TaPL

omosan0627

November 24, 2023

3 型無し算術式

抽象構文? 帰納的定義・証明? 評価? 実行時エラーのモデル化?
Chapter5: 型無しラムダ、名前束縛、代入 Chapter8: 型システム、静的型付け
Chapter9: 静的型付けラムダ

3.1 導入

文法: 本書では BNF
構文: 項、値などの組かな.
項: 計算の構文的表現 (つまりメタ変数 t に代入することができる表現)
式: あらゆる種類の構文的表現 (項式、条件式など)??
メタ変数: メタ言語の変数. (対象言語の変数ではなく). 何らかの項のための placement holder
抽象構文:?
値: 項の部分集合で、評価の結果

3.2 構文

帰納的な項の定義: 推論規則を満たす最小の集合具体的な項の定義: 前提を持つ規則を 1 回適用した項を集める。それを有限回繰り返して得られる集合
完全帰納法は全て帰納ステップになっているとみなせる。

3.3 項に関する帰納法

構造的帰納法: 「各項 s に対して、s の任意の直接の部分項 r に対して $P(r)$ がなりたつとき、$P(s)$ が証明できる」ならば、全ての s に対して $P(s)$ が成立するこれは具体的な項の定義から証明できる. これは一般的な帰納法になっている.

3.4 意味論のスタイル

表示的意味論 (モデル理論っぽ)、公理的意味論 (ホーア論理とか?) もあるが、操作的意味論をこの本では扱う

3.5 評価

評価関係: 関係は (項, 項) の集合であることに注意.
インスタンス: 規則の結論や前提のメタ変数それぞれに対し、一貫して同じ項による置き換えを行ったものである.
規則がある関係によって満たされる: 規則の任意のインスタンスについて, 結論がその関係に属するか, または前提の内の一つが属さないことである.
1 ステップ評価関係:規則を満たす最小の二項関係. これは項の定義同様具体的に構成できるし、構造的帰納法も使える. 導出に関する帰納法と言う.
1 ステップ評価の決定性: Coq での証明ができません. 項 t が正規形:t $\rightarrow$ t' となる t' が存在しないとき. 全ての値は正規形である. 正規形は値とは限らず、そうでないとき行き詰まりという。状態実行時エラーの解析に使われるかも.
多ステップ評価関係 $\rightarrow^*$: 1 ステップ評価の反射的推移的閉包. つまり有限回の 1 ステップ評価で到達できる項の関係. これも推論規則から定義できます.
停止尺度: 評価の停止性の証明で使われる関数のこと. 項について単調減少. 構文要素ってなんだ?

5 型無しラムダ計算

基礎

一般的な手続き、関数を引数に値を渡すことで具体化する.
ラムダ計算: 全てが関数. 変数、ラムダ抽象、関数適用の 3 種類の項のみで構成.

抽象構文

具象構文: プログラマが直接読み書きする文字列抽象構文: 単純な内部的表現. ラベル付き木 (抽象構文木 AST) として表現される.
字句解析器: 文字列 $\rightarrow$ トークン列
構文解析器: トークン列 $\rightarrow$ 抽象構文木
優先順位や結合法則を決めておくと、括弧を減らせる。
本書では抽象構文木を念頭に置くが、関数適用は左結合、ラムダ抽象の本体はできるだけ右に展開する.
$\lambda x.\lambda y.xyx$ は $(\lambda x.(\lambda y.(xy)))x$ と解釈することができるが、$\lambda x.(\lambda y.((xy)x))$ と解釈することにする。
変数とメタ変数の違いをここからは意識する必要がある. ただ文脈から大体わかるね.

スコープ

抽象 $\lambda x.t$ の本体 t の中に変数 x が現れるとき、その x の出現は束縛されているという. x を囲む抽象によって束縛されていない所に x が現れるとき、その x の出現は自由であるという.
自由変数のない項は閉じているという. 閉じた項はコンビネータとも呼ばれる.

操作的意味論

純粋なラムダ計算において「計算」が意味するのはただ一つ、引数に対する関数の適用である.

$$(\lambda x.t_{12})\ t_2 \rightarrow [x \mapsto t_2]t_{12}$$

ここで $[x \mapsto t_2]t_{12}$ は「t_{12} の中の x の自由な出現をすべて t_2 で置き換えることによって得られる項」を意味する. 例えば $(\lambda x.x(\lambda x.x))(u\ r)$ は $(u\ r)(\lambda x.x)$ と評価される.
$(\lambda x.t_{12})\ t_2$ を簡約基 (reducible expression) と言い、上記の規則で簡約基を書き換える操作のことをベータ簡約と呼ぶ.
評価戦略:

- 完全ベータ簡約: 任意の簡約基がいつでも簡約されうる.
- 正規順序戦略: 最も左で最も外側の簡約基が最初に簡約される.
- 名前呼び戦略: 正規順序戦略+抽象の内部での簡約を許さない.
- 値呼び戦略: 最も左で外側の簡約基を簡約するのだが、右側が既に値に簡約されている簡約基のみだけを簡約する. この本の戦略.

純粋なラムダ計算については λ、ブール値算術式を伴う拡張された体系に対しては λNB を用いる.

形式的議論

第 3 章と同じく、項を定義する抽象構文は、帰納的に定義された抽象構文木の集合に対する簡易的な表記として行う.

代入

第 6 章の de Bruijn 表現とは別の定義を使う。次のように定義すればよい.

$$
\begin{aligned}
[x \mapsto s]x &= s \\
[x \mapsto s]y &= y \\
[x \mapsto s](\lambda y.t_1) &= \begin{cases} \lambda y.t_1 & (y = x) \\ \lambda y.[x \mapsto s]t_1 & (y \neq x \wedge y \neq FV(s)) \end{cases} \\
[x \mapsto s](t_1 t_2) &= ([x \mapsto s]t_1)([x \mapsto s]t_2)
\end{aligned}
$$

ただ、$y \neq x \wedge y \neq FV(s)$ の場合で、代入が部分的になってしまっていることがわかる。なのでアルファ変換で t_1 で自由変数である y を s でも t_1 でも現れない変数に置き換えてあげれば良い。

操作的意味論

値がラムダ抽象値であることに注意.

6 項の名無し表現

アルファ変換は新しい変数名を与える事によって代入の問題を解決したが、それを決める際にバグを埋め込んでしまう可能性もある. 実際ちゃんと停止することが保証できる代入をアルファ変換を使って実装するのは難しそう. なので変数、項、抽象の標準形を考えることにする。

項と文脈

変数がそれを囲む k 番目の λ によって束縛されているとき、変数名を k に置き換える. これを de Bruijn インデックスや、静的距離という言葉で表現する. 例えば $\lambda m.\lambda n.\lambda s.\lambda z.ms(nsz)$ は $\lambda.\lambda.\lambda.\lambda.31(210)$ となる.

Definition 0.1 (項) $\mathcal{T}$ を、以下の条件を満たす集合の最小の族 $\{\mathcal{T}_0, \mathcal{T}_1, ...\}$ とする.

1. $0 \leq k < n$ ならば $k \in \mathcal{T}_n$
2. $t_1 \in \mathcal{T}_n$ かつ $n > 0$ ならば $\lambda.t_1 \in \mathcal{T}_{n-1}$
3. $t_1 \in \mathcal{T}_n$ かつ $t_2 \in \mathcal{T}_n$ ならば $(t_1 t_2) \in \mathcal{T}_n$

各 $\mathcal{T}_n$ の要素を n 項と呼ぶ.

$\mathcal{T}_n$ の要素は、高々n 種類の自由変数を持つ項であり、それらの変数には 0 から $n-1$ までの数が振られる. 自由変数を含む項を扱うためには、名前付け文脈の仕組みが必要. 例えば $\Gamma = x \mapsto 4, y \mapsto 3$ のとき、$\lambda w.yw$ を $\lambda.4\ 0$ と表現する. つまり、変数の加算集合 $\mathcal{V}$ から自由変数に使う集合 $\Gamma \subset \mathcal{V}$ を取り出して、Γ について、適切な自然数を割り振ることにする。

シフトと代入

Definition 0.2 (シフト) 項 t の、打ち切り値 c 以上の d 個シフトを $\uparrow_c^d (t)$ と書き、次のように定義する.

$$\uparrow_c^d (k) = \begin{cases} k & (k < c) \\ k + d & (k \geq c) \end{cases}$$
$$\uparrow_c^d (\lambda.t_1) = \lambda.\uparrow_{c+1}^d (t_1)$$
$$\uparrow_c^d (t_1 t_2) = \uparrow_c^d (t_1) \uparrow_c^d (t_2)$$

Definition 0.3 (代入) 項 t における、変数番号 j への項 s の代入を $[j \mapsto s]t$ と書き、次のように定義する.

$$[j \mapsto s]k = \begin{cases} s & (k = j) \\ k & (otherwise) \end{cases}$$
$$[j \mapsto s](\lambda.t_1) = \lambda[j+1 \mapsto \uparrow^1 (s)]t_1$$
$$[j \mapsto s](t_1 t_2) = ([j \mapsto s]t_1)([j \mapsto s]t_2)$$

評価

$(\lambda.t_{12})v_2 \to \uparrow^{-1} ([0 \mapsto \uparrow^1 (v_2)]t_{12})$

8 型つき算術式

第 3 章の言語に関して、静的な形で強化することを考える。
項を評価すると、結果は値になるか、行き詰まり状態になるかのどちらかであった。
行き詰まり状態になるような無意味なプログラムを、項を評価せずとも排除したい。よって型 Nat と Bool を導入する。
メタ変数は S, T, U はこの本を通じて型の上を動くものとする。
項の型の分析は保守的であり、静的な情報しか用いない。つまり、if (iszero 0) than 0 else false のような項が、実際には行き止まり状態にはならないにもかかわらず、型を持たないということである。

型付け関係

型付け判断式: $t : T$ のこと
型付け関係: 項と型の最小の二項関係。(注:評価は 2 つの項の二項関係でしたね)
型付け可能: 項 t に対してある T が存在して $t : T$ となる。
逆転補題 (生成補題): ある項が型を持つとき、部分項がどのように型付けされるかについての補題。$t : T$ が妥当な型付け判断式のとき (つまり型付け関係に含まれるような式のとき)、その証明がどのように生成されたか示してくれる。
型付け導出: 型付け規則のインスタンスの木。型付け規則に基づいた演繹

安全性=進行+保存

進行: 正しく型付けされた項は行き詰まり状態でないこと
保存: 正しく型付けされた項が評価できるならば、評価後の項もまた正しく型付けされているという事。主部簡約と呼ばれることもある。項 t が $t : T$ の主語 (主部) になっており、主部の簡約のもとこの性質が保存される。