Simple extensions to simply typed λ-calculus

for YL TSPL group

おおいわ

May 14, 2001

この章の目的

- 現実の言語の世界とここまでの話の Gap を埋める
  - 簡単に扱える範囲で現実の言語にある機能を simply-typed λ-calculus に拡張

Various Base Types (1)

- Unstructures Value の種類を追加
  - 例: Float (3.1415), String (“Hello”) 等
  - 型理論としては、その値に対する計算の具体的な性質はまったく関係ない

  → 話を抽象化・一般化しましょう

Various Base Types (2)

- 未知の基本型 A で $\lambda_{\rightarrow}$ を拡張
  - Figure 11.1
  - also called Uninterpreted/Atomic types

  - $(\lambda x{:}A.\ x) : A \rightarrow A$
  - $(\lambda f{:}A\rightarrow A.\ \lambda x{:}A.\ f\ (f\ x)) : (A\rightarrow A)\rightarrow A\rightarrow A$

Unit Type

- Singleton Type (唯一の要素をもつ型)
  - 唯一の値 unit : Unit (see Figure 11-2)
  - 主に副作用がある世界で有用:
    代入など返り値には意味がない式の
    返り値型として用いる

Sequencing (1)

- Sequencing: 2つの式を順に評価
  - 副作用ありの世界で重要
  - e; f ≡ (λx:Unit. f) e
  - (直接定義する時の規則:
    E-SEQ, E-SEQNEXT, T-SEQ)

Sequencing (2)

- Derived Form
  - 他の式の「省略形」として定義される構文
  - Theorem 11.3.1:
    Sequencing が Derived form である証明
    - (Proof: Induction on structure)
      - Top Level が sequencing の場合: E-Seq と E-App2, E-SeqNext と E-AppAbs が対応することを示す
      - それ以外の場合: e(t) の再帰定義の構造から自明

Wildcard

- 無名の変数束縛: 値を使わない
  - Derived form:
    $(\lambda \_:S.\ t) \equiv (\lambda x:S.\ t)$ [x fresh]

Ascription

- ある式をある型だと「みなす」
  - 構文: e as T
  - もちろんその型が正しいときに限る
    - Figure 11-3
  - 用途:
    - Documentation
    - type-checker の出力の「見た目」を制御
    - 型の抽象化, 限定化 (15章)

Let binding (1)

- 式中で変数に値を束縛する構文
  - let x = e_1 in e_2
  - 評価規則、型付け規則: Figure 11-4
    - ここでは Call-by-value semantics で定義

Let binding (2)

- Let は Derived Form?
  - let x = e_1 in e_2 ≡ (λ x:T_1.e_2) e_1
    - 問題点: T1 の型をどこから導出するか?
    - 解: Type checker で得られる (T-Let の T_1)
    - 型付けでは Derived Form としては扱えない
    - ML のような Unification-based な型多相を持つ言語では Let は特別扱い (22章)

Products, Tuples

- Product: 値のペア、直積
  - 値: {x, y}, 型: T × U
  - 操作: Pair, Projection (要素を取り出す)
  - Figure 11-6: straightforward
- Tuple: 3つ以上の値の組に拡張
  - Figure 11-7

Records

- ラベル付けされた値の組に拡張
  - let o = { name = “Oiwa”, year = 24}
    in o.year → 24
  - o : { name : String, year : Nat }

Record Patterns

- let をパターンマッチで拡張
  - let { name = n; year = y } = o in y → 24