コンパイラ演習
第 9 回

2007/11/29

小酒井 隆広　前田 俊行
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

- 例外処理 (exception handling)

例外処理機構

- エラーが発生したときに、現在の計算を中断してエラー処理用のコード（ハンドラ）にジャンプする機構
  - 開こうとしているファイルが見つからないとき
  - 配列の境界を越えてアクセスしたとき
  - ユーザの入力がおかしいとき
  - 関数に渡されるべきでない値が渡されたとき

O'Caml の例外処理コンストラクト

- 例外の宣言:

  exception *decl*

- 例外の明示的スロー:

  raise *exp*

- 例外の捕捉:

  try *exp* with
  | pat_1 -> exp_1
  | …
  | pat_n -> exp_n

例外処理の（わざとらしい）例

```
exception Zero_found

let prod_list_aux lst =
  List.fold_left (fun r e ->
    if e = 0 then raise Zero_found
    else r * e
  ) 1 lst

let prod_list lst =
  try prod_list_aux lst
  with Zero_found -> 0
```

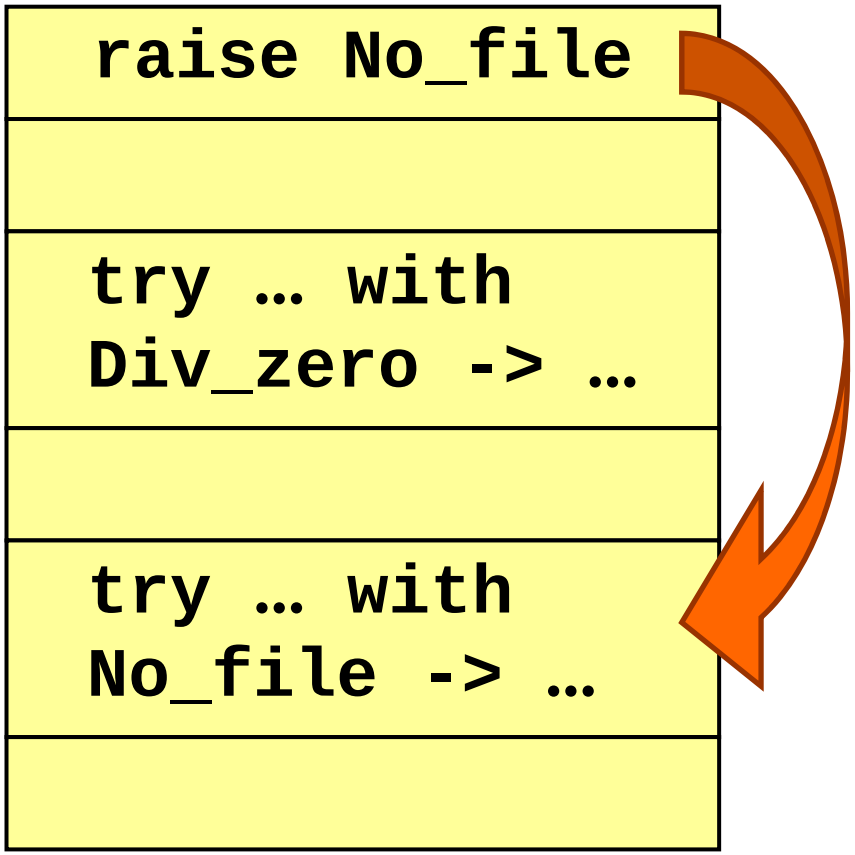
例外処理の実装のポイント

- どうやってスタックを適切に巻き戻し、制御をハンドラに移すか?

```
let f () =
 try raise No_file
 with Div_zero -> …

let g () =
 try f ()
 with No_file -> …

try g ()
with No_file -> …
```



制御をハンドラに移す方法

- Stack unwinding
  - ハンドラが見つかるまで関数のフレームを遡っていく
- Stack cutting
  - ハンドラが見つかるまで `try … with …` を遡っていく
- ハンドラを表現するクロージャを導入する
  - そもそもスタックを使わず CPS でコンパイルする
    - 通常の継続と例外発生時の継続を用意
    - 今回は説明しない

Stack unwinding その 1: Native-code stack unwinding 法

- 式 `try e with …` を、式 `e` の中で例外が発生したら対応するハンドラが実行されるよう、直接コード生成する
  - ハンドラがなければ関数からリターンするようにする

```
let g () =
 try f ()
 with E -> e
```

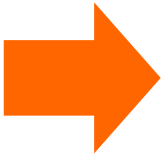
```
let g () =
 let r = f () in
 if (f で例外発生)
 then
  if (発生した例外 = E)
  then e else (リターン)
 else r
```

（非常に大雑把な変換のイメージ）

関数呼び出しが例外を発生したかどうかをどうやって知るか?

- 関数が以下のいずれかを返すようにする
  - 例外なしフラグ ＋ 返値
  - 例外ありフラグ ＋ 例外の情報
- 関数の呼び出し側では、返値を調べることで例外が発生したかどうか分かる

```
let g () =
 try f ()
 with E -> e
```



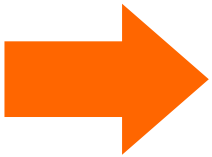
```
let g () =
 let r = f () in
 match r with
 | Exc(E) -> e
 | _ -> r
```

（非常に大雑把な変換のイメージ）

Stack unwinding その 2: Runtime stack unwinding 法

- 式 **`raise ...`** を stack unwinding を行うランタイムライブラリへの呼び出しにコンパイルする
  - あとはライブラリ側で何とかする

```
raise E
```



```
builtin_stack_unwind E
```

（非常に大雑把な変換のイメージ）

ランタイムライブラリが
やらなくてはならないこと

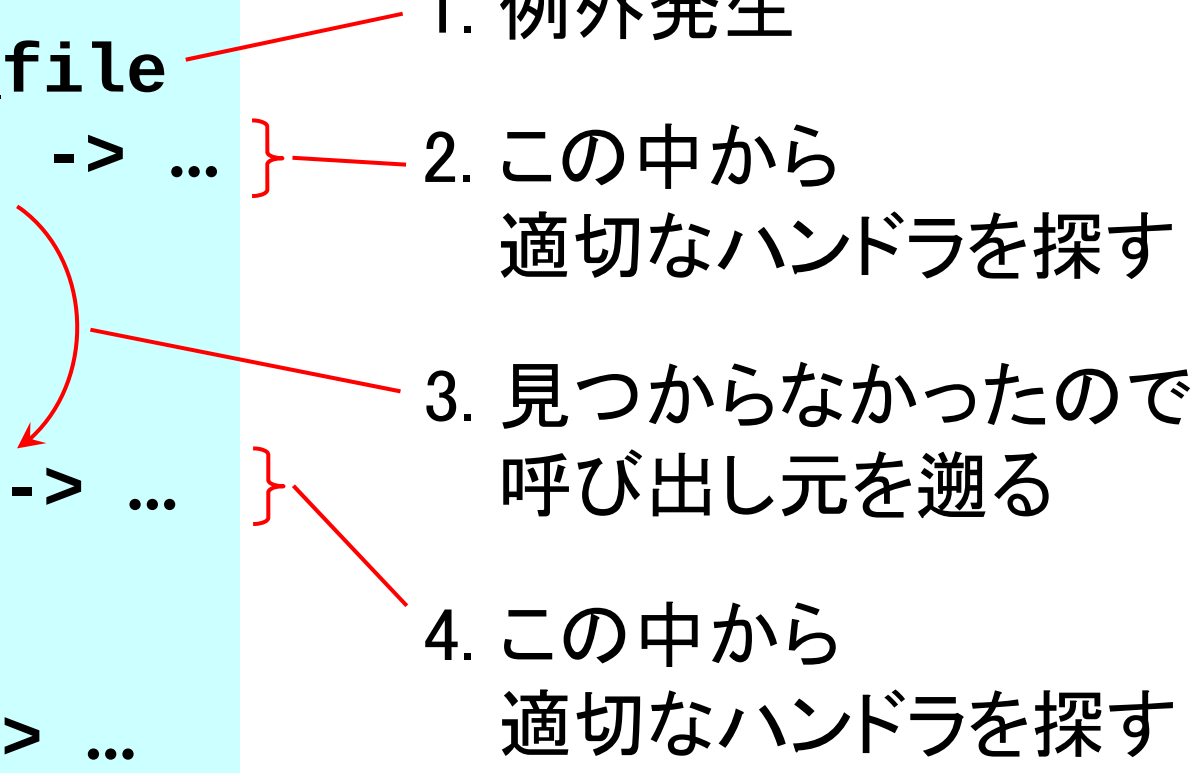
- 例外が発生した時点のプログラムカウンタ（つまり、ライブラリの呼び出し元）に基づき、適切なハンドラを探して実行する
- 適切なハンドラが見つからなければ、「呼び出し元の関数の呼び出し元」に遡って適切なハンドラを探す

ランタイムライブラリの動作例

```
let f () =
 try raise No_file
 with Div_zero -> …

let g () =
 try f ()
 with No_file -> …

try g ()
with No_file -> …
```



例外が発生したプログラムカウンタからどうやって適切なハンドラを探すか

- **`try … with …`** の “**`try`**” から “**`with`**” までのアドレスの範囲と、“**`with`**” 以下の例外ハンドラとの対応表を作っておく
  - これはコンパイル時に完全に決定できるので、コンパイラが行うとよい
- この表を検索し、適切なハンドラを実行する
  - なお、呼び出し規約に callee-save レジスタがある場合には、関数を遡るときに復元する必要がある

関数の呼び出し元をどうやって遡っていくか

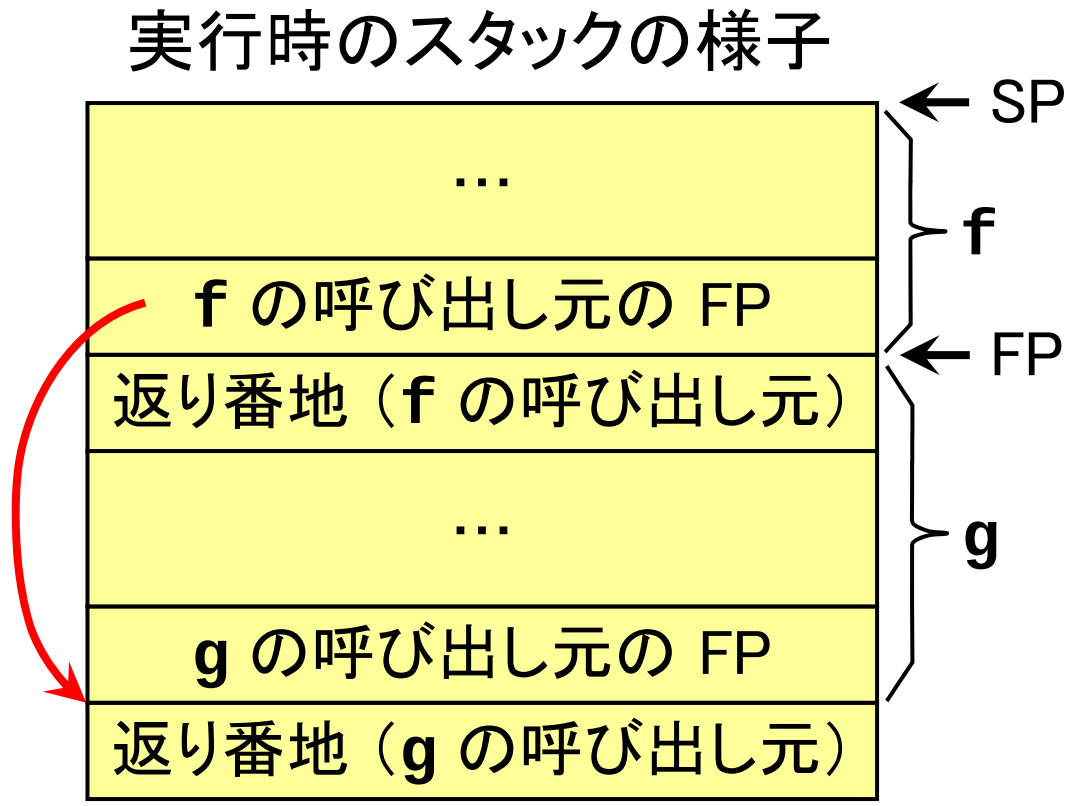
- 「フレームポインタ」を呼び出し規約に導入する
  - フレームポインタ：呼び出し元の関数のスタックのトップ
    - cf. スタックポインタ：実行中の関数のスタックのトップ

フレームポインタの例

- 関数の先頭で、呼び出し元の
  フレームポインタ（FP）を保存する場合

```
let f () =
  …
let g () =
  …
  f ()
  …

g ()
```



Stack cutting

- `try` の入口で、スタックポインタとプログラムカウンタの値をスタックに保存
  - 最後に保存した情報へのポインタを専用レジスタ等に保持
- `raise` では、最後に保存されたスタックポインタとプログラムカウンタの値を復元
  - これにより、実行スタックの上部が「切られる」

Stack cutting と callee-save レジスタ

- Stack cutting では callee-save レジスタを扱うのは難しい
  - 関数のフレームごとではなく `try … with …` ごとに処理が行われるため、スタックのどこに callee-save レジスタが保存されているか簡単には分からない
- Callee-save レジスタがなければ問題ない (例: O'Caml)

参考文献

- N. Ramsey and S. P. Jones, “A Single Intermediate Language That Supports Multiple Implementations of Exceptions” (ACM PLDI 2000)
- C. de Dinechin, “C++ Exception Handling” (IEEE Concurrency, 2000)
- T. Ogasawara, H. Komatsu, T. Nakatani, “A Study of Exception Handling and Its Dynamic Optimization in Java” (ACM OOPSLA 2001)
- 住井英二郎, 大根田裕一, 米澤明憲,「例外処理機構を備えた命令型言語の変換とその定式化」(情報処理学会論文誌, 2004年)

共通課題

- 1, 2 のどちらかを選んで回答せよ

1. Java のバイトコードには「例外テーブル」と呼ばれる情報が埋め込まれている。
   文献の調査や実験などに基づいて、
   例外テーブルが提供する情報、および、
   それが例外処理の実現にどう役立つかを
   解説せよ

共通課題

- 1, 2 のどちらかを選んで回答せよ（つづき）

2. 好きな計算機と処理系を選び、以下の C++ プログラムの性能を計測せよ
   a. 普通の再帰的なフィボナッチ数列計算関数
   b. 関数の先頭で try ブロックに入り、関数からの復帰時にそのブロックから抜けるように a を変更した関数（例外は投げない）
   c. 関数先頭で 1 回 setjmp を無駄に呼び出すように a を変更した関数 (longjmp は呼ばない)
   d. 例外のスローによって値を返し、例外ハンドラで返値を受け取るように a を変更した関数

コンパイラ係用選択課題

- 例外処理を自作コンパイラまたは MinCaml に実装せよ
  - 仕様は自由。最低限 try-with と raise に相当するものだけあればよい
    - Java の finally のようなものはなくてよい
    - 例外とハンドラの間の照合をせず、「スタック内で最も上にあるフレームのハンドラに必ず引っかかる」という仕様でもよい
  - 実装方式は stack unwinding でも stack cutting でも、その他の方法でもよい

課題の提出先と締め切り

- 提出先: compiler-report-2007@yl.is.s.u-tokyo.ac.jp
- 締め切り: 2 週間後 (12/13) の午後 1 時
  - コンパイラ係用課題の締め切り: 2008年 2月 29日
- Subject: **Report␣9␣**〈学籍番号〉

半角スペース 1 個ずつ

  - 例: **Report 9 71099**
- 本文にも氏名と学籍番号を明記のこと

◆ 質問は compiler-query-2007@yl.is.s.u-tokyo.ac.jp まで