Compilador da Linguagem T++: Relatório de Implementação

Sávio O. Camacam[1]

[1]Departamento Acadêmico de Computação –
Universidade Tecnológica Federal do Paraná (UTFPR)
Caixa Postal 271 – 87.301-899 – Campo Mourão – PR – Brazil

saviocamacam@alunos.utfpr.edu.br

Abstract. *This work describes the implementation phases of the compilation tool created during the school year in Compilers, aiming to fix the main concepts of the actual compiler construction steps. The complete work comprises 4 phases of implementation: lexical analysis, syntactic analysis, semantic analysis and, finally, code generation. Since the implementation process is incremental, the report is also done in incremental format, with the four main topics related to these steps and their respective internal stages and characteristics.*

Resumo. *Este trabalho descreve as fases de implementação da ferramenta de compilação criada durante o período letivo na disciplina de Compiladores, tendo o objetivo de fixar os principais conceitos das etapas de construção de compilador real. O trabalho completo compreende 4 fases de implementação: análise léxica, análise sintática, análise semântica e por fim, geração de código. Sendo incremental o processo de implementação, o relatório se faz também no formato incremental, com os quatro grandes tópicos relativos a estas etapas e suas respectivas etapas e características internas.*

1. Introdução

Este relatório descreve o processo de construção da ferramenta de compilação para a Linguagem T++, que possui uma extensão de funcionalidades e características, e adaptação para o português, da linguagem-exemplo TINNY, do livro-texto Compiler Construction Principles, de Kenneth Louden [Louden].

Um programa nessa linguagem tem uma estrutura relativamente simples, mas com características de línguagens como C , possuindo sequências de *statements*, possuindo declarações e chamadas de funções, tipos de dados dimensionados e operadores lógicos.

Este trabalho está organizado da forma como segue: na próxima seção é feita uma especificação mais detalhada da linguagem, sobre seus itens e dados e declarações; em seguida são descritas as palavras reservadas da linguagem que são aceitas no Português; adiante são descritos os símbolos e operadores de manipulação da linguagem; logo depois é dada uma especificação formal dos autômatos da linguagem com descrição as expressões regulares usadas para captura dos tokens, bem como os autômatos que os representam; depois é explicado sobre o processo de execução do programa, bem como é exibida uma figura de exemplo de uma saída desse processador e por fim, uma conclusão sobre esta etapa do trabalho.

2. Análise Léxica

O processo de escaneamento, ou análise lexica, é a fase do processo de compilação que a ferramenta tem a tarefa de ler o código fonte a partir de um arquivo [Louden], lendo caractere a caractere e submetendo-os às regras que geram os *Tokens* desta linguagem. Tokens são basicamente palavras em linguagem natural formado pela sequência de caracteres lida do arquivo fonte, e representa uma unidade de informação.

Uma vez que a tarefa de extração dos tokens é executada, é necessário então a aplicação dos padrões de combinação que classificam essas unidades de informação, de acordos com às classes de uma linguagem, tais como: palavras-chave, idenficadores ou constantes numéricas.

2.1. Especificação da Linguagem de Programação T++

A linguagem T++ objeto desse estudo é relativamente simples nos termos de suas funções, mas tem a complexidade inerente das questões de implementação de qualquer linguagem. Abaixo estão discriminados os símbolos, descritores, operações e opções suportadas pela linguagem.

2.1.1. Tipos de Dados

Por se tratar uma linguagem experimental para estudo de compiladores, T++ apresenta apenas os tipos de dados primitivos:

- Inteiro;
- Flutuante.

Entretanto, a linguagem permite a criação de Tipos de Vetores e Matrizes a partir desses tipos primitivos, na seguinte sintaxe, além do reconhecimento de números em notação científica:

```
inteiro : var1
flutuante : var1
var1 := 1.54334E-34
inteiro : arr1[indice1]
inteiro : arr2[indice1][indice2]
```

2.1.2. Palavras Reservadas

A linguagem atente ao parâmetro de uma linguagem discorrida no Português, o que significa o suporte à caracteres especiais dessa língua, como acentuação e "ç".

Suas palavras reservadas são: **se, então, senão, fim, repita, flutuante, retorna, até, leia, escreve, e inteiro**. A seguir, um exemplo de código em T++:

```
inteiro: n

inteiro fatorial(inteiro: n)
    inteiro: fat
    se n > 0 então {não calcula se n > 0}
        fat := 1
        repita
            fat := fat * n
            n := n - 1
        até n = 0
        retorna(fat) {retorna o valor do fatorial de n}
    senão
        retorna(0)
    fim
fim

inteiro principal()
    leia(n)
    escreva(fatorial(n))
    retorna(0)
fim
```

Figura 1. Código de exemplo em T++.

2.1.3. Símbolos e Operadores

Além do conjunto de palavras reservadas, essa linguagem também tem uma série de operadores mapeados para operações específicas:

- "*" : operador de ***multiplicação***;
- "-" : operador de ***subtração***;
- "+" : operador de soma***soma***;
- ":=" : operador de ***atribuição***;
- ":" : símbolo ***dois-pontos***;
- "<" : operador de comparação ***menor-quê***;
- ">" : operador de comparação ***maior-quê***;
- "=" : operador de ***igualdade***;
- "<=" : operador de comparação ***menor ou igual a***;
- ">=" : operador de comparação ***maior ou igual a***;
- "[" : símbolo abre ***colchetes***;
- "]" : símbolo fecha ***colchetes***;
- "(" : símbolo abre ***parênteses***;
- ")" : símbolo fecha ***parênteses***;
- "!" : operador lógico de ***negação***;
- "&&" : operador lógico ***conjunção***;
- "||" : operador lógico ***disjunção***;

2.2. Especificação Formal dos Autômatos

O sistema de varredura foi implementado com base em expressões regulares, descritas como se segue, gerando as marcas para identificação dos lexemas da linguagem:

- Palavra Reservada **t_PR = (se)|(então)|(senão)|(fim)|(repita)| (flutuante)|(retorna)|(até)|(leia)|(escreva)|(inteiro)**
  Para cada conjunto de caractere que representa uma palavra-chave é necessário criar uma regra específica que dará forma a um autômato para a aquela palavra, depois se unindo com os demais autômatos de outras regras.

- Identificador **t_ID = [a-zA-Zá-ñÁ-Ñ][a-zA-Zá-ñÁ-Ñ0-9_]**
  Regra que combina as definições de números e letras, acentuadas ou não, e underscore para identificar marcadores de identificadores da linguagem;

- Inteiro **t_INTEIRO = ?[0-9][0-9]***
  Regra genérica para captura de números inteiros;

- Flutuante **t_FLUTUANTE = ?[0-9][0-9]*[̇0-9]+([Ee][-]?[0-9]+)?**
  Regra genérica para captura de números flutuantes e em notação científica;

- Branco **t_ignore = [\t]**
  Regra para captura de símbolos de espaço e outros não imprimíveis que não fazem parte do programa;

- Símbolo **SB = (:=)|(:)|(>)|(<)|(()|())|(*)|(=)| (-)|(+)|(/)|(<=)|(>=)|([)|(])**
  Da mesma forma como ocorre com as palavras reservadas, um autômato de símbolo é composto pelos estados com transições com os caracteres específicos dos símbolos da linguagem, que se une com as demais regras formando o autômato da linguagem.

- Comentário **t_COMMENT = {[̃}]*}**
  Regra que casa abertura e fechamento de comentários reconhecido pela linguagem;

- Erro de comentário **t_ERRORCT = [{}]**
  Regra de exceção para abertura ou fechamento sem sequência de comentários;

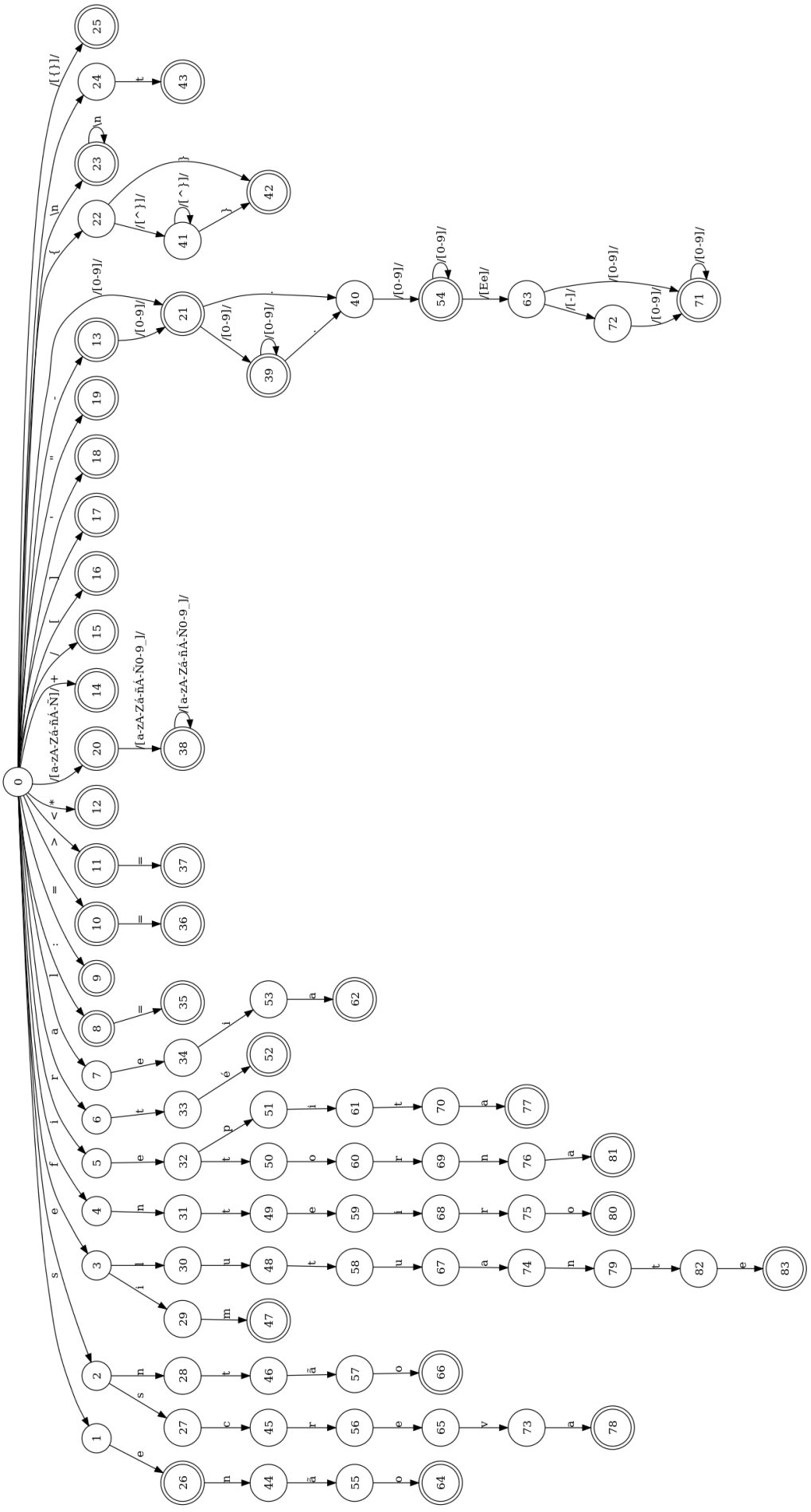

Figura 2. Autômato resultante da união das partes do analisador.

2.3. Detalhes de Implementação da Ferramenta

Considerando experiências passadas com o Java na implementação de ferramentas de compilação, para esse novo projeto foi usado o Python como linguagem-base para implementação com auxílio da biblioteca PLY.

O texto base para desenvolvimento da ferramenta pode ser encontrado em http://www.dabeaz.com/ply/ply.html [Beazley], que conta com exemplos básicos, detalhes de formulação de regras e modelo projeto.

Diferentemente da Linguagem Java, para o analisador léxico construído com o PLY é necessária a criação de um único arquivo de programação que representa uma única classe, nesse caso, chamada apenas de Lexer, que possui os métodos Test.

2.4. Resultados da Varredura

A entrada desse programa é composta de dois arquivos: o primeiro contém a descrição das regras de interpretação da linguagem em Lex; o segundo, contém o código da linguagem que deseja-se compilar.

O processo trata da leitura caractere a caractere do arquivo de entrada do programa - código fonte - como um *stream*, onde a partir dele, são buscadas combinações de regras léxicas dada a máquina de estados da linguagem que indica a criação das marcas dessa linguagem.

Como saída, é exibida no console do programa, a combinação <classe, valor> para cada marca válida da linguagem. Em situação de caractere ou sequência inválidos é apontado um erro, com especificação de caractere não identificado ou sequência inválida, como abertura e não fechamento de ***chaves*** que indica um comentário incompleto.

O projeto foi escrito usando a IDE PyCharm, e pode ser executado em Run > Run > nome_programa, sendo necessário também informar por parâmetro o nome do arquivo-fonte do programa a ser analisado.

Assumindo que o programa chegou até o caminho dos arquivos de entrada a saída será exibida dessa forma:

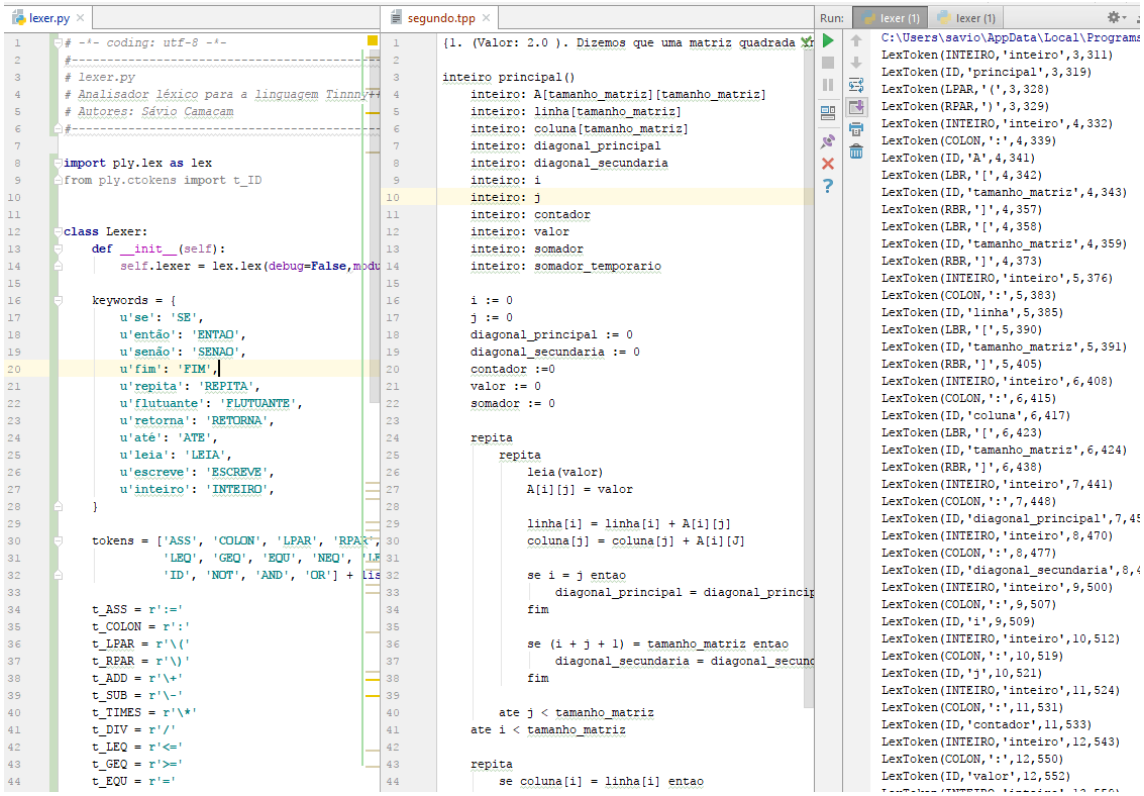

Figura 3. Situação de saída do programa.

3. Conclusão

Com a conclusão desta etapa da implementação da ferramenta de compilação para a linguagem T++, os conceitos de design de compiladores continuam a ser fixados e aprimorados com a revisão da estrutura do analisador, agora no uso da biblioteca PLY, o que mostrou ser mais simples, ao contrário do uso do java CUP que envolvia uma quantidade muito maior de classes e arquivos, que complicava o entendimento do processo de geração automática e agravava a migração do projeto para a fase seguinte, de implementação do analisar sintático.

Referências

Beazley, D. M. Ply version: 3.9.

Louden, K. C. *Compiler Construction Principles*.