
***Alice Rezende Ribeiro - 201810243***
***Mateus Carvalho Gonçalves - 201810245***

**Introdução**

A análise semântica é a etapa da compilação que trata de aspectos relacionados ao significado e contexto das instruções. Ela ocorre simultaneamente à análise sintática e utiliza a Tabela de Símbolos - definida na análise léxica - para auxiliar no processo.

Esta etapa do projeto de compiladores tem como objetivo, entre outras coisas, checagem de tipo, escopo e declarações de identificadores, também detectar a unicidade dos nomes deles e detectar uso correto de comandos de controle de fluxo. Como essas tarefas envolvem análise de meta-informações e que dependem de contexto, muitas vezes com elementos situados em diferentes partes do código fonte, as abordagens feitas anteriormente não resolvem o problema. Seria necessária a utilização de linguagens sensíveis ao contexto, porém o ganho é mínimo e o esforço é grande para a aplicação desse conceito. Por isso, atualmente usa-se a Tradução Dirigida pela Sintaxe para solucionar o problema.

É interessante ressaltar que, com essa abordagem, a análise semântica se torna uma "camada" da análise sintática, ou seja, são adicionados elementos e verificações às regras de sintaxe definidas anteriormente. Também, diferentemente das etapas anteriores, a análise semântica é uma etapa mais "artesanal" no projeto de compiladores, em que as regras são mais versáteis à finalidade da linguagem alvo, abrindo espaço para várias decisões aos projetistas.

1. Na etapa léxica, o programa é transformado em tokens.
2. Na etapa sintática, tem-se a validação das regras.
3. Na etapa semântica, verifica-se aspectos de tipos, escopos, declarações e como eles estão no código como um todo. Algumas perguntas a serem respondidas são:

- x foi declarado somente uma vez?
- x foi declarado antes do uso?
- x foi declarado mas não foi utilizado?
- o que x se refere?
- os tipos da expressão são compatíveis?
- entre outros

**Tradução Dirigida Pela Sintaxe**

A Tradução Dirigida pela Sintaxe é um esquema de gramática que inclui atributos para cada símbolo da gramática + ações semânticas.

Os atributos incluídos podem ser de dois tipos: herdados ou sintetizados. Os atributos herdados são aqueles que são definidos a partir de um nó ancestral, já os sintetizados são definidos a partir dos nós descendentes.

Dependendo do tipo de análise sintática usamos esquemas de tradução diferentes. Para análises ascendentes, utilizamos esquemas S-atribuídos, isso quer dizer que utilizaremos apenas atributos sintetizados na tradução; enquanto nas análises descendentes, utilizamos esquemas L-atribuídos, que fazem uso de atributos herdados e também sintetizados, isso porque primeiro descemos na árvore sintática e depois subimos com os resultados das ações semânticas.

As ações semânticas são verificações, feitas em código de programação, disparadas quando uma produção da gramática é chamada, utilizando os atributos. Contextualizando, os atributos são armazenados na tabela de símbolos (colunas) para serem acessados quando necessário.

**Tabela de Símbolos**

A tabela de símbolos é uma estrutura auxiliar fundamental para a análise semântica, ajuda a identificar a sensitividade do contexto ao passo que armazena as informações necessárias durante o esquema de tradução.

Assim, a tabela é acessada sempre que um elemento é mencionado:

- Busca se tem declaração;
    - Se não, adiciona;
- Verifica tipo, escopo ou outro atributo;
- Atualiza as informações;
- Remove quando não é necessário. Por exemplo, quando um escopo acaba retira suas variáveis da tabela.

Em um projeto de compiladores, várias escolhas devem ser feitas na hora de implementar a tabela, como armazenamento, a estrutura de dados utilizada, a função de acesso, os campos implementados, entre outras coisas. A principal decisão tratada neste relatório é a questão do número de tabelas, ou seja, todas as informações dos tokens serão guardadas em uma única tabela ou seriam criadas mais tabelas? A criação de múltiplas

tabelas pode ajudar principalmente na checagem de escopo de identificadores. Nesse caso, teremos uma tabela principal que irá apontar para diferentes tabelas de diferentes escopos.

**Verificação de tipos**

Como pode-se perceber, as verificações semânticas são feitas basicamente utilizando os esquemas de tradução com apoio da tabela de símbolos para acesso às informações.

Para verificar de tipos, por exemplo, podemos adicionar as seguintes ações semânticas na gramática ALGUMA, considerando que o parser é ascendente e os únicos tipos existentes são NumI, NumR e Str (valores booleanos seriam 0 ou 1):

| | |
|---|---|
| **1** | <ATRIBUICAO> ::= ATR <EXPARIT> A Var |
| **2** | <EXPARIT> ::= <TERMO> OpArit <EXPARIT$_1$> |
| **3** | <EXPARIT> ::= <TERMO> |
| **4** | <TERMO> ::= NumI |
| **5** | <TERMO> ::= NumR |
| **6** | <TERMO> ::= Var |

⇩

| | |
|---|---|
| **1** | if (EXPARIT == "Str" OR (EXPARIT == "NumR" AND Var.tipo == "NumI")) {<br>  semanticError();<br>} else {<br>  //fazer atribuicao de valor e atualizar tabela de símbolo<br>  //inclusão de mais verificações semânticas no processo<br>} |
| **2** | if (TERMO.tipo == "Str" OR EXPARIT$_1$ == "Str") {<br>  semanticError();<br>} else if (TERMO.tipo == "NumR" OR EXPARIT$_1$ == "NumR") {<br>  EXPARIT.tipo = "NumR";<br>} else {<br>  EXPARIT.tipo = "NumI";<br>} |
| **3** | EXPARIT.tipo = TERMO.tipo |
| **4** | TERMO.tipo = NumI |
| **5** | TERMO.tipo = NumR |
| **6** | TERMO.tipo = Var.tipo |

Obs: não foi utilizada a última versão da gramática alguma para fins de facilitar a exemplificação.

Em cada momento de redução as regras semânticas são disparadas para verificação, assim, é possível prever erros em tempo de compilação. A tabela de símbolos é consultada sempre que um atributo é verificado.

**Verificação de escopos**

Existem duas implementações da tabela de símbolos mais utilizadas para essa verificação: apenas uma tabela com o atributo escopo representado por níveis, e múltiplas tabelas, uma para cada escopo.

O papel das ações semânticas no esquema de tradução, nesse caso, pode ser visto como a manipulação da(s) tabela(s) de símbolos, entre outras coisas. Definir, em código de programação, a criação das tabelas, incremento e decremento dos atributos, exclusão das tabelas, etc.

Por exemplo, no seguinte código:

```
01 x : INT
02 ATRIBUI 0 A x
03 :FOO1
04      y : INT
05      ATRIBUIR 2 A y
06      ATRIBUIR y A x
07 :FOO2
08      ATRIBUIR 400 A x
09 :FOO3 (y: INT)
10      ATRIBUIR y A x
11 :PRINCIPAL
12      IMPRIMIR x
13      FOO1
14      IMPRIMIR x
15      FOO2
16      IMPRIMIR x
17      FOO3 (y: 245)
18      IMPRIMIR x
19      IMPRIMIR y
```

Quando existe apenas uma tabela de símbolos para todos os escopos, esse atributo pode guardar um inteiro para sua definição. Inicia-se em 0, e a cada novo escopo, o atributo é incrementado, e quando um escopo termina, é decrementado e as linhas daquele escopo excluídas.

A seguir acompanhamos o estado da tabela de símbolos em função do tempo na abordagem citada no parágrafo anterior:

| **Cadeia** | **Token** | **Categoria** | **Tipo** | **Valor** | **Escopo** |
|---|---|---|---|---|---|
| x | id | var | integer | 0 | 0 |
| 0 | num | - | - | 0 | |

| **Cadeia** | **Token** | **Categoria** | **Tipo** | **Valor** | **Escopo** |
|---|---|---|---|---|---|
| x | id | var | integer | 0 | 0 |
| 0 | num | - | - | 0 | |

| | | | | | |
|---|---|---|---|---|---|
| FOO1 | id | proced | - | - | 0 |

| **Cadeia** | **Token** | **Categoria** | **Tipo** | **Valor** | **Escopo** |
|---|---|---|---|---|---|
| x | id | var | integer | 0 | 0 |
| 0 | num | - | - | 0 | |
| FOO1 | id | proced | - | - | 0 |
| y | id | var | integer | - | 1 |

| **Cadeia** | **Token** | **Categoria** | **Tipo** | **Valor** | **Escopo** |
|---|---|---|---|---|---|
| x | id | var | integer | 2 | 0 |
| 0 | num | - | - | 0 | |
| FOO1 | id | proced | - | - | 0 |
| y | id | var | integer | 2 | 1 |
| 2 | num | - | - | 2 | 1 |

| **Cadeia** | **Token** | **Categoria** | **Tipo** | **Valor** | **Escopo** |
|---|---|---|---|---|---|
| x | id | var | integer | 2 | 0 |
| 0 | num | - | - | 0 | |
| FOO1 | id | proced | - | - | 0 |
| y | id | var | integer | 2 | 1 |
| 2 | num | - | - | 2 | 1 |
| FOO2 | id | proced | - | - | 0 |

| **Cadeia** | **Token** | **Categoria** | **Tipo** | **Valor** | **Escopo** |
|---|---|---|---|---|---|
| x | id | var | integer | 400 | 0 |
| 0 | num | - | - | 0 | |
| FOO1 | id | proced | - | - | 0 |
| y | id | var | integer | 2 | 1 |

| | | | | | |
|---|---|---|---|---|---|
| 2 | num | - | - | 2 | 1 |
| FOO2 | id | proced | - | - | 0 |
| 400 | num | | | 400 | 1 |

| **Cadeia** | **Token** | **Categoria** | **Tipo** | **Valor** | **Escopo** |
|---|---|---|---|---|---|
| x | id | var | integer | 400 | 0 |
| 0 | num | - | - | 0 | |
| FOO1 | id | proced | - | - | 0 |
| y | id | var | integer | 2 | 1 |
| 2 | num | - | - | 2 | 1 |
| FOO2 | id | proced | - | - | 0 |
| 400 | num | | | 400 | 1 |
| FOO3 | id | proced | - | - | 0 |
| y | id | par | integer | - | 1 |

| **Cadeia** | **Token** | **Categoria** | **Tipo** | **Valor** | **Escopo** |
|---|---|---|---|---|---|
| x | id | var | integer | 400 | 0 |
| 0 | num | - | - | 0 | |
| FOO1 | id | proced | - | - | 0 |
| y | id | var | integer | 2 | 1 |
| 2 | num | - | - | 2 | 1 |
| FOO2 | id | proced | - | - | 0 |
| 400 | num | | | 400 | 1 |
| FOO3 | id | proced | - | - | 0 |
| y | id | par | integer | - | 1 |
| 50 | num | - | - | 50 | 1 |

…

No principal, os resultados que seriam impressos serão 0,  2, 400, 245 e na última o compilador irá apontar erro, já que a variável y não existe no escopo atual (global). Também destaca-se que o único motivo pela qual foi possível a declaração de diferentes variável com o nome y, foi por que elas se encontram em escopos diferentes.

Já no caso de tabelas diferentes, diferentes escopos ficam em tabelas diferentes, então, quando estamos em um escopo, verificamos somente sua tabela e a tabela do escopo global. Isso torna possível que diferentes variáveis com o mesmo nome sejam declaradas, assim como assumirem diferentes valores durante a execução.

**Referências bibliográficas**

Compiladores para humanos - Análise Semântica. Disponível em https://johnidm.gitbooks.io/compiladores-para-humanos/content/part1/semantic-analysis.html Acesso em 04/08/2020.

Linguagens e Compiladores - Tradução dirigida por sintaxe. Vídeo da UNIVESP disponível em https://www.youtube.com/watch?v=_kGAgukarkk&feature=youtu.be. Acesso em 04/08/2020.

Slides de aula 12. Prof. Celso Olivete Júnior. Unesp. Disponível em: http://docs.fct.unesp.br/docentes/dmec/olivete/compiladores/arquivos/Aula12.pdf. Acesso em 05/08/2020.