IBM

IBM United Kingdom
Laboratories Limited

CBJ (PERS)

Formal Development of Correct Algorithms: An Example Based on Earley's Recogniser

C. B. Jones

Technical Report T.R.12.095

Technical Report T.R.12.095

Formal Development of Correct Algorithms: An Example Based on Earley's Recogniser

C. B. Jones

December 1971 215-8043-0

IBM United Kingdom Laboratories Limited
Hursley Park
Winchester Hampshire

Abstract

This paper contains the formal development of a correct algorithm from an implicit definition of the task to be performed. Each step of the development is accompanied by a proof of its correctness. As well as ensuring the correctness of the final program, the structured development gives considerable insight into the algorithm and possible alternatives.

The example used is a simplified form of the recognition algorithm due to Earley.

Acknowledgements

The author gratefully acknowledges that the idea of levels of an algorithm was suggested by P Lucas. The current paper represents a reasonably complex example and the author's own view of how to add the proofs.

Earley's paper provided not only the ideas of the algorithm but also the basis for some of the proofs of the first sections. The author is also grateful for useful discussions with C D Allen, D N Chapman, and P D Wright for providing the original running program.

Although differing in important respects, the author acknowledges the stimulus of the works of Dijkstra, Hoare and Mills relating to development of programs.

Note: Parts of this paper are to be presented at the ACM SIGPLAN symposium on "Proving Assertions about Programs", the current report having been printed to make the full details of the proofs available. As a courtesy to the intended publishers, distribution should be restricted.

0. Introduction

The published literature on proving programs correct has tended to confine itself to rather simple algorithms. Apart from the obvious difficulties of long papers, part of the reason for avoiding proofs of larger programs may be that they are almost invariably incorrect! In order to avoid writing a plausible algorithm and then constructing an equally plausible proof which misses the errors, it is necessary for the proof construction to be extremely detailed. The combination of large programs and detailed proofs makes the whole picture difficult to grasp: any errors that are present become elusive.

Is there an alternative? This paper supports an affirmative answer by going through a formal development of a reasonable-sized algorithm. The steps of development represent a, hopefully, natural evolution from the problem specification to the final program. Each step is stated in a formal notation which makes it possible to construct the proof. Although the overall size may appear large, the structure of the development makes it easy to absorb.

At each stage of development certain assumptions are made which may range from implicit function definitions to properties of data types and their operations. At the same time, functions are also defined further, or properties of data types added. A proof is constructed that, under the new set of assumptions, this stage of development fulfils the earlier assumptions. The new assumptions provide, of course, the specification for the next stage. Thus, at each level, it is only necessary to prove that the hypotheses of the preceding level hold: not to prove again earlier results. Moreover, the development is not a chain of equivalences. It is a completely structured development in which it is possible to think, and prove theorems, about one set of problems at a time.

An example of this development taken from the paper begins with an implicit definition of a set (that is, an object with no order); at the next stage operations are given whose closure on an initial set is shown to satisfy the implicit definition; a mapping of the set onto a list is then given as the next stage and the problems of mimicking the closure are considered.

This paper does not purport to present a worked-out method of formal development, although the author ventures some general comments in the next section. The hope is to encourage further work on a range of examples. Such work should throw light on some of the problems referred to in the discussion of Section 10. In particular there are some areas requiring a much more thorough approach. However, if this is handled correctly further developments should be less difficult than that given below because they can appeal to general theorems.

The body of the paper (Sections 3 through 7 and Section 9), consists of the development of the algorithm. The example chosen was a recogniser using a simplified form of the ideas of Earley. The reader is asked to refer to reference 1 for a description, since such a readable presentation would make detailed exposition in the current paper pointless. Suffice it to say that the algorithm carries out all possible top down parses in parallel, keeping track of them in sets of states. (The simplification made for the current paper was to omit Earley's H_k look-ahead).

As a result of a draft of this paper the question of modifying developed programs was posed: an attempt to address this question is made in Section 8 with an, albeit rather simple, modification.

1. General Comments on Method

As already mentioned, this section is far from presenting a complete general method for Formal Development. It is confined to making some observations resulting from the development contained below.

The most pervasive rule in Formal Development appears to be to define as little as possible at each step. Thus only enough new properties are introduced to give some point to the step. Addition of too much, firstly, clouds the structure of the proof and, secondly, may be regretted. The process consists of adding properties at each step; removal would require backing up to the point of introduction. An example of the addition of properties is the use of (finite) sets of data types until the main lines of the algorithm are clear, at which time the additional properties of ordering are considered.

The actual mode of development used by the author was at each step to sketch a solution using "understood" notation; then to go back and, in proving that step formally, write down exactly what properties were required of the notation. This provides a detailed specification for the next level.

Particular care must be taken over the use of logical operators between (potentially) undefined operands. If, for example, the second operand of a conjunction can be undefined when the first is false, care must be taken to record this fact (see 2-2). Reference 4 discusses the problems of such "three-valued" logic.

There is an important division in the subsequent development between the areas with which proofs are concerned. The proofs of Sections 4 and 5 are exclusively related to the Problem domain, the subsequent ones are related more to the Data and Language domains.

Comments on the process of modification are left to Section 8.

This section reviews the notation used below. Whilst an attempt has been made to minimise the quantity of non-standard notation, use has been made of parts of, so called, VDL (see reference 2 for a fuller treatment). Those parts adopted are defined below in a way which, hopefully, makes this paper self-contained.

FUNCTIONS

Standard notation is used to display functions (e.g. conditional expressions, where clauses) except that:

(2-1) $f : A \rightarrow B$

is taken to imply that f is total over its domain A.

Given a function:

$f : A \times B \rightarrow C$

its specialisation to a particular fixed value in A will be written:

$f_a : B \rightarrow C$

with obvious meaning.

SETS

Normal set notation is used (i.e. $\in, \notin, \subseteq, \cup, \cap, \phi$, $\{x \mid p(x)\}$) and the following sets are assumed:

$N = \{1,2,3,\ldots\}$

$N^o = \{0,1,2,\ldots\}$

$N^n = \{1,2,\ldots,n\}$

The power set (that is, set of all subsets) of S will be written:

$\beta(S) = \{s \mid s \subseteq S\}$

LOGICAL CONNECTIVES

In addition to the normal notation (i.e. $\sim, \wedge, \vee, \supset, \equiv, \exists$), bounded quantifiers:

$(\exists x_{x \in X})(p(x))$

are used where appropriate.

"There exists exactly one object" is written:

$(\exists ! x)(p(x))$ i.e. $(\exists x)(p(x)) \wedge (p(x) \wedge p(y) \supset x=y)$

Under the hypothesis that such a unique object exists it may be denoted using the iota operator:

$(\iota x)(p(x))$

The conditional expression form of "and" (i.e. false if first operand is false even if second is undefined) is written:

(2-2) $a \mathbin{\&} b$

OBJECTS/SELECTORS

Certain sets of elementary objects are given (e.g. N). Composite objects can be viewed as finite tree structures with elementary objects at the terminal nodes. Branches are named with, so called, selectors.

No two branches emanating from the same node may have the same selector name. Subtrees of a given object are called components. As well as naming branches, selectors are used to select components from given objects. Thus:

s(x)

denotes the component of x named s.

Selectors are used in the sequel only when such a component exists. Iterated selection is represented by:

s-2(s-1(x))

Predicates defining classes of objects are written in an obvious way, thus:

(2-3) is-p = (<s-1:is-p-1>,<s-2:is-p-2>,...,<s-n:is-p-n>)

represents a class of objects each with n sub-components etc.

The convention of using the prefixes "is-" and "s-", for predicates and selectors respectively, is followed below. Upper case occurrences of predicate names (excluding the "is-") are used to denote the set of objects satisfying the predicate:

(2-4) $P = \{x \mid \text{is-p}(x)\}$

The selectors of 2-3 can be described as:

(2-5) s-1 : P -> P-1

s-2 : P -> P-2

⋮

s-n : P -> P-n

Certain "obvious extensions" of strict VDL are employed, for example:

(2-6) is-p = is-(<s-1:is-p-1>,<s-2:is-p-2>)

<u>Lists</u> can be considered as objects whose components are named by a subset of the natural number (N^n). However, selection will be shown by conventional subscript notation. Thus:

(2-7) L = |el-1,el-2,...,el-n|

(2-8) $\ell(L) = n$ length of list

(2-9) for $1 \le i \le \ell(L)$: L_i = el-i

The class of objects satisfying <u>is-p-list</u> is the class of all lists whose elements satisfy is-p. Thus:

(2-10) ℓ: P-LIST -> N^o

(2-11) N^n : P-LIST -> P subscripts, where N^n is the set $\{1,2,\ldots,\ell(\text{list})\}$

It is also convenient to have certain lists indexed from zero. The notation:

(2-12) is-p-list^o

is used with obvious meaning.

The class of objects satisfying <u>is-p-set</u> is the class of all finite sets whose members satisfy is-p. Thus:

(2-13) is-p-set denotes a class of finite sets.

(2-14) ϵ: P × β(P) -> {T,F} "is a member of" written as an infix predicate symbol

(2-15) $x \in \{e \mid p(e)\} \equiv p(x)$ implicit set definition

(2-16) $\cup$: S × S -> S set union

such that $x \in (A \cup B) \equiv (x \in A \vee x \in B)$

(2-17) sets cannot contain "duplicate elements"

This section gives the definition of the task to be performed by the algorithm which is developed in subsequent sections. In outline: the class of grammars is defined in 3-1 to 3-4 in the familiar terminal/non-terminal way; the class of strings which can be produced from a given grammar is defined in 3-5 and 3-6; a function "root" is introduced in 3-9 which is assumed to provide the non-terminal from which acceptable strings must be derivable. The recognition task, for given string and grammar, can now be defined in 3-10 using the above notions. A useful Lemma on productions is given in 3-7.

The class of grammars is defined:

(3-1) is-grammar = is-rule-set

which is assumed to have properties 2-13, 2-14

(3-2) is-rule = (<s-lhs:is-nt>,<s-rhs:is-el-list>)

properties 2-5 and 2-10, 2-11 are assumed

$\bar{r}$ will be used as an abbreviation for ℓ(s-rhs(r))

(3-3) is-el = is-nt $\vee$ is-t

(3-4) $NT \cap T = \phi$ non-terminal and terminal sets are disjoint

The following <u>production symbols</u> are defined:

for is-el-list (α), is-el-list(β), is-grammar(G):

(3-5) $\alpha\ A\ \beta \underset{G}{\Rightarrow} \alpha\ \gamma\ \beta \quad \equiv \quad (\exists r_{r \in G})(\text{ s-lhs}(r) = A \wedge \text{s-rhs}(r) = \gamma)$

(3-6) $\alpha \quad \underset{G}{\overset{*}{\Rightarrow}} \beta \quad \equiv \quad (\exists m_{0 \le m})(\exists \alpha_0, \alpha_1, \ldots, \alpha_m)(\alpha = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_m = \beta))$

In the sequel, the grammar can be omitted from the production symbol with no danger of confusion.

The following Lemma can be proved from 3-5, 3-6:

(3-7) <u>Lemma</u>

$$\begin{aligned}\gamma \overset{*}{\Rightarrow} \tau \supset (\exists t_1, \ldots, t_{\ell(\gamma)})(1 \le t_1 \le t_2 \le \ldots \le t_{\ell(\gamma)} &= \ell(\tau) \quad \wedge \\ \gamma_1 \overset{*}{\Rightarrow} \tau_1 \ldots \tau_{t_1} &\quad \wedge \\ &\vdots \\ \gamma_{\ell(\gamma)} \overset{*}{\Rightarrow} \tau_{t_{\ell(\gamma)-1}+1} \ldots \tau_{t_{\ell(\gamma)}} &)\end{aligned}$$

The "specification" of the algorithm to be developed, say rec, can now be stated as follows:

(3-8) rec : T-LIST × GRAMMAR ->{ <u>YES</u>, <u>NO</u> }

(3-9) such that for is-t-list(X) assuming properties 2-10, 2-11

is-grammar (G) see 3-1

root : GRAMMAR -> NT a predefined function

(3-10) rec (X,G) = <u>YES</u> iff root(G) $\overset{*}{\Rightarrow}$ X

<u>NO</u> otherwise

Given that 3-8 requires that rec be total (see 2-1), the second part of 3-10 is not proved explicitly below.

The reader is assumed to be familiar with the concepts of states and state-sets as described by Earley in reference 1. (The simplification in the current paper drops the fourth element of Earley's states.) Collections of state-sets correspond to objects satisfying is-S (see 4-2) from which individual state-sets can be retrieved using the function "state-set". The only properties required of the data object containing state-sets (see 4-3) are the applicability of the predicate "is member of" and the lack of duplicate members. Objects containing states are defined by 4-4 to be triples whose contents will be referred to via given selectors or using the specified abbreviation. More properties of the storage will be added as the algorithm evolves, those given so far are enough to express the first ideas about the algorithm.

It is convenient to assume that there is only one rule of the grammar of which root(G) is the left-hand side. This restriction is made in 4-1 since it has no real effect on the class of valid grammars:

(4-1) $(\exists!_{r \in G}\, r)(\text{s-lhs}(r) = \text{root}(G))$

The object containing all state-sets satisfies:

(4-2) is-S

which is characterised by the existence of two functions:

len : S -> N

state-set : N^n x S -> STATE SET

<u>where</u> N^n is the set {0,1,...,len(S)}

(4-3) is-stateset = is-state-set

of which 2-14 and 2-17 are required to hold.

(4-4) is-state = (<s-rule:is-rule>, <s-j:is-n>, <s-f:is-n>)

of which 2-5 is required to hold.

The abbreviation:

<r,j,f>

is used for the object, say s, for which:

```
s-rule(s) = r
s-j(s)    = j
s-f(s)    = f
```

States will be used to record partial parses. The presence of a particular state, say <r,j,f>, in the ith state-set will mean that a string can be derived from the grammar, G, which consists of the first f characters of X (the string to be recognised) followed by left-hand side of r. It will also mean that the first j elements of the right-hand side of r can produce from G the string $X_{f+1}\ldots X_i$. This property is stated formally in 4-7 below and represents an upper bound for valid state-sets: any state present must possess this property. In order to permit the "rec" algorithm to decide recognition, not all of these states are essential, so 4-6 and 4-8 set a lower bound on valid state-sets: the former specifies a start element which must be present; the latter specifies that if <r,j,f> is in the ith state-set and the next element (that is, j+1 th) of the right-hand side of r can produce $X_{i+1}\ldots X_m$, then <r,j+1,f> must be a member of the mth state-set.

It is fundamental to the style of development proposed by this paper that the freedom to choose which set between the lower and upper bound will be generated, is left for the time being. The properties given permit the development of the first steps of the algorithm (see 4-5) and the proof that, under the assumptions on the creation of *S*, it fulfils the task described in Section 3. This freedom is used to show the correctness of an optimisation modification in Section 8 and could be used to validate many different algorithms including the use of Earley-style look-ahead. Furthermore, the proof of the actual state-sets constructed in Section 5 has been simplified by only having to prove that these looser properties hold.

The definition of a function, alleged to satisfy 3-8, 3-10, can now be given:

(4-5) $$\text{rec }(X,G) = \begin{cases} \langle rr, \overline{rr}, 0\rangle \in \text{state-set }(\ell(x),\mathcal{S}) & \rightarrow \underline{\text{YES}} \\ T & \rightarrow \underline{\text{NO}} \end{cases}$$

$\underline{\text{where}}$ is-S $(\mathcal{S})$

$\underline{\text{and}}$ len $(\mathcal{S}) = \ell(x)$

$\underline{\text{and}}$ $rr = (\iota_{r \in G}\, r)(\text{s-lhs}(r) = \text{root }(G))$

The correctness of this function relies on the following assumptions about $\mathcal{S}$. (These assumptions will, of course, form the definition for further development.)

Some base element must exist in $\mathcal{S}$:

(4-6) $\langle rr,0,0\rangle \in$ state-set $(0,\mathcal{S})$

Any element of a state-set must, at least, satisfy the properties:

(4-7) $0 \le i \le \ell(X)$ & $\langle r,j,f\rangle \in$ state-set $(i,\mathcal{S}) \supset$ $r \in G \wedge$

$(\exists_{\text{is-el-list}(\alpha)}\alpha)(\text{root}(G) \overset{*}{\Rightarrow} X_1 \ldots X_f \text{s-lhs}(r)\alpha) \wedge$

$(j>0 \supset \text{s-rhs}(r)_1 \ldots \text{s-rhs}(r)_j \overset{*}{\Rightarrow} X_{f+1} \ldots X_i) \wedge$

$j \le \overline{r} \quad \wedge$

$0 \le f \le i$

Certain elements must be in the state-sets:

(4-8) $0 \le i < \ell(x)$ & $\langle r,j,f\rangle \in$ state-set$(i,\mathcal{S}) \wedge j \ne \overline{r}$ & $(\exists_{i+1 \le m \le \ell(X)} m)(\text{s-rhs}(r)_{j+1} \overset{*}{\Rightarrow} X_{i+1} \ldots X_m)$

$\supset \langle r,j+1,f\rangle \in$ state-set$(m,\mathcal{S})$

(4-9) $\mathcal{S}$ must be created in a finite amount of time.

Property 4-9 above is given to ensure that any subsequent algorithm generates state-sets in a useful way and permits the proof (see 4-13) that 4-5 is total. After a trivial Lemma (4-10) on the finiteness of state-sets, the proofs that both implications of the first part of 3-10 hold are given in 4-11 and 4-12. Of course, these proofs all rely on the assumptions that $\mathcal{S}$ satisfies 4-6 to 4-9.

(4-10) Lemma

for $0 \le i \le \text{len}(\mathcal{S})$: state-set $(i,\mathcal{S})$ is finite.

proof

a) G is finite — 3-1,2-13

b) $r \in G \supset \overline{r} \in N$ — 3-2,2-10

i.e. is finite

c) Thus, since the cross-product of finite sets is also finite, the Lemma is proved. — a,b,4-7,4-3,2-17

(4-11) Theorem

rec $(X,G) = \underline{\text{YES}} \supset \text{root}(g) \overset{*}{\Rightarrow} X$

proof

a) assume : rec $(X,G) = \underline{\text{YES}}$

b) let : $rr = (\iota r)(\text{s-lhs}(r) = \text{root}(G))$

notice that use of iota is valid — 4-1

c)	then	$: \langle rr, \overline{rr}, 0\rangle \in$ state-set $(\ell(x), \mathcal{S})$	a,4-5
d)		s-rhs$(rr)_1 \ldots$ s-rhs$(rr)_{\overline{rr}} \overset{*}{\Rightarrow} X_1 \ldots X_{\ell(X)}$	c,4-7
e)		s-lhs$(rr) \overset{*}{\Rightarrow} X_1 \ldots X_{\ell(X)}$	d,3-6
f)		root$(G) \overset{*}{\Rightarrow} X$	e, b

(4-12) Theorem

root$(G) \overset{*}{\Rightarrow} X \supset$ rec(X,G) = <u>YES</u>

proof

a)	assume:	root$(G) \overset{*}{\Rightarrow} X$	
b)	let	$: rr = (\iota r)($s-lhs$(r) =$ root$(G))$	4-1
c)		s-rhs$(rr) \overset{*}{\Rightarrow} X_1 \ldots X_{\ell(X)}$	a,b,3-6
d)		$(\exists t_1, \ldots, t_{\overline{rr}})(1 \leq t_1 \leq t_2 \leq \ldots \leq t_{\overline{rr}} = \ell(X) \wedge$ s-rhs$(rr)_1 \overset{*}{\Rightarrow} X_1 \ldots X_{t_1} \wedge \; \vdots \;$ s-rhs$(rr)_{\overline{rr}} \overset{*}{\Rightarrow} X_{t_{\overline{rr}-1}+1} \ldots X_{\ell(X)})$	c, 3-7
e)		$\langle rr, 0, 0\rangle \in$ state-set $(0, \mathcal{S})$	4-6
using $\overline{rr}$ applications of			4-8
f)		$\langle rr, 1, 0\rangle \in$ state-set $(t_1, \mathcal{S})$ $\;\vdots\;$ $\langle rr, \overline{rr}, 0\rangle \in$ state-set$(\ell(X), \mathcal{S})$	e,d
g)		rec (X,G) = <u>YES</u>	f,4-5

(4-13) Theorem

rec is total

proof

state-set is used only on given domain	4-5,4-2
s $\in$ stateset is used only on given domain	4-5,4-3,2-14
the use of iota is valid	4-5,4-1
Furthermore:	
$\mathcal{S}$ is created in finite time	4-9
state-set $(\ell(X), \mathcal{S})$ is finite	4-10
thus membership test takes finite time	
Therefore rec terminates	

Section 4 introduced the concept of state-sets and some essential properties thereof, but offered no way of creating them. In fact ℓ(s-rhs(rr)) entries, presumably created by magic, would have been sufficient to fulfil the properties stated. This section shows how state-sets can be constructed using the main ideas (that is, prediction, completion, scanning) of Earley's method. In order to facilitate construction of the required elements, many other states are generated. It is interesting to consider the parallel with proof construction where an induction hypothesis stronger than the theorem statement is used in order to prove the latter.

First, the definition of rr which was in the last section a local convention, is adopted for the sequel.

(5-0) $\quad rr = (\iota r)_{r\in G}(\text{s-lhs}(r) = \text{root}(G))$ $\qquad$ see 4-1

Earley's operations of prediction etc, are, of course, described in terms of lists of states. As will be clear shortly, many of the interesting points about these operations can be brought out using sets of states. Again it is found that deferring part of the detail, in this case the special ordering problems imposed by using lists, will structure and clarify the proofs.

It will be necessary to use implicit set notation for creation of state-sets so now the restriction:

(5-1) $\quad$ Property 2-15 holds for is-stateset
is added.

Using sets of states which are cumulative, prompts the idea of using the notion of the closure of a set under an operation as generating function. Intuitively the closure of an operation on a set is the minimum set containing the base set and having the property that applying the operation to any element creates only elements already in the set. We shall use the following properties:

for op : $P \rightarrow \beta(P)$ $\qquad$ i.e. from elements of P to sets of such elements

(5-2) $\quad$ closure : $(op) \times \beta(P) \rightarrow \beta(P)$ $\qquad$ i.e. from sets of elements to sets of elements

such that for $S \subseteq P$:

$$x \in (\text{closure of op on } S) \equiv x \in S \vee (\exists y)_{y\in S}(x \in \text{closure of op on } (op(y)))$$

Notice that closure is total for suitable op.

It should be clear that closure is monotonic. That is:

(5-3) $\quad S_1 \subseteq S_2 \supset (\text{closure of op on } S_1) \subseteq (\text{closure of op on } S_2)$

Using closure it is now possible to show how the basic operations (that is, prediction etc) are used to create the state-sets. The base element is inserted by 5-4 which also shows that prediction is the only applicable operation on the 0 th state-set. The creation, for all other state-sets, of a kernel set of states resulting from the scanning operation on the preceding state-set is specified in 5-5. Both the prediction and completion operations are employed on all state-sets but the last, where the former is not required. The closure is created from the kernel set described. The importance of the reliance by the completer on $\mathcal{S}$ (strictly on a part of $\mathcal{S}$) is returned to in the next section.

(5-4)† $\quad \text{state-set}(0,\mathcal{S}) \;=\; \text{closure of predict}_{0,X,G} \text{ on } \{<rr,0,0>\}$

(5-5) $\quad$ for $1\le i\le \ell(X)$:

$$\text{state-set}(i,\mathcal{S})^{s} \;=\; \text{scan}(i,X,G,\text{state-set}(i-1,\mathcal{S}))$$

(5-6) $\quad$ for $1\le i<\ell(X)$:

$$\text{state-set}(i,\mathcal{S}) \;=\; \text{closure of} \begin{Bmatrix} \text{predict}_{i,X,G} \\ \text{complete}_{\mathcal{S},i,X,G} \end{Bmatrix} \text{ on state-set}(i,\mathcal{S})^{s}$$

(5-7) $\quad \text{state-set}(\ell(X),\mathcal{S}) = \text{closure of complete}_{\mathcal{S},i,X,G} \text{ on state-set}(\ell(X),\mathcal{S})^{s}$

† Notice use of function specialisation, see Section 2.

The definitions of the basic operations of prediction, scanning and completion can now be given. For reasons discussed in Section 1, it is important to guard against predicates becoming undefined by careless use of "$\wedge$" which is not normally defined for three valued logic.

(5-8) $\text{predict}(i,X,G,\langle r,j,f\rangle) = \{\langle s,0,i\rangle \mid j\neq\bar{r}\ \&\ \text{is-nt}(\text{s-rhs}(r)_{j+1})\ \&\ s\in G\ \&\ \text{s-lhs}(s)=\text{s-rhs}(r)_{j+1}\}$

(5-9) $\text{scan}(i,X,G,ss) = \{\langle r,j+1,f\rangle \mid \langle r,j,f\rangle\in ss \wedge j\neq\bar{r}\ \&\ \text{is-t}(\text{s-rhs}(r)_{j+1})\ \&\ X_i = \text{s-rhs}(r)_{j+1}\}$

(5-10) $\text{complete}(\mathcal{S},i,X,G,\langle r,j,f\rangle) =$

$\{\langle s,m+1,g\rangle \mid j=\bar{r}\ \&\ \langle s,m,g\rangle\in\text{state-set}(f,\mathcal{S})\ \&\ m\neq\bar{s}\ \&\ \text{is-nt}(\text{s-rhs}(s)_{m+1})\ \&\ \text{s-rhs}(s)_{m+1} = \text{s-lhs}(r)\}$

In order to resolve any possible difficulties deriving from too heavy a reliance on formalism the example used in reference 1 is now presented in the terms of the current paper. (Notice the addition of R to satisfy 4-1).

$G = \{r1,\ r2,\ r3,\ r4,\ r5,\ r6\}$

<u>where</u> $r1 = \langle\langle \text{s-lhs} : R\rangle, \langle \text{s-rhs} : [E]\rangle\rangle$

$r2 = \langle\langle \text{s-lhs} : E\rangle, \langle \text{s-rhs} : [T]\rangle\rangle$

$r3 = \langle\langle \text{s-lhs} : E\rangle, \langle \text{s-rhs} : [E,+,T]\rangle\rangle$

$r4 = \langle\langle \text{s-lhs} : T\rangle, \langle \text{s-rhs} : [P]\rangle\rangle$

$r5 = \langle\langle \text{s-lhs} : T\rangle, \langle \text{s-rhs} : [T,*,P]\rangle\rangle$

$r6 = \langle\langle \text{s-lhs} : P\rangle, \langle \text{s-rhs} : [a]\rangle\rangle$

such that is-nt(R), is-nt(E), is-nt(T), is-nt(P), is-t(+), is-t(*), is-t(a)

Consider the recognition of:

$X = [a,+,a]$

Let rr = r1

$\text{state-set}(0,\mathcal{S})$ = closure of predict on $\{\langle r1,0,0\rangle\}$

= closure of predict on $\{\qquad\qquad ,\langle r2,0,0\rangle\}$

= closure of predict on $\{\qquad\qquad\qquad\qquad ,\langle r4,0,0\rangle,\langle r5,0,0\rangle\}$

= closure of predict on $\{\qquad\qquad\qquad\qquad\qquad\qquad\qquad ,\langle r6,0,0\rangle\}$

= $\{\langle r1,0,0\rangle,\langle r2,0,0\rangle,\langle r4,0,0\rangle,\langle r5,0,0\rangle,\langle r6,0,0\rangle\}$

$\text{state-set}(1,\mathcal{S})^s$ = scan $(\text{state-set}(0,\mathcal{S}))$

= $\{\langle r6,1,0\rangle\}$

$\text{state-set}(1,\mathcal{S})$ = closure of predict/complete on $\text{state-set}(1,\mathcal{S})^s$

= closure of predict/complete on $\{\langle r6,1,0\rangle,\langle r4,1,0\rangle,\langle r5,1,0\rangle\}$

= closure of predict/complete on $\{\qquad\qquad\qquad\qquad\qquad\qquad ,\langle r2,1,0\rangle,\langle r3,1,0\rangle\}$

= closure of predict/complete on $\{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ,\langle r1,1,0\rangle\}$

= $\{\langle r6,1,0\rangle,\langle r4,1,0\rangle,\langle r5,1,0\rangle,\langle r2,1,0\rangle,\langle r3,1,0\rangle,\langle r1,1,0\rangle\}$

$\text{state-set}(2,\mathcal{S})^s$ = $\text{scan}(\text{state-set}(1,\mathcal{S}))$

= $\{\langle r3,2,0\rangle\}$

$\text{state-set}(2,\mathcal{S})$ = closure of predict/complete on state-set $(2,\mathcal{S})^s$

= closure of predict/complete on $\{\langle r3,2,0\rangle,\langle r4,0,2\rangle,\langle r5,0,2\rangle\}$

= closure of predict/complete on $\{\qquad\qquad\qquad\qquad\qquad\qquad ,\langle r6,0,2\rangle\}$

= $\{\langle r3,2,0\rangle,\langle r4,0,2\rangle,\langle r5,0,2\rangle,\langle r6,0,2\rangle\}$

$\text{state-set}(3,\mathcal{S})^s$ = scan $(\text{state-set}(2,\mathcal{S}))$

= $\{\langle r6,1,2\rangle\}$

state-set(3,S) = closure of complete on state-set(3,S)^s

= closure of complete on {<r6,1,2>,<r4,1,2>}

= closure of complete on { ,<r3,3,0>,<r5,1,2>}

= closure of complete on { ,<r1,1,0>}

= {<r6,1,2>,<r4,1,2>,<r3,3,0>,<r5,1,2>,<r1,1,0>}

Referring to 4-5 gives:

rec (X,G) = <u>YES</u>

The algorithm represents a way of generating some set of states between those specified by 4-8 and 4-7. The justification† consists of proving this containment (see 5-12, 5-13) and establishing 4-6 and 4-9 (see 5-11, 5-14 respectively). Notice that this is all! It is not necessary to prove again any of the results dealt with in Section 3.

(5-11) Theorem

<rr,0,0> $\in$ state-set(0,S)

proof

follows immediately from 5-4, 5-3.

(5-12) Theorem

$0 \le i \le \ell(X)$ & $<r,j,f> \in$ state-set(i,S) $\supset r \in G \wedge$

$(\exists \alpha_{\text{is-el-list}(\alpha)})(\text{root}(G) \overset{*}{\Rightarrow} X_1 \ldots X_f \text{ s-lhs}(r)\alpha) \wedge$

$(j>0 \supset \text{s-rhs}(r)_1 \ldots \text{s-rhs}(r)_j \overset{*}{\Rightarrow} X_{f+1} \ldots X_i) \wedge$

$j \le \bar{r} \wedge$

$0 \le f \le i$

proof: shows that the set satisfying the above is a fixed point of 5-4 to 5-7 (i.e. that given such a set the operations create only elements of that set). Use, without reference, is made of 5-3 below to argue that once inserted, elements are not lost.

a)	consider <rr,0,0>:	see 5-4
b)	rr $\in$ G	5-0
c)	root(G) $\overset{*}{\Rightarrow}$ s-lhs(rr)	5-0,3-6
d)	j = 0	a
e)	$0 \le \overline{rr}$	3-2,2-10
f)	$0 \le 0 \le 0$	
g)	thus <rr,0,0> is a member of the set satisfying 5-12.	b,c,d,e,f
h)	consider predict(i,X,G,<r,j,f>) on <r,j,f> $\in$ {set satisfying 5-12}: see	5-4,5-6
i)	r $\in$ G	h,5-12
j)	$(\exists \alpha_{\text{is-el-list}(\alpha)})(\text{root}(G) \overset{*}{\Rightarrow} X_1 \ldots X_f \text{ s-lhs}(r)\alpha)$	h,5-12
k)	$(j>0 \supset \text{s-rhs}(r)_1 \ldots \text{s-rhs}(r)_j \overset{*}{\Rightarrow} X_{f+1} \ldots X_i)$	h,5-12
l)	$j \le \bar{r}$	h,5-12

† It should be possible to construct a direct proof that a set given by 4-7, with equivalence replacing implication, is the minimum fixed point (see ref 3) of 5-4 to 5-10. The current author's attempts have foundered on the ordering problems caused by the completer.

```
m)            $0 \le f \le i$                                                        h,5-12
n)        case $j = \bar{r}$:
o)          no new elements created                                                  n,5-8
p)        case $j \ne \bar{r}$:
q)          case ~ is-nt(s-rhs(r)$_{j+1}$):
r)              no new elements created                                              q,5-8
s)          case is-nt(s-rhs(r)$_{j+1}$):
t)              let s be a member of G  such that s-lhs(s) = s-rhs(r)$_{j+1}$
u)        s $\in$ G                                                                  t
v)        $(\exists_{\text{is-el-list}(\alpha)}\ \alpha)$(root(G) $\overset{*}{=>}$ X$_1$...X$_f$X$_{f+1}$...X$_i$ s-lhs(s)$\alpha$)     j,k,3-6,t
w)        $0 \le i \le i$
x)        Thus <s,0,i> is a member of the set satisfying 5-12                        u,v,w
y)      Thus the predict operation stays within the set satisfying 5-12              o,r,x
z)      consider scan (i,X,G,ss):                                                 see 5-5
aa)       let <r,j,f> be a typical element of ss (i.e. of state-set(i-1,S))          assm
ab)           r $\in$ G                                                              aa,5-12
ac)           $(\exists_{\text{is-el-list}(\alpha)}\ \alpha)$(root(G) $\overset{*}{=>}$ X$_1$...X$_f$ s-lhs(r)$\alpha$)     aa,5-12
ad)           (j>0 $\supset$ s-rhs(r)$_1$...s-rhs(r)$_j$ $\overset{*}{=>}$ X$_{f+1}$...X$_{i-1}$)    aa,5-12
ae)           $j \le \bar{r}$                                                        aa,5-12
af)           $0 \le f \le i-1$                                                      aa,5-12
ag)       case $j = \bar{r}$:
ah)           no new elements created                                                ag,5-9
ai)       case $j \ne \bar{r}$:
aj)           case ~(is-t(s-rhs(r)$_{j+1}$) & X$_i$ = s-rhs(r)$_{j+1}$)
ah)               no new elements created                                            aj,5-9
al)           case is-t(s-rhs(r)$_{j+1}$) $\wedge$ X$_i$ = s-rhs(r)$_{j+1}$
am)               is-rhs(r)$_1$...s-rhs(r)$_{j+1}$ $\overset{*}{=>}$ X$_{f+1}$...X$_i$      ad,al,3-6
an)               $j+1 \le \bar{r}$                                                  ae,ai
ao)               Thus <r,j+1,f> is a member of the set with index i
                                                  satisfying 5-12                    ab,ac,am,an,af
ap)     Thus the scan function stays within the set satisfying 5-12                  ah,ak,ao
aq)     consider  complete (S,i,X,G,<r,j,f>) on <r,j,f> $\in$ set satisfying 5-12  see 5-6,5-7
ar)         r $\in$ G                                                                aq,5-12
as)         $(\exists_{\text{is-el-list}(\alpha)}\ \alpha)$(root(G) $\overset{*}{=>}$ X$_1$...X$_f$ s-lhs(r)$\alpha$)       aq,5-12
at)         j>0 $\supset$ s-rhs(r)$_1$...s-rhs(r)$_j$ $\overset{*}{=>}$ X$_{f+1}$...X$_i$      aq,5-12
```

au)	$j \le \bar{r}$	aq,5-12
av)	$0 \le f \le i$	aq,5-12
aw)	case $j \ne r$:	
ax)	no new elements created	aw,5-10
ay)	case $j = r$:	
az)	let $\langle s,m,g \rangle$ be a member of state-set$(f,\mathcal{S})$ such that	
ba)	$m \ne \bar{s}$	assm
bb)	is-nt$(\text{s-rhs}(s)_{m+1})$	assm
bc)	$\text{s-rhs}(s)_{m+1} = \text{s-lhs}(r)$	assm
bd)	$s \in G$	az,5-12
be)	$(\underset{\text{is-el-list}(\beta)}{\exists\beta})(\text{root}(G) \overset{*}{\Rightarrow} X_1 \ldots X_g\ \text{s-lhs}(s)\beta)$	az,5-12
bf)	$m > 0 \supset \text{s-rhs}(s)_1 \ldots \text{s-rhs}(s)_m \overset{*}{\Rightarrow} X_{g+1} \ldots X_f$	az,5-12
bg)	$m \le \bar{s}$	az,5-12
bh)	$0 \le g \le f$	az,5-12
bi)	$\text{s-rhs}(s)_1 \ldots \text{s-rhs}(s)_m\ \text{s-rhs}(s)_{m+1} \overset{*}{\Rightarrow} X_{g+1} \ldots X_f X_{f+1} \ldots X_i$	bf,bc,at,ay,3-6
bj)	$m+1 \le \bar{s}$	ba,bg
bk)	$0 \le g \le i$	bh,av
bl)	Thus $\langle s,m+1,g \rangle$ is a member of the set satisfying 5-12	bd,be,bi,bj,bk
bm)	Thus the complete operation stops within the set satisfying 5-12	ax,bl
	This completes the proof.	5-4,5-5,5-6,5-7,g,y,ap,bm

(5-13) Theorem

$$0 \le i < \ell(X)\ \&\ \langle r,j,f \rangle \in \text{state-set}(i,\mathcal{S}) \wedge j \ne \bar{r}\ \&\ (\underset{i+1 \le m \le \ell(X)}{\exists m})(\text{s-rhs}(r)_{j+1} \overset{*}{\Rightarrow} X_{i+1} \ldots X_m)$$

$$\supset \langle r,j+1,f \rangle \in \text{state-set}(m,\mathcal{S})$$

proof by induction on n (the number of $\Rightarrow$ steps in $\overset{*}{\Rightarrow}$) see 3-5,3-6

a)	$0 \le i < \ell(X)$	assm
b)	$\langle r,j,f \rangle \in \text{state-set}(i,\mathcal{S})$	assm
c)	$j \ne \bar{r}$	assm
d)	$(\underset{i+1 \le m < \ell(X)}{\exists m})(\text{s-rhs}(r)_{j+1} \overset{*}{\Rightarrow} X_{i+1} \ldots X_m)$	assm
e)	basis zero steps:	
f)	$\text{s-rhs}(r)_{j+1} = X_{i+1}$	e,3-6,d
g)	is-t$(\text{s-rhs}(r)_{j+1})$	f,3-10
h)	$\langle r,j+1,f \rangle \in \text{state-set}(i+1,\mathcal{S})$	5-5,5-9,b,c,g,f
i)	which covers this case	
j)	induction from generation sequences up to length n-1 to sequences of length n:	
k)	$n > 0$	j

l) $\text{s-rhs}(r)_{j+1} \Rightarrow \alpha_o \overset{*}{\Rightarrow} X_{i+1} \ldots X_m$ 3-6,k

m) $\text{is-nt}(\text{s-rhs}(r)_{j+1})$ 3-5,l

n) $i \neq \ell(X)$ 1,3-10,2-11

o) $(\exists s_{s \in G})(\text{s-lhs}(s) = \text{s-rhs}(r)_{j+1} \wedge \text{s-rhs}(s)_1 \ldots \text{s-rhs}(s)_{\bar{s}} \overset{*}{\Rightarrow} X_{i+1} \ldots X_m)$ 3-5,l

p) let s be such a rule:

q) $\langle s,0,i \rangle \in \text{state-set}(i,\mathcal{S})$ 5-6,5-8,c,m,p,o

r) $(\exists t_1, \ldots t_{\bar{s}})(i+1 \leq t_1 < t_2 \leq \ldots \leq t_{\bar{s}} = m \wedge$ o,3-7

$$\text{s-rhs}(s)_1 \overset{*}{\Rightarrow} X_{i+1} \ldots X_{t_1} \wedge$$
$$\vdots$$
$$\text{s-rhs}(s)_{\bar{s}} \overset{*}{\Rightarrow} X_{t_{\bar{s}-1}+1} \ldots X_{t_{\bar{s}}})$$

Applying induction hypotheses $\bar{s}$ times (productions less than n long) j

s) $\langle s,1,i \rangle \in \text{state-set}(t_1,\mathcal{S})$ r,5-13

$\vdots$

$\langle s,\bar{s},i \rangle \in \text{state-set}(m,\mathcal{S})$

t) $\langle r,j+1,f \rangle \in \text{state-set}(m,\mathcal{S})$

which completes the proof s,5-6,5-7,5-10,s,b,c,m,o,i,t

(5-14) Theorem

$\mathcal{S}$ is created in finite time

proof

The finiteness of the closures follows immediately from 4-10 and 5-12.
References to elements of $\left.\begin{array}{r}\text{s-rhs}\\ \text{string}\end{array}\right\}$ lists are all within bounds 2-2,5-8,5-9,5-10

state-set(i,$\mathcal{S}$) is used within bounds 4-7,5-10

So far state-sets have been assumed to be set-like objects and they have been created by closure. Whilst it would be possible to model this state of affairs in a conventional store it would be much more convenient if we could show how the sets can be mapped onto lists. The lists will obviously have extra properties, in particular an ordered property. If closure can now be replaced by a new, more efficient, operation which relies on this ordering, an increase in efficiency results. This section makes precisely this mapping. Subject to a restriction on grammars (see 6-9), the extension is performed by a single scan over the lists. It is, in the author's opinion, worthwhile understanding the cause of the restriction. It would for instance be a relatively simple task, <u>at this level</u>, to avoid introducing the restriction by utilising a more sophisticated scan.

So far the only properties assumed of state-sets are those required by 4-3 and 5-1. That is:

$\in$: STATE×STATESET -> {T,F}

$x \in \{state \mid p(state)\} \equiv p(x)$

state-sets do not contain duplicates

This section describes how these objects can be mapped onto lists. Thus:

(6-1) is-stateset = is-state-list

assuming properties 2-10 and 2-11

Now the membership and implicit set definitions are reinterpreted as follows:

(6-2) $e \in s$ becomes $(\exists i_{1 \le i \le \ell(s)})(s_i = e)$ but will be abbreviated to $e \underline{\in} s$

(6-3) $\{s \mid p(s)\}$ becomes create-state-list(p) but will be abbreviated to $\underline{\{}s \underline{|} p(s)\underline{\}}$

with the following properties:

let $\ell(\underline{\{}s\underline{|}p(s)\underline{\}}) = n$

$p(x) \supset (\exists ! j_{1 \le j \le n})((\underline{\{}s\underline{|}p(s)\underline{\}}_j = x)$

$1 \le j \le n \supset p(\underline{\{}s\underline{|}p(s)\underline{\}})_j$

A new operation will be required to manipulate state-sets.

(6-4) combine $(list_1, list_2)$ which will be written $\underline{\cup}$

with the following properties:

$s \underline{\in} (list_1 \ \underline{\cup}\ list_2) \equiv s \underline{\in} list_1 \vee s \underline{\in} list_2$

$(\exists j,k_{1 \le j,k \le \ell(list_1 \underline{\cup} list_2)})((list_1 \ \underline{\cup}\ list_2)_j = (list_1 \ \underline{\cup}\ list_2)_k \supset j = k)$

$1 \le i \le \ell(list_1) \supset (list_1 \ \underline{\cup}\ list_2)_i = (list_1)_i$

The properties required of state-sets can now be verified:

(6-5) Lemma

a) $\underline{\in}$: STATE × STATESET -> {T,F}

b) $x \underline{\in} \underline{\{}state \underline{|} p(state)\underline{\}} \equiv p(x)$

c) state-sets contain no duplicates

proofs

a) immediate from 6-2,6-1

b) immediate from 6-2, 6-1 and 6-3

c) true if basic lists contain no duplicates (which includes implicit definition) and all combinations made with $\underline{\cup}$ 6-3, 6-4

Operating on lists makes it desirable to replace the closure operations (see 5-2) with linear scans. It will be found necessary to introduce a restriction on grammars in order to fulfil property 5-2.

The construction of $\mathcal{S}$ is now given by:

(6-6) $\mathcal{S} = S^z$ <u>where</u> next $(z+1) =$ <u>end</u>

(6-7) next $(1) = (0,1)$

for $i>1$:

$$\text{next }(i) = \ell(\text{state-set}(j,S^{i-1})) = k \rightarrow j \neq \ell(X)\ \&\ \text{state-set}(j+1,S^{i-1}) \neq [\,] \rightarrow (j+1,1)$$

$$\begin{array}{lll} & \mathrm{T} & - \ \underline{\text{end}} \\ \mathrm{T} & \rightarrow (j,k+1) & \end{array}$$

<u>where</u> $(j,k) = \text{next }(i-1)$

(6-8) $\text{state-set}(0,S^0) = [\langle rr,0,0\rangle]$

for $0<i\leq\ell(X)$:

$\text{state-set}(i,S^0) = [\,]$

for $k\geq 0$:

$$j \neq \bar{r} \wedge i<\ell(X)\ \&\ \text{is-nt}(\text{s-rhs}(r)_{j+1}) \supset$$
$$\text{state-set}(i,S^{k+1}) = \text{state-set}(i,S^k)\,\underline{\cup}\ \underline{\{}s\underline{|}s\in\text{predict}(i,X,G,\langle r,j,f\rangle)\underline{\}}$$

$$j \neq \bar{r} \wedge i<\ell(X)\ \&\ \text{is-t}(\text{s-rhs}(r)_{j+1})\ \&\ \text{s-rhs}(r)_{j+1} = X_{i+1} \supset$$
$$\text{state-set}(i+1,S^{k+1}) = \text{state-set}(i+1,S^k)\,\underline{\cup}\underline{\{}s|s\in\text{scan}(i+1,X,G,\langle r,j,f\rangle)\underline{\}}$$

$$j = \bar{r} \supset \text{state-set}(i,S^{k+1}) = \text{state-set}(i,S^k)\,\underline{\cup}\underline{\{}s\underline{|}s\in\text{complete}(S^k,i,X,G,\langle r,j,f\rangle)\underline{\}}$$

$$\text{otherwise}\quad \text{state-set}(p,S^{k+1}) = \text{state-set}(p,S^k)$$

<u>where</u> $\langle r,j,f\rangle = \text{state-set}(i,S^k)_n$

<u>where</u> $(i,n) = \text{next }(k+1)$

The "algorithm" as shown would not work for all possible grammars! In particular, if the grammar includes rules which generate the empty string, the sequential pass over state-sets, coupled with the fact that although created many times only one copy of a state appears in a state-set, prevents the completer creating the full closure.

Consider $G = \{r1,r2\}$

<u>where</u> $r1 = \langle\langle\text{s-lhs} : R\rangle,\langle\text{s-rhs} : [a,E,E,a]\rangle\rangle$

$r2 = \langle\langle\text{s-lhs} : E\rangle,\langle\text{s-rhs} : [\,]\rangle\rangle$

$X = [a,a]$

$\text{state-set}(0,\mathcal{S}) = [\langle r1,0,0\rangle]$

$\text{state-set}(1,\mathcal{S}) = [\langle r1,1,0\rangle,\langle r2,0,1\rangle,\langle r1,2,0\rangle]$

and no more elements will be appended to the list.

The following restriction on grammars is made:

(6-9) $r \in G \supset \bar{r} > 0$

It should be clear that Lemma 3-7 now becomes:

(6-10) $$\gamma \overset{*}{\Rightarrow} \tau \supset (\exists t_1,\ldots,t_{\ell(\gamma)})(1\le t_1<t_2<\ldots<t_{\ell(\gamma)}=\ell(\tau) \wedge \gamma_1 \overset{*}{\Rightarrow} \tau_1\ldots\tau_{t_1} \wedge \gamma_{\ell(\gamma)} \overset{*}{\Rightarrow} \tau_{t_{\ell(\gamma)-1}+1}\ldots\tau_{t_{\ell(\gamma)}})$$

The proof that the state-sets of Section 5 and the state lists of this section contain the same elements (assuming 6-9) is now given in 6-11. Since closure was considered total (see 5-2) it is also necessary to show that the scanning which replaced it is total. Notice that, once again, the theorems revolve around only those things which have been changed, and the earlier theorems still apply because properties are only enhanced.

(6-11) Theorem

$s \underset{\sim}{\in}$ state-set(i,S^z)	see 6-8
iff	
$s \in$ state-set$(i,\mathcal{S})$	see 5-4,5-5,5-6,5-7

proof

The only differences in closure and the single scan is that:

1) Former applies all operations to all elements

2) Regardless of order of creation, all combinations are considered

But 1) Inspection of 5-8,5-9,5-10 shows that at most one operation can be applicable to any particular state

2) The only operation relying on combinations is the completer 5-8,5-9,5-10
The f state set will always be earlier than current scan (f<i) 6-10
Since next completes a state-set before proceeding to subsequent ones 6-7
The differences do not change the elements created

Therefore the sets are equivalent.

Because 5-2 claimed completeness for closure, it is also necessary to prove:

(6-12) Theorem

The creation of $\mathcal{S}$ given in 6-6 to 6-8 is total

proof

state-set is used within given bounds	6-7,6-8
ℓ is used within given bounds	6-7,6-8
s-rhs, X } lists are used within given bounds	6-8
next must terminate	6-7,4-10

7. Structure for Store

The preceding section represented a solution to the main parts of the recognition task. However, for a number of reasons it is not a program which can be run on a computer. The principal point is that the algorithm of Section 6 uses a storage object which cannot be represented in any known language, for example the grammar is just "referred to" whenever required. It might be possible to sit at a console and supply rules whenever needed, but a more useful program will result if the first step involves reading in both the grammar and string to be recognised in such a way that subsequent references are simple. The particular storage structure given below has been adapted from an actual program written in the conventional way and designed with a view to efficiency. This decision was taken so as to avoid the danger of choosing a data structure particularly suited to the proof but which put limitations on the potential efficiency of the final algorithm.

The input of grammar and string will be assumed to be carried out by routines called INGR and INSTR respectively. The grammar is stored in the s-RULES, s-NONT, s-RULED, s-STR and s-STRCHAR; the string in the s-INPUT components of the store given in 7-1 to 7-7 below. What were, in the algorithm, references to the original arguments, now become references to these storage components and the reinterpretations to be made are specified in 7-9 to 7-17.

The object is-S is also refined in this section and now occupies the s-S and s-STATE parts of 7-1 to 7-7.

The store is still presented as a VDL-style object and a further transition to PL/I Data Structures will be made later. The decision to introduce this step is perhaps questionable, but is probably justified since the object of 7-1 could be realised in other languages. The only properties required are those in 7-8 and although "bunches" were used with PL/I-based storage in mind, the only objection to an array-style implementation with numeric "selectors" would be that all elements would have to be of the same length.

Store is considered as an object satisfying:

(7-1) is-ξ =(<s-S : is-state-ptr-list^{o}>,
<s-STATE : is-STATE-bunch>,
<s-INPUT : is-t-list>,
<s-RULES : is-(<s-TYPE : is-T $\vee$ is-N>,
<s-N : is-n>) - list>,
<s-NONT : is-ruled-ptr-list>,
<s-RULED : is-RULED-bunch>,
<s-STR : is-(<s-START : is-n>)-list>,
<s-STRCHAR : is-char-list>)

<u>where</u>:

(7-2) is-state-ptr = is-sel

(7-3) is-STATE-bunch = {<κ : state> || is-sel(κ) $\wedge$ is-STATE(state)}

(7-4) is-STATE = (<s-SIZE : is-n>,
<s-N : is-n>,
<s-INFO : is-(<s-RULE : is-(<s-RPTR : is-ruled-ptr>,
<s-SSC : is-n>)>,
<s-RULPOS : is-n>,
<s-STRPOS : is-n>)-list>)

(7-5) is-ruled-ptr = is-sel

(7-6) is-RULED-bunch = {<κ : ruled> || is-sel(κ) $\wedge$ is-RULED(ruled)}

(7-7) is-RULED = (<s-NEXT : is-n>,

<s-RULE : is-(<s-START : is-n>,

<s-LENGT : is-n>)-list>)

(7-8) In all objects of 7-1 to 7-7:

selectors are assumed to have property 2-5

lists are assumed to have properties 2-10,2-11

Using the above data object as storage the algorithm can now use the following ways of referring to its arguments:

(7-9) ℓ(X) becomes ℓ(s-INPUT(ξ))

(7-10) X_i becomes $\text{s-INPUT}(\xi)_i$

Now, INGR defines a relation between rules of is-grammar and objects satisfying is-(<s-RPTR : is-ruled-ptr>,<s-SSC : is-n>) which is written:

$\mathcal{I}$ (r,(p,i))

(7-11) $\left.\begin{array}{l}\bar{r}\\ \ell(\text{s-rhs}(r))\end{array}\right\}$ becomes $\text{s-LENGT (s-RULE(p(s-RULED}(\xi)))_i\text{)}$

(7-12) $\text{s-rhs}(r)_j$ becomes $\text{s-RULES}(\xi)_{\text{s-START (s-RULE(p(s-RULED}(\xi)))_i)+j-1}$

for convenience, let $\mathcal{I}(\text{s-rhs}(r)_j)$ be an abbreviation for $\text{s-RULES}(\xi)_{\text{s-START(s-RULE(p(s-RULED}(\xi)))_i)+j-1}$

(7-13) $\text{is-t(s-rhs}(r)_j)$ becomes $\text{s-TYPE}(\mathcal{I}(\text{s-rhs}(r)_j)) = T$

(7-14) $\text{is-nt(s-rhs}(r)_j)$ becomes $\text{s-TYPE}(\mathcal{I}(\text{s-rhs}(r)_j)) = N$

(7-15) for $\text{is-t(s-rhs}(r)_j)$:

$\text{s-rhs}(r)_j$ becomes $\text{s-STRCHAR}(\xi)_{\text{s-N}(\mathcal{I}(\text{s-rhs}(r)_j))}$

(7-16) for $\text{is-nt(s-rhs}(r)_j)$:

$\text{s-rhs}(r)_j = \text{s-lhs}(s)$ becomes $\text{s-NONT}(\xi)_{\text{s-N}(\mathcal{I}(\text{s-rhs}(r)_j))} = q$

<u>where</u> $\mathcal{I}$(s,(q,k))

(7-17) $r \in G$ becomes is-RULED(p(s-RULED(ξ))) $\wedge$ i<s-NEXT(p(s-RULED(ξ)))

Now, it is necessary to show that the reinterpretations of these functions still satisfy the <u>stated</u> properties (3-8 and 3-1 to 3-4).

(7-18) Lemma

ℓ: T-LIST -> N^0 see 7-9,3-8,2-10

N^n: T-LIST -> T see 7-10,3-8,2-11

Both results follow immediately from 7-1,7-8,2-10,2-11.

(7-19) Lemma

a)	s-RULES,s-NONT,s-RULED,s-STR,s-STRCHAR are finite	see 3-1,2-13
b)	ϵ: (RULED-PTR $\times$ N) $\times$ (RULES,...,STRCHAR) -> {T,F}	see 7-17,3-1,2-16
c)	ℓ: (RULED-PTR $\times$ N) $\times$ (RULES,...,STRCHAR) -> N	see 7-11,3-2,2-10
d)	N^n: (RULED-PTR $\times$ N) $\times$ (RULES,...,STRCHAR) ->	see 7-15,7-16,3-2,2-11
e)	$\sim$(is-nt(s-rhs(r)$_j$) $\wedge$ is-t(s-rhs(r)$_j$))	see 7-13,7-14,3-3,3-4

proofs

a follows from 7-1,7-5,7-6,7-7,7-8

b follows from 7-17,7-1,7-5,7-6,7-7,7-8

c follows from 7-11,7-1,7-5,7-6,7-7,7-8

d follows from 7-15,7-16,7-1,7-5,7-6,7-7,7-8

e follows from 7-13,7-14

However, it is not possible to simply change the problem definition (see 3-10 and 3-5,3-6) which is stated in terms of the original operations (e.g. r ϵ G). It was clear from the beginning that routines like INSTR and INGR would be required and it would have been possible to use different operations in the algorithm and note the equivalences which must hold between the original and algorithm notations. This was not done since it would have made the subsequent writing more tedious. In making the change to the re-interpreted operations we must now, carefully, check what properties are required and use them as a specification of the INSTR and INGR routines.

(7-20) Constraints on INSTR:

$\ell(X) = \ell(\text{s-INPUT}(\xi))$

$X_i = \text{s-INPUT}(\xi)_i$

(7-21) Constraints on INGR:

let (I(r,(p,i))

r ϵ G $\equiv$ is-RULED(p(s-RULED(ξ))) $\wedge$ i <s-NEXT(p(s-RULED(ξ)))

$\bar{r}$ = s-LENGT (s-RULE(p(s-RULED(ξ)))$_i$)

is-t(s-rhs(r)$_j$) $\equiv$ s-TYPE (I(s-rhs(r)$_j$)) = T

is-nt(s-rhs(r)$_j$) $\equiv$ s-TYPE (I(s-rhs(r)$_j$)) = N

for is-t(s-rhs(r)$_j$):

s-rhs(r)$_j$ $\equiv$ s-STRCHAR(ξ)$_{\text{s-N}\ (I(\text{s-rhs}(r)_j))}$

for is-nt(s-rhs(r)$_j$):

s-rhs(r)$_j$ = s-lhs(s) $\equiv$ s-NONT(ξ)$_{\text{s-N}\ (I(\text{s-rhs}(r)_j))}$ = q

<u>where</u> I(s,(q,k))

The storage structure given in 7-1 to 7-4 also gives us an interpretation of the collection of state-sets (see 4-2):

(7-22) len(S) becomes ℓ^o(s-S(ξ))

(7-23)	state-set(i,S)	becomes	$s\text{-}S(\xi)_i(s\text{-}STATE(\xi))$

for is-state(s)

(7-24)	s-rule(s)	becomes	s-RULE(s)
(7-25)	s-j(s)	becomes	s-RULPOS(s)
(7-26)	s-f(s)	becomes	s-STRPOS(s)

It is, of course, necessary to check that the required properties still hold.

(7-27) Lemma

ℓ^o : S -> N^n	see 4-2
state-set : $N^n \times$ S -> STATE-SET	see 4-2
s-RULE :	see 4-4
s-RULPOS :	see 4-4
s-STRPOS :	see 4-4

proofs all follow from 7-1,7-2,7-3,7-4,7-8

The resulting reinterpretations of is-stateset are:

(7-28)	ℓ: STATE-SET -> N	becomes	s-N(STATE)
(7-29)	N^n: STATE-SET -> STATE	becomes	$s\text{-}INFO(STATE)_i$

(7-30) Lemma

The obvious equivalences hold.

Follows from 7-1,7-4,7-8

Notice that $\underline{\epsilon}$ and $\underline{\cup}$ (see 6-2 and 6-4) are automatically correct.

It is now possible to rewrite the "algorithm" of Section 6 so that it uses the data structures of 7-1 to 7-7, that is, using the various reinterpretations set out above.

(7-31) rec(X,G) = rec´ (ξ^o)

<u>where</u> ξ^o created by INSTR(X)

INGR(G)

and $\ell^o(s\text{-}S(\xi^o)) = \ell(X)$

and $s\text{-}S(\xi^o)_o\ (s\text{-}STATE(\xi^o)) = \underline{\{}<rr,0,0>\underline{\}}$

and for $1 \le i \le \ell(X)$:

$s\text{-}S(\xi^o)_i(s\text{-}STATE(\xi^o)) = \underline{\{\ \}}$

```
(7-32)  $rec'(\xi) =$
            $<rr,\overline{rr},0> \in s\text{-}S(\xi^z)_{\ell(X)}(s\text{-}STATE(\xi^z)) \rightarrow \underline{YES}$
            $T \rightarrow \underline{NO}$
                $\underline{where}\ \xi^z$ such that $\underline{next}\ (z+1) = \underline{end}$

(7-33)  $next(1) = (0,1)$
      for $i>1$ :
        $next\ (i) = s\text{-}N(s\text{-}S(\xi^{i-1})_j(s\text{-}STATE(\xi^{i-1}))) = k \rightarrow$
                $j \neq \ell(s\text{-}INPUT(\xi^{i-1}))\ \&\ s\text{-}N(s\text{-}S(\xi^{i-1})_{j+1}(s\text{-}STATE(\xi^{i-1}))) \neq 0 \rightarrow (j+1,1)$
                $T \rightarrow \underline{end}$
            $T \rightarrow (j,k+1)$
                $\underline{where}\ (j,k) = next\ (i-1)$

(7-34)  for $k\geq 0$:
        $j \neq \bar{r}\ \wedge\ i < \ell(s\text{-}INPUT(\xi^k))\ \wedge\ s\text{-}TYPE(I(s\text{-}rhs(r)_{j+1})) = N \supset$
            $s\text{-}S(\xi^{k+1})_i\ (s\text{-}STATE(\xi^{k+1})) = s\text{-}S(\xi^k)_i(s\text{-}STATE(\xi^k))\ \underline{\cup}\ \{\ s|s \in predict(i,X,G,<r,j,f>)\}$
        $j \neq \bar{r}\ \wedge\ i < \ell(s\text{-}INPUT(\xi^k))\ \wedge\ s\text{-}TYPE(I(s\text{-}rhs(r)_{j+1})) = T\ \&\ s\text{-}STRCHAR(\xi)_{s\text{-}N(I(s\text{-}rhs(r)j+1))} =$
                $s\text{-}INPUT(\xi^k)_{i+1} \supset$
            $s\text{-}S(\xi^{k+1})_{i+1}(s\text{-}STATE(\xi^{k+1})) = s\text{-}S(\xi^k)_{i+1}(s\text{-}STATE(\xi^k))\ \underline{\cup}\ \{\ s|s \in scan(i+1,X,G,<r,j,f>)\}$
        $j = \bar{r} \supset$
            $s\text{-}S(\xi^{k+1})_i(s\text{-}STATE(\xi^{k+1})) = s\text{-}S(\xi^k)_i(s\text{-}STATE(\xi^k))\ \underline{\cup}\{s|s \in complete(\xi^k,i,X,G,<r,j,f>)\}$
        otherwise
            $s\text{-}S(\xi^{k+1})_p(s\text{-}STATE(\xi^{k+1})) = s\text{-}S(\xi^k)_p\ (s\text{-}STATE(\xi^k))$
                $\underline{where}\ \ <r,j,f> = s\text{-}INFO(s\text{-}S(\xi^k)_i(s\text{-}STATE(\xi^k)))_n$
                $\underline{and}\ \ \bar{r} = s\text{-}LENGT(s\text{-}RULE(p(s\text{-}RULED(\xi^k))_i))$
                $\underline{where}\ \ (p,i) = r$
                $\underline{where}\ \ (i,n) = next\ (k+1)$
```

Changes of specification and new ideas to, for example, improve efficiency are facts of life: if Formal Development is to be an accepted way of constructing programs its influence on subsequent changes must be understood. It is the opinion of the author that, if generality and abstractness have been the keynotes of the development, the difficulty of installing changes should be in realistic relation to how drastic that change is. Certainly more work will be required than with a "quick patch", but the same gains of certainty as with Formal Development should make the investment worthwhile.

The modification made in this section is an optimisation to reduce the number of irrelevant states created and scanned. It is, again, taken from a conventionally built version of the program. The concept is to append to rules an inevitable terminal character (that is, a character with which any derivation of the rule must start), if such exists. This is specified formally in 8-2. During prediction this look-ahead character, if it exists, is compared to the next character of the string: if they are unequal no state for this rule need be generated. Any such state would, after some number of further predictions, have been shown to be a "blind alley" by the scanner. This test will be performed by the predicate "cond" whose essential property is stated in 8-3: an obvious realisation is given in 9-13.

The effect of this change is now traced, beginning with the definition of is-rule(3-2) which is extended to include the pre-computed look-ahead symbols:

(8-1) $\quad$ is-rule = (<s-lhs : is-nt>,
<s-rhs : is-el-list>,
< s-lk : is-t>)

Further, the restriction on the look-ahead character is:

(8-2) $\quad$ for $r \in G$: $\quad$ $\text{s-lk}(r) \neq \Omega \supset (\text{s-rhs}(r) \overset{*}{=>} \alpha \wedge \text{is-t}(\alpha_1) \supset \alpha_1 = \text{s-lk}(r))$

Now a check of the development process is made to determine the extent of the changes.

Consider the problem definition (Section 3) - there is no change to be made since the whole point is to recognise the same strings, but to do it more efficiently.

Consider the outline of Earley's state-sets (Section 4) - there will be no change since the cut-down state-sets will still fall between those which satisfy equations 4-7 and 4-8.

Consider the creation of state-sets (Section 5) - A predicate "cond" is introduced which is assumed to satisfy the property:

(8-3) $\quad$ for $r \in G$: $\quad$ $\text{s-rhs}(r) \overset{*}{=>} \alpha \wedge \text{is-t}(\alpha_1) \supset \text{cond}(r,\alpha_1)$

$\text{cond}: \text{RULE} \times T \rightarrow \{T,F\}$

Then 5-8 is modified to employ cond:

(8-4) $\quad$ $\text{predict}(i,X,G,<r,j,f>) = \{<s,0,i> | j \neq \bar{r} \;\&\; \text{is-nt}(\text{s-rhs}(r)_{j+1}) \;\&\; s \in G \;\&$
$\text{s-lhs}(s) = \text{s-rhs}(r)_{j+1} \wedge \text{cond}(s,X_{i+1})\}$

The proof that the state-sets still satisfy 4-6 to 4-9 is modified as follows:

Theorem 5-11	unchanged	
Theorem 5-12	unaffected, because the set satisfying 8-4 is clearly a subset of that satisfying 5-8.	
Theorem 5-13	changed by replacing line q with:	
qa)	$\text{cond}(s,X_{i+1})$	o,p,8-3
qb)	$<s,0,i> \in \text{state-set}(i,\mathcal{S})$	qa,5-6,5-8,c,m,p,o

Notice no other steps of 5-13 refer to 5-8.

Theorem 5-14   add:

predict only used for $i < \ell(X)$

thus reference to X_{i+1} is in range

cond only used over its domain

Consider the mapping of state-sets to lists (Section 6) - no change to be made.

Consider the structuring for store (Section 7) - the reinterpretation of 8-1 (see 7-17) includes:

(8-5)   s-lk (r)   becomes $\text{s-STRCHAR}(\xi)_{\text{s-N}}\ (I(\text{s-rhs}(r)_o))$

and the additional constraint on INGR (see 7-21):

(8-6)   s-lk (r)   = $\text{s-STRCHAR}(\xi)_{\text{s-N}}\ (I(\text{s-rhs}(r)_o))$

Notice that this simple change to the input properties would require a fairly drastic change to INGR in that space must now be left for all the look ahead characters in s-RULES.

This section discusses the PL/I version (see Appendix I) of the developed algorithm (see Section 7, plus 5-8,5-9,5-10 and modifications of Section 8). The transition is not shown in full detail since this would require a significant portion of the theory of PL/I to be formalised. This is not only outside the scope of the current paper, it would also be inappropriate since it should be part of a larger, more general, endeavour.

First a number of constructs of PL/I are related to the properties that have been used above; then a number of specific results are established.

PL/I data properties:

(9-1) Objects (as in 2-6) can be replaced by PL/I structures and the named selectors can be mimicked by qualified naming. Notice that this property only holds with selectors which cannot yield the null object (see reference 2).

(9-2) Where the selectors are arbitrary (that is, their names are not explicitly given) the objects can be replaced by a collection of allocations of a based variable. In this case the selectors are replaced by pointers. (<u>Note</u>: In the representation of STATE-SETS given, the REFER option has also been used to obtain the possibility to dynamically vary the maximum size of a state-set).

(9-3) Lists (as in 2-10, 2-11) can be replaced by PL/I arrays; these present upper bound problems which make the algorithm non-total. Referencing is via subscripting and the current length is usually stored in another variable.

(9-4) As an alternative, lists of characters can be replaced by PL/I variable character strings. In this case selection is via the SUBSTR built-in function, and length can be found via the LENGTH built-in function.

(9-5) Integers are represented by FIXED BINARY numbers of length 15. This again introduces a restriction on the domain of the algorithm which is difficult to specify.

PL/I statement properties:

(9-6) Conditional expressions can be replaced by PL/I statements of the form:

```
IF p_1 THEN...
 :
ELSE IF p_n THEN...
```

(9-7) With objects which extend monotonically (for example, state-sets) assignment can be used with no loss of information.

(9-8) The use of <u>where</u> clauses can be replaced by computing the required value and assigning it to a variable that is referred to wherever the name is used (this can be thought of as "assignment as an abbreviation").

(9-9) PL/I DO loops are used to specify the sequencing of operations. In particular, nested DO loops iterating through the collection of state-sets and through the members of a state-set are used to mirror the function of "next" (see 7-33). Notice that the correctness of this relies on the fact that PL/I re-evaluates the "while" clause of a DO loop at each iteration. Thus a DO BY TO construct would fail!

(9-10) The conditional expression form of "and" (see 2-2) is translated so that the second operand is not evaluated if the first is false.

Special results:

(9-11) Lemma

$$ss' = ss \cup \{<s>\}$$

yields the same result as a procedure whose argument is a state and which changes the state-set by side effect:

```
DO cv = 1 BY 1 TO ℓ(ss) ;
    IF ss(cv) = <s> THEN RETURN ;
END ;
ss_{ℓ(ss)+1} = <s> ;
```

(9-12) Lemma The effect of adding a set defined implicitly can be obtained by an ordered search, calling the procedure of 9-11 each time a candidate for addition is located.

(9-13) Lemma

$$\text{cond } (r,t) \equiv \text{s-lk}(r) = ‡ \vee t = \text{s-lk}(r)$$

satisfies 8-3

proof

a)	$\text{s-rhs}(r) \overset{*}{\Rightarrow} \alpha$	assm
b)	$\text{is-t}(\alpha_1)$	assm
c)	case $\text{s-lk}(r) = ‡$:	
d)	$\quad$ cond(r,t)	9-12
e)	case $\text{s-lk}(r) \neq ‡$:	
f)	$\alpha_1 = \text{s-lk}(r)$	8-2,e,a,b
g)	$\quad$ cond(r,t)	9-12
		d,g

This completes the proof of 8-3.

The program invokes procedures called INGR and INSTR to read grammar and string into store (see Appendices II, III): these routines were in fact taken from the conventionally built program. A procedure IPTEST is then invoked (see Appendix IV) to check restrictions 4-1,6-9,7-20 and 7-21,8-2. Clearly the relation to the input grammar cannot be checked directly and the expedient of printing the grammar as it "must have looked" is adopted. Finally, the parsing proper is performed (a listing of a run is given in Appendix V).

10. Discussion

This section attempts to discuss some of the open questions.

Breaking the development down into steps has given a clearer picture of the algorithm and its correctness: is the work justified? Making a comparison with the more widely used "write then attempt proof" there are few portions of the above work which are wasted. (One example is the need to set up the same inductive form for proofs at two or more steps.) In view of the difficulty in composing a complete algorithm correctly from the beginning, this cost is not great. The proofs given here also tend to follow the intuitive program ideas more closely. Comparisons with methods not involving formal proofs are more difficult, but the opinion of the present author is that without the discipline of proof such methods will migrate poorly from their original environment.

The comment was made by an experienced programmer during a presentation of this work, that the much maligned tendency to start coding too early may be a symptom of wanting to write something formal but lacking any alternative language in which to state the partially thought-out ideas!

The development given here was aided by the existence and knowledge of a working version of the program. In an actual development much more backtracking would be required: it is certainly not claimed that the formal development approach would give rise to inspirations like Earley's state-sets. However, the formal approach would certainly offer a framework in which to search.

One of the principal problems is the level of formality required in the proofs. The developed algorithm was keypunched and run on an Algol grammar. Although no errors have been uncovered, this does not establish that the above level of proof would be formal enough to provide security against errors. There is certainly an important distinction to be made between assumptions on the data which are extremely dangerous (being the source of many conventional programming errors), and formality in the deductions which is not attempted. In particular, the extremely attractive notion of machine checkable proofs may not be practical because of the tedium of formalising the deductive steps. A specific problem of this distinction is the failure in the development to separate variables of the problem and language (for example, control variables) domains.

There are many general problems which become apparent in the above development: is there a clear set of ways of using storage/assignment (see 9-7,9-8)? Are there better ways of specifying order than the usual DO loops? Questions like this could provide valuable feedback to language design from a problem, rather than the usual machine, angle.

It is the hope that work on Formal Development will be applied to a variety of examples in an attempt to resolve these and other questions.

1. EARLEY, J, "An Efficient Context-Free Parsing Algorithm", COMM ACM, Vol 13, No. 2, February 1970.

2. LUCAS, P and WALK, K, "On the Formal Description of PL/I", ANNUAL REVIEW IN AUTOMATIC PROGRAMMING, Vol 6, Part 3, Pergamon Press, 1969.

3. PARK, D, "Fixpoint Induction and Proofs of Program Properties", MACHINE INTELLIGENCE, Vol 5, Edinburgh University Press, 1969.

4. MANNA, Z and McCARTHY, J, "Properties of Programs and Partial Function Logic", AI Memo 100 from Stanford University, 1970.

This appendix contains a PL/I program corresponding to the algorithm of Sections 7 and 8. The transliteration uses the ideas of Section 9.

```
STMT LEV NT

   1           EARLY:  PROC       OPTIONS(MAIN);

   2   1       DCL (LENGTH,SUBSTR) BUILTIN;

   3   1       DCL INPUT CHAR(5000) VARYING EXTERNAL;

   4   1       DCL 1 RULES(4000) EXTERNAL,
                     2 TYPE CHAR(1),
                     2 N FIXED BIN(15);

   5   1       DCL 1 NONT(500) EXTERNAL PTR;

   6   1       DCL 1 RULED BASED,
                     2 MAXNRULES FIXED BIN(15),
                     2 NEXT FIXED BIN(15),
                     2 RULE(NRULES REFER (MAXNRULES)),
                      3 START FIXED BIN(15),
                      3 LENGT FIXED BIN(15);

   7   1       DCL 1 STR(250) EXTERNAL,
                     2 START FIXED BIN(15);

   8   1       DCL STRCHAR CHAR(4000) VARYING EXTERNAL INIT('');

   9   1       DCL NRULES FIXED BIN(15) EXTERNAL;
  10   1       DCL IPGR    ENTRY EXTERNAL;
  11   1       DCL IPSTR   ENTRY EXTERNAL;
  12   1       DCL IPTEST  ENTRY EXTERNAL;

  13   1       DCL 1 STATE BASED (SI) ,
                       2 SIZE FIXED BIN(15),
                       2 N     FIXED BIN(15),
                       2 INFO (STATE_SIZE REFER (SIZE)),
                          3 RULE,
                             4 RPTR PTR,
                             4 SSC FIXED BIN(15),
                          3 RULPOS FIXED BIN(15),
                          3 STRPOS FIXED BIN(15);

  14   1       DCL STATE_SIZE FIXED BIN(15) INIT(70);


  15   1       DCL I FIXED BIN(15);        /* STATE SET INDEX ,MAJOR LOOP              */
  16   1       DCL N FIXED BIN(15);        /* STATE INDEX     ,2ND LOOP                */
  17   1       DCL P1 PTR;                 /* PTR OF S-RULE OF STATE(N,STATE-SET(I)) */
  18   1       DCL I1 FIXED BIN(15);       /* INT OF S-RULE OF STATE(N,STATE-SET(I)) */
  19   1       DCL J  FIXED BIN(15);       /* RULPOS        OF STATE(N,STATE-SET(I)) */
  20   1       DCL F  FIXED BIN(15);       /* STRPOS        OF STATE(N,STATE-SET(I)) */

  21   1       DCL R_ FIXED BIN(15);       /* LEN-RHS(P1,I1)                           */

  22   1       DCL 1 RHS_EL,               /* RHS_EL(J+1,(P1,I1))                      */
                       2 TYPE CHAR(1),
                       2 N FIXED BIN(15);

  23   1       DCL P2 PTR;                 /* PTR PT OF RULES S.T.
                                                  RHS_NT_EQ_LHS((P1,I1),J+1,(P2,?))  */
  24   1       DCL K FIXED BIN(15);        /* PREDICTOR'S RULE INDEX                   */

  25   1       DCL LK CHAR(1);             /* LK_EL(P2,K)                              */
```

```
STMT LEV NT


 26    1       DCL KS FIXED BIN(15);       /* STATE INDEX WITHIN S(F)                       */

 27    1       DCL P3 PTR;                 /* PTR OF S-RULE (STATE(KS,STATE-SET(F))   */
 28    1       DCL I2 FIXED BIN(15);       /* INT OF S-RULE (STATE(KS,STATE-SET(F))   */
 29    1       DCL L  FIXED BIN(15);       /* S-RULPOS       (STATE(KS,STATE-SET(F))  */
 30    1       DCL G  FIXED BIN(15);       /* S-STRPOS       (STATE(KS,STATE-SET(F))  */

 31    1       DCL 1 SF_RHS_EL,             /* RHS_EL(P3,I2)                            */
                     2 TYPE CHAR(1),
                     2 N FIXED BIN(15);

 32    1       DCL SI PTR;                 /* CONTAINS S(I) : F COMP RESTR             */
 33    1       DCL SF PTR;                 /* CONTAINS S(F) : F COMP RESTR             */
 34    1       DCL NONT1PTR PTR;           /* CONTAINS NONT(1).PTR : F COMP RESTR      */
 35    1       CALL IPGR    ;
 36    1       CALL IPTEST  ;
 37    1       CALL IPSTR   ;
 38    1       PUT PAGE;
 39    1       PUT EDIT((SUBSTR(INPUT||'          ',J,10) DO J=1 BY 10
                     TO LENGTH(INPUT)))      (10(A(10),X(1)),SKIP);

                /* N.B. IPSTR HAS SQUEZED ALL BLANKS TO AVOID REQUIREMENT TO CODE
                        SAME IN ALGOL GRAMMAR  */

 40    1       BEGIN;
 41    2       DCL S(0:LENGTH(INPUT)) PTR;

 42    2          DO I = 0B BY 1B WHILE(I <= LENGTH(INPUT));
 43    2  1          ALLOCATE STATE;            /* SETS  SI */
 44    2  1          S(I) = SI;
 45    2  1          SI -> STATE.N = 0B;
 46    2  1       END;

 47    2          SI = S(0B);
 48    2          SI -> STATE.N = 1B;
 49    2          SI -> STATE.INFO(1B).RULE.RPTR = NONT(1B);
 50    2          SI -> STATE.INFO(1B).RULE.SSC  = 1B;
 51    2          SI -> STATE.INFO(1B).RULPOS    = 0B;
 52    2          SI -> STATE.INFO(1B).STRPOS    = 0B;

                          /* INITIAL S S.T. : IS-S(S)
                                              L(STATE_SET(0))=1
                                              STATE(1,STATE_SET(0)) =
                                                        <(R)(LAS(R)=ROOT(S)),0,0>
                                              1 <= I <= L(INPUT) IMP
                                                           L(STATE_SET(I,S))= 0 */
```

```
STMT LEV NT


 53    2         DO I = 0B BY 1B WHILE(I<= LENGTH(INPUT));
 54    2   1        SI = S(I);
 55    2   1        DO N = 1B BY 1B WHILE(N<= SI -> STATE.N);

 56    2   2           P1 = SI -> STATE.INFO(N).RULE.RPTR;
 57    2   2           I1 = SI -> STATE.INFO(N).RULE.SSC;
 58    2   2           J  = SI -> STATE.INFO(N).RULPOS;
 59    2   2           F  = SI -> STATE.INFO(N).STRPOS;

 60    2   2           R_ = P1  -> RULED.RULE(I1).LENGT;

 61    2   2           IF(J ¬= R_) &(I ¬= LENGTH(INPUT)) THEN
                          DO;
 62    2   3                 RHS_EL = RULES(P1 -> RULED.RULE(I1).START + J ) ;
 63    2   3                 IF RHS_EL.TYPE = 'N' THEN
                                DO;                                /* PREDICTOR */
 64    2   4                       P2 = NONT(RHS_EL.N);
 65    2   4                       DO K = 1B BY 1B TO P2 -> RULED.NEXT -1B;
 66    2   5                          LK = SUBSTR(STRCHAR,
                                                STR(RULES(P2->RULED.RULE(K)
                                                          .START-1B).N).START,
                                                1);

 67    2   5                          IF(LK=' ')|(LK=SUBSTR(INPUT,I+1B,1))THEN
                                         CALL ADD_STATE(P2,K,0B,I,I);
 68    2   5                       END;
 69    2   4                    END;

 70    2   3                 ELSE IF(RHS_EL.TYPE = 'T') THEN
                                IF SUBSTR(STRCHAR,STR(RHS_EL.N).START,1) =
                                               SUBSTR(INPUT,I+1,1) THEN
                                   CALL ADD_STATE(P1,I1,J+1B,F,I+1B);   /* SCANNER */
 71    2   3              END;




 72    2   2           ELSE IF J = R_ THEN
                          DO;                                      /* COMPLETER */
 73    2   3                 SF = S(F);
 74    2   3                 DO KS = 1B BY 1B TO SF -> STATE.N;
 75    2   4                    P3 = SF -> STATE.INFO(KS).RULE.RPTR;
 76    2   4                    I2 = SF -> STATE.INFO(KS).RULE.SSC;
 77    2   4                    L  = SF -> STATE.INFO(KS).RULPOS;
 78    2   4                    G  = SF -> STATE.INFO(KS).STRPOS;

 79    2   4                    IF L ¬= P3->RULED.RULE(I2).LENGT THEN
                                   DO;
 80    2   5                          SF_RHS_EL = RULES(P3->RULED.RULE(I2).START+L);
 81    2   5                          IF SF_RHS_EL.TYPE = 'N' THEN
                                         IF NONT(SF_RHS_EL.N) = P1 THEN
                                            CALL ADD_STATE(P3,I2,L+1B,G,I);
 82    2   5                       END;
 83    2   4                 END;
 84    2   3              END;
 85    2   2        END;
 86    2   1     END;


                                        /* CHECK FOR END_STATE IN S(L(X))        */
```

```
STMT LEV NT

  87    2          SI = S(LENGTH(INPUT));
  88    2          NONT1PTR = NONT(1B);

                   DO N = 1B BY 1B TO SI -> STATE.N;
  89    2             IF (SI -> STATE.INFO(N).RULE.RPTR = NONT(1B))
  90    2   1           &(SI -> STATE.INFO(N).RULE.SSC  = 1B )
                        &(SI -> STATE.INFO(N).RULPOS    = NONT1PTR -> RULEC.RULE(1B).
                                                                          LENGT)

                        &(SI -> STATE.INFO(N).STRPOS    = 0B   ) THEN

                         DO;
  91    2   2            PUT EDIT (' YES' ) (SKIP ,A); RETURN;
  93    2   2            END;
  94    2   1      END;

  95    2          PUT EDIT (' NO')(SKIP,A);
  96    2          ADD_STATE:
                   PROC (PTR,INT,J,F,I);
                           /* ADDS ((PTR,INT),J,F) TO STATE-SET(I) ,UNLESS THERE */


  97    3          DCL PTR PTR;
  98    3          DCL INT FIXED BIN(15);
  99    3          DCL J   FIXED BIN(15);
 100    3          DCL F   FIXED BIN(15);
 101    3          DCL I   FIXED BIN(15);

 102    3          DCL K FIXED BIN(15);    /* STATE INDEX WITHIN STATE_SET(I)         */

 103    3          DCL SI PTR;             /* CONTAINS S(I)   : F COMP RESTR */

 104    3             SI = S(I);
 105    3             DO K = 1B BY 1B TO SI -> STATE.N ;
 106    3   1            IF (SI -> STATE.INFO(K).RULE.RPTR = PTR )
                           &(SI -> STATE.INFO(K).RULE.SSC  = INT )
                           &(SI -> STATE.INFO(K).RULPOS    = J )
                           &(SI -> STATE.INFO(K).STRPOS    = F ) THEN
                                                           /* STATE ALREADY THERE */
                            RETURN;
 107    3   1         END;

 108    3             IF SI -> STATE.N  = SI -> STATE.SIZE THEN
                                                           /* STATE SET FULL */
                         DO;
 109    3   1               PUT EDIT(' STATE SET OVERFLOW ')(SKIP,A);
 110    3   1               STOP;
 111    3   1            END;
 112    3                                                  ELSE
                                                           /* ADD */
                         DO;
 113    3   1            SI -> STATE.N = SI -> STATE.N + 1B;
 114    3   1            SI -> STATE.INFO(SI -> STATE.N).RULE.RPTR = PTR;
 115    3   1            SI -> STATE.INFO(SI -> STATE.N).RULE.SSC  = INT;
 116    3   1            SI -> STATE.INFO(SI -> STATE.N).RULPOS    = J;
 117    3   1            SI -> STATE.INFO(SI -> STATE.N).STRPOS    = F;
 118    3   1            END;
 119    3           END;
 120    2          END;

 121    1          END;
```

This appendix contains that part of the conventionally written version of the program which reads in and stores the grammar.

```
STMT LEV NT


   1          IPGR:PROC;
   2    1     DCL (LENGTH,SUBSTR) BUILTIN;
   3    1               DCL 1 NONS(500) EXTERNAL,
                                  2 START FIXED BIN(15),
                                  2 LENGT  FIXED BIN(15),
                                  2 PTR PTR;
   4    1               DCL NONTCHAR CHAR(3000) VARYING INIT('') EXTERNAL;
   5    1               DCL NONTNAME CHAR(1000) VARYING;
   6    1               DCL STRNAME CHAR(1000) VARYING;
   7    1               DCL 1 TEMP,
                                  2 TYPE CHAR(1),
                                  2 N FIXED BIN(15);
   8    1          DCL      (        I,J,K,L,M,                  NEXTRULES INIT(1),
                           NUSTRS INIT(0)) FIXED BIN(15);
   9    1          DCL (NRULES,NUNONTS INIT(0)) FIXED BIN(15) EXTERNAL;
  10    1               DCL CARD CHAR(80);
  11    1               DCL ENDFILE BIT(1) INIT('0'B);
  12    1               DCL (P,Q                          ) PTR ;
  13    1              DCL  1 NONT(500) EXTERNAL PTR;
  14    1               DCL 1 STR(250) EXTERNAL,
                                  2 START FIXED BIN(15);
  15    1               DCL STRCHAR CHAR(4000) VARYING INIT('') EXTERNAL;
  16    1               DCL 1 RULES(4000) EXTERNAL,
                                  2 TYPE CHAR(1),
                                  2 N FIXED BIN(15);
  17    1               DCL 1 RULED BASED(Q),
                                  2 MAXNRULES FIXED BIN(15),
                                  2 NEXT     FIXED BIN (15),
                                  2 RULE(NRULES REFER (MAXNRULES)),
                                     3 START FIXED BIN(15),
                                     3 LENGT  FIXED BIN(15);


  18    1              ON ENDFILE(SYSIN) BEGIN;
  19    2                    ENDFILE='1'B;
  20    2                    GOTO SETRULE;
  21    2                    END;
  22    1              GET EDIT(CARD) (A(80));
  23    1     L0:      NONTNAME = '';
  24    1              DO I=2 BY 1 WHILE(SUBSTR(CARD,I,1) ¬='>');
  25    1   1                NONTNAME = NONTNAME||SUBSTR(CARD,I,1);
  26    1   1          END;
              /* I IS POSITION ON CARD */
  27    1              DO J=1 TO NUNONTS;
  28    1   1                IF NONTNAME = SUBSTR(NONTCHAR,NONS(J).START,
                                                          NONS(J).LENGT)
                                  THEN GOTO L1;
  29    1   1          END;


              /* A NEW NONTERMINAL HAS TO BE ADDED */
  30    1              NUNONTS = NUNONTS + 1;
  31    1              NONS(NUNONTS).START = LENGTH(NONTCHAR) + 1;
  32    1              NONS(NUNONTS).LENGT = LENGTH(NONTNAME);
  33    1                    NRULES = 5 ;
  34    1                    ALLOCATE RULED;
  35    1                    NONT(NUNONTS)      = Q;
  36    1                    RULED.NEXT      = 1;
  37    1              NONTCHAR = NONTCHAR||NONTNAME;
  38    1              J = NUNONTS;
              /* J CONTAINS NONTERMINAL IDENTIFIER */
  39    1     L1:      L = 0;
```

```
STMT LEV NT


  40    1                STRNAME=SUBSTR(CARD,73,1);
  41    1                DO K=1 BY 1 TO NUSTRS;
  42    1   1                  IF STRNAME=SUBSTR(STRCHAR,STR(K).START,1                  )
                                 THEN GOTO LA;
  43    1   1            END;
  44    1                NUSTRS=NUSTRS+1;
  45    1                STR(NUSTRS).START=LENGTH(STRCHAR)+1;
  46    1                STRCHAR=STRCHAR||STRNAME;
  47    1        LA:     RULES(NEXTRULES).N=K;
  48    1                RULES(NEXTRULES).TYPE='T';
  49    1                NEXTRULES=NEXTRULES+1;
         /* NOW PROCESS THE RHS OF THE CARD */
  50    1        L2:     L = L+1;
  51    1        L3:     IF I >= 71 THEN DO;
  52    1   1                  GET COPY EDIT(CARD) (A(80));
  53    1   1                  IF SUBSTR(CARD,1,1) = '<' THEN GOTO SETRULE;
  54    1   1                                          ELSE I = 1;
  55    1   1                            END;
  56    1                          ELSE I = I+1;
  57    1                IF SUBSTR(CARD,I,1) = ' ' THEN GOTO L3;
  58    1                IF SUBSTR(CARD,I,1) = '<' THEN GOTO NONTERM;
  59    1                 GOTO CHAR;
  60    1        NONTERM:NONTNAME = '';
  61    1                DO I = I+1 BY 1 WHILE (SUBSTR(CARD,I,1) ¬= '>');
  62    1   1                  NONTNAME = NONTNAME||SUBSTR(CARD,I,1);
  63    1   1            END;
  64    1                DO K=1 TO NUNONTS;
  65    1   1                  IF NONTNAME = SUBSTR(NONTCHAR,NONS(K).START,
                                                           NONS(K).LENGT)
                                     THEN GOTO L4;
  66    1   1            END;
         /* A NEW NONTERMINAL HAS TO BE ADDED */
  67    1                NUNONTS = NUNONTS + 1;
  68    1                NONS(NUNONTS).START = LENGTH(NONTCHAR) + 1;
  69    1                NONS(NUNONTS).LENGT = LENGTH(NONTNAME);
  70    1                      NRULES = 5 ;
  71    1                      ALLOCATE RULED;
  72    1                      NONT(NUNONTS)      = Q;
  73    1                      RULED.NEXT      = 1;
  74    1                NONTCHAR = NONTCHAR||NONTNAME;
  75    1                K=NUNONTS;
         /* INSERT VALUE IN RULE DICTIONARY */
  76    1        L4:     RULES(NEXTRULES).TYPE ='N';
  77    1                RULES(NEXTRULES).N =K;
  78    1                NEXTRULES = NEXTRULES + 1;
         /* RETURN TO PROCESS NEXT ELEMENT ON RHS OF RULE */
  79    1                GOTO L2;
  80    1        CHAR:   STRNAME =  SUBSTR(CARD,I,1);
  81    1                DO K=1 BY 1 TO NUSTRS;
  82    1   1                  IF STRNAME = SUBSTR(STRCHAR,STR(K).START,1)
                                 THEN GOTO L5;
  83    1   1            END;
         /* A NEW CHARACTER STRING HAS TO BE ADDED */
  84    1                NUSTRS = NUSTRS + 1;
  85    1                STR(NUSTRS).START = LENGTH(STRCHAR) + 1;
  86    1                STRCHAR = STRCHAR||STRNAME;
  87    1                K = NUSTRS;
         /* INSERT VALUE IN RULE DICTIONARY */
  88    1        L5:     RULES(NEXTRULES).TYPE = 'T';
  89    1                RULES(NEXTRULES).N = K;
  90    1                NEXTRULES = NEXTRULES + 1;
         /* RETURN TO PROCESS NEXT ELEMENT ON RHS OF RULE */
  91    1                GOTO L2;
         /* ADD RULE TO RULE DICTIONARY - ALLOCATING RULED AS NECESSARY */
  92    1        SETRULE:P = NONT(J)      ;
  93    1                IF P->RULED.MAXNRULES = P->RULED.NEXT      - 1
                               THEN DO;
  94    1   1                  NRULES = P-> RULED.MAXNRULES + 5 ;
  95    1   1                  ALLOCATE RULED;
  96    1   1                  RULED.NEXT      = P->RULED.NEXT     ;
```

```
STMT LEV NT


  97    1   1                   DO M=1 BY 1 TO P-> RULED.NEXT      - 1;
  98    1   2                         RULED.RULE(M) = P->RULED.RULE(M);
  99    1   2                   END;
 100    1   1                   NRULES=P->RULED.MAXNRULES;
 101    1   1                   FREE P->RULED;
 102    1   1                   P,NONT(J)      = Q;
 103    1   1                   END;
 104    1               M = P-> RULED.NEXT     ;
 105    1               P->RULED.NEXT     =P->RULED.NEXT     +1;
 106    1               P->RULED.RULE(M).START = NEXTRULES - L + 1 ;
 107    1               P->RULED.RULE(M).LENGT = L - 1 ;
                /* RETURN FOR NEXT CARD IF THERE ARE ANY */
 108    1               IF ENDFILE = '0'B THEN GOTO L0;
 109    1       END;
```

This appendix contains that part of the conventionally written version of the program which reads in and stores the string to be recognised.

```
STMT LEV NT

   1              IPSTR:PROC;
   2    1         DCL (LENGTH,SUBSTR) BUILTIN;
   3    1                   DCL J FIXED BIN(15);
   4    1                   DCL CARD CHAR(80);
   5    1                   DCL  BFLAG BIT(1) INIT('1'B);
   6    1                 DCL   INPUT CHAR(5000) VARYING EXTERNAL;
   7    1                 INPUT = '';
   8    1         SET1:   GET       FILE(IN) EDIT(CARD) (A(80));
   9    1                 IF SUBSTR(CARD,1,1) = '"' THEN   /* 7,8 AS END I/P FILE */
                             RETURN;
  10    1                 DO J=1 BY 1 TO 72;
  11    1   1                    IF SUBSTR(CARD,J,1) ¬= ' ' THEN DO;
  12    1   2                          INPUT=INPUT||SUBSTR(CARD,J,1);
  13    1   2                          BFLAG='0'B;
  14    1   2                                                         END;
  15    1   1                                                 ELSE DO;
  16    1   2                          IF BFLAG='0'B THEN DO;
  17    1   3                                BFLAG='1'B;
  18    1   3                                               END;
  19    1   2                                                         END;
  20    1   1                 END;
  21    1                     GOTO SET1;
  22    1          END;
```

This appendix contains a PL/I program to check that the various restrictions, which were relied on during the development, have been met. As mentioned, the relation of the stored to the given grammar is checked by printing out the former. The look-ahead character, if present, is printed between asterisks after the production arrow.

```
STMT LEV NT


   1          IPTEST:
              PROC;
   2     1    DCL (LENGTH,SUBSTR) BUILTIN;
   3     1              DCL  1 NONT(500) EXTERNAL PTR;
   4     1                DCL 1 STR(250) EXTERNAL,
                                    2 START FIXED BIN(15);
   5     1                DCL STRCHAR CHAR(4000) VARYING INIT('') EXTERNAL;
   6     1                DCL 1 RULES(4000) EXTERNAL,
                                    2 TYPE CHAR(1),
                                    2 N FIXED BIN(15);
   7     1                DCL 1 RULED BASED    ,
                                    2 MAXNRULES FIXED BIN(15),
                                    2 NEXT      FIXED BIN (15),
                                    2 RULE(NRULES REFER (MAXNRULES)),
                                        3 START FIXED BIN(15),
                                        3 LENGT  FIXED BIN(15);
   8     1                DCL NRULES FIXED BIN(15) EXTERNAL;
   9     1                DCL NUNONTS FIXED BIN(15) EXTERNAL;
  10     1                DCL 1 NONS(500) EXTERNAL,
                                    2 START FIXED BIN(15),
                                    2 LENGT FIXED BIN(15),
                                    2 PTR PTR;

  11     1                DCL NONTCHAR CHAR(3000) VARYING INIT('') EXTERNAL;

  12     1                DCL I         FIXED BIN(15) ;
  13     1                DCL J         FIXED BIN(15) ;
  14     1                DCL K         FIXED BIN(15) ;
  15     1                DCL P PTR;
  16     1    DCL N FIXED BIN(15);
  17     1    DCL R FIXED BIN(15);
  18     1    DCL NP PTR;
  19     1    DCL LK CHAR(1);
  20     1    DCL T BIT(2);
  21     1    DCL 1 TEMP , 2 TYPE CHAR(1),
                           2 N FIXED BIN(15);
  22     1       CALL PRINTDIC;
  23     1       PUT EDIT ('INPUT TEST') (PAGE,A);
  24     1       NP =NONT(1B)     ;
  25     1       PUT EDIT(' R1 - TEST NO OF ROOT(G) RULES: ')(SKIP,A);
  26     1       IF NP -> RULED.NEXT = 2 THEN
                    PUT EDIT ('R1 OK') (      A);
  27     1                              ELSE
                    PUT EDIT ('R1 FAILED') (      A);
  28     1       PUT EDIT(' R0- LENGTH OF RHS',' R3 - S-LK IS OK')(SKIP,A,SKIP,A);
  29     1       DO N = 1B BY 1B TO NUNONTS;
  30     1   1      NP = NONT(N)     ;
  31     1   1      DO R = 1B BY 1B TO NP->RULED.NEXT - 1B;
  32     1   2         IF NP->RULED.RULE(R).LENGT = 0 THEN
                          PUT EDIT ('R0 FAILED FOR ',N,R)(SKIP,A,F(5),F(5));
  33     1   2         LK = SUBSTR(STRCHAR,STR(RULES(NP->RULED.RULE(R).START -1B).N)
                                   .START,1);
  34     1   2         IF LK¬= ' ' THEN
                          DO;
  35     1   3            PUT EDIT (N,R)              (SKIP,F(5),F(5));
  36     1   3            TEMP = RULES(NP->RULED.RULE(R).START      );
```

```
STMT LEV NT

  37   1  3             IF TEMP.TYPE = 'T' THEN
                           IF SUBSTR(STRCHAR,STR(TEMP.N).START,1) = LK THEN
                              PUT EDIT (' OK')(A);
                                                                   ELSE
  38   1  3
                              PUT EDIT (' FAILED')(A);
  39   1  3                            ELSE
                              DO;
  40   1  4                      T = TEST_LK(0B,LK,NONT(TEMP.N)     );
  41   1  4                      IF SUBSTR (T,2,1) = '0'B THEN
                                    PUT EDIT (' FAILED    ')(A);
  42   1  4                                               ELSE
                                    IF SUBSTR(T,1,1) = '0'B THEN
                                       PUT EDIT (' UNDECIDED')(A);
  43   1  4                                                 ELSE
                                       PUT EDIT (' OK      ')(A);
  44   1  4                   END;
  45   1  3             END;
  46   1  2         END;
  47   1  1      END;


  48   1        PRINTDIC:
                PROC;
  49   2        DCL XREF CHAR(120) VARYING;
  50   2        DCL BL CHAR(120) INIT (' ');
  51   2        DCL CHAR BUILTIN;
  52   2           PUT PAGE;
  53   2           DO J= 1B TO NUNONTS;
  54   2  1           P = NONT(J)     ;
  55   2  1           IF P-> RULED.NEXT = 1B THEN
                         PUT    EDIT ('** NO RULES FOR ** ','<',
                                    SUBSTR(NONTCHAR,NONS(J).START,NONS(J).LENGT),
                                    '>')
                                   (SKIP,A,A,A(NONS(J).LENGT),A);
  56   2  1                                  ELSE
                         DO I = 1B TO P->RULED.NEXT-1B;
  57   2  2                 XREF = '' ;
  58   2  2                 PUT EDIT('<',SUBSTR(NONTCHAR,NONS(J).START,NONS(J).LENGT),
                                   '>','->')
                                  (SKIP,A,A(NONS(J).LENGT),A,X(2),A(2));
  59   2  2                 XREF = XREF || SUBSTR(BL,1,NONS(J).LENGT - 1B ) ||
                                          SUBSTR(CHAR(J,9),7,3)  || '   ';
  60   2  2                 IF SUBSTR(STRCHAR,STR(RULES(P->RULED.RULE(I).START -1B).N)
                                    .START,1) ¬= ' ' THEN
                               DO;
  61   2  3                       PUT EDIT('*',SUBSTR(STRCHAR,STR(RULES(P->RULED.
                                          RULE(I).START-1B).N).START,1),'*')
                                        (A,A(1),A);
  62   2  3                       XREF = XREF || '   ';
  63   2  3                    END;
  64   2  2                 DO K = 1B TO P-> RULED .RULE(I).LENGT;
  65   2  3                    TEMP = RULES(P->RULED.RULE(I).START + K -1B);
  66   2  3                    IF TEMP.TYPE ='N' THEN
                                  DO;
  67   2  4                          PUT EDIT('<',SUBSTR(NONTCHAR,NONS(TEMP.N).START,
                                                  NONS(TEMP.N).LENGT),'>')
                                            (X(1),A,A(NONS(TEMP.N).LENGT),A);
  68   2  4                          XREF = XREF || SUBSTR(BL,1,NONS(TEMP.N).LENGT) ||
                                                     SUBSTR(CHAR(TEMP.N,9),7,3);
  69   2  4                       END;
  70   2  3                                   ELSE
                                  DO;
  71   2  4                          PUT EDIT(SUBSTR(STRCHAR,STR(TEMP.N).START,1))
                                            (X(1),A(1));
  72   2  4                          XREF = XREF || '  ';
  73   2  4                       END;
  74   2  3                 END;
  75   2  2                 PUT EDIT(XREF) (SKIP,A(120));
  76   2  2              END;
  77   2  1      END;
```

```
STMT LEV NT


  78   2     END;
  79   1        TEST_LK:
                PROC (DEPTH,CHAR,NT) RECURSIVE RETURNS(BIT(2));
  80   2        DCL DEPTH FIXED BIN(15);
  81   2        DCL CHAR CHAR(1);
  82   2        DCL NT PTR;
  83   2        DCL RESULT BIT(2);
  84   2        DCL R FIXED BIN(15);
  85   2        DCL 1 TEMP , 2 TYPE CHAR(1),
                             2 N FIXED BIN(15);
  86   2           IF DEPTH = 101B THEN
                      RETURN ('01'B);
  87   2                           ELSE
                      DO;
  88   2  1              RESULT ='11'B ;
  89   2  1              DO R = 1B BY 1B TO NT->RULED.NEXT - 1B;
  90   2  2                 TEMP = RULES(NT->RULED.RULE(R).START)    :
  91   2  2                 IF TEMP.TYPE = 'T' THEN
                               IF SUBSTR(STRCHAR,STR(TEMP.N).START,1) ¬= CHAR THEN
                                  RESULT = RESULT & '10'B;
  92   2  2                                                                   ELSE
                                  RESULT = RESULT & '11'B;
  93   2  2                                    ELSE
                               RESULT = RESULT & TEST_LK(DEPTH+1B,CHAR,
                                                           NONT(TEMP.N)    );
  94   2  2              END;
  95   2  1              RETURN(RESULT);
  96   2  1           END;
  97   2        END;
  98   1     END;
```

This appendix contains a listing of a run of the recogniser.

```
<ROOT>  -> <NUMBER>
     1            2
<NUMBER>  -> <UNSIGNED_NUMBER>
       2                     9
<NUMBER>  ->*+* + <UNSIGNED_NUMBER>
       2                          9
<NUMBER>  ->*-* - <UNSIGNED_NUMBER>
       2                          9
<DIGIT>  ->*0* 0
      3
<DIGIT>  ->*1* 1
      3
<DIGIT>  ->*2* 2
      3
<DIGIT>  ->*3* 3
      3
<DIGIT>  ->*4* 4
      3
<DIGIT>  ->*5* 5
      3
<DIGIT>  ->*6* 6
      3
<DIGIT>  ->*7* 7
      3
<DIGIT>  ->*8* 8
      3
<DIGIT>  ->*9* 9
      3
<UNSIGNED_INTEGER>  -> <DIGIT>
                 4            3
<UNSIGNED_INTEGER>  -> <UNSIGNED_INTEGER> <DIGIT>
                 4                      4       3
<INTEGER>  -> <UNSIGNED_INTEGER>
        5                      4
<INTEGER>  ->*+* + <UNSIGNED_INTEGER>
        5                           4
<INTEGER>  ->*-* - <UNSIGNED_INTEGER>
        5                           4
<DECIMAL_FRACTION>  ->*.* . <UNSIGNED_INTEGER>
                 6                           4
<EXPONENT_PART>  ->*'* ' <INTEGER>
              7                  5
<DECIMAL_NUMBER>  -> <UNSIGNED_INTEGER>
               8                      4
<DECIMAL_NUMBER>  ->*.* <DECIMAL_FRACTION>
               8                         6
<DECIMAL_NUMBER>  -> <UNSIGNED_INTEGER> <DECIMAL_FRACTION>
               8                      4                  6
<UNSIGNED_NUMBER>  -> <DECIMAL_NUMBER>
                9                    8
<UNSIGNED_NUMBER>  ->*'* <EXPONENT_PART>
                9                      7
<UNSIGNED_NUMBER>  -> <DECIMAL_NUMBER> <EXPONENT_PART>
                9                    8               7
```

```
INPUT TEST
 R1 - TEST NO OF ROOT(S) RULES: R1 OK
 RO- LENGTH OF RHS
 R3 - S-LK IS OK
    2     2 OK
    2     3 OK
    3     1 OK
    3     2 OK
    3     3 OK
    3     4 OK
    3     5 OK
    3     6 OK
    3     7 OK
    3     8 OK
    3     9 OK
    3    10 OK
    5     2 OK
    5     3 OK
    6     1 OK
    7     1 OK
    8     2 OK
    9     2 OK
 -12.3'-4
  YES
```

This appendix contains a PL/I program corresponding to the algorithm of Sections 7 and 8. The transliteration uses the ideas of Section 9.

```
STMT LEV NT

  1          EARLY:  PROC       OPTIONS(MAIN);

  2    1     DCL (LENGTH,SUBSTR) BUILTIN;

  3    1     DCL INPUT CHAR(5000) VARYING EXTERNAL;

  4    1     DCL 1 RULES(4000) EXTERNAL,
                   2 TYPE CHAR(1),
                   2 N FIXED BIN(15);

  5    1     DCL 1 NONT(500) EXTERNAL PTR;

  6    1     DCL 1 RULED BASED,
                   2 MAXNRULES FIXED BIN(15),
                   2 NEXT FIXED BIN(15),
                   2 RULE(NRULES REFER (MAXNRULES)),
                    3 START FIXED BIN(15),
                    3 LENGT FIXED BIN(15);

  7    1     DCL 1 STR(250) EXTERNAL,
                   2 START FIXED BIN(15);

  8    1     DCL STRCHAR CHAR(4000) VARYING EXTERNAL INIT('');

  9    1     DCL NRULES FIXED BIN(15) EXTERNAL;
 10    1     DCL IPGR    ENTRY EXTERNAL;
 11    1     DCL IPSTR   ENTRY EXTERNAL;
 12    1     DCL IPTEST  ENTRY EXTERNAL;

 13    1     DCL 1 STATE BASED (SI) ,
                      2 SIZE FIXED BIN(15),
                      2 N    FIXED BIN(15),
                      2 INFO (STATE_SIZE REFER (SIZE)),
                         3 RULE,
                            4 RPTR PTR,
                            4 SSC FIXED BIN(15),
                         3 RULPOS FIXED BIN(15),
                         3 STRPOS FIXED BIN(15);

 14    1     DCL STATE_SIZE FIXED BIN(15) INIT(70);


 15    1     DCL I FIXED BIN(15);       /* STATE SET INDEX ,MAJOR LOOP              */
 16    1     DCL N FIXED BIN(15);       /* STATE INDEX     ,2ND LOOP                */
 17    1     DCL P1 PTR;                /* PTR OF S-RULE OF STATE(N,STATE-SET(I)) */
 18    1     DCL I1 FIXED BIN(15);      /* INT OF S-RULE OF STATE(N,STATE-SET(I)) */
 19    1     DCL J  FIXED BIN(15);      /* RULPOS        OF STATE(N,STATE-SET(I)) */
 20    1     DCL F  FIXED BIN(15);      /* STRPOS        OF STATE(N,STATE-SET(I)) */

 21    1     DCL R_ FIXED BIN(15);      /* LEN-RHS(P1,I1)                          */

 22    1     DCL 1 RHS_EL,              /* RHS_EL(J+1,(P1,I1))                     */
                    2 TYPE CHAR(1),
                    2 N FIXED BIN(15);

 23    1     DCL P2 PTR;                /* PTR PT OF RULES S.T.
                                               RHS_NT_EQ_LHS((P1,I1),J+1,(P2,?))  */
 24    1     DCL K FIXED BIN(15);       /* PREDICTOR'S RULE INDEX                  */

 25    1     DCL LK CHAR(1);            /* LK_EL(P2,K)                             */
```

```
STMT LEV NT


  26    1       DCL KS FIXED BIN(15);      /* STATE INDEX WITHIN S(F)                           */

  27    1       DCL P3 PTR;                /* PTR OF S-RULE (STATE(KS,STATE-SET(F))  */
  28    1       DCL I2 FIXED BIN(15);      /* INT OF S-RULE (STATE(KS,STATE-SET(F))  */
  29    1       DCL L  FIXED BIN(15);      /* S-RULPOS      (STATE(KS,STATE-SET(F))  */
  30    1       DCL G  FIXED BIN(15);      /* S-STRPOS      (STATE(KS,STATE-SET(F))  */

  31    1       DCL 1 SF_RHS_EL,            /* RHS_EL(P3,I2)                                   */
                      2 TYPE CHAR(1),
                      2 N FIXED BIN(15);

  32    1       DCL SI PTR;                /* CONTAINS S(I) : F COMP RESTR                     */
  33    1       DCL SF PTR;                /* CONTAINS S(F) : F COMP RESTR                     */
  34    1       DCL NONT1PTR PTR;          /* CONTAINS NONT(1).PTR : F COMP RESTR              */
  35    1       CALL IPGR   ;
  36    1       CALL IPTEST ;
  37    1       CALL IPSTR  ;
  38    1       PUT PAGE;
  39    1       PUT EDIT((SUBSTR(INPUT||'          ',J,10) DO J=1 BY 10
                      TO LENGTH(INPUT)))      (10(A(10),X(1)),SKIP);

                /* N.B. IPSTR HAS SQUEZED ALL BLANKS TO AVOID REQUIREMENT TO CODE
                        SAME IN ALGOL GRAMMAR  */

  40    1       BEGIN;
  41    2       DCL S(0:LENGTH(INPUT)) PTR;

  42    2          DO I = 0B BY 1B WHILE(I <= LENGTH(INPUT));
  43    2  1          ALLOCATE STATE;             /* SETS  SI */
  44    2  1          S(I) = SI;
  45    2  1          SI -> STATE.N = 0B;
  46    2  1       END;

  47    2          SI = S(0B);
  48    2          SI -> STATE.N = 1B;
  49    2          SI -> STATE.INFO(1B).RULE.RPTR = NONT(1B);
  50    2          SI -> STATE.INFO(1B).RULE.SSC  = 1B;
  51    2          SI -> STATE.INFO(1B).RULPOS    = 0B;
  52    2          SI -> STATE.INFO(1B).STRPOS    = 0B;

                           /* INITIAL S S.T. : IS-S(S)
                                               L(STATE_SET(0))=1
                                               STATE(1,STATE_SET(0)) =
                                                        <(R)(LAS(R)=ROOT(G)),G,0>
                                               1 <= I <= L(INPUT) IMP
                                                            L(STATE_SET(I,S))= 0 */
```

```
STMT LEV NT

 53   2          DO I = 0B BY 1B WHILE(I<= LENGTH(INPUT));
 54   2   1         SI = S(I);
 55   2   1         DO N = 1B BY 1B WHILE(N<= SI -> STATE.N);

 56   2   2            P1 = SI -> STATE.INFO(N).RULE.RPTR;
 57   2   2            I1 = SI -> STATE.INFO(N).RULE.SSC;
 58   2   2            J  = SI -> STATE.INFO(N).RULPOS;
 59   2   2            F  = SI -> STATE.INFO(N).STRPOS;

 60   2   2            R_ = P1  -> RULED.RULE(I1).LENGT;

 61   2   2            IF(J ¬= R_) &(I ¬= LENGTH(INPUT)) THEN
                          DO;
 62   2   3                 RHS_EL = RULES(P1 -> RULED.RULE(I1).START + J ) ;
 63   2   3                 IF RHS_EL.TYPE = 'N' THEN
                               DO;                                  /* PREDICTOR */
 64   2   4                       P2 = NONT(RHS_EL.N);
 65   2   4                       DO K = 1B BY 1B TO P2 -> RULED.NEXT -1B;
 66   2   5                          LK = SUBSTR(STRCHAR,
                                                 STR(RULES(P2->RULED.RULE(K)
                                                        .START-1B).N).START,
                                                 1);

 67   2   5                          IF(LK=' ')|(LK=SUBSTR(INPUT,I+1B,1))THEN
                                        CALL ADD_STATE(P2,K,0B,I,I);
 68   2   5                       END;
 69   2   4                    END;

 70   2   3                 ELSE IF(RHS_EL.TYPE = 'T') THEN
                                  IF SUBSTR(STRCHAR,STR(RHS_EL.N).START,1) =
                                                    SUBSTR(INPUT,I+1,1) THEN
                                     CALL ADD_STATE(P1,I1,J+1B,F,I+1B);  /* SCANNER */
 71   2   3              END;




 72   2   2            ELSE IF J = R_ THEN
                          DO;                                       /* COMPLETER */
 73   2   3                 SF = S(F);
 74   2   3                 DO KS = 1B BY 1B TO SF -> STATE.N;
 75   2   4                    P3 = SF -> STATE.INFO(KS).RULE.RPTR;
 76   2   4                    I2 = SF -> STATE.INFO(KS).RULE.SSC;
 77   2   4                    L  = SF -> STATE.INFO(KS).RULPOS;
 78   2   4                    G  = SF -> STATE.INFO(KS).STRPOS;

 79   2   4                    IF L ¬= P3->RULED.RULE(I2).LENGT THEN
                                  DO;
 80   2   5                          SF_RHS_EL = RULES(P3->RULED.RULE(I2).START+L);
 81   2   5                          IF SF_RHS_EL.TYPE = 'N' THEN
                                        IF NONT(SF_RHS_EL.N) = P1 THEN
                                           CALL ADD_STATE(P3,I2,L+1B,G,I);
 82   2   5                       END;
 83   2   4                 END;
 84   2   3              END;
 85   2   2          END;
 86   2   1       END;


                                          /* CHECK FOR END_STATE IN S(L(X))        */
```

```
STMT LEV NT

  87   2          SI = S(LENGTH(INPUT));
  88   2          NONT1PTR = NONT(1B);

  89   2          DO N = 1B BY 1B TO SI -> STATE.N;
  90   2   1         IF (SI -> STATE.INFO(N).RULE.RPTR  = NONT(1B))
                       &(SI -> STATE.INFO(N).RULE.SSC   = 1B )
                       &(SI -> STATE.INFO(N).RULPOS     = NONT1PTR -> RULEC.RULE(1B).
                                                                         LENGT)

                       &(SI -> STATE.INFO(N).STRPOS     = 0B   ) THEN

                        DO;
  91   2   2            PUT EDIT (' YES' ) (SKIP ,A); RETURN;
  93   2   2            END;
  94   2   1       END;

  95   2          PUT EDIT (' NO')(SKIP,A);
  96   2          ADD_STATE:
                  PROC (PTR,INT,J,F,I);
                         /* ADDS ((PTR,INT),J,F) TO STATE-SET(I) ,UNLESS THERE */


  97   3          DCL PTR PTR;
  98   3          DCL INT FIXED BIN(15);
  99   3          DCL J   FIXED BIN(15);
 100   3          DCL F   FIXED BIN(15);
 101   3          DCL I   FIXED BIN(15);

 102   3          DCL K FIXED BIN(15);    /* STATE INDEX WITHIN STATE_SET(I)        */

 103   3          DCL SI PTR;             /* CONTAINS S(I)   : F COMP RESTR */

 104   3             SI = S(I);
 105   3             DO K = 1B BY 1B TO SI -> STATE.N ;
 106   3   1            IF (SI -> STATE.INFO(K).RULE.RPTR = PTR )
                          &(SI -> STATE.INFO(K).RULE.SSC  = INT )
                          &(SI -> STATE.INFO(K).RULPOS    = J )
                          &(SI -> STATE.INFO(K).STRPOS    = F ) THEN
                           RETURN;                        /* STATE ALREADY THERE */
 107   3   1         END;

 108   3             IF SI -> STATE.N  = SI -> STATE.SIZE THEN
                        DO;                               /* STATE SET FULL */
 109   3   1               PUT EDIT(' STATE SET OVERFLOW ')(SKIP,A);
 110   3   1               STOP;
 111   3   1            END;
 112   3                                                  ELSE
                        DO;                               /* ADD */
 113   3   1            SI -> STATE.N = SI -> STATE.N + 1B;
 114   3   1            SI -> STATE.INFO(SI -> STATE.N).RULE.RPTR  = PTR;
 115   3   1            SI -> STATE.INFO(SI -> STATE.N).RULE.SSC   = INT;
 116   3   1            SI -> STATE.INFO(SI -> STATE.N).RULPOS     = J;
 117   3   1            SI -> STATE.INFO(SI -> STATE.N).STRPOS     = F;
 118   3   1            END;
 119   3          END;
 120   2        END;

 121   1        END;
```

This appendix contains that part of the conventionally written version of the program which reads in and stores the grammar.

```
STMT LEV NT

   1          IPGR:PROC;
   2    1     DCL (LENGTH,SUBSTR) BUILTIN;
   3    1              DCL 1 NONS(500) EXTERNAL,
                                 2 START FIXED BIN(15),
                                 2 LENGT  FIXED BIN(15),
                                 2 PTR PTR;
   4    1              DCL NONTCHAR CHAR(3000) VARYING INIT('') EXTERNAL;
   5    1              DCL NONTNAME CHAR(1000) VARYING;
   6    1              DCL STRNAME CHAR(1000) VARYING;
   7    1              DCL 1 TEMP,
                                 2 TYPE CHAR(1),
                                 2 N FIXED BIN(15);
   8    1          DCL      (       I,J,K,L,M,                NEXTRULES INIT(1),
                          NUSTRS INIT(0)) FIXED BIN(15);
   9    1          DCL (NRULES,NUNONTS INIT(0)) FIXED BIN(15) EXTERNAL;
  10    1              DCL CARD CHAR(80);
  11    1              DCL ENDFILE BIT(1) INIT('0'B);
  12    1              DCL (P,Q                          ) PTR ;
  13    1             DCL  1 NONT(500) EXTERNAL PTR;
  14    1              DCL 1 STR(250) EXTERNAL,
                                 2 START FIXED BIN(15);
  15    1              DCL STRCHAR CHAR(4000) VARYING INIT('') EXTERNAL;
  16    1              DCL 1 RULES(4000) EXTERNAL,
                                 2 TYPE CHAR(1),
                                 2 N FIXED BIN(15);
  17    1              DCL 1 RULED BASED(Q),
                                 2 MAXNRULES FIXED BIN(15),
                                 2 NEXT     FIXED BIN (15),
                                 2 RULE(NRULES REFER (MAXNRULES)),
                                      3 START FIXED BIN(15),
                                      3 LENGT  FIXED BIN(15);

  18    1             ON ENDFILE(SYSIN) BEGIN;
  19    2                   ENDFILE='1'B;
  20    2                   GOTO SETRULE;
  21    2                   END;
  22    1             GET EDIT(CARD) (A(80));
  23    1     L0:     NONTNAME = '';
  24    1             DO I=2 BY 1 WHILE(SUBSTR(CARD,I,1) ¬='>');
  25    1   1               NONTNAME = NONTNAME||SUBSTR(CARD,I,1);
  26    1   1         END;
              /* I IS POSITION ON CARD */
  27    1             DO J=1 TO NUNONTS;
  28    1   1               IF NONTNAME = SUBSTR(NONTCHAR,NONS(J).START,
                                                      NONS(J).LENGT)
                                 THEN GOTO L1;
  29    1   1         END;


              /* A NEW NONTERMINAL HAS TO BE ADDED */
  30    1             NUNONTS = NUNONTS + 1;
  31    1             NONS(NUNONTS).START = LENGTH(NONTCHAR) + 1;
  32    1             NONS(NUNONTS).LENGT = LENGTH(NONTNAME);
  33    1                   NRULES = 5 ;
  34    1                   ALLOCATE RULED;
  35    1                   NONT(NUNONTS)      = Q;
  36    1                   RULED.NEXT      = 1;
  37    1             NONTCHAR = NONTCHAR||NONTNAME;
  38    1             J = NUNONTS;
              /* J CONTAINS NONTERMINAL IDENTIFIER */
  39    1     L1:     L = 0;
```

```
STMT LEV NT

  40    1                 STRNAME=SUBSTR(CARD,73,1);
  41    1                 DO K=1 BY 1 TO NUSTRS;
  42    1   1                   IF STRNAME=SUBSTR(STRCHAR,STR(K).START,1                )
                                THEN GOTO LA;
  43    1   1             END;
  44    1                 NUSTRS=NUSTRS+1;
  45    1                 STR(NUSTRS).START=LENGTH(STRCHAR)+1;
  46    1                 STRCHAR=STRCHAR||STRNAME;
  47    1         LA:     RULES(NEXTRULES).N=K;
  48    1                 RULES(NEXTRULES).TYPE='T';
  49    1                 NEXTRULES=NEXTRULES+1;
                /* NOW PROCESS THE RHS OF THE CARD */
  50    1       L2:       L = L+1;
  51    1       L3:       IF I >= 71 THEN DO;
  52    1   1                   GET COPY EDIT(CARD) (A(80));
  53    1   1                   IF SUBSTR(CARD,1,1) = '<' THEN GOTO SETRULE;
                                                          ELSE I = 1;
  54    1   1                              END;
  55    1   1                         ELSE I = I+1;
  56    1                 IF SUBSTR(CARD,I,1) = ' ' THEN GOTO L3;
  57    1                 IF SUBSTR(CARD,I,1) = '<' THEN GOTO NONTERM;
  58    1                  GOTO CHAR;
  59    1       NONTERM:NONTNAME = '';
  60    1                 DO I = I+1 BY 1 WHILE (SUBSTR(CARD,I,1) ¬= '>');
  61    1   1                   NONTNAME = NONTNAME||SUBSTR(CARD,I,1);
  62    1   1             END;
  63    1                 DO K=1 TO NUNONTS;
  64    1   1                   IF NONTNAME = SUBSTR(NONTCHAR,NONS(K).START,
                                                              NONS(K).LENGT)
                                     THEN GOTO L4;
  65    1   1             END;
  66    1   1
                /* A NEW NONTERMINAL HAS TO BE ADDED */
  67    1                 NUNONTS = NUNONTS + 1;
  68    1                 NONS(NUNONTS).START = LENGTH(NONTCHAR) + 1;
  69    1                 NONS(NUNONTS).LENGT = LENGTH(NONTNAME);
  70    1                       NRULES = 5 ;
  71    1                       ALLOCATE RULED;
  72    1                       NONT(NUNONTS)      = Q;
  73    1                       RULED.NEXT      = 1;
  74    1                 NONTCHAR = NONTCHAR||NONTNAME;
  75    1                 K=NUNONTS;
                /* INSERT VALUE IN RULE DICTIONARY */
  76    1       L4:       RULES(NEXTRULES).TYPE ='N';
  77    1                 RULES(NEXTRULES).N =K;
  78    1                 NEXTRULES = NEXTRULES + 1;
                /* RETURN TO PROCESS NEXT ELEMENT ON RHS OF RULE */
  79    1                 GOTO L2;
  80    1       CHAR:     STRNAME =  SUBSTR(CARD,I,1);
  81    1                 DO K=1 BY 1 TO NUSTRS;
  82    1   1                   IF STRNAME = SUBSTR(STRCHAR,STR(K).START,1)
                                THEN GOTO L5;
  83    1   1             END;
                /* A NEW CHARACTER STRING HAS TO BE ADDED */
  84    1                 NUSTRS = NUSTRS + 1;
  85    1                 STR(NUSTRS).START = LENGTH(STRCHAR) + 1;
  86    1                 STRCHAR = STRCHAR||STRNAME;
  87    1                 K = NUSTRS;
                /* INSERT VALUE IN RULE DICTIONARY */
  88    1       L5:       RULES(NEXTRULES).TYPE = 'T';
  89    1                 RULES(NEXTRULES).N = K;
  90    1                 NEXTRULES = NEXTRULES + 1;
                /* RETURN TO PROCESS NEXT ELEMENT ON RHS OF RULE */
  91    1                 GOTO L2;
                /* ADD RULE TO RULE DICTIONARY - ALLOCATING RULED AS NECESSARY */
  92    1       SETRULE:P = NONT(J)      ;
  93    1                 IF P->RULED.MAXNRULES = P->RULED.NEXT      - 1
                                THEN DO;
  94    1   1                   NRULES = P-> RULED.MAXNRULES + 5 ;
  95    1   1                   ALLOCATE RULED;
  96    1   1                   RULED.NEXT      = P->RULED.NEXT     ;
```

```
STMT LEV NT

  97    1   1                   DO M=1 BY 1 TO P-> RULED.NEXT      - 1;
  98    1   2                       RULED.RULE(M) = P->RULED.RULE(M);
  99    1   2                   END;
 100    1   1                   NRULES=P->RULED.MAXNRULES;
 101    1   1                   FREE P->RULED;
 102    1   1                   P,NONT(J)      = Q;
 103    1   1                   END;
 104    1               M = P-> RULED.NEXT     ;
 105    1               P->RULED.NEXT     =P->RULED.NEXT     +1;
 106    1               P->RULED.RULE(M).START = NEXTRULES - L + 1 ;
 107    1               P->RULED.RULE(M).LENGT = L - 1 ;
              /* RETURN FOR NEXT CARD IF THERE ARE ANY */
 108    1               IF ENDFILE = '0'B THEN GOTO L0;
 109    1         END;
```

This appendix contains that part of the conventionally written version of the program which reads in and stores the string to be recognised.

```
STMT LEV NT

   1            IPSTR:PROC;
   2    1       DCL (LENGTH,SUBSTR) BUILTIN;
   3    1                 DCL J FIXED BIN(15);
   4    1                 DCL CARD CHAR(80);
   5    1                 DCL  BFLAG BIT(1) INIT('1'B);
   6    1                DCL   INPUT CHAR(5000) VARYING EXTERNAL;
   7    1                INPUT = '';
   8    1       GET1:    GET      FILE(IN) EDIT(CARD) (A(80));
   9    1                IF SUBSTR(CARD,1,1) = '"' THEN   /* 7,8 AS END I/P FILE */
                           RETURN;
  10    1                DO J=1 BY 1 TO 72;
  11    1   1                IF SUBSTR(CARD,J,1) ¬= ' ' THEN DO;
  12    1   2                    INPUT=INPUT||SUBSTR(CARD,J,1);
  13    1   2                    BFLAG='0'B;
  14    1   2                                                    END;
  15    1   1                                                 ELSE DO;
  16    1   2                    IF BFLAG='0'B THEN DO;
  17    1   3                        BFLAG='1'B;
  18    1   3                                                END;
  19    1   2                                                    END;
  20    1   1            END;
  21    1                GOTO GET1;
  22    1       END;
```

This appendix contains a PL/I program to check that the various restrictions, which were relied on during the development, have been met. As mentioned, the relation of the stored to the given grammar is checked by printing out the former. The look-ahead character, if present, is printed between asterisks after the production arrow.

```
STMT LEV NT

   1            IPTEST:
                PROC;
   2     1      DCL (LENGTH,SUBSTR) BUILTIN;
   3     1                DCL  1 NONT(500) EXTERNAL PTR;
   4     1                 DCL 1 STR(250) EXTERNAL,
                                    2 START FIXED BIN(15);
   5     1                 DCL STRCHAR CHAR(4000) VARYING INIT('') EXTERNAL;
   6     1                 DCL 1 RULES(4000) EXTERNAL,
                                    2 TYPE CHAR(1),
                                    2 N FIXED BIN(15);
   7     1                 DCL 1 RULED BASED    ,
                                    2 MAXNRULES FIXED BIN(15),
                                    2 NEXT      FIXED BIN (15),
                                    2 RULE(NRULES REFER (MAXNRULES)),
                                           3 START FIXED BIN(15),
                                           3 LENGT  FIXED BIN(15);
   8     1                 DCL NRULES FIXED BIN(15) EXTERNAL;
   9     1                 DCL NUNONTS FIXED BIN(15) EXTERNAL;
  10     1                 DCL 1 NONS(500) EXTERNAL,
                                    2 START FIXED BIN(15),
                                    2 LENGT FIXED BIN(15),
                                    2 PTR PTR;

  11     1                 DCL NONTCHAR CHAR(3000) VARYING INIT('') EXTERNAL;

  12     1                 DCL I         FIXED BIN(15) ;
  13     1                 DCL J         FIXED BIN(15) ;
  14     1                 DCL K         FIXED BIN(15) ;
  15     1                 DCL P PTR;
  16     1      DCL N FIXED BIN(15);
  17     1      DCL R FIXED BIN(15);
  18     1      DCL NP PTR;
  19     1      DCL LK CHAR(1);
  20     1      DCL T BIT(2);
  21     1      DCL 1 TEMP , 2 TYPE CHAR(1),
                             2 N FIXED BIN(15);
  22     1          CALL PRINTDIC;
  23     1          PUT EDIT ('INPUT TEST') (PAGE,A);
  24     1          NP =NONT(1B)     ;
  25     1          PUT EDIT(' R1 - TEST NO OF ROOT(S) RULES: ')(SKIP,A);
  26     1          IF NP -> RULED.NEXT = 2 THEN
                       PUT EDIT ('R1 OK') (      A);
  27     1                                ELSE
                       PUT EDIT ('R1 FAILED') (      A);
  28     1          PUT EDIT(' R0- LENGTH OF RHS',' R3 - S-LK IS OK')(SKIP,A,SKIP,A);
  29     1          DO N = 1B BY 1B TO NUNONTS;
  30     1   1         NP = NONT(N)      ;
  31     1   1         DO R = 1B BY 1B TO NP->RULED.NEXT - 1B;
  32     1   2            IF NP->RULED.RULE(R).LENGT = 0 THEN
                             PUT EDIT ('R0 FAILED FOR ',N,R)(SKIP,A,F(5),F(5));
  33     1   2            LK = SUBSTR(STRCHAR,STR(RULES(NP->RULED.RULE(R).START -1B).N)
                                      .START,1);
  34     1   2            IF LK¬= ' ' THEN
                             DO;
  35     1   3               PUT EDIT (N,R)               (SKIP,F(5),F(5));
  36     1   3               TEMP = RULES(NP->RULED.RULE(R).START      );
```

```
STMT LEV NT


  37    1  3              IF TEMP.TYPE = 'T' THEN
                             IF SUBSTR(STRCHAR,STR(TEMP.N).START,1) = LK THEN
                                PUT EDIT (' OK')(A);
  38    1  3                                                                    ELSE
                                PUT EDIT (' FAILED')(A);
  39    1  3                                            ELSE
                                DO;
  40    1  4                       T = TEST_LK(OB,LK,NONT(TEMP.N)     );
  41    1  4                       IF SUBSTR (T,2,1) = '0'B THEN
                                      PUT EDIT (' FAILED   ')(A);
  42    1  4                                                   ELSE
                                      IF SUBSTR(T,1,1) = '0'B THEN
                                         PUT EDIT (' UNDECIDED')(A);
  43    1  4                                                     ELSE
                                         PUT EDIT (' OK     ')(A);
  44    1  4                    END;
  45    1  3              END;
  46    1  2        END;
  47    1  1     END;


  48    1     PRINTDIC:
              PROC;
  49    2     DCL XREF CHAR(120) VARYING;
  50    2     DCL BL CHAR(120) INIT (' ');
  51    2     DCL CHAR BUILTIN;
  52    2        PUT PAGE;
  53    2        DO J= 1B TO NUNONTS;
  54    2  1        P = NONT(J)     ;
  55    2  1        IF P-> RULED.NEXT = 1B THEN
                       PUT    EDIT ('** NO RULES FOR ** ','<',
                                    SUBSTR(NONTCHAR,NONS(J).START,NONS(J).LENGT),
                                    '>')
                                   (SKIP,A,A,A(NONS(J).LENGT),A);
  56    2  1                               ELSE
                       DO I = 1B TO P->RULED.NEXT-1B;
  57    2  2              XREF = '' ;
  58    2  2              PUT EDIT('<',SUBSTR(NONTCHAR,NONS(J).START,NONS(J).LENGT),
                                   '>','->')
                                  (SKIP,A,A(NONS(J).LENGT),A,X(2),A(2));
  59    2  2              XREF = XREF || SUBSTR(BL,1,NONS(J).LENGT - 1B ) ||
                                        SUBSTR(CHAR(J,9),7,3)  || '    ';
  60    2  2              IF SUBSTR(STRCHAR,STR(RULES(P->RULED.RULE(I).START -1B).N)
                                    .START,1) ¬= ' ' THEN
                             DO;
  61    2  3                    PUT EDIT('*',SUBSTR(STRCHAR,STR(RULES(P->RULED.
                                        RULE(I).START-1B).N).START,1),'*')
                                       (A,A(1),A);
  62    2  3                    XREF = XREF || '   ';
  63    2  3                 END;
  64    2  2              DO K = 1B TO P-> RULED .RULE(I).LENGT;
  65    2  3                 TEMP = RULES(P->RULED.RULE(I).START + K -1B);
  66    2  3                 IF TEMP.TYPE ='N' THEN
                                DO;
  67    2  4                       PUT EDIT('<',SUBSTR(NONTCHAR,NONS(TEMP.N).START,
                                                       NONS(TEMP.N).LENGT),'>')
                                           (X(1),A,A(NONS(TEMP.N).LENGT),A);
  68    2  4                       XREF = XREF || SUBSTR(BL,1,NONS(TEMP.N).LENGT) ||
                                                    SUBSTR(CHAR(TEMP.N,9),7,3);
  69    2  4                    END;
  70    2  3                                   ELSE
                                DO;
  71    2  4                       PUT EDIT(SUBSTR(STRCHAR,STR(TEMP.N).START,1))
                                           (X(1),A(1));
  72    2  4                       XREF = XREF || '  ';
  73    2  4                    END;
  74    2  3              END;
  75    2  2              PUT EDIT(XREF) (SKIP,A(120));
  76    2  2           END;
  77    2  1     END;
```

```
STMT LEV NT

  78   2     END;
  79   1        TEST_LK:
                PROC (DEPTH,CHAR,NT) RECURSIVE RETURNS(BIT(2));
  80   2        DCL DEPTH FIXED BIN(15);
  81   2        DCL CHAR CHAR(1);
  82   2        DCL NT PTR;
  83   2        DCL RESULT BIT(2);
  84   2        DCL R FIXED BIN(15);
  85   2        DCL 1 TEMP , 2 TYPE CHAR(1),
                             2 N FIXED BIN(15);
  86   2           IF DEPTH = 101B THEN
                      RETURN ('01'B);
  87   2                                   ELSE
                      DO;
  88   2  1              RESULT ='11'B ;
  89   2  1              DO R = 1B BY 1B TO NT->RULED.NEXT - 1B;
  90   2  2                 TEMP = RULES(NT->RULED.RULE(R).START)    :
  91   2  2                 IF TEMP.TYPE = 'T' THEN
                               IF SUBSTR(STRCHAR,STR(TEMP.N).START,1) ¬= CHAR THEN
                                  RESULT = RESULT & '10'B;
  92   2  2                                                                    ELSE
                                  RESULT = RESULT & '11'B;
  93   2  2                                    ELSE
                               RESULT = RESULT & TEST_LK(DEPTH+1B,CHAR,
                                                           NONT(TEMP.N)    );
  94   2  2              END;
  95   2  1              RETURN(RESULT);
  96   2  1           END;
  97   2        END;
  98   1     END;
```

This appendix contains a listing of a run of the recogniser.

```
<ROOT>  -> <NUMBER>
     1             2
<NUMBER>  -> <UNSIGNED_NUMBER>
       2                      9
<NUMBER>  ->*+* + <UNSIGNED_NUMBER>
       2                           9
<NUMBER>  ->*-* - <UNSIGNED_NUMBER>
       2                           9
<DIGIT>  ->*0* 0
      3
<DIGIT>  ->*1* 1
      3
<DIGIT>  ->*2* 2
      3
<DIGIT>  ->*3* 3
      3
<DIGIT>  ->*4* 4
      3
<DIGIT>  ->*5* 5
      3
<DIGIT>  ->*6* 6
      3
<DIGIT>  ->*7* 7
      3
<DIGIT>  ->*8* 8
      3
<DIGIT>  ->*9* 9
      3
<UNSIGNED_INTEGER>   -> <DIGIT>
                  4           3
<UNSIGNED_INTEGER>   -> <UNSIGNED_INTEGER> <DIGIT>
                  4                      4       3
<INTEGER>  -> <UNSIGNED_INTEGER>
        5                       4
<INTEGER>  ->*+* + <UNSIGNED_INTEGER>
        5                            4
<INTEGER>  ->*-* - <UNSIGNED_INTEGER>
        5                            4
<DECIMAL_FRACTION>  ->*.* . <UNSIGNED_INTEGER>
                  6                            4
<EXPONENT_PART>  ->*'* ' <INTEGER>
               7                  5
<DECIMAL_NUMBER>  -> <UNSIGNED_INTEGER>
                8                      4
<DECIMAL_NUMBER>  ->*.* <DECIMAL_FRACTION>
                8                         6
<DECIMAL_NUMBER>  -> <UNSIGNED_INTEGER> <DECIMAL_FRACTION>
                8                      4                  6
<UNSIGNED_NUMBER>  -> <DECIMAL_NUMBER>
                 9                    8
<UNSIGNED_NUMBER>  ->*'* <EXPONENT_PART>
                 9                      7
<UNSIGNED_NUMBER>  -> <DECIMAL_NUMBER> <EXPONENT_PART>
                 9                   8                7
```

```
INPUT TEST
 R1 - TEST NO OF ROOT(S) RULES: R1 OK
 R0- LENGTH OF RHS
 R3 - S-LK IS OK
     2     2 OK
     2     3 OK
     3     1 OK
     3     2 OK
     3     3 OK
     3     4 OK
     3     5 OK
     3     6 OK
     3     7 OK
     3     8 OK
     3     9 OK
     3    10 OK
     5     2 OK
     5     3 OK
     6     1 OK
     7     1 OK
     8     2 OK
     9     2 OK
 -12.3'-4
  YES
```