Heuristiques d'inlining complexe

Adrien Simonnet

Sorbonne Université

Avril - Septembre 2023

Introduction

OCamlPro

- Fondée en 2011 par un chercheur de l'INRIA
- Spécialisée dans les langages de programmation

Équipe flambda

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

Optimisation cruciale pour beaucoup de compilateurs

Avantages

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

Coûts

- Temps de compilation
- Taille des exécutables

Nécessité de trouver des heuristiques

Introduction

Langage jouet

Sous-ensemble du noyau fonctionnel d'OCaml

- Lambda-calcul
- Opérations élémentaires
- Fonctions (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

- Fonctions unaires
- Filtrage par motif simple
- Pas de typage

AST généré par Menhir

Résolution des noms

Rafraîchissement de l'AST

- Numéro unique pour les variables
- Index pour le nom des constructeurs
- Autorise les variables libres

Exemple

$\text{Cons}(\text{“Some”}, [\text{“x”}])\ \{\text{“x”} \to 0\}\ \{\text{“Some”} \to 0\} \vdash \text{Cons}(0, [0])\ \emptyset\ \emptyset$

Exemple

$\text{Fun}(\text{“x”}, \text{Var “}y\text{”})\ \emptyset\ \emptyset \vdash \text{Fun}(0, \text{Var } 1)\ \{0 \to \text{“x”}\}\ \{\text{“y”} \to 1\}$

Graphe de flot de contrôle

Tranforme l'AST en basic blocks clos

- Expressions construisent des valeurs
- Déclaration = identifiant unique pour chaque expression
- Branchement = sauter d'un bloc vers un autre
- Instruction = déclaration ou branchement
- Basic block = suite de déclarations puis branchement
- Bloc associé à un branchement

Exemple

$$\text{Fun}(x, \text{Var } x)\ v\ \emptyset\ e \vdash \text{Let}(v, \text{Clos}(p, \emptyset), e)\ \emptyset\ \{p \to \text{Clos}([x], \emptyset, \text{Return } x)\}$$

Graphe de flot de contrôle

Spécialisation

Duplication de certains blocs

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

Contraintes :

- Conserver les invariants du CFG
- Modifier les appels directs

Heuristiques de décision

- Blocs choisis selon leur taille

Analyse de valeurs par interprétation abstraite

Étape la plus compliquée et probablement la plus importante

- Transformer les appels indirects en appels directs
- Informations cruciales pour les heuristiques

Analyse par points d'allocation

- Garanties de terminaison
- Gère la récursivité

Analyse sur le CFG

- Importance des invariants
- Méthode d'analyse standard par itérations successives

Entier, fermeture ou valeur taggée

Entier

Top : $\mathbb{I}$

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

Fermeture

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

Valeur taggée

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

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 de terminaison

- Pas de génération de code
- Point d'allocation = identifiant
- Nombre fini d'identifiants
- Abstraction de la pile d'appels

CFG exécutable

Représentation plus bas-niveau que le CFG

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

Exemple

$\text{Let}(v, \text{Cons}(t, e), i) \vdash \text{Let}(\overline{t}, \text{Tag } t, \text{Let}(\overline{e}, \text{Tuple } e, \text{Let}(v, \text{Tuple } [\overline{t}, \overline{e}], i)))$

Exemple

$\text{Match}(v, [t, p, [x, y], \{z\}], d, \emptyset) \vdash$
$\text{Let}(\overline{t}, \text{Get}(v, 0), \text{Let}(\overline{e}, \text{Get}(e, 1), \text{Switch}(\overline{t}, [t, p, [\overline{e}, z]], d, []))) \emptyset$

Exemple

$\text{MatchBranch}([x, y], \{z\}, \emptyset)\ i \vdash [\overline{e}, z]\ \text{Let}(x, \text{Get}(\overline{e}, 0), \text{Let}(y, \text{Get}(\overline{e}, 1), i))\ \emptyset$

CFG exécutable

Propagation

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

Exemple

$B = \{p_1 \to ([c], \text{Switch}(c, b, d, []))\}$
$\text{ApplyDirect}(p_1, [x], [p_2, [z]])\ B \vdash \text{Switch}(x, b, d, [p_2, [z]])$

Exemple

$B = \{p_1 \to ([y], \text{Return } y)\}$
$\text{ApplyDirect}(p_1, [x], [p_2, [z]])\ B \vdash \text{ApplyDirect}(p_2, [x, z], [])$

CFG exécutable

Interprétation

Interpréter le CFG

- Tester la validité des transformations
- Réaliser des benchmarks

Conclusion

Importance des représentations intermédiaires

- Transformations simples
- Conserver le plus 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

Expérimentations à mener

- Exploiter les résultats de l'analyse
- Nouvelles heuristiques de spécialisation
- Tester sur de vrais programmes