# INF05516 - Semântica formal N
# Ciência da Computação - UFRGS
# 2006-2

Marcus Ritt
mrpritt@inf.ufrgs.br

11/09/2006

## Agenda

Última aula:

- Linguagens realísticas: mais extensões de IMP.

Hoje:

- Linguagens realísticas: sistemas de típos, varíaveis booleanas.



## Introdução

- IMP não tem tipos: A gramática diferencia entre expressões aritméticas e booleanas
- IMP não tem variáveis com valores booleanas.
- Permitindo variáveis booleanas sem cautela, algumas expressões não fazem mais sentido: $2 + b$ com $b$ uma variável com valor booleana.
- Solução simples: Abusando a gramática, usamos categorias sintáticas diferentes para identificadores de expressões aritméticas e booleanas.
- Ná prática preferimos a liberdade de escolher o nome e declarar o tipo da variável.

## Sistemas de tipos

- Um sistema de tipos é uma método formal para provar que um programa não tem um (certo) comportamento errado.
- Idéia: Associar tipos com as expressões da linguagem e definir algoritmos (regras) de checagem.
- Objetivos:

  **Deteção de erros** Programas com (algumas) construções erradas são proibidos.
  **Abstração** O tipo de um interface e separado da implementação.
  **Documentação** A anotação com tipos serve para o entendimento.
  **Segurança** Protege a linguagem contra execuções erradas.
  **Eficiênca** Um sistema de tipos ajuda a compilação.



## Características

- Tipos estáticos ou tipos dinámicos (p.ex. Scheme).
- Tipos estáticos podem ser explicitos (p.ex. Java) ou implicitos (inferência de tipos, p.ex. OCaml).
- Tipos fracos ou fortes.

## Expressões

- Objetivo: Identificadores para números inteiros e valores booleanas.
- As expressões tem a mesma categoria sintática $\mathrm{Exp}$ (com meta-variável $\mathbf{e} \in \mathrm{Exp}$):

$$\circ_A = \{+, -, *, /\}$$
$$\circ_B = \{\wedge, \vee\}$$
$$\mathrm{cmp}_B = \{=, <, >, \geq, \leq\}$$
$$\mathbf{e} ::= \mathbf{n} | \mathbf{x} | \mathbf{e_1} \circ_\mathbf{A} \mathbf{e_2} | \mathbf{t} | \mathbf{e_1} \circ_\mathbf{B} \mathbf{e_2} | \neg \mathbf{e} | \mathbf{e_1} \mathrm{cmp}_\mathbf{B} \mathbf{e_2}$$

- A avaliação de expressões resulta em *valores* que podem ser inteiros ou booleanas.
- Portanto, definimos um conjunto com meta-variável

$$v \in V = Z \cup B$$

- O estado tem que ser modificado para

$$\sigma \in \Sigma = [\mathrm{Ident} \rightharpoonup V].$$



## Comandos

- Precisamos uma maneira de declarar o tipo de um local.
- Vamos usar os escopos da última aula:

$$c ::= \cdots | \texttt{let var } \mathbf{x} \texttt{:=} \mathbf{e} \texttt{ in } \mathbf{c} | \cdots$$

- Nessa forma, IMP é uma linguagem *implicitamente tipada*, porque a declaração do tipo de $\mathbf{x}$ é implicito. O sistema de tipos *infere* o tipo.
- Em linguagens *explicitamente tipadas* os tipos fazem parte do programa.

## Ambiente de tipos

- Para a inferência de tipos, um *ambiente de tipos* recorda os tipos conhecidos.
- Com um conjunto de tipos possíveis

$$\tau \in T = \{\mathrm{int}, \mathrm{bool}, \mathrm{void}\}$$

  o ambiente associa um tipo com os identificadores declarados

$$\gamma \in \Gamma = [\mathrm{Ident} \rightharpoonup T]$$



## Julgamento de tipo

- O ambiente de tipos (atual) é necessário no *julgamento de tipo* das expressões ou comandos

$$\gamma \vdash \mathbf{e} : \tau \qquad \gamma \vdash \mathbf{c} : \tau.$$

  (Lê: no ambiente de tipos $\gamma$ a expressão $\mathbf{e}$ tem o tipo $\tau$.)
- Uma frase $f$ é *bem tipada* se existe um $\tau$ tal que $\gamma \vdash f : \tau$.
- No tempo da compilação, o sistema de tipos vai julgar se o programa é tipada corretamente, usando *regras de tipo*.
- O julgamento $\vdash$ e uma relação, que depende da categoria sintática
  - $\vdash_E \subseteq \Gamma \times \mathrm{Exp} \times T$
  - $\vdash_C \subseteq \Gamma \times \mathrm{Com} \times T$

## Regras de tipo: Expressões

$$\frac{}{\gamma \vdash_E \mathbf{n} : \text{int}}\ \text{t-num}$$

$$\frac{}{\gamma \vdash_E \mathbf{t} : \text{bool}}\ \text{t-bool}$$

$$\frac{\gamma \vdash_E \mathbf{e_1} : \text{int} \quad \gamma \vdash_E \mathbf{e_2} : \text{int}}{\gamma \vdash_E \mathbf{e_1} \circ_{\mathbf{A}} \mathbf{e_2} : \text{int}}\ \text{t-aop}$$

$$\frac{\gamma \vdash_E \mathbf{e_1} : \text{int} \quad \gamma \vdash_E \mathbf{e_2} : \text{int}}{\gamma \vdash_E \mathbf{e_1}\ \text{cmp}_{\mathbf{B}}\ \mathbf{e_2} : \text{bool}}\ \text{t-acmp}$$



## Regras de tipo: Expressões...

$$\frac{\gamma \vdash_E \mathbf{e} : \text{bool}}{\gamma \vdash_E \neg \mathbf{e} : \text{bool}}\ \text{t-bn}$$

$$\frac{\gamma \vdash_E \mathbf{e_1} : \text{bool} \quad \gamma \vdash_E \mathbf{e_2} : \text{bool}}{\gamma \vdash_E \mathbf{e_1} \circ_{\mathbf{B}} \mathbf{e_2} : \text{bool}}\ \text{t-bop}$$

A única regra que usa ambiente de tipos:

$$\frac{\gamma(\mathbf{x}) = \tau}{\gamma \vdash_E \mathbf{x} : \tau}\ \text{t-ident}$$

## Exemplos

Seja $\gamma$ tal que $\gamma(x) = \text{int}$.

$$\gamma \vdash_E (x = 3) \vee \text{false} : \text{bool}?$$

$$\dfrac{\dfrac{\dfrac{}{\gamma \vdash_E x : \text{int}}\,\text{t-loc} \quad \dfrac{}{\gamma \vdash_E 3 : \text{int}}\,\text{t-int}}{\gamma \vdash_E (x = 3) : \text{bool}}\,\text{t-ae} = \quad \dfrac{}{\gamma \vdash_E \text{false} : \text{bool}}\,\text{t-bool}}{\gamma \vdash_E (x = 3) \vee \text{false} : \text{bool}}\,\text{t-bop} \vee$$

$$\gamma \vdash_E x + \text{true} : \tau \text{ para qualquer } \tau?$$

$$\dfrac{\dfrac{}{\gamma \vdash_E x : \text{int}}\,\text{t-loc} \quad \dfrac{}{\gamma \vdash_E \text{true} : \text{bool}}\,\text{t-bool}}{\gamma \vdash_E x + \text{true} : \tau}\,\text{t-aop} +$$



## Comandos

- Os comandos simplesmente tem o tipo void?
- Porque eles tem tipos?

$$\dfrac{}{\gamma \vdash_C \texttt{skip} : \text{void}}\,\text{t-skip}$$

$$\dfrac{\gamma \vdash_C \mathbf{c_1} : \text{void} \quad \gamma \vdash_C \mathbf{c_2} : \text{void}}{\gamma \vdash_C \mathbf{c_1}\,\texttt{;}\,\mathbf{c_2} : \text{void}}\,\text{t-seq}$$

$$\dfrac{\gamma \vdash_E \mathbf{e} : \text{bool} \quad \gamma \vdash_C \mathbf{c_1} : \text{void} \quad \gamma \vdash_C \mathbf{c_2} : \text{void}}{\gamma \vdash_C \texttt{if}\ \mathbf{e}\ \texttt{then}\ \mathbf{c_1}\ \texttt{else}\ \mathbf{c_2} : \text{void}}\,\text{t-if}$$

$$\dfrac{\gamma \vdash_E \mathbf{e} : \text{bool} \quad \gamma \vdash_C \mathbf{c} : \text{void}}{\gamma \vdash_C \texttt{while}\ \mathbf{e}\ \texttt{do}\ \mathbf{c} : \text{void}}\,\text{t-while}$$

$$\dfrac{\gamma \vdash_C \mathbf{x} : \tau \quad \gamma \vdash_E \mathbf{e} : \tau}{\gamma \vdash_C \mathbf{x}\texttt{:=}\mathbf{e} : \text{void}}\,\text{t-assign}$$

## Comandos: Escopos

A regra interessante é `let` que extende o ambiente de tipos ao encontrar uma declaração:

$$\frac{\gamma \vdash_E \mathbf{e} : \tau \qquad \gamma[x \mapsto \tau] \vdash_C \mathbf{c} : \text{void}}{\gamma \vdash_C \texttt{let var } \mathbf{x} \texttt{:=} \mathbf{e} \texttt{ in } \mathbf{c} : \text{void}} \text{ t-let}$$

Observações:

- O programador pode usar identificadores só dentro um escopo: No nível global, identificadores não tem tipos (com um ambiente de tipos inicialmente vazio).
- A expressão que define o tipo de um identificador, tem que ser bem tipada antes da definição. `let var x:=x in skip` não é bem tipado.



## Exemplo

```
begin
  var x:=2;
  var y:=2;

  begin
    var y:=true;
    x:=x+y
  end
end
```

ou

```
let var x:=2 in
  let var y:=2 in
    let var y:=true in
      x:=x+y
```

## Exemplo

Com $\gamma = \{x \mapsto \text{int}, y \mapsto \text{int}\}$, $\gamma_1 = \{x \mapsto \text{int}, y \mapsto \text{bool}\}$ e $\gamma_2 = \{x \mapsto \text{int}\}$:

$$\dfrac{\dfrac{}{\emptyset \vdash_E \texttt{2} : \text{int}}\,\text{t-num} \quad \dfrac{\dfrac{}{\gamma_2 \vdash_E \texttt{2} : \text{int}}\,\text{t-num} \quad \dfrac{\dfrac{}{\gamma \vdash_E \texttt{true} : \text{bool}}\,\text{t-bool} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\gamma_1(x) = \text{int}}\,\text{t-ident} \quad \dfrac{}{?}}{\gamma_1 \vdash x : \text{int} \qquad \gamma_1 \vdash y : \text{int}}\,\text{t-aop} \quad \dfrac{}{\gamma_1(x) = \text{int}}\,\text{t-ident}}{\gamma_1 \vdash_E \texttt{x+y} : \text{int} \qquad \gamma_1 \vdash_E \texttt{x} : \text{int}}\,\text{t-assign}}{\gamma_1 \vdash_C \texttt{x:=x+y} : void}\,\text{t-let}}{\gamma \vdash_C \texttt{let var y:=true in x:=x+y} : void}\,\text{t-let}}{\gamma_2 \vdash_C \texttt{let var y:=2 in let var y:=true in x:=x+y} : void}\,\text{t-let}}{\emptyset \vdash_C \texttt{let var x:=2 in let var y:=2 in let var y:=true in x:=x+y} : void}$$



## Semântica operacional

- O que muda na semântica operacional?
- `skip` e a sequência não tem modificações.
- O condicional e o laço while usavam expressões booleanas: Eles agora usam expressões.
- A atribuição e a declaração também trabalham com qualquer expressão:

$$\dfrac{\mathbf{e}, \sigma \Downarrow v}{\mathbf{x}\texttt{:=}\mathbf{e}, \sigma \Downarrow \sigma[x \mapsto v]}\,\text{assign}$$

$$\dfrac{\mathbf{e}, \sigma \Downarrow v \qquad \mathbf{c}, \sigma[x \mapsto v] \Downarrow \sigma'}{\texttt{let var x:=}\mathbf{e}\texttt{ in }\mathbf{c}, \sigma \Downarrow \sigma'[x \mapsto \sigma(x)]}\,\text{let}$$

### Unicidade de tipo

- Uma característica simples é importante é a *unicidade de tipo*.
- Para alguma frase $f$

$$\gamma \vdash f : \tau \wedge \gamma \vdash f : \tau' \rightarrow \tau = \tau'$$

  (lê: cada frase tem um único tipo).
- A unicidade de tipo pode ser provado usando indução estrutural.
- Ele é uma característica particular de nossa sistema.
- Em geral, um o tipo de uma frase não tem que ser único.



### Segurança

- Usando o sistema de tipos: o que ganhamos?
- Ótimo seria uma combinação de
  - progresso: uma expressão bem tipada sempre tem um valor
  - e preservação: o valor de uma expressão bem tipada é bem tipada.
- (Essas características justificam o moto "Segurança = progresso + preservação".)

## Segurança

- Um estado conforme com o ambiente de tipos

$$\gamma \vdash \sigma \overset{\text{def}}{\leftrightarrow} \mathrm{dom}(\sigma) = \mathrm{dom}(\gamma) \wedge \forall l \in \mathrm{dom}(\sigma) : \gamma \vdash \sigma(l) : \gamma(l)$$

- Então, se $\gamma \vdash \sigma$, queremos
  - Progresso

$$(\gamma \vdash \mathbf{e} : \tau) \rightarrow (\exists v \in V : \mathbf{e}, \sigma \Downarrow v)$$

    (uma expressão bem tipada tem um valor)
  - Preservação

$$(\gamma \vdash \mathbf{e} : \tau \wedge \mathbf{e}, \sigma \Downarrow v) \rightarrow \gamma \vdash v : \tau$$

    (o valor de uma expressão bem tipada é bem tipada)
- E os comandos?