

DIM0436

12. Tipagem

Richard Bonichon

20140902

Sumário

- 1 Introdução
- 2 While com tipos
- 3 Verificação de tipos
- 4 Inferência de tipos
- 5 Subtipagem

- 1 Introdução
- 2 While com tipos
- 3 Verificação de tipos
- 4 Inferência de tipos
- 5 Subtipagem

O que é tipagem ?

Definição simples

Tipar é o processo que consiste na atribuição de **tipos** aos dados que o programa usa.

Teoria dos tipos

- O problema fundamental da teoria dos tipos é garantir que os programas têm um significado.
- O problema fundamental **causado** por uma teoria dos tipos é que programas com significado pode não ter uma atribuição de significado
- A pesquisa na teoria dos tipos resulta dessa tensão.

Linguagens de programação

Linguagens de programação são divididos entre políticas de tipagem fracas ou fortes e estáticas ou dinâmica.

Tipagem estático

- Mais eficiente
- Ferramentas melhores
- Menos testes necessários
- Melhor documentação
- Segurança adicional na manutenção

Tipagem e semântica

- Semânticas operacionais são também conhecidas como semânticas **dinâmicas**.
- Tipagem é também conhecida como semântica **estática**

Tipagem dinâmico

- Linguagem mais simples
- Menos erros de compiladores (às vezes pouca legíveis)
- Menos verboso
- Mais simples para fases de exploração
- Nenhum limites de expressividade impostos (até execução ...)

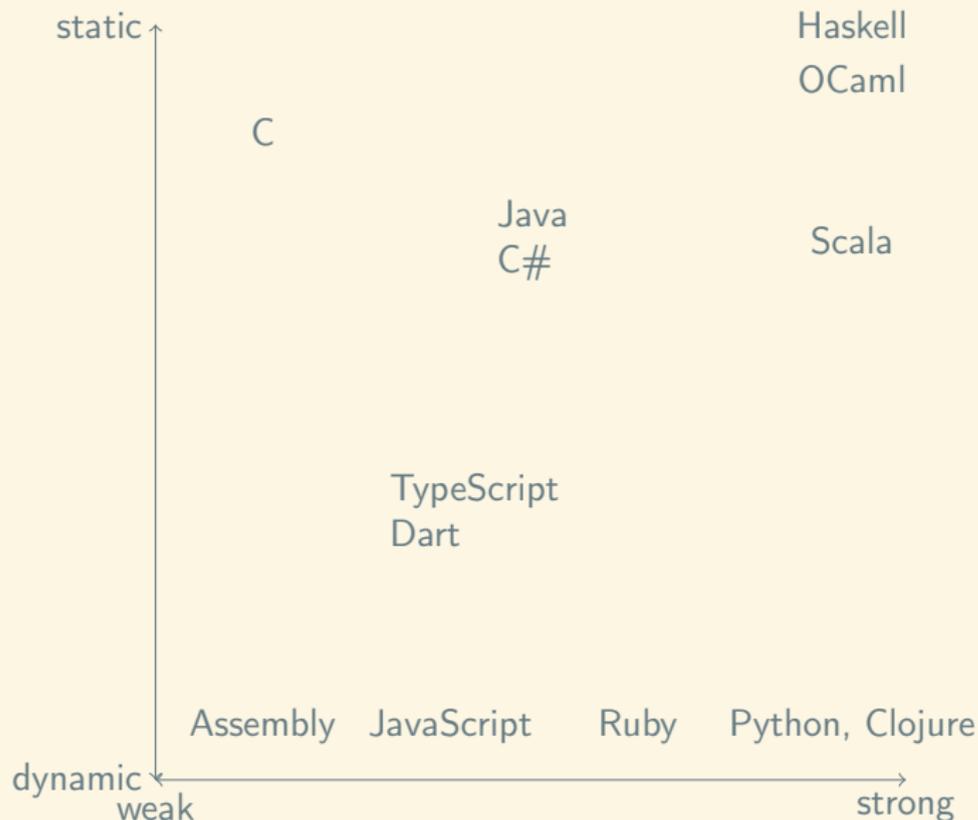
O caso contra tipos

- A queixa do programador Pascal *Não posso escrever o código que eu quero nesse sistema de tipos.*
- A do programador de linguagens de script (sem tipos) *É cansativo ficar com os tipos corretos durante o desenvolvimento*
- A do programador Java/C/C++ ... *Anotações de tipos são verbosas*
- A do programador que tem parcialmente razão *Erros de tipos são incompreensíveis*
- A queixa das queixas *Eu verdadeiramente não posso escrever o código que eu quero.*

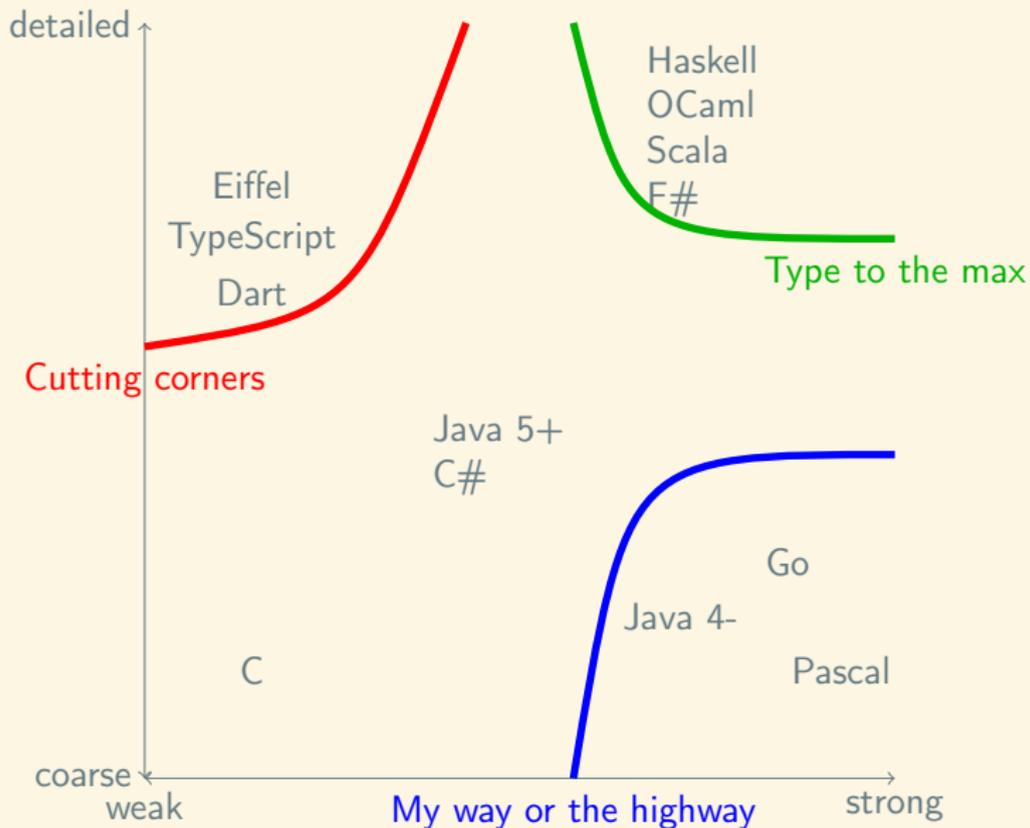
Alan Kay

I'm not against types, but I don't know of any type systems that aren't a complete pain, so I still like dynamic typing.

A paisagem dos sistemas de tipos (Odersky)



Sistemas de tipos estáticos



My way or the highway

Características

- Sistema de tipos simples
- Sem *generics*
- Não muito extensível

Resumo

- + Ferramentas simples
- Muito normativo

Tipagem máxima

Características

- Linguagem rica para descrever tipos
- Formas de combinações de tipos, incluindo *generics*
- Sistemas de tipos inspirados da lógica

Vs. tipagem dinâmica

Expressividade Sistemas de tipos ricos e sistemas de saída com verificação (*casts*).

Verbosidade Inferência de tipos

Facilidade explorativa Scala tem um tipo `Nothing` (???)

Cutting corners

- Uso da intuição do usuário (*generics* covariantes)
 - ▶ Funcionários são pessoas
 - ▶ Então funções de funcionários para empresas são também funções de pessoas para empresas, certo ?
- Não tem medo de incorreção
- Fácil e superficialmente simples
- Mas, fica um [hack](#)
- Pode-se construir desenhos extraordinários usando teorias falsas ?

- 1 Introdução
- 2 While com tipos
- 3 Verificação de tipos
- 4 Inferência de tipos
- 5 Subtipagem

While com rotinas

Sintaxe

$$\begin{aligned} a &::= n \mid x \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2 \\ b &::= \text{true} \mid \text{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\ S &::= x := a \mid \text{skip} \mid S_1; S_2 \\ &\quad \mid \text{if } b \text{ then } S_1 \text{ else } S_2 \\ &\quad \mid \text{while } b \text{ do } S \\ &\quad \mid \text{begin } D_V \ D_P \ S \ \text{end} \\ &\quad \mid \text{call } p \\ D_V &::= \text{var } x := a; D_V \mid \varepsilon \\ D_P &::= \text{proc } p \text{ is } S; D_P \mid \varepsilon \end{aligned}$$

$While_{\mathcal{T}}^+$: com anotações de tipos

Sintaxe

$$\begin{aligned} S & ::= \text{begin } D_V D_P E \text{ end} \\ E & ::= x := E \mid \text{skip} \mid E_1; E_2 \\ & \quad \mid \text{if } E_1 \text{ then } E_2 \text{ else } E_3 \\ & \quad \mid \text{while } E_1 \text{ do } E_2 \\ & \quad \mid \text{begin } D_V D_P E \text{ end} \\ & \quad \mid \text{call } p(E^*) \\ & \quad \mid \mathcal{C} \mid \mathcal{V} \mid E_1 + E_2 \mid E_1 * E_2 \mid E_1 - E_2 \\ & \quad \mid \text{true} \mid \text{false} \mid E_1 = E_2 \mid E_1 \leq E_2 \mid \neg E \mid E_1 \wedge E_2 \\ D_V & ::= x : \mathcal{T}; D_V \mid \varepsilon \\ D_P & ::= \text{proc } p(x_1 : \mathcal{T}_1, \dots, x_n : \mathcal{T}_n) : \mathcal{T} \text{ is } E; D_P \mid \varepsilon \end{aligned}$$

Exercício

Assunto

Definir a semântica natural de $While_{\mathcal{T}}^+$

Tipos simples

Definição

Seja U um alfabeto infinito cujos elementos são variáveis de tipos
Os tipos simples são o conjunto definido por

$$\begin{aligned} \mathcal{T} ::= & U \\ & | \mathcal{T} \times \mathcal{T} \\ & | \mathcal{T} \rightarrow \mathcal{T} \\ & | C_{\mathcal{T}} \end{aligned}$$

Tipos básicos

$$C_{\mathcal{T}} = \{int, string, real, bool, unit\} \subset \mathcal{T}(While_{\mathcal{T}})$$

Forma das regras

- Um contexto Γ , i.e. um mapeamento $\text{Symb} \rightarrow \mathcal{T}$, onde Symb pode ser um identificador ou um símbolo de função predefinido.

Forma geral de uma regra

$$\frac{\Gamma \vdash E_1 : T_1 \quad \Gamma \vdash E_2 : T_2}{\Gamma \vdash E_1; E_2 : T_2}$$

Tipagem das expressões aritméticas

Primitivas

- As operações $\{+, *, -\}$ são vistas como primitivas, i.e. funções cujo tipo é predeterminado.
- Assim, todo contexto de tipagem Γ implicitamente contém as definições dessas funções.
- $\mathcal{T}(+), \mathcal{T}(-), \mathcal{T}(*) \in \{\text{int} \times \text{int} \rightarrow \text{int}, \text{real} \times \text{real} \rightarrow \text{real}\}$
- Os constantes da linguagem fazem também parte do contexto implícito.

Exemplo de regra

$$\frac{\frac{+ \in \text{dom}(\Gamma)}{\Gamma \vdash + : \text{real} \times \text{real} \rightarrow \text{real}} \quad \frac{1.3 \in \text{dom}(\Gamma)}{\Gamma \vdash 1.3 : \text{real}} \quad \frac{1.2 \in \text{dom}(\Gamma)}{\Gamma \vdash 1.2 : \text{real}}}{\Gamma \vdash 1.3 + 1.2 : \text{real}}$$

- 1 Introdução
- 2 While com tipos
- 3 Verificação de tipos
- 4 Inferência de tipos
- 5 Subtipagem

Definição

O objetivo da **verificação de tipos** é assinalar os erros de tipos a partir das anotações iniciais dadas pelo usuário.

Linguagens

- Java
- C/C++
- Pascal

Regras

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : T} \text{ Var}$$

$$\frac{}{\Gamma \vdash \text{skip} : \text{unit}} \text{ Skip}$$

$$\frac{\Gamma \vdash E_1 : T_1 \quad \Gamma \vdash E_2 : T_2}{\Gamma \vdash E_1; E_2 : T_2} \text{ Seq}$$

$$\frac{\Gamma \vdash b : \text{bool} \quad \Gamma \vdash E : \text{unit}}{\Gamma \vdash \text{while } b \text{ do } E : \text{unit}} \text{ While}$$

$$\frac{\Gamma \vdash b : \text{bool} \quad \Gamma \vdash E_1 : T \quad \Gamma \vdash E_2 : T}{\Gamma \vdash \text{if } b \text{ then } E_1 \text{ else } E_2 : T} \text{ if}$$

$$\frac{\Gamma \vdash x : T \quad \Gamma \vdash e : T}{\Gamma \vdash x := e : \text{unit}} \text{ Assigns}$$

Contextos

Declarações de variáveis

$$\begin{aligned} |x : T; D_V|(\Gamma) &= |D_V|((\Gamma \setminus \{x\}) \cup \{x : T\}) \\ |\varepsilon|(\Gamma) &= \Gamma \end{aligned}$$

Rotinas

$$\begin{aligned} |\text{proc } p(x_1 : T_1, \dots, x_n : T_n) : T \text{ is } S \\ : T_1 \times \dots \times T_n \rightarrow T; D_P|(\Gamma) &= |D_P|(\{p : T_1 \times \dots \times T_n \rightarrow T\} \cup (\Gamma \setminus \{p\})) \\ &\text{se } \Gamma \vdash p : T_1 \times \dots \times T_n \rightarrow T \\ |\varepsilon|(\Gamma) &= \Gamma \end{aligned}$$

Notação (abusiva)

$$(\Gamma \setminus \{y\})(x) = \begin{cases} \Gamma(x) & \text{se } x \neq y \\ \emptyset & \text{senão} \end{cases}$$

Funções e blocos

$$\frac{D_P(D_V(\Gamma)) \vdash E : T}{\Gamma \vdash \text{begin } D_V \ D_P \ E \ \text{end} : T} \text{Block}$$

$$\frac{\Gamma \vdash p : T_1 \times \dots \times T_n \rightarrow T \quad \Gamma \vdash x_i : T_i \quad i \in [1, n]}{\Gamma \vdash \text{call } p(x_1, \dots, x_n) : T} \text{Call}$$

$$\frac{\Gamma \setminus \{x_i \mid i \in 1..n\} \cup \{x_1 : T_1, \dots, x_n : T_n\} \vdash E : T}{\Gamma \vdash \text{proc } p(x_1 : T_1, \dots, x_n : T_n) : T \ \text{is } E : T_1 \times \dots \times T_n \rightarrow T} \text{Proc}$$

Exercícios

Determine o tipo do programa abaixo

```
begin
  x: int;
  proc p(y:int) : bool is x = y;
  x := 0;
  i := 15;
  while i < 17 do
    begin
      p(i);
      i := i + 1;
    end;
  end
end
```

- Podemos tipar
proc fact(x:int) is if x = 0 then 1 else fact(x - 1) * x; ?
- Proponha uma modificação à regra Proc para poder tipar fact.

- 1 Introdução
- 2 While com tipos
- 3 Verificação de tipos
- 4 Inferência de tipos
- 5 Subtipagem

Definição

O objetivo da **inferência de tipos** é reconstruir/deduzir os tipos e assinalar as incoerências vistas nessa reconstrução: elas são os erros de tipos.

Linguagens

- OCaml, Haskell
- Scala
- C++ com auto

Syntaxe

$$\begin{aligned} S &::= E \\ E &::= x := E \mid \text{skip} \mid E_1; E_2 \\ &\mid \text{if } E_1 \text{ then } E_2 \text{ else } E_3 \\ &\mid \text{while } E_1 \text{ do } E_2 \\ &\mid \text{begin } E \text{ end} \\ &\mid \text{call } p(E^*) \\ &\mid \text{proc } p(x_1 : \mathcal{T}_1, \dots, x_n : \mathcal{T}_n) \text{ is } S \\ &\mid \mathcal{C} \mid \mathcal{V} \mid E_1 + E_2 \mid E_1 * E_2 \mid E_1 - E_2 \\ &\mid \text{true} \mid \text{false} \mid E_1 = E_2 \mid E_1 \leq E_2 \mid \neg E \mid E_1 \wedge E_2 \end{aligned}$$

Tipos das variáveis

- Variáveis não tem mais declarações de tipos
- Atribuições e usos permitem deduzir restrições de tipos
- Os tipos dos parâmetros de função são ainda especificados

Discussão

Tipos diferentes em ramos diferentes

- *random : unit → int*

```
x := random()
if x < random() then
  x := "foo";
else x := 1;
print (x)
```

Escopo e variáveis locais

```
x := random()
if x < random() then
  j := 3;
else j := 1;
print (j)
```

Versão sem problemas de escopo

```
x := random()
j := if x < random() then 3
      else 1;
print (j)
```

Regras

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : (\Gamma, T)} \text{Var}$$

$$\frac{}{\Gamma \vdash \text{skip} : (\Gamma, \text{unit})} \text{Skip}$$

$$\frac{\Gamma \vdash E_1 : \Gamma_1, T_1 \quad \Gamma_1 \vdash E_2 : \Gamma_2, T_2}{\Gamma \vdash E_1; E_2 : \Gamma_2, T_2} \text{Seq}$$

$$\frac{\Gamma \vdash E_1 : (\Gamma', \text{bool}) \quad \Gamma \vdash E_2 : (\Gamma_2, T)}{\Gamma \vdash \text{while } E_1 \text{ do } E_2 : (\Gamma, T)} \text{While}$$

$$\frac{\Gamma \vdash E_1 : (\Gamma_1, \text{bool}) \quad \Gamma \vdash E_2 : (\Gamma_2, T) \quad \Gamma \vdash E_3 : (\Gamma_3, T)}{\Gamma \vdash \text{if } E_1 \text{ then } E_2 \text{ else } E_3 : (\Gamma, T)} \text{If}$$

$$\frac{\Gamma \vdash e : (\Gamma, T)}{\Gamma \vdash x := e : ((\Gamma \setminus \{x\}) \cup \{x : T\}), \text{unit}} \text{Assigns}$$

Funções e blocos

$$\frac{D_P(\Gamma) \vdash E : (\Gamma', T)}{\Gamma \vdash \text{begin } D_P E \text{ end} : (\Gamma, T)} \text{Block}$$

$$\frac{\Gamma \vdash p : (\Gamma', T_1 \times \dots \times T_n) \rightarrow T \quad \Gamma \vdash x_i : (\Gamma_i, T_i) \quad i \in [1, n]}{\Gamma \vdash \text{call } p(x_1, \dots, x_n) : (\Gamma, T)} \text{Call}$$

$$\frac{(\Gamma \setminus \{x_i | i \in 1..n\}) \cup \{x_1 : T_1, \dots, x_n : T_n\} \vdash E : (\Gamma', T)}{\Gamma \vdash \text{proc } p(x_1 : T_1, \dots, x_n : T_n) : T \text{ is } E : ((\Gamma \setminus \{p\}) \cup \{p : T_1 \times \dots \times T_n \rightarrow T\}, T)} \text{Pr}$$

Declaração de funções sem tipos

Eliminação dos tipos

Suponha $\text{proc } p(x_1, \dots, x_n) \text{ is } S$.

- Qual é o tipo de x_i ?
- A presença de variáveis numa expressão permite deduzir o tipo esperado do parâmetro.

Regra (idêntica)

$$\frac{\Gamma \setminus \{x_i \mid i \in 1..n\} \cup \{x_1 : T_1, \dots, x_n : T_n\} \vdash E : (\Gamma', T)}$$

$$\frac{}{\Gamma \vdash \text{proc } p(x_1 : T_1, \dots, x_n : T_n) : T \text{ is } E : ((\Gamma \setminus \{p\}) \cup \{p : T_1 \times \dots \times T_n \rightarrow T\}, T)}$$

Método

- Usamos a **mesma** regras mas ...
- Criar novas variáveis de tipos para os x_i
- Criar uma quase árvore de tipagem
- Recolher as restrições dessa quase árvore para transformá-la em árvore correta

Syntaxe

$$\begin{aligned} S &::= E \\ E &::= x := E \mid \text{skip} \mid E_1; E_2 \\ &\mid \text{if } E_1 \text{ then } E_2 \text{ else } E_3 \\ &\mid \text{while } E_1 \text{ do } E_2 \\ &\mid \text{begin } E \text{ end} \\ &\mid \text{call } p(E^*) \\ &\mid \text{proc } p(x_1, \dots, x_n) \text{ is } S \\ &\mid \mathcal{C} \mid \mathcal{V} \mid E_1 + E_2 \mid E_1 * E_2 \mid E_1 - E_2 \\ &\mid \text{true} \mid \text{false} \mid E_1 = E_2 \mid E_1 \leq E_2 \mid \neg E \mid E_1 \wedge E_2 \end{aligned}$$

Resolução das restrições

Notação

- $FV(t)$ = Variáveis livres de t
- $t ::= \text{string}, \dots, \text{unit} \mid U \mid f_{\rightarrow}(t_1, t_2) \mid f_{\times}(t_1, t_2)$

Unificação

$$R \cup \{t \approx t\} \hookrightarrow R$$

$$R \cup \{f(s_0, \dots, s_k) \approx f(t_0, \dots, t_k)\} \hookrightarrow R \cup \{s_0 \approx t_0, \dots, s_k \approx t_k\}$$

$$R \cup \{f(s_0, \dots, s_k) \approx g(t_0, \dots, t_m)\} \hookrightarrow \perp$$

se $f \neq g$ ou $k \neq m$

$$R \cup \{f(s_0, \dots, s_k) \approx x\} \hookrightarrow R \cup \{x \approx f(s_0, \dots, s_k)\}$$

$$R \cup \{x \approx t\} \hookrightarrow R[x := t] \cup \{x \approx t\}$$

se $x \notin FV(t)$ e $x \in FV(R)$.

$$R \cup \{x \approx f(s_0, \dots, s_k)\} \hookrightarrow \perp$$

se $x \in FV(f(s_0, \dots, s_k))$

Exercício

Assunto

Aplicar a resolução de restrições por unificação aos seguintes conjuntos de restrições:

- 1 $\{f_{\rightarrow}(f_{\times}(int, int), bool) \approx f_{\rightarrow}(x, bool), y \approx x, f_{\rightarrow}(y, real) \approx f_{\rightarrow}(int, z)\}$ onde x, y, z são variáveis de tipo.
- 2 $\{x \approx z, f_{\times}(f_{\times}(int, int), bool) \approx f_{\times}(x, bool), y \approx x, f_{\rightarrow}(y, real) \approx f_{\rightarrow}(int, z)\}$ onde x, y, z são variáveis de tipo.

Exercício

Determine o tipo do programa abaixo

```
begin
  x := 0;
  proc p(y) is x = y;
  i := 15;
  while i < 17 do
    begin
      i := if p(i) then i + 1 else i + 2;
    end;
  end
end
```

Exercício

Determine o tipo do programa abaixo

```
begin
  i := 0;
  while i < 10 do
    begin
      if (i = 8) then i := "foo" else skip;
      i := i + 1;
    end;
  end
end
```

- 1 Introdução
- 2 While com tipos
- 3 Verificação de tipos
- 4 Inferência de tipos
- 5 Subtipagem

Coerções

- Numa linguagem com tipagem forte, é geralmente proibido misturar tipos.
- Por exemplo, de acordo com as regras de tipagem

$$\frac{\overline{\Gamma \vdash + : real \times real \rightarrow real} \quad \overline{\Gamma \vdash 1 : int} \quad \overline{\Gamma \vdash 1.2 : real}}{\Gamma \vdash 1 + 1.2 : \circlearrowleft}$$

- Linguagens usam um mecanismo para explicitar conversões de tipos

$$\frac{\overline{\Gamma \vdash + : real \times real \rightarrow real} \quad \overline{\Gamma \vdash 1.2 : real} \quad \frac{\overline{real : int \rightarrow real} \quad \overline{1 : int}}{\Gamma \vdash real(1) : real}}{\Gamma \vdash real(1) + 1.2 : real}$$

Porque subtipagem ?

- 1 Ao lugar de um real, pode-se usar um inteiro ?
- 2 Ao lugar de um registro com campos c_1, c_2 , pode-se usar um outro registro com campos c_1, c_2, c_3 ?

Sintaxe

$$\begin{aligned} E & ::= x := E \mid \text{skip} \mid E_1; E_2 \\ & \mid \text{if } E_1 \text{ then } E_2 \text{ else } E_3 \\ & \mid \text{while } E_1 \text{ do } E_2 \\ & \mid \text{begin } D_P E \text{ end} \\ & \mid \text{call } p(E^*) \\ & \mid \mathcal{C} \mid \mathcal{V} \mid E_1 + E_2 \mid E_1 * E_2 \mid E_1 - E_2 \\ & \mid \text{true} \mid \text{false} \mid E_1 = E_2 \mid E_1 \leq E_2 \mid \neg E \mid E_1 \wedge E_2 \\ & \mid x.l \\ & \mid \{l_1 = E_1, \dots, l_n = E_n\} \\ D_P & ::= \text{proc } p(x_1, \dots, x_n) \text{ is } S; D_P \mid \varepsilon \\ S & ::= \text{begin } D_P E \text{ end} \end{aligned}$$

Tipo de um registro

Definição

- Um registro é um conjunto de campos
- O tipo de um registro é um conjunto de pares $\text{Name} \times \mathcal{T}$

Definição (Tipos)

$$\begin{aligned} \mathcal{T} ::= & U \\ & | \mathcal{T} \times \mathcal{T} \\ & | \mathcal{T} \rightarrow \mathcal{T} \\ & | \{l_i : T_i^{1 \in 1..n}\} \end{aligned}$$

Relação de subtipagem

$$\frac{}{\overline{T} <: \overline{T}} \text{ Sub-refl}$$

$$\frac{}{\overline{T} <: \overline{Top}} \text{ Sub-top}$$

$$\frac{T_1 <: T_2 \quad T_2 <: T_3}{T_1 <: T_3} \text{ Sub-trans}$$

$$\frac{T_1 <: U_1 \quad U_2 <: T_2}{U_1 \rightarrow U_2 <: T_1 \rightarrow T_2} \text{ Sub-trans}$$

$$\{l_i^{j \in 1..n}\} \subseteq \{k_j^{j \in 1..m}\}$$

$$k_j = l_i \Rightarrow U_j <: T_i$$

$$\frac{\{k_j : U_j^{j \in 1..m}\} <: \{l_i : T_i^{i \in 1..n}\}}{\text{Sub-reg}}$$

Subtipagem

Uma regra adicional

$$\frac{\Gamma \vdash t : U \quad U <: T}{\Gamma \vdash t : T}$$

O tipo Top

- Corresponde mais ou menos ao tipo Object em Java

Operadores aritméticos mistos

Tipagem das expressões aritméticas

$$\frac{\frac{}{\Gamma \vdash + : \mathit{real} \times \mathit{real} \rightarrow \mathit{real}} \quad \frac{\frac{\Gamma \vdash 1 : \mathit{int} \quad \mathit{int} <: \mathit{real}}{\Gamma \vdash 1 : \mathit{real}}}{\Gamma \vdash 1 + 1.2 : \mathit{real}} \quad \frac{}{\Gamma \vdash 1.2 : \mathit{real}}}{\Gamma \vdash 1 + 1.2 : \mathit{real}}$$

Exercícios

Determine o tipo do programa abaixo

```
begin
  i := 0;
  proc p(y) is x = y.k;
  i := 15;
  while i < 17 do
    begin
      p({ k = i, foo = i * i});
      i := i + 1;
    end;
  end
end
```

Resumo

- 1 Introdução
- 2 While com tipos
- 3 Verificação de tipos
- 4 Inferência de tipos
- 5 Subtipagem

Referências

-  Luis Damas and Robin Milner, *Principal type-schemes for functional programs*, Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (New York, NY, USA), POPL '82, ACM, 1982, pp. 207–212.
-  Roger Hindley, *The principal type-scheme of an object in combinatory logic*, Trans. Amer. Math. Soc **146** (1969), 29–60.
-  Robin Milner, *A theory of type polymorphism in programming*, J. Comput. Syst. Sci. **17** (1978), no. 3, 348–375.
-  B.C. Pierce, *Types and programming languages*, MIT Press, 2002.

Perguntas ?



<http://dimap.ufrn.br/~richard/dim0436>