Regular Expressions of Conditions for Processing Language Modelling

POPA MARIN EMIL
Department of Computer Science
Lucian Blaga University of Sibiu, Faculty of Sciences
Sibiu, str. Dr. Ion Ratiu, no. 5-7, zip code 550012
ROMANIA

Abstract: - The generalized regular expressions were introduced by I. Gruska [2]. We can find similar results in [1], [3], [4] and [7]. Using the given by results T. Rus [6], we define in this paper the generalized regular expressions of conditions.

Key-Words: - generalized regular expression, generalized regular expression of conditions, language specification, lexicon specification, lexeme, universal lexeme.

1 Introduction

Let V be an alphabet, $L_1, L_2 \in P(V^*)$ two languages and $a \in V$. We consider a binary operation between these languages.

Definition 1. The operation of a-substitution Ψ_a attaches to the pair of languages (L_1, L_2), the following language:

$$L_1 \Psi_a L_2 = \{w \mid w = w_1 q_1 w_2 q_2 \ldots w_k q_k w_{k+1}, k \geq 0,$$
$$w_1 a w_2 a \ldots w_k a w_{k+1} \in L_1,$$

a doesn't appear in the word:

$$w_1 w_2 \ldots w_{k+1} \text{ and } q_j \in L_2, 1 \leq j \geq k\}$$

Note 1.

1) For every $a \in V$, the operation Ψ_a is associative, therefore

$$L_1 \Psi_a (L_2 \Psi_a L_3) = (L_1 \Psi_a L_2) \Psi_a L_3,$$

for every language $L_1, L_2, L_3 \in P(V^*)$.

2) We also have the next associate properties with the operators like Ψ_a:

a) $(L_3 \Psi_a L_4) \Psi_a L_2 = (L_3 \Psi_a L_2) \cup (L_4 \Psi_a L_2)$.

b) $(L_2 L_4) \Psi_a L_2 = (L_3 \Psi_a L_2)(L_4 \Psi_a L_2)$.

3) We also note the language $L \Psi_a L \Psi_a \ldots \Psi_a L$ also with $\Psi_a (L^j)$. Particularly we can note

$$\Psi_a (L^1) = L.$$

Definition 2. The closure through a-substitution of language L is the language L^{+a} that contains words w satisfying the next conditions:

1) $\exists j \geq 1$ so that $w \in \Psi_a (L^j)$;

2) a is not an infix of w.

Note 2. We can write:

$$1)\ L^{+a} = \left(\bigcup_{j \geq 1} \Psi_a (L^j) \right) \cap (V \setminus \{a\})^*$$

Next, we shall consider the formal system of generalized regular expressions.

Let $V = \{a_1, a_2, \ldots a_n\}$.

Definition 3. The symbols list of a formal system of generalized regular expressions is the set of symbols

$$\{\varepsilon, a_1, a_2, \ldots, a_n, |, \cdot, (,), P_{a_1}, P_{a_2}, \ldots, P_{a_n}\}.$$

The atomic formulas of this formal system are the following strings of symbols:

1) $\alpha \mid \beta$

2) $\alpha \cdot \beta$

3) $P_{a_i}(\alpha)$, $1 \leq j \leq n$,

where α and β are in the first $n + 1$ symbols.

Correct constructed formulas are:

1) $\varepsilon, a_1, a_2, \ldots, a_n$;

2) atomic formulas;

3) the strings of symbols:

$$\gamma \mid \delta, \gamma \cdot \delta, P_{a_1}(\gamma), \ldots, P_{a_n}(\gamma),$$

where γ and δ are correct constructed formulas.

These correct constructed formulas will be named forward *generalized regular expression*.

We will note L_{gre} the language of generalized regular expression,

$$L_{gre} \subseteq \{\varepsilon, a_1, a_2, \ldots, a_n, |, \cdot, (,), P_{a_1}, P_{a_2}, \ldots, P_{a_n}\}^*$$

If α is a generalized regular expression, the language $L(\alpha)$ is the language described by α.

Definition 4. The value application:

$$L : L_{gre} \to P(\{a_1, a_2, \ldots, a_n\}^*)$$

is expressed by:

1) $L(\varepsilon) = \varnothing$
2) $L(a_j) = \{a_j\}$, $1 \le j \le n$
3) $L(\alpha \mid \beta) = L(\alpha) \cup L(\beta)$
4) $L(\alpha \cdot \beta) = L(\alpha)L(\beta)$
5) $L(P_{a_j}(\alpha)) = L(\alpha)^{a_j}$, $1 \le j \le n$.

The next theorem is well known:

Theorem. *Every language described by a generalized regular expression is a language of type 2 and vice versa, every language of type 2 is described by a generalized regular expression.*

We shall next show some results obtained and presented by T. Rus [5].

2 First Level Lexicon Specification

A lexical entity in a programming language is the lowest level class of constructs in the alphabet of the language that has meaning within the language. For example, individual letters or digits generally do not have a specified meaning, but the class of identifiers does. Our approach is to use generalized regular expressions over finite sets of properties called conditions rather than over conventional alphabets.

The primitive lexical entities considered in this paper are constructed from a "universal" character set. Lexical items constructed from characters in this set may have a universal meaning such as number of word. The common property of all these classes of universal lexical entities is that elements of a class can be distinguished from the elements of other classes both syntactically and semantically by their syntactic form. In addition, the elements within a class can be distinguished by examining properties of their components. The lexical entities chosen as building blocks for the specification of a language lexicon are called universal lexemes and form the first level lexicon.

The lexemes that seem to be universally used by all programming languages are: letter sequences (identifiers), digit sequences (numbers), white spaces, unprintable characters, and other characters (punctuation, operators, separators, etc.). These universal lexical entities can be specified by the following generalized regular expressions over the character set:

I: identifiers, defined by the expressions
$L(I) = LetterLetter^*$,
N: numbers, defined by the expression
$L(N) = DigitDigit^*$,
W: white spaces and tabs,
U: unprintable characters (such as newline),
O: other characters(punctuation, separators,etc).

These classes of constructs are universal across nearly all programming languages, in the sense that they are used as building blocks of the actual lexical constructs found in the programming language.

In order to use these lexical items as fundamental entities in the construction of the lexicon of a programming language, we characterize them by the attributes Token which designates the class, i.e., $Token \in \{I, N, W, U, O\}$, *Lex*, which is the actual string of characters identifying the entity of a class, and *Len*, which is the number of characters making up a lexeme. Other attributes can be easily added and are obviously necessary, but will not be described here. The symbols Token, Lex, Len are further used as metavariables whose values are the respective properties of the universal lexemes. One tuple of attributes: < *Token, Lex, Len* > is called a universal lexeme.

3 Second Level Lexicon Specification

The second level lexicon is specified by generalized regular expressions of conditions over the first level entities, which show how to combine the independent language of the first level lexemes to obtain valid lexical constructs of a particular language.

3.1. Conditions

Conditions are properties of the universal lexemes expressible in terms of the fundamental attributes Token, Lex, Len that characterize them. To formally define the concept of a condition we observe that these attributes refer to three fundamental types of data: set, string and integer. Hence, we need to define set operations, string operations and integer operations on the universal lexeme. These operations, in turn, will allow us to construct the lexicon of actual programming languages as generalized regular expressions over expressions representing properties of universal lexemes. To express such properties, we will develop a language of fundamental properties of universal lexemes where data are tuples <*Token, Lex, Len*> and operations include relations on sets, strings and integers, and logical operators, such as $\wedge$, $\vee$ and $\neg$.

The Token attribute of an universal lexeme is expressed by set membership,

$$Token \in \{I, N, W, U, O\}.$$

Therefore, the set membership predicate must be supported as a fundamental operation. However, since the number of token types is finite the membership predicate can be expressed by the equality and logical-or operators, that is :

$$(Token = I) \vee (Token = N) \vee (Token = W) \vee (Token = U) \vee (Token = O).$$

The Lex attribute of a universal lexeme is a string. Hence, operations on strings must be supported. We allow fundamental operations of the form Lex rel string where $rel \in \{<, \leq, =, >, \geq\}$ and string is a constant of type string. However the interpretation of the relation <, ≤, =, >, and ≥, depends upon the Token attribute. That is, if *Token = N* then these are relations with numbers and if *Token* ≠ *N* then these are lexicographic relations. The Len attribute of a lexeme is an integer. Therefore, usual integer relations <, ≤, =, >, and ≥, also need to be supported. In addition to the operations specified above, one also needs operations on the component characters of a universal lexeme. This is because the definitions of generalized regular expressions accept characters of the language alphabet as lexical entities and the character sets that are used to compose lexemes of concrete programming languages are not necessarily disjoint. These operations are:

- *LexChar*(*i*) is the i^{th} character of the lexeme *Lex*.
- *LexChar*(*i*) *relC* where $rel \in \{<, =, >, \leq, \geq\}$ checks if the alphabetic relation *rel* holds between the i^{th} character of *Lex* and the constant character *C*.
- *LexChar*(*i*)*in*$\{C_1, C_2, \ldots, C_n\}$ checks if the i^{th} character of the lexeme *Lex* is in the set of characters $\{C_1, C_2, \ldots, C_n\}$. If this set is ordered and continuous over the character range then this operations can be expressed by

  $$LexChar(i)in[C_1, \ldots, C_n].$$

- *LexChar*(*i, j*)*in*$\{C_1, C_2, \ldots, C_n\}$ checks if the characters in the position *i* through *j* of the lexeme *Lex* are in the set $\{C_1, C_2, \ldots, C_n\}$. Again, if the set $\{C_1, C_2, \ldots, C_n\}$ is ordered then this can also be expressed by *Lex* (*i, j*)*in*$[C_1, \ldots, C_n]$.

Definition 5. The conditions are recursively defined by the following rules:

- A condition is a property of the attributes of a universal lexeme expressible in the language of fundamental properties defined above.
- A condition is a logical expression on conditions constructed with the operator *or*, *and* and *not*.

Note 3. Since the meaning of fields and operations varies by token, we require that the first relation occurring in a condition specifies the *Token* attribute. We call this relation the class specifier. All following relations in that condition (each of which may then specify either the *Lex*, *Len*, or other attributes), will refer to the same universal lexeme as the class specifier. With this in mind, the term condition will be used to refer to an expression that includes the class specifier and the term conditional property or property will refer to the subexpressions of a condition that does not include the class specifier. Thus, the condition :

$$Token = I \text{ and } (Len > 3 \text{ or } Lex = \text{“}aa\text{”})$$

specifies an identifier which has the property that its length is greather than three or its lexeme is the string “*aa*”.

3.2. Generalized Regular Expressions of Conditions

Let’s consider the alphabet $A = \{I, N, W, U, O, \varepsilon\}$ and the family of languages:

$$L(A) = \{ L(I), L(N), L(W), L(U), L(O), \varepsilon\}$$

where

ε is the symbol that denotes the empty string.

Definition 6. The generalized regular expressions of conditions and the language specified by them are constructed from conditions by the usual rules:

- ∅ is a generalized regular expression of conditions and the language specified by it is ∅.
- Any valid condition *c* whose token

  $$T_c \in \{I, N, W, U, O\}$$

  is a generalized regular expression of conditions and the language specified by it is

  $$L(c) = \{x \in L(T_c) \mid c(x) = true\}.$$

- If e_1 and e_2 are generalized regular expression of conditions then $e_1 \mid e_2$ (where | denotes the choice operation) is a generalized regular expression of conditions and the language specified by $e_1 \mid e_2$ is $L(e_1) \cup L(e_2)$.
- If e_1 and e_2 are generalized regular expressions of conditions then $e_1 \circ e_2$ (where ∘ denotes the operation of concatenation) is a generalized regular expression of conditions and the language specified by $e_1 \circ e_2$ is $L(e_1)L(e_2)$.
- If *e* is a generalized regular expression of conditions then $(e)^{+c_1}$, $(e)^{+c_2}$,…, $(e)^{+c_p}$ are generalized regular expressions of conditions and the language specified by $(e)^{+c_k}$, $1 \leq k \leq p$ are:

  $$L((e)^{+c_k}) = \left(\bigcup_{j \geq 1} \Psi_{c_k} L(e)^j\right) \bigcap (C \setminus \{c_k\})^*$$

where

$$C = \{c_1, c_2, \ldots, c_p\}$$

is the set of conditions used to construct *e*.

References:

[1] S. Greibach, *Full AFL’s and nested iterated substitution*, IEEE Conf. Record 10, Ann.Symp. Switching Automata Theory, 1969, pp. 222-230.

[2] I. Gruska, *A characterization of context-free languages*, J. Comput. System Sci., 1971, no. 5, pp. 353-364.
[3] J. Kral, *A modification of a substitution theorem and some necessary and sufficient conditions for sets to be context free*, Math. Systems Theory, 1970, no. 4, pp. 129-139.
[4] I. McWhirter, *Substitution expressions*, J. Comput. System Sci., 1971, no. 5, pp. 629-637.
[5] T. Rus, *A language Independent Scanner generator*, available at http://cs.uiowa.edu/rus/, 1999.
[6] T. Rus and T. Halverson, *Algebraic tools for languages processing*, Computer Languages 20, 1994, no. 4.
[7] M. Yntema, *Cap expressions for context free languages*, Information and Control, 1971, no. 18, pp. 311-318.