コンパイラ演習
第 3 回
(2011/10/20)

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

- クロージャ変換
  - ネストした関数定義を平らにする
    ⇒ K 正規形をさらにアセンブリ・機械語に近づける

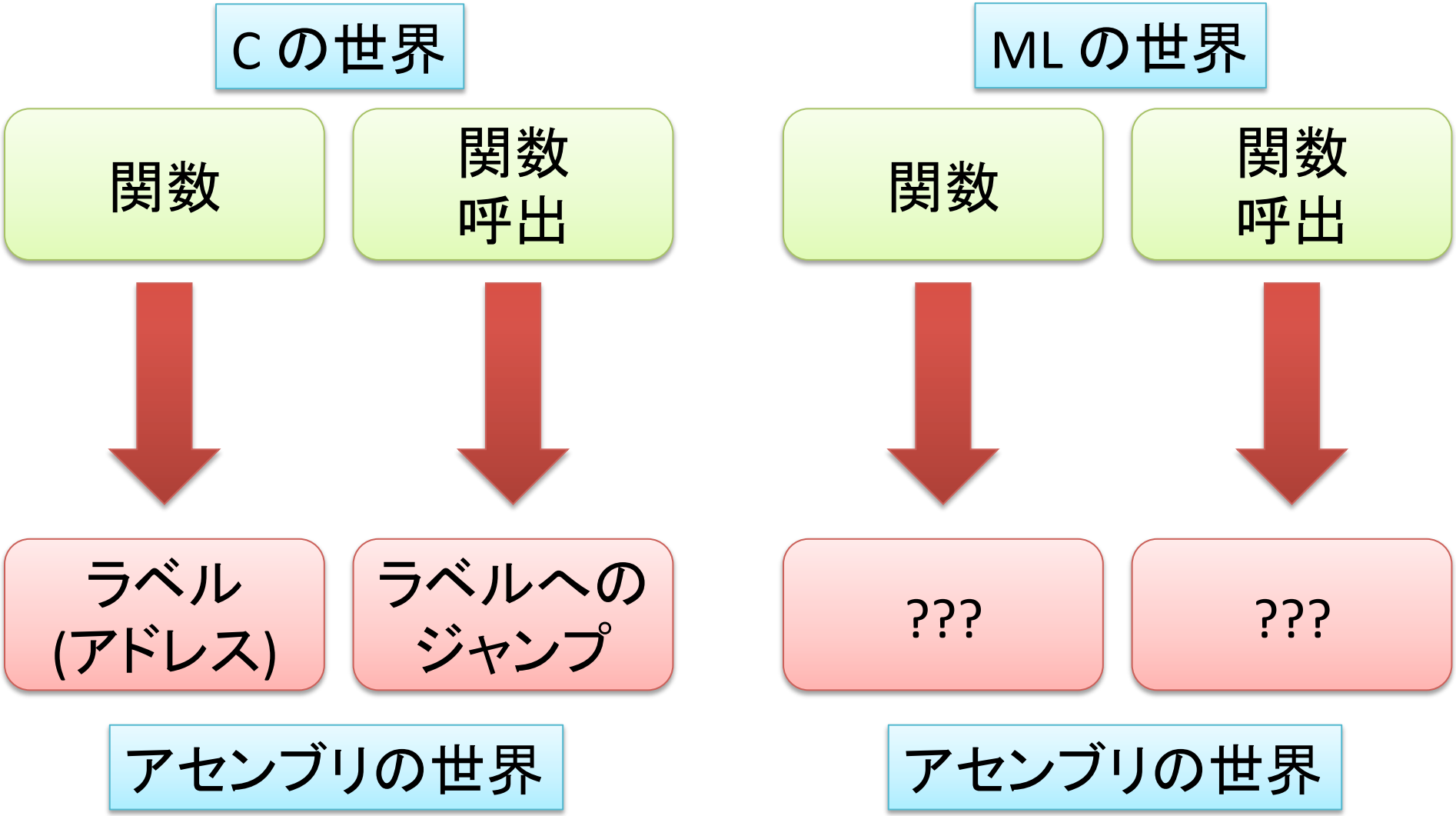
Motivating Example

- これをどうアセンブリに落とすか？

```
let rec f a =
  let rec g b = a + b
  in g 3
```

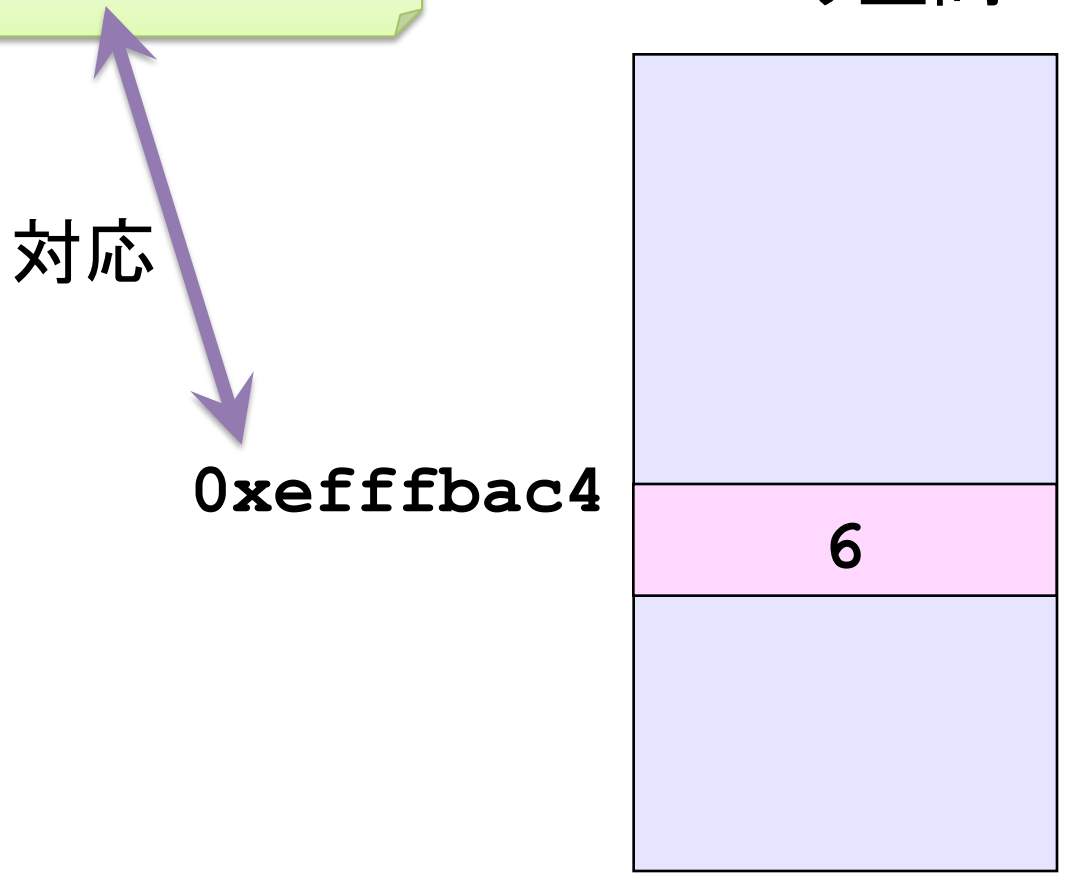
```
val f : int -> int = <fun>
# f 4;;
- : int = 7
```

関数の表現: C vs. ML


C の世界
関数
関数
呼出
ラベル
(アドレス)
ラベルへの
ジャンプ
アセンブリの世界
ML の世界
関数
関数
呼出
???
???
アセンブリの世界

C の世界: 値の対応

```
static int a = 6;
```



C の世界: 関数の対応

```
int foo(int a) {
  return a + 1;
}
```

対応

0x08048328

関数 foo の呼び出し

ラベル foo へのジャンプ

メモリ空間

```
foo:
pushl %ebp
movl %esp,%ebp
movl 8(%ebp),%eax
incl %eax
leave
ret
```

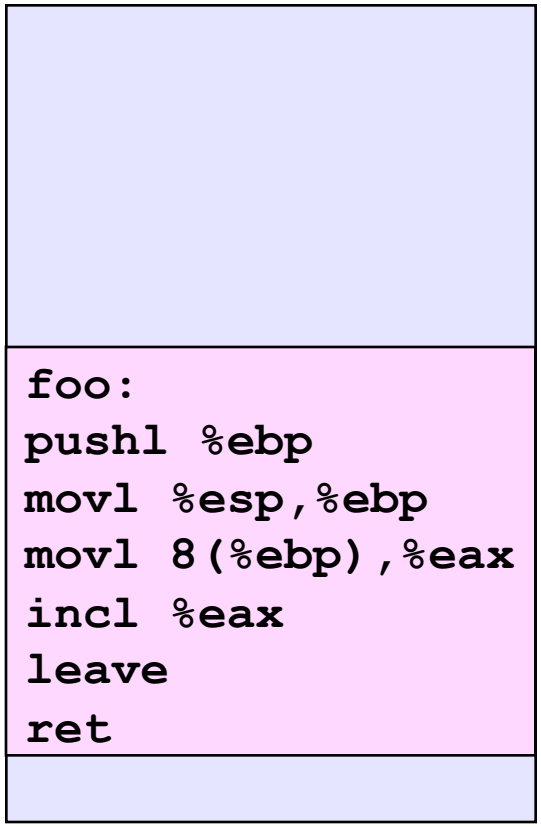
Q. ML でも同様にできるか?

```
let rec foo a =
  a + 1
```

対応?

0x08048328

メモリ空間



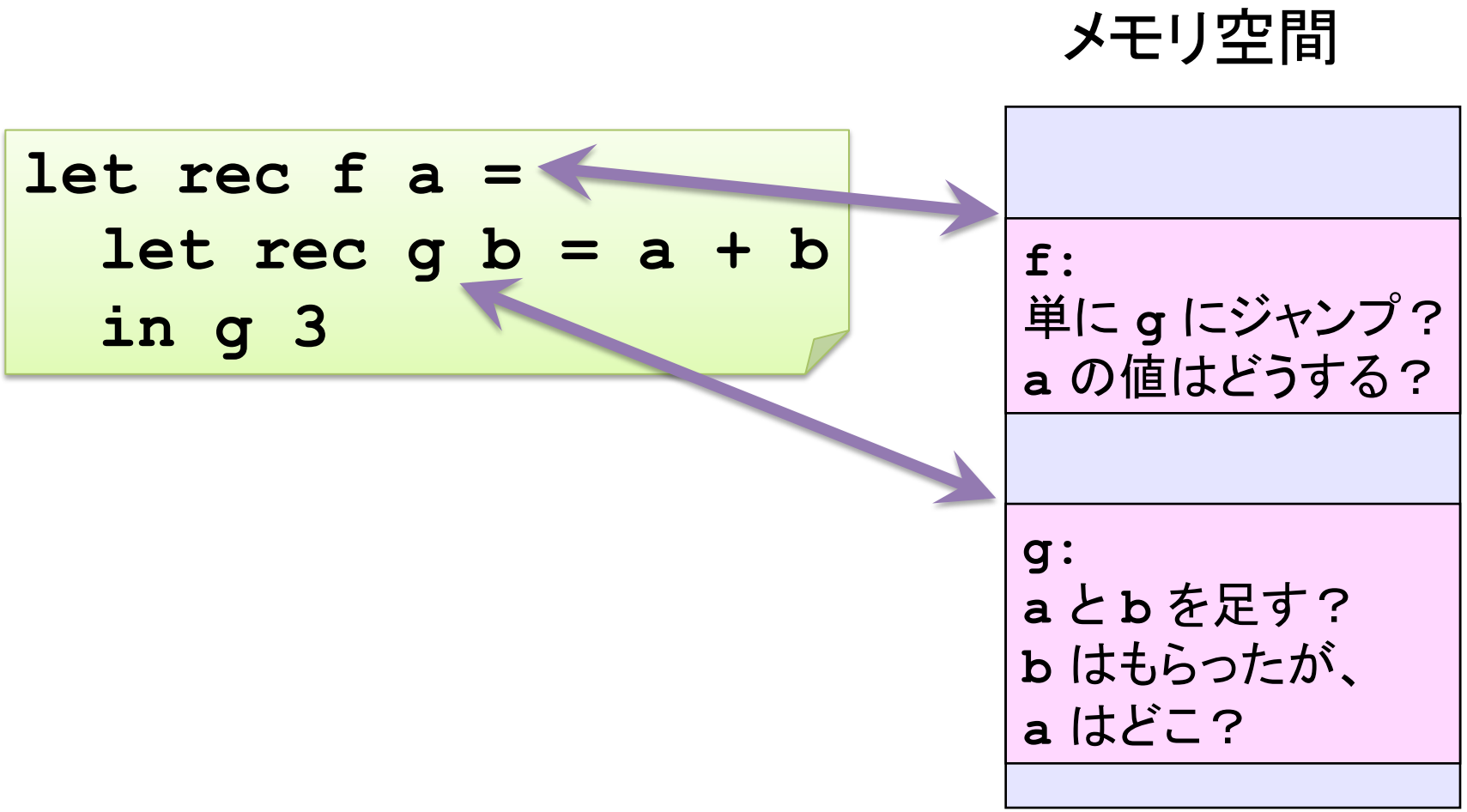
関数 foo の呼び出し

?

ラベル foo へのジャンプ

Q. ML でも同様にできるか?
A. 上手くいかない場合がある

```
let rec f a =
  let rec g b = a + b
  in g 3
```



なぜ上手くいかないか: その1

- 関数定義内に自由変数が存在しうるので

```
let rec f a =
  let rec g b = a + b
  in g 3
```

g の中で使用されているが、
g の外で定義されている

自由変数の問題に対する解決策

- その1: 自由変数を持つ関数を許さない
  - C, C++ のほとんどの処理系
- その2: 関数のアドレスと一緒に
  自由変数の値も保存しておく
  - Scheme, ML, Java (inner class) の処理系

なぜ上手くいかないか: その2

- 関数定義がネストしうるので

```
let rec f a =
  let rec g b = a + b
  in g 3
```

g が f の中で
ネストして定義されている

ネストした関数の問題に対する解決策 その1

- C, C++ の規格など
  - ネストした関数は許さない

ネストした関数の問題に対する解決策 その2

- GCC の言語拡張など
  - 関数定義内で関数を定義できる
    - 自由変数にもアクセスできる
  - 内側の関数のアドレスを変数に格納し
    それを使って後で呼び出すこともできる
    - ただし、外側の関数が return した後に
      内側の関数を呼び出すと何が起こるかわからない

ネストした関数の問題に対する解決策 その3

- ML など
  - ネストした関数定義や
    自由変数を持つ関数がなくなるように
    コンパイラが変換する
    - さらに、自由変数を持つ関数を
      スコープ制限なしに利用できるようにする
      - 例：内側の関数をリストに格納しておいて
        外側の関数が return した後でも呼び出せる

どうやって自由変数を関数定義からなくすか?

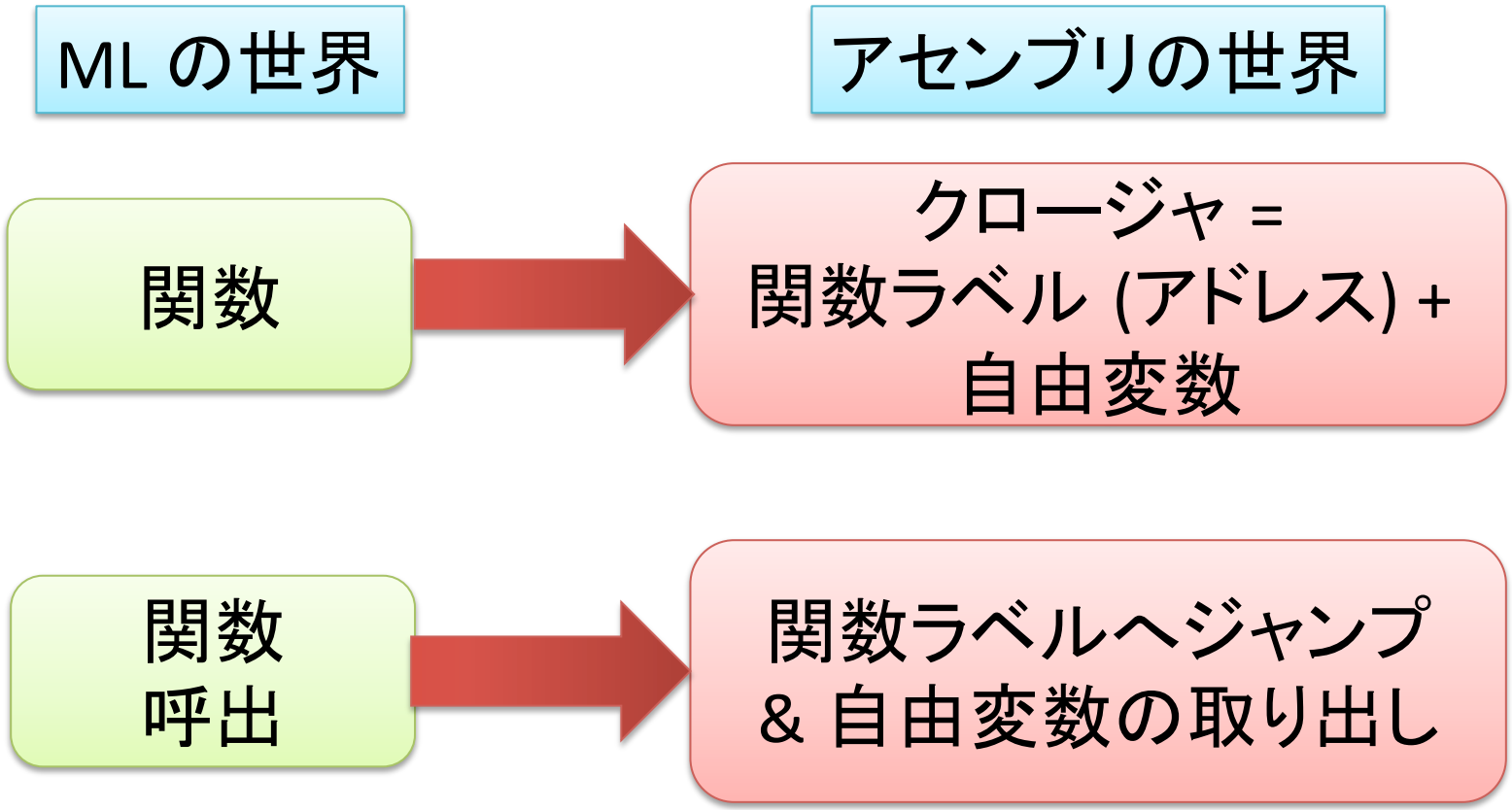
- 大雑把なアイディア:
  自由変数を関数の引数として渡すようにする

```
let rec g a b = a + b
let rec f a = g a 3
```

  - MinCaml では a は「クロージャ」を介して渡される
    - クロージャ = 関数ラベル (アドレス) と
      自由変数からなるタプル・レコード
  - 自由変数がなければ
    ネストした関数を平らに変換するのは難しくない

クロージャ変換

関数の表現: MinCaml



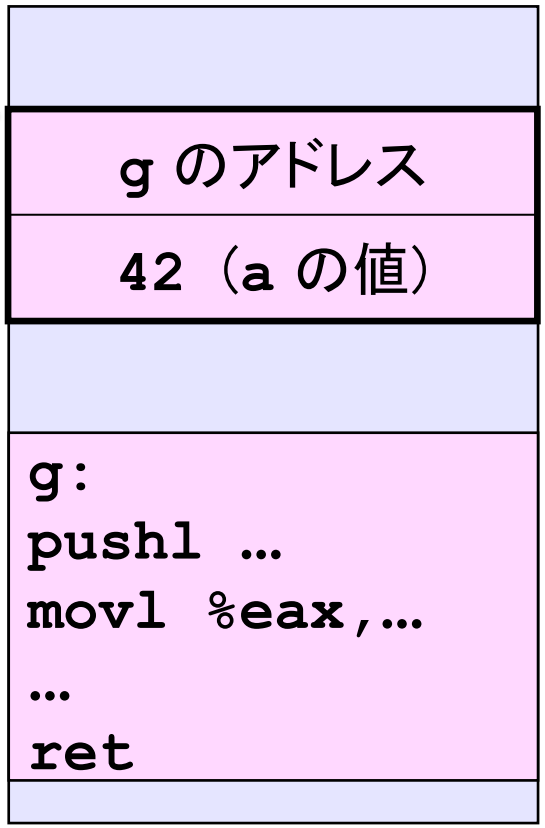
MinCaml でのクロージャ変換の概略

- 以下のようなコードに変換する
  - 関数を作るとき
    - ヒープにクロージャを作成し
      関数本体のアドレスと自由変数を保存する
  - 関数を呼び出すとき
    - クロージャから関数本体のアドレスを取り出し
      そこへジャンプ
  - 呼び出された関数
    - クロージャから自由変数を取り出す

(例) 関数作成の図

```
let rec f a =
  let rec g b = a + b
  in g 3
```

メモリ空間

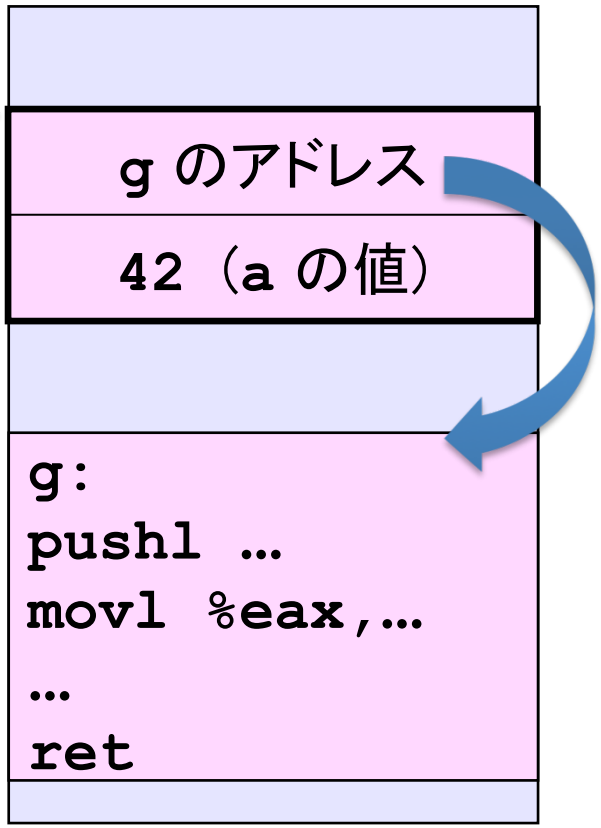


関数 g 本体のアドレスと
自由変数 a の値から
クロージャを作成

(例) 関数呼び出しの図

```
let rec f a =
  let rec g b = a + b
  in g 3
```

メモリ空間

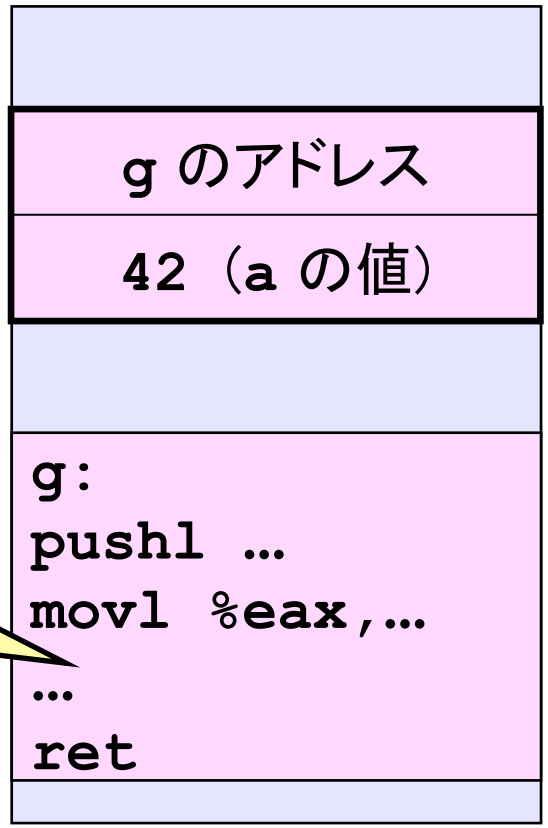


クロージャから
関数 g 本体のアドレスを
取り出しそこへジャンプ

(例) 呼び出された関数の図

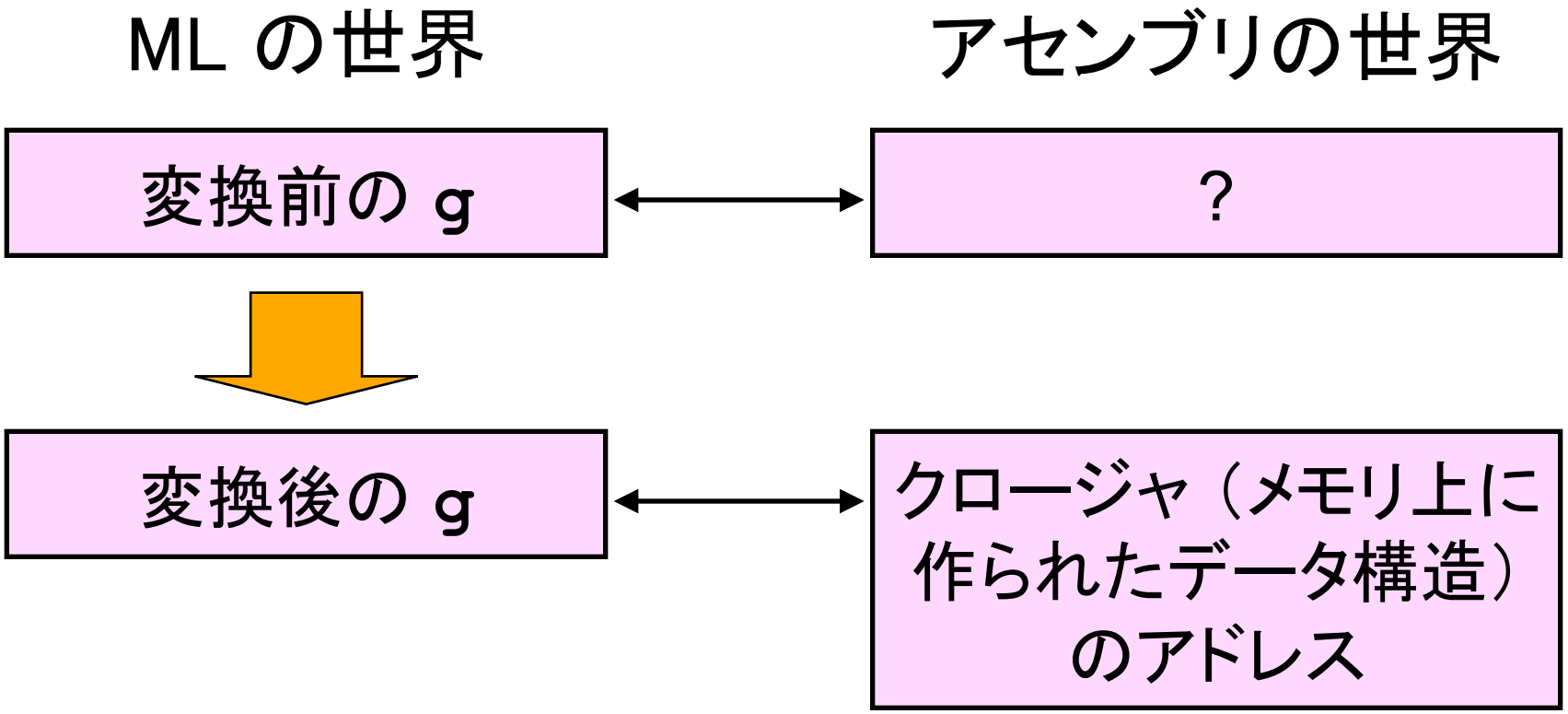
```
let rec f a =
  let rec g b = a + b
  in g 3
```

メモリ空間



クロージャから a の値を
取り出して
引数 b との和を返す

対応関係



クロージャ変換によって、ML の世界のデータを
アセンブリの世界のデータにダイレクト対応させられる

ここまで説明した
クロージャ変換の問題点

- 自由変数のない関数でも
  - クロージャ作成
  - クロージャからの関数アドレスの取り出し

  のオーバーヘッドがかかる

「賢い」クロージャ変換

- やや賢い
  - 「自由変数がない」とわかる関数は
    クロージャではなくラベルを使って直接呼び出す
- 賢い
  - 「やや賢い」変換の結果不要になったクロージャは
    そもそも作らない
  - MinCaml にも実装されている

「賢い」クロージャ変換の手順 (1)
(MinCaml では closure.ml)

- 「自由変数がない関数の集合」
  を管理 (変数 known)
  - 関数 fv を使って自由変数を計算

- その情報に応じて
  関数の呼び出し方を決める
  - AppCls (クロージャ経由) か AppDir (直接) か

「賢い」クロージャ変換の手順 (2) (MinCaml では closure.ml)

- 不要なクロージャ作成コードを削除する
  - let の本体に出現しない関数のクロージャは作らない (MakeCls を出力しない)

共通課題

- 共通課題の三つのうち
  二つ以上を解いてください。

共通課題 1

- ML 演習や Scheme 演習などで
  自分が今までに書いたプログラムの中から
  自由変数を持つ関数の例を挙げよ
- その例を自由変数がなくなる
  (クロージャがいらない) ように書き直せ

もしそういう例がなかったら…

逆に「自由変数を持つ関数を使えばもっと
簡単だったのに」というプログラムを探して
そのように書き直してください

共通課題 2

- Lambda lifting について調べ、実装せよ。
- また共通課題 1 や 3 の関数などに対して lambda lifting を行うなどして
  性能や変換手法について
  クロージャ変換と比較して論ぜよ
- 参考文献: “Implementing Functional Languages,” Simon Peyton Jones and David Lester. Published by Prentice Hall, 1992.
  - http://research.microsoft.com/en-us/um/people/simonpj/papers/pj-lester-book/

共通課題 3

- 次ページのプログラムを
「賢く」クロージャ変換したとき
枠の付いた各関数について以下を答えよ
  - クロージャが作られるかどうか
  - 作られる場合、
クロージャの中に含まれるラベルおよび変数は何か
    - 関数 `f` のラベルは `Lf` などと表記してください

共通課題 3 (つづき)

a.
```
let z = 4 in let rec f x = x - z in f 8
```

b.
```
let rec g x = x - 2 in g 6
```

c.
```
let rec f x = x - 1 in f
```

d.
```
let rec g h = let rec i x = h x in i in g
```

e.
```
let rec i x = x in
  let z = 4 in
  let rec f x = i (z - 5) in
    if z < 6 then (i, f 7) else (f, 8)
```

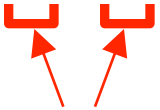
f.
```
let rec fact x =
    if x = 1 then 1
    else x * fact (x - 1)
  in fact 6
```

コンパイラ係用選択課題

- クロージャ変換後のコードに対する型検査を実装せよ
  - すなわち、Closure.prog に対して型検査を行う関数を実装せよ
  - ただし以下の条件を全て満たすこと(証明はしなくてもよい)
    - 型検査が健全であること
    - Syntax.t での型検査 Typing.f をパスした式(を K 正規化したもの) に対して Closure.f を適用して得られた変換後のコードが型検査に必ずパスすること

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 (11/03) の午後 1 時 (JST)
- Subject: **`Report␣␣3`** <学籍番号:5桁>



  - 例: **`Report 3 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