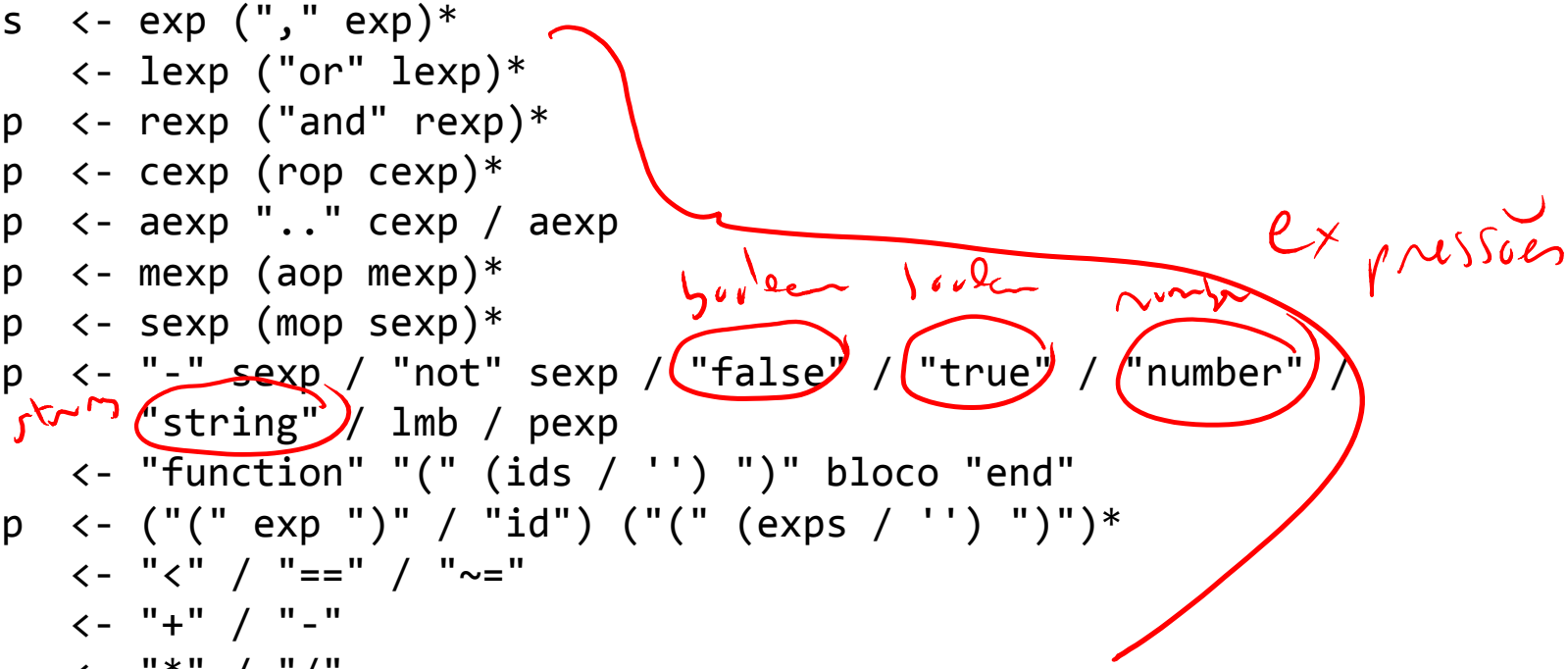
Compiladores II

Fabio Mascarenhas - 2014.2

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

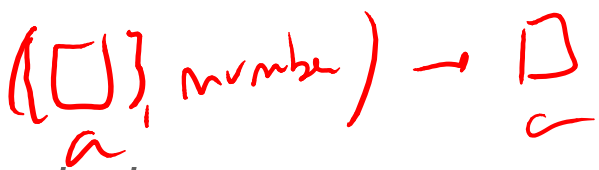
SmallLua - sintaxe

```
bloco <- stat* (ret / '')
stat  <- "while" exp "do" bloco "end" / "local" "id" "=" exp /
         "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)*
rexp  <- cexp (rop cexp)*
cexp  <- aexp ".." cexp / aexp
aexp  <- mexp (aop mexp)*
mexp  <- sexp (mop sexp)*
sexp  <- "-" sexp / "not" sexp / "false" / "true" / "number" /
         "string" / lmb / pexp
lmb   <- "function" "(" (ids / '') ")" bloco "end"
pexp  <- ("(" exp ")" / "id") ("(" (exps / '') ")")*
rop   <- "<" / "==" / "~="
aop   <- "+" / "-"
mop   <- "*" / "/"
```



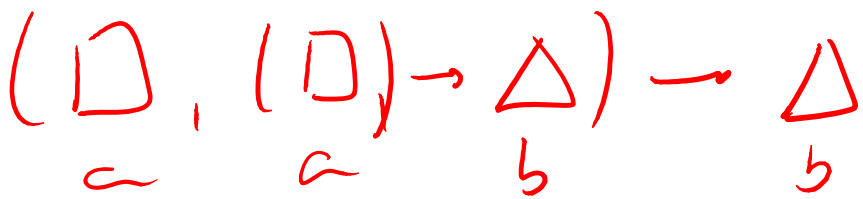
Parâmetros de tipos

- A ideia do *polimorfismo paramétrico* é poder ter *parâmetros* nos tipos, que são “buracos” onde podemos encaixar qualquer tipo



- Representamos esses parâmetros com *parâmetros de tipos*

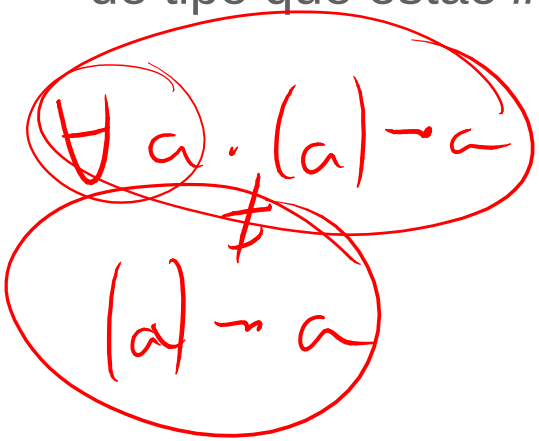
- Um tipo polimórfico tem um ou mais parâmetros como parte dele (podemos usar o mesmo parâmetro mais de uma vez!)



Generalização

- Criamos um tipo polimórfico por um processo de *generalização*: podemos generalizar um tipo toda vez que estamos definindo uma variável, e o tipo dessa variável corresponde a um valor *imutável* (funções e sequências, em SmallLua)

- Devemos tomar cuidado na generalização para só generalizar os parâmetros de tipo que estão *livres*



```
function foo(x: a): {number}
    local bar = seq(x)
    return seq(1) .. bar
end

function bar(x: a): number
    local baz = function (y: a): a
                    return x
                end
    return baz(1)
end
```