ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

Intermediate Graphical Language using SDT

Snehal H. Chaflekar [1], Rupali Shinganjude [2], Shrunkhala Wankhede[3], Ashwini Lokhande[4]

Associate Professor, Department of Information Technology, PBCOE, Nagpur, India

Associate Professor, Department of Information Technology, PBCOE, Nagpur, India

Associate Professor, Department of Computer Science and Engineering, PBCOE, Nagpur, India

Associate Professor, Department of Information Technology, PBCOE, Nagpur, India

ABSTRACT: IUPAC Nomenclature is an international standard for naming of organic compounds. This paper describes how IUPAC name of organic compounds can be translated into Intermediate Graphical Language which consists of graphical entities. These graphical entities can be used to generate the two – dimensional structure of organic compounds. This paper describes how this translation can be achieved by generating the front end of the compiler. The various compiler construction tools are also described in this paper.

KEYWORDS: compiler construction tools, Intermediate Graphical Language, IUPAC, organic compounds, two – dimensional structure

I. INTRODUCTION

In 1957, the International Union of Pure and Applied Chemistry (IUPAC) standardized a nomenclature for organic compounds which has been widely accepted among chemists. This nomenclature provides the organic chemist with a basis for identification of numerous compounds. A strict nomenclature system is obviously required to eliminate ambiguities in characterizing a chemical formula. The IUPAC system satisfies this requirement.
Syntax-directed translation can be advantageously used to transform formulas in the IUPAC nomenclature into Intermediate Graphical Language that can be further used to transform it into a standard two-dimensional representation. The purpose of this paper is to describe how this translation can be accomplished.

II. TRANSLATION

According to IUPAC naming rules, one can define compound names such as:

(1) 3-CHLORO-3-AMINO-2-PENT-4-ENONE
(2) 2,4-DIBROMO-CYCLO HEXANE
(3) 3-HEPTYNE

The above strings (implicitly) contain the following information necessary for the translation:

(a) the length of the carbon chain (e.g. PENT indicating a carbon chain of length 5 in (1), HEX meaning a chain length of 6 in (2));
(b) one or more unit prefixes, indicating the number of the carbon to which any functional group is attached, followed by the functional group itself (e.g. a chlorine atom and amino group are attached to carbon 3 in (1));
(c) the carbon number and type of unsaturation, which composes the primary suffix (e.g. a double bond at carbon 4 in (1); a triple bond at carbon 3 in (3));
(d) a secondary suffix again providing points of attachment of functional groups (e.g. carbon 2 is a carbonyl in (1));
(e) singular or multiple functionality (e.g. 2 bromine atoms are present in (2)).

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

The translation process involves the execution of semantic actions once the elements of a rule are recognized while parsing.

III. SYNTAX-DIRECT TRANSLATION

The process of Syntax-Directed Translation is a context-free grammar in which attributes are associated with the grammar symbols, and semantic actions, enclosed within braces ({}), are inserted in the right sides of the productions. Syntax – Directed Translation can be achieved by generating the front end of the compiler. The front end of the compiler consists of three phases which are described below:

A. Lexical Analysis Phase

In Lexical Analysis Phase, the *Lexical Analyzer* or *Scanner*, separates characters of the source language into groups that logically belong together; these groups are called *tokens.*

The compiler construction tool JFlex is used to generate the Lexical Analyzer.

The main design goals of JFlex are:

- Full unicode support
- Fast generated scanners
- Fast scanner generation
- Convenient specification syntax
- Platform independence
- JLex compatibility

LEXICAL SPECIFICATIONS:

A Lexical Specification File for JFlex consists of three parts divided by a single line starting with %%:
UserCode
%%
Options and declarations
%%
Lexical rules
In all parts of the specification comments of the form /* comment text */ and the Java style end of line comments starting with // are permitted. JFlex comments do nest - so the number of /* and */ should be balanced.

User Code:

The first part contains user code that is copied verbatim into the beginning of the source file of the generated lexer before the scanner class is declared. This is the place to put package declarations and import statements. It is possible, but not considered as good Java programming style to put own helper class (such as token classes) in this section. They should get their own .java file instead.

Options and Declarations:

The second part of the lexical specification contains options to customize your generated lexer (JFlex directives and Java code to include in different parts of the lexer), declarations of lexical states and macro definitions for use in the third section “Lexical rules” of the lexical specification file.
Each JFlex directive must be situated at the beginning of a line and starts with the % character. Directives that have one or more parameters are described as follows:
%class "classname"

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

means that you start a line with %class followed by a space followed by the name of the class for the generated scanner (the double quotes are *not* to be entered).

Lexical rules:

The "lexical rules" section of an JFlex specification contains a set of regular expressions and actions (Java code) that are executed when the scanner matches the associated regular expression.

B. Syntax Analysis Phase

The output of the Lexical Analyzer is a stream of tokens, which is passed to the next phase, the Syntax Analysis Phase. In Syntax Analysis Phase, the *Syntax Analyzer* groups tokens together into syntactic structures.

Syntax – Analyzer can be constructed using CUP Parser Generator. CUP is a system for generating LALR Parsers from simple specifications.

The specification for CUP consists of:

- package and import specifications,
- user code components,
- symbol (terminal and non-terminal) lists,
- precedence declarations, and
- the grammar.

To produce a parser from this specification CUP generator is used. If this specification is stored in a file parser.cup, then CUP is invoked using a command like:

java java_cup.Main < parser.cup

In this case, the system will produce two Java source files containing parts of the generated parser: sym.java and parser.java. These two files contain declarations for the classes sym and parser. The sym class contains a series of constant declarations, one for each terminal symbol. This is typically used by the scanner to refer to symbols. The parser class implements the parser itself.

The specification above, while constructing a full parser, does not perform any semantic actionS; it will only indicate success or failure of a parse. To calculate and print values of each expression, Java code must be embedded within the parser to carry out actions at various points. In CUP, actions are contained in *code strings* which are surrounded by delimiters of the form {: and :}.

Semantic Action is associated with each grammar rule in CUP Specification to achieve Syntax – Directed Translation.

REFERENCES

[1] Dr. O. P. Tandon, Dr. A. K. Virmani, "Organic Chemistry"
[2] Melvin J. Astle, J. Reid Shelton, "Organic Chemistry"
[3]http://en.wikipedia.org/wiki/IUPAC_nomenclature_of_organic_chemistry
[4] http://www.iupac.org
[5] http://catalog.compilertools.net/java.html
[6] https://class.coursera.org/automata/lecture/preview
[7] Gerwin Klein, "JFlex User's Manual"
[8] Alfred V. Aho, Jeffrey D. Ullman, "Principles of Compiler Design