THEORETICAL ISSUES OF THE IMPLEMENTATION OF PROGRAMMING LANGUAGES

by

Marvin Solomon

Computer Sciences Technical Report #300
June 1977

THEORETICAL ISSUES IN THE IMPLEMENTATION OF PROGRAMMING LANGUAGES*

by

Marvin Solomon

Technical Report 300

ABSTRACT

This report presents theoretical results about two issues relevant to the implementation of programming languages. The first issue concerns data types in a highly typed algorithmic language such as Algol 68. It has long been known that equivalence of types in such a language may be decided by an algorithm analogous to the equivalence algorithm for finite automata. In an earlier paper, we made this analogy precise by modelling data types with a lattice theoretical model based on ideas of Dana Scott. In chapter I of this work, we use this model to show that the similarity to finite automata no longer holds if parameters may be used in the definition of types. In fact, the types thus defined more closely resemble the deterministic context-free languages. More precisely, the equivalence problem for types defined using parameters is decidable if and only if the equivalence problem for deterministic context-free sets is decidable. The decidability of the latter equivalence problem is a well-known open question.

In chapter II, we consider the definition of programming languages by means of infinite grammars. We note that many formalisms for defining programming languages may be viewed as infinite grammars, including van Wijngaarden grammars, the attribute grammars of Knuth, and the indexed grammars of Aho. The definitions of unambiguous and LR(k) grammars may be generalized directly to infinite grammars. We show that if an infinite grammar is specified by an indexed grammar, then parsing may be carried out by an algorithm analogous to the SLR(1) algorithm of DeRemer. The key to the technique involves showing that the relations and sets of "items" which arise, although infinite, may be represented by finite regular expressions.

*This report constitutes the author's Ph.D. thesis at Cornell University.

Acknowledgements

I would like to express my sincere gratitude to the many people who supported me through the long, hard struggle to produce this document. First, to my parents for their love and support throughout my academic career. Next, to the Computer Science ddpartment of Cornell University for patience that was almost excessive, to my fellow graduate students, especially Janos Simon for being most excellent sounding boards, and to the members of my special committee: John Williams, Robert Constable, Anil Nerode, and above all Alan Demers, without whom this dissertation might never have been written. I believe it was John Hopcroft who defined a Ph.D. thesis as "a piece of research carried out by a faculty member under very difficult circumstances."

Portions of the research reported here were completed while the author was on the faculties of the University of Aarhus, Denmark, and the University of Wisconsin--Madison. I would like to thank Unix Nroff for patient and careful typing of this manuscript and Frank Horn for a remarkable job of proofreading.

<u>Table of Contents</u>

Chapter I

TYPE DEFINITIONS WITH PARAMETERS

1. INTRODUCTION

Several authors [9,11,13,32] have pointed out the usefulness of parameterized data types in a highly typed language such as ALGOL 68 [30] or Pascal [15]. For example, a general queuing facillity might be defined by

(1.1) type queue(x) = struct(x, ref queue(x))

and later used by including definitions such as

(1.2) type intlist = queue(int)

(1.3) type waitingline = queue(person)

(where person is some programmer-defined data type). We shall adopt the terminology of [13, 32] and call an object like queue a modal.

It should come as no great surprise that allowing modals creates new difficulties for the implementor. For example, if a parameter to a procedure can be declared to be of type queue(x), then the code generated may depend heavily on the size of objects of type x, and even if x is passed as parameter (of type type), a good deal of run-time checking may be needed. Lindsey [20] suggests allowing only variables of type ref queue(x), and Gehani [9] discusses restrictions under which several copies of the troublesome procedure may be generated.

All these problems arise from attempts to use modals in

variable declarations. The purpose of chapter I of this dissertation is to point out a more subtle problem that remains even if modals are never used in variable declarations, but only as a tool for defining (ordinary) types as in (1.1-3). The problem in question is to decide when two definitions are equivalent. As Kral [17] points out, the algorithm for equivalence of modes in ALGOL 68 is similar to the equivalence algorithm for finite automata. As we showed in [26], this similarity is due to the fact that modes defined by ALGOL 68-like mode declarations are regular, in the sense that they are characterized by regular sets of strings. With the introduction of parameters, this is no longer true; the definable types are now characterized by arbitrarily complex deterministic context-free languages. The result of this is that the equivalence problem becomes of the same level of difficulty as the equivalence problem for deterministic languages. Since the latter problem is at present a well-known open problem, we see that the addition of parameters adds an essential complication.

Gramatici certant, et adhuc sub judice res est.

(The grammarians are at variance and the matter is still undecided.)

Horace, Ars Poetica

1.1 Equivalence of Modes

The foregoing discussion presumes that there is an obvious canonical notion of "equivalence of types". This is not true. One school holds that any two distinct definitions define different types. Of course in this case the equivalence problem is trivial. We will take a point of view closer to that of ALGOL 68, one which has been supported in [18] and [26]. Briefly, we say that two types are equivalent if all values of one type are "the same shape as" values of the other type. To be somewhat more specific, we assume that there are atomic values partitioned into disjoint atomic type classes, and structuring operators which construct values from component values. Two values are the same type if they are both of the same atomic type or if they are constructed by the same operator from components which are, respectively, of the same types. This defines (recursively) an equivalence relation: "to be of the same type as", and types are nothing more than equivalence classes. The important point is that the type of a variable x should tell the following about the value of x: either the type of x is atomic; then the value of x is atomic and of that atomic type or the type of x is $f(t_1,\ldots,t_k)$ where f is a structuring operator and $t_1, \ldots, t_k$ are types; then k selection operations are applicable to the value of x and the result of applying the ith is a value of type t_i. By induction, then, the type of a variable is sufficient to determine whether any given finite sequence of selection

operations is applicable to x, and if so, what the type of the result will be. This means that the type information may be summarized as a (possibly infinite) tree.

1.2 Example

Figure 1a gives an example of a type definition which defines a modal of one argument and applies it to a constant type to obtain a new type. Figure 1b "unrolls" the definition by repeatedly replacing occurrences of the variable $v_\emptyset$ by its definition. The result of applying $v_\emptyset$ to int is shown in Figure 1c. As we shall see later, this type could not be defined without parameters. ⊠

The contents of chapter I of this dissertation are as follows: In section 2, we formally present a syntax for a type definition facility with parameters so as to have a concrete example to illustrate our ideas. In section 3, we endow the facility with a formal semantics based on ideas from [18, 22, 24, 26] and elsewhere. The reader who is satisfied that the trees of figures 1b and 1c accurately reflect the "meaning" of the definition in figure 1a may skip this section, at least on first reading. One should, however, look at those paragraphs labeled "notation". Section 4 contains the central results of chapter I (4.2.4 and 4.3.7). We show that types defined in our example language fragment are "deterministic context-free" and hence that the equivalence problem for types is of equal difficulty as the (open) equivalence problem for deterministic languages. Section 5 contains a discussion of these results and their implications for language design.

$$v_\emptyset(x_\emptyset) = \underline{\text{struct}}(x_\emptyset,\ v_\emptyset(\underline{\text{ref}}(x_\emptyset)))$$

$$v_1 = v_\emptyset(\underline{\text{int}})$$

Figure 1a

A type definition

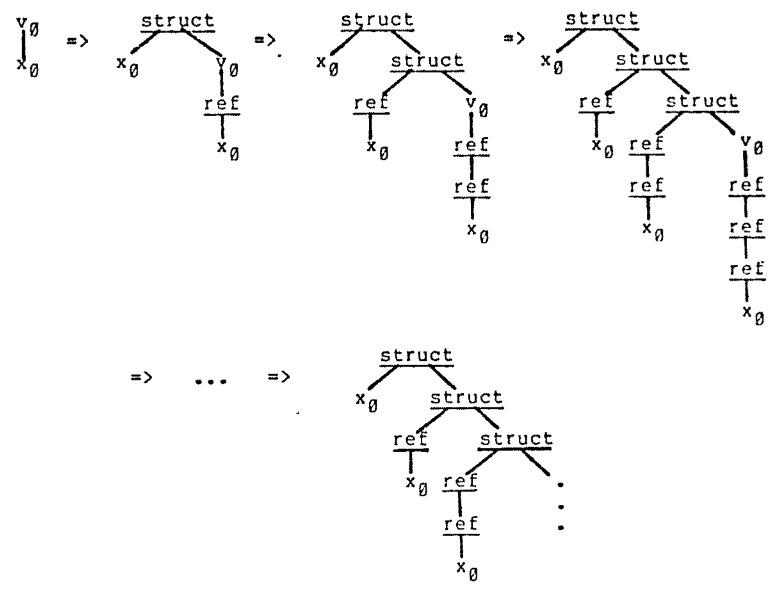


Figure 1b

The "unrolling" of 1a

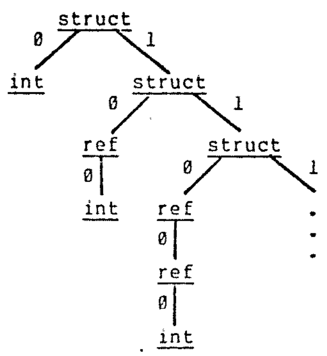


Figure 1c

A mode defined by 1a

2. A TYPE DEFINITION SYSTEM

Our example language fragment will be a stripped-down version of the type declaration facilities of ALGOL 68 [30] or Pascal [15] minimally augmented with the ability to specify a partially defined type as in (1.1) or figure 1a. We assume that the following lexical classes are defined:

- Σ, a set of modal constants. Each $\sigma \in \Sigma$ has a rank (number of arguments) which is a non-negative integer. Modal constants of rank 0 may be called type constants and correspond to atomic types such as int or real. Modal constants of rank > 0 are the built-in type constructors of the language, such as struct and ref of ALGOL 68.
- V, a set of modal variables. These also have ranks associated with them. A variable of rank 0 may be thought of as a type variable. Thus ordinary (non-parameterized) types are a special case of modals. For example, in figure 1a, v_0, $v_1 \in V$ and $\text{Rank}(v_0) = 1$, $\text{Rank}(v_1) = 0$.
- X, a set of parameter symbols. These are all of rank 0.

We now specify the syntax of a modal definition using BNF and some context-sensitive restrictions.

```
        <definition> ::= <declaration>
                       | <definition><declaration>
       <declaration> ::= <modal variable><formal parameters>
                       = <modal expression>
 <formal parameters> ::= <empty> | ( <parameter list> )
    <parameter list> ::= <parameter symbol>
                       | <parameter list> , <parameter symbol>
  <modal expression> ::= <modal constant> <actual parameters>
                       | <modal variable><actual parameters>
                       | <parameter symbol>
 <actual parameters> ::= <empty> | ( <expression list> )
   <expression list> ::= <modal expression>
                       | <expression list> , <modal expression>
```

Restrictions

(1) There is exactly one <declaration> for each <modal variable> appearing in the <definition>.

(2) If $\sigma \in \Sigma \sqcup V$ and $\text{Rank}(\sigma) = 0$, then each <formal parameters> or <actual parameters> following σ is <empty>; if $\text{Rank}(\sigma) = n$, then each <formal parameters> or <actual parameters> following σ has exactly n members.

(3) Each <parameter symbol> appearing in a <declaration> must appear in the <formal parameters> of that <declaration>. The <parameter symbol>s in any <parameter list> are all distinct. ⊠

2.1 Notation

To simplify notation, we will assume for the remainder of this section that we are talking about one fixed definition D, that the set of modal variables is $V = \{v_0, \ldots, v_{n-1}\}$ for some n, and that D has the form:

$$v_0(x_0,\ldots,x_{\mathrm{Rank}(v_0)-1}) = e_0$$

$$(2.1.1) \qquad \vdots$$

$$v_{n-1}(x_0,\ldots,x_{\mathrm{Rank}(v_{n-1})-1}) = e_{n-1}$$

where each e_i is an expression. We also adopt a convention due to B. Rosen [21] and omit mention of ranks of symbols when they are evident from the context or irrelevant. For example, $v_0(x_0,\ldots,x_{\mathrm{Rank}(v_0)-1})$ is written as $v_0(x_0,\ldots,x_{-1})$. In such a context, "-1" is pronounced "last". ⋈

3. SEMANTICS

Once we have constructed T, the set of types, it should be clear that the meaning of a modal t of rank n is a function $\mathrm{Fun}(t):T^n \rightarrow T$. If each modal constant σ is given a standard meaning $\mathrm{Fun}(\sigma)$ and meaning $\mathrm{Fun}(v_i)$ is assigned to each modal variable v_i, then each expression e_i represents a function $\mathrm{Fun}(e_i):T^k \rightarrow T$ in an obvious way. (Here k is any integer larger than any i such that x_i appears in e_i). Since $\mathrm{Fun}(e_i)$ depends on the assignments of $\mathrm{Fun}(v_i)$ to v_i, e_i may also be thought of as a function from $D_0 \times D_1 \times \ldots \times D_{-1}$ to D_i, where D_j is the set of functions from $T^{\mathrm{Rank}(v_j)}$ to T. In this way, a definition may be read as a set of simultaneous equations asserting the equality of various functions over T. Under certain circumstances, the equations have a unique solution. (See [26] for necessary and sufficient conditions for the solution to be unique. A <u>sufficient</u> condition is that no e_i is of the form $v_j(\ldots)$ or v_j.)

The set we use for T is the same as the one we constructed in [26]. See that paper, [18], and section I.1 of this dissertation for motivation for the choice of this model. We briefly review the construction of T and some of its properties.

3.1 Trees and Cpo's

3.1.1 Definition

Let $\mathbb{N}$ denote the set of non-negative integers. $\mathbb{N}^*$ is the set of finite sequences of elements of $\mathbb{N}$ including e, the sequence of length $\emptyset$.

Let Γ be any ranked set (a set together with a function $\mathrm{Rank}:\Gamma \to \mathbb{N}$). A tree over Γ is a partial function $t:\mathbb{N}^* \to \Gamma$ with domain $\mathrm{Dom}(t) \subseteq \mathbb{N}^*$ satisfying:

(3.1.1) If $\alpha n \in \mathrm{Dom}(t)$ for some $\alpha \in \mathbb{N}^*$ and $n \in \mathbb{N}$, then $n < \mathrm{Rank}(t(\alpha))$ and $\beta \in \mathrm{Dom}(t)$ for all prefixes β of α (including α). $\boxtimes$

3.1.2 Notation

Let $T[\Gamma]$ denote the set of trees over Γ. Let $T = T[\Sigma]$. We will write $t[\alpha]$ rather than $t(\alpha)$ to denote the value of t at the string $\alpha \in \mathbb{N}^*$. Let $\perp$ denote the tree with empty domain, and let "$t[\alpha] = \perp$" mean that $\alpha \notin \mathrm{Dom}(t)$. If $\gamma \in \Gamma$, then we also use γ to denote the function $\gamma:T[\Gamma]^{\mathrm{Rank}(\gamma)} \to T[\Gamma]$ where

$$\gamma(t_\emptyset,\ldots,t_{-1})[\alpha] = \begin{cases} \gamma & \text{if } \alpha = e \\ t_i[\beta] & \text{if } \alpha = i\beta \text{ and } i < \mathrm{Rank}(\gamma) \\ \text{undefined otherwise.} \end{cases}$$

It is easy to verify that this satisfies (3.1.1). $\boxtimes$

This gives us a notation for all finite trees as the following remark shows.

3.1.3 Remark

Let $F[\Gamma]$ denote the set of finite trees over Γ (trees with finite domain). Then $F[\Gamma]$ is the least set satisfying:

(3.1.2) $\perp \in F[\Gamma]$, and

if $\gamma \in \Gamma$ and $t_\emptyset,\ldots,t_{-1} \in F[\Gamma]$, then

$\gamma(t_\emptyset,\ldots,t_{-1}) \in F[\Gamma]$.

Moreover, each $t \in F[\Gamma]$ can be written in the form of (3.1.2) in a unique way. $\boxtimes$

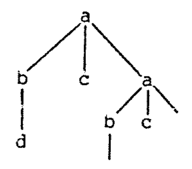
This result allows us to define functions on $F[\Gamma]$ by induction on the complexity of the tree.

3.1.4 Example

The tree defined formally by

$t[e] = a$	$t[\emptyset\emptyset] = d$
$t[\emptyset] = b$	$t[2\emptyset] = b$
$t[1] = c$	$t[21] = c$
$t[2] = a$	$t[\alpha] = \perp$ for all other α

may be illustrated:



and may be denoted by $t = a(b(d),c,a(b(\perp),c,\perp))$. $\boxtimes$

3.1.5 Theorem

(i) The relation $\leq$ on $T[\Gamma]$ defined by

(3.1.3) $t_1 \leq t_2$ if and only if $t_1[\alpha] = t_2[\alpha]$ or $t_1[\alpha] = \perp$ for all $\alpha \in \mathbb{N}^*$

is a partial order (reflexive, transitive, and anti-symmetric).

(ii) $\perp \leq t$ for all $t \in T$

(iii) If $t_\emptyset \leq t_1 \leq \ldots$ is a countable sequence of trees in $T[\Gamma]$, then there is a unique tree, denoted $\mathrm{lub}\{t_i\}$ such that $t_i \leq \mathrm{lub}\{t_i\}$ for all i, and if t is some tree such that $t_i \leq t$ for all i, then $\mathrm{lub}\{t_i\} \leq t$.

Proof

(i) and (ii) are trivial. For (iii), $\mathrm{lub}\{t_i\} = t$ where $t[\alpha] = \gamma$ if there is some $\gamma \neq \perp$ and some i such that $t_i[\alpha] = \gamma$. (There is at most one such γ and i). $t[\alpha] = \perp$ otherwise. ⊠

3.1.6 Definition

A set, together with an element $\perp$ and a relation $\leq$ satisfying the conditions of theorem 3.1.5 is called a cpo (chain-complete poset). ⊠

3.1.7 Theorem

Let $p_n : T[\Gamma] \to F[\Gamma]$ by

$$p_n(t)[\alpha] = \begin{cases} t[\alpha] & \text{if length}(\alpha) < n \\ \perp & \text{otherwise} \end{cases}$$

Given a tree $t \in T[\Gamma]$, let $\Psi(t)$ be the sequence $\langle t_\emptyset, t_1, \ldots \rangle$

of members of $F[\Gamma]$ defined by $t_i = p_i(t)$. Then

(i) $t_i \leq t_{i+1}$ for all i

(ii) $t_i = p_i(t_j)$ whenever $j \geq i$

(iii) $t = \mathrm{lub}\{t_i\}$

(iv) Ψ is an order isomorphism between $T[\Gamma]$ and the set of sequences over $F[\Gamma]$ satisfying (i) and (ii) and ordered component-wise.

Proof

Follows directly from 3.1.5. ⊠

3.1.8 Remark

By the above theorem, we could take 3.1.3 as the definition of $F[\Gamma]$ and define $T[\Gamma]$ to be the set of sequences satisfying (i) and (ii) of 3.1.7. This is the approach taken in [24] and [26]. We prefer the method here (which was inspired by [10]) since it seems more intuitively accessible. ⊠

3.1.9 Definition

If A and B are cpo's, then $f: A \to B$ is continuous if $f(\perp) = \perp$, $x \leq y$ implies $f(x) \leq f(y)$, and $x_\emptyset \leq x_1 \leq \ldots$ implies $\mathrm{lub}\{f(x_i)\} = f(\mathrm{lub}\{x_i\})$. $[A \to B]$ denotes the set of continuous functions from A to B, ordered point-wise, that is, $f \leq g$ if and only if $f(x) \leq g(x)$ for all $x \in A$. $A \times B$ denotes the Cartesian product of A and B ordered by $\langle x_\emptyset, y_\emptyset \rangle \leq \langle x_1, y_1 \rangle$ if and only if $x_\emptyset \leq x_1$ and $y_\emptyset \leq y_1$. ⊠

3.1.10 Proposition

(i) $[A \rightarrow B]$ and $A \times B$ are cpo's.

(ii) If $f \in [A \rightarrow B]$ and $g \in [B \rightarrow C]$, then $g \circ f \in [A \rightarrow C]$

(iii) If $f \in [A \rightarrow B]$ and $g \in [A \rightarrow C]$ then $f \times g \in [A \rightarrow B \times C]$ where $(f \times g)(x) = \langle f(x), g(x) \rangle$.

(iv) $\pi_j \in [A_0 \times \ldots \times A_{-1} \rightarrow A_j]$, where $\pi_j(x_0, \ldots, x_{-1}) = x_j$.

(v) If A is any set, then $2^A = \{B \mid B \subseteq A\}$ is a cpo, where $\bot = \emptyset$, $B \leq C$ if and only if $B \subseteq C$, and $\text{lub}\{B_i\} = \bigsqcup\{B_i\}$. If $f:A \rightarrow 2^B$ is any function, then $\hat{f} \in [2^A \rightarrow 2^B]$ where $\hat{f}(C) = \bigsqcup\{f(x) \mid x \in C\}$. If A is countable, then every $g \in [2^A \rightarrow 2^B]$ is $\hat{f}$ for some $f:A \rightarrow 2^B$.

Proof

The proofs are straightforward and may be found in the standard literature. $\bowtie$

3.1.11 Remarks

(1) If S is any set, then $\hat{S} = S \sqcup \{\bot\}$ is a cpo, where $\bot$ is a new symbol and $x \leq y$ if and only if $x = y$ or $x = \bot$.

(2) $T[\Gamma]$ is a sub-cpo of $[\hat{N}^* \rightarrow \hat{\Gamma}]$.

(3) If A and B are alphabets, then any substitution (as defined in [12, p. 124] or [3, p. 146]) $h:2^{A^*} \rightarrow 2^{B^*}$ is continuous by 3.1.1(v). $\bowtie$

3.2 Substitution Operators

We stated above that under the assignment of functions to modal variables, each expression e_i denotes a function "in an obvious way". We now make this more precise.

Recall from 3.1.2 that each $\sigma \in \Sigma$ also denotes a function $\sigma:T^{\text{Rank}(\sigma)} \rightarrow T$.

3.2.1 Lemma

σ is continuous. $\bowtie$

3.2.2 Definition

Let $\bar{e} = \langle e_0, \ldots, e_{-1} \rangle$ be a sequence of expressions--i.e. members of $E[\Sigma \sqcup V \sqcup X]$ and let $\bar{f} = \langle f_0, \ldots, f_{-1} \rangle$ be a sequence of functions where $f_i:T^{\text{Rank}(v_i)} \rightarrow T$. Then $\bar{e} \leftarrow \bar{f}$ is the function defined by induction on the complexity of $\bar{e}$ as follows:

$$\bar{e} \leftarrow \bar{f} = (e_0 \leftarrow \bar{f}) \times \ldots \times (e_{-1} \leftarrow \bar{f})$$
$$\sigma(e_0, \ldots, e_{-1}) \leftarrow \bar{f} = \sigma \circ (\bar{e} \leftarrow \bar{f})$$
$$v_i(e_0, \ldots, e_{-1}) \leftarrow \bar{f} = f_i \circ (\bar{e} \leftarrow \bar{f})$$
$$(x_i \leftarrow \bar{f}) = \pi_i$$

(By 3.1.3 this defines $\bar{e} \leftarrow \bar{f}$ completely). $\bowtie$

The reader may find this definition easier to understand by noting that $(f \circ (g_0 \times \ldots \times g_{-1}))(x) = f(g_0(x), \ldots, g_{-1}(x))$.

3.2.3 Lemma

(i) If each f_i is continuous, then $\bar{e} \leftarrow \bar{f}$ is continuous.

(ii) The operator E defined by $E(\bar{f}) = \bar{e} \leftarrow \bar{f}$ is continuous. That is, $E \in [D_0 \times \ldots \times D_{-1} \rightarrow D_0 \times \ldots \times$

$D_{-1}]$, where $D_i = [T^{\mathrm{Rank}(v_i)} \to T]$.

Proof

(i) follows from 3.1.10(ii, iii, and iv) and 3.2.1.

(ii) follows from the fact that the operators $\circ$ and $\times$ are continuous. ⋈

3.2.4 Theorem

If A is any cpo and $f \in [A \to A]$, then f has a fixed point, an element $x \in A$ such that $f(x) = x$. In fact, f has a least fixed point, denoted Yf which may be computed by

$$f_0 = \bot$$
$$f_{n+1} = f(f_n)$$
$$Yf = \mathrm{lub}\{f_n\}$$

⋈

3.2.5 Corollary

For each definition D as in (2.1.1), there is a sequence $F = \langle f_0,\ldots,f_{-1}\rangle$ such that $\bar{f} = \bar{e} \leftarrow \bar{f}$. ⋈

This is the promised semantics for type definitions.

Summarizing, a mode definition D is of the form $\{v_i(x_0,\ldots,x_{-1}) = e_i \mid i = 0, 1, \ldots\}$. A solution to the sequence of equations is a sequence $\bar{f}$ of continuous functions such that $f_i(t_0,\ldots,t_{-1}) = (e_i \leftarrow \bar{f})(t_0,\ldots,t_{-1})$ for all i and all $t_0, \ldots, t_{-1}$.

3.2.6 Notation

We will say that the solution of D is the least fixed point Y computed as in theorem 3.2.4, and the function defined by D is the first component of this fixed point,

$\pi_0(Y)$. Finally, and most important, two definitions are equivalent if they define the same function.

3.3 Functions and Trees

In 3.1.2, we used a symbol $\gamma \in \Gamma$ to denote a function. More generally, many (not all) functions on $T[\Gamma]$ can be represented by trees.

3.3.1 Definition

Let $X_n = \{x_0,\ldots,x_{-1}\}$ and let $t \in T[\Sigma \sqcup X_n]$. Then for all $m \geq n$ and all Γ, such that $\Sigma \subseteq \Gamma$, t denotes a function $\mathrm{Fun}(t):T[\Sigma]^m \to T[\Gamma]$, where $\mathrm{Fun}(t)(t_0\ldots,t_{m-1})$ is the result of replacing each occurrence of x_i by t in t_i. More precisely, let s be the tree:

$$s[\alpha] = t[\alpha] \qquad \text{if } t[\alpha] \in \Sigma$$
$$s[\alpha\beta] = t_i[\beta] \qquad \text{if } t[\alpha] = x_i.$$

It is straightforward to check that this defines a unique tree. Let $\mathrm{Fun}(t)(t_0,\ldots,t_{m-1}) = s$. ⋈

Notice that this definition extends 3.1.2 in the sense that $\sigma = \mathrm{Fun}(\sigma(x_0,\ldots,x_{-1}))$ (as a function). Next, we extend this notion of "replacing occurrences of the symbol τ in t with the tree s" to the case in which $\mathrm{Rank}(\tau) \neq 0$--i.e., in which τ can appear at an interior node.

3.3.2 Definition

Let $\bar{e} = \langle e_0,\ldots,e_{-1}\rangle$ and $\bar{t} = \langle t_0,\ldots,t_{-1}\rangle$ be finite sequences of trees where $e_i \in F[\Sigma \sqcup V \sqcup X]$ and $t_i \in T[\Sigma \sqcup V \sqcup X]$. Then $\bar{e} \leftarrow \bar{t}$, the result of replacing v_i by t_i in $\bar{e}$ is defined by induction on the complexity of $\bar{e}$ as follows:

$$\bar{e} \leftarrow \bar{t} = \langle e_\emptyset \leftarrow \bar{t}, \ldots, e_{-1} \leftarrow \bar{t} \rangle$$
$$\sigma(e_\emptyset, \ldots, e_{-1}) \leftarrow \bar{t} = \sigma(\bar{e} \leftarrow \bar{t})$$
$$v_i(e_\emptyset, \ldots e_{-1}) \leftarrow \bar{t} = \mathrm{Fun}(t_i)(\bar{e} \leftarrow \bar{t})$$
$$x_i \leftarrow \bar{t} = x_i$$

(See, for example, figure 1b or 2a). $\boxtimes$

3.3.3 Remark

The restriction that each e_i be finite is not essential. We could define $\bar{e} \leftarrow \bar{t}$ directly (rather than inductively) for arbitrary $\bar{e}$, but the definition would be much more complicated, and we are only interested in the case of finite e_i. $\boxtimes$

In view of the similarity between definitions 3.2.2 and 3.3.2, it is not surprising that we have the following result.

3.3.4 Lemma

$\mathrm{Fun}(\bar{e} \leftarrow \bar{t}) = \bar{e} \leftarrow \mathrm{Fun}(\bar{t})$. $\boxtimes$

3.3.5 Remark

In the statement of 3.3.4, we implicitly extended the domain of Fun from trees to tuples of trees by letting $\mathrm{Fun}(\langle t_\emptyset, \ldots, t_{-1} \rangle) = \langle \mathrm{Fun}(t_\emptyset), \ldots, \mathrm{Fun}(t_{-1}) \rangle$. We will continue to make such extensions without explicit mention. $\boxtimes$

3.3.6 Lemma

(i) $\mathrm{Fun}: T[\Sigma \sqcup X_n] \rightarrow [T^n \rightarrow T]$ is continuous

(ii) Fun is one-to-one

(iii) $\leftarrow$ as an operation on $T[\Sigma \sqcup V \sqcup X]$ is associative wherever it is defined. That is, $\bar{e} \leftarrow (\bar{e}' \leftarrow \bar{t}) =$

$(\bar{e} \leftarrow \bar{e}') \leftarrow \bar{t}$. $\boxtimes$

3.3.7 Theorem

The minimal fixed point YE of 3.2.3 can be described by trees. More precisely, $YE = \mathrm{Fun}(\bar{t})$, where $\bar{t} = \mathrm{lub}\{\bar{t}^i\}$ and $\bar{t}^i$ is defined by

$$\bar{t}^\emptyset = \langle \bot, \ldots, \bot \rangle$$
$$\bar{t}^{n+1} = \bar{e} \leftarrow \bar{t}^n$$

Proof

By 3.2.4, $YE = \mathrm{lub}\{E^n(\bot)\}$, where $\bot$ is the nowhere defined function.

We will show by induction on n that $E^n(\bot) = \mathrm{Fun}(\bar{t}^n)$:

$E^\emptyset(\bot) = \bot$ (where $\bot$ is the nowhere defined function)
$= \mathrm{Fun}(\bar{\bot})$ (where $\bot$ is the nowhere defined tree)
$= \mathrm{Fun}(\bar{t}^\emptyset)$

$E^{n+1} = E(E^n(\bot))$
$= \bar{e} \leftarrow E^n$ by definition of E
$= \bar{e} \leftarrow \mathrm{Fun}(\bar{t}^n)$ by induction hypothesis
$= \mathrm{Fun}(\bar{e} \leftarrow \bar{t}^n)$ by lemma 3.3.4
$= \mathrm{Fun}(\bar{t}^{n+1})$ by definition of $\bar{t}^n$

Hence $YE = \mathrm{lub}\{E^n(\bot)\}$
$= \mathrm{lub}\{\mathrm{Fun}(\bar{t}^n)\}$
$= \mathrm{Fun}(\mathrm{lub}\{\bar{t}^n\})$ by 3.3.6 (i)
$= \mathrm{Fun}(\bar{t})$

3.3.8 Corollary

The function defined by definition D is $\mathrm{Fun}(\pi_0(\mathrm{lub}\{\bar{t}^n\}))$.

⊠

3.3.9 Notation

Let D be a definition, and $\bar{t} = \mathrm{lub}\{\bar{t}^n\}$ be derived from D as in 3.3.7. Then $\bar{t}$ is said to be the tuple of trees defined by D. The tree defined by D is the first component of this tuple, $\pi_0(\bar{t})$. ⊠

3.3.10 Remark

Although a definition D technically defines a function, by virtue of 3.3.8, we can pretend that D defines a tree: if f_0 is the function defined by D and t_0 is the tree defined by D, then $f_0 = \mathrm{Fun}(t_0)$. Since Fun is one-to-one, two definitions are equivalent if and only if they define the same tree. ⊠

3.3.11 Example

Return to example 1.2. The meaning of figure 1b can now be made more precise. Notice that the tuples $\bar{t}^n$ of 3.3.7 have the property that $\bar{t}^n = \bar{e}^n \leftarrow \langle \bot,\dots,\bot \rangle$, where $\bar{e}^n = \bar{e} \leftarrow \bar{e} \leftarrow \dots \leftarrow \bar{e}$ (n factors) or (by induction)

$$\bar{e}^0 = \langle v_0(x_0,\dots,x_{-1}),\dots,v_{-1}(x_0,\dots,x_{-1})\rangle$$

$$\bar{e}^{n+1} = \bar{e} \leftarrow \bar{e}^n = \bar{e}^n \leftarrow \bar{e} \text{ (by the associativity of } \leftarrow).$$

The first 4 trees of figure 1b represent e_0^2, $e_0^1 = e_0$, $e_0^2 = e_0 \leftarrow \bar{e}$, and $e_0^3 = e_0 \leftarrow \bar{e} \leftarrow \bar{e}$. (Since v_1 does not appear in e_0, we need not show e_1). The last tree in 1b represents t_0, and the tree of 1c represents $\mathrm{Fun}(t_0)(\underline{int}) = t_1$. ⊠

4. TREES AND LANGUAGES

In this section, we establish an intimate connection between modals and deterministic context-free languages, and use this connection to prove the main results of chapter I. The first step has already been taken by noting (in 3.3.10) that we can confine our attention to trees. The next step is to notice that the trees involved all have finite range.

4.1 Lemma

All trees mentioned in theorem 3.3.7 have finite range. (The range of t is $\{\gamma \mid t[\alpha] = \gamma \text{ for some } \alpha \in \mathbb{N}^*\}$).

Proof

Clearly, if t is any of these trees, then $t[\alpha] = \gamma$ only if γ appears somewhere in D. But D contains a finite number of symbols. ⊠

Now let Γ be a (not necessarily finite) ranked set, and let $t \in T[\Gamma]$. Suppose the range of t is finite. Let $n = \max\{\mathrm{Rank}(\gamma) \mid \gamma \in \mathrm{Range}(t)\}$, and let $[n] = \{0,1,\dots,n-1\}$. For each $\gamma \in \mathrm{Range}(t)$, $t^{-1}[\gamma] \subseteq [n]^*$; that is, $t^{-1}[\gamma]$ is a language over the finite alphabet $[n]$. $t^{-1}[\gamma]$ may be thought of as "the set of addresses of nodes where γ lives". This finite collection of languages completely characterizes t. This observation is so important in what follows, that we state it as a theorem.

4.2 Theorem

If $t \in T[\Gamma]$ has a finite range, then there is a finite alphabet $[n] \subseteq \mathbb{N}$ and a finite collection of disjoint languages $L_\gamma \subseteq [n]^*$ such that

(4.1) $t[\alpha] = \gamma$ if and only if $\alpha \in L_\gamma$, and

$t[\alpha] = \perp$ if and only if $\alpha \notin \bigsqcup\{L_\gamma\}$.

$\bowtie$

In [26], we showed that for ordinary type definitions (i.e., those without parameters), these languages are all regular sets, and, conversely, that given any finite collection of regular sets such that (4.1) defines a tree, t is definable by some definition. Notice, however, that the definable tree t shown in 1c has the property that $t^{-1}[\underline{int}] = \{1^n 0^{n+1} \mid n \geq 0\}$ which is not a regular set [12].

4.1 Description Languages

In the previous section we showed how a tree can be described by a finite collection of languages. Here we introduce a device for describing a tree by a single language.

4.1.1 Definition

Let Γ be a ranked set and let $\hat{\Gamma} = \Gamma \sqcup \{\langle\gamma,n\rangle \mid \gamma \in \Gamma$ and $n < \text{Rank}(\gamma)\}$.

Let $\text{Descrip}: T[\Gamma] \to 2^{\hat{\Gamma}^*}$ by

$$\text{Descrip}(t) = \{\langle\gamma_0,n_0\rangle\ldots\langle\gamma_{k-1},n_{k-1}\rangle\gamma_k \mid$$
$$k \geq 0 \text{ and } t[n_0\ldots n_{j-1}] = \gamma_j \text{ for all } j \leq k\}.$$

Note that if Γ is infinite, then $\hat{\Gamma}$ is infinite, but if

Range(t) is finite, then $\text{Descrip}(t) \subseteq A^*$ for some finite subset $A \subseteq \hat{\Gamma}^*$. Descrip(t) is called the description language of t.

$\bowtie$

4.1.2 Lemma

Descrip is continuous.

$\bowtie$

4.1.3 Definition

For each $\gamma \in \Gamma$, let h_γ be the finite substitution ([12, p. 124] or [3, p. 196]) defined by

$h_\gamma(\langle\tau,i\rangle) = \{i\}$ for all $\tau \in \Gamma$

$h_\gamma(\tau) = \emptyset$ if $\tau \neq \gamma$

$h_\gamma(\gamma) = \{e\}$.

Since $h_\gamma(\alpha)$ contains at most one string for any string α, we will regard h_γ as a partial function and write $h_\gamma(\alpha) = \beta$ rather than $h_\gamma(\alpha) = \{\beta\}$.

$\bowtie$

4.1.4 Lemma

$t^{-1}[\gamma] = h_\gamma(\text{Descrip}(t))$.

$\bowtie$

4.1.5 Corollary

$t_0 = t_1$ if and only if $\text{Descrip}(t_0) = \text{Descrip}(t_1)$. $\bowtie$

We have already given two meanings to the operator $\leftarrow$. Now we give a third.

4.1.6 Definition

Let $\hat{\Gamma}$ be as in 4.1.1 where $\Gamma = \Sigma \sqcup V \sqcup X$. Let $M_0,\ldots,M_{-1} \subseteq \hat{\Gamma}^*$, where the number of M's is $|V|$ (the cardinality of V). Let Ψ be the substitution defined by

$$\Psi(v_j) = M_j - \hat{\Gamma}^* X$$

$= \{\alpha \in M_j \mid \alpha \neq \beta x_i \text{ for any } \beta \text{ and } i\}$

$\mathcal{W}(\langle v_j, i\rangle) = M_j/\{x_i\} = \{\alpha \in \hat{\Gamma}^* \mid \alpha x_i \in M_j\}$

$\mathcal{W}(\tau) = \{\tau\}$ for any other $\tau \in \hat{\Gamma}^*$.

Let $L_\emptyset,\ldots,L_{-1} \subseteq \hat{\Gamma}^*$. Then $\langle L_\emptyset,\ldots,L_{-1}\rangle \leftarrow \langle M_\emptyset,\ldots,M_{-1}\rangle$ = $\langle \mathcal{W}(L_\emptyset),\ldots,\mathcal{W}(L_{-1})\rangle$. $\bowtie$

The reason for "overloading" the symbol $\leftarrow$ is the following:

4.1.7 Proposition

Let $\bar{t}$ and $\bar{e}$ be tuples of trees in $T[\Sigma \sqcup V \sqcup X]$. Then $\text{Descrip}(\bar{t} \leftarrow \bar{e}) = \text{Descrip}(\bar{t}) \leftarrow \text{Descrip}(\bar{e})$. $\bowtie$

4.1.8 Corollary

Let t be the tree defined by the definition D. Then

$\text{Desrip}(t) = \pi_\emptyset(\bar{L})$ where

$\bar{L}^\emptyset = \langle\emptyset,\ldots,\emptyset\rangle$

$\bar{L}^{n+1} = \text{Descrip}(\bar{e}) \leftarrow \bar{L}^n$

and $\bar{L} = \bigsqcup\{\bar{L}^n \mid n \geq \emptyset\}$.

Proof

By 3.3.7, 4.1.2, and the fact that $\text{Descrip}(\perp) = \emptyset$. $\bowtie$

4.2 From Definitions to Pushdown Automata

Through Thee will we push down our enemies; through Thy name will we tread them under who rise up against us.

Psalm 44

In this section we show how, given a definition D and a symbol σ appearing in D, we can construct a deterministic

pushdown automaton (dpda) P such that $L(P) = t^{-1}[\sigma]$. Our notation for dpda's is exactly as in [3] (q.v.). The description languages introduced in section 4.1 are not used directly in the construction, but are used in the proof of correctness of the construction. First we present the construction, then we give an example.

4.2.1 Construction

Let D be a definition as in (2.1.1) and let σ be a symbol appearing in D. Let $P = \langle Q, [m], \Delta, \delta, q_\emptyset, Z_\emptyset, F\rangle$ where

Q (the set of states) $= \{\langle i,\alpha\rangle \mid \alpha \in \text{Dom}(e_i)\}$

(A state is a node in some tree of D)

[m] (the input alphabet) $= \{\emptyset,\ldots,m-1\}$, where

$m = \max\{\text{Rank}(\tau) \mid \tau \in \Sigma \text{ appears in D}\}$.

Δ (the stack alphabet) $= \{\langle i,\alpha\rangle \in Q \mid e_i[\alpha] \in V\} \sqcup Z_\emptyset$

(A stack symbol is a node labeled by a modal variable).

$q_\emptyset$ (the initial state) $= \langle\emptyset,e\rangle$

(the root of the first tree in D).

F (the set of final states) $= \{\langle j,\alpha\rangle \in Q \mid e_j[\alpha] = \sigma\}$

(A final state is a node labeled σ).

and δ (the transition function) is defined as follows:

(i) If $e_i[\alpha] \in \Sigma$, then $\delta(\langle i,\alpha\rangle,j,Z) = (\langle i,\alpha j\rangle,Z)$ for all $Z \in \Delta$

(ii) If $e_i[\alpha] = v_k$, then $\delta(\langle i,\alpha\rangle,e,Z) = (\langle k,e\rangle,\langle i,\alpha\rangle,Z)$ for all $Z \in \Delta$

(iii) If $e_i[\alpha] = v_k$, then $\delta(\langle i,\alpha\rangle,e,\langle j,\beta\rangle) = (\langle j,\beta k\rangle,e)$.

$\bowtie$

4.2.2 Example

Let σ, τ, $\rho \in \Sigma$, and let Rank(σ) = 3, Rank(τ) = Rank(ρ) = $\emptyset$. Let D be the definition $v_\emptyset(x_\emptyset,x_1)$ = $\sigma(v_\emptyset(\sigma(\tau,x_\emptyset,\rho),\sigma(\tau,\rho,x_1)),x_\emptyset,x_1)$. Let t be the tree defined by D. The derivation of t is illustrated in figure 2a. Figures 2b and 2c illustrate the action of P on input $\emptyset\emptyset 11\emptyset$. $L(P) = t^{-1}[\tau] = \{\emptyset^m 1^n \emptyset \mid m \geq n \geq 1\} \sqcup \{\emptyset^m 2^n \emptyset \mid m \geq n > 1\}$. (notice that L(P) is not an LL language [3]). ⋈

4.2.3 Theorem

Let P be the dpda constructed in 4.2.1. Then $L(P)$ = $t^{-1}[\sigma]$, where t is the tree defined by D. ⋈

4.2.4 Corollary

If the dpda equivalence problem is decidable, then the type equivalence problem is decidable.

Proof

Given two definitions D and D', D is equivalent to D' if and only if $t_\emptyset = t'_\emptyset$, where $t_\emptyset$ and $t'_\emptyset$ are the trees defined by D and D', by 3.3.1$\emptyset$. For each symbol σ appearing in D or D', construct dpda's P_σ and P'_σ according to 4.2.1. Then $t_\emptyset = t'_\emptyset$ if and only if $L(P_\sigma) = t_\emptyset^{-1}[\sigma] = t'^{-1}_\emptyset[\sigma] = L(P'_\sigma)$ for each σ. ⋈

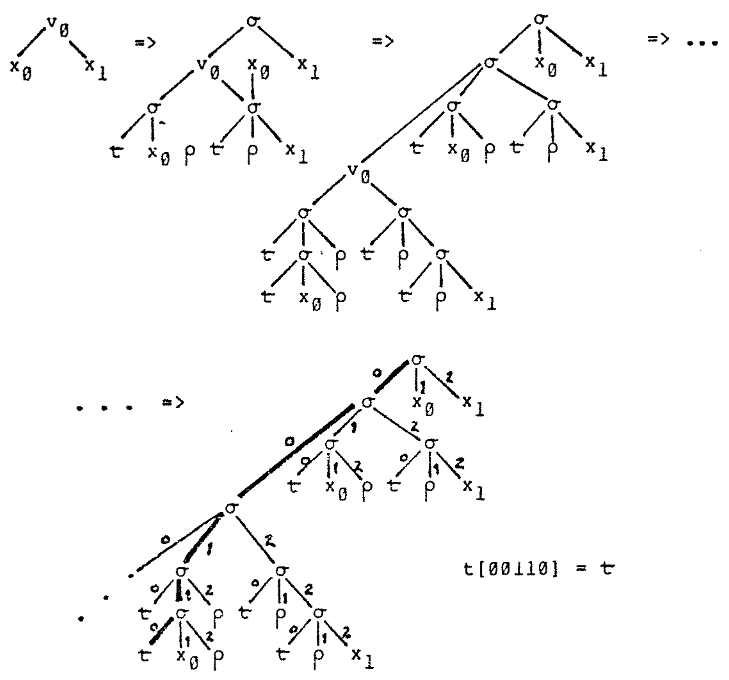
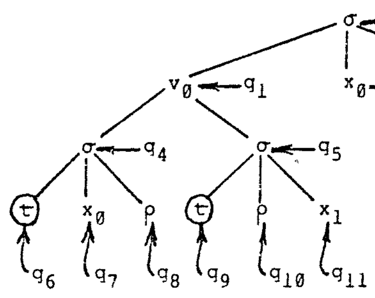


Figure 2a

The mode defined by example 4.2.2



States of P

(Final states are circled)

Figure 2b

States of the pushdown automaton corresponding to t in figure 2a

State	Stack	Remaining Input	Move
q_0	z_0	00110	i
q_1	z_0	0110	ii
q_0	q_1z_0	0110	i
q_1	q_1z_0	110	ii
q_0	$q_1q_1z_0$	110	i
q_2	$q_1q_1z_0$	10	iii
q_4	q_1z_0	10	i
q_7	q_1z_0	0	iii
q_4	z_0	0	i
q_6	z_0	e	accept

Figure 2c

The actions of the automaton of figure 2b on input 00110

4.3 From Pushdown Automata to Definitions

By theorem 4.2.3, if t is a definable tree, then there is a finite set of deterministic languages $\{L_\sigma\}$ which completely characterize t in the sense of (4.1). Unfortunately, the converse is not true. Even if t has finite range and $t^{-1}[\sigma]$ is deterministic context-free for all σ, it may not be the case that t is definable. Fortunately, the converse of 4.2.4 does not need the full converse of 4.2.3; a weaker result suffices. Given any deterministic context-free language L (satisfying certain properties), there is a definable tree t built up from the symbols σ and t such that $t^{-1}[\sigma] = L$ and $t^{-1}[t]$ is completely determined by L. The first step is to reduce an arbitrary dpda to an easily manageable form. Fortunately, this has already been done in [3].

4.3.1 Definition (quoted from [3, p. 691])

A dpda $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is in normal form if it has all the following properties:

(1) P is loop-free. Thus, on each input, P can make only a bounded number of moves.

(2) F has a single member, q_f, and if $(q_0, w, Z_0) \vdash^* (q, e, \gamma)$, then $\gamma = Z_0$. That is, if P accepts an input string, then P is in the final state q_f and the pushdown list consists of the start symbol alone.

(3) Q can be written as $Q = Q_s \sqcup Q_w \sqcup Q_e \sqcup \{q_f\}$, where Q_s, Q_w, and Q_e are disjoint sets, called the scan, write, and erase states, respectively; q_f is in none

of these three sets. The states have the following properties:

(a) If q is in Q_s, then for each $a \in \Sigma$, there is some state p_a such that $\delta(q,a,Z) = (p_a,Z)$ for all Z. Thus, if P is in a scan state, the next move is to scan the input symbol. In addition, this move is always independent of the symbol on the top of the pushdown list.

(b) If q is in Q_w, then $\delta(q,e,Z) = (p,YZ)$ for some p and Y and for all Z. A write state always prints a new symbol on top of the pushdown list, and the move is independent of the current input symbol and the symbol on top of the pushdown list.

(c) If q is in Q_e, then for each $Z \in \Gamma$, there is some state p_Z such that $\delta(q,e,Z) = (p_Z,e)$. An erase state always removes the topmost symbol from the pushdown list without scanning a new input symbol.

(d) $\delta(q_f,a,Z) = \emptyset$ for all a in $\Sigma \sqcup \{e\}$ and $Z \in \Gamma$. No moves are possible in the final state.

(4) If $(q,w,Z) \vdash^+ (p,e,Z)$, then $w \neq e$. That is, a sequence of moves which (possibly) enlarges the stack and returns to the same level cannot occur on e input. A sequence of moves $(q,w,Z) \vdash^+ (p,e,Z)$ will be called a <u>traverse</u>. Note that the possibility or impossibility of a traverse for given q, p, and w is independent of Z, the symbol on the top of the pushdown list.

In short, a scan state reads the next input symbol, a write state prints a new symbol on the stack, and an erase state examines the top stack symbol, erasing it. Only scan states may shift the input head. $\boxtimes$

<u>4.3.2</u> <u>Theorem</u> (<u>quoted from</u> [3, p. 691])

If $L \subseteq \Sigma^*$ is a deterministic language, and ¢ is not in Σ, then $L¢$ is $L(P)$ for some dpda P in normal form. $\boxtimes$

We will assume henceforth that $\Sigma = \{0, \ldots, n-1\}$ for some n.

<u>4.3.3</u> <u>Definition</u>

Let P be a dpda in normal form. Then the function δ may be characterized by three functions: The <u>scan goto</u> function f, the <u>push-pop</u> function g, and the <u>push-goto</u> function h defined as follows:

$f:Q_s \sqcup \Sigma \rightarrow Q$ where for all $Z \in \Gamma$, $q_i \in Q_s$, $j \in \Sigma$

$$\delta(q_i,j,Z) = (f(q_i,j),Z)$$

$$g:Q_w \times Q_e \rightarrow Q \text{ and } h:Q_w \rightarrow Q - Q_e$$

where for all $Z \in \Gamma$, $q_i \in Q_w$, and $p_j \in Q_e$,

$$\delta(q_i,e,Z) = (h(q_i),Y_iA)$$

for some $Y_i \in \Gamma$ depending only on q_i such that

$$\delta(p_j,e,Y_i) = (g(q_i,p_j),e)$$

(Notice that $h(q_i) \notin Q_e$, since otherwise there would be a traverse

$$(q_i,e,Z) \vdash (h(q_i),e,Y_iZ) \vdash (g(q_i,p_j),e,Z)$$

contradicting point (4) of 4.3.1. $\bowtie$

Now we can describe the construction of a type definition from a normal form dpda P.

4.3.4 Construction

Let P be a dpda in normal form. Without loss of generality, we may assume that $q_\emptyset \in Q_e$. Let $\Sigma = \{\sigma, t\}$, where $\mathrm{Rank}(\sigma)$ = (size of the input alphabet of P), and $\mathrm{Rank}(t) = \emptyset$. Let $V = \{v_\emptyset, \dots, v_{-1}\}$ where $|V| = |Q - Q_e|$ and $\mathrm{Rank}(v_i) = |Q_e|$ for each i. Let $\bar{x}$ denote $\langle x_\emptyset, \dots, x_{|Q_e|-1} \rangle$. Arbitrarily order Q_e and $Q - Q_e$ so that $Q_e = \{p_\emptyset, \dots, p_{-1}\}$, $Q - Q_e = \{q_\emptyset, \dots, q_{-1}\}$, and $q_\emptyset$ is the initial state.

For each q_i, let $k(q_i)$ be the expression (tree) "$v_i(\bar{x})$" $\in T[\Sigma \sqcup V \sqcup X]$ and for each p_i, let $k(p_i)$ be the expression "x_i" $\in T[\Sigma \sqcup V \sqcup X]$.

Let D be the definition

$$v_\emptyset(\bar{x}) = e_\emptyset$$
$$\vdots$$
$$v_{-1}(\bar{x}) = e_{-1}$$

where

$$e_i = \begin{cases} \sigma(k(f(q_i,\emptyset)), \dots, k(f(q_i,n-1))) & \text{if } q_i \in Q_s \\ v_j(k(g(q_i,p_\emptyset)), \dots, k(g(q_i,p_{-1}))) & \text{if } q_i \in Q_w \\ & \quad \text{and } h(q_i) = q_j \\ t & \text{if } q_i = q \end{cases}$$

$\bowtie$

4.3.5 Example

Let P be the normal form dpda in figure 3a. $L(P) = \{\emptyset^n 1 \emptyset^n 1 \mid n \geq \emptyset\}$. The function used and the definition constructed according to 4.3.4 are illustrated in figure 3b.

4.3.6 Theorem

Let P be a dpda in normal form and let D be the definition constructed from P by 4.3.4. Let $t_\emptyset$ be the tree defined by D. Then

$$t_\emptyset[\alpha] = \begin{cases} t & \text{if } \alpha \in L(P) \\ \perp & \text{if } \alpha \notin L(P) \text{ but } \alpha = \beta\gamma \text{ for some } \beta \in L(P) \\ \sigma & \text{otherwise} \end{cases}$$

$\bowtie$

4.3.7 Corollary

If the type equivalence problem is decidable, then the dpda equivalence problem is decidable.

Proof

Let L, L' $\subseteq \Sigma^*$ be the deterministic languages, where $\Sigma = \{\emptyset, \dots, n-1\}$. By 4.3.2, we can construct dpda's P and P' such that $L(P) = Ln \subseteq (\Sigma \sqcup \{n\})^*$ and $L(P') = L'n \subseteq (\Sigma \sqcup \{n\})^*$. Let D and D' be definitions defining trees $t_\emptyset$ and $t'_\emptyset$, constructed according to 4.3.4. Then

D is equivalent to D'

iff $t_\emptyset = t'_\emptyset$ (by 3.3.10)

iff $L(P) = L(P')$ (by 4.3.6)

iff $L = L'$

$\bowtie$

$Q_w = \{q_0, q_1\}$

$Q_s = \{q_2, q_3, q_4\}$

$Q_e = \{p_0, p_1\}$

$F = \{q_5\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{a, b\}$

$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[r, "\text{push } a"] & q_2 \arrow[r, "1"] \arrow[d, bend left, "0"] & q_3 \arrow[r, "1"] \arrow[d, bend left, "0"] & p_1 \arrow[r, "\text{pop } a"] \arrow[d, "\text{pop } b"] & q_5 \\
& & p_1 \arrow[u, bend left, "\text{push } b"] & p_0 \arrow[u, bend left, "\text{pop } b"] \arrow[r, "\text{pop } a"] & q_4 \arrow[loop right, "0{,}1"] &
\end{tikzcd}$$

Figure 3a

A pushdown automaton in normal form

f	0	1
q_2	q_1	q_3
q_3	p_0	p_1
q_4	q_4	q_4

g	p_0	p_1
q_0	q_5	q_5
q_1	q_3	q_4

h	
q_0	q_2
q_1	q_2

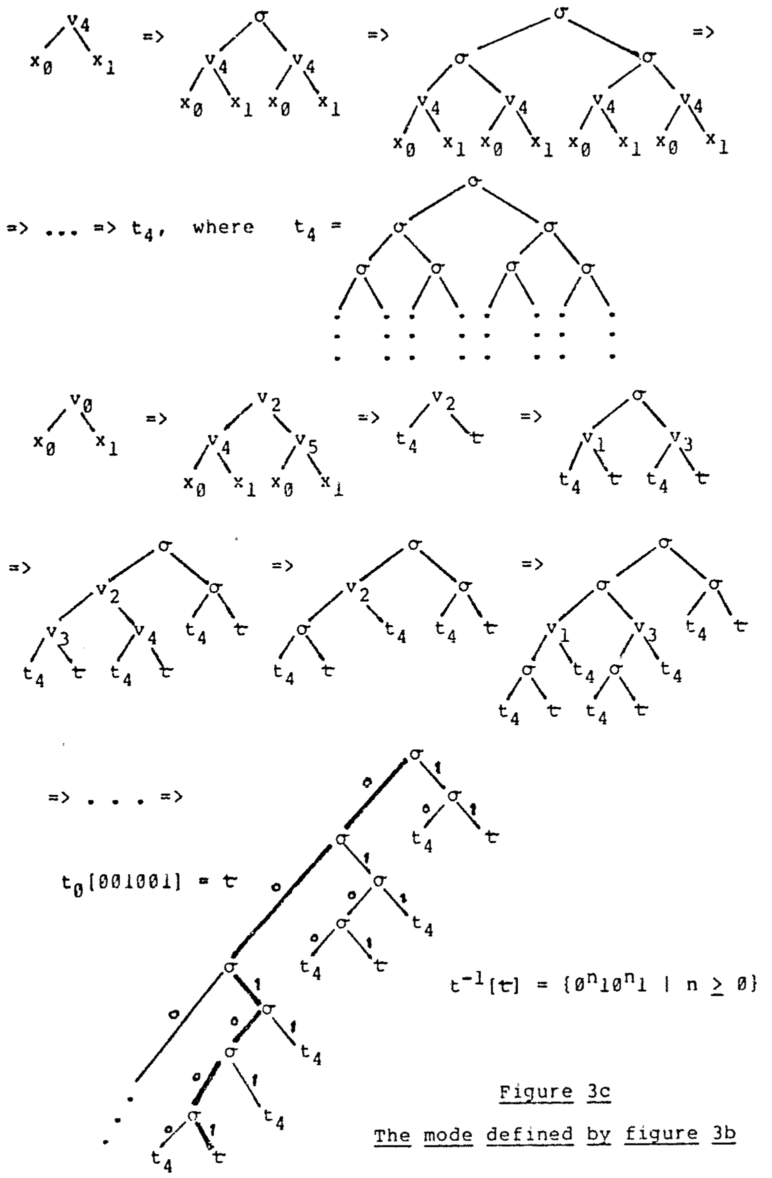
k∘f	0	1
q_2	$v_1(x_0,x_1)$	$v_3(x_0,x_1)$
q_3	x_0	x_1
q_4	$v_4(x_0,x_1)$	$v_4(x_0,x_1)$

k∘g	p_0	p_1
q_0	$v_4(x_0,x_1)$	$v_5(x_0,x_1)$
q_1	$v_3(x_0,x_1)$	$v_4(x_0,x_1)$

$$\begin{aligned}
v_0(x_0,x_1) &= v_2(v_4(x_0,x_1),v_5(x_0,x_1)) \\
v_1(x_0,x_1) &= v_2(v_3(x_0,x_1),v_4(x_0,x_1)) \\
v_2(x_0,x_1) &= \sigma(v_1(x_0,x_1),v_3(x_0,x_1)) \\
v_3(x_0,x_1) &= \sigma(x_0,x_1) \\
v_4(x_0,x_1) &= \sigma(v_4(x_0,x_1),v_4(x_0,x_1)) \\
v_5(x_0,x_1) &= \tau
\end{aligned}$$

Figure 3b

The type definition derived from figure 3a



$t_0[001001] = \tau$

$t^{-1}[\tau] = \{0^n10^n1 \mid n \geq 0\}$

Figure 3c

The mode defined by figure 3b

And here Alice began to get rather sleepy, and went on saying to herself, in a dreamy sort of way, "Do cats eat bats? Do cats eat bats?" and sometimes, "Do bats eat cats?" for, you see, as she couldn't answer either question, it didn't matter which way she put it.

Lewis Carroll, Alice in Wonderland

5. SUMMARY AND CONCLUSIONS

We have presented a modest extension of the usual type definition facilities found in current programming languages. We have given a precise mathematical semantics to this extended facility using a so-called "lattice theoretic" or "Scott-like" model. Using this model, we were able to show an intimate connection between definable types and deterministic context-free languages. In particular, we have shown that the equivalence problem for definable types is of equal difficulty as the equivalence problem for deterministic languages. Since the latter problem remains open in the face of several years of concentrated efforts of researchers to close it, we must abandon attempts to decide equivalence of types, restrict the range of types that may be defined, or use a more trivial notion of type equivalence.

What are the practical implications of this result? As we mentioned in section 1.1, we can avoid the problem entirely by refusing to consider separately declared types equivalent, or to consider them equivalent only if the declarations are of "essentially the same form" in some sense. On the other hand, it may be that even the modest extension we presented here is unnecessarily strong. For example, definitions (1.1), (1.2), and (1.3), which were presented as motivation for modals, define regular types and thus could be defined without the use of parameters. In this case, the parameters serve merely as a convenience, allowing various aspects of the type <u>intlist</u> to be specified separately. Thus one could require that all types defined be regular (in the sense that $t^{-1}[\sigma]$ is a regular set for all σ). In view of the fact that it is decidable whether a given deterministic context-free set is regular [12, p. 230], it should not be hard to show that there is an algorithm to enforce this restriction. It remains to be seen whether non-regular types are of any practical use.

In fact, a somewhat stronger restriction, which is <u>sufficient</u> to ensure that all types declared are regular, is to change the syntax so that actual parameters to modal variables must be parameter symbols or modal constants of rank Ø. This still allows definitions such as (1.1) but would be easier to enforce than the restriction of the previous paragraph. More experience with modals is necessary before we can state whether such restrictions prohibit any truly useful type definitions.

Chapter II
PARSING INFINITE GRAMMARS

1. INTRODUCTION

Syntax is the best understood and most extensively studied area of programming language theory. Context-free grammars have proved to be a tool capable of specifying most aspects of the syntax of a programming language in a manner which is at the same time understandable, rigorous, and amenable to computerized checking. Much work has been devoted to enhancement of the latter aspect of context-free grammars: starting with the pioneering work of Floyd[8] and Irons[14] and continuing to the present day, dozens of authors have published algorithms which mechanically transform the context-free specification of a language to a parsing program which checks alleged programs in the language for conformity to the specifications. Nearly all these parsers also gather data about the structure imposed on valid programs by the grammar. This structural aspect of grammar is used in describing the semantics of a language; structural data gathered by the parser may be used by a compiler. Many grammar analysis algorithms can certify a class of grammars as being unambiguous--the grammar imposes one and only one structure on any valid program. Over the years, algorithms have improved in terms of the range of grammars admissible, the size and speed of the parsers produced, and the facility

with which the information gathered by the parser may be used by the rest of the compiler.

Another direction of research has concerned the extension of syntax description methods beyond the limits of context-free grammars. There are two principal reasons for this. First, context-free grammars are not capable of specifying <u>all</u> aspects of the syntax of most programming languages. This was first rigorously demonstrated by Floyd[7]. Second, a more powerful syntax description technique allows more of the work of the compiler to be placed on the shoulders of the syntax analyzer. Since syntax analysis is the best understood phase of language implementation, it makes sense to move as much of the task as possible to the syntax analyzer.

An example will serve to illustrate both of these benefits. A typical grammar rule may take the form

statement $\rightarrow$ variable ':=' expression (1)

If the language in question has both integer and character string data types, and the language designer would like to prohibit the assignment of a value of one type to a variable of another type, he/she can add a note in English stating such a restriction, or use two rules:

statement $\rightarrow$ integer-variable, ':=',
integer-expression (2)

statement $\rightarrow$ string-variable, ':=', string-expression

The latter technique has the advantage that the specification is more precise and has less chance of being misinterpreted, and it also means that any parse according to this grammar will contain information about type, which can be used by other phases of a compiler. On the other hand, if there is a large number of types in the language, a large number of rules is required. If it were desired to state some relationship among the types of the components of a statement (other than simple equality), the number of rules might increase quadratically or worse with the number of types, and if there is no a priori bound on the number of types (as, for example, in a language which allows definitions of new types), then no finite set of rules will suffice. A solution is to give one parameterized rule, such as

```
    statement → TYPE variable, ':=', TYPE expression (3)
```

together with a specification of the set of permissible substitutions for TYPE. This shorthand notation can also be profitably used to abbreviate parts of a grammar which are normally written out at length. For example, the syntax of expressions could be succinctly specified by:

```
   PREC-expression → PREC-expression, PREC-operator,
                          PREC+1-expression
                        | PREC+1-expression
       6-expression → variable
                        | '(', 1-expression, ')'         (4)
         1-operator → '|'
         2-operator → '&'
         3-operator → '=' | '≠' | '<' | '≤' | '>' | '≥'
         4-operator → '+' | '-'
         5-operator → '*' | '/'
```

together with a specification that PREC is some positive in-

teger.

Two important versions of this technique are the two-level or van Wijngaarden grammars used to specify the syntax of Algol 68 [30] and attribute grammars which have been presented in many forms, but are probably best known as presented by Knuth [16].

In the Algol 68 terminology, rules like those in (3) and (4) are called "hyper-rules" and the parameters, such as TYPE or PREC are called "metanotions". The set of possible replacements for a metanotion is always a set of strings specified by a context-free grammar. When all occurrences of metanotions are replaced by strings ("protonotions") in a given hyper-rule--occurrences of the same metanotion being replaced by the same protonotion--the resulting rule is called a "strict" rule. The set of all such strict rules comprises an (infinite) context-free grammar. The restriction that metanotions always have strings as values is not essential, since encodings can always be used. For example, the first rule of (4) could be stated

```
   PREC-expression → PREC-expression, PREC-operator,
                          PREC-plus-one-expression
```

where PREC is defined by meta-rules

```
              PREC → one | PREC-plus-one
```

The reader is referred to [30] for many ingenious examples of the use of two-level grammars.

The structure of chapter II of this dissertation is as follows: In the remainder of this section, we present a more specific definition of infinite context-free grammars, and discuss various formalisms which may be viewed as infinite grammars. In section 2, we give an overview of LR parsing to establish notation and terminology and to highlight the special problems for LR parsing created by dropping finiteness assumptions. Section 3 is the core of chapter II. In this section we define a slight modification of Aho's[1] indexed grammars, propose two possible definitions of "LR(k) indexed grammar", and derive results concerning them. The first definition implies that the language first be parsed ignoring the "attributes" or "metanotions", and the attribute values be filled in as each reduction is performed. This is analogous to many versions of "attribute grammars" (c.f. [4] and [19]). A second, more interesting definition allows attributes to interact with the context-free "skeleton" in arbitrary manners. We show how such grammars may be used to perform a left to right deterministic bottom-up parse in a manner entirely analogous to the "traditional" LR methods.

The algorithms of section 3 require techniques for manipulating certain kinds of relations on the set of strings over a finite alphabet. All our results concerning such relations have been collected in section 4. Section 5 contains a summary and some directions for future research.

1.1 Infinite Grammars

A (generalized) context-free grammar is a 4-tuple $G = (N, \Sigma, P, S)$, where N and Σ are (not necessarily finite) sets, P is a subset of $N \times (N \cup \Sigma)^*$, and $S \in N$. Note that

$(N \cup \Sigma)^*$ still means the set of <u>finite</u> strings of symbols from N. It is well known that the cardinality of Σ^* is the same as the cardinality of Σ, if Σ is infinite. In all cases of interest in this dissertation, all sets will be countable. The notions of direct derivation ($\Rightarrow$) and derivation ($\overset{*}{\Rightarrow}$) are precisely the same as in the finite case (a derivation still has a finite number of steps). Nonetheless, the class of languages generated is much larger. In fact, if $L \subseteq \Sigma^*$ is <u>any set whatsoever</u>, the grammar $G = (\{A\}, \Sigma, \{A \rightarrow x \mid x \in L\}, A)$ generates L. Of course, if L is not recursively enumerable, then neither is any grammar for L, so we are interested in those grammars which have a finite representation, just as (ordinary) grammars are finite representations for the sets they generate.

A grammar does more than define a set; it also defines a structure on that set. Thus, for any grammar, no matter how specified, it makes sense to ask whether it is ambiguous. The usual definition of ambiguity applies without change to the generalized case. The important classes LR(k) and SLR(k) of unambiguous grammars also generalize without change. Section 2 repeats these definitions with annotations indicating the significance of the fact that G may be infinite. While the LR(k) and SLR(k) definitions generalize directly, many of their consequences do not. For example, every finite LR(k) grammar can be parsed bottom up by a deterministic push-down automaton. This certainly does not hold in the infinite case. More importantly, it is decidable, for fixed k, whether a given finite grammar is LR(k).

If the grammar is infinite, then the existence of a decision procedure depends, of course, on the manner in which the grammar is presented. One should at least suppose, for example, that the set of productions is recursive (that is, that it is decidable whether a given object is a production of G).

1.2 Formalisms for Infinite Grammars

One way of presenting an infinite grammar is to view the set of productions as a language and give some sort of grammar or grammar-like mechanism which generates it. This technique is best known from its application to the definition of Algol 68 [30]. Other methods of specifying languages, however, may also be viewed as infinite grammars. For example, the attribute grammars of Knuth [16] are specified by giving a context-free grammar in which sequences of attributes are attached to various grammar symbols. The values of the attributes are constrained by giving semantic functions relating the attribute values of symbols appearing in a given production. The idea is to parse an input string according to the underlying context-free grammar. The semantic functions then give local conditions specifying attribute values at a given node of the parse tree in terms of attribute values at its neighbors. If the grammar passes a check against circularity, then attribute values can be filled in in exactly one way. These values then help define the semantics of the input string. In addition, one attribute value might be error, so that the attributes can be

used to give more stringent conditions for well-formedness than can be formalized by a context-free grammar alone. Bochmann [4] gives conditions under which the assignment of attribute values may be carried on simultaneously with the parsing of the input (either top-down or bottom-up) by the context-free grammar. However, in each case, he assumes that the underlying context-free grammar (with attributes deleted) can be parsed deterministically; the attributes are not used to help discover the structure of input string.

An attribute grammar G may be regarded as a finite specification of an infinite grammar $\overline{G}$ as follows: the symbols of $\overline{G}$ consist of symbols of G together with particular values of their attributes. The productions of $\overline{G}$ are these instantiations of the productions of G in which the attribute values supplied "match up" according to the semantic functions. For example, suppose G has a symbol X with two attributes, one taking on values in the set A and the other taking on values in B, and a symbol Y with one attribute having values in A. Suppose G has a production

$$Y_x \rightarrow Y_y X_{wz} \qquad x = f(w,z,y),\ w = g(x,z).$$

Then among the symbols of $\overline{G}$ are $\{\langle Y,a\rangle \mid a \in A\}$ and $\{\langle X,a,b\rangle \mid a \in A \text{ and } b \in B\}$ and among the productions of $\overline{G}$ are

$$\{\langle Y,a\rangle \rightarrow \langle Y,b\rangle\langle X,d,c\rangle \mid a = f(d,c,b) \text{ and } d = g(a,c)\}$$

Viewed in this light, the attribute functions f and g are just a convenient way of defining relations among the attributes appearing in the immediate vicinity of nodes of a parse tree, or what amounts to the same thing, a relation among the attributes in productions of $\overline{G}$. Knuth classifies attribute domains as <u>inherited</u> or <u>synthesized</u>: the value of a synthesized attribute at a given node must depend in a functional manner on other attributes at that node and on attributes of its sons. Similarly, an inherited attribute of a node depends on attributes at its father and brothers. Stated in terms of infinite grammars, the set of productions of $\overline{G}$ is a union of sets of the form

$$\{X^{\emptyset}_{a_{\emptyset\emptyset}a_{\emptyset 1}\cdots} \rightarrow X^{1}_{a_{1\emptyset}a_{11}\cdots} X^{2}_{a_{2\emptyset}a_{21}\cdots} \cdots X^{j}_{a_{j\emptyset}a_{j1}\cdots} \mid \langle a_{\emptyset\emptyset}, a_{\emptyset 1}, \ldots a_{1\emptyset}, a_{11}, \ldots, a_{j\emptyset}, a_{j1}, \ldots \rangle \in R\}$$

where R must be a conjunction of equations of the form $a_{ij} = f(a_{i_1 j_1}, \ldots, a_{i_n j_n})$. Moreover, the set of a_{ij} which may appear on the left side of these equations is further restricted as follows: For each symbol X, the set of subscript positions is partitioned into "inherited" and "synthesized" positions. a_{ij} may appear on the left side of an equation if and only if $i = \emptyset$ and position j of $X^{\emptyset}$ is synthesized, or $i > \emptyset$ and position j of X^{i} is inherited. The reason for these restrictions on the form of R is that they make possible an algorithm to determine whether any parse tree generated by the underlying grammar can have attributes filled in in more than one way which is consistent with all the restrictions R (see [16]). If two assignments of attributes to a tree can never occur, then $\overline{G}$ is unambiguous provided G is. In other words, underlying the inherited/synthesized distinction is the assumption that an input will first be parsed ignoring attributes, and the attributes will be filled in later: some standard technique such as LR(k) may be used to insure that the former step prevents ambiguities;

Knuth's algorithm may be used to insure that no ambiguities are introduced in the latter step. Notice that the restrictions on the R's above make no assumption on the nature of the functions f used, other than that they are indeed functions (that is, single-valued).

Similar remarks apply to the attributed translation grammars of Lewis, Rosenkranz, and Stearns [19], and to earlier similar formalisms such as the property grammars of Stearns and Lewis [27] which may be viewed as special cases of attribute grammars.

In the framework of infinite grammars, the <u>indexed grammars</u> of Aho[1] may be viewed in some senses as an extension and in others as a restriction of the attribute grammars of Knuth. The extension comes from dropping the distinction between synthesized and inherited attributes. On the other hand, each non-terminal symbol has exactly one attribute, the domain of each attribute is the set F* for some finite F called the set of "flags", terminal symbols have no attributes, and the "semantic functions" are all of the form $f(x) = ax$, for some $a \in F$.

The purpose of the foregoing is to show that whereas the following development applies only to indexed grammars (with a mild generalization), it may be viewed as a test case of a technique with wider applicability.

2. LR(k) GRAMMARS

To emphasize the problems which arise in analyzing infinite grammars, and to establish notation and terminology, we present here a summary of the theory of LR parsing paying careful attention to those parts which are affected by assumptions of finiteness. For simplicity, and for the benefit of later sections of this chapter, we will largely limit the discussion to the most practically useful subclass of LR grammars, the SLR(1) grammars, and the related class of LR(0) grammars.

2.1 Definition

A grammar

$$G = (N, \Sigma, P, S)$$

consists of two disjoint sets N and Σ, an element $S \in N$, and a subset $P \subseteq N \times (N \sqcup \Sigma)^*$. (In the finite [i.e. usual] case, N, Σ, and P are all assumed to be finite. If A is any set, not necessarily finite, A^* denotes the set of finite strings of members of A, including the empty string e.) Let $V = N \sqcup \Sigma$. We write $A \rightarrow \alpha$ for $A \in N$ and $\alpha \in V^*$ rather than $\langle A, \alpha \rangle \in P$. If $\alpha, \beta \in V^*$, we write $\alpha \Rightarrow \beta$ if $\exists\, \gamma, \delta, \eta \in V^*$, $A \in N$, such that $\alpha = \gamma A \delta$, $\beta = \gamma \eta \delta$, and $A \rightarrow \eta$. $\overset{*}{\Rightarrow}$ is the transitive and reflexive closure of the relation $\Rightarrow$. (That is, $\overset{*}{\Rightarrow}$

is the smallest transitive and reflexive relation on V^* which contains $\Rightarrow$ (as a subset). Equivalently, $\alpha \overset{*}{\Rightarrow} \beta$ if and only if $\exists$ a finite sequence $\alpha_0, \ldots, \alpha_n$ $(n \geq 0)$ such that $\alpha = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n = \beta$). $\alpha \in V^*$ is a sentential form if $S \overset{*}{\Rightarrow} \alpha$. α is a sentence if in addition, $\alpha \in \Sigma^*$. The language generated by G is $L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$. A derivation of length n in G is a sequence $S \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n$. (By the above remarks, every sentence or sentential form has a derivation). The derivation is rightmost if for each i, $\exists\, \beta, \gamma, w, A$ such that

$$\alpha_i = \beta A w, \ \alpha_{i+1} = \beta \gamma w, \text{ and } w \in \Sigma^*.$$

We use the notations $\underset{r}{\Rightarrow}$ and $\underset{r}{\overset{*}{\Rightarrow}}$ for rightmost derivations. given a grammar $G = (N, \Sigma, P, S)$, the augmented grammar is $G' = (N \sqcup \{S'\}, \Sigma \sqcup \{\langle, \rangle\}, P \sqcup \{S' \rightarrow \langle S \rangle\}, S')$, where S', $\langle$, $\rangle$ are new symbols not in V. (We have deviated slightly from Aho and Ullman [3] by introducing end markers, for reasons of notational convenience). From now on we will assume, without explicit mention, that all grammars have been so augmented. G is LR(k) if the conditions

(1) $S \underset{r}{\overset{*}{\Rightarrow}} \alpha A w \underset{r}{\Rightarrow} \alpha \beta w$,

(2) $S \underset{r}{\overset{*}{\Rightarrow}} g B x \underset{r}{\Rightarrow} \gamma \delta x = \alpha \beta y$, and

(3) $\mathrm{FIRST}_k(w) = \mathrm{FIRST}_k(y)$

imply that $\alpha A y = \gamma B x$--i.e., $\alpha = \gamma$, $A = B$, and $x = y$. ($\mathrm{FIRST}_k(w)$ = if $k > |w|$ then w else the first k symbols of w fi.) In the case $k = 0$, (3) is, of course, vacuous. $\gamma \in V^*$ is a viable prefix of G if $\exists$ a derivation $S \underset{r}{\overset{*}{\Rightarrow}} \alpha A w \underset{r}{\Rightarrow} \alpha \beta w$ such that γ is a prefix of $\alpha\beta$. An LR(0) item for G is a pair $\langle p,j\rangle$ where $p \in P$ and $0 \le j \le$ length of the right hand side of p. We write $[A \to \alpha\bullet\beta]$ instead of $\langle p,j\rangle$ if $p = A \to \alpha\beta$ and $|\alpha| = j$. An item of the form $[A \to \bullet\alpha]$ is called an <u>initial</u> <u>item</u>; $[A \to \alpha\bullet]$ is a <u>final</u> <u>item</u>.

If $\gamma = \alpha\beta_1$ is a viable prefix for G and $I = [A \to \beta_1\bullet\beta_2]$ is an item, then I is <u>valid</u> <u>for</u> <u>γ</u> if there is a derivation $S \overset{*}{\underset{r}{\Rightarrow}} \alpha A w \underset{r}{\Rightarrow} \alpha\beta_1\beta_2 w$. $V(\gamma) = \{I \mid I$ is a valid item for $\gamma\}$. The <u>e-free</u> <u>first</u> function is defined by EFF(X) = <u>if</u> $X \in \Sigma$ <u>then</u> $\{X\}$ <u>else</u> $\{a \in \Sigma \mid A \overset{*}{\underset{r}{\Rightarrow}} \beta \underset{r}{\Rightarrow} ax$ for some $x \in \Sigma^*$, where $\beta \ne Aax$ for any $A \in N\}$. FOLLOW(A) = $\{a \in \Sigma \mid S \overset{*}{\Rightarrow} \alpha A a \beta\}$. (In the finite case, EFF(X) and FOLLOW(A) are both subsets of Σ, hence finite. In general, this may not hold). Let q be any set of items (also called, for reasons to become clear later, a <u>state</u>). q is <u>LR(0)</u> <u>inadequate</u> if q contains at least two items, at least one of which is final. q is <u>SLR(1)</u> <u>inadequate</u> (or simply <u>inadequate</u>) if

(1) q contains two items $[A \to \alpha\bullet X\beta]$ and $[B \to \gamma\bullet]$ such that FOLLOW(B) $\sqcap$ EFF(X) $\ne \emptyset$, or

(2) q contains two items $[A \to \alpha\bullet]$ and $[B \to \gamma\bullet]$ such that FOLLOW(A) $\sqcap$ FOLLOW(B) $\ne \emptyset$. Case (1) is called a <u>shift-reduce</u> <u>conflict</u>. Case (2) is called a <u>reduce-reduce</u> <u>conflict</u>.

G is <u>SLR(1)</u> if and only if no $V(\gamma)$ is inadequate for any viable prefix γ. Let q be any set of items. Then a <u>closure</u> <u>nonterminal</u> of q is any $A \in N$ such that there is an item $[B \to \alpha\bullet A\gamma] \in q$. ClosureNT(q) denotes the set of closure nonterminals of q. The <u>closure</u> of q is the set Closure(q) $= q \sqcup \{[A \to \bullet\alpha] \mid A \to \alpha$ and $B \overset{*}{\Rightarrow} A\beta$ for some $\beta \in V^*$ and some $B \in$ ClosureNT(q)$\}$. The <u>basis</u> of q is the set Basis(q) of non-initial items of q.

$\text{Scan}(q,X) = \{[A \to \alpha X\bullet\beta] \mid [A \to \alpha\bullet X\beta] \in q\}.$

$\text{Goto}(q,X) = \text{Closure}(\text{Scan}(q,X)).$

$\text{Goto}(q,\alpha X) = \text{Goto}(\text{Goto}(q,\alpha),X).$ ⌧

2.2 <u>Proposition</u>

$V(¢) = \text{Closure}(\{[S' \to ¢\bullet S¢]\}).$

$V(\gamma X) = \text{Goto}(V(\gamma),X).$ ⌧

In the finite case, the set of all possible items is finite, hence only finitely many distinct sets may occur as $V(\gamma)$. By the above proposition, these may besystematically generated. Hence an algorithm for testing for the SLR(1) property is as follows:

2.3 <u>Algorithm</u> (<u>Test</u> <u>for</u> <u>SLR(1)</u> <u>Property</u>)

(1) Let $\mathbf{S} = \{V(\gamma) \mid \gamma$ is a viable prefix of G and $|\gamma| \ge 1\}$. Generate $\mathbf{S}$ by letting $q_0 = V(¢) = \text{Closure}(\{[S \to ¢\bullet S¢]\})$, initially adding q_0 to $\mathbf{S}$, and whenever q is added to $\mathbf{S}$, adding Goto(q,X) to $\mathbf{S}$ for all $X \in V$, until no new sets q are added.

(2) Check each $q \in \mathbf{S}$ for inadequacy. ⌧

Of course, step 2 can be integrated with step 1, checking each q as it is added and halting the procedure as soon as an inadequate q is found. Careful analysis of this algorithm shows that the only operations which are not straightforward are calculation of $\{B \in N \mid A \Rightarrow B\delta\}$, EFF(A), and

FOLLOW(A) for each $A \in N$. These may be computed as follows:

2.4 Algorithm (Calculation of Relations)

1. Determine, for each $A \in V$, whether $A \overset{*}{\Rightarrow} e$.
2. Define the relations Left, Adj, Right, and EFLeft b

$$\begin{aligned}
\text{Left} &= \{\langle Y,X\rangle \mid X \rightarrow \alpha Y\beta \text{ and } \alpha \overset{*}{\Rightarrow} e\}\\
\text{Right} &= \{\langle X,Y\rangle \mid X \rightarrow \alpha Y\beta \text{ and } \beta \overset{*}{\Rightarrow} e\}\\
\text{Adj} &= \{\langle Y,X\rangle \mid Z \rightarrow \alpha X\beta Y\gamma \text{ and } \beta \overset{*}{\Rightarrow} e\}\\
\text{EFLeft} &= \{\langle Y,X\rangle \mid X \rightarrow Y\beta\}
\end{aligned}$$

3. $A \overset{*}{\Rightarrow} B\delta$ for some δ iff A Left* B, EFF(A) = $\{a \in \mathbf{S} \mid$ a EFLeft* A$\}$, and FOLLOW(A) = $\{a \in \mathbf{S} \mid$ a Left* o Adj o Right* A$\}$.

Here o denotes Pierce product of relations and * is transitive and reflexive closure of a relation. These functions can be computed from a Boolean matrix representation of the relations by well-known techniques (see, for example, Warshall [29]). If G contains no e-productions (rules of the form $A \rightarrow e$), then definitions are greatly simplified.

$$\begin{aligned}
\text{Right} &= \{\langle X,Y\rangle \mid X \rightarrow \alpha Y\}\\
\text{Adj} &= \{\langle Y,X\rangle \mid Z \rightarrow \alpha XY\beta\}\\
\text{Left} &= \text{EFLeft} = \{\langle Y,X\rangle \mid X \rightarrow Y\alpha\}
\end{aligned}$$

Condition (1) of the definitions of inadequate state becomes

(1') q contains two items $[A \rightarrow \alpha\bullet]$ and $[B \rightarrow \alpha\bullet a\beta]$ such that $a \in$ FOLLOW(A)

provided q is closed. (If $[B \rightarrow \alpha\bullet X\beta] \in q$ and $a \in$ EFF(X), then $[C \rightarrow \bullet a\gamma]$ is another item in q). ⊠

Having established that G is SLR(1), a parsing table for G may be constructed as follows:

2.5 Algorithm (Calculation of Parsing Actions)

For each $q \in \mathbf{S}$ and $X \in V$, Action(q,X) is one of:

Shiftto q' for some $q' \in \mathbf{S}$,

Reduceby p for some $p \in P$, or

Error.

specifically:

$$\text{Action}(q,a) = \begin{cases} \text{Shiftto Goto}(q,X) \text{ if some} \\ \qquad [A \rightarrow \alpha\bullet X\beta] \in q \text{ and } a \in \text{EFF}(X), \\ \text{Reduceby } A \rightarrow \alpha \text{ if } [A \rightarrow \alpha\bullet] \in q \\ \qquad \text{and } a \in \text{FOLLOW}(A) \\ \text{Error otherwise.} \end{cases}$$ ⊠

Since G is SLR(1), at most one of the above applies.

The parsing algorithm uses this table and a stack to parse an input $\{a_1 \ldots a_n\}$ as follows:

2.6 Algorithm (LR Parsing)

```
(1)     Set i = 1, Current = V(⊀), stack empty.
(2)
        (a)     If Action(Current,$a_i$) = Shiftto q then
                   increment i
                   push Current onto the stack
                   set Current = q
        (b)     If Action(Current,$a_i$) = Reduce $A \rightarrow \alpha$, then
                   announce that $A \rightarrow \alpha$ has been recognized
                   If $|\alpha| > 0$, then pop $|\alpha| - 1$ states off
```

the stack.

If $\alpha = e$, then push Current onto the stack.

set Current = Goto(q,A), where q is the top item on the stack

(c) If Action(Current,a_i) = Error, then halt and report error.

(3) Repeat step 2 until S' → {S} has been announced.

⊠

Of course, many details of implementation have been glossed over here. In particular, in an actual compiler, the act of "announcing" a production will usually involve some sort of compiling action. If the sequence of productions "announced" is $p_1, \ldots, p_k$, then $p_k, \ldots, p_1$ gives the unique rightmost derivation of $\{a_1 \ldots a_k\}$ from S' in G.

3. INDEXED GRAMMARS

3.1 Introduction

If a grammar is infinite, the definitions of LR(k) and SLR(k) generalize without change, but the "algorithms" described in section 2 are no longer obviously effective, because many of the sets which must be manipulated may be infinite. In this case, we must show that the sets of interest may be finitely represented and that the operations required may be effectively carried out by manipulating the representations. The extent to which this is possible depends, of course, on the manner of representing the grammar itself.

We have chosen the formalism of indexed grammars, introduced by Aho [1], since we feel that they are sufficiently complex to introduce many of the complications involved in any formalism of infinite grammars, yet restricted enough to be mathematically tractable. Indexed grammars are not as powerful as two-level grammars or attribute grammars: the indexed grammars as defined in [1] generate a proper subclass of the context-sensitive languages; two-level grammars are capable of generating all recursively enumerable sets (Sintzoff [25]); attribute grammars can define arbitrary sets (Knuth [16]). Nonetheless, the kinds of restrictions normally required for definition of computer languages, such as the examples in section 1, can be formalized within the framework of indexed grammars.

Roughly stated, indexed grammmars differ from ordinary context-free languages in that non-terminal symbols may carry stacks of "flags", symbols from some finite set disjoint from the terminal and non-terminal symbols of the grammar. There are two kinds of production. An "ordinary" production looks like a production in a context-free grammar except that each occurrence of a non-terminal on the right hand side may be followed by a sequence of flags. For example, if A, B, and C are non-terminals, c is a terminal, and f and g are flags, a production of this kind might be

$$A \rightarrow Bfg\ c\ Ag$$

When such a production is applied to a sentential form to expand an occurrence of A, the sequence of flags following that occurrence is concatenated onto the end of the sequence of flags following each non-terminal on the right hand side of the production. For example, using the above production,

$$c\ \underbrace{Afg}\ Ag \Rightarrow c\ \underbrace{Bfgfg\ c\ Agfg}\ Ag$$

The other kind of production, which may be denoted $Af \rightarrow \ldots$ is applied similarly, but it must only be applied to an occurrence of A whose string of flags begins with f; this initial f is deleted before the sequence is concatenated onto the sequences of flags on the right hand side. For example, $Af \rightarrow Bfg\ c\ Ag$ may be applied to the first A in c Afg Ag to yield $c\ \underbrace{Bfgg\ c\ Agg}\ Ag$, but it cannot be applied to the

second A. The following example is from Aho [1].

3.1.1 Example

Let A, B, S be non-terminals, a, b, and c be terminals, and f and g be flags. Let the productions be

$$S \rightarrow a\ Af\ c$$
$$A \rightarrow a\ Ag\ c \quad | \quad B$$
$$Bf \rightarrow b$$
$$Bg \rightarrow b\ B$$

and let S be the start symbol. A typical derivation is

$$S \Rightarrow a\ Af\ c \Rightarrow aa\ Agf\ cc \Rightarrow aaa\ Aggf\ ccc$$
$$\Rightarrow aaa\ Bggf\ ccc \Rightarrow aaab\ Bgf\ ccc \Rightarrow aaabb\ Bf\ ccc$$
$$\Rightarrow aaabbbccc$$

In fact, it is not hard to see that the language generated is $\{a^n b^n c^n \mid n \geq 1\}$. $\boxtimes$

We call the sequence of flags following a non-terminal symbol a "stack" since it is manipulated in a last-in-first-out manner. (The "top" (most recent) end of the stack is at the left).

An indexed grammar may be viewed as an infinite grammar by considering each non-terminal together with its stack of flags to be a non-terminal of the infinite grammar, and each production, e.g., $Af \rightarrow Bfg\ c\ Ag$ to be an abbreviation for $\{Afx \rightarrow Bfgx\ c\ Agx \mid x$ is a sequence of flags$\}$. Readers should convince themselves that this convention implies precisely the rules for application of productions described above.

We have extended Aho's definition slightly, removing certain restrictions which we feel are not essential to the discussion. First, we drop the ban on flags following terminal symbols. The restriction that flags only follow nonterminals amounts to a requirement that the terminal vocabulary be finite. If strings of flags are to be considered "attributes", we think it is reasonable to assume that certain terminal symbols--that is, inputs to the parser--come endowed with attributes, perhaps supplied by the lexical scanner. Nonetheless, we enforce a partition of the vocabulary into symbols which take parameters and those which do not.

Second, we allow more than one flag on the left-hand side: a production of the form $Af_1f_2\ldots f_n \rightarrow \beta$ could always be replaced by

$$Af_1 \rightarrow A_1;\quad A_1f_2 \rightarrow A_2;\ \ldots;$$
$$A_{n-1}f_n \rightarrow A_n;\ A_n \rightarrow \beta$$

where $A_1,\ldots,A_n$ are new non-terminals.

Third, we allow the start symbol to have an arbitrary attribute. Finally, our notation is different from Aho's. However, the correspondence between his notation and ours should be clear. Since the difference between our definition and Aho's is mostly notation, we shall continue to use the term "indexed grammar".

3.1.2 Definition

An indexed grammar is a 5-tuple

$$G = (V_\emptyset, V_1, \Sigma, F, P, S)$$

where $V_\emptyset$, V_1 and F are disjoint finite sets, $\Sigma \subseteq V_\emptyset \sqcup V_1$ and

P, S, Σ are defined below.

Let

$$V = V_\emptyset \sqcup V_1$$
$$\overline{V}_1 = V_1 \times F^* \qquad \overline{V}_\emptyset = V_\emptyset$$
$$\overline{V} = V_\emptyset \sqcup \overline{V}_1$$

and similarly

$$\overline{\Sigma} = (\Sigma - V_1) \sqcup ((\Sigma \sqcap V_1) \times F^*)$$

$V_\emptyset$ is the set of zero-parameter symbols, V_1 is the set of one-parameter symbols, F is the set of flags. A string x $\in F^*$ is called a parameter. $\overline{V}$ is the set of parameterized symbols, S is a member of $\overline{V}$, P is a finite subset of $(\overline{V} - \overline{\Sigma}) \times \overline{V}^*$. Σ is called the set of terminal symbols, members of V − Σ are nonterminal symbols, S is the start symbol, and P is the set of productions. A member $\langle A,\alpha\rangle \in P$, where $A \in \overline{V} - \overline{\Sigma}$ and $\alpha \in \overline{V}^*$, is written $A \rightarrow \alpha$. V may be considered a subset of $\overline{V}$ by identifying $X \in V_1$ with $\langle X,e\rangle \in \overline{V}_1 = V_1 \times F^*$. If $X \in \overline{V}$, $w \in F$ then

$$X(w) \;=\; \begin{array}{l} X \text{ if } X \in V_\emptyset \\ \langle A,vw\rangle \text{ if } X = \langle A,v\rangle \in \overline{V}_1 \end{array}$$

If $\alpha = X_1\ldots X_n \in \overline{V}^*$, then If $p = A \rightarrow \alpha \in P$, then $p(w) = A(w) \rightarrow \alpha(w) \in \overline{V} \subseteq \overline{V}^*$. The (infinite context-free) grammar derived from G is

$$\overline{G} = (\overline{V} - \overline{\Sigma}, \overline{\Sigma}, \overline{P}, S)$$

where $\overline{P} = \{p(w) \mid p \in P,\ w \in F^*\}$. The language defined by G, L(G), is the language defined by $\overline{G}$ in the usual sense.

⊠

The reader may check that this definition corresponds directly to Aho's definition in the case in which $\Sigma = V_\emptyset$, $S \in V_1$, and $|w| < 1$ for all $\langle A,w\rangle \rightarrow \alpha$. The relationship between G and $\bar{G}$ may be seen from two points of view: First, one can regard $\bar{G}$ as an infinite grammar defining a language, and G as a finite representation of $\bar{G}$. Alternatively, one might view G as a (finite) context-free grammar together with some extra context-dependent information. The next section pursues the latter interpretation. We shall return to the former interpretation in section 3.3.

3.2 Skeletal LR(k) Grammars

3.2.1 Definition

The skeleton of $X \in \bar{V}$ is X if $X \in V_\emptyset$, and Y if $X = \langle Y,w\rangle \in V_1$. Similarly, the skeleton of a string or production is obtained by erasing all attributes. The (context-free) skeleton of G is the grammar

$$\tilde{G} = (\tilde{V} - \tilde{\Sigma}, \tilde{\Sigma}, \tilde{P}, \tilde{S})$$

where $\tilde{P}$ and $\tilde{S}$ are the skeletons of P and S, respectively.

A derivation $\Delta = (\alpha = \tilde{S} \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_k)$ in G is consistent with G if there exists an assignment of attributes to occurrences of one-attribute symbols in the α_i which makes Δ into a derivation in $\bar{G}$. In particular, this means that

(1) If $S \in \bar{V}_1$, then $\alpha_1 = S = \langle\tilde{S},w\rangle$.

(2) If the production $X_\emptyset \rightarrow X_1 \ldots X_n$ is used in Δ, and w_i is the attribute assigned to the indicated occurrence of X_i, then $X_\emptyset(w_\emptyset) \rightarrow X_1(w_1) \ldots X_n(w_n)$ is a production in $\bar{P}$. In other words, there is a production $Y_\emptyset \rightarrow Y_1 \ldots Y_n \in P$ and $w \in F^*$ such that $Y_i(w) = X_i(w_i)$ for $i = \emptyset, \ldots, n$. ⊠

As an example, if the production $A \rightarrow BcD$ is used in Δ, where $A, B, D \in V_1$, $c \in V_\emptyset$, then there must be a production $\langle A,t\rangle \rightarrow \langle B,u\rangle c\langle d,v\rangle$ in P and a string $w \in F^*$ such that the attributes assigned to the indicated occurrences of A, B, and D are, respectively, tw, uw, and vw. ⊠

3.2.2 Definition

G is "skeletal" LR(k) if and only if

(1) $\tilde{G}$ is LR(k) and

(2) For each $A \rightarrow \alpha$, $B \rightarrow \beta \in P$ and $v,w \in F^*$ such that A and B have the same skeleton, if $\alpha(v) = \beta(w)$ then $A(v) = B(w)$. ⊠

This definition says that G may be parsed bottom-up ignoring attributes, and that whenever a reduction is performed, the attribute of the left-hand side may be unambiguously determined from the attributes of the right-hand side.

3.2.3 Proposition

If G is skeletal LR(k) then $\bar{G}$ is unambiguous.

Proof

If $\bar{G}$ is ambiguous, then some string has two parse trees with respect to $\bar{G}$, t_1 and t_2. Since $\tilde{G}$ is LR(k), $\tilde{G}$ is unambiguous. Let $\tilde{t}_i$ be the result of erasing all attributes from t_i. Each $\tilde{t}_i$ is a parse tree with respect to $\tilde{G}$, hence

$\tilde{t}_1 = \tilde{t}_2$. It remains to be shown that the attribute at any node of t_1 is the same as the attribute at the corresponding node of t_2. We proceed by induction on the height of a node. Leaves are terminal symbols. Since, by hypothesis, t_1 and t_2 are parse trees of the same terminal string, corresponding leaves have equal attributes. Considr any interior node of t_1. Say it is labeled by $X \in \bar{V}$, and its sons "spell out" $\gamma \in \bar{V}^*$. Since t_1 is a parse tree with respect to $\bar{G}$, there is some production $A \rightarrow \alpha \in P$ and some $v \in F^*$ such that $X = A(v)$ and $\gamma = \alpha(v)$. Similarly, if the corresponding node of t_2 is labeled Y and has sons labeled δ, then there is some $B \rightarrow \beta \in P$ and $w \in F^*$ such that $Y = B(w)$ and $\delta = \beta(w)$. By induction hypothesis, $\gamma = \delta$--i.e., $\alpha(v) = \beta(w)$. Hence by property (2) of skeletal LR(k) grammars, $X = A(v) = B(w) = Y$. In other words, not only are the two considered nodes equal "skeletally", they also have the same attributes. ⊠

3.2.4 Proposition

It is effectively decidable whether a grammar is skeletal LR(k).

Proof

It is well known that property (1) is decidable. For property (2), it suffices to consider pairs of productions with the same skeleton. Let $X_\emptyset, \ldots, X_n \in V$ and let $A_\emptyset \rightarrow \alpha = A_1, \ldots A_n$ and $B_\emptyset \rightarrow \beta = B_1, \ldots, B_n$ be productions in P such that for each i, if $X_i \in V_\emptyset$ then $A_i = B_i = X_i$ and if $X_i \in V_1$, then $\exists$ $v_i, w_i \in F^*$ such that $A_i = \langle X_i, v_i \rangle$ and $B_i =$

$\langle X_i, w_i \rangle$. We claim that $\alpha(v) = \beta(w)$ if and only if

(i) $\exists$ u such that $v = uw$ and $\forall i$ such that $X_i \in V_1$, $v_i u = w_i$

or (ii) $\exists$ u such that $w = uv$ and $\forall i$ such that $X_i \in V_1$, $w_i u = v_i$.

If (i) holds, then $\forall i$, $A_i(v) = \langle X_i, v_i \rangle(uw)$

$$= \langle X_i, v_i uw \rangle = \langle X_i, v_i u \rangle(w) = \langle X_i, w_i \rangle(w) = B_i(w).$$

Hence $\alpha(v) = \beta(w)$. Similarly for (ii).

Conversely, suppose $\alpha(v) = \beta(w)$ but neither (i) nor (ii) holds. There are two cases:

(1) For some i, $X_i \in V_1$ and neither of v_i, w_i is a prefix of the other. Then clearly, $v_i v \neq w_i w$ so $A_i(v) \neq B_i(w)$ and $\alpha(v) \neq \beta(w)$.

(2) $\exists$ i, j such that X_i, $X_j \in V_1$ and $u \in F^*$ with

$v_i u = w$ and $v \neq uw$

or $v_i = w_i u$ and $w \neq vw$.

By symmetry, we need consider only the first case

$$A_i(v) = B_i(w)$$

whence $A_i(v) = \langle x_i, v_i \rangle(v) = \langle X_i, v_i v \rangle$

$$= B_i(w) = \langle X_i, w_i \rangle(w) = \langle X_i, w_i w \rangle = \langle X_i, v_i uw \rangle.$$

Therefore $v_i v = v_i uw$ so $v = uw$ which proves the claim.

Now, given a candidate for a violation of property (2) as above, we can easily find a u such that $v_i u = w$ $\forall i$ or $v_i = w_i u$ $\forall i$ or determine that no such u exists. If no such u exists, then no violation is present. If such a u exists, it is clearly unique. Suppose (WLG) that $\forall i$ such that $X_i \in V_1$, $v_i u = w_i$. If $A_\emptyset = B_\emptyset = X_\emptyset \in V_\emptyset$, then $A_\emptyset(v) = a_\emptyset = b_\emptyset = b_\emptyset(w)$ for <u>any</u> v, w and no violation is possible. Suppose $A_\emptyset$

$= \langle X_\emptyset, v_\emptyset\rangle$ and $B_\emptyset = \langle Y_\emptyset, w_\emptyset\rangle$. If $v_\emptyset u = w_\emptyset$, then $\alpha(v) = \beta(w)$

$$\begin{aligned} &\iff \quad v = uw \\ &\iff \quad A_\emptyset(v) = \langle x_\emptyset, v_\emptyset v\rangle = \langle X_\emptyset, v_\emptyset uw\rangle = \langle X_\emptyset, w_\emptyset w\rangle \\ &\qquad\quad = \langle B_\emptyset(w) \end{aligned}$$

and no violation occurs.

If $v_\emptyset u \neq w_\emptyset$ then $\alpha(u) = \beta(e)$, yet $A_\emptyset(v) = \langle X_\emptyset, v_\emptyset u\rangle \neq \langle X_\emptyset, w_\emptyset\rangle$ $= B_\emptyset(e)$. ⋈

In the above definitions and propositions, we never used the specific properties of LR(k) grammars. Similar definitions and results hold for SLR(k) grammars, LL(k) grammars, or any other recursive family of unambiguous grammars. What we have called "skeletal LR(k)" grammars correspond roughly to what Lewis _et al_ [19] call "S-attributed".

Unfortunately, our definition of "skeletal LR(k)" is too restrictive: we require that the context-free structure of a sentence be determined without any help from the attributes. The following section runs to the opposite extreme.

_3.3_ _Parsing Indexed Grammars_

For the remainder of section 3 we will return to the interpretation of an indexed grammar G as a finite specification of the infinite grammar $\overline{G}$ as in definition 3.1.2. We will show how to parse $\overline{G}$ by an algorithm analogous to algorithm 2.6. However, rather than pre-analyze the grammar for so-called SLR conflicts, we discover such conflicts as we parse. If no conflicts are discovered for a given input string, then our procedure gives the unique parse consistent with the grammar.

The algorithm works by dealing with finite representations of the infinite states of a "traditional" SLR(1) parser.

_3.3.1_ _Definition_

Let $G = (V_\emptyset, V_1, \Sigma, F, P, S)$ be an indexed grammar, and let $\overline{G} = (\overline{V}-\overline{\Sigma}, \overline{\Sigma}, \overline{P}, S)$ be the associated infinite context-free grammar as defined in 3.1.2. Recall from definition 2.1 that an (LR(0)) _item_ for $\overline{G}$ is a pair $I = \langle p,j\rangle$ such that $p \in \overline{P}$ and $0 \leq j \leq$ length of the right hand side of p. By 3.1.2, $p = p'(w)$ for some $p' \in P$ and some $w \in F^*$. Say that I is _simple_ if $w = e$. In other words, a simple item is a pair $\langle p,j\rangle$ where $p \in P$. Clearly, there are finitely many simple items. Let $I = \langle p,j\rangle$ be an item, $w \in F^*$. Then $I(w)$ denotes the item $\langle p(w),j\rangle$. If q is a set of items, then $q(w)$ denotes $\{I(w) \mid I \in q\}$. Vec(q) is a vector of subsets of F^*, one for each simple item $I = \langle p,j\rangle$, $p \in P$, such that

$$\text{Vec}(q)[I] = \{w \mid p(w) \in q\}.$$

⋈

_3.3.2_ _Note_

(i) P (as opposed to $\overline{P} = \{p(w) \mid p \in P\}$) is finite. Therefore, there are finitely many simple items, so Vec(q) has finitely many components. We prefer to index such vectors by items rather than integers, since we feel that the exposition is clearer. However, the reader who prefers the more traditional approach may assume that some fixed but arbitrary ordering $\{I_1,\ldots,I_n\}$ has been given to items, and the notation Vec(q)[I] is an abbreviation for $(\text{Vec}(q))_i$, where $I = I_i$.

(ii) Throughout this section we shall assume that G (and hence $\overline{G}$) is e-free. That is, there are no productions of the form $X \rightarrow e$. ⊠

One important result proved below is that all q which arise (that is, all sets of the form $q = V(\mathcal{Y})$, where $\mathcal{Y}$ is a viable prefix) have the property that Vec(q) is a vector of <u>regular</u> subsets of F*. Since Vec(q) completely describes q, this gives us a notation for representing states, viz., a vector of regular expressions.

<u>3.3.3</u> <u>Definition</u>

Let n be the number of simple items, and let m be the size of $V = V_\emptyset \sqcup V_1$. Define the n by m matrices of relations on F* <u>First</u>, <u>Last</u>, <u>Init</u>, <u>Fin</u>, <u>Before</u>, and <u>After</u> as follows:

$$\begin{aligned}
\text{First}[I,X] &= \{v \mid I = [\ldots \rightarrow \bullet X(v) \ldots]\}\\
\text{Last}[I,X] &= \{v \mid I = [\ldots \rightarrow \ldots X(v)\bullet]\}\\
\text{Init}[I,X] &= \{v \mid I = [X(v) \rightarrow \bullet \ldots]\}\\
\text{Fin}[I,X] &= \{v \mid I = [X(v) \rightarrow \ldots \bullet]\}\\
\text{Before}[I,X] &= \{v \mid I = [\ldots \rightarrow \ldots \bullet X(v) \ldots]\}\\
\text{After}[I,X] &= \{v \mid I = [\ldots \rightarrow \ldots X(v)\bullet \ldots]\}
\end{aligned}$$

(The elipsis [...] denotes some arbitrary member of $\overline{V}$ or $\overline{V}*$ as appropriate).

For each $a \in V_\emptyset$, let Scan_a be the n by n matrix defined by

$$\text{Scan}_a[J,I] = \begin{cases} \{e\} \text{ if } I=[Y \rightarrow \alpha\bullet a\beta] \text{ and } J = [Y \rightarrow \alpha a\bullet\beta] \\ \quad \text{for some } Y,\ \alpha, \text{ and } \beta \end{cases}$$

$$\begin{cases} \\ \emptyset \text{ otherwise} \end{cases}$$

For each $A \in V_1$, let

$$\text{Scan}_A[J,I] = \begin{cases} \{w \mid I = [Y \rightarrow \alpha\bullet\langle A,w\rangle\beta] \text{ and} \\ \quad J = [Y \rightarrow \alpha\langle A,w\rangle\bullet\beta]\} \end{cases}$$

⊠

<u>3.3.4</u> <u>Remark</u>

Definition 3.3.3 may be somewhat misleading because it obscures the fact that two essentially different cases are being combined through a notational "trick." If $X \in V_\emptyset$, then $X(v) = X = X(u)$ for any $u,v \in F*$. Thus the "pattern" X(v) "matches" any occurrence of X, regardless of the value of v. For example, if $X \in V_\emptyset$, then

$$\text{First}[I,X] = \begin{cases} F* \text{ if } I = [\ldots \rightarrow \bullet X\ldots] \\ \emptyset \quad \text{otherwise.} \end{cases}$$

In other words, if $I = [\ldots \rightarrow \bullet X\ldots]$, then $\langle u,v\rangle \in \text{First}[I,X]$ if and only if v is a suffix of u. (Refer to 4.2.6 for our conventions regarding the identification of certain subsets of F* with relations on F*). On the other hand, if $X \in V_1$, then

$$\text{First}[I,X] = \begin{cases} \{v\} \text{ if } I = [\ \ldots \rightarrow \bullet\langle X,v\rangle \ldots] \\ \emptyset \quad \text{otherwise.} \end{cases}$$

Similar remarks apply to other relations. ⊠

Another consequence of this notation is that the following essentially trivial statements may not be immediately obvious.

3.3.5 Lemma

(i) If R is any of the relations First, Last, Init, Fin, Before, or After, I is an item, $u,v,x \in F^*$, and $X(u) = X(v)$ (i.e., $X \in V_\emptyset$ or $u = v$), then

$\langle u,x\rangle \in R[I,X]$ if and only if $\langle v,x\rangle \in R[I,X]$

(ii) $X(u)(v) = X(uv)$. ⊠

Next we show how to represent various relevant relations among the symbols and items of $\bar{G}$ in terms of combinations of the above matrices.

3.3.6 Lemma

(i) $X(v) \rightarrow Y(w) \ldots$ (in $\bar{G}$) if and only if $\langle w,v\rangle \in (\text{First}^T\text{Init}^{-1})[Y,X]$

(ii) $X(v) \rightarrow \ldots Y(v)$ if and only if $\langle w,v\rangle \in (\text{Last}^T\text{Fin}^{-1})[Y,X]$

Proof

Suppose $X(v) \rightarrow Y(w) \ldots$. Then $\exists$ an item $I = [X(v_1) \rightarrow \bullet Y(w_1) \ \ldots]$ and an attribute $u \in F^*$ such that $X(v_1u) = X(v)$ and $Y(w_1u) = Y(w)$. Then $\langle v_1u,u\rangle \in \text{Init}[I,X]$ and by 3.3.5, $\langle v,u\rangle \in \text{Init}[I,X]$ so $\langle u,v\rangle \in \text{Init}[I,X]^{-1}$. Similarly, $\langle w_1u,u\rangle \in \text{First}[I,Y]$ so $\langle w,u\rangle \in \text{First}[I,Y] = \text{First}^T[Y,I]$. Thus $\langle w,v\rangle \in \bigsqcup\{\text{First}^T[Y,I]\text{Init}^{-1}[I,x] \mid I \text{ is an item}\} = (\text{First}^T\text{Init}^{-1})[Y,I]$

Conversely, if $\langle w,v\rangle \in (\text{First}^T\text{Init}^{-1})[Y,X]$, then there is some item I and attribute u such that

(a) $\langle w,u\rangle \in \text{First}[I,Y]$ and

(b) $\langle v,u\rangle \in \text{Init}[I,X]$.

By (a), I has the form $[\ldots \rightarrow \bullet Y(w_1) \ \ldots]$ for some w_1 such that $w_1u = w$. By (b), I has the form $[X(v_1) \rightarrow \bullet \ \ldots]$ for some v_1 such that $v_1u = v$. Hence there is a production $p = X(v_1) \rightarrow Y(w_1) \ldots$ and $p(u) = X(v_1u) \rightarrow Y(w_1u) \ldots = X(v) \rightarrow Y(w) \ldots$. This proves (i). The proof of (ii) is completely symmetrical. ⊠

3.3.7 Lemma

If M is any matrix of relations, then $\langle u,v\rangle \in M^*_{ij}$ if and only if there is a sequence $\{i_1,\ldots,i_n\}$ of indices and a sequence $\{u_1,\ldots,u_n\}$ of attributes such that $i_1 = i$, $i_n = j$, $u_1 = u$, $u_n = v$, and for $i = 1,\ldots,n-1$, $\langle u_i,u_{i+1}\rangle \in M_{i_i i_{i+1}}$.

Proof

By definition, $M^* = \bigsqcup\{M^n \mid n \geq \emptyset\}$ so $\langle u,v\rangle \in M^*$ if and only if $\langle u,v\rangle \in M^n$ for some n. We claim that $\langle u,v\rangle \in M^n$ if and only if the indicated sequences of length n exist. The proof is by induction on n. The cases $n = \emptyset$ and $n = 1$ are trivial. By 4.1.3 and 4.3.1, $\langle u,v\rangle \in M^{n+1}_{ij} = (M^nM)_{ij} = \bigsqcup\{M^n_{ik}M_{kj}\}$ if and only if $\exists\, i_n, u_n$ such that $\langle u,u_n\rangle \in M^n_{ik}$ and $\langle u_n,v\rangle \in M_{kj}$. The claim, and hence the lemma, follows from the inductive hypothesis. ⊠

3.3.8 Corollary

(i) $X(v) \overset{*}{\Rightarrow} Y(w)\ldots$ if and only if $\langle w,v\rangle \in \text{Left}^*[Y,X]$ where $\text{Left} = \text{First}^T\text{Init}^{-1}$.

(ii) $X(v) \overset{*}{\Rightarrow} \ldots Y(w)$ if and only if $\langle w,v\rangle \in \text{Right}^*[Y,X]$, where $\text{Right} = \text{Last}^T\text{Fin}^{-1}$.

Proof

$X(u) \overset{*}{\Rightarrow} Y(v)\ldots$ if and only if there is a sequence $\{X_i(w_i)\}$ such that $X(w) = X_1(w_1)$, $Y(v) = X_n(w_n)$, and $X_i(w_i) \rightarrow X_{i+1}(w_{i+1}) \ldots$ for all i. By lemma 3.3.6, this holds exactly when $\langle w_{i+1}, w_i\rangle \in \text{Left}[X_{i+1}, X_i]$ for all i. The result now follows from lemma 3.3.7. The proof of part (ii) is similar. ⊠

3.3.9 Theorem

Vec(Closure(q)) = Close·Vec(q) where Close is the matrix $(\text{Init}^{-1}\cdot\text{Left}^{*}\cdot\text{Before}^{T}) \sqcup \text{Identity}$

Proof

$I(v) \in \text{Closure}(q)$ if and only if $I(v) \in q$ or I has the form $[X(x) \rightarrow \bullet\ldots]$ and for some J and w, $J(w) \in q$, $J = [\ldots \rightarrow \ldots\bullet Y(y)\ldots]$, and $Y(yw) \overset{*}{\Rightarrow} X(xv)\ldots$. Suppose $I(v) \in \text{Closure}(q)$. If $I(v) \in q$, there is nothing more to prove, since $\text{Vec}(q) \subseteq \text{Close}\cdot\text{Vec}(q)$. Otherwise, by corollary 3.3.8, $\langle xv, yw\rangle \in \text{Left}^*[X,Y]$. By definition of Init, $\langle xv, v\rangle \in \text{Init}[I,X]$ so $\langle v, xv\rangle \in \text{Init}^{-1}[X,I]$. By definition of Before, $\langle yw, w\rangle \in \text{Before}[J,Y] = \text{Before}^T[Y,J]$. Thus $\langle v,w\rangle \in (\text{Init}^{-1}\cdot\text{Left}^*\cdot\text{Before}^T)[I,J] \subseteq \text{Close}[I,J]$. $w \in \text{Vec}(q)[J]$, so $v \in (\text{Close}\cdot\text{Vec}(q))[I]$.

Conversely, if $v \in (\text{Close}\cdot\text{Vec}(q))[I]$, then either $v \in \text{Vec}(q)[I]$ and we are done, or $\langle v,w\rangle \in (\text{Init}^{-1}[I,X] \cdot \text{Left}^*\cdot\text{Before}^T)[Y,J]$ for some X, Y, J, and w such that $J(w) \in \text{Vec}(q)$. In the latter case, there exist x and y such that $\langle v, xv\rangle \in \text{Init}[I,X]$, $\langle xv, yw\rangle \in \text{Left}^*[X,Y]$, and $\langle yw, y\rangle \in$

$\text{Before}^T[Y,J] = \text{Before}[J,Y]$. This, in turn, implies that $J(w) = [\ldots \rightarrow \ldots\bullet Y(yw)\ldots] \in q$, $Y(yw) \overset{*}{\Rightarrow} X(xv)\ldots$, and $I(v) = [X(xv) \rightarrow \bullet\ldots]$, which proves that $I(v) \in \text{Closure}(q)$ and hence $v \in \text{Vec}(\text{Closure}(q))[I]$. ⊠

3.3.10 Proposition

$\text{Vec}(\text{Scan}(q,a)) = \text{Scan}_a\cdot\text{Vec}(q)$

Proof

Let $v \in \text{Vec}(\text{Scan}(q,a))[I]$. Then $I(v) = [X(v) \rightarrow \alpha(v)a\bullet\beta(v)] \in \text{Scan}(q,a)$, so $J(v) = [X(v) \rightarrow \alpha(v)\bullet a\beta(v)] \in q$; hence $v \in \text{Vec}(q)[J]$. Moreover, by the definition of Scan_a and the forms of I and J, we know that $\text{Scan}_a[I,J] = \text{Rel}(\{e\}) = \{\langle v,v\rangle \mid v \in F^*\}$. Thus $v \in \text{Vec}(q)[J] = \text{Scan}_a[I,J]\cdot\text{Vec}(q)[J] \subseteq \bigsqcup\{\text{Scan}_a[I,J]\cdot\text{Vec}(q)[J] \mid J \text{ is an item}\} = (\text{Scan}_a\cdot\text{Vec}(q))[I]$.

Conversely, if $v \in (\text{Scan}_a\cdot\text{Vec}(q))[I]$, then there is an item J and an attribute w such that $\langle v,w\rangle \in \text{Scan}_a[I,J]$ and $w \in \text{Vec}(q)[I]$. The former fact implies that $v = w$, $I = [X \rightarrow \alpha a\bullet\beta]$, and $J = [X \rightarrow \alpha\bullet a\beta]$. Together with the latter, this implies that $J(v) \in \text{Vec}(q)$. But then $I(v) \in \text{Scan}(q,a)$, so $v \in \text{Vec}(\text{Scan}(q,a))$. ⊠

3.3.11 Corollary

$\text{Vec}(\text{Goto}(q,a)) = \text{Close}\cdot\text{Scan}_a\cdot\text{Vec}(q)$ ⊠

Putting these results together, we have:

3.3.12 Theorem

If q is any state such that Vec(q) is a vector of regular sets, and $x \in V_{\emptyset}^{*}$, then Goto(q,x) is a vector of regular sets which can be computed effectively from Vec(q).

Proof

Each Goto(-,a) can be modelled by a matrix of regular relations. By proposition 4.3.2, the resulting vector is regular. $\boxtimes$

The above may be summarized by saying that the Goto function under zero-attribute symbols may be represented by a matrix. For one-attribute symbols, the situation is slightly more complicated:

3.3.13 Proposition

If $A \in V_1$, $w \in F^*$, then $\mathrm{Vec}(\mathrm{Scan}(q,\langle A,w\rangle)) = Z$, where Z is the vector

$$Z[I] = \begin{cases} \mathrm{After}[I,A]^{-1}\{w\} \\ \quad \text{if } w \in \bigsqcup \{\mathrm{Scan}_A[I,J]\cdot\mathrm{Vec}(q)[J] \mid J \text{ is an item}\} \\ \emptyset \text{ otherwise} \end{cases}$$

Proof

Suppose $I(v) \in \mathrm{Scan}(q,\langle A,w\rangle)$. Then I has the form $[X \rightarrow \alpha\langle A,w_1\rangle\bullet\beta]$, where $w_1 v = w$, and $J(v) = [X \rightarrow \alpha\bullet\langle A,w_1\rangle\beta]$ is in q--i.e., $v \in \mathrm{Vec}(q)[J]$. Then $\mathrm{Scan}_A[I,J] = \{w_1\}$, so $w = w_1 v \in \{w_1\}(\mathrm{Vec}(q)[J]) = \mathrm{Scan}_A[I,J]((\mathrm{Vec}(q))[I])$, and $\mathrm{After}^{-1}[I,A]\{w\} = \{w_1^{-1}\}\{w\} = \{v\}$, so $v \in Z[I]$.

Conversely, if $v \in Z[I]$, then $v \in \mathrm{After}^{-1}[I,A]\{w\}$, so I has the form $[\ldots \rightarrow \ldots\langle A,w_1\rangle\bullet\ldots]$ for some w_1 such that $v_1 v = w$. Also $\exists$ J such that $w \in \mathrm{Scan}_A[I,J]((\mathrm{Vec}(q)[I])$, so $\mathrm{Scan}_A[I,J] \neq \emptyset$, which implies that J has the form $[\ldots \rightarrow \ldots\langle A,w_1\rangle\bullet\ldots]$, that $\mathrm{Scan}_A[I,J] = \{w_1\}$, and that $w \in$

$\{w_1\}(\mathrm{Vec}(q)[J])$. Thus there is an item $J(v) = [\ldots \rightarrow \ldots\bullet\langle A,w_1 v\rangle\ldots] = [\ldots \rightarrow \ldots\bullet\langle A,w\rangle\ldots]$ in q. It follows that $I(v) = [\ldots \rightarrow \ldots\langle A,w\rangle\bullet\ldots] \in \mathrm{Scan}(q,\langle A,w\rangle)[I]$ so $v \in \mathrm{Vec}(\mathrm{Scan}(q,\langle A,w\rangle))[I]$. $\boxtimes$

3.3.14 Proposition

(i) If $A \in V_1$, then for arbitrary q, w, each component of $\mathrm{Vec}(\mathrm{Scan}(q,\langle A,w\rangle))$ is a singleton or empty.

(ii) $\mathrm{Goto}(q, \langle A,w\rangle)$ is a vector of regular sets.

(iii) If Vec(q) is a vector of regular sets, then $\mathrm{Vec}(\mathrm{Goto}(q,\langle A,w\rangle))$ may be effectively computed from Vec(q).

Proof

(i) Each component of $\mathrm{Vec}(\mathrm{Scan}(q,\langle A,w\rangle))$ is either empty or of the form $\mathrm{After}^{-1}[I,A]\{w\}$. But $\mathrm{After}^{-1}[I,A]$ contains at most one member.

(ii) $\mathrm{Goto}(q,\langle A,w\rangle) = \mathrm{Closure}(\mathrm{Scan}(q,\langle A,w\rangle))$, so $\mathrm{Vec}(\mathrm{Goto}(q,\langle A,w\rangle)) = \mathrm{Close}\cdot Z$, where Z is the vector of finite (hence regular) sets defined in 3.3.13. Hence by 4.3.2, $\mathrm{Vec}(\mathrm{Goto}(q,\langle A,w\rangle))$ is a vector of regular sets.

(iii) The calculation of Z in 3.3.13 involves testing w for membership in a finite number of regular sets. $\boxtimes$

3.3.15 Theorem

For every viable prefix γ, $\mathrm{Vec}(V(\gamma))$ is a computable vector of regular sets.

Proof

The result follows immediately from theorem 3.3.12 and proposition 3.3.14. $\bowtie$

Now that we have shown how to represent each state as a vector of regular sets, we will show how to use this information to compute a matrix that tells when two grammar symbols may be adjacent in a sentential form.

3.3.16 Lemma

X(v) immediately precedes Y(w) in the right-hand-side of some production of $\overline{P}$ if and only if $\langle w,v\rangle \in (\mathrm{Before}^T\mathrm{After}^{-1})[Y,X]$.

Proof

If $p(u) = \ldots \rightarrow \ldots X(v)Y(w)\ldots$ is a production in $\overline{P}$, let $I(u) = [\ldots \rightarrow \ldots X(v)\bullet Y(w)\ldots]$. Without loss of generality, $I = [\ldots \rightarrow \ldots X(v_1)\bullet Y(w_1)\ldots]$ such that $v_1u = v$ and $w_1u = w$. (We are using 3.3.5 here. Compare with the proof of 3.3.6). Then $\langle w,u\rangle = \langle w_1u,u\rangle \in \mathrm{Before}[I,Y] = \mathrm{Before}^T[Y,I]$ and $\langle v,u\rangle = \langle v_1u,u\rangle \in \mathrm{After}[I,X]$ so $\langle u,v\rangle \in \mathrm{After}^{-1}[I,X]$. It follows that $\langle w,v\rangle \in \mathrm{Before}^T[Y,I]\mathrm{After}^{-1}[I,X] \subseteq (\mathrm{Before}^T\mathrm{After}^{-1})[Y,X]$. If $\langle w,v\rangle \in (\mathrm{Before}^T\mathrm{After}^{-1})[Y,X]$, then there is some item I and some string u such that $\langle w,v\rangle \in \mathrm{Before}[I,Y]$ and $\langle u,v\rangle \in \mathrm{After}[I,X]$. It follows that I has the form $\langle p,j\rangle = [\ldots \rightarrow \ldots X(v_1)\bullet Y(w_1)\ldots]$ where $v_1u = v$ and $w_1u = w$. Thus $p(u) = \ldots \rightarrow \ldots X(v)Y(w)\ldots$ is in $\overline{P}$ $\bowtie$

3.3.17 Proposition

There is a sentential form $\ldots X(x)Y(y)\ldots$ if and only if $\langle y,x\rangle \in \mathrm{Follow}[Y,X]$ where Follow =

$\mathrm{Left}^{*}\cdot\mathrm{Before}^T\cdot\mathrm{After}^{-1}\cdot\mathrm{Right}^{*-1T}$.

Proof

X(x) is adjacent to Y(y) in a sentential form if and only if there exist U, u, V, and v such that there is a production $\ldots \rightarrow \ldots U(u)V(v)\ldots$ in $\overline{P}$, $U(u) \overset{*}{\Rightarrow} \ldots X(x)$, and $V(v) \overset{*}{\Rightarrow} Y(y)\ldots$. If such a sentential form exists, then by 3.3.7 (i and ii) and 3.3.16, $\langle y,v\rangle \in \mathrm{Left}^*[Y,V]$, $\langle v,u\rangle \in (\mathrm{Before}^T\mathrm{After}^{-1})[V,U]$, and $\langle x,u\rangle \in \mathrm{Right}^*[X,U]$, so $\langle y,x\rangle \in \mathrm{By}[Y,X]$. The proof of the converse is similar. $\bowtie$

3.3.18 Definition

Let q be a set of items, and let $B \in \overline{V}$. Let X be the skeleton of B (the unique element of V such that $B = X(v)$ for some v) and let $X^{-1}(B) = \{v \mid X(v) = B\}$. For each $p = Y \rightarrow \alpha \in P$, let $I = [A \rightarrow \alpha\bullet] = (p,|\alpha|)$. Then $\mathrm{Reduce}(q,p,B) = \mathrm{Vec}(q)[i] \sqcap (\mathrm{Fin}^{-1}\cdot\mathrm{Follow}^{-1T})[I,X]\cdot X^{-1}(B)$.

3.3.19 Proposition

If each element of Vec(q) is regular, then Reduce(q,p,B) is a regular set which is effectively computable from q, p, and B.

Proof

$X^{-1}(B)$ is either a singleton or F^*. Hence by 4.2.8, $(\mathrm{Fin}^{-1}\cdot\mathrm{Follow}^{-1T})[I,X]\cdot X^{-1}(B)$ is regular. But the intersection of regular sets is regular. $\bowtie$

3.3.20 Proposition

w ∈ Reduce(q,p,B) if and only if

(i) I(w) = (p(w),|α|) = [A(w) → α(w)•] ∈ q, and

(ii) there is some sentential form in $\bar{G}$ of the form ...A(w)B... .

Proof

If w ∈ Reduce(q,p,B) then w ∈ Vec(q)[I] so I(w) ∈ q by definition. This proves (i).

To prove (ii), let x ∈ X^{-1}(B) (i.e., X(x) = B) such that <w,x> ∈ (Fin^{-1}·$Follow^{-1T}$)[I,X]. Then there exists Y and y such that <y,w> ∈ Fin[I,Y] and <x,y> ∈ Follow[X,Y]. <y,w> ∈ Fin[I,Y] implies that y = vw for some v such that I = [Y(v) → α•] = [A → α•]. Hence A(w) = Y(v)(w) = Y(vw) = Y(y). <x,y> ∈ Follow[X,Y] implies that Y(y) immediately precedes X(x) in some sentential form. Since A(w) = Y(y) and X(x) = B there is a sentential from of the form ...A(w)B... as required.

Conversely, if w satisfies (i) and (ii), then by (i), w ∈ Vec(q)[I]. Choose X, Y, x, y, such that X(x) = B and Y(y) = A. I = [A → α•] = [Y(y) → α•] so <yw,w> ∈ Fin[I,Y]. By (ii), A(w) = Y(yw) precedes B = B = X(x), so by 3.3.17, <x,yw> ∈ Follow[X,Y]. Hence <w,x> ∈ Fin^{-1}[I,Y]·$Follow^{-1T}$[Y,X] ⊆ (Fin^{-1}·$Follow^{-1T}$)[I,X], and x ∈ X^{-1}(B), which proves the theorem. ⊠

3.3.21 Proposition

Shift is an action in state q on input B if and only if Vec(Goto(q,B)) is not the empty vector (∅,...,∅).

Proof

Follows immediately from 2.5. ⊠

3.3.22 Definition

For each state q and symbol B ∈ $\bar{V}$, Action(q,B) is a set of actions of the form Shift or Reduce(p,w), where Shift ∈ Action(q,B) if and only if Vec(Goto(q,B)) is not the empty vector (∅,...,∅). Reduce(p,w) ∈ Action(q,B) if and only if w ∈ Reduce(q,p,B). ⊠

3.3.23 Theorem

Action(q,B) is effectively computable and regular in the sense that

(i) It is decidable whether Shift ∈ Action(q,B), and

(ii) For each p, {w | Reduce(q,w) ∈ Action(q,B)} is regular and effectively computable. ⊠

Hence, it is decidable whether Action(q,B) has zero, one, or more than one member. If Action(q,B) = ∅, then B is an illegal input in state q. If Action(q,B) = {Shift}, or Action(q,B) = {Reduce(p,w)}, then the appropriate action may be taken. If Action(q,B) has more than one member, then an SLR(1) conflict has been found in G.

4. STACKLIKE RELATIONS

It is a melancholy truth, that even great men have their poor relations.

Dickens, Bleak House

The purpose of this section is to state and prove the properties of certain kinds of relations on a set of strings which are used in sections 3 and 5.

4.1 Definitions

In the following, let F be a fixed, finite alphabet. We are concerned with relations on F^*--i.e., subsets of $F^* \times F^*$. In particular, we are interested in relations which treat members of F^* "like a stack", in the following sense:

4.1.1 Definition

For each $a \in F$, let $\hat{a} : F^* \to F^*$ be the function $\hat{a}(x) = ax$. Let $\widehat{a^{-1}} : F^* \to F^*$ be the partial function $\widehat{a^{-1}}(x) =$ if $x = ay$ then y else undefined fi. $\hat{a}$ "pushes" an a; $\widehat{a^{-1}}$ "pops" an a. Viewing $\hat{a}$ and $\widehat{a^{-1}}$ as relations, we may write $\hat{a} = \{\langle ax, x\rangle \mid x \in F^*\}$, $\widehat{a^{-1}} = \{\langle x, ax\rangle \mid x \in F^*\}$. (Note that the "input" is on the right). $\boxtimes$

At this point we emphasize that our conventions with regard to relations may be unfamiliar.

If $R \subseteq A \times B$, then we write $x\ R\ y$ to mean $\langle x,y\rangle \in R$, $Ry = \{x \in A \mid xRy\}$, and $xR = \{y \in B \mid xRy\}$. If $f : A \to B$ is a function (or partial function), then Graph(f) (or sometimes, somewhat ambiguously f) is the relation $\{\langle y,x\rangle \mid y = f(x)\} \subseteq B \times A$. This allows us to write fx unambiguously for functions and relations (up to identification of x with $\{x\}$).

4.1.2 Definition

Let $F^{-1} = \{a^{-1} \mid a \in F\}$ be a new set of symbols, isomorphic to and disjoint from F. If $w = w_1 \ldots w_k \in (F \sqcup F^{-1})^*$, let w^{-1} denote $w_k^{-1} \ldots w_1^{-1}$, where $w_i^{-1} =$ if $w_i = a^{-1} \in F^{-1}$ then a elif $w_i = a \in F$ then a^{-1} fi. Unless specified otherwise, lower case Roman letters near the end of the alphabet denote strings in F^*. Thus, for example, the notation vw^{-1} denotes a string in F^*F^{-1*}, where all symbols of v are in F, and all symbols in w^{-1} are in F^{-1}.

The hat may be extended to strings and sets of strings as follows:

If $v,w \in (F \sqcup F^{-1})^*$, then

$$\widehat{vw} = \hat{v} \circ \hat{w} = \{\langle x,y\rangle \mid \exists z \in F^* \text{ such that } \langle x,z\rangle \in \hat{v} \text{ and } \langle z,y\rangle \in \hat{w}\}.$$

If $R \subseteq (F \sqcup F^{-1})^*$, then

$$\hat{R} = \{\hat{v} \mid v \in R\}.$$

If $S \subseteq F^*$ then $\hat{R}S = \{y \mid y\hat{R}x \text{ for some } x \in S\}$. For typographical reasons, we will occasionally write Rel(R) instead of $\hat{R}$. $\boxtimes$

As examples of this notation, we have

(i) $\mathrm{Rel}(a^{-1})S = \{x \mid ax \in S\}$

(ii) $x\hat{R}y$ if and only if $\exists v \in R$ such that $x\hat{v}y$.

(iii) $\mathrm{Rel}(a^{-1}a)$ = identity function = $F^* \times F^*$

(iv) $\mathrm{Rel}(aa^{-1}) \subseteq$ identity function; $x\mathrm{Rel}(aa^{-1})x$ if and only if x begins with a.

4.1.3 Lemma

Rel is a homomorphism with respect to union and composition. In other words, if $R_i \subseteq (F \sqcup F^{-1})^*$ for all $i \in I$, then $\mathrm{Rel}(\sqcup\{R_i \mid i \in I\}) = \sqcup\{\mathrm{Rel}(R_i) \mid i \in I\}$. If $R, S \subseteq (F \sqcup F^{-1})^*$, then $\mathrm{Rel}(RS) = \hat{R}\circ\hat{S}$. (As usual, $RS = \{xy \mid x \in R$ and $y \in S\}$ and $\hat{R}\circ\hat{S} = \{\langle x,y\rangle \mid x\hat{R}z$ and $z\hat{S}y$ for some $z \in F^*\}$).

Proof

Immediate. $\boxtimes$

Notice that not all relations over F^* can be defined in this way. In fact:

4.1.4 Lemma

Let $S \subseteq F^* \times F^*$. Then $\exists R \subseteq (F \sqcup F^{-1})^*$ such that $S = \hat{R}$ if and only if S is right invariant. In other words, $\forall x,y,z \in F^*$, if xSy then $xzSyz$.

Proof

That any $\hat{R}$ is right invariant is trivial. The converse is achieved by letting $R = \{vw^{-1} \mid vSw\}$. $\boxtimes$

This lemma says that $\hat{}$ is not surjective. It is also not injective as example (iii) above shows. In fact, any set of the form $\{v^{-1}v \mid v \in S\}$ for some $S \subseteq F^*$ corresponds to the identity relation. Among the many sets that represent a given relation we shall be interested in two which are, in a sense, the largest and smallest.

4.1.5 Definition

If $R \subseteq F^* \times F^*$, let $\mathrm{Full}(R) = \{vw^{-1} \mid vRw\}$. If $R \subseteq (F \sqcup F^{-1})^*$, let $\mathrm{Full}(R) = \mathrm{Full}(\hat{R})$. $\boxtimes$

4.1.6 Proposition

If $R \subseteq F^* \times F^*$, then $R = \mathrm{Rel}(\mathrm{Full}(R))$. If $R \subseteq F^*F^{-1*}$, then $\mathrm{Full}(R) = \{vuu^{-1}w^{-1} \mid v,u,w \in F^*$ and $vw^{-1} \in R\}$ $\boxtimes$

4.1.7 Definition

If $v \in (F \sqcup F^{-1})^*$, let Cancel$(v)$ be the result of repeatedly deleting all substrings of the form $a^{-1}a$ (but not of the form aa^{-1}) from v until none remain. More formally, let $\equiv$ denote the smallest congruence on $(F \sqcup F^{-1})^*$ such that $a^{-1}a \equiv e$ for all $a \in F$. Then Cancel(v) is the shortest u such that $u \equiv v$. Let $\mathrm{Cancel}(S) = \{\mathrm{Cancel}(v) \mid v \in S\}$ and let $\mathrm{Min}(S) = \mathrm{Cancel}(S) \sqcap F^*F^{-1*}$. $\boxtimes$

4.1.8 Lemma

If $u \equiv v$ then $\hat{u} = \hat{v}$. $\boxtimes$

4.1.9 Lemma

If $v = \mathrm{Cancel}(v)$ (i.e., if v contains no $a^{-1}a$), then $\hat{v} \neq \emptyset$ if and only if $v \in F^*F^{-1*}$. $\boxtimes$

4.1.10 Proposition

$Rel(Min(R)) = Rel(R)$.

Proof

Let $S = Cancel(R)$ and $T = S \sqcap F^*F^{-1*} = Min(R)$. By lemma 4.1.8, $\hat{S} = \hat{R}$.

$\hat{S} = \bigsqcup\{\hat{v} \mid v \in S\}$ by definition 4.1.7

$= \bigsqcup\{\hat{v} \mid v \in S \sqcap F^*F^{-1*}\}$ by lemma 4.1.9

$= \hat{T}$

⋈

4.1.11 Proposition

If $R \subseteq (F \sqcup F^{-1})^*$, then $Full(R) = \{vuu^{-1}w^{-1} \mid u \in F^*$ and $vw^{-1} \in Min(R)\}$.

Proof

$Full(R) = Full(\hat{R})$ by definition 4.1.5

$= Full(Rel(Min(R)))$ by proposition 4.1.10

$= \{vuu^{-1}w^{-1} \mid u \in F^*$ and $vw^{-1} \in Min(R)\}$

by proposition 4.1.6

⋈

4.2 Regular Relations

I see that time divided is never long, and that regularity abridges all things.

Stevens, Life of Madame de Staël

4.2.1 Theorem

If $R \subseteq (F \sqcup F^{-1})^*$ is regular, then $Min(R)$ is regular. A finite automaton accepting $Min(R)$ may be effectively constructed from a finite automaton accepting R.

Proof

Let $R \subseteq (F \sqcup F^{-1})^*$ be a regular set. Let $R_1 = \{u_1 \ldots u_k \mid u_i \in F \sqcup F^{-1}$ and $\exists\ v_0 \ldots v_k \in (F \sqcup F^{-1})^*$ such that $Cancel(v_i) = e$ and $v_0u_1v_1 \ldots v_{k-1}u_kv_k \in R\}$. In words, R_1 is the set of all strings obtainable from R by cancelling substrings. By definition of Cancel, $Cancel(R) = R_1 \sqcap \{w \mid$ there is no $x,y \in (F \sqcup F^{-1})^*$ and $a \in F$ such that $xa^{-1}ay = w\}$. Since the latter set is regular, since $Min(R) = Cancel(R) \sqcap F^*F^{-1*}$, and since the family of regular sets is closed under intersection, it suffices to show that R_1 is regular. Let $M = (F \sqcup F^{-1}, Q, \delta, q_0, F)$ be a finite automaton such that $T(M) = R$. Let $S = \{\langle p,q\rangle \in Q \times Q \mid \delta(p,v) = q$ for some $v \in (F \sqcup F^{-1})^*$ such that $Cancel(v) = e\}$. Let M_1 be the non-deterministic finite automaton obtained from M by adding the e-moves $\delta(p,e) = q$ whenever $\langle p,q\rangle \in S$. An accepting computation of M_1 on input $u_1 \ldots u_k$ consists of moves of M on the symbols $u_1, \ldots, u_k$, separated by (possibly empty) sequences of e-moves. Each sequence of e-moves corresponds to the action of M on some v_i such that $Cancel(v_i) = \epsilon$. and thus $v_0u_1 \ldots j_kv_k \in T(M) = R$, whence $u_1 \ldots u_k \in R_1$. The above argument read backwards shows that $R_1 \subseteq T(M_1)$. Thus R_1, and hence $Min(R)$ is regular.

To prove that the construction is effective, we must show how to determine which pairs of states are in S. Let G be the context-free grammar $G = (\{A\}, F \sqcup F^{-1}, \{A \rightarrow a^{-1}Aa \mid a \in F\} \sqcup \{A \rightarrow AA, A \rightarrow e\}, A)$. It should be clear that $L(G)$ is precisely the set $\{v \mid \mathrm{Cancel}(v) = e\}$. Then $\langle p,q\rangle \in S$ if and only if $L(G) \sqcap \{v \mid \delta(p,v) = q\} \neq \emptyset$. The second factor of the intersection is a regular set, so the intersection is context-free. Emptiness is decidable for context-free sets. ⋈

4.2.2 Definition (Aho, Hopcroft, and Ullman [2], p. 195)

A closed semiring is a system $(S, +, \cdot, \emptyset, 1)$, where S is a set of elements, and + and $\cdot$ are binary operations on S, satisfying the following five properties:

1. $(S, +, \emptyset)$ is a monoid, that is, it is closed under + [i.e., $a+b \in S$ for all a and b in S], + is associative [i.e., $a+(b+c) = (a+b)+c$ for all a, b, c in S], and $\emptyset$ is an identity [i.e., $a+\emptyset = \emptyset+a = a$ for all a in S]. Likewise, $(S, \cdot, 1)$ is a monoid. We also assume $\emptyset$ is an annihilator, i.e., $a\cdot\emptyset = \emptyset\cdot a = \emptyset$.
2. + is commutative, i.e., $a+b = b+a$, and idempotent, i.e., $a+a = a$.
3. $\cdot$ distributes over +, that is, $a\cdot(b+c)=a\cdot b+a\cdot c$ and $(b+c)\cdot a = b\cdot a+c\cdot a$.
4. If $a_1, a_2,\ldots,a_i,\ldots$ is a countable sequence of elements in S, then $a_1+a_2+\ldots+a_i+\ldots$ exists and is unique. Moreover, associativity, commutativity, and idempotence apply to infinite as well as finite sums.

5. $\cdot$ must distribute over countable infinite sums as well as finite ones (this does not follow from property 3).

Thus (4) and (5) imply

$$(\Sigma a_i)\cdot(\Sigma b_j) = \Sigma\, a_i\cdot b_j = \Sigma(\Sigma(a_i\cdot b_j)).$$

If $a \in S$, then a^* denotes $\Sigma\{a^i \mid i \geq \emptyset\}$, where $a^\emptyset = 1$ and $a^{i+1} = a\cdot a^i$. ⋈

4.2.3 Proposition

$(L, \sqcup, \cdot, \emptyset, \{e\})$ is a closed semiring, where L is the set of languages over $F \sqcup F^{-1}$ or the set of regular languages over $F \sqcup F^{-1}$. In the latter case, the operations of +, $\cdot$, and * are effectively computable. ⋈

4.2.4 Proposition

If $R \subseteq F^* \times F^*$ is any relation, then $\mathrm{Rel}(R^*) = (\mathrm{Rel}(R))^*$. (The first * denotes Kleene closure; the second * denotes transitive and reflexive closure of a relation.)

Proof

Recall that Rel is a homomorphism with respect to concatenation and arbitrary union. Therefore $\mathrm{Rel}(R^*) = \mathrm{Rel}(\sqcup\{R^n \mid n \geq \emptyset\}) = \sqcup\{\mathrm{Rel}(R)^n \mid n \geq \emptyset\} = \mathrm{Rel}(R)^*$. ⋈

4.2.5 Corollary

The set of regular relations forms a closed semi-ring in which the operations +, $\cdot$, and * are all effective. ⋈

4.2.6 Remark

Let L be the set of subsets of $(F \sqcup F^{-1})^*$, and let $R = 2^{F^* \times F^*}$, the set of relations on F^*. By 4.1.3, $\mathrm{Rel} : L \to R$ is a homomorphism. By 4.1.4, the image Rel(L) is the set of right invariant relations. The kernel congruence of Rel is the relation $\equiv$ of 4.1.7 extended to sets by

$R_0 \equiv R_1$ if and only if for all $x \in R_0$, there is some $y \in R_1$ such that $x \equiv y$, and conversely.

By 4.1.10, the quotient algebra $L/\equiv$ is isomorphic to F^*F^{-1*}, and the canonical map $\pi : L \to L/\equiv$ which sends each element to its equivalence class is Min (up to an isomorphism). The "first isomorphism theorem" may be shown in pictures as:

$$\begin{tikzcd}
L \arrow[r, "\mathrm{Rel}"] \arrow[d, "\mathrm{Min}"] & \mathrm{Rel}(L) = \text{right invariant relations} \subseteq F^* \times F^* \\
L/\equiv\ = F^*F^{-1*} \arrow[ur, "\mathcal{W}"'] &
\end{tikzcd}$$

where $\mathcal{W}$ is the isomorphism $\mathcal{W}(S) = \{\langle v,w\rangle \mid vw^{-1} \in S\}$. By 4.2.1, all of this still holds if we restrict L to the regular subsets of $(F \sqcup F^{-1})^*$, but now all the arrows represent effectively computable functions. In this case, the image of Rel is called the set of regular relations. We will often confuse the three corners of the triangle, relying on implicit applications of the appropriate homomorphisms. For example, we will say "the relation $\{v^{-1}w\}$" when we really mean "the relation $\mathrm{Rel}(\mathrm{Min}(\{v^{-1}w\}))$", which is itself an abbreviation for

```
if $v = wx$ then $\{\langle xy,y\rangle \mid y \in F^*\}$
elif $w = vx$ then $\{\langle y,xy\rangle \mid y \in F^*\}$
    else $\emptyset$ fi.
```

However, Rel is not a homomorphism with respect to intersection or complement. For example, if R is the equality relation and S is the inequality relation, then $\mathrm{Min}(R) = \{e\}$, whereas $\mathrm{Min}(S) = \{uab^{-1}v^{-1} \mid u,v \in F^*, a,b \in F, a \neq b\}$. ⊠

4.2.7 Proposition

If R is a regular relation, then $R^{-1} = \{\langle v,w\rangle \mid \langle w,v\rangle \in R\}$ is regular (effectively).

Proof

There is some regular $S \subseteq F^*F^{-1*}$ such that $R = \mathrm{Rel}(S) = \{\langle wx,vx\rangle \mid wv^{-1} \in S, x \in F^*\}$. Then $R^{-1} = \mathrm{Rel}(S^{-1})$, where $S^{-1} = \{vw^{-1} \mid wv^{-1} \in S\}$. It suffices to show that S^{-1} is regular. Let $M = (Q, F \sqcup F^{-1}, \delta, q_0, Q_f)$ be a finite automaton accepting S. Then $M' = (Q \sqcup \{q_0'\}, F \sqcup F^{-1}, \delta', q_0', \{q_0\})$ accepts S^{-1}, where $\delta'(q_0',e) = Q_f$, and $\delta'(q,a) = \{p \in Q \mid q \in \delta(p,a^{-1})\}$ for all $q \in Q$, $a \in F \sqcup F^{-1}$. ⊠

4.2.8 Proposition

If R is a regular relation and $x \in F^*$, then Rx is a regular set. The construction is effective.

Proof

Without loss of generality, R is a regular subset of F^*F^{-1*}. Then $Rx = \{vx_2 \mid vx_1^{-1} \in R$, where $x_1x_2 = x\}$. Let $R = L(M)$ where $M = (Q, F \sqcup F^{-1}, \delta, q_0, Q_f)$ is a (deterministic) finite automaton. Let $S = \{x_2 \mid x_2$ is a suffix of $x\}$. Define a non-deterministic finite automaton M' by

$$M' = (Q \sqcup S,\ F,\ \delta',\ q_\emptyset,\ \{e\})$$

where

(i) $\delta'(q,a) = \{\delta(q,a)\}$ for $q \in Q$ and $a \in F$.

(ii) $\delta'(q,e) = T = \{x_2 \in S \mid \delta(q,x_1^{-1}) \in Q_f$ and $x_1x_2 = x\}$

(iii) $\delta'(ay,a) = \{y\}$ for all $a \in F$ and $y \in F^*$

(iv) $\delta'(e,a) = \delta'(by,a) = \emptyset$ if $b \neq a$

Intuitively, M' simulates M for a while (case(i)) and then "guesses" that it has just read the v of vx_2. It verifies that $vx_1^{-1} \in R$, where $x_1x_2 = x$ (case (ii)). Finally, it checks to see whether its guess was correct--whether the remaining input really was the guessed x_2 (cases (iii) and (iv)).

The construction is effective, since the T in case (ii) can be computed by considering each splitting $x = x_1x_2$. ⊠

4.3 Matrices of Relations

In the standard (non-infinite) case, LR analysis makes use of relations among grammar symbols represented by Boolean matrices. For example, one computes

$$By[Y,X] = \begin{cases} 1 & \text{if there is a sentential form } \ldots XY \ldots \\ \emptyset & \text{otherwise} \end{cases}$$

The analogous concept for indexed grammars is a matrix of relations:

$$By[Y,X] = \{\langle y,x\rangle \mid \text{there is a sentential form } \ldots X(x)Y(y)\ldots\}$$

This is a proper extension of the concept of Boolean ma-

trices, for in the case of two zero-parameter symbols a and b,

$$By[b,a] = \begin{cases} F^* \times F^* & \text{if there is a sentential form } \ldots ab \ldots \\ \emptyset & \text{otherwise} \end{cases}$$

where $F^* \times F^*$ is the "one" of $F^* \times F^*$ and $\emptyset$ is the "zero".

This section presents the results we need to manipulate such matrices. We are particularly interested in the case of matrices of <u>regular</u> <u>relations</u>.

4.3.1 Definition

Let R be an n by m matrix of relations (or sets) and let S be an m by p matrix. Then R·S is the n by p matrix defined by

$$(R{\cdot}S)_{ij} = \sqcup\{R_{ik}{\cdot}S_{kj} \mid 1 \leq k \leq m\},$$

R^T is the transpose of R, and R^{-1} is defined by $(R^{-1})_{ij} = R_{ij}^{-1} = \{\langle v,w\rangle \mid \langle w,v\rangle \in R_{ij}\}$. Warning: R^{-1} is an abuse of notation and is not meant to be in any sense an inverse of R. ⊠

We will follow the usual practice of linear algebra and identify an n component vector with the n by 1 matrix (column vector) and identify a subset of $(F \sqcup F^{-1})^*$ with the one-element vector and the 1 by 1 matrix. Thus the above formula also defines the application $R\bar{x}$ (where $\bar{x}$ is a vector) and the inner product of two vectors $\bar{x}^T\bar{y}$ (where ^{T} denotes transpose). Note also, that if each component of $\bar{x}$ is a subset of F^*, then the above definition extends the operation of applying a relation to a set: Suppose we wish to describe a transformation between sets of "items", where "item" might be anything capable of carrying an attribute in F*. Suppose there is a finite set $\mathbf{I} = \{I_1,\dots,I_n\}$ of "basic" items such that every set of interest is a "linear combination" of these items--that is, every set is of the form $q = \bigsqcup\{I_i(w) \mid w \in s_i,\ 1 \leq i \leq n\}$ for some vector Vec(q) $= (s_1,\dots,s_n)$. Suppose, finally, that τ, the transformation of interest, can be expressed by $I_j(w) \in \tau(q)$ if and only if $I_i(v) \in q$ for some v such that $\langle w,v\rangle \in A_{ij}$, where $A = (A_{ij})$ is a matrix of regular relations. It should be clear that what we have described is analogous to a linear transformation. In fact, $\mathrm{Vec}(\tau(q)) = A \cdot \mathrm{Vec}(q)$. It is the goal of section 3.3 to show that all transformations relevant to the parsing of indexed grammars are linear in this sense.

The following proposition shows that many operations on matrices of <u>regular relations</u> can be carried out effectively in the following sense: a regular relation can be represented by a regular set which, in turn, may be represented by a regular expression. Thus a matrix of relations can have a finite representation as a matrix of regular expressions.

4.3.2 Proposition

Given finite representations of matrices R and S, we can effectively compute representations of $R \sqcup S$, $R \circ S$, R^{-1}, R^T, and $R^* = \bigsqcup\{R^n \mid n \quad \emptyset\}$.

<u>Proof</u>

The proofs for $R \sqcup S$, $R \circ S$, R^{-1}, and R^T are trivial. An algorithm for R* is the following paraphrase of Algorithm 5.5, page 198 of Aho, Hopcroft, and Ullman[3]. Define a sequence $C^{\emptyset},\dots,C^n$ of matrices of regular expressions, where

$$C^{\emptyset}_{ij} = \begin{cases} \mathrm{RExp}(R_{ij}) \text{ if } i \neq j \\ \mathrm{RExp}(R_{ij}) \sqcup \{e\} \text{ if } i = j \end{cases}$$

(RExp(R) is a regular expression for R).

For k = 1, ..., n compute

$$C^k_{ij} = C^{k-1}_{ij} \sqcup C^{k-1}_{ij} \cdot (C^{k-1}_{kk})^* \cdot C^{k-1}_{kj}$$

Straightforward but tedious "arrow-chasing" (together with Kleene's theorem) shows that $C_{ij}{}^n$ is a regular expression representing $R_{ij}{}^*$. ⋈

4.3.3 Corollary

For each n, the set of n by n matrices of regular relations is a closed semi-ring (4.2.2) $(S, \sqcup, \cdot, \emptyset, 1)$, where $\sqcup$ is computed component-wise, $\cdot$ is as defined in 4.3.1, and $\emptyset$ and 1 are the matrices:

$(\emptyset)_{ij} = \emptyset$,

$(1)_{ij} =$ <u>if</u> i=j <u>then</u> {e} <u>else</u> ∅ <u>fi</u>.

⋈

5. CONCLUSIONS AND DIRECTIONS FOR FURTHER RESEARCH

The work reported in chapter II probably raises more questions than it answers. We have shown that the usual definition of LR(k) grammars may be generalized to indexed grammars in two ways: one way considers the indexed grammar to be a context-free "skeleton" together with extra information, and requires the skeleton to be LR(k). Thus the extra information is not used to help determine the skeletal structure of an input string. This approach is analogous to the usual treatment of attribute grammars and related formalisms [4,19,16,27]. The second and, we feel, more interesting generalization considers an indexed grammar to be a specification of an infinite grammar. The LR(k) definition extends without modification to infinite grammars and, _a fortiori_, to indexed grammars. The principal result of chapter II is that the SLR(1) parsing algorithm may be applied to infinite grammars which are specified by indexed grammars.

Several important questions remain unanswered. First, the "traditional" theory gives more than a parsing algorithm: a grammar may be tested for the LR(k) property before any inputs are parsed. If the grammar passes the test, then it is certified to be unambiguous, and the parsing algorithm

is guaranteed to parse all strings in the language correctly. Conversely, if the grammar fails the LR(k) test, then the parser will fail to parse at least one input string and the grammar may or may not be ambiguous. (Whether the grammar is in fact ambiguous is recursively undecidable ([12])). We do not know whether an analogous result holds for indexed grammars--that is, we do not know whether the class of LR(1) indexed grammars (or, for that matter, SLR(1) or even LR(0) indexed grammars) is recursive. A demonstration that the SLR(1) property (or some more restrictive property) is decidable would be of considerable practical value, since it would make possible the determination that the type-checking or "coercion" rules of a language (cf. [30]) are unambiguous.

A second area requiring more research is an investigation into the practicality of SLR(1) indexed grammars for specifying language features. More experience is required to determine whether the method of indexed grammars is adequately expressive, and whether the SLR(1) restriction is sufficiently loose. A related issue is computational complexity. Results in section 4 show that various properties are decidable, but no attempt has been made to measure the amount of time or space needed for the decision procedure. If the complexity of the parsing algorithm turns out to be excessive, then perhaps some method can be found for partially pre-computing the actions when the grammar is analyzed, thus reducing the amount of computation required while parsing.

Finally, as we indicated in section 1, we feel there is considerable potential for extending the technique beyond the limits imposed by indexed grammars. First, it would be convenient to allow more than one attribute on a symbol. Second, it might be useful to have attributes to come from domains other than F*. For example, we showed in chapter I of this dissertation that data types can be modelled by infinite trees. Therefore, the choice of domain for a "type" attribute is T[Γ], the set of trees over the alphabet Γ (c.f. 3.1.2 of chapter I).

God keep me from ever completing anything. This whole book is but a draught--nay, but the draught of a draught. Oh, Time, Strength, Cash, and Patience!

Melville, Moby Dick

References

[1] Aho, A. V. Indexed grammars--an extension of context-free grammars. J. ACM 15 (1968), 647-671.

[2] Aho, A. V., Hopcroft, J. E., and Ullman, J. D. The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Mass., 1974.

[3] Aho, A. V. and Ullman, J. D. The Theory of Parsing, Translation and Compiling. Prentice-Hall, Englewood Cliffs, N.J., 1973.

[4] Bochmann, G. V. Semantic evaluation from left to right. Comm. ACM 19, 2 (Feb., 1976), 55-62.

[5] Courcell, B. Recursive schemes, algebraic trees, and deterministic languages. IEEE Symposium on Switching and Automata Theory, v. 15, 1974, pp. 52-62.

[6] Fischer, M. J. Grammars with Macro-like Productions. Ph.D. Thesis. Harvard University, 1968.

[7] Floyd, R. W. On the non-existence of a phrase structure grammar for Algol 60. Comm. ACM 5, 9 (Sept., 1962), 483-484.

[8] Floyd, R. W. Syntactic analysis and operator precedence. J. ACM 10, 2 (July, 1963), 316-333.

[9] Gehani, N. Data Types for Very High Level Languages. Ph. D. Thesis. Tech. Rept. TR 75-258, Computer Science Dept., Cornell University, Ithaca, N.Y., 1975.

[10] Goguen, J. A. and Thatcher, J. W. Initial algebra semantics. IEEE Symposium on Switching and Automata Theory, v. 15, 1974, 63-77.

[11] Gries, D. and Gehani, N. Some ideas on data types in high level languages. Tech. Rept. TR 75-244, Computer Science Dept., Cornell University, Ithaca, N. Y., 1975.

[12] Hopcroft, J. E. and Ullman, J. D. Formal Languages and their Relation to Automata. Addison-Wesley, Reading, Mass., 1969.

[13] IFIP Working Group 2.1. Report of the subcommittee on data-processing and transport--modals, with application to sorting. Algol Bulletin AB 37.4.3 (May 1971).

[14] Irons, E. T. A syntax directed compiler for Algol 60. Comm. ACM 4, 1 (Jan, 1961), 51-55.

[15] Jensen, K. and Wirth, N. Pascal User Manual and Report. Lecture Notes in Computer Science, v. 18.

Springer Verlag, Belin, 1974.

[16] Knuth, D. E. Semantics of context-free languages. Math. Syst. Theory 2 (1968), 127-145. See also correction: Math. Syst. Theory 5 (1971), 95-96.

[17] Kral, J. The equivalence of modes and the equivalence of finite automata. Algol Bulletin AB 35.4.5 (March 1973).

[18] Lewis, C. H. and Rosen, B. K. Recursively defined data types, part I. ACM Symposium on Principles of Programming Languages, Boston, Mass., 1973.

[19] Lewis, P. M., Rosenkrantz, D. J., and Stearns, R. E. Attributed translations. J. Comp. and Syst. Sci, 9 (1974) 279-307.

[20] Lindsey, C. H. Modals. Algol Bulletin AB 37.4.3 (1974).

[21] Rosen, B. K. Tree-manipulating systems and Church-Rosser theorems. J. ACM 20, 1, (January, 1973), 160-187.

[22] Rosen, B. K. and Lewis, C. H. Recursively defined data-types, part II. Report RC 4713, IBM T. J. Watson Research Center, Yorktown Heights, N. Y., 1974.

[23] Samelson, K., and Bauer, F. L. Sequential formula translation. Comm. ACM 4, 2 (Feb., 1960), 76-83.

[24] Scott, D. The Lattice of flow diagrams, in _Semantics of Algorithmic Languages_, E. Engler, ed., Lecture Notes in Mathematics, v. 188, Springer Verlag, Berlin, 1971.

[25] Sintzoff, M. Existence of a van Wijngaarden syntax for every recursively enumerable set. _Am. Soc. Sci de Bruxelle_, _81_ (1967).

[26] Solomon, M. Modes, values and expressions. Second ACM Symposium on Principles of Programming Languages, Palo Alto, Calif., 1975, 149-159. (Also available as Tech. Rept. TR 74-219, Computer Science Dept., Cornell University, Ithaca, N. Y., 1974).

[27] Stearns, R. E., and Lewis, P. M. Property grammars and table machines. _Info. and Control_ _14_ (1969), 524-549.

[28] Tarski, A. A lattice-theoretical fixpoint theorem and its applications. _Pacific J. of Math._ _5_ (1955), 285-309.

[29] Warshall, S. A theorem on Boolean matrices. _J. ACM_ _9_, 1 (Jan 1962), 11-12.

[30] Wijngaarden, A. van _et al_. Revised report on the algorithmic language ALGOL 68. _Acta Informatica_ _5_ (1975), 1-236.