Sistema de tipos: linguagem de tipos

Para definir um sistema de tipos, precisamos especificar *quais os tipos possíveis* e também *como associar programas a tipos*.

No caso de L0, há valores booleanos e números naturais.

Definição: o conjunto `Tipos` é definido abaixo

$$\begin{array}{lcl} \mathtt{T}, \mathtt{T}_1, \ldots & \in & \mathtt{Tipos} \\ \mathtt{T} & ::= & \mathtt{bool} \mid \mathtt{nat} \end{array}$$

Nota: ao contrário da linguagem de tipos acima, geralmente o conjunto de tipos é infinito.

Sistema de tipos: julgamentos de tipos

Um **julgamento de tipos** é uma associação de expressões a tipos. Escreve-se

$$\vdash \mathtt{e} : \mathtt{T}$$

e lê-se *"a expressão e é do tipo T"*.

Nota: o símbolo $\vdash$ é no momento somente parte da notação. Terá significado maior na próxima linguagem (L1).

Os julgamento de tipos serão formalmente descritos por uma relação $(\vdash \_ : \_) \subseteq \mathtt{L0} \times \mathtt{Tipos}$.

Nota: uma expressãoe para o qual existe tipo $\mathtt{T}$ tal que $\vdash \mathtt{e} : \mathtt{T}$ é denominada **bem-tipada**. Caso $\mathtt{T}$ não exista, dizemos que $\mathtt{e}$ é **mal-tipada**.

Sistema de tipos: regras I

A relação $(\vdash \_ : \_) \subseteq \texttt{L0} \times \texttt{Tipos}$ é a **menor** relação tal que as seguintes regras valem.

$$\frac{}{\vdash \texttt{true} : \texttt{bool}} \quad \text{(T-True)}$$

$$\frac{}{\vdash \texttt{false} : \texttt{bool}} \quad \text{(T-False)}$$

$$\frac{\vdash \texttt{e}_1 : \texttt{bool} \qquad \vdash \texttt{e}_2 : \texttt{T} \qquad \vdash \texttt{e}_3 : \texttt{T}}{\vdash \texttt{if } \texttt{e}_1 \texttt{ then } \texttt{e}_2 \texttt{ else } \texttt{e}_3 : \texttt{T}} \quad \text{(T-If)}$$

$$\vdots$$

Sistema de tipos: regras II

$$\vdots$$

$$\frac{}{\vdash \texttt{0} : \texttt{nat}} \quad \text{(T-Zero)}$$

$$\frac{\vdash \texttt{e} : \texttt{nat}}{\vdash \texttt{succ e} : \texttt{nat}} \quad \text{(T-Succ)}$$

$$\frac{\vdash \texttt{e} : \texttt{nat}}{\vdash \texttt{iszero e} : \texttt{bool}} \quad \text{(T-IsZero)}$$

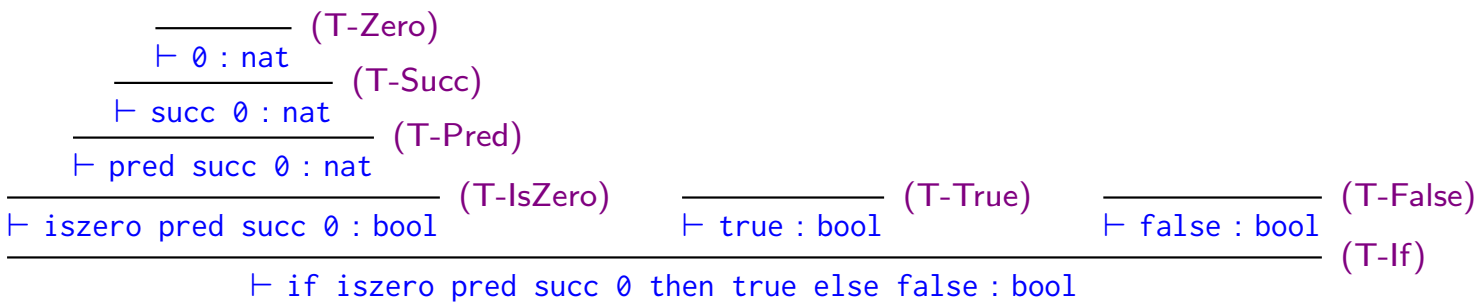
$$\frac{\vdash \texttt{e} : \texttt{nat}}{\vdash \texttt{pred e} : \texttt{nat}} \quad \text{(T-Pred)}$$

Sistema de tipos: exemplo

O programa

```
if iszero (pred (succ 0))
  then true
  else false
```

está associado ao tipo `bool`, conforme justifica a derivação abaixo



E o programa abaixo? ele é bem tipado ou mal-tipado?

```
if  pred (succ 0)
  then true
  else false
```

Conservadorismo dos sistemas de tipos

Sistemas de tipos estáticos são **conservadores** no sentido de não atribuírem tipos a alguns programas que não levam a erro.

Exemplo: ambos programas abaixo são mal-tipados segundo o sistema de tipos de L0

```
(1) if pred succ 0 then true else false
(2) if true then 0 else true
```

Nota: note que o programa (1) leva a erro de avaliação, porém o programa (2) não leva a erro, avaliando corretamente para 0.

A meta é ter o maior número de programas que não levam a erro como bem-tipados, e todos os que levam a erros como mal-tipados.

Problemas computacionais

Os problemas computacionais abaixo são relativos ao sistema de tipos.

1. Verificação de tipo
   Dados uma expressão e e um tipo T, é verdade que $\vdash e : T$?

2. Inferência de tipo
   Dado uma expressão e, encontre (se houver) um tipo T tal que $\vdash e : T$.

3. Habitação de tipo
   Dado um tipo T, encontre (se houver) uma expressão e tal que $\vdash e : T$.

Sistema de Tipos para L0

$$\frac{}{\vdash \texttt{true} : \texttt{bool}}\ (\text{T-True}) \qquad \frac{}{\vdash \texttt{false} : \texttt{bool}}\ (\text{T-False}) \qquad \frac{}{\vdash \texttt{0} : \texttt{nat}}\ (\text{T-Zero})$$

$$\frac{\vdash \texttt{e} : \texttt{nat}}{\vdash \texttt{succ e} : \texttt{nat}}\ (\text{T-Succ}) \qquad \frac{\vdash \texttt{e} : \texttt{nat}}{\vdash \texttt{iszero e} : \texttt{bool}}\ (\text{T-IsZero}) \qquad \frac{\vdash \texttt{e} : \texttt{nat}}{\vdash \texttt{pred e} : \texttt{nat}}\ (\text{T-Pred})$$

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

$$\frac{}{\mathsf{typeInfer}(\texttt{true}) = \texttt{bool}} \qquad \frac{}{\mathsf{typeInfer}(\texttt{false}) = \texttt{bool}} \qquad \frac{}{\mathsf{typeInfer}(0) = \texttt{nat}}$$

$$\frac{\mathsf{typeInfer}(\texttt{e}) = \texttt{nat}}{\mathsf{typeInfer}(\texttt{succ}(\texttt{e})) = \texttt{nat}} \qquad \frac{\mathsf{typeInfer}(\texttt{e}) = \texttt{nat}}{\mathsf{typeInfer}(\texttt{iszero}(\texttt{e})) = \texttt{bool}} \qquad \frac{\mathsf{typeInfer}(\texttt{e}) = \texttt{nat}}{\mathsf{typeInfer}(\texttt{pred}(\texttt{e})) = \texttt{nat}}$$

$$\frac{\mathsf{typeInfer}(\texttt{e}_1) = \texttt{bool} \qquad \mathsf{typeInfer}(\texttt{e}_2) = \texttt{T} \qquad \mathsf{typeInfer}(\texttt{e}_3) = \texttt{T}}{\mathsf{typeInfer}(\texttt{if}(\texttt{e}_1, \texttt{e}_2, \texttt{e}_3)) = \texttt{T}}$$

Algoritmo de inferência de tipos para L0

```
typeInfer(e) =
  caso e
      true ==> bool
    | false ==> bool
    | 0 ==> nat
    | succ (e1) ==> se typeInfer(e1) = nat entao nat senão erro
    | pred (e1) ==> se typeInfer(e1) = nat entao nat senão erro
    | iszero (e1) ==> se typeInfer(e1) = nat entao bool senão erro
    | if(e1,e2,e3) ==>
        se typeInfer(e1) = bool então
            T2 = typeInfer(e2)
            T3 = typeInfer(e3)
            se T2 = T3 então T2 senão erro
        senão erro
```

Segurança e compleza de algoritmos

É esperado que o algoritmo `typeInfer` esteja de acordo com a relação $(\vdash \_ : \_)$

O algoritmo `typeInfer`

- termina sempre sua execução
- é **seguro** *(sound)*, quando a seguinte propriedade é válida
  se `(typeInfer e) = T` então $\vdash$ `e : T`
- é **completo**, quando a seguinte propriedade é válida
  se $\vdash$ `e : T` então `(typeInfer e) = T`

Pergunta: é possível um algoritmo ser somente seguro ou somente completo? Se possível, apresente exemplos com base em `typeInfer`.

Propriedades de linguagens I

A seguir enunciamos algumas propriedades de linguagens de programação que fazem referência à semântica e/ou ao sistema de tipos.

- todo valor é forma normal
  se $\mathtt{v} \in \mathtt{Values}$ então $\mathtt{FN(v)}$

- toda expressão possui forma normal (normalização)
  se $\mathtt{e} \in \mathtt{L0}$ então $\mathtt{PossuiFN(e)}$

- nenhuma expressão progride para duas expressões distintas (determinismo)
  se $\mathtt{e} \longrightarrow \mathtt{e}_1$ e $\mathtt{e} \longrightarrow \mathtt{e}_2$ então $\mathtt{e}_1 = \mathtt{e}_2$

Propriedades de linguagens II

- a semântica *small-step* e a semântica *big-step* são compatíveis
  $e \longrightarrow^* v$ se, e somente se, $e \Downarrow v$

- toda expresão está associada a, no máximo, um tipo
  (unicidade de tipos)
  se $\vdash e : T_1$ e $\vdash e : T_2$, então $T_1 = T_2$

- tipos são preservados pela avaliação *small-step* (preservação)
  se $\vdash e : T$ e $e \longrightarrow e'$, então $\vdash e' : T$

- expressões bem-tipadas ou são valores, ou progridem (progresso)
  se $\vdash e : T$, então `Value(e)` ou existe e' tal que $e \longrightarrow e'$

Propriedades de linguagens III

- expressões bem-tipados não levam a erros de avaliação (segurança do sistema de tipos)
  se $\vdash$ `e` : `T` então $\neg$`Leva-a-Erro(t)`.

Como os conjuntos `L0`, `Values` e `Tipes` e as relações $\longrightarrow$, $\Downarrow$ e $\vdash$ `_` : `_` são objetos matemáticos definidos de forma **indutiva**, podemos utilizar **indução matemática** para provar as propriedades válidas para uma linguagem.

Se a propriedade não for válida, é comum o processo de prova por indução nos levar até um **contraexemplo**.