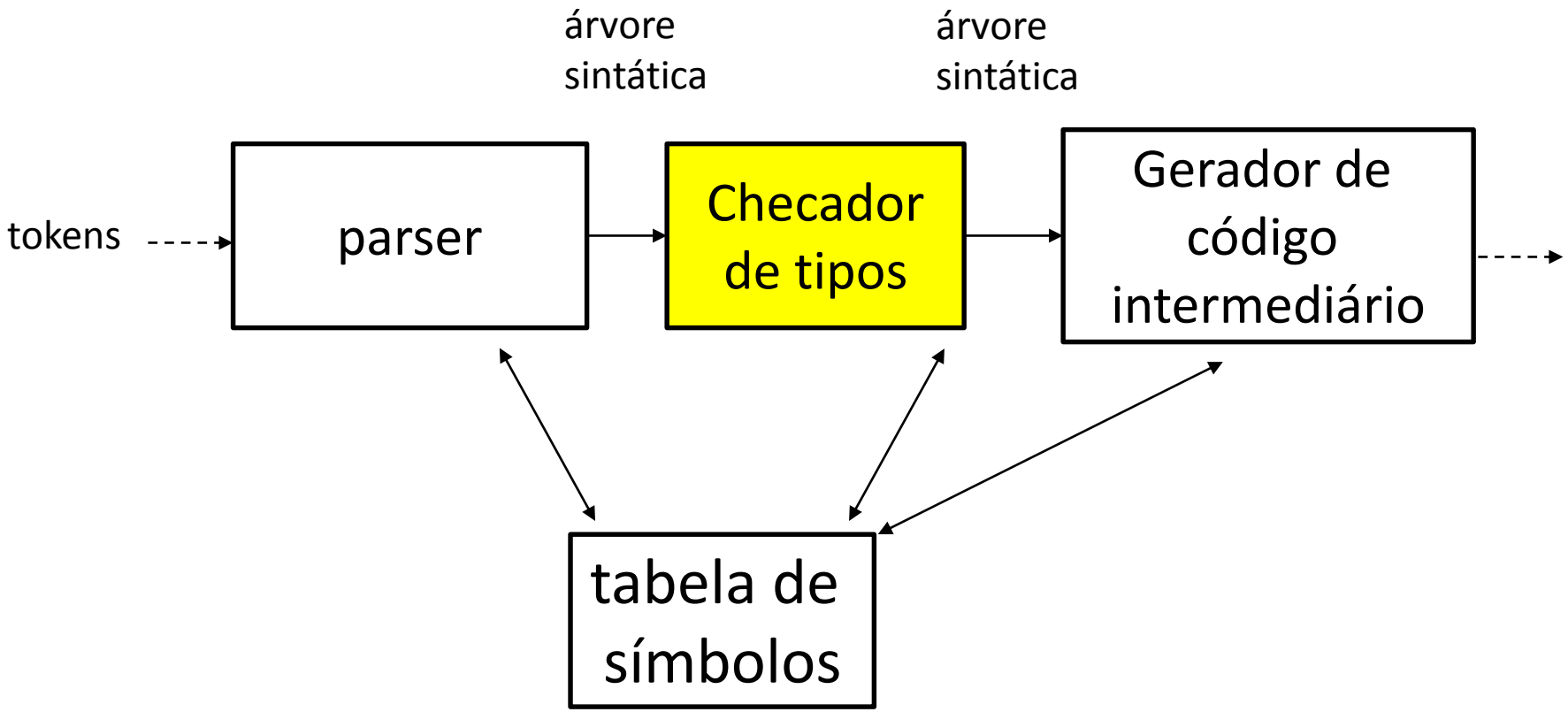
ANÁLISE SEMÂNTICA

Organização do compilador



Checagem de Tipos

- Procura certos tipos de erros semânticos
  - Estaticamente (abordagem pessimista)
    - Evita escapamento de erros simples
  - Dinamicamente (abordagem otimista)
    - Mais comum em linguagens de script

Exemplos de erro (ou warning)

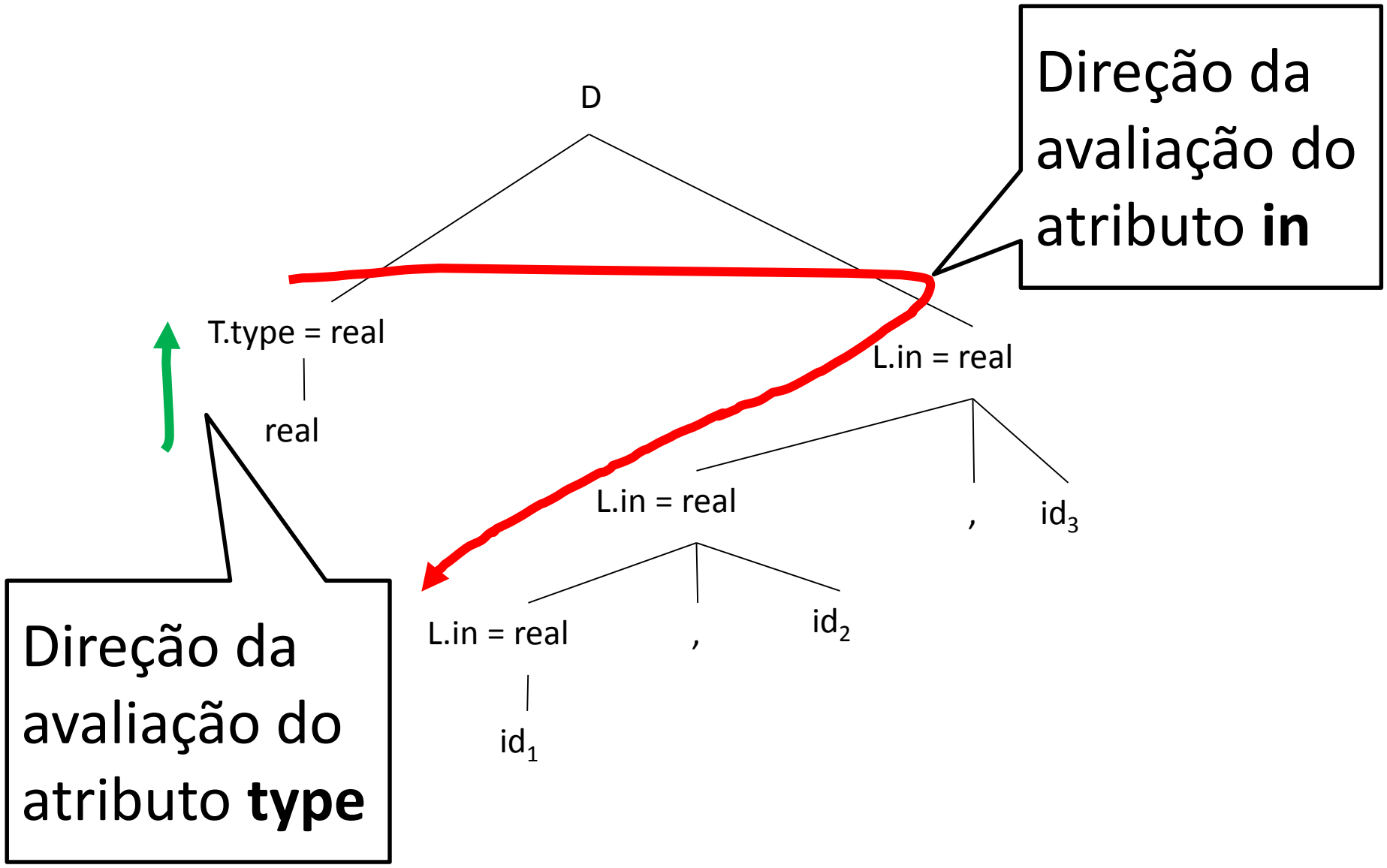
- Tipos incompatíveis
  - soma, atribuição, chamada de método, etc.
- Fluxo de controle
  - `break` ou `continue` fora de loop ou `switch`
- Definição de variável sem uso (warning)
- Uso de variável sem definição
- Indexação em variável não array
- De-referência em variável não ponteiro, etc.

Exemplo de Gramática de Atributos

Produção	***Regra semântica***
D → T L	L.in = T.type
T → **int**	T.type = integer
T → **real**	T.type = real
L → L_1 **, id**	L_1.in = L.in
L → **id**	addtype(**id**.entry, L.in)

Atributo **type** é sintetizado e atribuite **in** é herdado

Árvore de “**real** id1, id2, id3” decorada



Exercício (aberto)

- Considerando programas com apenas uma função, como você faria para capturar uso de variáveis sem declaração?

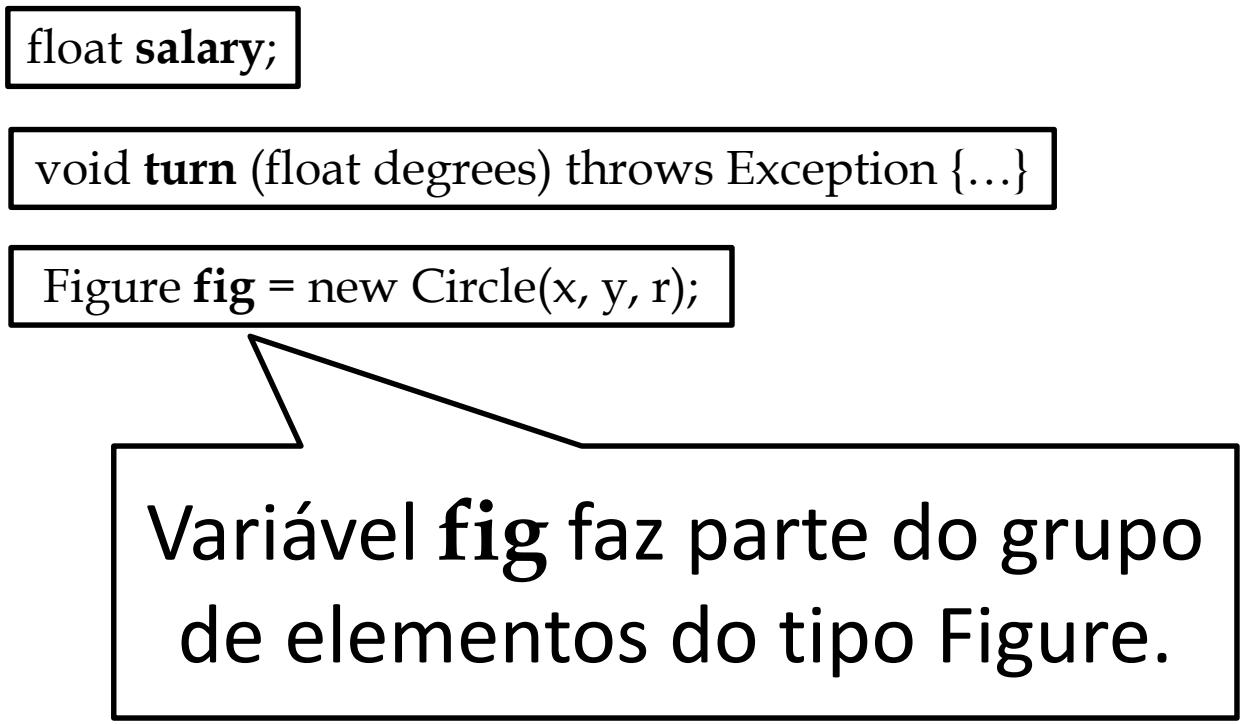
Resposta

- Defina um atributo sintetizado reach
  - O atributo reach armazena um conjunto de variáveis usadas potencialmente sem definição
    - Cada novo uso atualiza o conjunto
    - Uma variável deve ser eliminada do conjunto caso uma declaração correspondente for encontrada
    - Se reach for não vazio no nó raiz, declare erro

SISTEMA DE TIPOS

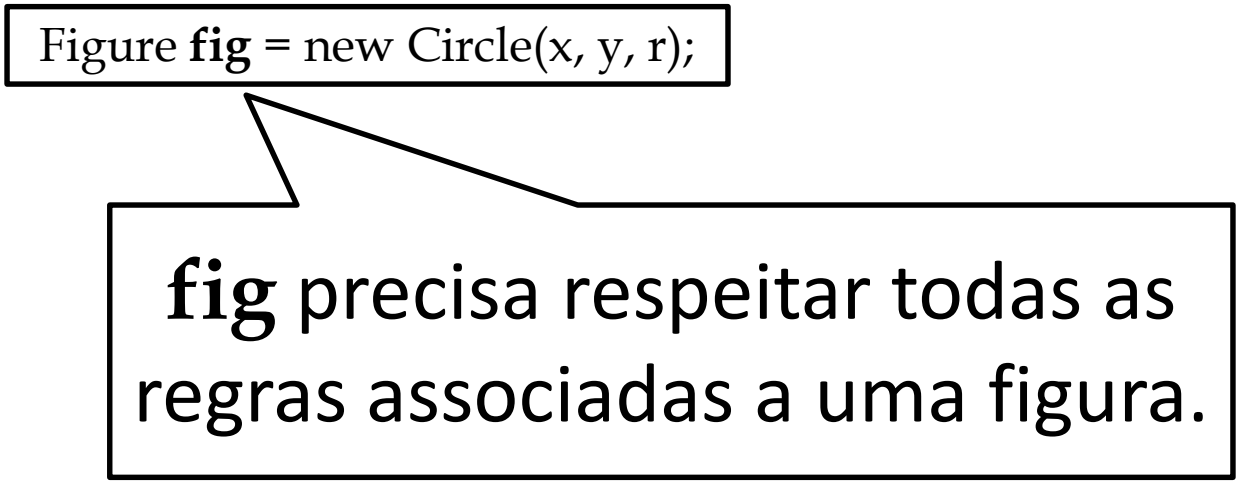
Tipos

- Um tipo é uma categoria de elementos de programação visando seu uso disciplinado



Sistema de Tipos

- Coleção de regras que que limitam como programas pode ser escritos
  - Violação de regra => erro de tipo



Tipos escalares e derivados

- Escalares representa um elemento (e.g., integer), tipos derivados representam vários
- Exemplos de expressões de tipos (derivados)

Array

array[1..10] of integer

Ponteiro

^row

Produto cartesiano

$T_1 \times T_2$

Função

integer × integer → integer

Tipo Registro

```
record
  address: integer;
  lexeme: array [1..15] of char;
end;
```

Exemplos de regras de tipo

- Pascal
  - "Se os dois operandos dos operadores aritméticos de adição, subtração e multiplicação são do tipo inteiro, então o resultado é do tipo inteiro."
- C (manual de referência)
  - "O resultado do operador unário & é um ponteiro para o obj. referido pelo operando. Se o tipo do operando for T o tipo do resultado é ponteiro para T"

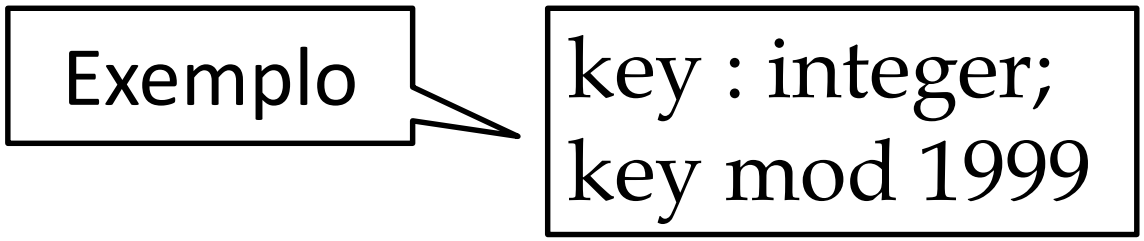
Sistema de tipos simples

P → D **;** E

D → D **;** D | **id :** T

T → **char** | **integer** | **array [num] of** T | **^**T

E → **literal** | **num** | **id** | E **mod** E | E **[** E **]** | E**^**



Sistema de tipos simples

P → D **;** E

D → D **;** D

D → **id :** T {addtype(id.entry, T.type)}

T → **char** {T.type = char}

T → **integer** {T.type = integer}

T → **array [num] of** T
{T.type = array(1..num.val, T1.type)}

T → **^**T {T.type = pointer(T_1.type)}

Sistema de tipos simples

E → **literal** {E.type = char}

E → **num** {E.type = integer}

E → **id** {E.type = lookup(id.entry)}

E → E **mod** E

{E.type = if (E_1.type == integer &&
E_2.type == integer) integer
else type_error }

Sistema de tipos simples

$E \rightarrow E\ [\ E\ ]$

{E.type = if (E_2.type == integer &&
E_1.type == array(s,t)) = t
else type_error }

$E \rightarrow E\ \hat{}$

{E.type = if (E_1.type == pointer(t)) t
else type_error }

Exercício

- Adicione regras semânticas para comandos

S → **id =** E

S → **if** E **then** S

S → **while** E **do** S

Resposta

S → **id =** E

{if (lookup(id.entry) != E.type) type_error }

S → **if** E **then** S

{if (E.type != boolean) type_error }

S → **while** E **do** S

{if (E.type != boolean) type_error }

Exercício

- Adicione regras semânticas às novas produções da gramática, considerando chamada de métodos (com apenas um arg.)

$$T \rightarrow T \rightarrow T$$

$$E \rightarrow E \; (\; E \;)$$

Resposta

$T \rightarrow T \rightarrow T$

$\{T.type = T_1.type \rightarrow T_2.type\}$

$E \rightarrow E$ **(** E **)**

$\{E.type = if\ (E_2.type == s\ \&\&$

$E_1.type == s \rightarrow t)\ t$

$else\ type\_error\ \}$

Exercício

- Escreva visitors para fazer a checagem de tipos conforme definição anterior

DEMO

OUTROS CONCEITOS RELEVANTES

Conversão de tipos

- **Co(nv)ersão: Implícita e Explícita**
- **Implícita**: O compilador adiciona implicitamente função de conversão
  - Normalmente não há perda de informação. Por exemplo, de int para float
- **Explícita**: O programador precisa indicar

Casting

```
void interpret(Instruction insn) {
  switch (insn.opcode()) {
  case Opcode.IADD :
     execArith((ArithInstruction) insn));
     break;  …}
```

Tempo de checagem de uma linguagem

- **Statically**-typed
  - checagem durante compilação
- **Dynamically**-typed
  - Checagem durante execução

Motivo de grande debate até hoje!

"Dynamic" mais flexível, porém pode deixar escapar erros e checagem de tipos pode ser custosa.

Rigor de uma linguagem

- **Strongly**-typed
  - Não deixam escapar certos tipos de erros
- **Weakly**-typed
  - São mais flexíveis; é possível que erros escapem

Diferença está na flexibilidade no uso de tipos. C permite fazer casting de tipos **não relacionados**.

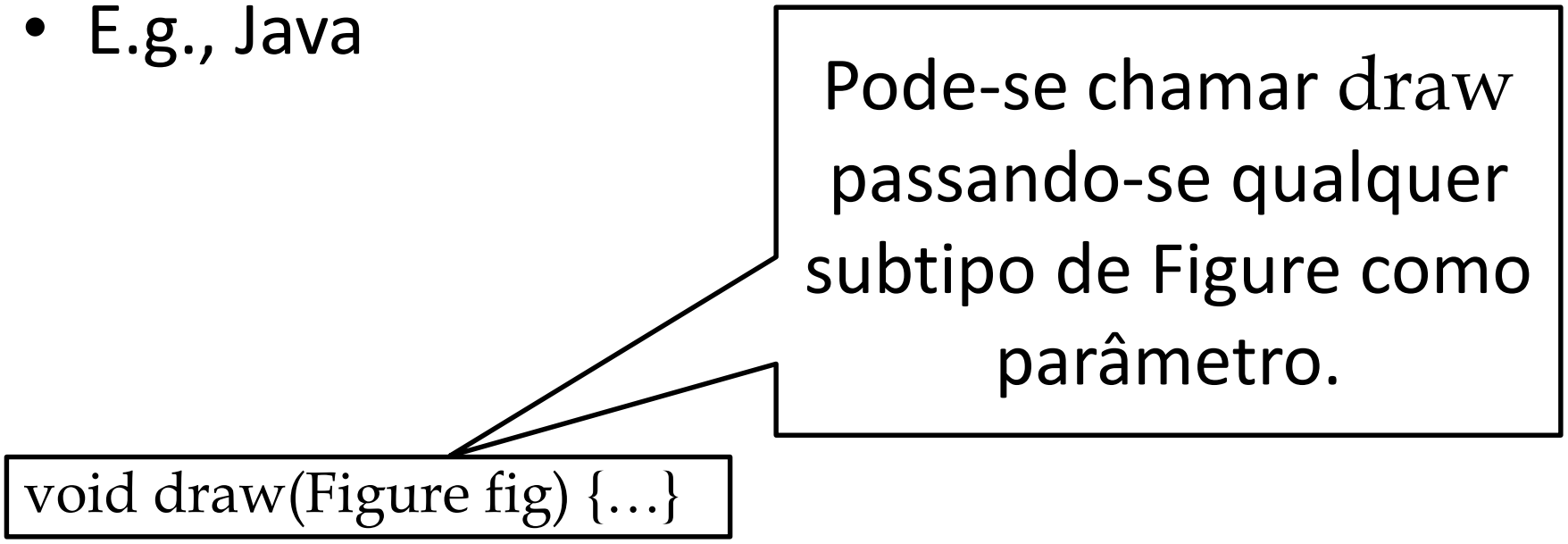
Na prática, algumas verificações só podem ser feitas dinamicamente. E.g., índices de arrays.

Polimorfismo *ad hoc* (sobrecarga)

- Mesmo **nome** de função (ou operador) é usado em contextos diferentes
- Exemplos
  - Java: operador + para strings, inteiros, reais
  - Ada: operador “()” usado para indexar arrays, chamada de funções, e conversão de tipos

Polimorfismo de subtipo

- Qualquer subtipo de T pode ser usado no contexto onde objeto de tipo T é esperado
- E.g., Java



Exemplo Java

```
interface Figure {}

class BitMapFigure implements   Figure {…}
class VectorFigure implements Figure {…}

class Circle extends VectorFigure {}
class Square extends VectorFigure {}

class Util {
  void draw(Figure fig) {…}
}

void main() {
  Figure fig = new Circle(...);
  Util.draw(fig);
  fig = new Square(...);
  Util.draw(fig);
}
```

Aceita qualquer tipo de figura!

Polimorfismo paramétrico

- Facilita definição de funções que manipulam objetos com estruturas diferentes
- E.g. definição da função length para listas

Está para linguagens funcionais assim como polimorfismo de subtipo está para OO.

OO dá suporte com generics.

Exemplo negativo em Pascal

```
type link = ^cell;
        cell = record info: integer; next: link; end;
function length (lptr : link) : integer;
var len : integer;
begin len = 0;
        while (lptr <> nil) do
        begin
           len := len + 1;
           lptr := lptr^.next;
        end;
        length := len;
end;
```

Exemplo positivo em Haskell

t é uma variável de tipo. Neste caso, qualquer lista é aceita como parâmetro.

```
length :: [t] -> Int;
length [] = 0
length (a:as) = 1 + length as
```

Exemplo positivo em Java

```
interface List<T> { int size(); ...}
...
void main() {
  List<?> li = new ArrayList<Integer>();
  System.out.println(li.size());
}
```

Tipos é um conceito adaptável

http://types.cs.washington.edu/checker-framework/

- Sistema de tipos plugáveis
  - Checker Framework
    - Usa anotação para declaração de tipos
- Exemplo “Nullness Checker”

```
void draw(@RequiresNonNull Figure fig) {…}
```

Toda chamada a draw precisa garantir que o tipo do parâmetro passado é **@NonNull**.