TigerBook 輪講資料（第１回：序章、一章）

序章 (Preface)

ここ 10 年間 (この本が書かれたのが 1998 年なので、1980 年台末〜 90 年代のことか) で、コンパイラ作成方法にいくつかの変化があった。

要因その１：あたらしい種類のプログラミング言語の利用

- オブジェクト指向言語: 動的メソッド
- 関数型言語: 入れ子になったスコープ, 第一級の関数クロージャ

これら言語の多くはガベージコレクションを必要とする。
(なにを「クロージャ」と呼ぶべきか、呼ぶべきでないか、については意見がわかれているらしい。SICP は上の「第一級の関数クロージャ」のような文脈で「クロージャ」という語を用いることには否定的)

要因その２：ハードウエアの変化

- 多くのレジスタ集合をもつようになった
- メモリアクセスのペナルティ（遅延）の増大

コンパイラが次のことをうまくやることで、ずっと速くなる：

- 命令のスケジューリング
- 命令とデータキャッシュローカリティのマネージメント

前者は、より多くなったレジスタの有効利用があらたな鍵となっていることを示していると思われる。
後者は、おおきくなったキャッシュミスペナルティ、また、NUMA ライクな非対称性とうまくつき合う必要が生じたことを示している（と思われる）。

この教科書は……

- 半期または通年の授業で用いることを想定したコンパイラの教科書である。
- つぎのことがらを学ぶことが可能である。
  - コンパイラのそれぞれのコンポーネントの背景にある理論
  - 理論を実践するための、プログラミング技術
  - コンパイラをモジュール化するためのインタフェース

この教科書においては、インタフェースとプログラム例は明晰に、かつ、具体的に書かれる。ML で。

コンパイラ作成プロジェクト

本教科書が提案する「学生実験コンパイラ」は、単純だが、現在ひろく使われているいくつかの重要な技術を体験できるようになっている。

- 抽象構文木（「ちゅうしょうこうぶんぎ」と読むの？）: 文法と意味論をもつれさせないために
- 命令選択を、レジスタ割り当てから分離
- コピープロパゲーションにより、コンパイラの早い段階に柔軟性を
- ターゲットマシン依存部の局所化

おおくの学生向けコンパイラと異なり、これは単純だが洗練された back end である。命令選択の後に、適切がレジスタ割り当てが行われる。

章構成と、進捗目標

この輪講では、Part 1 全部と、Part 2 のうちいくつかをやりたい。
海野の希望では、13, 17, 18, 19 をやりたい。

```
10. Liveness → 17. Data Flow → 18. Loop          → 19. SSA
    Analysis        Analysis         Optimization        Form

13. Garbage
    Collection
```

補足等

第一級のオブジェクト

プログラミング要素の第一級身分という考え方は、英国の計算機科学者、Christopher Strachey (1916-1975) によるらしい。

> 一般にプログラミング言語には、計算要素を扱う方法にいろいろな制限があるものだ。制限の殆んどない要素は第一級 (first-class) 身分を持つという。第一級要素の「権利と特権」は：
>
> - 変数として名前がつけられる
> - 手続きに引数として渡せる
> - 手続きの結果として返される
> - データ構造に組み込める
>
> である。Lisp は他の通常のプログラミング言語と違い、手続きに完全な第一級身分を授与した。そのため、効率のよい実装は難しくなったが、表現力として得たものは絶大である。

以上は、SICP (2nd ed. 1996) 1.3.4 「値として返される手続き」より。また、脚注として以下のように述べられている。

> 第一級手続きの実装の大きな代価は、手続きを値として返すためには、手続きを実行しない時でも、自由変数のための記憶場所をとっておく必要があることだ。4.1 節で学ぶ Scheme の実装では、これらの変数は手続きの環境の中に格納してある。

1 導入 (Introduction)

[Program] Straight-line program interpreter

この章では、Straight-line program interpreter をひとつ「作ります」!!
CompoundStm の内部変数 (stm * stm) を取り出す方法がわからなくてまごついたんだけど、パターンを用いて変数に束縛できるもよう。

```
[unno@noether]% sml p10.sml

Standard ML of New Jersey v110.65 [built: Tue Jun 12 09:45:12 2007]
[opening p10.sml]
type id = string
datatype binop = Div | Minus | Plus | Times
datatype stm
  = AssignStm of string * exp
  | CompoundStm of stm * stm
  | PrintStm of exp list
datatype exp
  = EseqExp of stm * exp
  | IdExp of string
  | NumExp of int
  | OpExp of exp * binop * exp
val prog =
  CompoundStm (AssignStm ("a",OpExp #),CompoundStm (AssignStm #,PrintStm #))
  : stm
- val CompoundStm t = prog;
val CompoundStm t = prog;
stdIn:1.5-1.25 Warning: binding not exhaustive
          CompoundStm t = ...
val t =
  (AssignStm ("a",OpExp (#,#,#)),CompoundStm (AssignStm (#,#),PrintStm [#]))
  : stm * stm
- #1(t);
#1(t);
val it = AssignStm ("a",OpExp (NumExp #,Plus,NumExp #)) : stm
```

```
- #2(t);
#2(t);
val it = CompoundStm (AssignStm ("b",EseqExp #),PrintStm [IdExp #]) : stm
- val CompoundStm (lh, rh) = prog;
val CompoundStm (lh, rh) = prog;
stdIn:1.5-2.6 Warning: binding not exhaustive
          CompoundStm (lh,rh) = ...
val lh = AssignStm ("a",OpExp (NumExp #,Plus,NumExp #)) : stm
val rh = CompoundStm (AssignStm ("b",EseqExp #),PrintStm [IdExp #]) : stm
-
```

p10.sml には、PROGRAM 1.5 の宣言につづけて p.10 なかほどにある prog 定義が含まれている。

タプルとして t に束縛して、#1(t) とか #2(t) とかで参照するのも可能なようだし、val CompoundStm (lh, rh) = prog; のように、「多重代入」的な書き方もできるみたい。警告が出ているくらいだから、なんか間違っているのかも。

これと、あとは list の長さを調べるコードが書ければ、p.11 の 1 はできそう。これは、length でいいみたい。

```
- length [1, 2, 3, 4];
length [1, 2, 3, 4];
val it = 4 : int
```

(よだん) ある値に関数を適用する、

```
f(a, b)
```

のような式を、「関数 f にパラメータ a, b を渡して評価」と読むことも可能。でも、f を作用素のようにとらえて、f を値の組 (a, b) に作用させると読むこともできる。

SML の文法とか、エラーメッセージとかをながめていると、後者のような解釈に沿っているのかな、と思った。

(Perler ぴょこいさん向け)

```
#!/bin/env perl

sub print_n_times {
    my($n) = @\_;
    print $n, $/;
    if ($n > 1){
 print_n_times($n - 1);
    }
}

&print_n_times(10);
```

```
for(my($i) = 10; $i > 0; $i--){
    print $i, $/;
}
```

p.11 part 1, 2 作成例

別紙にて。
実行結果をいかに
まず、その１：

```
- maxargs(prog);
val it = 2 : int
```

つぎ、その２：

```
- interp(prog);
I faild to write the print function ;-<
I faild to write the print function ;-<
val it = [IntegerRecord ("b",80),IntegerRecord ("a",8)] : ?.record list
```

おそらく型推論がらみのエラーメッセージを山盛りだしつつ、なんとか、print 以外の部分はつくったのですが、print がつくりきれませんでした。

- table の要素をただの id * int 組 にしていたら、型推論でけんとおこられた（たぶん）ので、datatype で明示して逃れた
- print を書こうとすると、おこられる（泣き

すみませんが、継続課題とさせてください。

回しかたの提案

- なにがあっても二時間できりあげて、場所を変える（笑。
- エクササイズは次にまわす。そのほうが、全員が理解してからやれる。
- エクササイズの発表者は、その日にサイコロできめる。
- さいころで当たったのに、やってなかったら、全員になにかおごること。