Universidade Federal de Santa Maria
Programa de Pós-Graduação em Informática - PPGI

Expressões Aritméticas Tipadas e Cálculo Lambda Simplesmente Tipado

Linguagens de Programação – ELC921

Profª Drª Juliana Kaiser Vizzotto

Alunos: Alberto Kummer, Camila Nogueira, Daniel di Domenico, Fernando Campagnolo, José Puiati

8.1 Tipos (Expressões Aritméticas Tipadas)

- Avaliação de termos sem tipos:
  - ✓ Resulta em um valor (`true`, `false` ou `nv` [`0` ou `succ nv`]).
  - ✓ Ou trava em algum estágio da avaliação no qual nenhuma regra de avaliação se aplica (`pred false`).

8.1 Tipos (Expressões Aritméticas Tipadas)

- Objetivo dos tipos:
  - ✓ Identificar termos com erros que irão ocasionar um travamento antes de avaliá-los.
  - ✓ Tipos criados: `Bool` (*booleans*) 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.

8.2 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 booleans

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

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

$$\frac{t_1 : \texttt{Bool} \quad t_2 : T \quad t_3 : T}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : T} \qquad \text{(T-If)}$$

Adaptado de (Pierce, 2002).

Regras de tipos para números

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

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

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

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

Adaptado de (Pierce, 2002).

8.2 Relação dos tipos

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

8.2 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.

8.3 Segurança = Progresso + Preservação

- Garantias 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 noutro 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, ...).

8.3.1 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:

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

8.2.1 Inversão da Relação de Tipos

Lemma (Inversão da Relação de Tipos)

1. Se $\mathtt{true} : R$, então $R = \mathit{Bool}$.
2. Se $\mathtt{false} : R$, então $R = \mathit{Bool}$.
3. Se $\mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3 : R$, então $t_1 : \mathit{Bool}$, $t_2 : R$, e $t_3 : R$.
4. Se $0 : R$, então $R = \mathit{Nat}$.
5. Se $\mathtt{succ}\ t_1 : R$, então $R = \mathit{Nat}$ e $t_1 : \mathit{Nat}$.
6. Se $\mathtt{pred}\ t_1 : R$, então $R = \mathit{Nat}$ e $t_1 : \mathit{Nat}$.
7. Se $\mathtt{iszero}\ t_1 : R$, então $R = \mathit{Bool}$ e $t_1 : \mathit{Nat}$.

Prova: Imediato a partir das Formas Canônicas.

8.3.2 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-FALSE e T-ZERO são casos diretos: itens 1, 4 e 2 do Lema de Relação de Tipos, respectivamente.
- T-If:

$$\texttt{t} = \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3$$

$$\texttt{t}_1 \texttt{ : Bool} \qquad \texttt{t}_2 \texttt{ : T} \qquad \texttt{t}_3 \texttt{ : T}$$

  - ✓ 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:

$$\texttt{t} = \texttt{if } \texttt{t}_1\texttt{' then } \texttt{t}_2 \texttt{ else } \texttt{t}_3$$

8.3.2 Teorema: Progresso

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

- T-Succ:

$$\texttt{t} = \texttt{succ } \texttt{t}_1 \qquad \texttt{t}_1 : \texttt{Nat}$$

  - ✓ Se $\texttt{t}_1$ 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' e aplica-se E-Succ:

$$\texttt{t} = \texttt{succ } \texttt{t}_1\texttt{'}$$

8.3.2 Teorema: Progresso

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

- T-Pred:

$$\texttt{t} = \texttt{pred } \texttt{t}_1 \qquad \texttt{t}_1 : \texttt{Nat}$$

  - ✓ Se $\texttt{t}_1$ for um valor, então ele é do tipo numérico (Nat) e pode ser avaliado por uma das seguintes regras: E-PredZero ou E-PredSucc
  - ✓ Senão t $\rightarrow$ t' e aplica-se E-Pred

$$\texttt{pred } \texttt{t}_1'$$

8.3.2 Teorema: Progresso

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

- T-IsZero:

$$\texttt{t} = \texttt{iszero } \texttt{t}_1 \qquad \texttt{t}_1 : \texttt{ Nat}$$

  - ✓ Mesmas condições de T-Pred com a ressalva de que as regras aplicáveis a t são E-IszeroZero, E-IszeroSucc e E-IsZero

8.3.3 Teorema: Preservação

Suponha $t : T$ e $t \to t'$. Então $t' : T$.

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

$$t = \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3$$
$$t_1 : \texttt{Bool} \qquad t_2 : T \quad t_3 : T$$

  - ✓ Caso $t_1 : \texttt{Bool}$: Aplica-se E-IfTrue ou E-IfFalse.
  - ✓ Se t_1 não é um valor, $t_1 \to t_1'$ e então aplica-se E-If

$$t' = \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3$$
$$t_1' : \texttt{Bool} \qquad t_2 : T \quad t_3 : T$$

8.3.3 Teorema: Preservação

Suponha $\texttt{t} : \texttt{T}$ e $\texttt{t} \rightarrow \texttt{t'}$. Então $\texttt{t'} : \texttt{T}$.

- T-SUCC:

$$\texttt{t} = \texttt{succ } \texttt{t}_1 \qquad \texttt{T} = \texttt{Nat} \qquad \texttt{t}_1 : \texttt{Nat}$$

  - ✓ Só é possível avaliar via E-SUCC:

$$\frac{\texttt{t}_1 \rightarrow \texttt{t}_1\texttt{'}}{\texttt{succ } \texttt{t}_1 \rightarrow \texttt{succ } \texttt{t}_1\texttt{1}} \text{ (E-SUCC)}$$

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.

Tipos de Funcoes

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

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

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: "$\lambda x.t$" será "$\lambda x : T1.t2$"

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:

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

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:

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

9.3.1 - Lemma [Inversion of the typing relation]

- If $\Gamma \vdash x : R$, then $x : R \in \Gamma$.
- If $\Gamma \vdash \lambda x : T_1.t_2 : R$, then $R : T_1 \to R_2$ for some R_2 with $\Gamma, x : T1 \vdash t_2 : R_2$.
- If $\Gamma \vdash t_1\ t_2 : R$, then there is some type T_{11} for some R_2 with $\Gamma, x : T_1 \vdash t_2 : R_2$.
- If $\Gamma \vdash \mathit{true} : \mathit{Bool}$, then $R : \mathit{Bool}$.
- If $\Gamma \vdash \mathit{false} : \mathit{Bool}$, then $R : \mathit{Bool}$.
- If $\Gamma \vdash$ if then t_2 else $t_3 : R$, then $\Gamma \vdash t_1 : \mathit{Bool}$ and $t_1\ t_2 : R$.

9.3.3 - Theorem [Uniqueness of Types]

- Give a context Γ and a Term t, t must have at most one type.
- [Proof] If $\Gamma \vdash x : T$ and $\Gamma \vdash x : S$ then $S = T$ by T-Var.

9.3.4 - Lemma [Canonical Forms]

- If v : Bool, then v is *true* | *false*.
- If v : $T_1 \to T_2$, then $v = \lambda x : T_1.t_2$.

9.3.5 - Theorem [Progress]

- If $\vdash k : T$ then either k is a value, or $k \rightarrow k'$ to some k'.
- The variable case cannot occur.
- The abstraction occur, since abstractions are values.
- The application is not so simple.
  - Case T-App: $k' = k_1\ k_2 \mid \vdash k_1 : T_{11} \rightarrow T_{12}$ and $k_2 : T_{11}$.
  - k_1 is a term or it can make a step, in the same way k_2.
  - If k_1 can make a step, then applies E-App1.
  - If k_1 is a value and k_2 can make a step, then applies E-App2.
  - If booth are value, then the canonical form of k_1 is $\lambda x : T_{11}.k_{12}$, and applies E-AppAbs.

9.3.6 - Lemma [Permutation]

- If $\Gamma \vdash e : T$ and Γ' is a permutation of Γ, then $\Gamma' \vdash e : T$.
- Proof: $\Gamma \vdash e : T$

9.3.7 - Lemma [Weakening]

- If $\Gamma \vdash e : T$ and $x \notin dom(\Gamma)$, then $\Gamma, x : S \vdash t : T$.
- Proof: $\Gamma \vdash e : T$

9.3.8 - Lemma [Preservation of Types Under the Substitution][1]

- If $\Gamma, x : T' \vdash e : T$, and $\Gamma \vdash e' : T'$, then $\Gamma \vdash [x \to e']e : T$
- Proof: If $\Gamma, x : T' \vdash e : T$.

? Case T-Var:

  ? $\mathbf{t = z}$ **with** $\mathbf{z : T} \in \mathbf{(\Gamma, x : S)}$
  - If $z = x$, then $[x \to s]z = s$
  - Otherwise, $[x \to s]z = z$

? Case T-Abs

  ? $\mathbf{t} = \lambda\ \mathbf{y :}\ T_2\ \mathbf{.}\ t_1$
  $\mathbf{T} = T_2 \to T_1$
  $\Gamma\ \mathbf{x : S, y :}\ T_2 \vdash t_2 : T_1$
  - $x \neq y$, and $y \notin FV(s)$.
  - Permutation $\Gamma y : T_2, x : S \vdash t_1 : T_1$
  - Weakening $\Gamma, y : T_2 \vdash s : S$
  - By induction $\Gamma, y : T_2 \vdash [x \to s]t_1 : T_1$
  - By T-Abs $\Gamma \vdash \lambda y : T_2.[x \to s]t_1 : T_2 \to T_1$

9.3.8 - Lemma [Preservation of Types Under the Substitution][2]

? Case T-App:
- By induction $\Gamma \vdash [x \to s]t_1 : T_2 \to T_1$ and $\Gamma \vdash [x \to s]t_2 : T_2$
- By T-App $\Gamma \vdash [x \to s]t_1[x \to s]t_2 : T$

? Case T-True and T-False
- ? **t = true**
  t = false
  T = Bool
- $[x \to s]t = (true \mid false)$, $\Gamma \vdash [x \to s]t : T$

? Case T-If
- ? **t = if** t_1 **the** t_2 **else** t_3
  Γ **x : S** $\vdash t_1$ **: Bool**
  Γ **x : S** $\vdash t_2$ **: T**
  Γ **x : S** $\vdash t_3$ **: T**
- By induction we have:
  $\Gamma \vdash [x \to s]t_1$: Bool
  $\Gamma \vdash [x \to s]t_2$: T
  $\Gamma \vdash [x \to s]t_2$: T

9.3.9 - Theorem [Preservation]

- If $\Gamma \vdash e : T$ and $e \to e'$, then $\Gamma e' : T$.

A correspondência Curry-Howard

- Ligação entre a teoria dos tipos e a lógica.
- Série de resultados na fronteira entre a lógica matemática e a teoria da computabilidade de forma a estabelecer uma relação entre a demonstração formal de um sistema lógico e um modelo computacional.

A correspondência Curry-Howard

O símbolo lógico "$\rightarrow$"vem com regras de dois gêneros:

1. Uma regra de introdução (T-Abs) que descreve como elementos do tipo podem ser criados.
2. Uma regra de eliminação (T-App), que descreve como os elementos do tipo podem ser usados.

$$\frac{\Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1.t_2 \; : \; T_1 \rightarrow T_2}$$

$$\frac{\Gamma \vdash t_1 \; : \; T_{11} \rightarrow T_{12} \quad \Gamma \vdash t_2 \; : \; T_{11}}{\Gamma \vdash t_1 \; t_2 \; : T_{12}}$$

A correspondência Curry-Howard

- Em Lógicas construtivas a prova de uma proposição P consiste em evidencias concretas para P.
- Curry e Howard notaram que tais evidencias possuem uma relação forte com a computação.
- Ex.: A prova de uma proposição $P \supset Q$ pode ser vista como um procedimento mecânico que, dada uma prova de P, pode-se construir uma prova de Q. Da mesma forma uma prova $P \wedge Q$ consiste em uma prova de P em conjunto com uma prova de Q.

A correspondência Curry-Howard

Esta observação dá origem a seguinte correspondência:

Lógica	Linguagens de Programação
Proposições	Tipos
Tipos	Tipo P → Q
Proposição P ⊃ Q	Tipo P × Q
Proposição P ∧ Q	Termo t do tipo P

Figura : Tabela

A correspondência Curry-Howard

- A correspondência Curry-Howard não é limitada para um sistema de um tipo particular e uma logica específica, pelo contrario pode ser estendido a uma enorme variedade de sistemas de tipos e lógicas.

Correção e Tipagem

- Anotações de tipo não desempenham qualquer papel na avaliação.
- A maioria dos compiladores para linguagens de programação em grande escala, evitam mostrar comentários em tempo de execução: eles são usados durante o typechecking (e durante a geração de código, em compiladores mais sofisticados).
- Tipos não aparecem no cálculo lambda simplesmente tipado na sua forma compilada. Os programas são convertidos para uma forma sem tipos antes de serem avaliadas.

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