Instituto Tecnológico de Aeronáutica

Laboratório de CES-41

Série I: Questão 2

Gianluigi Dal Toso

Professor:
Fábio Carneiro Mokarzel

03 de Junho de 2020

Sobre os códigos e arquivos de dados

Anexos à este relatório está um arquivo comprimido em formato `.zip` contendo todos os códigos-fonte desenvolvidos para este exercício, bem como os arquivos de entrada e saída utilizados para testar a implementação de cada tarefa.

Questão 2: Análise Sintática

O Analisador Sintático foi implementado em linguagem de programação C, seguindo o padrão da implementação sugerida no Capítulo 3.4. Tentei incrementar o tratamento dos erros, tentando sempre identificar se uma parte futura do código poderia estar válida sintaticamente. Isso tornou os diagramas um pouco mais extensos, mas acredito que o resultado final fique bastante interessante. As Figuras 1 a 8 ilustram os diagramas de transição para o analisador sintático implementado.

Ademais, durante a implementação do código, fiz uma modificação na função que recebe os erros para imprimir (`Esperado()` de forma que ela receba como parâmetro duas *strings*: uma referente ao tipo esperado e outra contendo uma mensagem num formato mais amigável para o usuário tentando descrever o erro. Dessa forma, posso customizar a chamada dessa função para tentar facilitar o mapeamento dos erros, tornando mais fácil para o programador encontrar um erro sintático em seu código.

Em caso do programa passar pelo analisador léxico e ele não identificar nenhum erro, será impressa a mensagem "Programa livre de erro sintático!" no arquivo de saída. Caso contrário, o programa ira alertar o programador pelo console que erros foram encontrados durante a análise sintática e o relatório completo dos erros encontrados se encontrará no arquivo de saída. Para cada execução do analisador sintático, o arquivo de saída por padrão possui o nome do arquivo de entrada adicionado à extensão '`.log`'. Por exemplo, utilizando-se como entrada um arquivo de nome `arquivo.dat`, a saída da execução do analisador sintático será o arquivo `arquivo.dat.log`. Isto será utilizado para facilitar salvar as saídas e relacioná-las aos arquivos de entrada.

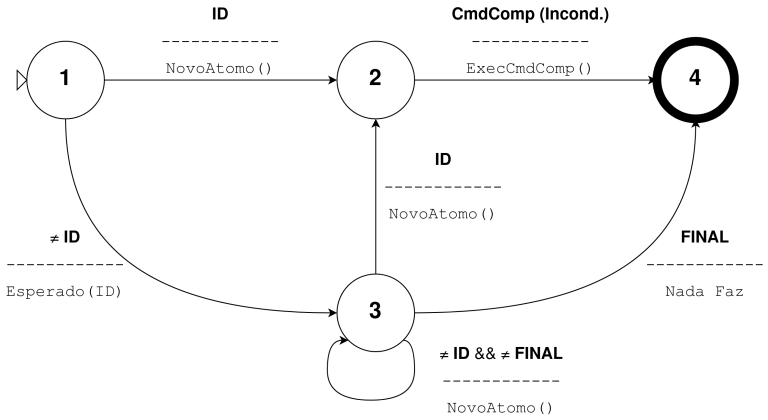

Figura 1: Diagrama de Transição para a transição: *Prog* → **ID** *CmdComp*

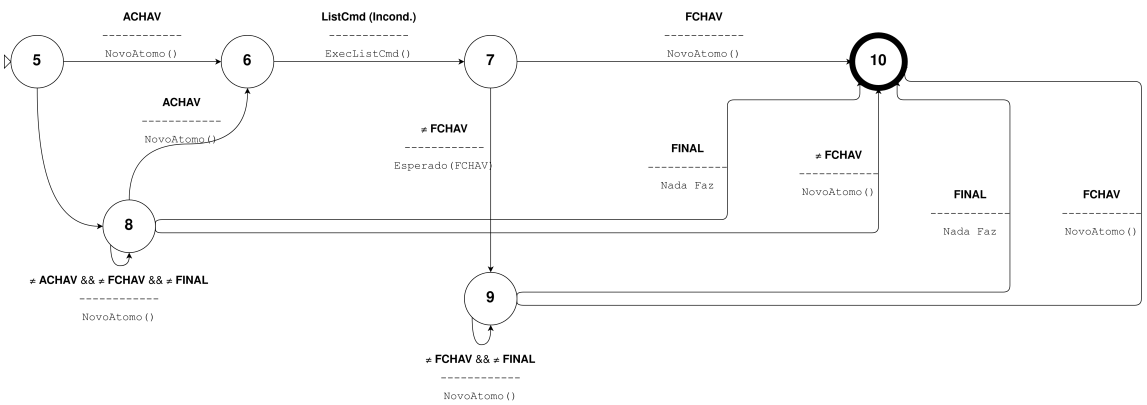

Figura 2: Diagrama de Transição para a transição: *CmdComp* → { *ListCmd* }

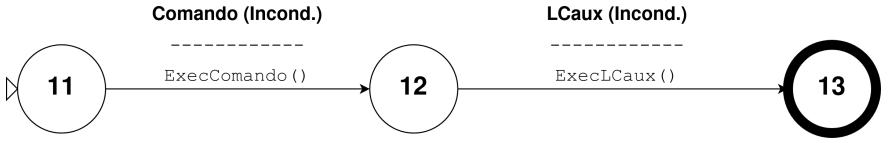

Figura 3: Diagrama de Transição para a transição: *ListCmd* → *Comando LCaux*

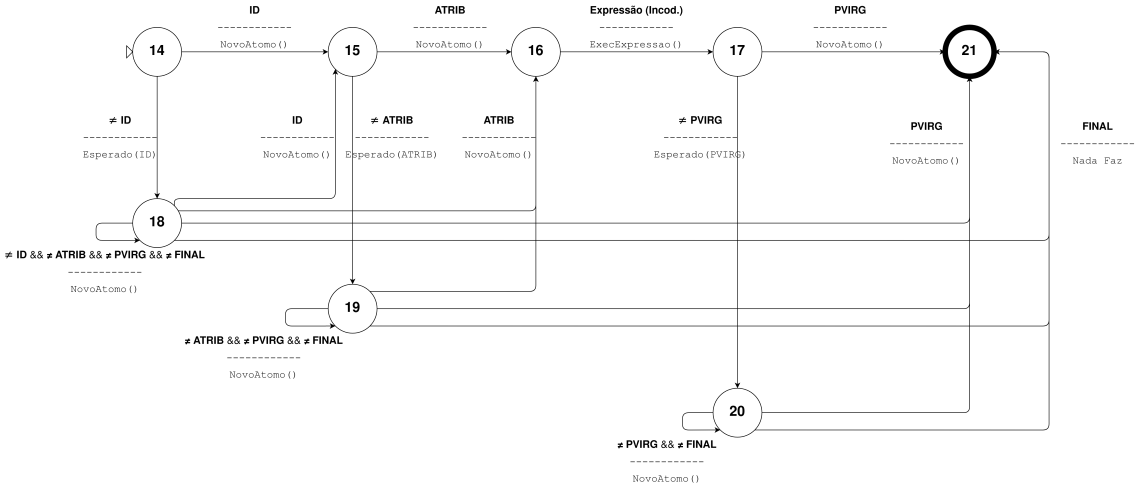

Figura 4: Diagrama de Transição para a transição: *Comando* → **ID** = *Expressão* **;**

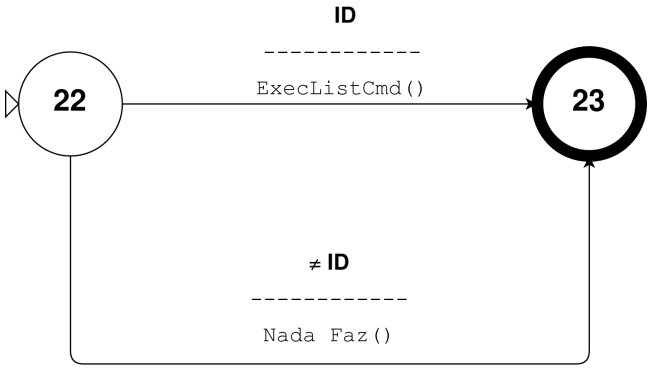

Figura 5: Diagrama de Transição para a transição: *LCaux* → ϵ | *ListCmd*

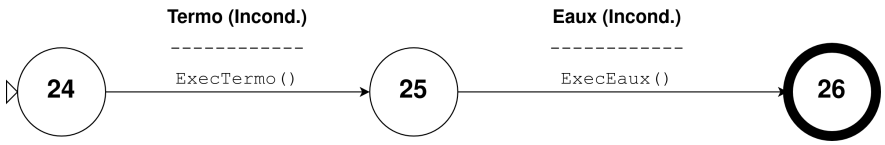

Figura 6: Diagrama de Transição para a transição: *Expressão* → *Termo Eaux*

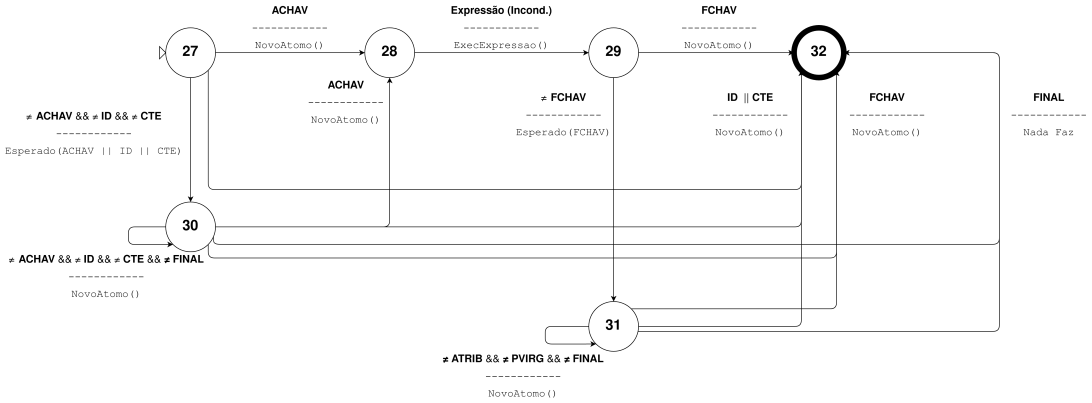

Figura 7: Diagrama de Transição para a transição: *Termo* → (*Expressão*) | **ID** | **CTE**

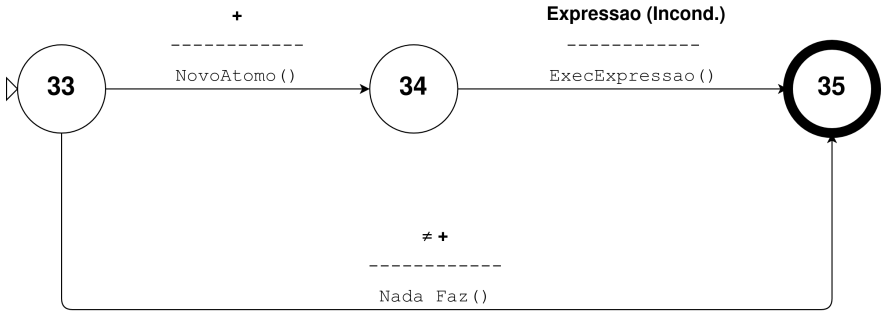

Figura 8: Diagrama de Transição para a transição: *Eaux* → ϵ | + *Expressão*

Atualização do programa

O programa implementado tendo como base os diagramas de transição apresentados teve seu funcionamento testado e validado. Porém, depois de já ter escrito esta seção do relatório, foi feita uma modificação no funcionamento do analisador, para deixar as mensagens de saída ainda mais atraentes, e acabou alterando alguns itens em relação aos diagramas de transição apresentados. O que foi alterado é que a chamada das funções `Esperado()` não ocorre diretamente na transição para o nó do digrama responsável por tratar dos erro, e sim na transições dos nós de tratamento de erro para eles mesmos (as setas que saem dele e entram nele mesmo). Dessa forma o nó de tratamento pode imprimir diversas mensagens de erro até encontrar uma transição válida. Achei que após esta modificação a execução do código ficou mais interessante.

Testando o analisador sintático

Inicialmente, vamos executar o analisador em um programa sintaticamente correto para verificar o seu funcionamento. O Código 1 contém o código válido a ser testado (arquivo `valido.dat`) e o Código 2 contém a saída obtida. Os testes foram executados em uma plataforma de processador de arquitetura x86_64 e sistema operacional Linux.

Código 1: Arquivo `valido.dat`: Código válido sintaticamente.

```
programName {
    toso = 25;
    turma = 21;
    anoInicio = 1965;
    respostaUniverso = 42;

    var1 = ( anoInicio + respostaUniverso ) + (256 + toso);
```

```
    var2 = var1 + (turma) + (anoInicio + (var1 + toso));
}
```

Código 2: Arquivo `valido.dat.log`: Saída obtida pela execução do analisador no arquivo `valido.dat`.

```
Programa livre de erro sintático!
```

Os arquivos a seguir contém pares de arquivos de entrada e saída obtido para cada execução. Em cada caso, ilustrar-se-á a identificação de diferentes tipos de erros sintáticos.

Código 3: Arquivo `inicio.dat`: Código com problema na inicialização.

```
{ 1994 programName {
    turma = 21;
}
```

Código 4: Arquivo `inicio.dat.log`: Saída para o arquivo `inicio.dat`.

```
[Error: Expected ID] Átomo: { : O programa deve inicializar com um átomo do tipo ID
[Error: Expected ID] Átomo: 1994 : O programa deve inicializar com um átomo do tipo ID
```

Código 5: Arquivo `cmdcomp.dat`: Código com problema na abertura e fechamento de chaves.

```
programName aquiDeveSerAbreChave {
    toso = 25;
    turma = 21;
```

Código 6: Arquivo `cmdcomp.dat.log`: Saída para o arquivo `cmdcomp.dat`.

```
[Error: Expected ACHAV] Átomo: aquiDeveSerAbreChave : Imediatamente após o nome do programa deve existir
uma abertura de chaves
[Error: Expected FCHAV] Átomo: <<atomo inválido>> : Término inesperado
```

Código 7: Arquivo `atribuicoes.dat`: Código com problemas diversos nas atribuições.

```
programName {
    toso 25;
    turma = 21
    anoInicio = 1965;
    respostaUniverso = 42;

    var1 = (anoInicio + respostaUniverso + toso;
}
```

Código 8: Arquivo `atribuicoes.dat.log`: Saída para o arquivo `atribuicoes.dat`.

```
[Error: Expected ATRIB] Átomo: 25 : Faltando átomo de atribuição (=)
[Error: Expected PVIRG] Átomo: anoInicio : Um comando deve encerrar com ponto-vírgula (;)
[Error: Expected PVIRG] Átomo: = : Um comando deve encerrar com ponto-vírgula (;)
[Error: Expected PVIRG] Átomo: 1965 : Um comando deve encerrar com ponto-vírgula (;)
[Error: Expected FPAR] Átomo: ; : Imediatamente ao término de uma expressão deve haver um fechamento
de parênteses
[Error: Expected FCHAV] Átomo: <<atomo inválido>> : Término inesperado
```

Nota-se que o diagrama de estados com o tratamento simples utilizado não é o suficiente para trazer mensagens de erros tão precisas. Nota-se alguns equívocos neste caso. Ainda assim, acredito que seja um excelente resultado para uma implementação inicial. Acredito que um programador com conhecimento da linguagem seria capaz de rapidamente encontrar a partir das mensagens de erro o equívoco comentido durante a programação.

Código 9: Arquivo `errosSintaticos.dat`: Erros sintáticos distintos.

```
123 456 programa 789 {
    teste = $4 + 2;
    var #= 21;
    aux = 27 + 32 @;
    00000000000
}
```

Código 10: Arquivo `errosSintaticos.dat.log`: Saída para o arquivo `errosSintaticos.dat`.

```
[Error: Expected ID] Átomo: 123 : O programa deve inicializar com um átomo do tipo ID
[Error: Expected ID] Átomo: 456 : O programa deve inicializar com um átomo do tipo ID
[Error: Expected ACHAV] Átomo: 789 : Imediatamente após o nome do programa deve existir uma abertura de
chaves
[Error: Expected APAR or CTE or ID] Átomo: $ : Termo inválido, deve ser uma expressão entre parênteses
ou um número
[Error: Expected ATRIB] Átomo: # : Faltando átomo de atribuição (=)
[Error: Expected PVIRG] Átomo: @ : Um comando deve encerrar com ponto-vírgula (;)
[Error: Expected FCHAV] Átomo: 0 : Após os comandos o programa deve ser encerrado imediatamente com um
fechamento de chaves
```