$\emptyset^* = \{\varepsilon\}$

U

5. Análise semântica #teórico-prático

deve garantir que o programa fonte faz sentido

erros + comuns	variável/função não definida
	tipos incompatíveis atribuir valor real a variável inteira
	instruções em contextos errados break fora de ciclo for
	aplicação sem sentido de instrução import de package inexistente

deteção em tempo dinâmico, ou seja, durante o tempo de execução ou em tempo estático, ou seja, durante tempo de compilação

em alguns casos a verificação pode ser feita durante a análise sintática, no entanto muitas vezes só pode ser feita depois da construção da árvore sintática
↳ avaliação dirigida pela sintaxe

Sistemas de tipos

sistema formal com um conjunto de regras semânticas que por associação de uma propriedade (tipo) a entidades de linguagem (expressões, variáveis, métodos, ...) permitem a deteção de erros de tipo

é aplicável a várias operações, sendo válidas quando existe conformidade entre as propriedades de tipo das entidades

↳ se um tipo T_1 pode ser usado onde se espera um T_2

- operações aplicáveis
  - atribuição de valor $v = e$
    o tipo de e tem de ser conforme com o tipo de v
  - aplicação de operadores $e_1 + e_2$
    existe um operador + aplicável aos tipos de e_1 e e_2
  - invocação de funções $f(a)$
    ∃ função f que aceita argumentos a conformes com os argumentos formais declarados nessa função
  - utilização de classes ou estruturas data.field
    existe um campo field na estrutura de data

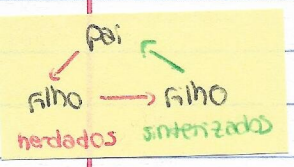
argumento formal
indicado na declaração formal da função

argumento atual
indicado aquando da invocação da função

gramáticas de atributos

tem como objetivo **associar** atributos aos nós da árvore sintática

cada nó pode ter 0 ou + atributos de qualquer tipo, tendo em consideração a dependência de informação necessária, podendo ainda alguns depender da vizinhança (operações aritméticas) e outros de informação remota (+ distante na árvore) (invocação de variáveis)



dependência local | sintetizados dependem apenas de nós descendentes 1+2
herdados dependem de nós irmãos ou ascendentes (depende de int a;)

rever

podemos declarar atributos na gramática diretamente, ou utilizar o array associativo ParseTreeProperty. Em alternativa, visitors.

para simular a passagem de argumentos no Listener c/ ParseTreeProperty basta atribuir-lhe o valor antes de percorrer o respetivo nó (enter...) para o retorno, usa-se o exit...

síntese | atributos sintetizados Listener ou visitor
atributos herdados Listener
atributos locais variáveis locais

tabela de símbolos

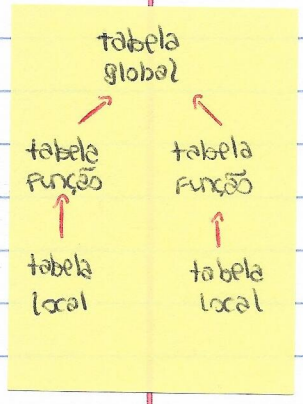
para lidarmos com símbolos que não tenham dependência local, ou seja, direta na árvore sintática, ou que resida no processamento de outro código fonte

símbolos
variáveis
funções
registos
classes

sempre que a linguagem utiliza símbolos para representar entidades do programa, torna-se necessário associar à sua identificação a sua definição
=> tabela de símbolos que é um array associativo

alcance pode ser global ou local (p.e numa função)

propriedades nome nome do símbolo (chave do array)
categoria o que representa chave, método, variável
tipo tipo de dados do símbolo
valor associado só nos interpretadores
visibilidade restrições de acesso para linguagens com encapsulamento



agrupar símbolos por contextos

numa linguagem simples com um único contexto o tempo de vida dos seus símbolos é o do programa, bastando uma tabela de sim.

no entanto, se houverem mais contextos é fundamental definir os seus tempos de vida e/ou visibilidade

VIP os contextos podem ser definidos dentro de outros contextos

representação desta informação faz-se estruturando as diferentes tabelas de símbolos numa árvore onde cada nó representa uma pilha de tabelas de símbolos a começar nesse nó até à raiz (global)

pode haver repetição de nomes de símbolos, valendo o definido na tabela mais próxima

instruções restringidas por contexto

para restringir instruções a determinados contextos, pode-se efetuar a verificação semântica durante a análise sintática, recorrendo a predicados semânticos e um contador (pilha) que regista o contexto

exemplo

```
@parser::members {
  int acceptBreak = 0;
}

ForLoop : 'for' '(' expr ';' expr ';' expr ')'
    {acceptBreak++}
    instruction
    {acceptBreak--}
    ;

break: {acceptBreak > 0}? 'break' ';'
    ;
```