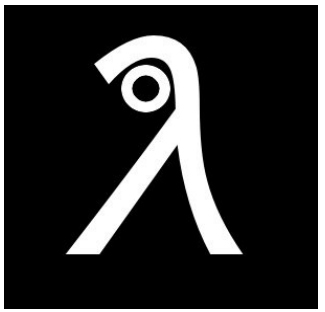
λ コンパイラ係 λ

C[12]斑コンパイラ係
xenophobia(桑原 拓也)

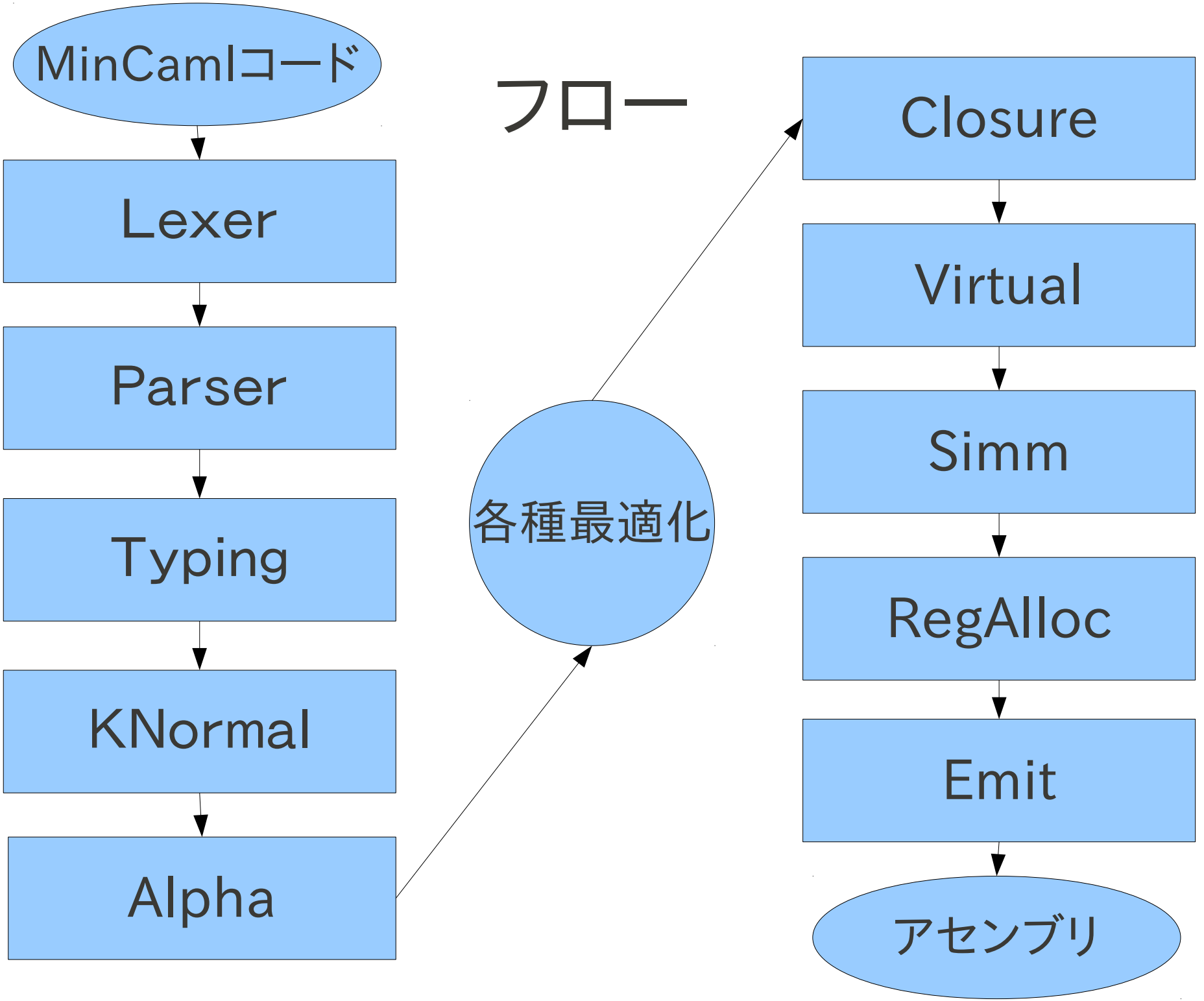


話すこと

- デバッグを乗りきるには
- 最適化について

その前に……

- 多分もう誰かが説明してると思うけどまずはMinCamlコンパイラについて
- OCaml製のOCamlサブセット(MinCaml)コンパイラ
- 大半の人はこのコードを改造してコンパイラを作ることになる（と思う）


フロー
MinCamlコード
Lexer
Parser
Typing
KNormal
Alpha
各種最適化
Closure
Virtual
Simm
RegAlloc
Emit
アセンブリ

話すこと

- デバッグを乗りきるには
- 最適化について

話すこと

- 狂気のデバッグを乗りきるには
- 最適化について

#デバッグやれ

- とにかくプログラムがでかい＆コーナーケースがどんどん出てくるのでバグりまくる
- min-rtは約2300行のOCamlプログラム→副作用・混みいった関数呼び出し・再帰呼び出しと、なんでもありのプログラムなので、1つでもバグがあると（ほぼ）動いてくれることはない。

テクニック

- 1.各種中間表現のビジュアライザ（プリティプリンタ）を作る
- 2.サンプルを作る＆テストを行なう
- 3.デバッグ機能を作る／使う

1.中間表現が見えるようにする

- MinCamlコンパイラの内部ではコードが様々な中間表現（木）に変換されながら処理される
  - syntax.t
  - kNormal.t
  - asm.t
  - etc...
- これらのプリティプリンタをまず作っておく（課題としてsyntax.t, kNormal.tのプリティプリンタ作成課題が出るだろうが、それ以外のデータ構造も見えるようにしておくべき）
- それを見て改造・デバッグをする（直にアセンブリ見るよりずっと楽）
- つーかコンパイラ係のCPU実験は大体こいつらとにらめっこするクソゲー

1.中間表現が見えるようにする

- インタフェースを

  printer :: FilePath → MidData → IO ()

  とするのではなく

  printer :: FilePath → MidData → IO MidData

  としておくとよい
- つまり、(デバッグモードで)コンパイラを走らせると外部ファイルにそのときフローを通った中間表現を出力するようにする。

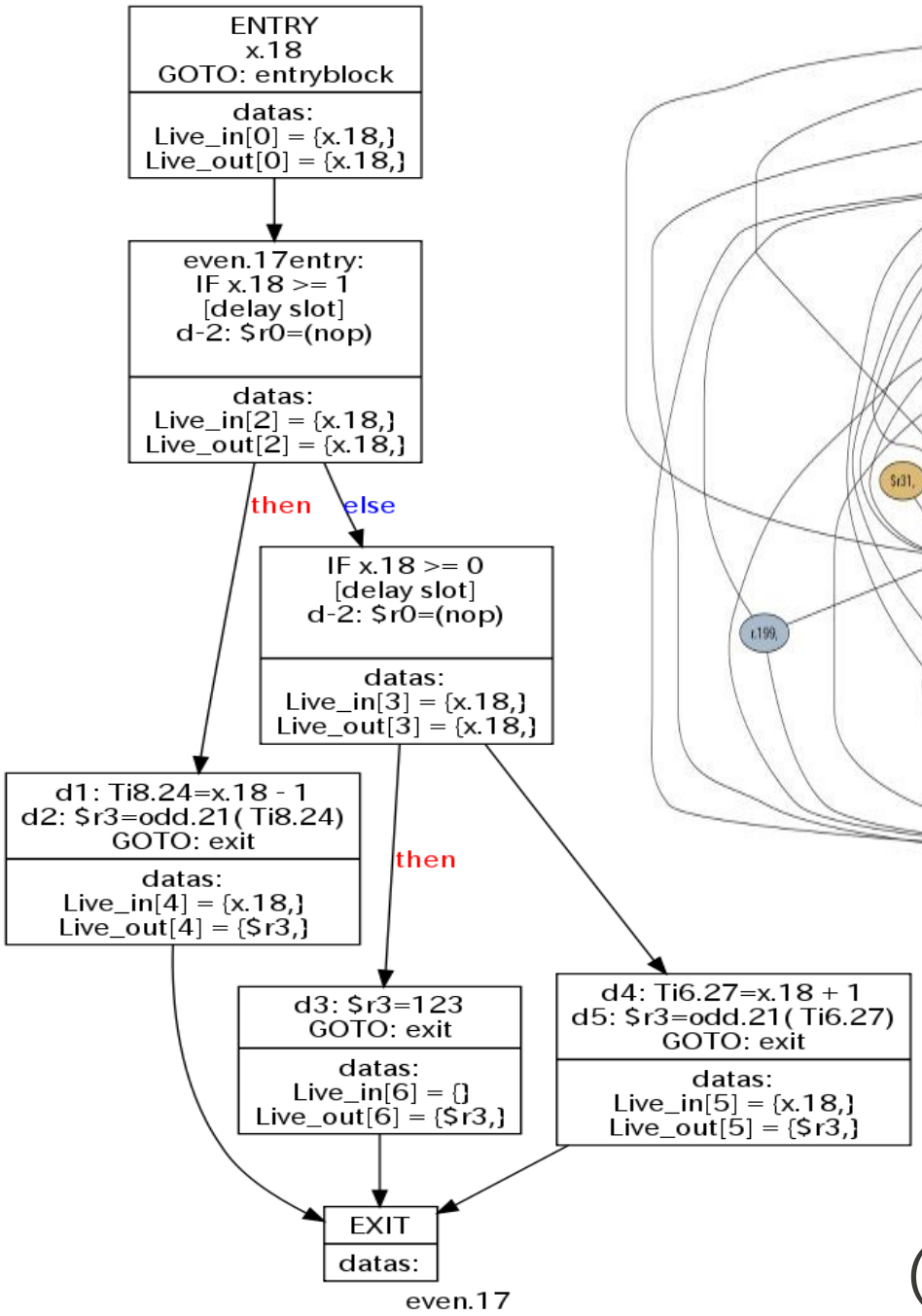
- 例）syntax.t

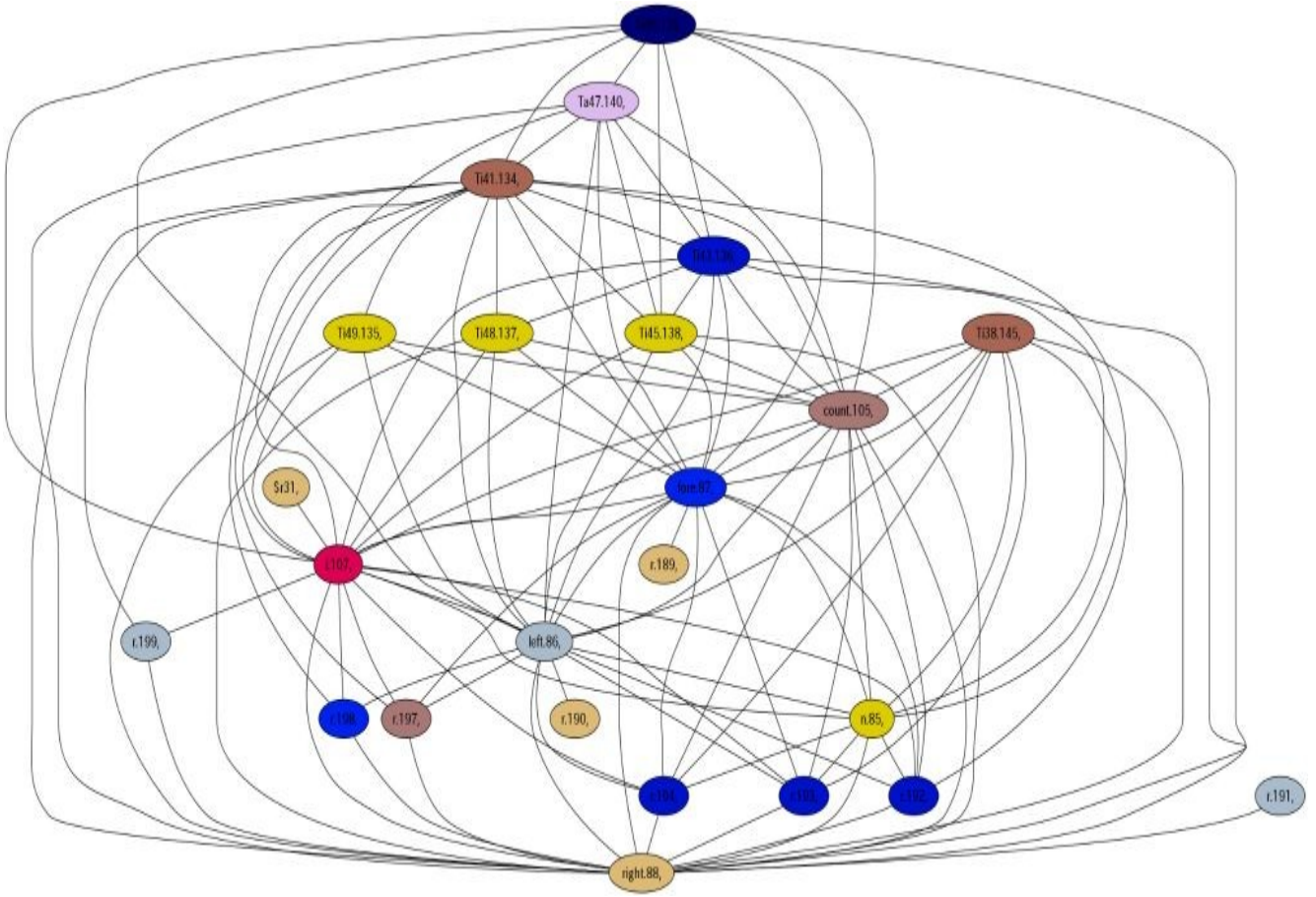
```
Let x(:Undefined) =
  Var(read_int)
    ()
In
  If
    Not
      LE
        Var(x)
        Int(2)
  Then
    If
      Not
        LE
          Var(x)
          Int(0)
    Then
      Var(print_int)
        Int(0)
    Else
      Var(print_int)
        Int(1)
  Else
    If
      Not
        LE
          Var(x)
          Int(0)
    Then
      Var(print_int)
        Int(0)
    Else
      Var(print_int)
        Int(1)
```

- 例）asm.t

```
Tu1.17 = ()
x.9 = read_int(Embedding) ()
 = if x.9 >= 3 then
   [delay slot
    $r0 = ()
    $r0 = ()
    $r0 = ()
    goto: dummy
   ]
    Ti8.12 = 0
     = min_caml_print_int (Ti8.12)
   else
     = if x.9 >= 1 then
       [delay slot
        $r0 = ()
        $r0 = ()
        $r0 = ()
        goto: dummy
       ]
        Ti5.15 = 0
         = min_caml_print_int (Ti5.15)
       else
        Ti4.16 = 1
         = min_caml_print_int (Ti4.16)
```

- フローグラフ・レジスタ干渉グラフを作ったときも、視覚的にグラフが見られるようにしておくと捗る（＆テンションが上がる）





（画像はgraphvizを用いて出力）

2.サンプル入力を作る

- MinCamlコンパイラには、いくつかのMinCamlサンプルプログラムが付属している。
- min-rtがコンパイルできないとき、またそうでなくてもデバッグの際にバグの正体がなんであるかを探るときにはこれらサンプルプログラムをコンパイルして検証することが多くなる。
- サンプルをコンパイル→正解と出力比較という簡易テストスクリプトを作っておくと捗る。

2.サンプル入力を作る

- が、正直付属のサンプルは弱い。
- 自分である程度の大きさを持ったプログラムを作っておくとよい。
- NQueen、Mandelbrot、GCM、etc...
- 関数呼び出しが複雑なもの、分岐が多いものなど、タイプが異なるサンプルを作っておくとよい。
- 発行命令数が大きすぎてもバグの特定が難しいので、複雑ながらもサイズの小さいものを。

3.デバッガ的な機能

- シミュレータとも重なるが、コンパイラ側でもできることがある。
- 私の斑ではdebug()という特殊な関数を用意し、これをMinCamlプログラムの任意の場所に挟むとその場所にdbgという命令（意味はnopと同じ）が挿入されるようにし、シミュレータがこれをブレークポイントとして認識するようにした。
- このような機能を追加しておくと、関数呼び出しを検出することができたりして便利。

3.デバッガ的な機能

- 欲しい機能があったらシミュレータ係に頼みましょう。
- レジスタの値の追跡・メモリの値の追跡・ジャンプ先アドレス異常検知など

話すこと

- 狂気のデバッグを乗りきるには
- 最適化について

最適化

- 基本的には、最適化モジュールを作ってさっきのフローの間に挟む形になるはず。

最適化

- これが

```
let lexbuf outchan l = (* バッファをコンパイルしてチャンネルへ出力する *)
  Id.counter := 0;
  Typing.extenv := M.empty;
  Emit.f outchan
    (RegAlloc.f
       (Simm.f
          (Virtual.f
             (Closure.f
                (iter !limit
                   (Alpha.f
                      (KNormal.f
                         (Typing.f
                            (Parser.exp Lexer.token l)))))))))
```

最適化

- 最終的にはこんなカオスな感じになります（ビジュアライザが10行ぐらい混じってるけど）

```
let lexbuf outchan l = (* バッファをコンパイルしてチャンネルへ出力する (caml2html: main_lexbuf) *)
  Id.counter := 0;
  Typing.extenv := M.empty;
  Emit.f outchan
    ((ifout $ AsmProgPrinter.f "../../CPU_OUTPUT/Register_Allocated_Asm_Prog.txt")
       (ConstructProgram.f
          ((if false then FlowGraphVisualizer.f "../../CPU_OUTPUT/FlowGraph/FlowGraph_Escaped" else (fun e -> e))
             (DelaySlotInsertion.f
                (Escape.f
                   (GlobalRegAlloc.f
                      (DeadCodeElim.f
                         (UnusedArgsReduction.f
                            (LiveVariablesAnalysis.f
                               (LoopEntryInsertion.f
                                  (CallFunctionSearch.f
                                     (FlowGraph.f
                                        ((ifout $ AsmProgPrinter.f "../../CPU_OUTPUT/Optimized_Asm_Prog.txt")
                                           (Simm.f
                                              ((ifout $ AsmProgPrinter.f "../../CPU_OUTPUT/Virtual_after_Asm_Prog.txt")
                                                 (Virtual.f
                                                    ((ifout $ ClosureProgPrinter.f "../../CPU_OUTPUT/Tuple_Eliminated_Closure_Prog.txt")
                                                       (BetaClosure.f
                                                          (TupleElim.f
                                                             ((ifout $ ClosureProgPrinter.f "../../CPU_OUTPUT/Tuple_Flattening_Closure_Prog.txt")
                                                                (TupleFlattening.f
                                                                   ((ifout $ ClosureProgPrinter.f "../../CPU_OUTPUT/Closure_Prog.txt")
                                                                      (Closure.f
                                                                         ((ifout $ KNormalPrinter.f "../../CPU_OUTPUT/Optimized_Knormal_t_Output.txt")
                                                                            (iter !limit
                                                                               ((ifout $ KNormalPrinter.f "../../CPU_OUTPUT/Alpha_Transform_Knormal_t_Output.txt")
                                                                                  (Alpha.f
                                                                                     ((ifout $ KNormalPrinter.f "../../CPU_OUTPUT/Init_Knormal_t_Output.txt")
                                                                                        (KNormal.f
                                                                                           ((ifout $ SyntaxPrinter.f "../../CPU_OUTPUT/Typed_Syntax_t_Output.txt")
                                                                                              (Typing.f
                                                                                                 ((ifout $ SyntaxPrinter.f "../../CPU_OUTPUT/Syntax_t_Output.txt")
                                                                                                    (Parser.exp Lexer.token l)))))))))))))))))))))))))))))))))
```

いろいろな最適化

- インライン展開
- 定数畳み込み
- 局所共通部分式除去
- タプル展開
- 不要引数除去
- デッドコード除去
- グローバルレジスタ割り付け
- 即値最適化
- バックエンドでの小細工

いろいろな最適化

- インライン展開＆定数畳み込み
- もとからあるモジュール
- インライン展開：Leaf関数（再帰しない関数）を検出し、展開するように変更
- 定数畳み込み：いろいろやったけど忘れた……

いろいろな最適化

- 局所共通部分式除去
- 同じブロック内で同じ式を計算していたら冗長なので除去
- 意外とバグりやすいので注意

いろいろな最適化

- タプル展開
- let func a b =

  let (c, d) = a in … → let func c d b = …
- ヒント1）

  let func a b = … → let func c d b =

  let a = (c, d) in …

  のように書きかえ→aがいらなかったら削除

いろいろな最適化

- 即値最適化
- SPARC用のMinCamlコンパイラを改造する場合は即値が13bit以下のものだけ最適化するようになっているので、これを自分たちのアーキテクチャに合わせて拡大・縮小

いろいろな最適化

- バックエンドでの小細工
- 正直これが一番効いた（emit.mlだけで850行ぐらい）
- 特に効果があったやつ
  - ライブラリ関数のインラインアセンブラ化：ライブラリ関数は呼びだすのではなくインラインアセンブラをベタ張りして高速化（関数呼び出し時のオーバーヘッドがなくなる）
  - （遅延スロット埋め）

いろいろな最適化

- 不要引数除去
- デッドコード除去
- グローバルレジスタ割り付け

- 詳しくは虎本嫁
- データ構造を最初にきっちり設計しとかないとカオスと化す。

おまけ

- 私が長期間ハマったバグ紹介
- 小数のサイズ
  - SPARCでは小数は8byteになってしまっている＆数カ所これがマジックナンバーで埋めこまれている
- 浮動小数点演算の誤差
  - Mandelbrot集合の誤差がいつまでも残り、原因不明だった
    - →OCaml/Cの演算器と手作り浮動小数演算器の誤差が原因。Mandelbrot集合の数pixelの誤差は気にしないこと。

おまけ

- 私が長期間ハマったバグ紹介
- フローグラフ→中間表現変換
  - if文で分岐したブロックからどこへ戻るべきか、はよく考察しないとハマる
  - 適当に組みたてれば動くとかいう人もいるけど、そんなことはなかった