Modern Compiler Implementation in ML; 第12章

久野　靖*

1998.11.30

1　第12章: すべてを組み立てる

□ 2〜11章でフロントエンド/バックエンドの部品ひととおりが揃った。

□ それぞれが何をしていてどういうアルゴリズムが使われてるかわかった。

□ 目標:　「できる限り単純に、しかし単純すぎないように」

□ 本章では Tiger 言語とコンパイラを設計するに当たって悩んだことを記す。

1.1　入れ子になった関数について

□ これがあるため静的リンクが必要で複雑に。なければもっと簡単。たとえば FindEscape がいらない。

□ でもあった方がいいと決めた理由:

- 関数型言語では入れ子の関数が極めて有用。
- エスケープを調べることはどのみち必要 (例:　アドレス取得)

1.2　構造化された左辺値

□ Tiger にはレコード値がない (ぜんぶレコードへのポインタ)。

□ これは簡単のため省いた。入れてもあんまり新しいことは必要ない。

*筑波大学大学院経営システム科学専攻

1.3 木構造の中間表現

□ Tree 言語の欠陥: 手続きの入口/出口を表現できない。

- なので Frame モジュールの隠蔽された手続きで処理。
- ということは、Pentium 用の Tree と Sparc 用の Tree は同じにできない。

□ また、Tree に含まれている情報は不完全。

- 解釈実行もできない (view shift が表現されていない)。
- ということは、手続き間最適化もできない。

□ 結局、Tree は低レベル中間表現

- 命令選択や手続き内最適化に適している。

□ 前記の目的にはもっとソースレベルの情報を残した高レベル中間表現。

- そのかわり、それだとソースレベルへの依存は強まる。

1.4 レジスタ割り付け

□ レジスタ彩色は広く使われているが「出来る限り簡単」か?

- このため、広域データフロー解析やレジスタ干渉グラフが必要に。
- ということはバックエンドが大きく。

□ レジスタ彩色を使わないとどうだろう?

- 全部の変数をスタックフレームに入れ、命令で使う時だけテンポラリにロード
- 単一ブロック内の冗長なロードを簡単なブロック内 liveness 解析で除去
- Tree の中間ノードへのレジスタ割り当てはアルゴリズム 11.9、11.10 で

□ しかしそうすると他の部分はぐっと醜くなる。

- Tree を正規化するときのテンポラリもアドホックに処理
- Frame インタフェースもあちこちでレジスタを扱うためもっと複雑に
- いくらでもテンポラリや move を作ってよくて、あとでレジスタ彩色で綺麗にしてもらう、というのは結局単純化になっている。

2 プログラム: 手続きの出入口

□ Frame モジュールの残りの部分を実装せよ。

- マシン依存部分をすべて含んでいる。レジスタセット、呼び出し列、フレームの配置、など。

□ プログラム 12.1 が Frame のシグニチャ。ほとんどはやってある。残りは:

2.1 registers

□ マシンが備えているすべてのレジスタ (彩色の「色」)。

2.2 tempMap

□ 各レジスタごとに Temp.temp を用意→そのレジスタに対応した色を最初から持っているテンポラリ。特定レジスタを使う Assem 命令でこれを使う。手続きの呼び出し命令や出入口のところなど。

2.3 procEntryExit1

□ パラメタを受け取る各レジスタについて、その値を実際に関数内のコードがアクセスする場所 (フレーム内の位置、またはテンポラリ) におさめる。

- それにはたとえば newFrame で Tree.MOVE の列を生成し、procEntryExit1 ではこれを単に先頭にくっつける。

□ また、呼ばれ側保存のレジスタ (含む: 戻り番地レジスタ) を退避/ 回復するコードも必要。

□ もしスピルを実装してなければ、、、

- 入口で全部フレームに入れて最後に戻す。ということは各レジスタについて allocLocal でフレームの場所を確保して Tree.MOVE を生成して保存/復元する。
- そうしておけば、十分レジスタが空いてスピルなしである程度のプログラムがコンパイルできるかも (ある程度以上はどのみち無理)。

□ もしスピルを実装してれば、、、

- ほっといてよい。レジスタが足りてればアロケータはそのまんまにしといてくれる。足りなければ必要なだけスピルしてくれる。
- ただし、最初から色を持ってるテンポラリはスピルされない→ procEntryExit1 で呼ばれ側保存レジスタに対応するテンポラリを用意し、最初に全部そこに move し、最後に move し戻す→スピルされなければこれらの move は coalescing でなくなるのでオーバヘッドにはならない。

2.4 procEntryExit3

□ アセンブラでプロローグ/エピローグを生成。

□ まずフレーム中のパラメタ渡し用領域の大きさを求める。

- これは CALL 命令にいくつパラメタがついてるかで分かる。
- しかし Assem になった後はこの情報がない→ procEntryExit2 で予めコードをスキャンしてこの大きさを記録しておくか、可能な最大値を使う

□ これがわかれば出入口やスタック調整のコードが出せる (prologue, epilogue)

2.5 string

□ 文字列リテラルは STRING になってるが、これを最後はアセンブラの文字リテラルにしないといけない。

- 関数 Frame.string でアセンブリ言語命令を返す。例:

  ```
  Frame.string(Temp.namedLabel("L3"), "hello")
  ```

  に対して次のようなのを出す:

  ```
  L3: .ascii "hello"\n
  ```

  関連: p169.

3 プログラム: 動かす

□ あなたの Tiger コンパイラが動くコードを出すようにしなさい。

□ プログラミングプロジェクト、、、