

コンパイラ演習

第4回

(2011/10/27)

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

MinCaml → アセンブリ言語への道: 前回までのあらすじ

- K 正規化 (第2回)
 - 式の評価で現れる中間結果に
全て明示的に変数を束縛した
 - » ついでに幾つかの最適化を行った
- クロージャ変換 (第3回)
 - ネストした関数定義を平らにした

でもアセンブリ言語とはまだ隔たりがある

- メモリ・メモリ操作が構造化・抽象化されている
 - タプル・クロージャ・配列など
 - アセンブリ言語のメモリはただのフラットなバイト列
- 変数が任意数個存在しうる
 - アセンブリ言語では有限個のレジスタしかない
- 関数呼出しが存在する
 - アセンブリ言語にはジャンプ・分岐命令しかない
- if-then-else がある

MinCaml における今後の変換

- メモリ操作をアセンブリ命令の組み合わせに変換する (virtual.ml)
- 変数をレジスタに対応させる (regalloc.ml)
- 関数呼出しをアセンブリ命令に変換する (regalloc.ml)
 - 関数呼出し規約に従ってレジスタの退避・復帰させる
- if-then-else をラベルと分岐命令の組み合わせに変換する (emit.ml)

今日の内容

MinCaml の仮想マシンコード

- 特徴
 - メモリ操作はアセンブリ言語風
 - レジスタは無限個ある (とみなす)
 - 関数呼出しがある
 - if-then-else がある
- 定義は SPARC/asm.ml にある
 - (実際はレジスタ割り当て後も使い回しているが.....)

メモリ操作の仮想マシンコードへの変換

- ここまで 1 命令として扱われていた以下のメモリ操作を複数の命令の組み合わせに変換する
 - クロージャ作成・クロージャからの値の読み出し
 - タプル作成・タプルからの値の読み出し
 - 配列の要素の読み書き

MinCaml での浮動小数点数数の処理

- 関数の引数を
浮動小数点数とそれ以外に分ける
 - (汎用レジスタと浮動小数点数レジスタが異なることを仮定しているため)
 - 型情報を利用する
 - これは後のフェーズのための準備
- 等号・不等号プリミティブを
浮動小数点数用とそれ以外用に分離する
 - これも型情報を利用する
- 浮動小数点数の定数テーブルを作成
 - 浮動小数点数の即値セットのため
 - (浮動小数点数を即値として扱えないことを仮定しているため)

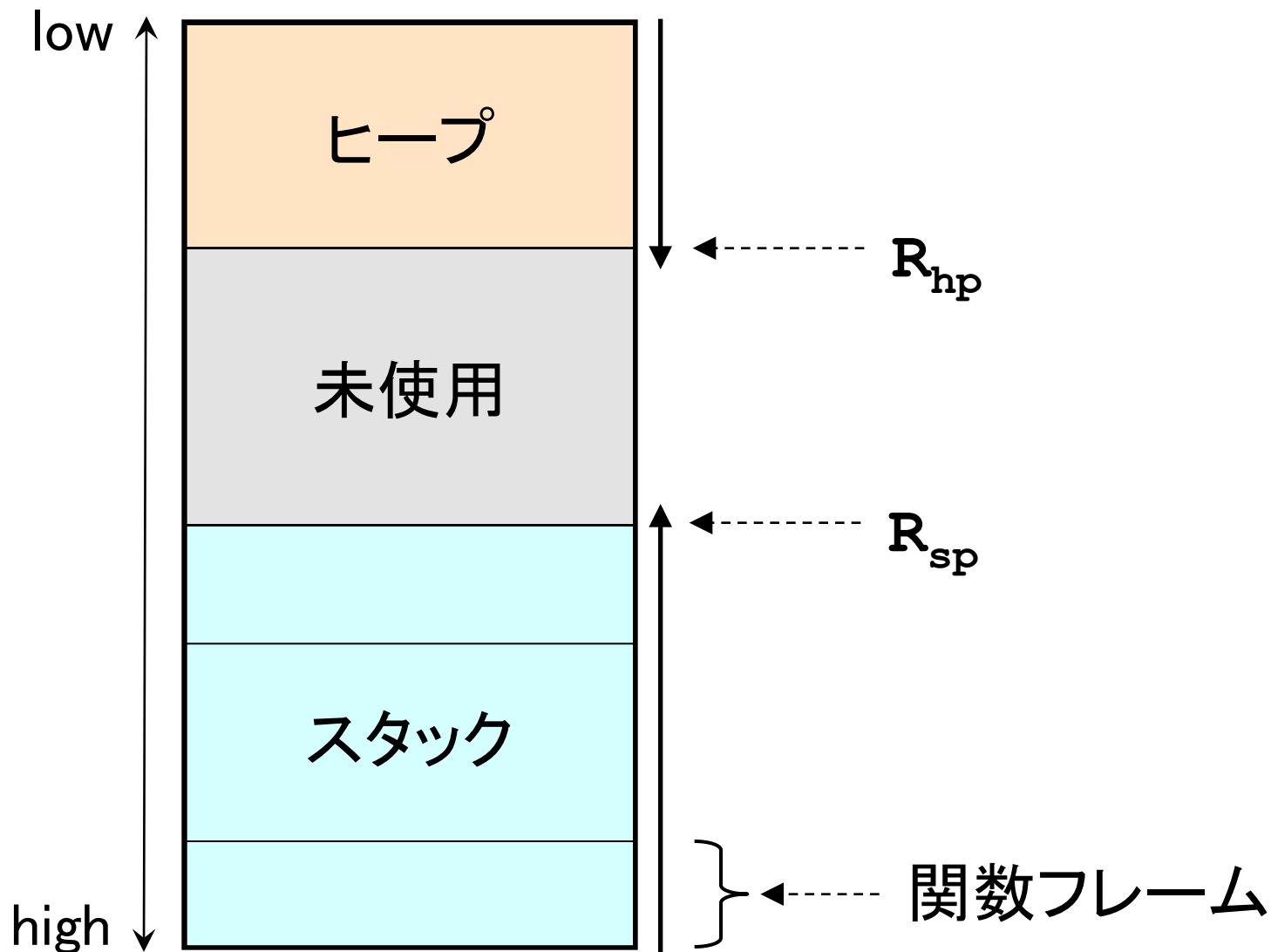
関数呼出し規約とは?

- 関数の呼出し元と
呼出される関数との間の約束事
 - 関数呼出し時に各レジスタが保持すべき値
 - 関数呼出し時のあるべきメモリの状態
 - 呼出された関数による
レジスタ・メモリの状態への影響
 - etc.
- 普通は各コンパイラ・アーキテクチャごとに
決められている

関数呼出し規約の例： レジスタの使い方

- 汎用レジスタ: R_0 から R_{n-1} まで n 個
- 返り番地用レジスタ: R_{ret}
- ヒープポインタ用レジスタ: R_{hp}
- スタックポインタ用レジスタ: R_{sp}

ヒープポインタ・スタックポインタの 使い方の例



関数呼出し規約の例： 呼出し側

- 生きている変数の値をスタックに退避する
- 呼び出される関数のクロージャのアドレスを R_0 に入れる
 - (クロージャを通じて呼び出す場合)
- 引数を R_1 、 R_2 、 R_3 、... に入れる
- 返り番地を R_{ret} に入れる
- 呼び出される関数のアドレスをクロージャから取り出す
 - (クロージャを通じて呼び出す場合)
- 取り出したアドレスにジャンプする

関数呼出し規約の例： 呼出される側

- R_0 の指すクロージャから
自由変数の値を取り出す
–（自由変数がある場合）
- R_1 、 R_2 、 R_3 、... を引数として
関数本体を実行する
- 返り値を R_1 に入れる
- R_{ret} にジャンプする

レジスタの使い方の例： SPARC アーキテクチャなら...

g0	不使用：常に値が 0 なので
g1~g7	不使用：ライブラリの都合
o0~o4	一般レジスタとして使用 ($R_{13} \sim R_{17}$)
o6	不使用：OSやデバッガのため
o5、o7	R_0 、 R_{ret}
l0~l7	一般レジスタとして使用 ($R_5 \sim R_{12}$)
i0、i1	R_{sp} 、 R_{hp}
i2~i5	一般レジスタとして使用 ($R_1 \sim R_4$)
i6、i7	不使用：OSやデバッガのため

※ 上は MinCaml での規約で、標準の規約とは異なるので、
main 関数や外部関数にはスタブが要る

レジスタの使い方の例： PowerPC アーキテクチャなら...

r0	不使用：命令によっては値が 0 なので
r2、r5～r29	一般レジスタとして使用 ($R_1 \sim R_{26}$)
r30、lr	R_0 、 R_{ret}
r3、r4	R_{sp} 、 R_{hp}

※ これも標準とは異なるので、main 関数や外部関数にはスタブが要る

レジスタの使い方の例： POWER アーキテクチャなら...

r0	不使用：命令によっては値が 0 なので
r2	不使用：ライブラリの都合
r3～r29	一般レジスタとして使用 ($R_1 \sim R_{27}$)
r30、lr	R_0 、 R_{ret}
r1、r31	R_{sp} 、 R_{hp}

※ これも標準とは異なるので、main 関数や外部関数にはスタブが要る

callee-save register を 導入する手もある

- 各レジスタを
caller 側で保存・復元するか
callee 側で保存・復元するかは規約しだい
- 特に小さい関数や leaf 関数で
callee-save register の効果が出る

共通課題

- 三つの共通課題のうち一つ以上を解いてください

共通課題 1

- 自分たちのアーキテクチャの関数呼出し規約を定義・説明せよ
- また他のアーキテクチャについて関数呼び出し規約がどうなっているか調べて述べよ
 - 少なくとも2つ以上の種類について調べること
 - 既存の CPU アーキテクチャ (Itanium, ARM, etc.) や他の班のアーキテクチャでもよい
- 以上少なくとも3つの関数呼出し規約について比較し類似点・相違点・利害得失等を論ぜよ

共通課題 2

- クロージャ変換後の中間コードに含まれる
ネストしたタプルの平坦化を実装せよ

```
let a =  
    (1, (2, 3)) in  
...
```



```
let a =  
    (1, 2, 3) in  
...
```

- a がアクセスされる箇所で注意が必要

共通課題 3

- クロージャ変換後の中間コードに含まれる
タプル (や配列) の生成を
不要なら削除するような最適化を実装せよ

```
let a = (1, 2) in
...
let (x, y) = a in
...
```



```
let x = 1 in
let y = 2 in
...
...
```

- a が関数の引数や返り値に含まれるときや
配列に代入されるときには注意が必要

課題3をよりまじめに解くためのヒント: 関数の引数のタプルを扱う

- 関数の引数を展開したものを用意することで対応できる

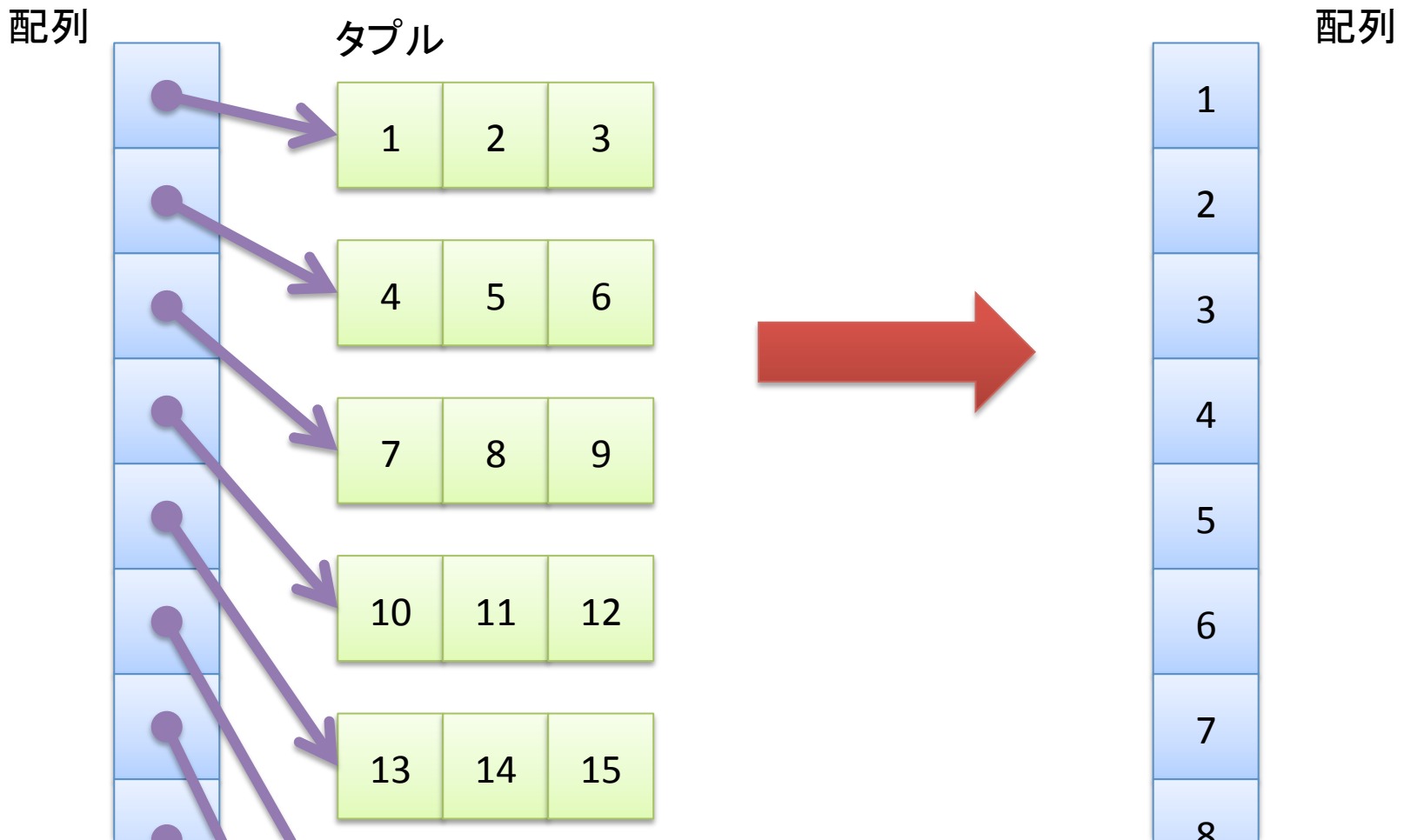
```
let rec f a =  
  let (x, y) =  
    a in  
  ...  
in f (1, 2)  
...
```



```
let rec f' x' y' =  
  let (x, y) =  
    (x', y') in  
  ...  
in f' 1 2  
...
```

課題3をよりまじめに解くためのヒント: 配列中のタプルを扱う

- 配列の中にタプルを埋め込むこともできる



課題3をよりまじめに解くためのヒント: タプルを配列に埋め込むときの注意

- 配列内のタプルに副作用がないか、
あってもプログラムの意味が変わらないか
注意する必要がある

– 例えば右のような場合
注意が必要

- 注意が必要な
場合は他にもあるので
注意

```
...  
let a = (1, 2) in  
b.(0) <- a;  
let c = (3, 4) in  
b.(0) <- c;  
let (x, y) = a in  
x
```

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 (11/10) の午後 1 時 (JST)
- Subject: **Report 4** <学籍番号: 5 桁>

半角スペース 1 個ずつ

– 例: **Report 4 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