Heuristique d'inlining complexe

Heuristique d'inlining complexe

Adrien Simonnet

Sorbonne Université

Avril - Septembre 2023

Table of Contents

Introduction

Langage source

Le langage source est un sous-ensemble fonctionnel d'OCaml.

- Lambda-calcul
- Opérations élémentaires
- Fermetures (mutuellement) récursives
- Types Somme et filtrage par motifs

Exemple

```
type int_list = Nil | Cons of int * int_list
let rec map = fun f -> fun l -> match l with
| Nil -> Nil
| Cons (x, ls) -> Cons (f x, map f ls)
in map (fun x -> x + 10) (Cons (1, Cons (2, Nil)))
```

Lexique

- Sous-ensemble de celui d'OCaml
- Supporter les fonctionnalités intéressantes
- Analyse lexicale réalisée par OCamllex

Arbre de Syntaxe Abstraite (AST)

- La grammaire est la même que celle d'OCaml
- Les identifiants sont des chaînes de caractères
- Déconstruction de termes uniquement possible dans les filtrages par motif
- Opérations limitées aux entiers
- Pas de vérification de typage
- Analyse syntaxique réalisée par Menhir

Résolution des noms

- AST légèrement différent
- Les identifiants sont des entiers naturels uniques
- Index pour le nom des constructeurs
- Autorise les variables libres

Conversion

$$\mathbb{E}_{ast} \times (\mathbb{V}_{ast} \mapsto \mathbb{V}) \times (\mathbb{T}_{ast} \mapsto \mathbb{T}) \vdash_{\text{ast'}} \mathbb{E} \times (\mathbb{V} \mapsto \mathbb{V}_{ast}) \times (\mathbb{V}_{ast} \mapsto \mathbb{V})$$

Exemple

$$e\ A\ C \vdash_{\text{ast'}} e'\ S\ L$$

Graphe de flot de contrôle

- Ensemble de basic blocks clos
- Chaque type de bloc a sa sémantique
- Expressions construitent par des valeurs
- Valeurs déclarées par un identifiant unique
- Instruction = déclaration ou branchement
- Basic block = suite de déclarations puis branchement

Conversion

$$\mathbb{E}_{ast'} \times \mathbb{V} \times \mathcal{P}(\mathbb{V}) \times \mathbb{I} \vdash_{\text{cfg}} \mathbb{I} \times \mathcal{P}(\mathbb{V}) \times (\mathbb{P} \mapsto (\mathbb{B} \times \mathbb{I}))$$

Exemple

$$e\ v\ V\ i \vdash_{\text{cfg}} e'\ V_e\ B_e$$

Le CFG peut contenir des alias.

Exemple

$(\text{Var } x_1)\ \overline{0}\ \emptyset\ (\text{Return } \overline{0}) \vdash_{\text{cfg}} (\text{let } \overline{0} = x_1 \text{ in Return } \overline{0})\ \{x_1\}\ \emptyset$

La passe de nettoyage supprime tous les alias.

Exemple

$\text{let } \overline{0} = x_1 \text{ in Return } \overline{0} \to \text{Return } x_1$

Graphe de flot de contrôle

Spécialisation

La spécialisation consiste à dupliquer des blocs.

- Améliore la précision de l'analyse
- Première étape de l'inlining

La copie doit :

- Conserver les invariants du CFG
- Modifier les appels directs

Les blocs sont choisis selon leur taille.

Analyse de valeurs par interprétation abstraite

Etape la plus compliquée et probablement la plus importante.

- Rendre les appels directs pour inliner
- Informations cruciales pour les heuristiques

L'analyse se fait par point d'allocation.

- Garanties de terminaison
- Autorise la récursivité

L'analyse a lieu sur le CFG.

- Importance des invariants
- Disposer des blocs

Les entiers sont représentés de la manière la plus simple qui soit, c'est à dire des singletons munis de Top.

Abstraction

Top : $\mathbb{I}$
Singleton : $\mathbb{Z} \mapsto \mathbb{I}$

L'union de deux entiers donne toujours Top sauf lorsqu'il s'agit de deux singletons de même valeur.

Union

$$x \sqcup y = \begin{cases} \text{Singleton } i \text{ si } x = y = \text{Singleton } i \\ \text{Top sinon} \end{cases}$$

Le domaine pour les fermetures est un environnement d'identifiant vers contexte, où l'identifiant correspond au pointeur de fonction, et le contexte correspond aux variables libres.

Abstraction

$\mathbb{F} := \mathbb{P} \mapsto \mathcal{P}(\mathcal{P}(\mathbb{V}))$ (fermeture)

L'union de deux fermetures consiste à conserver les entrées distinctes et d'unir les points d'allocations des entrées communes.

Union

$$x \sqcup y = z \to \begin{cases} \{x(z)(i) \cup y(z)(i), i \in x(z) \text{ et } i \in y(z)\} \text{ si } z \in \mathcal{D}(x) \text{ et } z \in \mathcal{D}(\\ x(z) \text{ si } z \in \mathcal{D}(x) \\ y(z) \text{ si } z \in \mathcal{D}(y) \end{cases}$$

Le domaine pour les unions taggées est un environnement d'identifiant vers contexte, où l'identifiant correspond au tag, et le contexte correspond au contenu de l'union.

Abstraction

$\mathbb{C} := \mathbb{T} \mapsto \mathcal{P}(\mathbb{V})^*$ (union taggée)

L'union de deux valeurs taggées consiste à conserver les entrées distinctes et d'unir les points d'allocations des entrées communes.

Union

$$x \sqcup y = z \to \begin{cases} (x(z)_i \cup y(z)_i)_{i=1}^{i=n} & \text{si } z \in \mathcal{D}(x) \text{ et } z \in \mathcal{D}(y) \text{ et } |x(z)| = |y(z)| \\ x(z) \text{ si } z \in \mathcal{D}(x) \\ y(z) \text{ si } z \in \mathcal{D}(y) \end{cases}$$

Une valeur abstraite est un entier, une fermeture ou une valeur taggée.

Abstraction

IntDomain : $\mathbb{I} \mapsto \mathbb{A}$
ClosureDomain : $\mathbb{F} \mapsto \mathbb{A}$
ConstructorDomain : $\mathbb{C} \mapsto \mathbb{A}$

Détection de motifs pour détecter des récursions.

Exemple

ABCBC a un motif BC de taille 2 et sera remplacée par ABC

Conserver uniquement les n derniers appels (n-CFA).

Exemple

En 1-CFA, la pile d'appels ABCBC sera remplacée par C

Analyse de valeurs par interprétation abstraite

Terminaison

Quelques idées justifiant la terminaison de l'analyse.

- Identifiants jamais générés
- Un point d'allocation est un identifiant
- L'union de deux valeurs converge
- L'union de deux usines converge
- Abstraction de la pile d'appels

Définitions

L'algorithme d'analyse prend une liste des blocs à analyser (pointeur, en-tête, pile et usine), une fonction d'abstraction de la pile, l'ensemble des blocs du programme, l'ensemble des usines pour chaque contexte de chaque bloc déjà analysé et renvoie en-tête et usine pour chaque bloc.
analyse : $(\mathbb{P} \times \overline{\mathbb{B}} \times \mathbb{S} \times \mathbb{U})^* \times (\mathbb{S} \mapsto \mathbb{S}) \times (\mathbb{P} \mapsto \mathbb{B}) \times (\mathbb{P} \mapsto ((\mathbb{S} \times \overline{\mathbb{B}}) \mapsto \mathbb{U})) \mapsto (\mathbb{P} \mapsto (\overline{\mathbb{B}} \times \mathbb{U}))$
L'analyse d'un bloc se fait par la fonction '$analyse_d\, e_b\, loc$' qui à partir d'un bloc, d'une pile, d'un environnement et d'une usine renvoie une liste de blocs à analyser.
'$env_d\, e_b\, loc$' est une fonction qui génère l'environnement à partir de l'en-tête d'un bloc et les valeurs abstraites qui lui sont passées.

Algorithme

Algorithm 1: Analyse du programme

Input : l pile_abs B U

```
// Il ne reste plus aucun bloc à analyser.
1 if l est vide then
2 |   return U telle que pour chaque bloc sa valeur soit l'union des
  |       usines et paramètres de chaque contexte de pile de ce bloc.
  |       // Tous les contextes de pile sont fusionnés.
3 else
4 |   p, b, s, u ← hd(l) // Le premier bloc à analyser.
5 |   l' ← tl(l) // Les autres blocs à analyser.
6 |   s̄ ← pile_abs(s) // Abstraction de la pile.
7 |   c ← (s̄, b) // Le contexte (pile et en-tête).
  |
  |   // Ce bloc a déjà été analysé.
8 |   if p ∈ 𝒟(U) then
9 |   |   U_p ← U[p] // Les usines associées à ce bloc.
  |   |
  |   |   // Ce contexte a déjà été analysé pour ce bloc.
```

Terminaison

Quelques idées justifiant la terminaison de l'analyse.

- Identifiants jamais générés
- Un point d'allocation est un identifiant
- L'union de deux valeurs converge
- L'union de deux usines converge
- Abstraction de la pile d'appels

Propagation

La propagation modifie le CFG pour y faire apparaître les résultats de l'analyse.

- Expressions transformées en constantes
- Éliminations de branches
- Appels indirects transformés en appels directs

CFG exécutable

Représentation plus bas-niveau que le CFG.

- Concrétise des traits de langage (n-uplets)
- Unifie les blocs et branchements
- Fixe la sémantique des sauts
- Explicite les opérations sur la pile

Conversion

$\mathbb{B}_{cfg} \times \mathbb{I} \vdash_{\text{cfg'}} \mathbb{B} \times (\mathbb{P} \mapsto \mathbb{B})$

Exemple

$a\ i \vdash_{\text{cfg'}} a'\ i'\ B$

CFG exécutable

Inlining

Intégrer le contenu d'un bloc à la place d'un appel direct.

- Renomme les arguments
- Modifie les piles
- Autorise les sauts vers l'intérieur d'une fonction

Sont actuellement inlinés tous les blocs appelés exactement 1 fois.

Conversion

$\mathbb{B}_{cfg} \times \mathbb{I} \vdash_{\text{cfg'}} \mathbb{B} \times (\mathbb{P} \mapsto \mathbb{B})$

Exemple

$a\ i \vdash_{\text{cfg'}} a'\ i'\ B$

CFG exécutable

Interprétation

Interpréter le CFG avec des tests.

- Teste la validité des transformations
- Réalise des benchmarks

Conclusion

Importance des représentations intermédiaires.

- Avoir des transformations simples
- Fixer une sémantique naturelle
- Mieux vaut trop de représentations que pas assez

L'interprétation abstraite, c'est compliqué.

- Difficultés pour traiter les piles
- Complexité (certainement) exponentielle
- Résultats satisfaisants

Résultats sur l'inlining.

- Toujours inliner les petits blocs
- Pas toujours intéressant pour les blocs de moyenne taille
- Inintéressant pour les plus gros blocs sans autre heuristique

D'autres expérimentations à mener.

- De plus gros tests
- Ajouter de nouveaux traits de langage
- Automatiser les tests