Dependently Typed Assembly Language について

(Memorandum: Preview for PL and SIGPOPL Meeting)

oiwa

平成 14 年 1 月 8 日

はじめに

明けましておめでとうございます。御迷惑をおかけして済みません。ちょっと発表資料をまとめるには間に合わなかったので、再来週のミーティングでサーベイする予定の DTAL に関するメモを公開しておきます。図は文献 [1] を参照して下さい。

1 DTAL とは

- PCC, TAL, DTAL など → 証明付き低級コード
- TAL: 型付きアセンブリ言語
  - Type Safe, Pointer Safe → “型付け可能 ⇒ 安全なプログラム実行”
  - 配列境界チェックは全てのメモリアクセス命令に内蔵
    - 全てのメモリアクセスにオーバヘッド
    - JVM Language などと同じ方式
  - 安全性の検証は容易: シンプルな型チェッカ
- **DTAL**: TAL + Dependent Type
  - Dependent Type =「値に依存した型」
    - 例 1: $x : int\ array(3)$: 長さ 3 の配列
    - 例 2: $y : int(2)$: 値が 2 の整数
    - 例 3: $z : \exists a : \{a : int \mid 0 \leq a < 3\}.int(a)$: 値が 3 以下の自然数
    - 例 4: $w : \exists a : \{a : int \mid 4 \leq a\}.int\ array(a)$: 長さ 4 以上の配列
    - 上のような型が付いている時に、$x[y]$ とか $w[z]$ とかはアクセス違反を引き起こさないことが静的に保証可能
    - 通常の型 int は $\exists a : \{a : int\}.int(a)$ とみなされる
  - 依存型導入の第一目的: 配列境界チェックの除去
    - ループ外への吐き出し、共通チェックの省略、etc ...

2 DTAL の概要

- 構文: Figure 3, Figure 4 (文献の p. 5)
  - 型の表現を除いてはほぼ TAL 類似
  - load/store 命令は範囲チェックをしないことを想定している
  - 依存型に現れる値の式を index sorts と呼ぶ
    - 実行時の値の型である int 等とは独立 → 定式化に重要
    - 実行時の値とは単値型 int(x) を通じて結合
- Dynamic Semantics: Figure 6
  - ほぼ TAL と同じと思ってよい
  - 範囲外アクセスに対しては stuck する
    - 型付け可能なら stuck が起こらないことをあとで保証
- Static Semantics: Figure 7 から 13 辺り... ... 一見すると複雑... ... まぁこの論文のキモなので。

  配列範囲チェックがどうやってうまくいくのか、に注目してみてみると... ...
  - まず (type-store-array) の side condition に注目。
    明示的な依存型などによって導入された制約条件は全部 ϕ に入っている。
    $\phi; \Delta; \Sigma \vdash v : int(y)$ で v が $int(y)$ 型であること、つまり v の値が y に課せられる制約を満たすことを要求。
    同様に、$\phi, \Delta, \Sigma \vdash r_d : \tau\ array(x)$ で、配列 r_d のサイズが x に課せられる制約を満たすことを要求。
    次に、$\phi \vDash 0 \leq y < x$ (x は配列型のサイズ部分) で、その y がとり得る値が実際に配列範囲に入っていることを要求。
    つまりこの 3 つの式で、配列の範囲が正しいことが保証できない限り store 文は型チェッカを通過してくれないことを意味している。
    このルールだけで、例えば前ページの例の $x[y]$ とか $w[z]$ なんかは判定できる。後者の場合、実際のプログラムは例えば

    ```
    wz: { i: int, j: int | 0 <= i < 3 /\ j >= 4 }
        [rz: int(i), rw: int array(j)]
            load r2, rw(rz)
    ```

    のようになり、w と z の不等式制約は ϕ に入るので、

    $$0 \leq i < 3 \wedge j \geq 4 \vDash 0 \leq i < j$$

    が証明すべき制約式となる。
  - 次に、では動的チェックをする場合どうなるかなのだが、動的チェックをするプログラムコードは

    `arraysize r,` *r_s*`; sub r, r, v; blte r,` *error_handler*

のようなコードで、キモとなるのは分岐命令 blte の型付け規則。論文では省略されているので (^^;, 代わりに bne 辺りの規則を見てみると、(type-beq) の 2 つ目の side condition $(\phi, x \neq 0; \Delta; \Sigma \vdash I)$ に $x \neq 0$ という条件が増えているのがわかる。つまり、条件分岐すると、各分枝の型チェックの際には制約式が新たに追加されるのである。この追加された条件で、load/store の型チェックは成功することができる。これは、元のプログラムに立ちかえって見れば、分岐を通過したところで比較された値の型がより詳細になっているともいえる。

原論文だと Figure 14, 15 辺りで説明しているので参照されたい。

- Soundness: 当然 sound じゃないと使い物にならない。論文の 5 章で Progress の性質 (well-typed なら stuck しない) を証明しているのだが、証明追いかけるの辛いっす (^^;

3 拡張

- Sum Type の導入 (6 節)
  - Tagged Union Type を導入。増えたルールはそんなに多くなくて、choose$(x, \tau_0, \ldots, \tau_{n-1})$ という型コンストラクタを導入。これは、タグ x が 0 の時 τ_0, 1 の時 tau_1, (以下略...) な型で、x がさっきの ϕ で v に一意に決定できるなら τ_v とみなせる、という coercion rule が与えられていて、これが trick になっている。

    でもって、本命の Sum Type である $sum(\tau_0, \ldots, \tau_{n-1})$ は、

    $$sum(\tau_0, \ldots, \tau_{n-1}) := \exists a : nat_n.int(a) * choose(a, \tau_0, \ldots, \tau_{n-1})$$

    (但し $nat_n = \{a : int \mid 0 \leq a < n\}$) と定義されている。一旦 tuple の第 1 要素で分岐すれば、上述の coercion rule によって第 2 要素を特定の型で取り出せるので、命令は一切増えていない。

    しかし、list にこの表現使うと効率が悪くてしょうがないので、list 専用の null value と、null pointer check だけは特別に拡張している。当然っちゃ当然ではあるが、この部分、他に応用するには adhoc というかなんというか、使い辛いかなぁ... ... と、この論文の ICFP 版を読んでちょっと期待してしまった私は思ったりして。

4 DTAL でできることとできないこと

で、私の研究の本題としては、Fail-safe C を DTAL にコンパイルできたらいいなぁ、と考えていたわけだが、配列の範囲チェックにターゲットを限定しているので、他の性質の表現に使うのは結構厄介かも知れないなと思ったりしている。

まずそもそも型の $\exists$ がないので、package みたいなもの、例えば C++ の method dispatch の玩具みたいなものとして $\exists\tau : *.(\tau \rightarrow int) * \tau\ array$ みたいなものはどうも書けなさそうである。とすると、型が違っていた際の処理ルーチンを値の型ごとに作って、そこに分岐してやって... ... ... みたいな処理は書くのがめんどくさいかなぁ。プログラムに出てくる型を全部並べて sum type にしてやればいいのかもしれないけど、それもちょっとねぇ... ... 。

というわけで、まぁ DTAL を拡張してもいいんだけど、だったら型システムとか制約の生成法とかを参考にしつつ、Proof Carrying Code に直接コード吐いてもいいのかなぁ、と思いはじめている。

参考文献

[1] Hongwei Xi and Robert Harper. A Dependently Typed Assembly Language. Technical Report CSE-99-008, Oregon Graduate Institute, July 1999.

[2] Hongwei Xi and Robert Harper. A Dependently Typed Assembly Language. Proc. ICFP, Florence, Italy. September, 2001.

おまけ: 冬休みの活動

DTAL 論文読み まぁ、これ。

プリンタ買った 論文を家で両面印刷したい、というだけの欲求で買ってしまいました (^^;。ブラザーの HL-1670N、8 万円で A4 両面印刷、PS 互換インタプリタ、ネットワーク対応。ちょっと PS インタプリタがバグってるけど速くて快適。調子にのって DTAL の Technical Report (39pages) とかがしがし印刷しまくっていたらあっというまにページカウンタが 500 を突破した... ... 。

初日の出 また犬吠崎まで遠征。でも厚い雲に阻まれてわりと残念な結果に終わりました。まぁ雲から出るのは見られたけど。

風邪ひいた はい。御迷惑おかけして済みません。_o_