Heuristique d'inlining complexe

Heuristique d'inlining complexe

Adrien Simonnet

Sorbonne Université

Avril - Septembre 2023

Plan

Introduction

OCamlPro

- Fondée en 2011
- Issue de l'INRIA
- Spécialisée dans les langages de programmation

Équipe flambda

- Vincent Laviron et Pierre Chambart
- Optimisations dans le compilateur OCaml

Optimisation centrale d'un compilateur

Avantages

- Améliore significativement les performance
- Permet aux autres optimisations de s'activer

Coûts

- Temps de compilation
- Taille des exécutables

Nécessité de trouver des heuristiques

Sous-ensemble du noyau 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)))
```

Analyse lexicale

Lexique identique à celui d'OCaml

- Uniquement les fonctionnalités intéressantes

Jetons générés par OCamllex

Analyse syntaxique

Grammaire (presque) identique à celle d'OCaml

- Pas de filtrage par motif "profond"
- Pas de valeurs récursives
- Pas de types

AST généré par Menhir

Résolution des noms

Rafraîchissement de l'AST

- Les variables deviennent des identifiants uniques
- Index pour le nom des constructeurs
- Autorise les variables libres

Conversion

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

Exemple

Fun("x", Var "y") $\emptyset$ $\emptyset \vdash$ Fun(0, Var 1) $\{0 \rightarrow$ "x"$\}$ $\{$"y" $\rightarrow 1\}$

Exemple

Cons("Some", ["x"]) $\{$"x" $\rightarrow 0\}$ $\{$"Some" $\rightarrow 0\}$ $\vdash$ Cons(0, [0]) $\emptyset$ $\emptyset$

Graphe de flot de contrôle

- Ensemble de basic blocks clos
- Chaque type de bloc a sa sémantique
- Expressions construisent 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

$$\text{Fun}(1, \text{Var } 1)\ 0\ \emptyset\ e \vdash \text{Let}(0, \text{Clos}(0, \emptyset), e)\ \emptyset\ \{0 \to \text{Clos}([1], \emptyset, \text{Return } 1)\}$$

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

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

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

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

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

Union

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

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

$\mathbb{C} \coloneqq \mathbb{T} \mapsto \mathcal{P}(\mathbb{V})^*$ (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

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

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

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

Idées justifiant la terminaison de l'analyse

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

CFG exécutable

Représentation plus bas-niveau que le CFG

- Concrétise les 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

Faire apparaître les résultats de l'analyse

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

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 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

Interpréter le CFG

- Tester la validité des transformations
- Réaliser des benchmarks

Conclusion

Importance des représentations intermédiaires

- Transformations simples
- Sémantique riche d'informations

L'interprétation abstraite c'est compliqué

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

Résultats sur l'inlining

- Toujours inliner les petits blocs
- D'autres heuristiques nécessaires pour les plus gros blocs

Expérimentations à mener

- Tester sur de vrais programmes
- Exploiter les résultats