Typed Aritmetic Expressions and Typed Lambda Calculus

21 de Maio de 2015

Tipos (Aritmetic Expressions)

- Avaliação de termos sem tipos:
  - ✓ Resulta em um valor (`true`, `false` ou `nv` [0 ou `succ nv`]); ou
  - ✓ Trava em algum estágio onde nenhuma regra de avaliação se aplica (`pred false`).

Tipos (Aritmetic Expressions)

- Objetivo dos tipos:
  - ✓ Identificar termos com erros que irão ocasionar um travamento antes de avaliá-los;
  - ✓ Tipos criados: `Bool` (booleanos) e `Nat` (naturais);
  - ✓ De forma estática: `t : T` (`t` possui tipo `T`) - indica que `t` irá avaliar para um valor de forma correta (sem travamento) dispensando a necessidade de avaliá-lo.
  - ✓ Análise conservadora: usa apenas a informação estática, não permitindo expressões como `if true then 0 else false`, mesmo que elas não ocasionem um travamento.

Relação dos tipos

- Conjunto de regras de inferência que atribuem tipos ao termos, onde `t : T` (o termo `t` tem tipo `T`):

Regras de tipos para booleanos

$$\texttt{true} : \texttt{Bool} \qquad \text{(T-True)}$$

$$\texttt{false} : \texttt{Bool} \qquad \text{(T-False)}$$

$$\frac{\texttt{t}_1 : \texttt{Bool} \qquad \texttt{t}_2 : \texttt{T} \qquad \texttt{t}_3 : \texttt{T}}{\texttt{if}\ \texttt{t}_1\ \texttt{then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3 : \texttt{T}} \qquad \text{(T-If)}$$

Adaptado de (Pierce, 2002).

Relação dos tipos

Regras de tipos para números

$$\texttt{0} : \texttt{Nat} \qquad \text{(T-Zero)}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{succ t}_1 : \texttt{Nat}} \qquad \text{(T-Succ)}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{pred t}_1 : \texttt{Nat}} \qquad \text{(T-Pred)}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{iszero t}_1 : \texttt{Bool}} \qquad \text{(T-IsZero)}$$

Relação dos tipos

- Derivação de tipos:
  - ✓ Árvores de instâncias de tipos.
  - ✓ Exemplo: `if iszero 0 then 0 else pred 0`.

Relação dos tipos

Theorem (Teorema da unicidade dos tipos)

Indica que, se um termo possui um tipo, esse tipo é único e existe apenas uma regra de inferência que deriva a sua construção.

Segurança = Progresso + Preservação (8.3)

- Garantia que a avaliação de um programa bem tipado não travará.
- Dois pilares da segurança:
  - ✓ **Progresso:** termos bem tipados são avaliados completamente (até sua forma normal).
  - ✓ **Preservação:** um passo de avaliação de um termo bem tipado resulta em outro termo igualmente bem tipado.
- No decorrer do Capítulo 8.3, Pierce (2002) mostra exemplos de Progresso e Conservação, onde os termos são classificados em alguma regra de inferência (como T-TRUE, T-IF e T-SUCC) e é aplicada alguma regra de avaliação a ele (E-IFTRUE, E-IF, E-SUCC, ...).

Formas Canônicas

Lemma (Formas Canônicas I)

Se v é um valor do tipo `Bool` *então v é* `true` *ou* `false`*.*

$$
\begin{array}{lll}
t & ::= & \\
 & \texttt{true} & \text{valor ``verdadeiro''} \\
 & \texttt{false} & \text{valor ``falso''}
\end{array}
$$

Lemma (Formas Canônicas II)

Se v é um valor do tipo `Nat` *então v é um valor numérico conforme a gramática:*

$$
\begin{array}{lll}
v & ::= \ldots & \\
 & nv & \text{valor numérico} \\
nv & ::= & \\
 & 0 & \text{valor ``zero''} \\
 & \texttt{succ}\ nv & \text{sucessor}
\end{array}
$$

Inversão da Relação de Tipos

Lemma (Inversão da Relação de Tipos)

1. *Se* `true` $:$ *R então R* $=$ `Bool`.
2. *Se* `false` $:$ *R então R* $=$ `Bool`.
3. *Se* `if t1 then t2 else t3` $:$ *R, então* `t1` $:$ `Bool`, `t2` $:$ *R, e* `t3` $:$ *R.*
4. *Se* `0` $:$ *R, então R* $=$ `Nat`.
5. *Se* `succ t1` $:$ *R, então R* $=$ `Nat` *e* `t1` $:$ `Nat`.
6. *Se* `pred t1` $:$ *R, então R* $=$ `Nat` *e* `t1` $:$ `Nat`.
7. *Se* `iszero t1` $:$ *R, então R* $=$ `Bool` *e* `t1` $:$ `Nat`.

Prova: Imediato a partir das Formas Canônicas.

Teorema: Progresso

Suponha `t` um termo bem tipado. Então ou `t` é um valor ou ele será avaliado (`t` $\rightarrow$ `t'`).

- Por indução, T-True, T-Zero e T-False são casos diretos: itens 1, 4 e 2 do Lema de Relação de Tipos, respectivamente.
- `T-If`:
  - ✓ se a guarda do `if` é um valor então ele será `true` ou `false` conforme as Formas Canônicas e uma das seguintes regras serão aplicadas: `E-IfTrue` ou `E-IfFalse`.
  - ✓ senão `t` $\rightarrow$ `t'` e aplica-se `T-If` sob a guarda avaliada `t'`.
- `T-Succ`:
  - se `t` for um valor então ele é do tipo numérico (`Nat`) e, segundo as Formas Canônicas, avalia para `0` ou `succ nv`
  - senão `t` $\rightarrow$ `t'` via E-Succ.

Teorema: Progresso

Suponha `t` um termo bem tipado. Então ou `t` é um valor ou ele será avaliado (`t` $\rightarrow$ `t'`).

- `T-Pred`:
  - ✓ se `t` for um valor então ele é do tipo numérico (`Nat`) e pode ser avaliado por uma das seguintes regras: E-PredZero ou E-PredSucc e

    `pred t : R` então `R = Nat` e `T = Nat`

  - ✓ senão `t` $\rightarrow$ `t'` e `pred t` $\rightarrow$ `pred t'`.
- `T-IsZero`
  - ✓ Mesmas condições de T-Pred com a ressalva de que as regras aplicáveis a `t` são E-IszeroZero, E-IszeroSucc e E-IsZero

Teorema: Preservação

Suponha `t : T` e `t` $\rightarrow$ `t'`. Então `t' : T`.

- `T-True`, `T-False` e `T-Zero`:
  - ✓ t é um valor e o teorema não se aplica.
- `T-If`

  `t = if t1 then t2 else t3 t1 : Bool t2 : T t3: T`

  - zim zum

Tipos de Funções

- Para construir um tipo que combine booleanos com primitivas do cálculo lambda é preciso adicionar uma classificação para os termos cuja avaliação resulta em uma função;
- A fim de ter certeza de que a função irá se comportar corretamente quando for chamada, precisamos manter o controle de qual o tipo de argumento que ela espera.

- Para manter esta informação, podemos utilizar um novo tipo:

$$
\begin{aligned}
T ::= & \\
& \mathit{Bool} \\
& T \to T
\end{aligned}
$$

Tipos de Funções

- Exemplo:

$$Bool \to Bool$$
$$(Bool \to Bool) \to (Bool \to Bool)$$

Relação de Tipos

- Para saber o tipo de uma abstração como "$\lambda x.t$", precisamos calcular o que acontece quando essa abstração é aplicada a algum argumento;
- Abordagem utilizada agora: anotar a abstração com o tipo esperado para seus argumentos.
- Exemplo: "$\lambda x.t$" ser'a "$\lambda x : T1.t2$"

Relação de Tipos

- Termos podem conter abstrações aninhadas. Com isso em mente, utilizaremos $\Gamma \vdash t : T$ onde Γ é um conjunto com as variáveis livres de t e seus respectivos tipos. Sendo assim, a regra de tipo para abstrações será:

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1.t_2 : T_1 \to T_2} \qquad (T - Abs)$$

Relação de Tipos

- A regra para variável é:

$$\frac{x{:}T \in \Gamma}{\Gamma \vdash x : T} \qquad (T - Var)$$

- A regra para aplicação é:

$$\frac{\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \qquad \Gamma \vdash t_2{:}T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \qquad (T - App)$$

B.C. Pierce. *Types and Programming Languages*. MIT Press, 2002. ISBN 9780262162098.