TIC'98 の報告

sumii

1998 年 4 月 23 日

1 はじめに

もう一ヶ月ほどたってしまいましたが、4 月末に京都で開かれた The Second International Workshop on Types in Compilation (TIC'98) [1]の報告をします。tutorial が半日と、workshop が二日半あったのですが、前半のほうが後半よりも（僕にとっては）おもろしい話が多かったので、ちょっとアンバランスな報告になっているかもしれません。

2 第 0 日 (4/24)

2.1 Tutorial: An introduction to compiling functional programming languages, Xavier Leroy (INRIA Rocquencourt)

ocaml や LinuxThread で有名な Leroy [2] による tutorial。4 部構成。

第 1 部は関数型言語の紹介と、closure の表現に関する話。3 年生の冬学期に田浦さんの授業でやったのと同じようなものなので、省略。

第 2 部は flow analyses（複数形）について。プログラムの各部分がとりうる値をおおざっぱに見積もって、その結果を利用する。たとえば `if a then b else c` という式のとりうる値の集合は、`b` のとりうる値の集合と `c` のとりうる値の集合の和という具合。Scheme などで type checking を省く (soft typing) とか、集合の要素が一つでそれが小さい関数だったら inlining を行うとか、いろいろな最適化に応用できる。

第 3 部はデータの表現のしかたについて。tag のつけかたの工夫とか、boxing/unboxing とかの話。たとえば ML の `fn x => (x, x)` のような、polymorphic function の実装を考えてみる。すぐに思いつくのは 1 word に収まらないデータはすべてヒープに allocate して pointer をやりとりする方法だが、これはあまりに inefficient。逆にデータの実現は型によって変えるけれども、実行時にその型をやりとりする (type passing) ことにより、データのありかやサイズを知る方法もあるが、実行時に型情報を操作しないといけないので、やはりあまり効率的ではない。そこで、普段は型によって異なる実現を用い、polymorphic function に渡すときには、ヒープに allocate して pointer を渡すようにし (box)、逆に polymorphic function の戻り値は pointer として受け取り、その先にある値を読み出して使う (unbox) 方法が考えられる。あるいは、最初は引数や戻り値は全部 boxed にしておいて、inlining や flow analyses で「box して unbox」とか「unbox して box」とかを削除していく方法もある。たとえば

```
let
  fun f x y = x * y + 1.23
in
  f 4.56 7.89
```

[1] `http://www.kurims.kyoto-u.ac.jp/%7Eohori/TIC98.html`

[2] `http://pauillac.inria.fr/%7Exleroy/` の写真は怖いが、実物は全然違って、よくいるフランス人青年という印象だった。

```
end
```

というプログラムは、とりあえず

```
let
  fun f x y = unbox(x) * unbox(y) + 1.23
in
  f box(4.56) box(7.89)
end
```

としておき、 inlining を行うと

```
unbox(box(4.56)) * unbox(box(7.89)) + 1.23
```

となるので、 unbox(box(...)) を削除して

```
4.56 * 7.89 + 1.23
```

となる。 TIC'97 の Leroy の論文によると、この方針に基づく方法が一番速いそうだ。

第 4 部はコード生成の話。中間言語レベルでの様々な最適化、 stack frame に frame の内容に関する情報を付加して tag-free GC などに利用する話など。

3 第 1 日 (4/25)

3.1 Invited talk: Intermediate Representations and Optimizations for Object-Oriented Languages, Craig Chambers (University of Washington)

すみません、この話の内容はほとんど忘れてしまいました。確か Vortex という (オブジェクト指向言語向けの) 中間言語 (とそのコンパイラ) があって、その上でいろいろな最適化をするという話だったと思います。わりと一般的な最適化の他に、プログラムを実際に実行して得られる統計的な情報をフィードバックして program database というものを作り、もう一度コンパイルするとき最適化に利用するという話などをしていた記憶があります。

3.2 Compiling Java to a Typed Lambada-Calculus: A Preliminary Report, Andrew Wright (InterTrust) and Suresh Jagannathan and Cristian Ungureanu (NEC Research Institute) and Aaron Hertzmann (New York University)

題名の通りの内容。中間言語が typed lambda calculus であることによって、関数型言語に対する (特に type-based な) 様々な最適化や、生成されるコードの正当性の保証などを行うことができるという主張。具体的にどういうメリットがあるのかは正直いってよくわからなかった。

3.3 An Approach to improve Locality using Sandwich Types, Daniela Genius, Martin Trapp, and Wolf Zimmermann (Universität Karlsruhe)

linked list や hash table のような container class を実現するときに、中身をばらばらに allocate するとメモリアクセスにおいて不利になるので、各 object ごとに heap を用意することによって locality を向上させよう、というお話。「中身が外部から直接触られることはない」という class を自動的に発見して、そういう最適化を行う。 list 操作を繰り返す簡単なテストでは、 total の実行時間が最大で 5 倍速くなる場合もあったという。

3.4 Formal Description for Collecting Reachable Garbage via Dynamic Type Inference, Haruo Hosoya and Akinori Yonezawa (University of Tokyo)

細谷さんの type inference GC のお話。

3.5 Invited talk: An Overview of Type-directed Partial Evaluation, Olivier Danvy (BRICS, University of Aarhus)

この話は、当時全然予備知識がなくて理解できなかったのですが、最近ちょっと勉強してみて、ようやくあのとき何をいっていたのかわかってきました。

たとえば Scheme で `(define v (lambda (a) (lambda (b) a)))` のようにどこかで定義された値 v を与えられたとする。そのとき、その与えられた値 v だけから、逆にそれを表す式 `'(lambda (x) (lambda (y) x))` を得ることできるだろうか？

一見すると (逆アセンブルでもしない限り) そんなことはできないように思えるが、実はおおざっぱにいって[3]その値の型がわかっていれば、それを表す式の $\beta\eta$-normal form を得られる、というのがこの type-directed partial evalution のお話。

上の例でいくと、v の型は `S -> T -> S` のような形をしている。それがわかっていれば、v が関数であることはわかるので、`'(lambda (x) hoge)` という形をしているはずである。そうしたら、次に hoge の部分を表す式を求める。v は関数なのだから `(v 'x)` のように関数適用を行うことができる。[4] さらに、`(v 'x)` は `T -> S` という型を持っているから、`'(lambda (y) puha)` という形をしているはず。よって `((v 'x) 'y)` のように関数適用を行うことができるはず。

```
> (define v (lambda (a) (lambda (b) a)))
> v
#<procedure v>
> (v 'x)
#<procedure>
> ((v 'x) 'y)
x
```

これで puha は x とわかった。結局 v を表す式は `'(lambda (x) (lambda (y) x))` とわかる。

これを一般的にやるプログラムは以下の通り。

```
> (define id 0)
> (define (gensym!)
    (set! id (1+ id))
    (string->symbol (string-append "x" (number->string id))))
> (define (reify t v)
    (cond
     [(eq? (car t) 'base)
      v]
     [(eq? (car t) 'func)
      (let ([x (gensym!)])
        `(lambda (,x) ,(reify (cddr t) (v (reflect (cadr t) x)))))]))
> (define (reflect t e)
```

[3]正確にいうと、closed でないとどうとか、base type の出現位置がどうとかいう制限がある。

[4]Scheme だからこの時点で型エラーは起きない。 ML だと話はもっとややこしくなる。

```
    (cond
     [(eq? (car t) 'base)
      e]
     [(eq? (car t) 'func)
      (lambda (v) (reflect (cddr t) `(,e ,(reify (cadr t) v))))]))
```

使用例は次の通り。

```
> (define S
    (lambda (f) (lambda (g) (lambda (x) ((f x) (g x))))))
> (define K
    (lambda (a) (lambda (b) a)))
> (define I ((S K) K))
> (I 123)
123
> (I "abc")
"abc"
> (define t '(func . ((base . T) . (base . T))))
> (reify t I)
(lambda (x1) x1)
```

3.6 Optimal Type Lifting, Bratin Saha and Zhong Shao (Yale University)

polymorphic function を type passing を用いて実装した場合、 type passing のコストが問題になる。この話はそのコストを減らして、静的な定数でおさえられるというもの。たとえば

```
ΛS.λx:S.
  let
    f = ΛT.λy:T.(x,y)
  in
    f[int](123)
```

という関数だったら、λx:S の内側にΛT や f[int] があるが、それを逆にしてしまう。そのとき、x が f の自由変数のままでは困るので、ラムダ抽象して f を呼び出すときに与える。

```
ΛS.
  let
    f = ΛT.λx:S.λy:T.(x,y)
    f' = f[int]
  in
    λx:S f'(x)(123)
```

これと同様の方法で、すべての (type abstraction と) type application は value abstraction の外側に持ってくることができるので、リンク時ないしは実行開始時にしか起こらない。つまり、実行中の cost には影響しない。もちろん、上の例からもわかるように、この変換によって value abstraction/application は増えてしまうわけだが、実験結果を見るとおしなべて実行時間は短縮している。[5]

[5] type passing といえば tag-free GC の話もあるので、メモリ使用量についても具体的な測定結果がほしかった。おそらく減少するはずだとは思うが。

3.7 Polymorphic Equality - No Tags Required, Martin Elsman (University of Copenhagen)

ML の `fn (x, y) => (x = y)` のような polymorphic equality を実現するのに、従来のように tag に頼るのではなくて、 equality を判定する関数を引数として渡すように変換するという方法の話。 ML Kit 上での実験結果によると、最大で 2 倍以上の速度向上を得ており、最悪の場合でも悪化はほとんどしていない。

4 番外編 (Workshop Reception)

25 日の夜にホテルで開かれた reception party。前日の tutorial で聞き逃した部分も含めていろいろ聞きたいことがあったので、ちょっとがんばって Leroy とお話をした。

> 前から疑問に思っていたのだが、 ocaml のような statically typed な言語でなぜ tagging が必要なのか。 type passing をすれば polymorphism があっても GC などでは困らないのでは。 C との interface をとるのにも邪魔ではないか。

ということを聞いたら、

> tagging は実際にはかなり efficient に実装できるし、 type passing をすると型を表現したものを GC 時に interpret しないといけないので、 ML のように GC が頻繁に起こる言語では効率が悪い。

とのお答え。そこで、假家君の卒論をふまえて

> じゃあ、普段は region-based なメモリ管理を行い、ときどき GC をすればいいのでは。

というと、

> それだと dangling pointer という問題があって、 GC がうまくいかない。

と予想通りの反応だったので、

> これは人の受け売りなんだけど、そもそも dangling pointer が発生するのは region inference で得られた静的な情報を利用した結果なのだから、それを GC にも教えてやれば解決できるという話があるのだが。

といってみたら、”That may be good.” とのお言葉だった。

ocaml の method dispatch の実装 (特に `fun obj -> obj # meth;;` のような関数の場合) についても知りたいことがあったのだが、そのときは聞き忘れてしまった。[6] Leroy と話した他は、大堀先生に polymorphic record calculus の質問 (Rémy のと大堀先生のとの違いなど) をしたり、お茶の水女子大の佐藤さんという方とお話をしたり、それから (もちろん) 飲み食いもしたりして過ごす。

5 第 2 日 (4/26)

5.1 Invited talk: Region-Based Memory Management, Mads Tofte (University of Copenhagen)

region inference のお話。特に目新しいところはなかったと思う。「もうすぐ ML Kit 3 が出る」ということぐらいか。

[6] 後で caml のメーリングリストに投げたら、 Rémy が答えていたが。

5.2 A Type-Based Semantics for User-Defined Marshalling in Polymorphic Languages, Dominic Duggan (Stevens Institute of Technology)

ごめんなさい、この発表は全然理解できませんでした。「どんな型の object に対してもうまく働く一般的な marshalling function を定義するのは不可能に近いので、型に応じた marshalling function を、 typecase を使って user が define できるようにしよう、という既存の研究があるが、それをふまえて…」という感じの出だしで始まったのですが、その後ごちゃごちゃとした syntax やら rule やらが次々と出てきて、結局どういう貢献があったのかまったくわかりませんでした。

5.3 Stack-Based Typed Assembly Language, Greg Morrisett, Karl Crary, Neal Glew, and David Walker (Cornell University)

TAL (Typed Assembly Language) に stack をつけくわえました、というお話。 TAL は (中間言語としての仮想的な) アセンブリに型をつけて、正当性を示したり最適化を行ったりするのに利用しようというもの。それに explicit に stack を付け加えて、 heap ではなく stack allocation も取り扱い、 Java や Pascal もサポートできるだけの表現力を得たとしている。

5.4 Formalizing Resource Allocation (Preliminary Report), Peter Thiemann (Universität Tübingen)

A-normal form というのは中間言語の一種で、たとえば

```
(+ (* x (+ 1 y)) (* 2 z))
```

を

```
let t0 = (+ 1 y) in
  let t1 = (* x t0) in
    let t2 = (* 2 z) in
      (+ t1 t2)
```

のように変換したもの。 CPS のように評価順序や control flow が explicit で、計算の途中結果にもすべて名前がついている。今回の話は、 A-normal form のコンパイルにおける register allocation や spilling などの話を formalize して、その正しさなどを証明しました、というもの。

5.5 Invited talk: Type-based Analysis of Concurrent Programs, Naoki Kobayashi (University of Tokyo)

小林さんのお話。 process calculus における linearity や deadlock-freedom のための type system と、それに基づく reasoning や最適化など。今回唯一の並列関係の話題だった。

5.6 Determination of Dynamic Method Dispatches Using Run-time Code Generation, Nobuhisa Fujinami (Sony Computer Science Laboratory, Tokyo)

run-time code generation は、実行時にはじめて得られる値を利用して partial evaluation を行うというもの。静的な解析では不可能な最適化を行うことができる (特に run-time constant が存在する場合) 。この話は C++ における virtual function dispatch の overhead を、 run time code generation で軽減するというもの。たとえばレイトレのプログラムでは 1.4 倍の速度向上が得られたとのこと。

5.7 Strong Normalization by Type-Directed Partial Evaluation and Run-Time Code Generation, Vincent Balat (École Normale Supérieure de Cachan) and Olivier Danvy (BRICS, University of Aarhus)

TDPE (type-directed partial evaluation) において、 text のかわりに bytecode を出力させ、それを ocaml の機能を利用して load するという話。たとえば (極端な例だと思うが) church 数を定義して、与えられた church 数を 1000 増やす関数を定義し、 TDPE を行って normalize すると、そのまま適用した場合と比べて 4,800 倍の速度が得られるそうだ。もちろん TDPE 自体にかかる時間はあるが、何度も使う関数に対しては有効だろう。

6 第 3 日 (4/27)

6.1 Invited talk: A Toolkit for Constructing Type-Based Program Analyses, Alex Aiken (University of California, Berkeley)

BANE という、 constraint-based なプログラム解析を行うツールの宣伝。

一般に type system に基づいてプログラムを解析するときは型推論をすることになるが、

$$\frac{f\colon \sigma \to \tau \quad x\colon \sigma}{f x\colon \tau}$$

という simply typed λ-calculus の型付け規則一つをとってみても、 e の引数の型と f の型は暗黙のうちに等しいことが要求されている。それを

$$\frac{f\colon \sigma \to \tau \quad x\colon \sigma' \quad \sigma = \sigma'}{f x\colon \tau}$$

という風に $\sigma = \sigma'$ という explicit な constraint をつけて考えれば、型推論も contraint solving と考えられる。

そこで BANE は集合に関する constraint を解く scalable な engine を与えてくれる、のだそうだ。[7]

6.2 How Generic is a Generic Back-End? Andrew Bernard, Robert Harper, and Peter Lee (Carnegie Mellon University)

TILT という ML コンパイラ と、その back end の MLRISC の話。どういう novelty があるのかはよくわからなかった。

6.3 Optimizing ML Using a Hierarchy of Monadic Types, Andrew Tolmach (Portland State University)

ML のような call-by-value で副作用もある言語において、最適化などのプログラム変換を naive に行うと、停止するはずの関数が停止しなくなったり、副作用の起こり方が変わってしまったりするおそれがある。そこで effect inference のようなことをして副作用を見積もり、正しい変換が行えるようにしようという話。普通の effect inference よりも simple だと主張するが、具体的にどこがどう違うのかは不明。

6.4 Type-Directed Continuation Allocation, Zhong Shao and Valery Trifonov (Yale University)

たとえば ML と C のように、二つの異なる言語で書かれた関数を互いに呼び合うときに解決する必要のある問題、特に call/cc の扱いについて。二つの言語を一つの typed な中間言語に変換し、その上で check を行うこと

[7]なお、論文には “The distribution may be obtained from the BANE home-page at http://bane.cs.berkeley.edu.” とあるが、そこを見てもいまだに公開はされておらず ”The alpha distribution of BANE is still in preparation. It will definitely be available early May.” と書かれている。

により、 ML から C の関数を呼んでいるような場合でもあまり efficiency を損なわずに safety が保証できるとのこと。

7 感想

発表中も頻繁に質問が出たり議論が起こったりしていたのが印象的だった。[8]

[8]あと、京大の北白川学舎（一泊 1,600 円）は意外とまともな宿泊施設だった。