

UNIVERSIDADE FEDERAL DE SANTA MARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA - PPGI

Subtipos e Metateoria dos Subtipos

LINGUAGENS DE PROGRAMAÇÃO – ELC921
PROF^A DR^A JULIANA KAISER VIZZOTTO

ALUNOS: Alberto Kummer, Daniel Di Domenico, Fernando
Campagnolo, Jéssica Lasch de Moura e José Puiati

15 Subtyping

- Também chamado de *subtype polymorphism*;
- Característica presente nas linguagens orientadas a objetos;
- Cálculo Lambda simplesmente tipado com subtipos: $\lambda_{<}$:

15.1 Subsumption

- Sem subtipos:

- ✓ Regras de tipos bastante rígidas;
- ✓ Rejeição de expressões que, aos olhos do programador, são bem tipadas.

- Exemplo:

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

$(\lambda r : \{x : \text{Nat}\}. \quad r.x) \quad \{x = 0, y = 1\}$ Inválido?

15.1 Subsumption

- Sem subtipos:

- ✓ Regras de tipos bastante rígidas;
- ✓ Rejeição de expressões que, aos olhos do programador, são bem tipadas.

- Exemplo:

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

$(\lambda r : \{x : \text{Nat}\}. \quad r.x) \quad \{x = 0, y = 1\}$ Válido

15.1 Subsumption

■ Objetivo dos subtipos:

- ✓ Refinamento das regras de tipos;
- ✓ Se S é subtipo de T ($S <: T$), qualquer termo do tipo S pode ser utilizado no contexto onde T é esperado;
- ✓ Princípio da substituição segura (*safe substitution*).

Regra *Subsumption*

$$\frac{\Gamma \vdash t : S \quad S <: T}{\Gamma \vdash t : T} \quad (\text{T-SUB})$$

Adaptado de (Pierce, 2002).

15.1 Subsumption

- Exemplo: $(\lambda r: \{x : \text{Nat}\}. r.x) \quad \{x = 0, y = 1\}$
 - ✓ Considerando que: $\{x : \text{Nat}, y : \text{Nat}\} <: \{x : \text{Nat}\}$
 - ✓ A regra T-SUB permite a aplicação pois são tipos válidos.

15.2 A relação de subtipos

- Coleção de regras de inferência para derivar declarações

$$S <: S \quad (\text{S-REFL})$$

$$\frac{S <: U \quad U <: T}{S <: T} \quad (\text{S-TRANS})$$

$$\{\prod_i : T_i \mid i \in 1..n+k\} <: \{\prod_i : T_i \mid i \in 1..n\} \quad (\text{S-RCDWIDTH})$$

15.2 A relação de subtipos

- Exemplos:

- $\{x : \text{Nat}\}$

- $\{x = 3\}, \{x = 5\}, \text{ e } \{x = 3, a = \text{true}, b = \text{true}\}$

- $\{x : \text{Nat}, y : \text{Nat}\}$

- $\{x = 3, y = 100\} \text{ e } \{x = 3, y = 100, z = \text{true}\}$

15.2 A relação de subtipos

- É seguro permitir que os tipos dos campos variem desde que os tipos correspondentes nos dois registros estejam na relação de subtipos;

$$\frac{\text{for each } i \quad S_i <: T_i}{\{_i : S_i \mid i \in 1..n\} <: \{_i : T_i \mid i \in 1..n\}} \quad (\text{S-RCDDEPTH})$$

15.2 A relação de subtipos

- A ordem dos campos no registro não faz diferença em como podemos usá-los;

$$\frac{\{k_j : S_j^{j \in 1..n}\} \text{ is a permutation of } \{ _i : T_i^{i \in 1..n} \}}{\{k_j : S_j^{j \in 1..n}\} <: \{ _i : T_i^{i \in 1..n} \}} \quad (\text{S-RCDPERM})$$

15.2 A relação de subtipos

- Como estamos trabalhando com uma linguagem que não terá apenas números e registros, mas também funções funções podem ser utilizadas como argumentos, precisamos especificar sobre quais circunstâncias é seguro usar uma função de um determinado tipo em um contexto onde é esperada uma função de um tipo diferente;

$$\frac{T_1 <: S_1 \quad S_2 <: T_2}{S_1 \rightarrow S_2 <: T_1 \rightarrow T_2} \quad (\text{S-ARROW})$$

15.2 A relação de subtipos

- É conveniente ter um tipo que seja um "supertipo" de cada tipo, para isso introduzimos a nova constante de tipo "Top";

$S <: \text{Top}$

$(S\text{-TOP})$

15.4 The Top and Bottom Types

Formas de subtipos - Top e Bot

Formas sintáticas:

$$\begin{array}{l} T ::= \dots \\ \quad \textit{Top} \\ \quad \textit{Bot} \end{array}$$

Regras de subtipos:

$$S <: \textit{Top} \quad (\text{S-Top})$$
$$\textit{Bot} <: T \quad (\text{S-Bot})$$

Adaptado de (Pierce, 2002).

15.4 The Top and Bottom Types

■ Top:

- ✓ Elemento **máximo** da relação de subtipos;
- ✓ Equivale ao tipo *Object* das linguagens orientadas a objetos;
- ✓ Dispositivo técnico sofisticado em sistemas que combinam subtipos com polimorfismo.

■ Bot:

- ✓ Elemento **mínimo** da relação de subtipos;
- ✓ Tipo vazio (não existem valores do tipo Bot);
- ✓ Muito útil para expressar algumas operações que não visam retorno de valores, como exceções, pois:
 - Permite ao programador definir expressões sem retorno com o tipo Bot;
 - Indica ao *typechecker* que a expressão pode ser utilizada com segurança em qualquer contexto.

15.4 The Top and Bottom Types

- Exemplo:

$\lambda x : T.$

if $\langle \text{valor apropriado para } x \rangle$ then

$\langle \text{calcula o resultado} \rangle$

else

 error

15.4 The Top and Bottom Types

- Tipo Bot dificulta a implementação;
- Mudança da regra de tipo da aplicação:
 - ✓ $t_1 \ t_2 :$
 - ✓ t_1 pode ser tanto do tipo seta ($T_1 \rightarrow T_2$) ou do tipo Bot.

15.6 Semântica de coerção para sistema de subtipos

- Subtipos fornecem ao programador uma maior flexibilidade na escrita do código, **o que pode ocasionar alguma sobrecarga em tempo de execução**
 - × seja por consequência da representação de dados em uma máquina real

Int <: Float

- × ou pela natureza das linguagens de programação funcionais

$$\{l_i = v_i \mid i \in 1..n\}.l_j \rightarrow v_j \quad (\text{E-PROJRCD})$$

Semânticas de coerção

- Ideia principal: **substituir** expressões que envolvem subtipos por expressões de mais “baixo nível”
 - ✓ processo que ocorre em tempo de execução
 - ✓ geralmente as expressões são compiladas para uma linguagem próxima a de máquina
- Tudo em razão da performance:
 - ✓ reduzir a necessidade de **buscas** durante a execução de código
- No livro-texto da disciplina
 - ✓ Linguagem utilizada pelo programador: λ_{\leftarrow}
 - ✓ Linguagem de baixo nível: λ_{\rightarrow}

15.6 Semântica de coerção para sistema de subtipos

Semânticas de coerção

- É vista como uma função que **toma um tipo e devolve outro**
- É expressa por $\llbracket _ \rrbracket$

$$\begin{array}{ll}\llbracket \text{Top} \rrbracket & = \text{Unit} \\ \llbracket T_1 \rightarrow T_2 \rrbracket & = \llbracket T_1 \rrbracket \rightarrow \llbracket T_2 \rrbracket \\ \llbracket \{1_i : T_i \mid i \in 1..n\} \rrbracket & = \{1_i : \llbracket T_i \rrbracket \mid i \in 1..n\}\end{array}$$

Semânticas de coerção

- Em tempo de execução as coerções são inseridas nos locais onde ocorrem *subsumptions*
- Pierce (2002) sugere a formalização de coerções como **funções de derivação de tipos**

$$\mathcal{C} :: S \prec: T$$

$$\mathcal{D} :: \Gamma \vdash t:T$$

15.6 Semântica de coerção para sistema de subtipos

Semânticas de coerção

- Mas não basta saber que $S <: T$
 - ✓ não basta inventar uma regra e adotá-la
 - ✓ as regras de subtipagem justificam o emprego das coerções
 - ✓ interpretadas como **árvores de derivação de subtipos**

$$\begin{aligned} \llbracket \frac{}{T <: T} \text{ (S-REFL)} \rrbracket &= \lambda x : \llbracket T \rrbracket . x \\ \llbracket \frac{C_1 :: S <: U \quad C_2 :: U <: T}{S <: T} \text{ (S-TRANS)} \rrbracket &= \lambda x : \llbracket S \rrbracket . \llbracket C_2 \rrbracket (\llbracket C_1 \rrbracket \ x) \end{aligned}$$

15.6 Semântica de coerção para sistema de subtipos

Semânticas de coerção

- **Lema:** Se $\mathcal{C} :: S <: T$, então $\vdash \llbracket \mathcal{C} \rrbracket : \llbracket S \rrbracket \rightarrow \llbracket T \rrbracket$



15.6 Semântica de coerção para sistema de subtipos

Semânticas de coerção

- Coerções interagem com o contexto de tipos
 - ✓ interpretadas como **derivações de tipos**

$$\left[\left[\frac{\mathcal{D} :: \Gamma \vdash t : S \quad \mathcal{C} :: S <: T}{\Gamma \vdash t : T} \text{ (T-SUB)} \right] \right] = \llbracket \mathcal{C} \rrbracket \llbracket \mathcal{D} \rrbracket$$

15.6 Semântica de coerção para sistema de subtipos

Semânticas de coerção

- **Teorema:** Se $\mathcal{D} :: \vdash t : T$, então $\llbracket \Gamma \rrbracket$ é a extensão ponto a ponto entre os contextos de tipos $\llbracket \emptyset \rrbracket = \emptyset$ e $\llbracket \Gamma, x:T \rrbracket = \llbracket \Gamma \rrbracket, x:\llbracket T \rrbracket$



Semânticas de coerção

- Cuidado com a **coerência**
 - ✓ Derivação de tipos não deve incluir ambiguidades na linguagem
- **Definição:** Uma tradução $\llbracket - \rrbracket$ para a derivação de tipos de uma linguagem para termos de outra é coerente se, para cada par de derivações \mathcal{D}_1 e \mathcal{D}_2 com a mesma conclusão $\Gamma \vdash t : T$, as traduções $\llbracket \mathcal{D}_1 \rrbracket$ e $\llbracket \mathcal{D}_2 \rrbracket$ apresentam o mesmo comportamento na linguagem de destino.

15.7 Os tipos Interseção e União

$$T_1 \wedge T_2 <: T_1 \quad (\text{S-INTER1})$$

$$T_1 \wedge T_2 <: T_2 \quad (\text{S-INTER2})$$

$$\frac{S <: T_1 \quad S <: T_2}{S <: T_1 \wedge T_2} \quad (\text{S-INTER3})$$

$$S \rightarrow T_1 \wedge S \rightarrow T_2 <: S \rightarrow (T_1 \wedge T_2) \quad (\text{S-INTER4})$$

De forma semelhante para o tipo **União**

SLIDES KADICO

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