

UMA LINGUAGEM ALGORÍTMICA PARA SIMULAÇÃO DE REDES DE FILAS.

Vanessa Gomes de Oliveira, Renata Spolon Lobato. - Ciência da Computação -
Ciência da Computação - Departamento de Ciências de Computação e Estatística - Instituto
de Biociências, Letras e Ciências Exatas - Campus de São José do Rio Preto.

Este trabalho consistiu na proposta e implementação da primeira versão de uma linguagem algorítmica para simulação de redes de filas. Os comandos da LiSReF (Linguagem para Simulação de Redes de Filas) foram redigidos na língua portuguesa estruturada e foi fundamentada na forma de descrição de algoritmos de linguagens de programação, tal como as palavras PROGRAMASIMULACAO e FIMDASIMULACAO que delimitam o início e o final do programa, respectivamente. Isto foi feito para facilitar sua utilização pelo seu público alvo, composto de programadores que não possuem tanto conhecimento a respeito de simulação de redes de filas.

A LiSReF foi construída para gerar código para a biblioteca RFOO (Redes de Filas Orientada a Objetos) (Di Chiacchio, 2005), uma vez que esta possui todos os métodos necessários para a elaboração de uma simulação de redes de filas. A RFOO possui classes que fazem a construção de centros de serviço, geradores de números pseudo-aleatórios, controle do relógio da simulação e da lista de eventos futuros, que são as estruturas fundamentais para a construção de um ambiente de simulação.

Para a construção da LiSReF são necessárias as fases de *front-end* e *back-end* (Aho *et al.*, 1987) de um processo de compilação. Neste trabalho foi implementada a fase de *front-end*, que é composta pelos analisadores léxico, sintático e semântico.

O analisador léxico irá identificar e fornecer ao analisador sintático os símbolos (*tokens*) encontrados no programa fonte. O léxico analisa o arquivo fonte caracter a caracter e agrupa as construções encontradas nos chamados *tokens*. Estes *tokens* consistem nas palavras válidas para a LiSReF, que são seus identificadores, seus métodos, caracteres de pontuação e operação aritmética e as palavras reservadas (como Inteiro e Real, que denotam os dois tipos de dado numéricos da LiSReF). Vale lembrar que os identificadores da LiSReF são formados somente por letras, maiúsculas ou minúsculas.

Para o auxílio na construção deste analisador foi utilizada a ferramenta de programação *Flex* (*Fast Lexical Analyzer*) (Levine *et al.*, 1992) no ambiente UNIX. Esta ferramenta é própria para a implementação deste tipo de analisador, pois auxilia o programador na elaboração da gramática regular da linguagem a ser construída. A chamada gramática regular contém todas as regras para a elaboração dos *tokens* a serem reconhecidos pela linguagem.

Como exemplo de funcionamento da análise léxica é mostrado um trecho de código escrito na LiSReF. Este trecho mostra a forma de declaração de um dado do tipo Inteiro:

Inteiro valor;

Quando o léxico o analisar, irá identificar três tipos de *tokens*:

- Inteiro: palavra reservada;
- valor: identificador;
- ; (ponto-e-vírgula): caracter de pontuação.

O espaço em branco e o caracter de final de linha serão ignorados pelo analisador.

Além da definição da gramática, o analisador léxico também contém uma forma de identificação dos erros possíveis de serem encontrados nesta fase. Os erros que este analisador pode identificar são relativamente simples, devido à natureza do mesmo. Caracteres inválidos e formação inválida de identificadores são os erros que o léxico pode encontrar. Estes erros, caso existam, são exibidos ao usuário ao final das análises léxica e sintática.

A análise sintática verifica as construções com os símbolos advindos do analisador léxico, uma vez que estes dois analisadores trabalham conjuntamente. Para o auxílio na construção deste analisador foi utilizada a ferramenta *Bison* (Donnelly & Stallman, 2005) no ambiente UNIX. Esta ferramenta é própria para a construção deste tipo de analisador, pois auxilia o programador na elaboração da gramática livre de contexto da linguagem. Esta gramática possui regras para a formação

das sentenças válidas para a LiSReF.

A seguir é mostrado um trecho de código escrito na LiSReF para ilustrar o funcionamento deste analisador. Este trecho ilustra a forma de atribuição de um valor a um identificador.

```
valor <- 10;
```

Quando o sintático analisar este trecho, ele verificará que este é uma construção válida para a atribuição de valores. Os caracteres “<-” representam a forma de atribuição.

Nesta fase da compilação também são tratados e contados seus respectivos erros. Os erros aqui encontrados são mais complexos que os da análise léxica, pois o sintático verifica a consistência das sentenças formadas pelos *tokens* retornados pelo léxico. Como exemplo de erro nesta fase, pode citar-se a referente à construção do comando SE, equivalente ao comando *if* da linguagem C (Kernighan, 2000). A tabela a seguir mostra uma das formas de representação deste comando e uma possível construção inválida. A construção inválida é assim denominada, pois falta o *token* COMEÇO, que delimita o espaço desta estrutura. COMEÇO, FIMSE e FIMSENAO referem-se a alguns dos delimitadores de estruturas da LiSReF.

Tabela 1: Construções do comando SE.

Construção Válida	Construção Inválida
SE (expressao) COMEÇO comandos FIMSE SENÃO COMEÇO comandos FIMSENAO	SE (expressao) COMEÇO comandos FIMSE SENÃO comandos

Um outro tipo de erro muito comum nesta fase inclui a validade dos identificadores encontrados ao longo do programa fonte. Se um identificador encontrado ainda não foi declarado é gerado um erro sintático. Esta validade é verificada de acordo com suas especificações na chamada tabela de símbolos. Esta tabela é uma estrutura que também foi implementada para a elaboração da LiSReF e contém as informações relevantes a respeito de um identificador. Estas informações incluem dados como, por exemplo, nome e tipo (se Real ou Inteiro), e são incluídas na tabela assim que um novo identificador é encontrado. O início da construção da tabela de símbolos é realizado pelo analisador sintático.

A última fase da *front-end*, chamada de análise semântica verifica a compatibilidade dos tipos dos símbolos envolvidos nas construções da análise sintática. Este analisador foi construído através da linguagem C (Kernighan & Ritchie, 1987), do compilador *gcc* no ambiente UNIX. Para a sua construção foram implementadas as chamadas ações semânticas da linguagem. Estas ações, dependendo da estrutura a ser analisada, irão verificar se os atributos pertinentes à mesma estão corretos. As verificações que este analisador executa são relacionadas com os tipos de dados envolvidos em:

- Operações aritméticas;
- Operações de comparação;
- Verificação dos índices em estruturas;
- Chamadas a procedimentos da linguagem.

Para as operações aritméticas e de comparação são verificados se os dados envolvidos na operação são compatíveis entre si. A verificação dos índices analisa os índices de um identificador do tipo vetor ou matriz não foram ultrapassados ou mesmo não estão de acordo com a dimensão determinados para o mesmo. Para as chamadas a procedimentos serão verificados se os dados envolvidos são compatíveis com os especificados para o método.

Como nas outras duas fases, a análise semântica também possui uma estrutura para o tratamento

e contagem de seus erros. Uma verificação da análise semântica observa, por exemplo, se a um identificador declarado como Inteiro, não está sendo atribuído um valor do tipo Real, tal como no trecho de código a seguir.

```
Inteiro numero;  
numero <- 5.2;
```

Para testar a LiSReF, o usuário deve redigir um código escrito nesta linguagem. Ao se analisar o arquivo fonte, o compilador da LiSReF irá realizar as análises léxica, sintática e semântica.

Se o compilador na fase de análise léxica encontra o caracter @ (inválido para a LiSReF) na linha 23, por exemplo, ele reporta o erro ao usuário da seguinte forma:

ERROS LÉXICOS.

Número de erros léxicos: 1.

Caracter @ inválido na linha 23.

O mesmo vale tanto para os erros sintáticos como semânticos. O conjunto de erros encontrados para as três análises é reportado ao usuário somente ao final da análise do programa fonte. Se ao fim da análise somente são encontrados erros semânticos, os erros correspondentes a esta fase serão exibidos. Se nenhum erro for encontrado nas três análises, será reportada a seguinte mensagem ao usuário:

Fase de front-end da compilação concluída!

A primeira versão da LiSReF irá auxiliar usuários que não possuem tanto conhecimento sobre redes de filas a implementar um modelo referente às mesmas. O usuário irá redigir o código correspondente e o compilador da linguagem reportará sobre possíveis erros na análise do programa fonte.

Como continuidade do trabalho, propõe-se a implementação da fase de *back-end* da compilação, que consistirá do gerador de aplicação para a biblioteca RFOO.

Referências Bibliográficas

- (Aho *et al.*, 1987) AHO, A. V., SETHI R., ULLMAN J. D., *Compilers: Principles, Techniques and Tools*, Addison Wesley, 1987.
- (Ascencio & Campos, 2003) ASCENCIO, A. F. G., CAMPOS, E. A. V., *Fundamentos da Programação de Computadores: Algoritmos, Pascal e C/C++*, Prentice Hall, 2003.
- (Di Chiacchio, 2005) DI CHIACCHIO, R. L. C., *Biblioteca Orientada a objetos para simulação de redes de filas*, Monografia de projeto final, IBILCE - UNESP, 2005.
- (Donnelly & Stallman, 2005) DONNELLY, C., STALLMAN, R., *Bison: The Yacc-compatible parser generator*, Manual da ferramenta Bison, 2005.
- (Kernighan & Ritchie, 1987) KERNIGHAN, B. W., RITCHIE, D. M., *C: A Linguagem de Programação*, Campus, 3ª edição, 1987.
- (Levine *et al.*, 1992) LEVINE, J. R., MASON, T., BROWN, D., *Lex & Yacc*, O'Reilly, 1992.