Compiladores – Análise de Tipos

Fabio Mascarenhas – 2015.2

http://www.dcc.ufrj.br/~fabiom/comp

Tipos

- Um *tipo* é:
  - Um conjunto de valores
  - Um conjunto de operações sobre esses valores
- Os tipos de uma linguagem podem ser pré-definidos, mas normalmente as linguagens também permitem que o programador defina seus tipos
- Os tipos de uma linguagem formam sua própria mini-linguagem

podem

GRAMÁTICA

Sistema de Tipos

- O *sistema de tipos* de uma linguagem especifica a *sintaxe* dos tipos, e quais operações são válidas nesses tipos

- O compilador usa as regras do sistema de tipos para fazer a *verificação de tipos* do programa

- O objetivo é rejeitar programas que contêm operações inválidas

- Várias linguagens adiam essa verificação até o momento em que o programa está executando

Tipagem estática/dinâmica e forte/fraca

- Uma linguagem tem *tipagem forte* se a verificação de tipos sempre é feita para todas as operações
  - A maior parte das linguagens (incluindo Java) tem tipagem forte, pois ela tem implicação direta na *segurança* dos programas
  - A linguagem C tem tipagem fraca, pois o sistema de tipos é facilmente "desligado", podendo-se manipular diretamente os bytes da memória
- Uma linguagem é *estaticamente tipada* se quase toda a verificação de tipos é feita pelo compilador antes do programa ser executado, e *dinamicamente tipada* se quase toda a verificação é feita no momento de execução

Verificação de Tipos Estática

- Poderíamos dar todas as regras de verificação de tipos de uma linguagem informalmente, mas existem formalismos que tornam essa especificação mais precisa

- A especificação das regras de verificação de tipo de uma linguagem se dá através de *regras de dedução*

- As regras de dedução dão um esquema de como podemos *deduzir* o tipo de uma expressão dados os tipos de suas subexpressões

- Os *axiomas* do sistema de tipos dão a tipagem dos literais e identificadores que aparecem no programa

Regras de Dedução

- Tradicionalmente usamos uma notação “barra” para as regras de dedução, em que as hipóteses da regra ficam acima de uma barra horizontal e a conclusão abaixo dessa barra

- Tanto as hipóteses quanto a conclusão são escritas da forma $\vdash e : t$, onde *e* é uma expressão, *t* um tipo e o símbolo $\vdash$ é a “catraca”

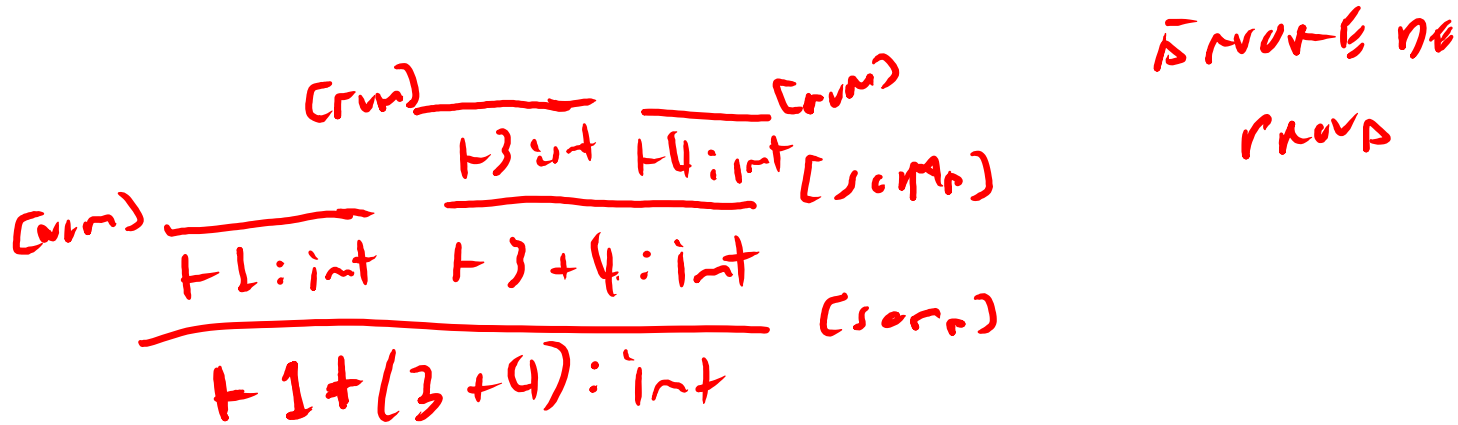
  - Lê-se “pode-se provar que *e* tem tipo *t*”

$$\frac{}{\vdash num : \text{int}}\ [num]$$

$$\frac{\vdash e_1 : \text{int} \quad \vdash e_2 : \text{int}}{\vdash e_1 + e_2 : \text{int}}\ [soma]$$

Exemplo – tipagem de expressões simples

- Vamos deduzir o tipo de 1 + (3 + 4):



Árvore de prova

Consistência e completude

- Como todo sistema lógico, podemos falar na *consistência* e *completude* de um sistema de tipos

- Um sistema de tipos é *consistente* se tudo que ele consegue provar é verdade, ou seja, se todo valor que uma expressão *e* com $\vdash e : t$ produz em tempo de execução tem tipo *t*

- Um sistema de tipos é *completo* se podemos tipar todos os programas corretos

- Em geral queremos que os sistemas de tipos sejam consistentes, mas dificilmente eles são completos

Exemplo - consistência

- A regra abaixo é consistente? Por quê?

$$\frac{\vdash e_1 : \text{int} \quad \vdash e_2 : \text{int}}{\vdash e_1 / e_2 : \text{boolean}}$$

- E quanto à regra abaixo?

$$\frac{\vdash e_1 : \text{int} \quad \vdash e_2 : \text{int}}{\vdash e_1 / e_2 : \text{int}}$$

- Consistência depende do comportamento da linguagem!

Subtipagem

- O conjunto de valores de um tipo pode ser um subconjunto do conjunto de valores de outro tipo
- Podemos querer expressar isso no sistema de tipos através de uma *relação de subtipagem* ≤
- Em uma linguagem OO essa relação é declarada pelo programador; em Java ela é dada pelas cláusulas *extends* e *implements*, e em MiniJava pela *extends*
- A relação de subtipagem é *simétrica* (t ≤ t) e *transitiva* (r ≤ s e s ≤ t implica r ≤ t)
- Podemos usar a relação de subtipagem explicitamente nas regras, ou podemos introduzir uma *regra de subsunção*

Subtipagem explícita vs. subsunção

- Subtipagem explícita:

$int \leq double$

$$\frac{\vdash e_1 : double \quad \vdash e_2 : t \quad t \leq double}{\vdash e_1 + e_2 : double}$$

- Subsunção:

$$\frac{\vdash e : t_1 \quad t_1 \leq t_2}{\vdash e : t_2}$$

$$\frac{\vdash e_1 : double \quad \vdash e_2 : double}{\vdash e_1 + e_2 : double}$$

- Usar subsunção deixa o sistema mais sintético, usar a subtipagem explícita deixa ele mais fácil de implementar

Tipagem de variáves

- Qual o tipo de uma variável?
- Não podemos determinar esse tipo sintaticamente, ele depende do *contexto*
- Vamos dar esse contexto usando uma *tabela de símbolos* que irá associar cada nome ao seu tipo declarado:

ambiente de tipos

T ⊢ *id*: T.procurar(id)

T[id]

- Declarações de variáveis inserem os tipos na tabela

$$\frac{T \vdash e_1 : int \quad T \vdash e_2 : int}{T \vdash e_1 + e_2 : int}$$

- A verificação de tipos pode ser feita em paralelo com a análise de escopo!

Tipos em TINY

- Atualmente todas as variáveis em TINY são números inteiros, e a própria sintaxe da linguagem está garantindo que todas as operações do programa são válidas

- Vamos mudar a linguagem para ter três tipos, `int`, `real` e `bool`, com `int` ≤ `real`, incluindo declaração de tipos na própria linguagem

```
VAR  -> var DECLS ;
      |
DECLS -> DECLS , DECL
       | DECL
DECL -> IDS : TIPO
IDS  -> IDS , id
      | id
```

```
TIPO -> int
      | real
      | bool
```

Tipagem de expressões

- As expressões aritméticas possem tipo inteiro se ambos os operandos forem inteiros; um operando real faz elas terem tipo real

- O verificador de tipos pode inserir *casts* (conversões de tipo) explícitos nos pontos em que precisa usar subtipagem, para facilitar o trabalho do gerador de código

$$\frac{T \vdash e_1 : int \quad T \vdash e_2 : int}{T \vdash e_1 \oplus e_2 : int}$$

$$\frac{T \vdash e_1 : real \quad T \vdash e_2 : t \quad t \leq real}{T \vdash e_1 \oplus e_2 : real}$$

$$\frac{T \vdash e_1 : t \quad T \vdash e_2 : real \quad t \leq real}{T \vdash e_1 \oplus e_2 : real}$$

$$\frac{T \vdash e_1 : t_1 \quad T \vdash e_2 : t_2 \quad t_1 \leq real \quad t_2 \leq real}{T \vdash e_1 \leq e_2 : bool}$$

$$T \vdash num : int$$

$$T \vdash id : T[id]$$

Tipagem de comandos

$\Gamma \vdash c$

- Os comandos TINY por si só não têm tipos, mas as regras de tipagem garantem que toda expressão usada dentro de um comando está consistente

$$\frac{\Gamma \vdash e : bool \quad \Gamma \vdash b_1, \Gamma_1 \quad \Gamma \vdash b_2, \Gamma_2}{\Gamma \vdash \text{if } e \text{ then } b_1 \text{ else } b_2 \text{ end}}$$

$$\frac{\Gamma \vdash id : t_2 \quad \Gamma \vdash e : t_2 \quad t_2 \leq t_1}{\Gamma \vdash id := e}$$

$$\frac{\Gamma \vdash b, \Gamma_1 \quad \Gamma_1 \vdash e : bool}{\Gamma \vdash \text{repeat } b \text{ until } e}$$

$$\frac{\Gamma[id : t] \vdash e}{\Gamma \vdash \text{var } id : t; e, \Gamma[id : t]}$$

$$\frac{\Gamma \vdash e : t}{\Gamma \vdash \text{write } e}$$

$$\frac{\Gamma \vdash id : t}{\Gamma \vdash \text{read } id}$$