Compiladores II

Fabio Mascarenhas - 2014.2

http://www.dcc.ufrj.br/~fabiom/comp2

SmallLua - sintaxe

```
bloco <- stat* (ret / '')
stat <- "while" exp "do" bloco "end" / "local" "id" "=" exp /</pre>
         "id" "=" exp / "function" "id" "(" (ids / '') ")" bloco "end" /
         "if" exp "then" bloco ("else" bloco / '') "end" /
         pexp
ret <- "return" exp
ids <- "id" ("," "id")*
exps <- exp ("," exp)*
exp <- lexp ("or" lexp)*
lexp <- rexp ("and" rexp)*</pre>
rexp <- cexp (rop cexp)*</pre>
cexp <- aexp ".." cexp / aexp
                                                             ex presson
aexp <- mexp (aop mexp)*</pre>
mexp <- sexp (mop sexp)*</pre>
sexp <- "-" sexp / "not" sexp / ("false")</pre>
                                                      "number"
   "string" | lmb / pexp
lmb <- "function" "(" (ids / '') ")" bloco "end"</pre>
pexp <- ("(" exp ")" / "id") ("(" (exps / '') ")")*</pre>
    <- "<" / "==" / "~="
rop
aop <- "+" / "-"
    <- "*" / "/"
mop
```

Tipagem estática

- Lua é uma linguagem dinamicamente tipada: o interpretador Lua sabe o tipo de cada valor, e verifica cada operação antes de ser executada, para checar se os tipos dos operandos estão corretos
- Em linguagens estaticamente tipadas, o *compilador* sabe o tipo de cada *termo* (expressão e variável) do programa, e pode verificar se os tipos dos operandos de cada operação estão corretos sem precisar executar o programa
- Um sistema de tipos é um sistema lógico que dá suporte à verificação de tipos em uma linguagem estaticamente tipada

Um sistema de tipos simples para SmallLua

- Nossa linguagem SmallLua tem três tipos atômicos: number, boolean e string
- Funções são valores de primeira classe em SmallLua, então também temos tipos compostos: tipos da forma (t1, ..., tn) -> tr representam funções de n parâmetros de tipos t1 a tn, e tvalor de retorno de tipo tr.
- Funções que não retornam nada têm tipo de retorno () que chamamos de unit (é como o tipo void de C, só que aqui é um tipo de primeira classe)
- A PEG abaixo dá a sintaxe dos tipos de SmallLua:

```
type <- "number" / "string" / "boolean" / tfunc / "(" ")"
tfunc <- "(" types ")" "->" type / ' ( ' ') ' ' ' ' ' '
types <- type ("," type)*</pre>
```

Anotações de tipos

- A forma mais simples de implementar tipagem estática é exigindo anotações de tipos explícitas
- Em SmallLua, podemos anotar apenas parâmetros e tipos de retorno de funções, já que as variáveis locais podem pegar seus tipos da sua expressão de inicialização

```
-- o mesmo para stat
lmb <- "function" "(" (prms / '') ")" ":" type bloco "end"
prms <- "id" ":" type ("," "id" ":" type)*</pre>
```

Regras de dedução

- As regras de dedução de um sistema de tipos dão um esquema de como podemos deduzir o tipo de uma expressão dados os tipos de suas subexpresões
- 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 ⊢ e: t, onde e é uma expressão, t um tipo e o símbolo ⊢ é a "roleta"
 - ⊢ n: number ~~~
 - Lê-se "pode-se provar que e tem tipo t"

Tipagem de variáves

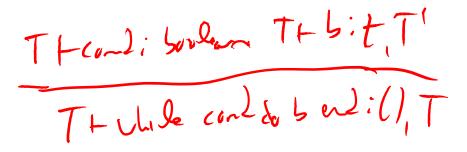
- Qual o tipo de uma variável?
- Não podemos determinar esse tipo sintaticamente, ele depende do contexto
- Vamos dar esse contexto usando um <u>ambiente de tipos</u> que associa nomes de variáveis a seu tipo:

$$\frac{(x \to t) \in T}{\text{T} \vdash x : t}$$

• Declarações de variáveis estendem o ambiente:

$$\frac{T \vdash e: t}{T \vdash local \ x = e: (), T[x \to t]}$$

Sequência de comandos



- Para tipar uma sequência de comandos, examinamos o primeiro comando da sequência: se o tipo dele for (), usamos o ambiente estendido para tipar o resto da sequência
- Se o tipo dele n\u00e3o for (), ignoramos o resto da sequência, e esse tipo \u00e9 o tipo da sequência
- Isso permite uma maneira simples de tipar o efeito do comando return

Funções

- Para tipar a declaração de uma função, estendemos o ambiente associando o nome de cada parâmetro a seu tipo, depois tipamos o corpo da função, e checamos se o tipo bate com o tipo de retorno declarado
- Tipar a chamada de uma função é mais simples, bastando verificar que o número de argumentos bate com o de parâmetros (aridade), e o tipo de cada argumento bate com o de cada parâmetro