VFAST Transactions on Software Engineering
http://vfast.org/journals/index.php/VTSE@ 2013 ISSN(e): 2309-3978;ISSN(p): 2411-6246
Volume 1, Number 1, January-December, 2013

pp. 10-18

VERIFICATION OF IDENTIFIER AND RESERVED WORD OF LEXICAL ANALYZER USING Z NOTATION

ZAHEER AHMAD , MUHAMMAD RIZWAN BABAR AND F AHMAD

Faculty of Information Technology, University of Central Punjab Lahore, Pakistan
Email: {zaheer190@gmail.com, rizwanbabar1984@gmail.com}

Revised June 18, 2013

ABSTRACT. *In this paper, we formalize the lexical analysis into a Z specification for verifying it in Z tool. Lexical analyzer is described as automata, further, we write the specification of it using an integration of automata and Z. This gives formal approach which can leads to its correctness. In this paper, we present the development and verification of a generic and simple lexical analyzer which is obtained from the integration of automata and Z-notations.*

Keywords: Lexical analyzer; Formal Methods, Z-Notations.

1. **Introduction.** In the development of high quality software, formal methods and testing are used which are the main approaches [1]. The compiler must generate the correct object code because it is like heart in software problems [2]. A compiler can be described formally as C: SL->TL, where SL is the source language and TL is the target language. For verification of the compiler, we must take care of every part it. As we know lexical analysis is the first part of the compiler and its correctness is based on ases. Lexical analysis takes input as source code and produces tokens from it. These tokens must be valid because tokens are input of syntax analysis which checks the grammatical parts of program. In this paper, we formalize the lexical analysis into Z Notation for giving its correctness proof and it is also an application of integration of formal methods and automata. Z is a formal methods technique for checking the models. Formal methods are mathematical techniques for specification, verification and development of software and hardware system [5]. Formal methods can detect errors in the early stages [3]. Formal methods are applicable in many real time applications such as development of safety critical system, complex system and security critical system. Compiler correctness is not a safety critical application but if we want the error free compiler then we must take care of its backend as well as front end. The formal methods have powerful feature such as Schema calculus and mathematical logic which makes more suitable to the scenario we are working on [4]. There exist some work of interest, for example, In [7], it is worked out on the development and verification of the lexical analyzer. In [8], specification of compiler front end is written. In [9], the writer presents the development and formal verification of a compiler backend from C minor to PowerPC assembly code. In [10], the writer presents the verified compiler for the idealized assembly language from a small functional language. In [11], it is worked out on correctness proof of a substantial fragment of C0-to-DLX compiler. Gerhard Goos, describes the way to get the correct compiler for real programming languages and used abstract state machine for formalizing [2]. Thompson presents an implementation of regular expressions and finite automata in Miranda [12].

Finite automata have different implementations therefore if we give a mathematical analysis and describe formal specifications of it before implementing them then correctness of the model can be argued [6]. There are some methods for verifying the compiler. First we assure that source code is correct and after verifying it we compile the source code, we will get object code and we again assure that object code is correct, after verifying it we checked both source code and object code are semantically equivalent [13]. There are four necessary steps in the verification of compiler i.e. the semantics of source code, semantics of object code, a specification of the compiler itself and a proof that the compiler is meant preserving [14].

2. Formal modeling of Lexical Analyzer. Lexical analysis consists of four parts i.e. Identifier, Number, Punctuation marks and reserved words. First we describe the state diagram of Identifier then the formal specification of described in Z-eves. Following is the state diagram of the Identifier

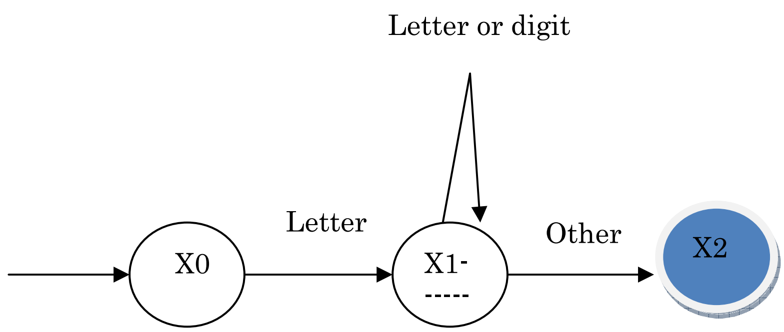

Fig: 2.1 Automata of Recognizing the

$Q ::= X0 \mid X1 \mid X2 \mid X3 \mid X4 \mid X5 \mid X21 \mid X6 \mid X7 \mid X8 \mid X9 \mid X10$
$\mid X11 \mid X12 \mid X13 \mid X14 \mid X15 \mid X16 \mid X17 \mid X18 \mid X19 \mid X20$
$\mid X22 \mid X23 \mid X24 \mid X25 \mid X26 \mid X27 \mid X28 \mid X29 \mid X30 \mid X31$
$\mid X32 \mid X33 \mid X34 \mid X35 \mid X36 \mid X37 \mid X38 \mid X39 \mid X40 \mid X41$

$A ::= a \mid b \mid c \mid d \mid e \mid f \mid g \mid h \mid i \mid j \mid k \mid l \mid m$
$\mid n \mid o \mid p \mid q \mid r \mid s \mid t \mid u \mid v \mid w \mid x \mid y \mid z \mid ea$

$Op ::= lessthan \mid greaterthan \mid equalto \mid add \mid increment \mid mul \mid sub$

Q is the set of states, A contains the alphabets.

$alpha : \mathbb{P}\ A$

$true$

The alpha is set of all possible alphabets which are given for moving one state to another and *Op* contains the operators that are used in specification.

$alpha : \mathbb{P}\ A$
$operators : \mathbb{P}\ Op$

$true$

Digit1 is set of all digits which are less than nine because the compiler will read only one operator at a time and Operators are all possible operators which may be logical operators, relational operators and arithmetic operators.

$$
\begin{array}{|l}
digit1 : \mathbb{P}\,\mathbb{Z} \\
\hline
\forall d : digit1 \bullet d \geq 0 \land d \leq 9
\end{array}
$$

Digit1 is set of all digits which are less than nine because the compiler will read only one operator at a time.

$$
\begin{aligned}
Pun ::= \; & singleqstart \\
| \; & doubleqstart \\
| \; & singleqend \\
| \; & doubleqend \\
| \; & semicolon \\
| \; & slash \\
| \; & star
\end{aligned}
$$

Pun contains the punctuations symbols that are used in specification.

$$
\begin{array}{|l}
\underline{\quad LexicalAnalyzer \quad} \\
states : \mathbb{P}\, Q \\
trans : Q \times (digit1 \times A) \rightarrow Q \\
trans1 : Q \times Op \rightarrow Q \\
trans2 : Q \times digit1 \rightarrow Q \\
trans3 : Q \times Pun \rightarrow Q \\
trans4 : Q \times A \rightarrow Q \\
X0 : Q \\
finals : \mathbb{P}\, Q \\
\hline
\# states \neq 0 \\
finals \subseteq states \\
\forall q0 : Q; a : A; d : digit1 \mid q0 \in states \land a \in alpha \land d \in digit1 \\
\quad \bullet \exists q1 : Q \mid q1 \in states \bullet trans(q0, (d, a)) = q1 \\
\forall q0 : Q; op : Op \mid q0 \in states \land op \in operators \\
\quad \bullet \exists q1 : Q \mid q1 \in states \bullet trans1(q0, op) = q1 \\
\forall q0 : Q; d : digit1 \mid q0 \in states \land d \in digit1 \\
\quad \bullet \exists q1 : Q \mid q1 \in states \bullet trans2(q0, d) = q1 \\
\forall q0 : Q; p : Pun \mid q0 \in states \bullet \exists q1 : Q \mid q1 \in states \bullet trans3(q0, p) = q1 \\
\forall q0 : Q; a : A \mid q0 \in states \land a \in alpha \\
\quad \bullet \exists q1 : Q \mid q1 \in states \bullet trans4(q0, a) = q1 \\
\hline
\end{array}
$$

Invariants:

a) The set of states *states* is a nonempty set.

b) The set of final states *finals* is a subset of *states*.

c) For each input alphabet *a* and states *q0* and *q1* an output alphabet, *a* belongs to alpha, *d* belongs to *digit1*, *trans (q0, (d, a))* where transaction function acting on *q* and order pair (*d,a)* and gives a next state.

d) For each input operator *op* and states *q0* and *q1* an output alphabet, *op* belongs to *Operators*, *trans (q0, op)* where transaction function acting on *q0* and operator *op* and gives a next state.

e) For each input punctuation symbol *p* and states *q0* and *q1* an output alphabet, *p* belongs to *Pun*, *trans (q0, p)* where transaction function acting on *q0* and punctuations symbol *p* and gives a next state.

f) For each input alphabet a and states *q0* and *q1* an output alphabet, a belongs to *alpha*, *trans (q0, a)* where transaction function acting on *q0* and alphabet *a* and gives a next state.

We introduced variable states to define the set of states of the Lexical Analysis. Each element q in set *states* is of type Q therefore *states* is a type of power set of Q.
To describe the sets of input alphabet we describe the alpha as Power set of Q. The transition functions *trans* of type $Q \times (digit1 \times A) \rightarrow Q$ and $trans1: Q \times Op \rightarrow Q$
is introduced to describe the transitions of the Identifier and Operator for each input. $trans(q0, (d, a))$ where $q0$ is a state and order pair is describe that alphabet may be digit or alphabets and there must be a unique output $q1$ of type power set of Q.
The set of initial states $X0$ is of type power set of Q and the set of final states finals is of type PQ. The transition functions *trans* of type $trans3(q0, p) = q1$ and
$trans4(q0, a) = q1$ is introduced to describe the transitions of the punctuation and reserved word for each input.

$$
\begin{array}{|l}
\textit{IDTrans} \\
\hline
\Xi LexicalAnalyzer \\
ed: digit1 \\
\hline
(\exists q0: Q; a: A; d: digit1 \\
\quad \mid q0 \in states \land q0 = X0 \land a \in alpha \land d \in digit1 \land d = ed \\
\quad \bullet (\exists q1: Q \mid q1 \in states \bullet trans(q0, (d, a)) = q1)) \\
\land (\exists q1: Q; a: A; d: digit1 \mid q1 \in states \land q1 = X1 \land a \in alpha \land d \in digit1 \\
\qquad \bullet (\exists q2: Q \mid q2 \in states \land a = ea \bullet trans(q1, (d, a)) = q2)) \\
\lor (\exists q1: Q; a: A; d: digit1 \mid q1 \in states \land d = ed \land q1 = X1 \\
\qquad \bullet (\exists q2: Q \mid q2 \in states \land q2 = X2 \bullet trans(q1, (d, a)) = q2)) \\
\hline
\end{array}
$$

Invariants:

a) There exists a alphabet *a* and states *q0* and *q1* an output alphabet, *a* belongs to alpha and *d* is empty digit denoted by *ed*. trans (q0, (d, a)) = q1) where transaction function acting on q and order pair *(d,a)* and give a next state *q1* that is equal to *X1.*
b) At *X1*, it can read both alphabet and digit but remains the same state *X1*. there is an *or* operator is used in specification because it can read alphabet or digit.
Ed stands for empty digit, it transaction reads the alphabet then it reads ed. *ea*
Stands for empty alphabet when the transition function reads the digit then alpha bet will be empty. Scanner will read only one character at one time.

lessthan, *lessthan equal to* and *not equal to*.

OpTrans

$$
\begin{array}{l}
\Xi LexicalAnalyzer \\
\hline
\exists q0: Q;\ op: Op \mid q0 \in states \wedge op \in operators \wedge op = lessthan \wedge q0 = X0 \\
\quad \bullet\ \exists q1: Q \mid q1 \in states \wedge q1 = X21 \bullet trans1\,(q0, op) = q1 \\
\exists q0: Q;\ op: Op \mid q0 \in states \wedge op \in operators \wedge op = equalto \wedge q0 = X21 \\
\quad \bullet\ \exists q1: Q \mid q1 \in states \wedge q1 = X23 \bullet trans1\,(q0, op) = q1 \\
\exists q0: Q;\ op: Op \mid q0 \in states \wedge op \in operators \wedge op = greaterthan \wedge q0 = X21 \\
\quad \bullet\ \exists q1: Q \mid q1 \in states \wedge q1 = X24 \bullet trans1\,(q0, op) = q1 \\
\exists q0: Q;\ op: Op \mid q0 \in states \wedge op \in operators \wedge op = equalto \wedge q0 = X0 \\
\quad \bullet\ \exists q1: Q \mid q1 \in states \wedge q1 = X27 \wedge q1 \in finals \bullet trans1\,(q0, op) = q1 \\
\exists q0: Q;\ op: Op \mid q0 \in states \wedge op \in operators \wedge op = equalto \wedge q0 = X27 \\
\quad \bullet\ \exists q1: Q \mid q1 \in states \wedge q1 = X29 \wedge q1 \in finals \bullet trans1\,(q0, op) = q1
\end{array}
$$

In operation transition operation, first we read the lexical analysis.
Invariants:

a) There exits an operator *op* and states *q0* and *q1* an output alphabet, *op* belongs to *Op* and *q0* belongs to state and *q0* is equal to *X21* and operator is equal to *lessthan*. *trans1 (q0, op) = q1* where transaction function acting on q0 and operator op and give a next state q1 that is equal to X23. After reading the *less than* operator, the lexical analyzer can have more than one path. It can read again *less than* or *equal to* or *other*. If it reads the other, then state will not change and scanner recognize that *less than* operator is found.

ArithmaticOpTrans

$$
\begin{array}{l}
\Xi LexicalAnalyzer \\
\hline
\exists q0: Q;\ op: Op \mid q0 \in states \wedge op \in operators \wedge op = add \wedge q0 = X0 \\
\quad \bullet\ \exists q1: Q \mid q1 \in states \wedge q1 = X17 \bullet trans1\,(q0, op) = q1 \\
\exists q0: Q;\ op: Op \mid q0 \in states \wedge op \in operators \wedge op = increment \wedge q0 = X17 \\
\quad \bullet\ \exists q1: Q \mid q1 \in states \wedge q1 = X19 \bullet trans1\,(q0, op) = q1 \\
\exists q0: Q;\ op: Op \mid q0 \in states \wedge op \in operators \wedge op = equalto \wedge q0 = X17 \\
\quad \bullet\ \exists q1: Q \mid q1 \in states \wedge q1 = X20 \bullet trans1\,(q0, op) = q1 \\
\exists q0: Q;\ op: Op \mid q0 \in states \wedge op \in operators \wedge op = mul \wedge q0 = X0 \\
\quad \bullet\ \exists q1: Q \mid q1 \in states \wedge q1 = X34 \wedge q1 \in finals \bullet trans1\,(q0, op) = q1 \\
\exists q0: Q;\ op: Op \mid q0 \in states \wedge op \in operators \wedge op = equalto \wedge q0 = X34 \\
\quad \bullet\ \exists q1: Q \mid q1 \in states \wedge q1 = X35 \wedge q1 \in finals \bullet trans1\,(q0, op) = q1 \\
\exists q0: Q;\ op: Op \mid q0 \in states \wedge op \in operators \wedge op = sub \wedge q0 = X0 \\
\quad \bullet\ \exists q1: Q \mid q1 \in states \wedge q1 = X36 \wedge q1 \in finals \bullet trans1\,(q0, op) = q1 \\
\exists q0: Q;\ op: Op \mid q0 \in states \wedge op \in operators \wedge op = equalto \wedge q0 = X36 \\
\quad \bullet\ \exists q1: Q \mid q1 \in states \wedge q1 = X37 \wedge q1 \in finals \bullet trans1\,(q0, op) = q1
\end{array}
$$

f) There exits an operator *op* and states *q0* and *q1* an output alphabet, *op* belongs to *Op* and *q0* belongs to state and *q0* is equal to *X18* and operator is equal to *add*. *trans1 (q0, op) = q1* where transaction function acting on *q0* and operator *op* and give a next state q1 that is equal to *X19*. After reading the add operator, the lexical analyzer can have more than one path. It can read again *increment* or *add* *equal to* or *other*. If it reads the other, then state will not change and scanner recognize that *add* operator is found.

NumberTrans

$\Xi LexicalAnalyzer$
$point: digit1$
$plus: digit1$
$minus: digit1$
$E: digit1$

$\exists q0: Q; d: digit1 \mid q0 \in states \land d \in digit1 \land q0 = X0$
$\quad \bullet \exists q1: Q \mid q1 \in states \land q1 = X3 \bullet trans2(q0, d) = q1$
$\exists q0: Q; d: digit1 \mid q0 \in states \land d \in digit1 \land q0 = X3$
$\quad \bullet \exists q1: Q \mid q1 \in states \land q1 = X3 \land q1 \in finals \bullet trans2(q0, d) = q1$
$\exists q0: Q; d: digit1 \mid q0 \in states \land q0 = X3 \land d \in digit1 \land d = point$
$\quad \bullet \exists q1: Q \mid q1 \in states \land q1 = X5 \bullet trans2(q0, d) = q1$
$\exists q0: Q; d: digit1 \mid q0 \in states \land q0 = X5 \land d \in digit1$
$\quad \bullet \exists q1: Q \mid q1 \in states \land q1 = X6 \bullet trans2(q0, d) = q1$
$\exists q0: Q; d: digit1 \mid q0 \in states \land q0 = X6 \land d \in digit1$
$\quad \bullet \exists q1: Q \mid q1 \in states \land q1 = X7 \land q1 \in finals \bullet trans2(q0, d) = q1$
$\exists q0: Q; d: digit1 \mid q0 \in states \land d \in digit1 \land q0 = X6 \land d = E$
$\quad \bullet \exists q1: Q \mid q1 \in states \land q1 = X8 \bullet trans2(q0, d) = q1$
$\exists q0: Q; d: digit1 \mid q0 \in states \land d \in digit1 \land d = minus \lor d = plus \land q0 = X8$
$\quad \bullet \exists q1: Q \mid q1 \in states \land q1 = X9 \bullet trans2(q0, d) = q1$
$\exists q0: Q; d: digit1 \mid q0 \in states \land d \in digit1 \land q0 = X8$
$\quad \bullet \exists q1: Q \mid q1 \in states \land q1 = X10 \bullet trans2(q0, d) = q1$
$\exists q0: Q; d: digit1 \mid q0 \in states \land d \in digit1 \land q0 = X9$
$\quad \bullet \exists q1: Q \mid q1 \in states \land q1 = X10 \land q1 \in finals \bullet trans2(q0, d) = q1$
$\exists q0: Q; d: digit1 \mid q0 \in states \land d \in digit1 \land d = E \land q0 = X3$
$\quad \bullet \exists q1: Q \mid q1 \in states \land q1 = X8 \bullet trans2(q0, d) = q1$

---

f) There exists a digit *d* and states *q0* and *q1* an output alphabet, *d* belongs to *digit1* and *q0* belongs to state and *q0* is equal to *X0* and operator is equal to *add*. *trans2 (q0, d)* = q1 where transaction function acting on *q0* and digit *d* and give a next state q1 that is equal to *X3*. After reading the *d* digit, the lexical analyzer can have more than one path. It can read again *digit* or *point* or *E*. If it reads the other, then state will not change and scanner recognize that integer is found. If it reads the point, then state will change and the next state will be *X5.*

If it reads the exponent *E*, then state will change and the next state will be *X8.* At state *X5* it will read digit then move to next state *X7.* It can read more digits at this state but when the other character is entered by the user, it recognizes the float Number. At state X6, it will read exponent E then moves to next state *X8.* At *X8* it can read plus or minus and goes to next state *X9.* At *x9* it will read digit and goes to state *X10.* When it reads the other it goes to final state and recognizes that the exponent number is found.

$$
\begin{array}{|l}
\hline
\textit{SDQuotes} \\
\Xi LexicalAnalyzer \\
ed: digit1 \\
\hline
\exists q0: Q; p: Pun \mid q0 \in states \wedge q0 = X0 \wedge p = singleqstart \\
\quad \bullet \exists q1: Q \mid q1 \in states \wedge q1 = X12 \bullet trans3(q0, p) = q1 \\
(\exists q0: Q; a: A; d: digit1 \mid q0 \in states \wedge q0 = X12 \wedge a \in alpha \wedge d \in digit1 \\
\quad\quad \bullet (\exists q1: Q \mid q1 \in states \wedge a = ea \wedge q1 = X13 \bullet trans(q0, (d, a)) = q1)) \\
\vee (\exists q0: Q; a: A; d: digit1 \mid q0 \in states \wedge d = ed \wedge q0 = X12 \\
\quad\quad\quad \bullet (\exists q1: Q \mid q1 \in states \wedge q1 = X13 \bullet trans(q0, (d, a)) = q1)) \\
\exists q0: Q; p: Pun \mid q0 \in states \wedge q0 = X13 \wedge p = singleqend \\
\quad \bullet \exists q1: Q \mid q1 \in states \wedge q1 = X14 \bullet trans3(q0, p) = q1 \\
\exists q0: Q; p: Pun \mid q0 \in states \wedge q0 = X0 \wedge p = doubleqstart \\
\quad \bullet \exists q1: Q \mid q1 \in states \wedge q1 = X15 \bullet trans3(q0, p) = q1 \\
\exists q0: Q; a: A; d: digit1 \mid q0 \in states \wedge q0 = X15 \wedge a \in alpha \wedge d \in digit1 \\
\quad \bullet \exists q1: Q \mid q1 \in states \wedge a = ea \wedge q1 = X15 \bullet trans(q0, (d, a)) = q1 \\
\exists q0: Q; p: Pun \mid q0 \in states \wedge q0 = X15 \wedge p = doubleqend \\
\quad \bullet \exists q1: Q \mid q1 \in states \wedge q1 = X16 \bullet trans3(q0, p) = q1 \\
\exists q0: Q; p: Pun \mid q0 \in states \wedge q0 = X0 \wedge p = semicolon \\
\quad \bullet \exists q1: Q \mid q1 \in states \wedge q1 = X25 \bullet trans3(q0, p) = q1 \\
\hline
\end{array}
$$

f) There exists a Punctuation *Pun* and states *q0* and *q1* an output alphabet, *p* belongs to *Pun* and *q0* belongs to state and *q0* is equal to *X0* and p is equal to *singleqstart*. trans3 (q0, p) = q1 where transaction function acting on *q0* and *Pun p* and give a next state q1 that is equal to X12. We want to recognize letter or digit at state *X12* from moving the next state *X13*. Another transition function trans (q0, (d, a)) = q1)) is used that is acting on q0 and order pair *d and a* and gives a next state *q1* that is equal to *X13*. At *X13* it reads the single qend and moves to next state which is equal to *X14*. The scanner recognizes that the single quotes is found.

g) There exists a Punctuation *Pun* and states *q0* and *q1* an output alphabet, *p* belongs to *Pun* and *q0* belongs to state and *q0* is equal to *X0* and p is equal to *doubleqstart*. trans3 (q0, p) = q1 where transaction function acting on *q0* and Pun p and give a next state q1 that is equal to X15. We want to recognize letters or digits at state *X12* and will not change the states. Another transition function trans (q0, (d, a)) = q1)) is used that is acting on *q0* and order pair *d and a* and gives a next state *q1* that is equal to *X15*. At *X15* it reads the double qend and moves to next state which is equal to *X14*. The scanner recognizes that the double quotes is found.

Comments

$$\Xi LexicalAnalyzer$$
$$ed : digit1$$

$$\exists q0 : Q; p : Pun \mid q0 \in states \wedge q0 = X0 \wedge p = slash$$
$$\bullet \exists q1 : Q \mid q1 \in states \wedge q1 = X30 \bullet trans3 (q0, p) = q1$$
$$\exists q0 : Q; p : Pun \mid q0 \in states \wedge q0 = X30 \wedge p = star$$
$$\bullet \exists q1 : Q \mid q1 \in states \wedge q1 = X31 \bullet trans3 (q0, p) = q1$$
$$(\exists q0 : Q; a : A; d : digit1 \mid q0 \in states \wedge q0 = X31 \wedge a \in alpha \wedge d \in digit1$$
$$\bullet (\exists q1 : Q \mid q1 \in states \wedge a = ea \wedge q1 = X31 \bullet trans (q0, (d, a)) = q1))$$
$$\vee (\exists q0 : Q; a : A; d : digit1 \mid q0 \in states \wedge d = ed \wedge q0 = X31$$
$$\bullet (\exists q1 : Q \mid q1 \in states \wedge q1 = X31 \bullet trans (q0, (d, a)) = q1))$$
$$\exists q0 : Q; p : Pun \mid q0 \in states \wedge q0 = X31 \wedge p = star$$
$$\bullet \exists q1 : Q \mid q1 \in states \wedge q1 = X32 \bullet trans3 (q0, p) = q1$$
$$\exists q0 : Q; p : Pun \mid q0 \in states \wedge q0 = X32 \wedge p = slash$$
$$\bullet \exists q1 : Q \mid q1 \in states \wedge q1 = X33 \wedge q1 \in finals \bullet trans3 (q0, p) = q1$$

f) There exists a Punctuation *Pun* and states *q0* and *q1* an output alphabet, *p* belongs to *Pun* and *q0* belongs to state and *q0* is equal to *X0* and *p* is equal to *slash*. *trans3 (q0, p) = q1* where transaction function acting on *q0* and Pun *p* and give a next state q1 that is equal to X30. When reaching at *X30*, we will read punctuation marks p that is equal to star and we obtain next state *q1* which is equal to *X31*. We want to recognize letter or digit at state *X31* and will not change the state. Then we read punctuation symbol star and go to another state *q1* that is equal to *X32*. At *X32* we will read slash and go to final state that is *X33*. The compiler ignores the comments and treated as blanks.

$R ::= rword$

a data type R is give a message when a reserved word is found in the specification.

R_IF

$$\Xi LexicalAnalyzer$$
$$i : A$$
$$f : A$$
$$report : \mathbb{P}\ R$$

$$\exists q0 : Q; a : A \mid a = i \wedge q0 \in states \wedge q0 = X0$$
$$\bullet \exists q1 : Q \mid q1 \in states \wedge q1 = X40 \bullet trans4 (q0, a) = q1$$
$$\exists q0 : Q; a : A; report1 : report$$
$$\mid a = f \wedge report1 = rword \wedge q0 \in states \wedge q0 = X0$$
$$\bullet \exists q1 : Q \mid q1 \in states \wedge q1 = X41 \bullet trans4 (q0, a) = q1$$

There exits an alphabet *a* and states *q0* and *q1* an output alphabet, *a* belongs to alpha and *q0* belongs to state and *q0* is equal to *X0* and *a* is equal to *i*. *trans4 (q0, a) = q1* where transaction function acting on *q0* and alpha *a* and give a next state q1 that is equal to *X40*. At *X40* it will read another character that is f and gives a report that reserved word found and changes its state from *X40* to *X41*. we may write all specification of punctuation marks, number and operators. In this paper we gave an procedure of representing the formal specification of lexical analyzer in Z-eves, it will also increase the modeling power of lexical analysis because the integration of automata and Z will increase the modeling power. The *trans* function for recognizing the operator will be trans1: Q x Op f Q and recognizing the punctuation marks will be trans3: Q x Pun f Q and for number is trans2: Q x digit1 f Q.

3. Conclusion and Future Work. Every software is totally dependent on the compiler, either it is safety critical or not its mean the compiler should be bugs and error free. In this work we have presented the formal specification and verification of a recognizing the Identifier and Reserved word of Lexical Analyzer. Formal verification of compiler optimization is very important and may be verified later in Z specification.

REFERNCES

[1] Bogdanov, K., Bowen, J. P., Cleaveland, R., Derrick, J., Dick, J., Gheorghe, M., ... & Zedan, H. (2003). Working together: Formal methods and testing. *ACM Computing Surveys (December 2003).*

[2] Goos, G., & Znnmerrnaun, W. (1999). Verification of compilers. In Correct System Design (pp. 201-230). Springer Berlin Heidelberg.Chicago

[3] Zafar, N. A., Sabir, N., & Ali, A. (2008, April). Construction of intersection of nondeterministic finite automata using Z notation. In *Proceedings of World Academy of Science, Engineering and Technology* (Vol. 30, pp. 591-596).

[4] Liu, S., Tamai, T., & Nakajima, S. (2009, March). Integration of formal specification, review, and testing for software component quality assurance. In*Proceedings of the 2009 ACM symposium on Applied Computing* (pp. 415-421). ACM.

[5] www.wikipedia.com

[6] Zafar, N. A., & Shah, S. H. H. (2009, February). Algorithmic Formal Proof of Equivalence of Nondeterministic and Deterministic Finite Automata. In*Electronic Computer Technology, 2009 International Conference on* (pp. 108-112). IEEE.

[7] Blech, J. O., Glesner, S., Leitner, J., & Mülling, S. (2005). Optimizing code generation from SSA form: A comparison between two formal correctness proofs in Isabelle/HOL. *Electronic Notes in Theoretical Computer Science,141*(2), 33-51.

[8] Blazy, S., Dargaye, Z., & Leroy, X. (2006). Formal verification of a C compiler front-end. In *FM 2006: Formal Methods* (pp. 460-475). Springer Berlin Heidelberg.

[9] Leroy, X. (2009). A formally verified compiler back-end. *Journal of Automated Reasoning*, *43*(4), 363-446.

[10] Chlipala, A. (2010, January). A verified compiler for an impure functional language. In *ACM Sigplan Notices* (Vol. 45, No. 1, pp. 93-106). ACM.

[11] Strecker, M. (2005). *Compiler verification for C0*. Technical report, Universite Paul Sabatier, Toulouse.

[12] Thomson. "Regular Expression and Automata Theory."1995.

[13] Curzon, P. (1992). Of what use is a verified compiler specification?.

[14] Stringer-Calvert, D. W. J. (1998). *Mechanical verification of compiler correctness* (Doctoral dissertation, University of Yor