procédant en deux phases:

1. En créant un système de contraintes entre les ensembles:
   - pour chaque expression du programme, un certain nombre de contraintes sont générées.
2. Le système de contraintes est résolu:
   - des étiquettes sont propagées d'un ensemble à l'autre jusqu'à ce que les contraintes soient satisfaites;
   - on s'intéresse à la *plus petite solution* aux contraintes.

Première analyse

Génération des contraintes:

Si $e_l = \#f_l$, *aucune contrainte*.
Si $e_l = \mathrm{x}_l$, $\alpha_l \supseteq \alpha_{\mathrm{x}}$
Si $e_l = (\lambda_l \mathrm{x}.\ e_{l_1})$, $\alpha_l \supseteq \{l\}$ (2)
Si $e_l = ({}_l e_{l_1}\ e_{l_2})$, pour tout $l_3 \in \alpha_{l_1}$ t.q. $e_{l_3} = (\lambda_{l_3} \mathrm{x}.\ e_{l_4})$ (7)
$\alpha_{\mathrm{x}} \supseteq \alpha_{l_2}$ (5)
$\alpha_l \supseteq \alpha_{l_4}$ (6)
Si $e_l = (\mathrm{if}_l\ e_{l_1}\ e_{l_2}\ e_{l_3})$, $\alpha_l \supseteq \alpha_{l_2} \cup \alpha_{l_3}$ (3)
Si $e_l = (\mu_l \mathrm{x}.\ e_{l_1})$, $\alpha_l \supseteq \alpha_{l_1}$
$\alpha_{\mathrm{x}} \supseteq \alpha_{l_1}$ (8)
Si $e_l = (\mathrm{cons}_l\ e_{l_1}\ e_{l_2})$, $\alpha_l \supseteq \alpha_{l_1} \cup \alpha_{l_2}$ (4)
Si $e_l = (\mathrm{car}_l\ e_{l_1})$, $\alpha_l \supseteq \alpha_{l_1}$
Si $e_l = (\mathrm{cdr}_l\ e_{l_1})$, $\alpha_l \supseteq \alpha_{l_1}$
Si $e_l = (\mathrm{pair?}_l\ e_{l_1})$, $\alpha_l \supseteq \alpha_{l_1}$

Première analyse

Commentaires:

1. Les contraintes forcent les étiquettes à se propager d'un ensemble à l'autre.
2. *Évaluer* une λ-expression ne retourne que la fonction elle-même; la valeur de e_{l_1} sera retournée à l'*appelant* de la fonction.
3. La valeur du test est oubliée.
4. On considère que détenir une paire qui contient une fonction, c'est comme détenir la fonction elle-même.
5. Correspond au passage de l'argument.
6. Correspond au retour du résultat de l'appel.
7. Des contraintes s'ajoutent *durant* l'analyse.
8. L'expression de point fixe s'évalue à tout ce à quoi sa sous-expression peut s'évaluer. Du même coup, la variable est liée à toutes ces valeurs.

...

Première analyse

Commentaires (suite):

9. Concrètement, il n'est pas nécessaire de générer les contraintes explicitement; il faut simplement s'assurer de les faire respecter.

10. La propagation des étiquettes simule une exécution.

11. Seules les étiquettes des λ-expressions sont propagées.

12. Les résultats de l'analyse sont conservateurs: si une fonction issue de $(\lambda_l \mathrm{x}.\ e)$ est le résultat de e_{l_1} lors de l'exécution concrète du programme, alors $l \in \alpha_{l_1}$.

13. L'inverse n'est pas vrai: si $l \in \alpha_{l_1}$, ça ne garantit pas qu'il y a une fonction issue de $(\lambda_l \mathrm{x}.\ e)$ qui est le résultat de e_{l_1} lors de l'exécution concrète.

14. Pour éviter une trop grande pollution par des étiquettes inutiles, on cherche la plus petite solution aux contraintes.

15. Comme on ne tient pas compte des paires et de la valeur fausse, on n'est pas en mesure d'effectuer d'élimination de tests de sûreté, i.e.:

    - dans $({}_l e_{l_1}\ e_{l_2})$, car e_{l_1} pourrait s'évaluer à autre chose que des fonctions;
    - dans $(\mathrm{car}_l\ e_{l_1})$ et $(\mathrm{cdr}_l\ e_{l_1})$ même si $\alpha_{l_1} = \emptyset$, car e_{l_1} pourrait s'évaluer à $\#f$.

Deuxième analyse

Nous développons une analyse de flot de contrôle qui tient compte des trois types de données, c'est-à-dire, des fonctions, des paires et des booléens.

Voici ce que la deuxième analyse devrait produire à partir d'un programme E:

- pour chaque expression e_l, l'ensemble α_l contient les objets auxquels e_l pourrait s'évaluer et
- pour chaque variable x, l'ensemble α_{x} contient les objets auxquels x pourrait être lié;
- lorsqu'une fonction issue de la λ-expression $(\lambda_l \mathrm{x}.\ e_{l_1})$ est propagée jusqu'à un ensemble $\alpha_?$, on l'indique en insérant λ_l dans $\alpha_?$;
- lorsqu'une paire issue de l'expression $(\mathrm{cons}_l\ e_{l_1}\ e_{l_2})$ est propagée jusqu'à un ensemble $\alpha_?$, on l'indique en insérant P_l dans $\alpha_?$;
- lorsque le booléen faux est propagé jusqu'à un ensemble $\alpha_?$, on l'indique en insérant $\#f$ dans $\alpha_?$.

Comme dans le cas de la première analyse, on effectue l'analyse en générant des contraintes entre les ensembles $\alpha_?$ et en cherchant la plus petite solution à ces contraintes.

Deuxième analyse

Exemple:

Deuxième analyse appliquée au même programme:

$$
\begin{array}{l}
({}_1({}_2(\lambda_3\mathrm{f}.\ (\lambda_4\mathrm{x}.\ ({}_5\mathrm{f}_6\ ({}_7\mathrm{f}_8\ \mathrm{x}_9)))) \\
\qquad (\lambda_{10}\mathrm{y}.\ (\mathrm{cons}_{11}\ \#f_{12}\ \mathrm{y}_{13}))) \\
\quad (\mathrm{cons}_{14}\ \#f_{15}\ \#f_{16}))
\end{array}
$$

Deuxième analyse

Exemple:

Deuxième analyse appliquée au même programme:

$$
\begin{array}{l}
({}_1({}_2(\lambda_3\mathrm{f}.\ (\lambda_4\mathrm{x}.\ ({}_5\mathrm{f}_6\ ({}_7\mathrm{f}_8\ \mathrm{x}_9)))) \\
\quad\ (\lambda_{10}\mathrm{y}.\ (\mathrm{cons}_{11}\ \#f_{12}\ \mathrm{y}_{13}))) \\
\ (\mathrm{cons}_{14}\ \#f_{15}\ \#f_{16}))
\end{array}
$$

$$
\begin{array}{llll}
\alpha_1 = \{P_{11}\} & \alpha_6 = \{\lambda_{10}\} & \alpha_{11} = \{P_{11}\} & \alpha_{16} = \{\#f\} \\
\alpha_2 = \{\lambda_4\} & \alpha_7 = \{P_{11}\} & \alpha_{12} = \{\#f\} & \alpha_{\mathrm{f}} = \{\lambda_{10}\} \\
\alpha_3 = \{\lambda_3\} & \alpha_8 = \{\lambda_{10}\} & \alpha_{13} = \{P_{11}, P_{14}\} & \alpha_{\mathrm{x}} = \{P_{14}\} \\
\alpha_4 = \{\lambda_4\} & \alpha_9 = \{P_{14}\} & \alpha_{14} = \{P_{14}\} & \alpha_{\mathrm{y}} = \{P_{11}, P_{14}\} \\
\alpha_5 = \{P_{11}\} & \alpha_{10} = \{\lambda_{10}\} & \alpha_{15} = \{\#f\} &
\end{array}
$$

Deuxième analyse

Génération des contraintes:

Si $e_l = \#f_l, \quad \alpha_l \supseteq \{\#f\}$

Si $e_l = \mathrm{x}_l, \quad \alpha_l \supseteq \alpha_{\mathrm{x}}$

Si $e_l = (\lambda_l \mathrm{x}.\ e_{l_1}), \quad \alpha_l \supseteq \{\lambda_l\}$

Si $e_l = ({}_l e_{l_1}\ e_{l_2}), \quad$ pour tout $\lambda_{l_3} \in \alpha_{l_1}$ t.q. $e_{l_3} = (\lambda_{l_3} \mathrm{x}.\ e_{l_4})$

$\alpha_{\mathrm{x}} \supseteq \alpha_{l_2}$

$\alpha_l \supseteq \alpha_{l_4}$

Si $e_l = (\mathrm{if}_l\ e_{l_1}\ e_{l_2}\ e_{l_3}), \quad \alpha_l \supseteq \alpha_{l_2} \cup \alpha_{l_3}$

Si $e_l = (\mu_l \mathrm{x}.\ e_{l_1}), \quad \alpha_l \supseteq \alpha_{l_1}$

$\alpha_{\mathrm{x}} \supseteq \alpha_{l_1}$

Si $e_l = (\mathrm{cons}_l\ e_{l_1}\ e_{l_2}), \quad \alpha_l \supseteq \{P_l\}$ (2)

Si $e_l = (\mathrm{car}_l\ e_{l_1}), \quad$ pour tout $P_{l_2} \in \alpha_{l_1}$ t.q. $e_{l_2} = (\mathrm{cons}_{l_2}\ e_{l_3}\ e_{l_4})$

$\alpha_l \supseteq \alpha_{l_3}$

Si $e_l = (\mathrm{cdr}_l\ e_{l_1}), \quad$ pour tout $P_{l_2} \in \alpha_{l_1}$ t.q. $e_{l_2} = (\mathrm{cons}_{l_2}\ e_{l_3}\ e_{l_4})$

$\alpha_l \supseteq \alpha_{l_4}$

Si $e_l = (\mathrm{pair?}_l\ e_{l_1}), \quad \alpha_l \supseteq \{P_{l_2} \mid P_{l_2} \in \alpha_{l_1}\} \cup \{\#f\}$

Deuxième analyse

Commentaires:

1. Les résultats d'analyse sont encore conservateurs mais, cette fois-ci, ils donnent des informations sur les trois types d'objets, i.e.

   - si e_l peut s'évaluer à $\#f$ concrètement, alors $\#f \in \alpha_l$;
   - si e_l peut s'évaluer à $(\lambda_{l_1}\mathrm{x}.\ e_{l_2})$ concrètement, alors $\lambda_{l_1} \in \alpha_l$;
   - si e_l peut s'évaluer à $(\mathrm{cons}_{l_1}\ v_1\ v_2)$ concrètement, alors $P_{l_1} \in \alpha_l$;
   - similairement pour les variables...

2. Comme il y a maintenant des paires abstraites dans l'analyse, il n'est plus nécessaire de faire propager expressément les données contenues dans les paires. Il y a maintenant une différence entre les paires et leur contenu.

3. Non seulement les résultats d'analyse nous permettent d'effectuer les analyses de flot de données conventionnelles mais ils nous permettent d'optimiser directement dans certains cas.

   Dans notre exemple précédent, les résultats nous permettent d'éliminer les tests de sûreté dans les appels e_1, e_2, e_5 et e_7, puisque les ensembles α_2, α_3, α_6 et α_8, respectivement, ne contiennent que des fonctions.

Deuxième analyse

Exemple:

Testons notre deuxième analyse sur ce programme.

$$
\begin{array}{l}
({}_1({}_2(\lambda_3 \mathrm{z}.\ (\mathrm{if}_4\ \mathrm{z}_5\ \mathrm{z}_6\ (\lambda_7 \mathrm{x}.\ \mathrm{x}_8))) \\
\quad (\mathrm{car}_9\ (\mathrm{pair?}_{10}\ (\mathrm{cons}_{11}\ \#f_{12}\ \#f_{13})))) \\
\ \#f_{14})
\end{array}
$$

Deuxième analyse

Exemple:

Testons notre deuxième analyse sur ce programme.

$$
\begin{array}{l}
({}_1({}_2(\lambda_3 \mathrm{z}.\ (\mathrm{if}_4\ \mathrm{z}_5\ \mathrm{z}_6\ (\lambda_7 \mathrm{x}.\ \mathrm{x}_8))) \\
\quad (\mathrm{car}_9\ (\mathrm{pair?}_{10}\ (\mathrm{cons}_{11}\ \#f_{12}\ \#f_{13})))) \\
\ \#f_{14})
\end{array}
$$

$$
\begin{array}{llll}
\alpha_1 = \{\#f\} & \alpha_5 = \{\#f\} & \alpha_9 = \{\#f\} & \alpha_{13} = \{\#f\} \\
\alpha_2 = \{\#f, \lambda_7\} & \alpha_6 = \{\#f\} & \alpha_{10} = \{\#f, P_{11}\} & \alpha_{14} = \{\#f\} \\
\alpha_3 = \{\lambda_3\} & \alpha_7 = \{\lambda_7\} & \alpha_{11} = \{P_{11}\} & \alpha_{\mathrm{z}} = \{\#f\} \\
\alpha_4 = \{\#f, \lambda_7\} & \alpha_8 = \{\#f\} & \alpha_{12} = \{\#f\} & \alpha_{\mathrm{x}} = \{\#f\}
\end{array}
$$

Deuxième analyse

Exemple: (suite...)

- Parmi les tests de sûreté présents dans les expressions e_1, e_2 et e_9, seul celui de e_2 peut être éliminé.
- Néanmoins, les tests de types présents dans les expressions e_4 et e_{10} ont un résultat prédéterminé et pourraient être optimisés.

Faiblesses de l'analyse:

- L'évaluation abstraite de $(\text{pair? } e)$ retourne toujours $\#f$ sans égard aux valeurs que peut produire e.
- L'évaluation abstraite de $(\text{if } e_{l_1} \ e_{l_2} \ e_{l_3})$ retourne la valeur de e_{l_2} et e_{l_3} sans tenir compte de la valeur de e_{l_1}.

Facile à corriger!

Troisième analyse

Génération des contraintes:

Si $e_l = \#f_l$, $\quad \alpha_l \supseteq \{\#f\}$

Si $e_l = \mathrm{x}_l$, $\quad \alpha_l \supseteq \alpha_{\mathrm{x}}$

Si $e_l = (\lambda_l \mathrm{x}.\ e_{l_1})$, $\quad \alpha_l \supseteq \{\lambda_l\}$

Si $e_l = ({}_l e_{l_1}\ e_{l_2})$, $\quad$ pour tout $\lambda_{l_3} \in \alpha_{l_1}$ t.q. $e_{l_3} = (\lambda_{l_3} \mathrm{x}.\ e_{l_4})$

$\alpha_{\mathrm{x}} \supseteq \alpha_{l_2}$

$\alpha_l \supseteq \alpha_{l_4}$

Si $e_l = (\mathrm{if}_l\ e_{l_1}\ e_{l_2}\ e_{l_3})$, $\quad$ si $\alpha_{l_1} - \{\#f\} \neq \emptyset$, $\quad \alpha_l \supseteq \alpha_{l_2}$ $\quad$ (1)

si $\#f \in \alpha_{l_1}$, $\quad \alpha_l \supseteq \alpha_{l_3}$

Si $e_l = (\mu_l \mathrm{x}.\ e_{l_1})$, $\quad \alpha_l \supseteq \alpha_{l_1}$

$\alpha_{\mathrm{x}} \supseteq \alpha_{l_1}$

Si $e_l = (\mathrm{cons}_l\ e_{l_1}\ e_{l_2})$, $\quad \alpha_l \supseteq \{P_l\}$

Si $e_l = (\mathrm{car}_l\ e_{l_1})$, $\quad$ pour tout $P_{l_2} \in \alpha_{l_1}$ t.q. $e_{l_2} = (\mathrm{cons}_{l_2}\ e_{l_3}\ e_{l_4})$

$\alpha_l \supseteq \alpha_{l_3}$

Si $e_l = (\mathrm{cdr}_l\ e_{l_1})$, $\quad$ pour tout $P_{l_2} \in \alpha_{l_1}$ t.q. $e_{l_2} = (\mathrm{cons}_{l_2}\ e_{l_3}\ e_{l_4})$

$\alpha_l \supseteq \alpha_{l_4}$

Si $e_l = (\mathrm{pair?}_l\ e_{l_1})$, $\quad$ soit $\pi = \{P_{l_2} \in \alpha_{l_1}\}$

$\alpha_l \supseteq \pi$ $\quad$ (2)

si $\alpha_{l_1} - \pi \neq \emptyset$, $\quad \alpha_l \supseteq \{\#f\}$

Troisième analyse

Commentaires:

1. Durant l'évaluation abstraite de $(\mathrm{if}_l\ e_{l_1}\ e_{l_2}\ e_{l_3})$,

   - la branche "then" e_{l_2} ne contribue à la valeur de l'expression que si le test e_{l_1} peut produire une valeur vraie et
   - la branche "else" e_{l_3} ne contribue à la valeur de l'expression que si le test peut produire une valeur fausse.

2. Durant l'évaluation abstraite de $(\mathrm{pair?}_l\ e_{l_1})$,

   - toutes les paires produites par e_{l_1} contribuent à la valeur de l'expression et
   - si e_{l_1} produit autre chose que des paires alors $\#f$ est ajoutée à la valeur de l'expression.

Exercice:

Appliquer la troisième analyse au programme de l'exemple précédent. Tous les tests de sûreté sont censés devenir redondants.

Troisième analyse

Exemple:

Testons notre troisième analyse sur ce programme.

$$
\begin{array}{l}
({}_1(\lambda_2 \mathrm{f}.\ ({\mathrm{if}}_3\ \#f_4 \\
\qquad\qquad ({}_5\mathrm{f}_6\ \#f_7) \\
\qquad\qquad ({}_8\mathrm{f}_9\ (\mathrm{cons}_{10}\ \#f_{11}\ \#f_{12}))))) \\
\quad (\lambda_{13}\mathrm{x}.\ (\mathrm{car}_{14}\ \mathrm{x}_{15})))
\end{array}
$$

Troisième analyse

Exemple:

Testons notre troisième analyse sur ce programme.

$$
\begin{array}{l}
({}_1(\lambda_2\mathrm{f}.\ (\mathrm{if}_3\ \#f_4 \\
\qquad\qquad ({}_5\mathrm{f}_6\ \#f_7) \\
\qquad\qquad ({}_8\mathrm{f}_9\ (\mathrm{cons}_{10}\ \#f_{11}\ \#f_{12}))))) \\
\quad (\lambda_{13}\mathrm{x}.\ (\mathrm{car}_{14}\ \mathrm{x}_{15})))
\end{array}
$$

$$
\begin{array}{llll}
\alpha_1 = \{\#f\} & \alpha_6 = \{\lambda_{13}\} & \alpha_{11} = \{\#f\} & \alpha_{\mathrm{f}} = \{\lambda_{13}\} \\
\alpha_2 = \{\lambda_2\} & \alpha_7 = \{\#f\} & \alpha_{12} = \{\#f\} & \alpha_{\mathrm{x}} = \{\#f, P_{10}\} \\
\alpha_3 = \{\#f\} & \alpha_8 = \{\#f\} & \alpha_{13} = \{\lambda_{13}\} & \\
\alpha_4 = \{\#f\} & \alpha_9 = \{\lambda_{13}\} & \alpha_{14} = \{\#f\} & \\
\alpha_5 = \{\#f\} & \alpha_{10} = \{P_{10}\} & \alpha_{15} = \{\#f, P_{10}\} &
\end{array}
$$

Troisième analyse

Exemple: (suite...)

- Le test de sûreté dans l'expression e_{14} n'est pas optimisable car les résultats d'analyse indiquent que x peut être liée à $\#f$.
- D'où vient cette valeur superflue? Elle vient du fait que l'appel $({}_5\mathrm{f}_6\ \#f_7)$ est évalué.
- Le fait que la valeur de la conditionnelle n'est pas influencée par la valeur de e_5 n'a pas de lien avec le fait que e_5 est *évaluée*.
- Pour corriger la situation, il faudrait une analyse qui ne causerait pas l'évaluation abstraite de e_5.

Faiblesse de l'analyse:

En règle générale, il faudrait une analyse qui évite de déclencher l'évaluation abstraite d'expressions qui n'ont pas besoin d'être évaluées.

Facile à réaliser!

Quatrième analyse

Spécifications:

- En plus des ensembles $\alpha_?$, cette analyse utilise des variables booléennes $\delta_?$.
- Il y a une variable δ_l pour chaque expression e_l du programme.
- Une variable δ_l est un commutateur contrôlant l'évaluation abstraite de l'expression e_l.
- Lorsque δ_l est vraie, e_l doit être évaluée; sinon e_l n'est pas évaluée.
- L'analyse fonctionne toujours en générant des contraintes pour le programme et ensuite en résolvant ces contraintes.
- On cherche toujours une plus petite solution aux contraintes.
- Si on considère "faux" comme étant plus petit que "vrai", une plus petite solution aux contraintes tend donc à laisser le plus possible de variables $\delta_?$ à "faux".
- Contrairement au calcul des ensembles $\alpha_?$, le calcul des variables booléennes se fait d'une expression vers les sous-expressions.

Quatrième analyse

Génération des contraintes:

Si $e_l = \#f_l, \quad \delta_l \Rightarrow \alpha_l \supseteq \{\#f\}$

Si $e_l = \mathrm{x}_l, \quad \delta_l \Rightarrow \alpha_l \supseteq \alpha_\mathrm{x}$

Si $e_l = (\lambda_l \mathrm{x}.\ e_{l_1}), \quad \delta_l \Rightarrow \alpha_l \supseteq \{\lambda_l\}$

$(\alpha_\mathrm{x} \neq \emptyset) \Rightarrow \delta_{l_1}$ (1)

Si $e_l = ({}_l e_{l_1}\ e_{l_2}), \quad \delta_l \Rightarrow \delta_{l_1}$

$\delta_l \Rightarrow \delta_{l_2}$

pour tout $\lambda_{l_3} \in \alpha_{l_1}$ t.q. $e_{l_3} = (\lambda_{l_3}\mathrm{x}.\ e_{l_4})$

$\alpha_\mathrm{x} \supseteq \alpha_{l_2}$

$\alpha_l \supseteq \alpha_{l_4}$

Si $e_l = (\mathrm{if}_l\ e_{l_1}\ e_{l_2}\ e_{l_3}), \quad \delta_l \Rightarrow \delta_{l_1}$

$(\alpha_{l_1} - \{\#f\} \neq \emptyset) \Rightarrow \delta_{l_2}$

$(\#f \in \alpha_{l_1}) \Rightarrow \delta_{l_3}$

$\alpha_l \supseteq \alpha_{l_2} \cup \alpha_{l_3}$ (2)

Si $e_l = (\mu_l \mathrm{x}.\ e_{l_1}), \quad \delta_l \Rightarrow \delta_{l_1}$

$\alpha_l \supseteq \alpha_{l_1}$

$\alpha_\mathrm{x} \supseteq \alpha_{l_1}$

...

Quatrième analyse

Génération des contraintes: (suite. . .)

Si $e_l = (\text{cons}_l\ e_{l_1}\ e_{l_2})$, $\quad \delta_l \Rightarrow \delta_{l_1}$
$\delta_l \Rightarrow \delta_{l_2}$
$\delta_l \Rightarrow \alpha_l \supseteq \{P_l\}$

Si $e_l = (\text{car}_l\ e_{l_1})$, $\quad \delta_l \Rightarrow \delta_{l_1}$
pour tout $P_{l_2} \in \alpha_{l_1}$ t.q. $e_{l_2} = (\text{cons}_{l_2}\ e_{l_3}\ e_{l_4})$
$\alpha_l \supseteq \alpha_{l_3}$

Si $e_l = (\text{cdr}_l\ e_{l_1})$, $\quad \delta_l \Rightarrow \delta_{l_1}$
pour tout $P_{l_2} \in \alpha_{l_1}$ t.q. $e_{l_2} = (\text{cons}_{l_2}\ e_{l_3}\ e_{l_4})$
$\alpha_l \supseteq \alpha_{l_4}$

Si $e_l = (\text{pair?}_l\ e_{l_1})$, $\quad \delta_l \Rightarrow \delta_{l_1}$
soit $\pi = \{P_{l_2} \in \alpha_{l_1}\}$
$\alpha_l \supseteq \pi$
si $\alpha_{l_1} - \pi \neq \emptyset$, $\quad \alpha_l \supseteq \{\#f\}$

Contrainte additionnelle sur le programme $E = e_1$, $\quad \delta_1 =$ vrai $\qquad$ (3)

Quatrième analyse

Commentaires:

1. Durant l'évaluation abstraite de $(\lambda_l \mathrm{x}.\ e_{l_1})$, ce n'est le fait d'évaluer e_l qui force l'évaluation de e_{l_1} mais plutôt le fait que la fonction résultante, λ_l, soit appelée.
2. Durant l'évaluation abstraite de $(\mathrm{if}_l\ e_{l_1}\ e_{l_2}\ e_{l_3})$, la contribution à α_l des deux branches est contrôlée indirectement via l'évaluation sélective desdites branches.
3. La contrainte additionnelle sert à forcer l'évaluation abstraite du programme, sans quoi l'évaluation n'aurait pas lieu ($\delta_1 = \text{faux}$).
4. L'analyse reste conservatrice.
5. Elle est relativement précise:
   - les trois types d'objets sont pris en compte;
   - seules les données pertinentes circulent;
   - seules les expressions devant être évaluée le sont.

Exercice:
Refaire l'analyse du programme précédent à l'aide la quatrième analyse.

Quatrième analyse

Question:

Est-ce que cette analyse peut être déjouée? (C'est-à-dire, peut-elle produire des résultats trop pessimistes?)

Réponse:

Oui, facilement. En fait, toute analyse statique conservatrice qui se termine à tout coup est forcément pessimiste dans certains cas. (Résultat indirect de l'informatique théorique.)

Quatrième analyse

Exemple:

$$
\begin{array}{l}
({}_1(\lambda_2 \mathrm{f}.\ ({}_3(\lambda_4 \mathrm{z}.\ ({}_5({}_6\mathrm{f}_7\ (\lambda_8 \mathrm{y}.\ \#f_9))\ \#f_{10}))} \\
\qquad\qquad (\mathrm{car}_{11}\ ({}_{12}\mathrm{f}_{13}\ (\mathrm{cons}_{14}\ \#f_{15}\ \#f_{16}))))) \\
\quad (\lambda_{17}\mathrm{x}.\ \mathrm{x}_{18}))
\end{array}
$$

Quatrième analyse

Exemple:

$$
\begin{array}{l}
({}_1(\lambda_2\mathrm{f}.\ ({}_3(\lambda_4\mathrm{z}.\ ({}_5({}_6\mathrm{f}_7\ (\lambda_8\mathrm{y}.\ \#f_9))\ \#f_{10}))\\
\qquad\qquad (\mathrm{car}_{11}\ ({}_{12}\mathrm{f}_{13}\ (\mathrm{cons}_{14}\ \#f_{15}\ \#f_{16})))))\\
\quad (\lambda_{17}\mathrm{x}.\ \mathrm{x}_{18}))
\end{array}
$$

$$
\begin{array}{llllllllll}
\delta_1\ \surd & \delta_2\ \surd & \delta_3\ \surd & \delta_4\ \surd & \delta_5\ \surd & \delta_6\ \surd & \delta_7\ \surd & \delta_8\ \surd & \delta_9\ \surd \\
\delta_{10}\ \surd & \delta_{11}\ \surd & \delta_{12}\ \surd & \delta_{13}\ \surd & \delta_{14}\ \surd & \delta_{15}\ \surd & \delta_{16}\ \surd & \delta_{17}\ \surd & \delta_{18}\ \surd
\end{array}
$$

$$
\begin{array}{llll}
\alpha_1 = \{\#f\} & \alpha_7 = \{\lambda_{17}\} & \alpha_{13} = \{\lambda_{17}\} & \alpha_{\mathrm{f}} = \{\lambda_{17}\} \\
\alpha_2 = \{\lambda_2\} & \alpha_8 = \{\lambda_8\} & \alpha_{14} = \{P_{14}\} & \alpha_{\mathrm{x}} = \{\lambda_8, P_{14}\} \\
\alpha_3 = \{\#f\} & \alpha_9 = \{\#f\} & \alpha_{15} = \{\#f\} & \alpha_{\mathrm{y}} = \{\#f\} \\
\alpha_4 = \{\lambda_4\} & \alpha_{10} = \{\#f\} & \alpha_{16} = \{\#f\} & \alpha_{\mathrm{z}} = \{\#f\} \\
\alpha_5 = \{\#f\} & \alpha_{11} = \{\#f\} & \alpha_{17} = \{\lambda_{17}\} & \\
\alpha_6 = \{\lambda_8, P_{14}\} & \alpha_{12} = \{\lambda_8, P_{14}\} & \alpha_{18} = \{\lambda_8, P_{14}\} &
\end{array}
$$

Quatrième analyse

Commentaires:

Constats:

- Selon l'analyse, e_5 et e_{11} ne seraient pas sécuritaires.
- Tous les résultats semblent aussi précis que possible sauf pour α_6 et α_{12}.

Comment expliquer les résultats pessimistes?

- Les **2** appels distincts à la fonction λ_{17} *dans l'évaluation concrète* sont simulés par **1** appel à la fonction λ_{17} *dans l'évaluation abstraite*.
- Dans cet appel abstrait, λ_{17} reçoit P_{14} *et* λ_8 et doit donc les retourner tous les deux.

En général, l'analyse simule l'ensemble des appels concrets à une fonction donnée, peu importe le nombre de ceux-ci, à l'aide d'un seul appel abstrait.

Cinquième analyse

Notions de base:

- Afin d'éviter de mêler différents appels concrets en un seul appel abstrait, on introduit les *contours*.
- Les contours sont les équivalents abstraits des contextes d'évaluation concrets (C^{α} et C^{β}).
- Par exemple, l'expression $(\mathrm{cons\ x\ y})$ n'a pas de valeur en soi; il faut un contexte d'évaluation qui spécifie éventuellement la valeur de x et de y pour que l'évaluation de $(\mathrm{cons\ x\ y})$ ait un sens.
- Un contexte où $\mathrm{x} = \#f$ et $\mathrm{y} = \#f$ n'est certainement pas équivalent à un autre où $\mathrm{x} = (\#f\ .\ \#f)$ et $\mathrm{y} = \#f$ et la valeur de $(\mathrm{cons\ x\ y})$ s'en trouve influencée.
- Les contours n'ont pas à être aussi précis que les contextes d'évaluation.
- Sous un contour, une variable peut valoir plusieurs choses à la fois.
- Toutefois, sous un contour i, l'ensemble des valeurs qui peuvent être liées à une variable n'est pas nécessairement le même que celles qui peuvent être liées à la variable sous un contour j.
- Les contours peuvent être dénotés par des numéros (par ex., 1 à 10), des étiquettes, des types, ou autres, et servent seulement de représentants pour des ensembles de contextes d'évaluation.
- Dans tous les cas, les contours doivent être en nombre fini, pour être sûr que l'analyse se termine.

Cinquième analyse

Caractéristiques:

- Nous choisissons d'utiliser les étiquettes du programme comme contours.
- L'évaluation abstraite sous un contour l indique que la fonction dont le corps en cours d'évaluation a été appelée depuis l'appel e_l.
- Plus précisément, une expression $e_{l'}$ s'évalue sous le contour l si la λ-expression $e_{l''}$ qui englobe immédiatement $e_{l'}$ a produit la fonction f et si f a été appelée depuis l'appel e_l.
- À titre d'exemple, une fonction f qui est appelée depuis e_2 n'a pas à agir comme lorsqu'elle est appelée depuis e_7. Dans un cas, le corps de f s'évalue sous le contour 2 et, dans l'autre, il s'évalue sous le contour 7.
- Certaines expressions ne font pas partie du corps d'aucune λ-expression (par ex., l'expression principale); on décrète qu'elles s'évaluent sous le contour $\top$.
- L'ensemble des contours que l'on va utiliser pour effectuer l'analyse d'un programme donné E est $\{l \mid e_l \text{ est un appel dans } E\} \cup \{\top\}$.

Cinquième analyse

Caractéristiques: (suite. . .)

Voici la description des variables utilisées pour effectuer l'analyse d'un programme.

- Les variables $\alpha_{l,k}$ contiennent l'ensemble des valeurs auxquelles peut s'évaluer l'expression e_l sous le contour k.
- Le nouvel indice k permet de ramasser un ensemble de valeurs différent pour l'évaluation de e_l sous chacun des contours possibles.
- Les variables $\delta_{l,k}$ indiquent si l'expression e_l doit être évaluée abstraitement sous le contour k.
- À nouveau, le nouvel indice k permet d'avoir un contrôle de l'évaluation indépendant pour chaque contour.
- Les variables $\beta_{\mathrm{x},l,k}$ contiennent l'ensemble des valeurs qui seraient lues si, en l'expression e_l et sous le contour k, on effectuait un référence à la variable x.
- Cette nouvelle suite de variables permet de canaliser la propagation des valeurs contenues dans les variables en tenant compte de la position dans le programme et du contour en vigueur.
- Il faut spécifier l'étiquette l de l'expression où se fait la référence dans $\beta_{\mathrm{x},l,k}$ car on ne réfère pas nécessairement à la même instance de la variable d'une expression à l'autre.
- Par exemple, étant donné le fragment de programme $(\lambda_5\mathrm{y}.\ (\lambda_6\mathrm{z}.\ e_7))$, les ensembles $\beta_{\mathrm{x},6,99}$ et $\beta_{\mathrm{x},7,99}$ sont reliés à la valeur de x sous le contour 99 mais, dans un cas, lorsqu'une fonction issue de e_5 est appelée et, dans l'autre cas, lorsqu'une fonction issue de e_6 est appelée, respectivement.

Cinquième analyse

Caractéristiques: (suite. . .)

Voici les données abstraites qui sont manipulées par l'analyse:

- La valeur faux est dénotée par $\#f$.
- Une paire créée à partir de $(\mathrm{cons}_l\ e_{l'}\ e_{l''})$ sous le contour k est dénotée $P_{l,k}$.
- Une fonction créée à partir de la λ-expression e_l sous le contour k est dénotée $\lambda_{l,k}$.
- L'indice supplémentaire k chez les paires abstraites permet de produire des paires distinctes en fonction du contour en vigueur au moment de leur création et ainsi leur permettre de contenir des valeurs différentes.
- Similairement, les fonctions abstraites peuvent mémoriser des environnement lexicaux distincts selon le contour sous lequel elles ont été créées.
- On ne conserve qu'une valeur faux abstraite puisqu'il n'existe qu'une valeur faux concrète.

Cinquième analyse

Génération des contraintes:

Les contraintes sont relativement semblables à celles de la quatrième analyse sauf pour la gestion des contours et des références aux variables.

Soit $E = e_1$ le programme à analyser. Soit $K = \{l \mid e_l \text{ est un appel dans } E\} \cup \{\top\}$. Soit V l'ensemble des variables apparaissant dans E.

On a la contrainte unique suivante: $\delta_{1,\top} = \text{vrai}$.

Pour chaque expression e_l de E et pour chaque $k \in K$, on a:

Si $e_l = \#f_l$, $\quad \delta_{l,k} \Rightarrow \alpha_{l,k} \supseteq \{\#f\}$
Si $e_l = \mathrm{x}_l$, $\quad \delta_{l,k} \Rightarrow \alpha_{l,k} \supseteq \beta_{\mathrm{x},l,k}$
Si $e_l = (\lambda_l \mathrm{x}.\ e_{l_1})$, $\quad \delta_{l,k} \Rightarrow \alpha_{l,k} \supseteq \{\lambda_{l,k}\}$
$\qquad\qquad\qquad\qquad\qquad\qquad (\beta_{\mathrm{x},l_1,k} \neq \emptyset) \Rightarrow \delta_{l_1,k}$ (1)
...

Cinquième analyse

Génération des contraintes: (suite...)

Si $e_l = ({}_l e_{l_1}\ e_{l_2})$, pour tout $\lambda_{l_3,k'} \in \alpha_{l_1,k}$ t.q. $e_{l_3} = (\lambda_{l_3}\mathrm{x}.\ e_{l_4})$
$\quad$ pour tout $v \in V$, $\beta_{\mathrm{v},l_4,l} \supseteq \beta_{\mathrm{v},l_3,k'}$
$\quad\quad \beta_{\mathrm{x},l_4,l} \supseteq \alpha_{l_2,k}$ (2)
$\quad\quad \alpha_{l,k} \supseteq \alpha_{l_4,l}$

Si $e_l = (\mathrm{if}_l\ e_{l_1}\ e_{l_2}\ e_{l_3})$, $\delta_{l,k} \Rightarrow \delta_{l_1,k}$
$\quad (\alpha_{l_1,k} - \{\#f\} \neq \emptyset) \Rightarrow \delta_{l_2,k}$
$\quad (\#f \in \alpha_{l_1,k}) \Rightarrow \delta_{l_3,k}$
$\quad \alpha_{l,k} \supseteq \alpha_{l_2,k} \cup \alpha_{l_3,k}$

Si $e_l = (\mu_l \mathrm{x}.\ e_{l_1})$, $\alpha_{l,k} \supseteq \alpha_{l_1,k}$
$\quad \beta_{\mathrm{x},l_1,k} \supseteq \alpha_{l_1,k}$

Si $e_l = (\mathrm{cons}_l\ e_{l_1}\ e_{l_2})$, $\delta_{l,k} \Rightarrow \alpha_{l,k} \supseteq \{P_{l,k}\}$

Si $e_l = (\mathrm{car}_l\ e_{l_1})$, pour tout $P_{l_2,k'} \in \alpha_{l_1,k}$ t.q. $e_{l_2} = (\mathrm{cons}_{l_2}\ e_{l_3}\ e_{l_4})$
$\quad \alpha_{l,k} \supseteq \alpha_{l_3,k'}$

...

Cinquième analyse

Génération des contraintes: (suite. . .)

Si $e_l = (\mathrm{cdr}_l\ e_{l_1})$, pour tout $P_{l_2,k'} \in \alpha_{l_1,k}$ t.q. $e_{l_2} = (\mathrm{cons}_{l_2}\ e_{l_3}\ e_{l_4})$

$\alpha_{l,k} \supseteq \alpha_{l_4,k'}$

Si $e_l = (\mathrm{pair?}_l\ e_{l_1})$, soit $\pi = \{P_{l_2,k'} \in \alpha_{l_1,k}\}$

$\alpha_{l,k} \supseteq \pi$

si $\alpha_{l_1,k} - \pi \neq \emptyset$, $\alpha_{l,k} \supseteq \{\#f\}$

Si e_l a un parent e_{l_p},

si e_{l_p} n'est pas $(\lambda_{l_p}\mathrm{x}.\ e_l)$ ni $(\mathrm{if}_{l_p}\ e_{l_1}\ e_{l_2}\ e_{l_3})$,

$\delta_{l_p,k} \Rightarrow \delta_{l,k}$ (3)

si e_{l_p} n'est pas $(\lambda_{l_p}\mathrm{x}.\ e_l)$,

pour tout $v \in V$, $\beta_{\mathrm{v},l,k} \supseteq \beta_{\mathrm{v},l_p,k}$ (4)

Cinquième analyse

Commentaires:

1. Le k dans la première contrainte n'est pas nécessairement relié au k dans la deuxième. En d'autres mots, le corps d'une fonction peut s'évaluer sous d'autres contours que la λ-expression ayant produit la fonction.

2. Ce sont les seules contraintes où on change le contour courant. Comme le contour sert à indiquer d'où a été appelée la fonction dont le corps s'évalue, le fait d'appeler une fonction $\lambda_{l_3,k'}$ à partir de l'appel e_l force l'évaluation de son corps sous le contour l.

3. La gestion du contrôle de l'évaluation (les $\delta_{l,k}$) est maintenant centralisée. Une expression est évaluée sous un contour k dès que son parent l'est aussi. Il y a exception lorsque le parent est une λ-expression ou une conditionnelle car l'évaluation des sous-expressions de ce genre d'expressions obéit à d'autres règles.

4. La propagation des variables de l'environnement lexical est centralisée. La valeur d'une variable en une certaine expression et sous un certain contour est la même que chez son parent sous le même contour sauf lorsque le parent est une λ-expression. Dans ce cas, on simule le fait que les variables sont *mémorisées* par la fermeture.

Cinquième analyse

Exercice:

Refaire l'analyse du programme précédent à l'aide de la cinquième méthode d'analyse. Toutes les expressions devraient apparaître comme étant sécuritaires.

Exemple:

$$({}_1(\lambda_2 \mathrm{i}.\ ({}_3({}_4\mathrm{i}_5\ \mathrm{i}_6)\ \#f_7))\\ \quad (\lambda_8 \mathrm{x}.\ ({}_9(\lambda_{10}\mathrm{y}.\ \mathrm{x}_{11})\ \#f_{12})))$$

[Résultats trop volumineux!]

Ceux-ci devraient montrer que e_3 n'est pas sécuritaire car $\#f$ pourrait être appelé.

Analyses de flot de contrôle

Commentaires:

Même la meilleure analyse de notre cours est facile à mêler.

Toutefois, sur des programmes typiques, les résultats commencent à être de qualité acceptable.

Évidemment, on aurait pu développer des analyses encore plus puissantes:

- ② On pourrait utiliser des contours à deux étiquettes: celle d'où s'est fait l'appel à la fonction en cours d'évaluation et celle d'où s'est fait l'appel menant à l'invocation de la fonction appelante. + coûteuse. + puissante.
- On pourrait généraliser cette idée à l'utilisation de n étiquettes (famille d'analyses).
- On pourrait utiliser des types comme contours. Par exemple, le contour est le type de la valeur contenue dans le paramètre de la fonction en cours d'évaluation; i.e. un parmi $\{\mathrm{Faux}, \mathrm{Paire}, \mathrm{Closure}\}$.

  Exemple: Dans $\ldots(\lambda_{18}\mathrm{x}.\ (\lambda_{19}\mathrm{y}.\ e_{20}))\ldots$, l'expression e_{20} peut s'évaluer sous le contour F, P ou C selon la valeur contenue dans 'y'. Si $e_{20} = (\mathrm{if}_{20}\ \mathrm{y}_{21}\ e_{22}\ e_{23})$, le test sera nécessairement vrai sous les contours P et C mais faux sous le contour F. Si, au lieu, $e_{20} = (\mathrm{if}_{20}\ \mathrm{x}_{21}\ e_{22}\ e_{23})$, il faut lire 'x' et voir ensuite.

Analyses de flot de contrôle

- ③ Pourquoi se limiter au type du paramètre? On pourrait avoir des contours qui indiquent le type de *chacune* des variables visibles; i.e. un élément parmi $\{\mathrm{F}, \mathrm{P}, \mathrm{C}\}$ par variable. Par exemple, dans

$$(_1(\lambda_2 \mathrm{x}.\ (\lambda_3 \mathrm{y}.\ (\lambda_4 \mathrm{z}.\ e_5)))\\ (\lambda_6 \mathrm{v}.\ e_7))$$

  e_7 s'évalue sous F, P ou C mais e_5 s'évalue sous FFF, FFP, FFC, FPF, etc. (27 combinaisons)

- ① Pourquoi se limiter à un type superficiel? Une paire P_l ou $P_{l,k}$ n'est peut-être pas assez précise. On pourrait modéliser les paires avec le type des valeurs contenues dans les deux champs. Par exemple: $P^{\mathrm{F},\mathrm{C}}$, $P^{\mathrm{P},\mathrm{F}}$, $P^{\mathrm{C},\mathrm{C}}$, ... Pareil pour les fermetures.

- Pourquoi se limiter à une connaissance à deux niveaux des données? On pourrait généraliser à n niveaux.

Les analyses plus puissantes sont généralement plus coûteuses:

- Par un facteur constant. ①
- Plus que par un facteur constant mais néanmoins en restant polynômiales. ②
- En devenant exponentielles. ③

Analyses de flot de contrôle

Quelle analyse choisir?

- Dépend des besoins en rapidité du compilateur.
- Dépend des besoins en optimisation du code généré.

Sachant le degré d'optimisation versus la rapidité désirés, comment choisir la meilleure analyse?

- Dépend du programme:
  - Complexité des appels (flot de contrôle)
  - Complexité des données (par exemple, y a-t-il des listes de fonctions lesquelles retournent des listes de fonctions?)
- Le choix étant quasi-infaisable au moment d'écrire le compilateur, on peut se tourner vers des approches adaptatives: l'analyse découvre elle-même, pour le programme donné, la "bonne" combinaison contours/données abstraites.

Lien analyse-optimisation

Revoyons le lien entre l'analyse de flot de contrôle (et de types) et les optimisations (en se référant aux résultats produits par la quatrième analyse).

Tests de sûreté à éliminer:

- L'appel $({}_{l}e_{l_1}\ e_{l_2})$ est optimisable si $\#f \notin \alpha_{l_1}$ et $\{P_{l'} \in \alpha_{l_1}\} = \emptyset$.
- L'extraction $(\mathrm{car}_{l}\ e_{l_1})$ est optimisable si $\alpha_{l_1} = \{P_{l'} \in \alpha_{l_1}\}$.

Appels de fonctions directs et *inlining*: si on a l'appel $({}_{l}e_{l_1}\ e_{l_2})$ et que $\alpha_{l_1} = \{\lambda_{l'}\}$, alors on peut générer un saut direct à $\lambda_{l'}$ ou intégrer ("*inliner*") le corps de $\lambda_{l'}$.

Créations inutiles de fermetures à éliminer:

- Si toutes les variables capturées par $(\lambda_{l}\mathrm{x}.\ e)$ sont des constantes, on n'a qu'à créer la fermeture une fois.
- Si les seules *utilisations* des fermetures provenant de $(\lambda_{l}\mathrm{x}.\ e)$ sont des appels et que ces derniers savent qu'ils appellent des fermetures provenant de λ_{l}, il n'est pas nécessaire d'allouer les fermetures, il suffit de propager leurs environnements.

Lien analyse-optimisation

Élimination du code mort: si $\delta_l = \text{faux}$, alors e_l est du code mort.

"Déterminisation" des tests de types ordinaires:

- Si $\#f \notin \alpha_{l_1}$, on peut compiler $(\text{if}_l\ e_{l_1}\ e_{l_2}\ e_{l_3})$ comme $(\text{begin}\ e_{l_1}\ e_{l_2}\ \text{end})$.
- Si $\alpha_{l_1} = \{\#f\}$, on peut compiler $(\text{if}_l\ e_{l_1}\ e_{l_2}\ e_{l_3})$ comme $(\text{begin}\ e_{l_1}\ e_{l_3}\ \text{end})$.
- Si $\alpha_{l_1} = \emptyset$, on peut compiler $(\text{if}_l\ e_{l_1}\ e_{l_2}\ e_{l_3})$ comme e_{l_1}.
- Similairement pour $(\text{pair?}_l\ e_{l_1})$.

Autres analyses

Si on considère un langage plus complet et que celui-ci inclut les nombres, on peut vouloir obtenir une idée des valeurs manipulées.

- Analyse: pour chaque variable ou expression, on calcule un intervalle $[\phi, \psi]$ qui inclut au moins toutes les valeurs possibles.
- Optimisation 1: omettre le test de sûreté dans la division x/y si $0 \notin [\phi_{\mathrm{y}}, \psi_{\mathrm{y}}]$.
- Optimisation 2: omettre le test de sûreté $\mathrm{i} \geq 0$ dans la lecture dans un tableau $\mathrm{A}[\mathrm{i}]$.
- Optimisation 3: omettre le test de sûreté $\mathrm{i} < l$ dans la lecture dans un tableau $\mathrm{A}[\mathrm{i}]$ si 'A' a une longueur de l cases.

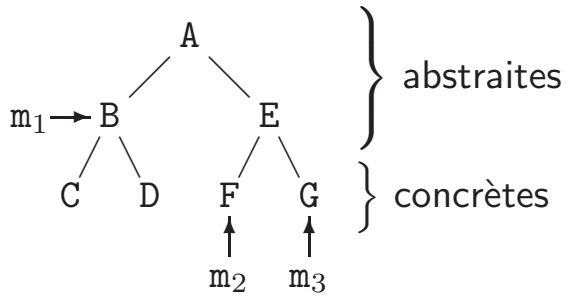
Autres analyses

Considérons un langage orienté-objet, i.e. incluant les objets, les classes et l'héritage.

- Analyse: pour chaque expression, chaque énoncé et chaque variable, on veut connaître quelles sont les références qui peuvent s'y trouver. Par exemple, les résultats d'analyse pourraient indiquer que la variable 'x' peut contenir des références vers des objets de classes `A`, `C` et `F` et une référence bidon (`NULL`).

- Optimisation 1. Dans l'expression `o.m()`, si on sait que 'o' ne peut pas contenir une référence bidon, on peut éliminer le test de sûreté.

- Optimisation 2. On peut diminuer le coût de la résolution des appels de méthodes virtuelles. Par exemple, considérons l'ensemble de classes suivant:



où chaque classe concrète fournit une version de la méthode 'm', l'appel `o.m()` et la déclaration de 'o' comme étant de classe 'A'. Si l'analyse détermine que les seules références que peut contenir 'o' sont vers des objets de classes 'C' et 'D', alors le passage de message invoque nécessairement la méthode m_1.

Autres analyses

Le fait de connaître la durée de vie des objets par rapport à l'évaluation des expressions environnantes peut permettre de sélectionner une meilleure méthode d'allocation.

- Analyse d'échappement: fournit une réponse conservatrice à la question: "Soit la fonction λ_l qui, durant son invocation, alloue l'objet X. Est-ce que X peut être encore vivant après que l'appel à la fonction se termine?"
- Optimisation: on peut effectuer l'allocation de X sur la pile plutôt que sur le tas.

Autres analyses

Une analyse de vivacité permet de déterminer, à la compilation, si un objet est certainement mort à un certain point de l'exécution du programme.

- Analyse: soit 'v' une variable contenant un objet X, soit l un point du programme; l'analyse tente de répondre à la question: "Est-ce qu'en le point l, l'objet X est dorénavant abandonné?"
- Optimisation 1: dans l'affirmative, on peut le désallouer sur le champ (tout dépendant du type de GC utilisé).
- Optimisation 2: dans l'affirmative et en supposant qu'un objet de même taille doive être alloué (immédiatement) après, on peut recycler X et écraser son contenu plutôt que de l'abandonner et allouer un nouvel objet neuf.

Autres analyses

La représentation des objets n'a pas à respecter quelque convention que ce soit pourvu que le programme s'évalue *apparemment* de la façon attendue.

- Analyse: soit P_l une paire créée en l'expression e_l, qui est passée en divers points du programme et dont les champs sont plus tard extraits à l'aide de 'car' et de 'cdr'; l'analyse vise à déterminer si la paire a une raison d'être en elle-même ou bien si elle sert temporairement de contenant pour deux valeurs.

- Optimisation: on peut éliminer la création de la paire en tant que telle et simplement propager directement les deux valeurs qu'elle est censée contenir. Par exemple, l'expression

  $$((\lambda \mathrm{p}.\ ((\mathrm{f}\ (\mathrm{car}\ \mathrm{p}))\ (\mathrm{cdr}\ \mathrm{p})))\ (\mathrm{cons}\ e'\ e''))$$

  peut être remplacée par l'"expression"

  $$((\lambda \mathrm{p_a}\ \mathrm{p_d}.\ ((\mathrm{f}\ \mathrm{p_a})\ \mathrm{p_d}))\ \mathrm{p_a}\ \mathrm{p_d})$$

Autres analyses

L'évaluation partielle est une optimisation très puissante et générale.

- Optimisation visant à produire automatiquement du code spécialisé en fonction de valeurs déjà connues.
- Par exemple: l'expression $(\text{if } \#f \ e_{l_1} \ e_{l_2})$ est un cas évident où l'évaluation partielle est applicable; l'expression peut être remplacé par l'expression e_{l_2}.
- Autre exemple: considérons l'expression $\ldots(\lambda_{l_1}\text{x}.\ (\lambda_{l_2}\text{y}.\ (\text{if x } e_{l_3} \ e_{l_4})))\ldots$.
  - 'x' est connue dans un premier temps et est mémorisée par une fermeture λ_{l_2}.
  - À chaque invocation de λ_{l_2}, on doit refaire un test sur la valeur de 'x' dont l'issue est connue d'avance.
  - Par évaluation partielle, on peut remplacer l'expression par

$$\ldots(\lambda_{l_1}\text{x}.\ (\text{if x } (\lambda_{l_2}\text{y}.\ e_{l_3}) \\ (\lambda_{l_2}\text{y}.\ e_{l_4})))\ \ldots$$

- Autre exemple: si l'expression est plutôt $\ldots(\lambda_{l_1}\text{x}.\ (\lambda_{l_2}\text{y}.\ (\text{if } (\text{f x}) \ e_{l_3} \ e_{l_4})))\ldots$, peut-on la remplacer par l'expression suivante?

$$\ldots(\lambda_{l_1}\text{x}.\ (\text{if } (\text{f x}) \ (\lambda_{l_2}\text{y}.\ e_{l_3}) \\ (\lambda_{l_2}\text{y}.\ e_{l_4})))\ \ldots$$