コンパイラ演習
第 2 回
(2011/10/13)

中村 晃一　野瀬 貴史　前田 俊行
秋山 茂樹　池尻 拓朗
鈴木 友博　渡邊 裕貴
潮田 資秀
小酒井 隆広
山下 諒蔵　佐藤 春旗
大山 恵弘　佐藤 秀明
住井 英二郎

今日の内容

- 型推論 (type inference)
- K 正規化 (K-normalization)
  - 式の評価で現れる中間結果を
    全て明示的に変数に束縛する
- α 変換 (α-conversion)
  - 異なる変数が異なる名前を持つようにする
- 最適化
  - インライン化 (inlining)
  - 定数畳み込み (constant folding)
  - 不要な束縛の除去

型推論

- MinCaml では typing.ml
- 部分式の型を unify しながら型を決めていく
  - **`X = Y`**
    - **X** の型と **Y** の型を unify、式全体は bool 型
  - **`X + Y`**
    - **X** の型と int 型を unify、**Y** の型と int 型を unify、式全体は int 型
  - **`if X then Y else Z`**
    - **X** の型と bool 型を unify、**Y** の型と **Z** の型を unify、式全体は **Y** の型
- let, let rec では型環境を拡張

MinCaml 特有の型推論処理

- 型環境に含まれない変数は
  外部変数とみなす
  - typing.ml での **`Var(x)`** の処理部分に注目
- 型が決まらない変数は int 型とみなす
  - typing.ml での **`Var({contents=None})`** の処理部分に注目
- 多相型はない
  - **`let rec f x = x in (f 1.23, f 123)`** は型エラー

K 正規形

- ML Kit (http://www.it-c.dk/research/mlkit/)
  というコンパイラの中間言語
- 計算の中間結果が
  全て変数として明示化された形

K 正規化

- MinCaml では kNormal.ml
- 部分式を明示的に変数へ束縛することで K 正規形に変換する
  ⇒ ML を RISC アセンブリに少し近づける
  - 例: `(x - y) - (y - z)`

```
(x - y) - (y - z)
    ⟶ let i1 = x - y in
      let i2 = y - z in
        i1 - i2
```

K 正規化の簡単な最適化

- 最初から変数になっている部分式はそのままにしておく
  - たとえば、**123 - x** は
    ```
    let i1 = 123 in
     let i2 = x in i1 - i2
    ```
    ではなく、単に
    ```
    let i1 = 123 in i1 - x
    ```
    とする
- この最適化は **insert_let** という関数によって実現されている

α 変換

- MinCaml では alpha.ml
- 束縛変数の名前を “fresh” なものに付け替える
  ⇒ 全ての変数には全て異なる名前を持たせ以降の処理を容易にする
  - 例:

```
let x = 123 - 456 in
let x = x - 789 in -x
```

↓

```
let x1 = 123 - 456 in
let x2 = x1 - 789 in -x2
```

MinCaml での最適化

- インライン化 (inlining)
- 定数畳み込み (constant folding)
- 不要な束縛の除去

関数呼び出しのオーバーヘッド

- 関数呼び出しにはオーバーヘッドがかかる
  - 呼び出し時：
    レジスタ退避、ジャンプ、スタックポインタ更新
  - リターン時：
    スタックポインタ更新、ジャンプ、レジスタ復帰
- 関数として呼び出さずに
  関数本体の処理が行えれば
  オーバーヘッドはかからない

インライン化

- 関数の呼び出しを
  その関数の本体で置き換える

```
    let rec f x y = M in …(f a b)…
→ let rec f x y = M in …([a,b/x,y]M)…
```

  - インライン化するとき、関数本体の式を
    α 変換する必要があることに注意

  – また、むやみにインライン化すると…
    - コードサイズが爆発する
    - コンパイルが停止しない

    ⇒ 何らかの heuristics を用いる必要がある

定数畳み込み

- 演算のオペランドが「明らかに」定数だったらコンパイラが計算してしまう
  - 例: `let x = 7 in let y = 3 in x - y`
    → `let x = 7 in let y = 3 in 4`
  - インライン化の後に行うと吉

不要な束縛の除去 (変数定義の場合)

- 副作用がなく、変数も使用されないような let を省略する:

  `let x = M in N` → `N`

  - ただし以下の場合に限る
    - `x` は `N` に現れない
    - `M` は副作用を持たない
      - » これは「関数適用や配列への代入を含まない」で近似するのが簡単

- インライン化や定数畳み込みの後に行うと吉

不要な束縛の除去
(関数定義の場合)

- 関数定義自体に副作用はないので、
  使用されない関数定義はすべて除去してよい
  - 一般に相互再帰的な関数定義

    let rec … in N

    において
    - **N** に現れる関数は使用される
    - 使用される関数の定義に現れる関数は使用される
  - 面倒だったら「一つでも **N** に現れる関数が
    あったらすべて残す」でも十分

最適化の繰り返し (MinCamlの場合)

- 最適化による変化がなくなるまで
  最適化処理を繰り返す
- ただし一定の回数を超えたらそこで打ち切る

```
let rec iter n e =
   Format.eprintf "iteration %d@." n;
   if n = 0 then e else
   let e' =
      Elim.f (ConstFold.f
                (Inline.f (Assoc.f (Beta.f e))))
   in if e = e' then e else iter (n - 1) e'
```

各処理の詳細

- 次のドキュメントを参考にしてください
  - 住井先生ご提供
    「MinCamlの疑似コード」
    http://min-caml.sourceforge.net/min-caml.pdf

参考: A 正規化

- ネストした let を平らにする

  let x = (let y = M1 in M2) in M3

  → **let y = M1 in let x = M2 in M3**

  – ただし **y** は **M3** の中に現れないものとする
  （正しく α 変換されていれば常に成り立つ）

  – 結果の式は A 正規形 (A-normal form) と呼ばれる

- 参考論文:
  Flanagan, Sabry, Duba, Felleisen. “The Essence of Compiling with Continuations”. In PLDI 1993.

課題

共通課題

- 今回は全3問**すべて**解いてください

共通課題(1)

- 下の式を K 正規化および α 変換せよ
  - 前出の「K 正規化の簡単な最適化」はしてもよいがそれ以外の最適化はしないこと

```
let rec x x = let x = let x = x - -x
in x - (let x = -x in x - -x)
in x - -x in let x = x 125 in x - -x
```

- 前問題で得られた答を A 正規化せよ

共通課題(2)

- 共通部分式除去を実装せよ
  - K正規形の式に対して同じ式が現れていたら最初の式を再利用する

```
…
let a = x + y in
…
let b = x + y in
…
```

```
…
let a = x + y in
…
let b = a in
…
```

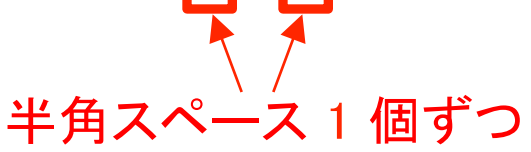
共通課題(3)

- インライン化の際に α 変換を行わないと
  プログラムの意味が変わってしまう例を挙げよ
  - ただしインライン化する前のプログラムは
    元から正しく α 変換されているものとする

- インライン化・定数畳み込み等の最適化を繰返す際に
  「プログラムのサイズは増えないが
  回数制限がないと止まらない」 ような例を挙げよ
  - ただし、最適化をせず普通に実行したら
    正常に停止するような例に限る

課題の提出先と締め切り

- 提出先: compiler-report-2011@yl.is.s.u-tokyo.ac.jp
- 締め切り: 2 週間後 (10/27) の午後 1 時 (JST)
- Subject: **Report␣␣2␣** <学籍番号:5桁>



  - 例: **Report␣␣2␣␣11099**
- 本文にも氏名と学籍番号を明記のこと

◆ 質問は compiler-query-2011@yl.is.s.u-tokyo.ac.jp まで