KYBERNETIKA — VOLUME 26 (1990), NUMBER 4

ON THE SKELETAL STRUCTURE OF PASCAL STATEMENT

MAREK VAŠÍN

This paper contributes to the *error recovery* method for *syntactical analysis* of deterministic contex-free languages introduced in [1]. The method is based on the notion of *skeletal set*. A skeletal set is a subset of the terminal alphabet meeting special conditions. Due to these conditions, any string containing errors can be corrected so that the parser always recovers on the next skeletal symbol. To illustrate the practical applicability of the above mentioned method we prove that a convenient subset of Pascal terminal alphabet forms a skeletal set for the language of *Pascal statements*.

The reader is assumed to be familiar with basic concepts of the theory of formal languages [2].

1. INTRODUCTION

In any compiler, error detection and error recovery belong to the most important aspects of source language processing. Most compilers use error recovery methods based on the structure of their parsing algorithms. The method of error recovery for deterministic contex-free languages described in [1] is fully language oriented.

It is based on the structure of certain key symbols in the parsed string, in analogy to "panic mode" recovery. However, in this method, the key symbols are not independent; they form an interrelated "skeletal set" of the language. These skeletal symbols decompose the parsed string into segments. The correctness of any segment within the input string depends only on the skeletal structure of the string. If the segment is incorrect the method replaces it by a correct one.

To find a non-trivial skeletal set for a language may be a non-routine task. As proved in [1], it is, in general, algorithmically unsolvable. Nevertheless, when a skeletal set is found, it can be used for automatic generation of an error handling routine for any parser, which has the form of a deterministic push-down automaton. Therefore finding a good skeletal set for a programming language is an important contribution to the construction of its parsers.

In this paper, the property of skeletal set is proved for a set of symbols of the language of Pascal statements. It can be used as a practical example of a skeletal

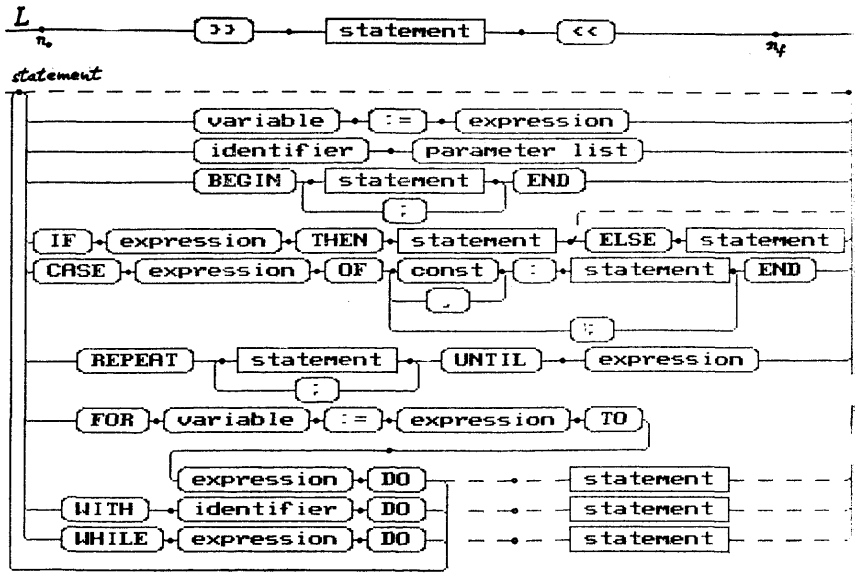

Fig. 1.

set of a language. The language is defined by Figure 1. In the full description of Pascal, the non-terminal symbols ⟨*expression*⟩, ⟨*constant*⟩, ⟨*variable*⟩ etc. are further expanded by detailed syntax diagrams. For the sake of simplicity we shall treat them as terminal symbols here.

2. NOTATION AND THE MAIN RESULT

In what follows, L will denote a language over an alphabet X, K will be a subset of X, and V will stand for the set $X - K$. For the sake of simplicity of some concepts, the alphabet X is assumed to contain two special symbols $\gg$ and $\ll$ (endmarkers) surrounding any string from L (thus $\gg \ll$, $\gg$ *BEGIN END* $\ll$, $\gg$ *IF expression THEN BEGIN*; *END ELSE* $\ll$ will be legal examples of strings from our language of Pascal statements). We shall use the symbol K to denote also the homomorphism $X^* \to K^*$ which deletes in a given string all symbols from V and leaves the symbols from K intact.

Definition 1 (cf. [1]). Let K be a subset of X such that $\{\gg, \ll\} \subseteq K$. We shall say that K is a *skeletal set* (or a set of skeletals) of the language L if for each $a, b \in K$, $u', u'' \in V^*$, $x', x'', y', y'' \in X^*$ such that $K(x') = K(x'')$, the following condition holds:

$$x'au'by' \in L \,\&\, x''au''by'' \in L \quad \text{implies} \quad x'au''by' \in L \tag{1}$$

For any string w in the alphabet X, we call $K(w)$ the *skeleton* of w. It is easy to see that the sets $\{\gg, \ll\}$ and X are skeletal sets of L. We shall call them the *trivial skeletal sets.*

Theorem. The set

$$K = \{\gg, BEGIN, REPEAT, IF, CASE,;, END, ELSE, UNTIL, \ll\} \qquad (2)$$

forms a minimal non-trivial skeletal set for the language

$$L = \{\gg\}\ \langle statement\rangle\ \{\ll\} \quad \text{w.r.t. inclusion .}$$

3. PROOF (Part I)

First we shall prove the minimality: Let us assume that some set K_0 is a non-trivial skeletal set of L; we shall show for each element of K that it is either an element of K_0 or it replaces an element of K_0:

a) Let us assume that the *semicolon* is not an element of K_0. Since K_0 is by our assumption a non-trivial skeletal set, there exists a word $\gg\ \alpha\ \ll \in L$ such that $|K_0(\alpha)| > 0$. Let us consider two cases:

a_1) $ELSE \notin K_0$: From the syntax diagram it is clear that the following two strings (3), (4) belong to the language L:

$$\gg\ BEGIN\ IF\ expression\ THEN\ IF\ expression\ THEN\ BEGIN\ \boxed{\alpha\ END}\ ELSE\ \boxed{BEGIN\ \alpha}\ END\ ELSE\ END \ll \qquad (3)$$

$$\gg\ BEGIN\ IF\ expression\ THEN\ IF\ expression\ THEN\ BEGIN\ \boxed{\alpha\ END}\ ;\ \boxed{BEGIN\ \alpha}\ END\ END \ll \qquad (4)$$

Hence also the string (5) should belong to L. It follows from (1) when we use as the skeletal a the last skeletal symbol of the string $\alpha\ END$, and as b the first skeletal symbol of $BEGIN\ \alpha$, as shown above:

$$\gg\ BEGIN\ IF\ expression\ THEN\ IF\ expression\ THEN\ BEGIN\ \boxed{\alpha\ END}\ ;\ \boxed{BEGIN\ \alpha}\ END\ ELSE\ END \ll \qquad (5)$$

It is a contradiction because the symbol *ELSE* is separated from the symbols *IF* by the *semicolon*.

a_2) If $ELSE \in K_0$, we can derive a contradiction analogically. The correctness of the following two strings:

$$\gg\ BEGIN\ IF\ expression\ THEN\ IF\ expression\ THEN\ \boxed{ELSE}\ \boxed{BEGIN\ \alpha}\ END\ ELSE\ END \ll$$

$$\gg\ BEGIN\ IF\ expression\ THEN\ IF\ expression\ THEN\ \boxed{ELSE}\ ;\ \boxed{BEGIN\ \alpha}\ END\ END \ll$$

implies a contradiction

$$\gg \textit{BEGIN IF expression THEN}$$
$$\textit{IF expression THEN } \boxed{\textit{ELSE}} \; ; \; \boxed{\textit{BEGIN } \alpha} \; \textit{END ELSE END} \ll \in L$$

when we define a in (1) as *ELSE* and b as the first skeletal symbol in the string *BEGIN* α. Thus the *semicolon* symbol is an element of any non-trivial skeletal set of the language L.

Using the same arguments we shall find contradictions forcing other symbols to be elements of the set K_0:

b) **Begin:**

$$\begin{array}{lll} \gg \textit{BEGIN} \boxed{;} & \boxed{;} & \textit{END} \ll \in L, \\ \gg \textit{BEGIN} \boxed{;} \; \textit{BEGIN} & \boxed{;} & \textit{END END} \ll \in L \Rightarrow \\ \gg \textit{BEGIN} \boxed{;} \; \textit{BEGIN} & \boxed{;} & \textit{END} \ll \in L. \end{array}$$

c) **Repeat:**

$$\begin{array}{lll} \gg \textit{REPEAT} \boxed{;} & \boxed{;} & \textit{UNTIL expression} \ll \in L, \\ \gg \textit{REPEAT} \boxed{;} & \boxed{;} & \textit{UNTIL expression UNTIL expression} \ll \in L \\ \Rightarrow \gg \textit{REPEAT} \boxed{;} \; \textit{REPEAT} & \boxed{;} & \textit{UNTIL expression} \ll \in L \end{array}$$

d) **End:**

$$\begin{array}{lll} \gg \boxed{\textit{BEGIN}} \; \textit{END} & \boxed{;} & \textit{BEGIN END} \ll \in L, \\ \gg \boxed{\textit{BEGIN}} & ; & \textit{END} \ll \in L \Rightarrow \\ \gg \boxed{\textit{BEGIN}} & \boxed{;} & \textit{BEGIN END} \ll \in L. \end{array}$$

e) **Else:**

$$\begin{array}{lll} \gg \textit{BEGIN IF expression THEN ELSE BEGIN} \boxed{\textit{END}} & \boxed{\textit{END}} & \ll \in L, \\ \gg \textit{BEGIN IF expression THEN BEGIN} \boxed{\textit{END}} \; \textit{ELSE} & \boxed{\textit{END}} & \ll \in L \Rightarrow \\ \gg \textit{BEGIN IF expression THEN ELSE BEGIN} \boxed{\textit{END}} \; \textit{ELSE} & \boxed{\textit{END}} & \ll \in L. \end{array}$$

f) **Until:** Let $\{\textit{expression}, \textit{UNTIL}\} \cap K_0 = \emptyset$; then

$$\begin{array}{lll} \gg \boxed{\textit{REPEAT}} \; \textit{UNTIL expression} & \boxed{;} & \textit{REPEAT UNTIL expression} \ll \in L, \\ \gg \boxed{\textit{REPEAT}} & ; & \textit{UNTIL expression} \ll \in L \Rightarrow \\ \gg \boxed{\textit{REPEAT}} & \boxed{;} & \textit{REPEAT UNTIL expression} \ll \in L. \end{array}$$

Thus *UNTIL* and/or *expression are elements of* K_0.

g) **Case:** Let $\{\textit{CASE}, \textit{expression}, :, \textit{OF}\} \cap K_0 = \emptyset$, then

$$\begin{array}{lll} \gg \textit{BEGIN} \boxed{;} & \boxed{;} & \textit{END} \ll \in L, \\ \gg \textit{BEGIN} \boxed{;} \; \textit{CASE expression OF constant:} & \boxed{;} & \textit{END END} \ll \in L \Rightarrow \\ \gg \textit{BEGIN} \boxed{;} \; \textit{CASE expression OF constant:} & \boxed{;} & \textit{END} \ll \in L. \end{array}$$

Thus at least one of symbols from $\{\textit{CASE}, \textit{expression}, :, \textit{OF}\}$ is an element of K_0.

h) **If:** Let $\{IF, THEN, expression\} \cap K_0 = \emptyset$, then

$$\left.\begin{array}{l} \gg \\ \gg \\ \gg \end{array}\right| \; IF\ expression\ THEN \; \left|\begin{array}{l} BEGIN \\ BEGIN \\ BEGIN \end{array}\right. \; \begin{array}{ll} END\ ELSE & \ll \in L, \\ END & \ll \in L \Rightarrow \\ END\ ELSE & \ll \in L. \end{array}$$

Thus at least one of symbols from $\{IF, THEN, expression\}$ is an element of K_0.

In fact, we have proved that symbols ;,*BEGIN*, *END*, *REPEAT*, *ELSE* are elements of any non-trivial skeletal set of L and that symbols *UNTIL*, *IF* and *CASE* cannot be omitted from any non-trivial skeletal set without replacing them by other symbols (e.g. *IF* by *THEN*, *CASE* by *OF*, etc.).

4. ON THE SYNTAX DIAGRAMS

We use syntax diagrams to describe the syntax of the language of Pascal statements. Since the exact representation and meaning of syntax diagrams are not so apparent as their intuitive use, we shall present some definitions (cf. [3]):

Definition 2. A *recursive transition net* (or RTN) is any quintuplet $R = (\Sigma, X, \delta, n_0, F)$ such that:

Σ is a non-empty set (*set of nodes*),
X is a finite *input alphabet* such that $X \cap \Sigma = \emptyset$,
δ is a (partial) *transition mapping*: $\delta: \Sigma \times (\Sigma \cup X) \to \exp(\Sigma)$,
n_0 is an element of Σ (*initial node*),
F is a subset of Σ (*set of final nodes*).

An RTN can be seen in the syntax diagram as follows

- The set Σ contains points of the syntax diagram which can be reached while going through the diagram. In our Pascal syntax diagram, we mark these points with the sign "•".
- The alphabet X contains all terminal symbols from the syntax diagram (i.e. all the symbols enclosed within ovals).
- The mapping δ maps the pair (n, t) to the set $\delta(n, t)$ containing all the nodes $n' \in \Sigma$ such that there is an edge from the node n to the node n' labelled by the symbol t in the syntax diagram. The symbol t can be either from X or from Σ. In the second case, t is a non-terminal symbol (enclosed within a rectangular box) referring to the entry point of a component of the syntax diagram.
- The node n_0 corresponds with the entry point of the main component.
- The set F contains the exit points of the diagram components.

Definition 3. We say that C is a *configuration of the RTN* $R = (\Sigma, X, \delta, n_0, F)$ if $C = \langle n, x, \gamma \rangle$ for some $n \in \Sigma$, $x \in X^*$ and $\gamma \in \Sigma^*$.

Definition 4. We shall say that a *configuration* C' of the RTN $R = (\Sigma, X, \delta, n_0, F)$

leads to a configuration C'' (or $C' \Rightarrow_R C''$) if one of the following conditions holds:

a) $C' = \langle n', tx, \gamma \rangle$, $C'' = \langle n'', x, \gamma \rangle$, $t \in X$, for some $n'' \in \delta(n', t)$

b) $C' = \langle n', x, \gamma \rangle$, $C'' = \langle n'', x, n\gamma \rangle$ for some $n \in \delta(n', n'')$

c) $C' = \langle n', x, n''\gamma \rangle$, $C'' = \langle n'', x, \gamma \rangle$ for some $n' \in F$ and $n'' \in \Sigma$.

The reflexive and transitive closure of the relation $\Rightarrow_R$ will be denoted as $\Rightarrow_R^*$.

Definition 5. Let $R = (\Sigma, X, \delta, n_0, F)$ be an RTN. We shall say that the *language* L *is recognized by* R (or $L = L(R)$) if

$$L = \{x \in X^* \mid \langle n_0, x, \lambda \rangle \Rightarrow_R^* \langle n, \lambda, \lambda \rangle \text{ for some } n \in F\}$$

Thus RTN may be seen to be a non-deterministic automaton with a push-down to save return points of uncompleted diagram components. The configuration describes the actual state of the computation, i.e. the current node, the remaining part of the input string and the content of the push-down. In each step one of the following actions is done:

a) If there is an edge from the current node labelled by the same terminal symbol as on input, the symbol can be accepted and the control moves along the edge.

b) If there is an edge from the current node labelled by a non-terminal symbol (node, component name), this node (diagram component) can be entered. The node of the return point is actually pushed onto the push-down.

c) If the current node is a final one, the computation can return back to the node popped from the top of the push-down; if the push-down is empty, the computation finishes and the input string is accepted as a string from L.

There are two differences in our Pascal diagram w.r.t. the above definitions; the edge of "empty statement" and the edge by-passing the "*ELSE* clause" are not labelled by any symbol. Since these "λ-edges" point to the final node, we can avoid them by including nodes preceding these edges to the set of final nodes F, too. As can be seen from the diagram, these nodes are the only points where the computations would not be deterministic; therefore we shall prefer the action a) in such nodes (according to Pascal semantics).

5. PROOF (Part II)

Now we shall show that K is a skeletal set of L: Let us modify the RTN of Pascal statements from Figure 1 to eliminate right recursion of the ⟨*statement*⟩ non-terminal in edges of *WITH*, *WHILE* and *FOR* clauses as shown in Figure 1. Thus we remove the edge of *DO* in clauses of *WITH*, *WHILE*, *FOR* resp., the node entered by this edge and the edge of ⟨*statement*⟩ going from that node. Then we replace them by an edge of *DO* entering the node from which the edge of *WITH*, *WHILE*, *FOR* resp. went. The modified RTN evidently accepts the same language as the original net.

The following lemma shows that all computations in the modified net over strings with the same skeleton coincide after each skeletal accepted:

Lemma. Let $z'by'$, $z''by'' \in L$, $z', z'', y', y'' \in X^*$, $K(z') = K(z'')$, $b \in K$ and let for some $m', m'' \in \Sigma$

$$\langle n_0, z'by', \lambda\rangle \Rightarrow_R^* \langle m', by', \gamma'\rangle \Rightarrow_R \langle n', y', \gamma'\rangle \Rightarrow_R^* \langle n_f, \lambda, \lambda\rangle ,$$

$$\langle n_0, z''by'', \lambda\rangle \Rightarrow_R^* \langle m'', by'', \gamma''\rangle \Rightarrow_R \langle n'', y'', \gamma''\rangle \Rightarrow_R^* \langle n_f, \lambda, \lambda\rangle ,$$

be accepting computations of words $z'by'$ and $z''by''$ in the modified RTN; then $n' = n''$ and $\gamma' = \gamma''$.

Proof. We shall use induction on $k = |K(z')| = |K(z'')|$: Let $k = 0$, then $z' = z'' = \lambda$, $b = \gg$ and the statement evidently holds. Let the statement hold for all strings zby such that $|K(z)| < k$ and let $k > 0$. Then we can write $z'by' = x'au'by'$, $z''by'' = x''au''by''$ for some $x', x'' \in X^*$, $K(x') = K(x'')$, $a \in K$, $u', u'' \in V^*$.

From the above assumption it follows that there exist unique n and γ such that

$$\langle n_0, x'au'by', \lambda\rangle \Rightarrow_R^* \langle n, u'by', \gamma\rangle \Rightarrow_R^* \langle m', by', \gamma'\rangle \Rightarrow_R \langle n', y', \gamma'\rangle \Rightarrow_R^* \langle n_f, \lambda, \lambda\rangle$$

$$\langle n_0, x''au''by'', \lambda\rangle \Rightarrow_R^* \langle n, u''by'', \gamma\rangle \Rightarrow_R^* \langle m'', by'', \gamma''\rangle \Rightarrow_R \langle n'', y'', \gamma''\rangle \Rightarrow_R^* \langle n_f, \lambda, \lambda\rangle.$$

We shall prove that then $n' = n''$ and $\gamma' = \gamma''$: The condition $n' = n''$ holds since the skeletal symbol b determines the succeeding node of computation uniquely. It is evident from the diagram; the symbol ; with multiple occurrences is the only exception. However, as can be seen from the diagram in this case, the skeletal symbol a is either from the *same* level of ⟨*statement*⟩ call (e.g. symbol *BEGIN* or *REPEAT* or *CASE* or;) or from the *nested* call of ⟨*statement*⟩. Since γ and n are unique after a was accepted, $n' = n''$ holds evidently.

When accepting string $u'b$, $u''b$ resp., the push-down is changed from γ to γ', γ'' resp. We shall show that this change depends only on the current push-down content γ and on symbols a and b: As can be seen, we can determine the direction of the push-down changes from the symbols a and b since all recursive calls of non-terminal ⟨*statement*⟩ are prefixed by skeletal symbols:

a/b	*BEGIN*	*REPEAT*	*CASE*	*IF*	*ELSE*	;	*END*	*UNTIL*	≪
≫	↘	↘	↘	↘					=
BEGIN	↘	↘	↘	↘		=	=		
REPEAT	↘	↘	↘	↘		=		=	
CASE	↘	↘	↘	↘		=	=		
IF	↘	↘	↘	↘	=	↗	↗	↗	↗
ELSE	↘	↘	↘	↘	↗	↗	↗	↗	↗
;	↘	↘	↘	↘		=	=	=	
END					↗	↗	↗	↗	↗
UNTIL					↗	↗	↗	↗	↗

(↘) One symbol is pushed onto the push-down; i.e. $\gamma' = \gamma'' = v\gamma$, where v is node entered by the edge labelled by ⟨*statement*⟩ in the branch determined by the skeletal

symbol a and by (in the case of symbol; with multiple occurrences) the actual node of computation n:

From the diagram it is evident that the skeletal symbols *BEGIN* or *REPEAT* or *IF* or *CASE* (i.e. the symbol b) can be accepted only in a new recursive call of non-terminal ⟨*statement*⟩; thus the the push-down grows. On the other hand, all recursive calls of ⟨*statement*⟩ are prefixed by (the same) skeletals; thus the push-down grows by just one symbol.

(=) Push-down remains unchanged: It is evident that there exists a unique path connecting a and b and not going through another skeletal. The push-down cannot grow since b is not from {*BEGIN*, *REPEAT*, *IF CASE*}, and cannot drop since a is not from {*END*, *UNTIL*. *ELSE*, ≪}.

(↗) Push-down decreases: Having gone through the same level of ⟨*statement*⟩ call from the symbol a to a final node, the push-down has the same content, namely γ. Leaving this level new current node is popped from the push-down. From this node, there evidently exists a unique path to either another final node (in the case of implicit end of nested *THEN* or *ELSE* clauses) or directly to the skeletal symbol b.

Since the initial contents γ of push-down is unique, and the changes are determined only by the push-down and by the symbols a and b, we are sure that $\gamma' = \gamma''$ holds. □

One can see that the unique content of the push-down reflects the equality of skeletons of prefixes $K(x') = K(x'')$. The reason we have reduced right recursion in branches of non-skeletals *WITH*, *FOR*, *WHILE* was to reduce "non-interesting" return points and make the push-down content "isomorphic" with the strings $K(x') = K(x'')$.

The lemma proved immediately implies the property (1) of skeletal set for the set K. The strings $x'a$ and $x''a$ lead in the modified RTN to the same node of computation and to the same push-down content as well as the strings $x'au'b$ and $x'au''b$ do. Thus the strings $x'au'by'$ and $x'au''by'$ are equally accepted or rejected as strings of L.

ACKNOWLEDGEMENT

The author is deeply indebted to Michal Chytil for reading the first drafts and for many valuable comments. (Received January 10, 1989.)

REFERENCES

[1] M. P. Chytil and J. Demner: Panic mode without panic. In: Automata, Languages and Programming (Thomas Ottman, ed., Lecture Notes on Computer Science 267), Springer-Verlag, Berlin—Heidelberg—New York 1987, pp. 260—268.

[2] J. E. Hopcroft and J. D. Ullman: Formal Languages and Their Relation to Automata. Addison-Wesley, Reading 1969.

[3] M. Chytil and S. Scherlová: Sémantika programovacích jazyků — atributové gramatiky (Semantics of Programming Languages). Univerzita Karlova, Praha 1984.

RNDr. Marek Vašín, Univerzita Karlova, matematicko-fyzikální fakulta (Charles University — Faculty of Mathematics and Physics), V Holešovičkách 2, 180 00 Praha 8. Czechoslovakia.