N71-34523

SUBTREE REPLACEMENT SYSTEMS[†]

by

Barry K. Rosen

2-71

Center for Research in Computing Technology

VE RI TAS

Harvard University
Cambridge, Massachusetts 02138

SUBTREE REPLACEMENT SYSTEMS[†]

by

Barry K. Rosen

2-71

Center for Research in Computing Technology

Harvard University

Cambridge, Massachusetts 02138

[†]The first four chapters of this report formed the author's Ph. D. thesis at Harvard University, 1971. Some of these results were announced at the Second ACM Symposium on the Theory of Computing, Northampton, Massachusetts, May, 1971. This research was supported in part by the National Aeronautics and Space Administration under Grant No. NGR 22-007-076, and by the National Science Foundation under Grant No. GS-1934.

PREFACE

At the end of September 1969 I began to study some recent work on finite tree automata and tree transducers. I was also concerned with another kind of tree-manipulating system: formal computations in McCarthy's calculus for recursive definitions [34]. Recursively defined functions were obviously singlevalued, even in versions of the calculus that allowed much more freedom than the original one in deciding which formal procedure call to evaluate next, yet a rigorous proof was strangely elusive.

The proof that eventually emerged had two stages. First the "subtree replacement system" defined by any recursive definition was shown to be "unequivocal" and "closed." Then singlevaluedness was derived from these properties alone. The ensuing search for other applications of the abstract singlevaluedness theorem led immediately to some minor results about tree transducers. I then conceived the outline of a relatively simple proof of the Church-Rosser Theorem for the lambda calculus. By late November it was clear that a short paper should be written.

The paper grew. During 1970 it gradually turned into a long paper [42] and then into this thesis. More questions about the application areas arose, and some of them were answerable with the help of the growing theory. The current state of the theory of subtree replacement systems and most of the applications explored so far are presented here. This work does not solve any software engineering problems; it provides mathematical tools that may help those who do solve them.

In 1960 McCarthy [33] pioneered the mathematical basis for designing, judging, and communicating practical nonnumerical algorithms, especially linguistic algorithms. Modest but nontrivial progress has been made since then. The software engineer now has some science to apply to his problems. He still needs experience, creativity, perseverance, and luck – as do all the other engineers. He must still envy the riches of his brother in hardware, who can draw on solid state physics and electromagnetic theory. I hope this thesis will help him feel a little less impoverished.

I wish to thank P. J. Downey for suggesting an improvement in the definition of closed subtree replacement systems. Prof. R. Hindley and Dr. G. Mitschke kindly supplied copies of their dissertations, [21] and [36], for comparison with the work reported in [41]. Prof. Hindley also called my attention to [47] and suggested some improvements in the proof of the classical Church-Rosser Theorem. Profs. F. D. Lewis and W. A. Woods suggested many improvements in the exposition. I am especially grateful to Prof. R. V. Book, who supervised this research. His guidance and support have been beyond the power of words to acknowledge.

This research was supported by the National Science Foundation under Grant No. GS-1934 and by the National Aeronautics and Space Administration under Grant No. NGR-22-007-176.

CONTENTS

INDEX OF MAJOR DEFINITIONS

The following index locates the definitions for the major concepts involved in this work. Every concept defined in one chapter and used in another is included, except for the standard mathematical terminology reviewed in §2.

Page intentionally left blank

Page intentionally left blank

SYNOPSIS

Subtree replacement systems form a broad class of tree-manipulating systems including many of the special cases from logic, linguistics, and automata theory. Subtree replacement systems with the Church-Rosser property are appropriate for evaluation or translation processes. In a Church-Rosser system, the final result of operating on a tree (then on the resulting tree, and so on, until no further changes are possible) does not depend on which of several possible operations is performed at each stage. This singlevaluedness permits the system to specify "meanings" or "values" for trees unambiguously. Such flexibility has both theoretical and practical advantages.

Chapter 1 motivates the mathematical study of subtree replacement systems and outlines the main results. We also review common mathematical terminology to be used throughout the rest of this thesis.

Chapter 2 presents the abstract theory of general replacement systems (§3) and subtree replacement systems (§5). The preliminary results in §3 apply to any situation where changes in the data are made in discrete steps, whether or not the data are trees. Theorem 3.5 and Lemma 3.6 in this section were discovered by Hindley [21] and independently by the author. The other results and the diagrammatic proof technique are new.

Slight extensions of Brainerd's arithmetic of trees [6] are described in §4. Using this formal computational technique in §5, we establish sufficient conditions for the Church-Rosser property in subtree replacement systems. The abstract conditions are fairly easy to

verify for several important subtree replacement systems that bear little resemblance to each other at the level of concrete detail. The concepts and results in §5 are new. It is sometimes rather tedious to verify our sufficient conditions. In order to avoid burdensome repetition in these verifications, we present part of the argument abstractly in §6.

Chapter 3 applies the theory to McCarthy's calculus for recursive definitions [34], extended and disambiguated by the addition of an explicit choice between call-by-value and call-by-name. In §7 we prove a singlevaluedness result that had not been proven before without severe restrictions on the use of call-by-name [3][31]. Recursive definitions may also be interpreted as equations to be solved: an assignment of functions to the function variables in a definition may or may not force all its equations to be true statements. In §8 we show that the functions specified by the extended McCarthy calculus for any recursive definition do solve the definition, and we show how this solution is related to any other solutions that may exist. This result verifies a conjecture of Morris [37]. When restricted to definitions that call all their parameters by value, our result is closely related to the "first recursion theorem" [26]: any partial recursive functional has a unique minimal fixed point that is partial recursive. The proofs of the results in §7 and §8 are not based on the proofs for the previously known special cases.

Chapter 4 applies the theory to the full lambda calculus, including eta and delta rules as defined by Curry and Feys [14]. We define the calculus in §9, using a novel approach to alphabetic equivalence that permits a higher degree of mathematical rigor than does the conventional approach. In §10 we prove the classical Church-Rosser Theorem of [14] with a divide-and-conquer strategy. The lambda calculus is analyzed as a hierarchy of simpler systems. The main theorem from §5 implies that the parts are Church-Rosser. We then apply results from §3 to the effect that appropriately connected Church-Rosser parts form Church-Rosser wholes. The hierarchy method was discovered by Hindley [21] and independently by the author [41]. Mitschke [36] independently used a construction similar to ours for the proof that the part formed by beta reduction is Church-Rosser. Mitschke's proof that the construction is correct differs substantially from ours. Neither proof has much in common with traditional approaches to beta reduction [13][14][21][22][48].

Chapter 5 applies the theory of subtree replacement systems to finite tree transducers and describes the use of these transducers in compiler design. In §11 we review some relevant topics in formal language theory. In §12 we sketch a model for syntax-directed compilation as a five-stage process. One of the five stages is a tree transducer: an abstract device which maps trees to trees. The model is a natural combination of two traditions in computer science: formal language theory (including the theory of tree transducers) and more concrete discussions of compiler design problems [7][43][54]. The basic theory of tree transducers is developed in §13. This section is mostly exegesis on the work of Rounds [45][46] and Thatcher [51][52], with emphasis on the significance of this work for compiler construction. In §14 we consider the question of whether the composition of two maps, both definable by a certain type of transducer, is also definable by a transducer of the same type. We elaborate slightly upon theorems of Rounds [46] and Thatcher [52] to the effect that certain classes of transductions are indeed closed under composition. (A transduction is the map defined by a transducer.) We also show that there is a pair of "linear" and "partial deterministic" transductions whose composition is not computable by any finite tree transducer, deterministic or non-deterministic, that reads input trees from the top down. This result is new.

Chapter 6 summarizes our conclusions and indicates some possible directions for further research on the theory and applications of subtree replacement systems.

CHAPTER 1

INTRODUCTION

This chapter begins with a preliminary intuitive description of the main results of this thesis and their motivation. It concludes with a survey of the basic mathematical notation to be used in the following, more technical chapters.

1. Motivation and Overview

This section reviews one of the several ways to use trees in representing the structures of expressions in natural or artificial languages. We then consider methods for specifying the meanings of expressions in terms of rules for manipulating their tree structures. We introduce "subtree replacement systems" as a general class of such tree-manipulating systems. The abstract study of subtree replacement systems leads to results useful in understanding many of the particular systems from logic, linguistics, and automata theory. The main results of this thesis are outlined here on an intuitive level.

We begin with a type of structure familiar from mathematics and logic. Consider the arithmetic expression

$$(47 \times 23 - 981) \times 6\,!$$

built up from numerals that denote numbers and <u>operators</u> that denote <u>operations</u> on numbers. A sequence of <u>operands</u> is associated with each occurrence of an operator: the operands are subexpressions denoting the numbers on which the corresponding operation is to be performed. The first occurrence of X has the operands 47 and 23. The only occurrence of - has the operands 47 X 23 and 981.

The structure of operators and operands in (47 X 23 - 981) X 6 ! is displayed in the tree at the top of Figure 1-1. For each occurrence of an operator there is a node labelled by that operator. If a node m is labelled by an operator α, then m has a son for each operand of the corresponding occurrence of α. The subtree rooted at each son of m displays the structure of the operand it represents. We call such trees <u>operator-operand</u> <u>structures</u>. Let $\overline{47}$ be the tree with one node labelled by the numeral 47 and no other nodes. Let $\overline{\times}(\overline{47}, \overline{23})$ be the tree consisting of a root labelled X and then two sons of the root, with the first son labelled 47 and the second one labelled 23. (Thus we are naming trees by means of strings in a vocabulary consisting of the possible labels plus grouping symbols. The same idea with different punctuation is used in labelled bracket notation and in Cambridge Polish notation.) The tree at the top of Figure 1-1 can be described as $\overline{\times}(\overline{-}(\overline{\times}(\overline{47}, \overline{23}), \overline{981}), \overline{!}(\overline{6}))$.

The pair of trees

$$\overline{\times}(\overline{47}, \overline{23}) \longrightarrow \overline{1081}$$

represents the fact that 47 X 23 = 1081. The familiar algorithms for computing with decimal numerals generate an infinite set $\mathbb{R}$ of pairs of trees, including

$$\overline{!}(\overline{6}) \longrightarrow \overline{720}$$

$$\overline{-}(\overline{1081}, \overline{981}) \longrightarrow \overline{100}$$

$$\overline{\times}(\overline{100}, \overline{720}) \longrightarrow \overline{72000}.$$

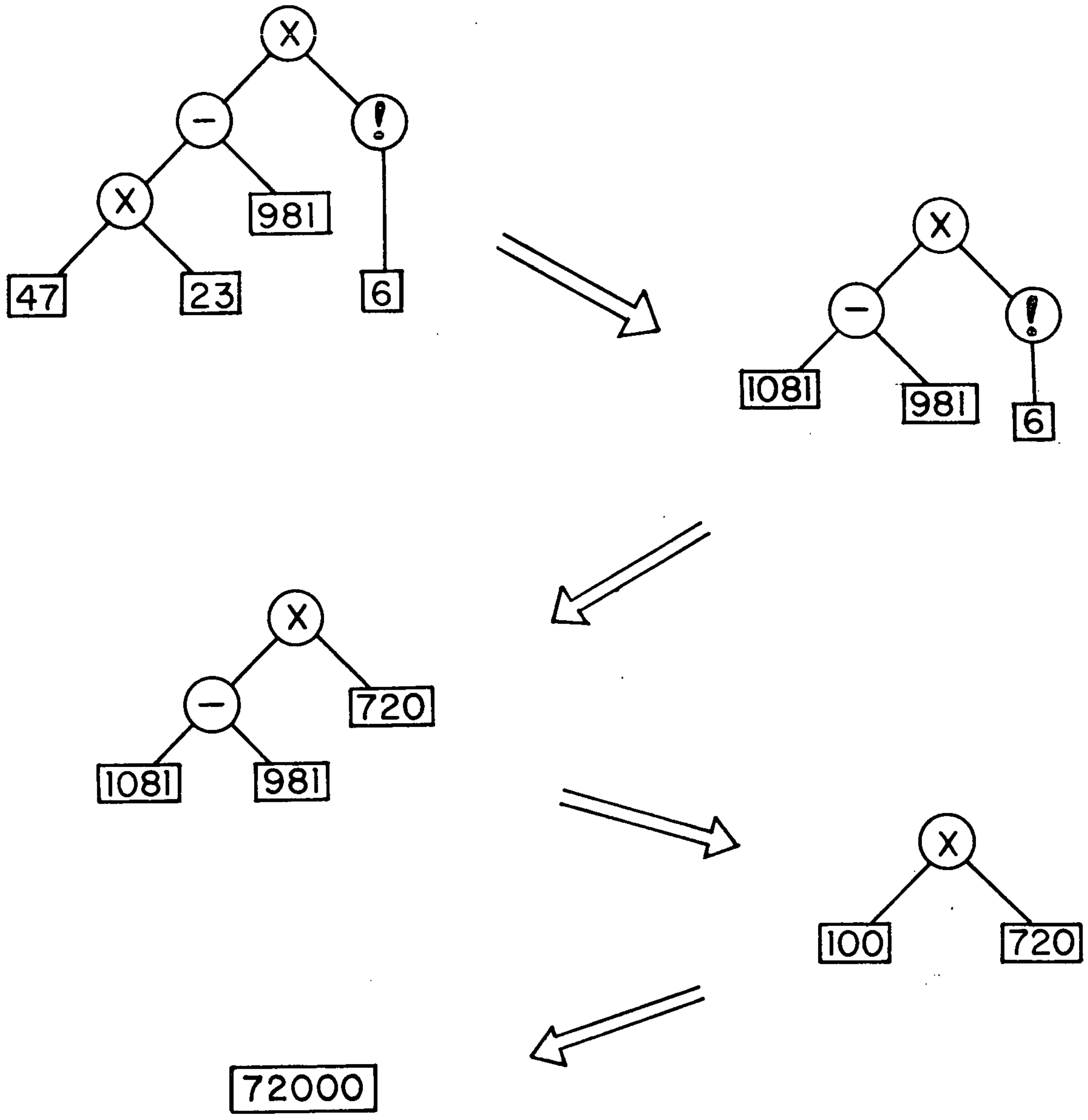


Figure 1-1. One of three ways to derive 72000 from

(47 X 23 - 981) X 6! in bottom-up arithmetic.

As the use of an arrow rather than brackets and commas suggests, we will think of pairs of trees as "rules" for replacing parts of trees by other trees. To apply a rule to a tree we must do two things: match the left half of the rule against a subtree of the tree and then replace that subtree by the right half of the rule.

When a tree R can become another tree S by the application of a rule to a subtree of R, we write $R \Longrightarrow S$ and pronounce this as "R can become S." In Figure 1-1 we display one of the ways to derive the tree $\overline{72000}$ from the operator-operand structure of $(47 \times 23 - 981) \times 6\,!$.

The evaluation process is a <u>nondeterministic</u> <u>algorithm</u>: an algorithm that includes instructions to choose among several alternatives as well as the more familiar assignment and branching instructions. To evaluate a tree R we must apply a rule to R, then apply a rule to the resulting tree, and so on, until eventually we arrive at a tree T to which no rules can be applied. We say that T is a <u>normal form</u> for R. At any step in the process there may be several rules that could be applied. In this example the nondeterminism has no net effect: each tree R has just one normal form T, and any sequence of applications of rules beginning at R will lead to T. Our system for evaluating trees by means of rules from arithmetic has two desirable properties:

(a) Every R has at most one normal form T.

(b) Every sequence $R_0, R_1, R_2, \ldots$ such that $R_i \Longrightarrow R_{i+1}$ for each i is a finite sequence.

Property (a) permits the set of rules to specify normal forms unambiguously without the additional complication of a sequencing mechanism that computes which rule to apply next and where to apply it, at each stage of the evaluation process. If our system had property (a) but not property (b), then we could still speak of "the normal form, if any, of R" but we would not have complete freedom in choosing a sequencing mechanism. An infelicitous choice might omit some normal forms by defining an infinite sequence $R = R_0 \Longrightarrow R_1 \Longrightarrow \ldots$ even when R does have a normal form.

The evaluation of arithmetic expressions is one example of a system for evaluating trees by applying rules. In general, a subtree replacement system is specified by

(1) a vocabulary V with which to label nodes of trees

(2) a set $\mathbb{F}$ of trees with labels in V

(3) a set $\mathbb{R}$ of rules (pairs of trees)

(4) the binary relation $\Longrightarrow$ on $\mathbb{F}$ defined by the application of rules to trees at nodes.

Application of rules is defined just as in our example. First, match the left half of a rule against the subtree of R at some node. Second, replace that subtree by the right half of the rule, so that R becomes a new tree S. We write $R \Longrightarrow S$. We remark that this matching and replacement process is similar to the way a grammar's rules are applied to strings in formal language theory.

Suppose a subtree replacement system has property (a): normal forms are unique. The system defines an unambiguous notion of semantics for trees: the "meaning" or "value" of a tree is its unique normal form, if any. Normal forms are defined by a simple mathematical system, and knowledge of their properties will presumably be helpful in choosing an efficient sequencing mechanism to actually find them. If property (b) holds too, then the choice of sequencing mechanism is free from concern about infinite computations. If property (b) does not hold, then we must try to choose a sequencing mechanism that finds a normal form for every tree that has one, but at least we are assured that the normal forms we do find are correct.

Subtree replacement systems are defined formally in Chapter 2, where sufficient conditions for uniqueness of normal forms are established. (Property (b), the finiteness of all sequences of applications of rules, will not be studied in any detail in this thesis.) The Main Theorem (5.6) in Chapter 2 asserts that every "unequivocal" and "closed" subtree replacement system is "Church-Rosser." It is not very difficult to establish that several important systems are unequivocal and closed; it is quite trivial to prove that normal forms are unique in Church-Rosser systems. Formal definitions for the words "unequivocal" and "closed" and "Church-Rosser" are in Chapter 2, but the intuitive content of these notions can be sketched here.

We begin with the Church-Rosser property. Suppose that S and S′ are trees that can both be derived from a tree R by applying rules. Is there a tree T that can be derived from both S and S′ by applying rules? A _Church-Rosser_ system is one where the answer to this question is always affirmative. If one sequence of applications of rules leads from R to S and another leads from R to S′, then both sequences can be extended so as to meet at a common tree T. If $S \neq S'$ then both trees can be processed further and neither can be a normal form for R. Therefore normal forms are unique in Church-Rosser systems. To exploit this observation we need sufficient conditions for the Church-Rosser property that are easier to verify than the property itself. We therefore consider "unequivocal" and "closed" systems.

A subtree replacement system is unequivocal if the set of rules is a partial function on trees: no two rules have the same left half. Our example from arithmetic is such a system. If $\varphi \longrightarrow \psi$ is a rule then φ has the form $\overline{\alpha}(\overline{x})$ or $\overline{\alpha}(\overline{x}, \overline{y})$, where α is an arithmetic operator while x and y are numerals. The right half ψ must be of the form $\overline{z}$, where z is the numeral representing the result of applying the operation denoted by α to the numbers denoted by x or x and y. Because the operations are singlevalued, z is determined by α together with x or x and y. Therefore ψ is determined by φ, and there could not be another rule $\varphi \longrightarrow \psi'$ with $\psi' \neq \psi$.

The evaluation process in an unequivocal system is still nondeterministic. At most one rule is applicable at any node in a tree because there is at most one rule whose left half is the subtree rooted at that node, but there may be several nodes where rules could be applied. The rules

(i) $\overline{\times}(\overline{47}, \overline{23}) \longrightarrow \overline{1081}$

and

(ii) $\overline{!}(\overline{6}) \longrightarrow \overline{720}$

are both applicable to the tree at the top of Figure 1-1. Applying (i) leads to the tree

(1) $\overline{\times}(\overline{-}(\overline{1081},\overline{981}),\overline{!}(\overline{6}))$.

Applying (ii) leads to the tree

(2) $\overline{\times}(\overline{-}(\overline{\times}(\overline{47},\overline{23}),\overline{981}),\overline{720})$.

By applying (ii) to (1) and (i) to (2), we can derive

(3) $\overline{\times}(\overline{-}(\overline{1081},\overline{981}),\overline{720})$,

so the difference between (1) and (2) is only transitory. Our system is indeed Church-Rosser, and in part because it is unequivocal, but there is a subtler phenomenon involved as well: applying (i) at one place in a tree does not interfere with the attempt to apply (ii) somewhere else. Having derived S from R and S′ from R, we know just which rules we wish to use in deriving some common tree T from both S and S′; we also know that we can indeed apply these rules.

Now we must complicate our example in order to show how rules might interfere with each other, but not seriously enough to prevent the system from being Church-Rosser. We add conditional expressions to our arithmetic. In any expression of the form

if P **then** A **else** B

we say that the *conditional operator* C has the operands P, A, B. We introduce **true**, **false**, and predicate symbols for building up test expressions P. To evaluate conditional expressions we use a set of rules that includes

(iii) $\overline{C}(\mathbf{true},\overline{!}(\overline{6}),\overline{\times}(\overline{47},\overline{23}) \longrightarrow \overline{!}(\overline{6})$

and

(iv) $\overline{C}(\mathbf{true},\overline{720},\overline{\times}(\overline{47},\overline{23})) \longrightarrow \overline{720}$.

If (iii) is applicable at a node n in a tree R, so that the subtree of R at n is $\overline{\mathrm{C}}(\utilde{\mathrm{true}}, \overline{!}(\overline{6}), \overline{\mathrm{X}}(\overline{47}, \overline{23}))$, then (ii) is applicable at the second son of n. Call this node p. Applying (iii) at n would destroy the opportunity to apply (ii) at p; applying (ii) at p would destroy the opportunity to apply (iii) at n. No permanent harm is done, however. Suppose we apply (iii) at n, so that R is replaced by a tree S that is like R except that the subtree of S at n is $\overline{!}(\overline{6})$. Applying (ii) at n in S, we get a tree T that is like R except that the subtree of T at n is $\overline{720}$. On the other hand, suppose we apply (ii) at p, so that R is replaced by a tree S′ that is like R except that the subtree of S′ at p is $\overline{720}$. Then the subtree of S′ at n is $\overline{\mathrm{C}}(\utilde{\mathrm{true}}, \overline{720}, \overline{\mathrm{X}}(\overline{47}, \overline{23}))$, so we may apply (iv) at n to derive a tree T′ that is like R except that the subtree of T′ at n is $\overline{720}$. But this means that T = T′.

The reason that (ii) and (iii) can interfere at all is that (ii) is applicable at the second son of the root in (iii) and hence at the second son of any node in a tree where (iii) is applicable. The reason that such interference does not destroy the Church-Rosser property is that another rule (iv) can compensate for an application of (ii) that preempts an application of (iii): applying (ii) and then (iv) has the same effect as applying (iii) and then (ii). The rule (iv) is formed from (iii) by applying (ii) at the second son of the root in $\overline{\mathrm{C}}(\utilde{\mathrm{true}}, \overline{!}(\overline{6}), \overline{\mathrm{X}}(\overline{47}, \overline{23}))$ $\underline{\text{and}}$ at the corresponding node (it happens to be the root) where $\overline{!}(\overline{6})$ reappears in $\overline{!}(\overline{6})$ on the right half of (iii).

In general we are concerned about interference between rules whenever a little rule $\varphi \longrightarrow \psi$ is applicable inside the left half φ_0 of a big rule $\varphi_0 \longrightarrow \psi_0$. Applying $\varphi \longrightarrow \psi$ in φ_0 leads to a tree φ_1. If we can apply $\varphi \longrightarrow \psi$ at appropriate places in ψ_0 to form a tree ψ_1 such that $\varphi_1 \longrightarrow \psi_1$ is a rule of the system, then we say the system is closed. Figure 1-2 illustrates this idea under the assumption that $\varphi \longrightarrow \psi$ is applicable at n in φ_0 and at p and q in ψ_0.

In our example with (ii), (iii), and (iv), the appropriate nodes in ψ_0 were obvious, but for the general theory we must define just what kind of assignment of "corresponding" nodes in ψ_0 to each node in φ_0 will be allowed. In Chapter 2 we define a residue map for $\varphi_0 \longrightarrow \psi_0$ to be any assignment of sets of nodes in ψ_0 to nodes in φ_0 such that certain conditions hold. The most important of these is that, whenever n is a node in φ_0 and q is one of the "residues" assigned to n in φ_0, then the subtree of ψ_0 at q is a copy of the subtree of φ_0 at n. Any little rule $\varphi \longrightarrow \psi$ applicable at n in φ_0 will then be applicable at each residue of n in ψ_0. By applying $\varphi \longrightarrow \psi$ at n in φ_0 and at each residue of n in ψ_0, we can form a pair of trees $\varphi_1 \longrightarrow \psi_1$. The crucial property of a closed subtree replacement system is that $\varphi_1 \longrightarrow \psi_1$ is a rule of the system. In that case the interference between $\varphi \longrightarrow \psi$ and $\varphi_0 \longrightarrow \psi_0$ can be shown to be harmless. As we remarked earlier, the Main Theorem in Chapter 2 asserts that every unequivocal closed subtree replacement system is Church-Rosser.

The Main Theorem and some other abstract results from Chapter 2 are applied to various situations in the succeeding chapters. The first application is to recursive definitions.

A function may be defined by an equation of the form

$$f(x) := \ldots$$

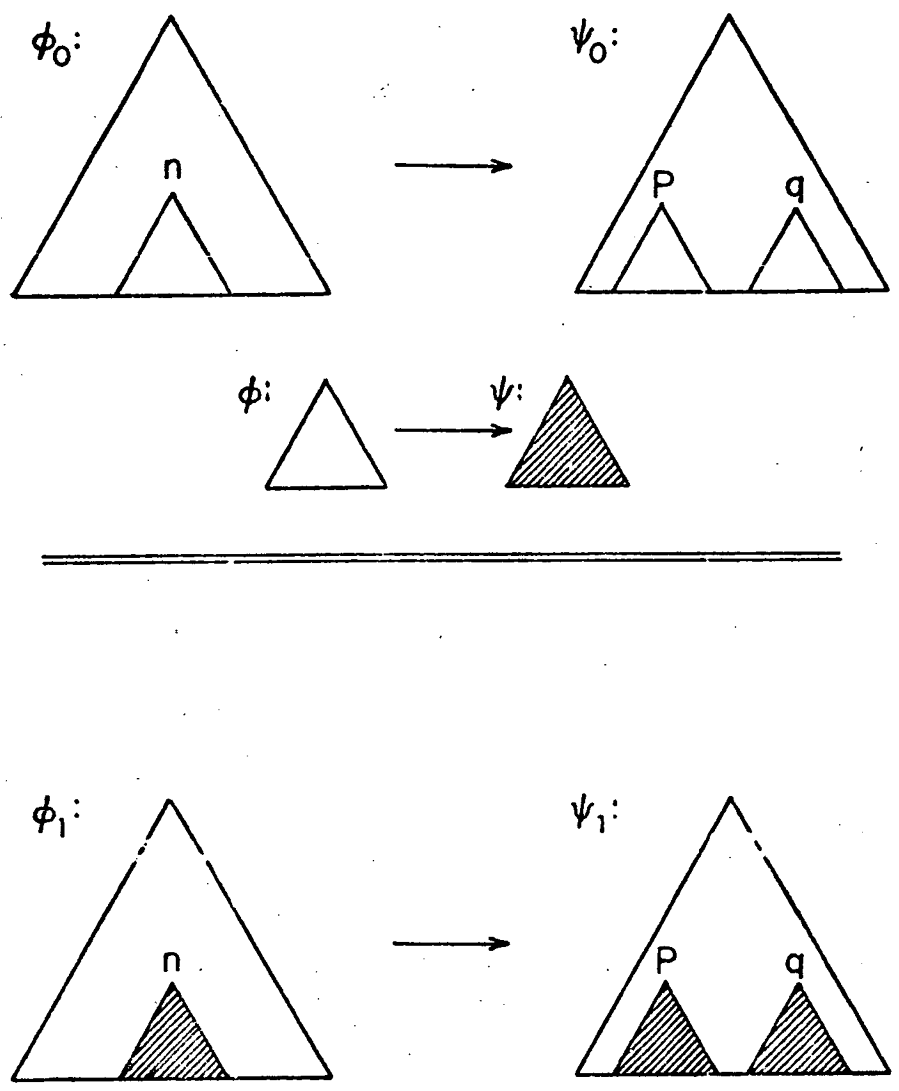


Figure 1-2. The rule $\varphi \longrightarrow \psi$ is applied at n in φ_0 and at p and q in ψ_0 to form another rule $\varphi_1 \longrightarrow \psi_1$.

where the ... is built up from x and constants and previously defined functions. This is an explicit definition and its meaning is clear. It is not so clear how the factorial is defined by

(1) $f(x) := \underset{\sim}{\text{if}}\ x = 0\ \underset{\sim}{\text{then}}\ 1\ \underset{\sim}{\text{else}}\ x \times f(x-1),$

since the name of the function we are defining reappears on the right. In general we could write a system of equations

$$\begin{aligned} f_1(x) &:= e_1 \\ f_2(x) &:= e_2 \\ &\ \vdots \end{aligned}$$

that looks very much like an explicit definition except that some of the $f_1, f_2, \ldots$ may reappear (or recur) in some of the $e_1, e_2, \ldots$. Perhaps f_1 does not itself recur in e_1, but if f_1 is in e_2 and f_2 is in e_1 then the definition is still not explicit and still in need of further explanation. Such systems of equations are called recursive definitions.

McCarthy [34, p. 42] explained such definitions in terms of a nondeterministic algorithm for calculating values of recursively defined functions. For example, one could evaluate conditionals, multiply numbers, appeal to (1), and so on to compute f(1) as follows:

$$\begin{aligned} f(1) &\Longrightarrow \underset{\sim}{\text{if}}\ 1 = 0\ \underset{\sim}{\text{then}}\ 1\ \underset{\sim}{\text{else}}\ 1 \times f(1-1) \\ &\Longrightarrow \underset{\sim}{\text{if}}\ \underset{\sim}{\text{false}}\ \underset{\sim}{\text{then}}\ 1\ \underset{\sim}{\text{else}}\ 1 \times f(1-1) \\ &\Longrightarrow 1 \times f(1-1) \\ &\Longrightarrow 1 \times f(0) \\ &\Longrightarrow 1 \times (\underset{\sim}{\text{if}}\ 0 = 0\ \underset{\sim}{\text{then}}\ 1\ \underset{\sim}{\text{else}}\ 1 \times f(0-1)) \\ &\Longrightarrow 1 \times (\underset{\sim}{\text{if}}\ \underset{\sim}{\text{true}}\ \underset{\sim}{\text{then}}\ 1\ \underset{\sim}{\text{else}}\ 1 \times f(0-1)) \\ &\Longrightarrow 1 \times 1 \\ &\Longrightarrow 1. \end{aligned}$$

According to this algorithmic explanation, the function defined by (1) is the set of all ordered pairs $\langle \xi, \eta \rangle$ of numbers such that $f(\xi)$ can be evaluated to η by some sequence of applications of the formal rules specified by (1) and by the computations of the given functions. In general (although not in this particular example) this is only a partial function: for some values of ξ there may be no formal computation for $f(\xi)$ that terminates at a numeral.

Recursive definitions may also be explained semantically by considering them as implicit definitions. Just as

(2) $$Y^2 + Y - 6 = 0$$

implicitly defines the set of numbers $\{2, -3\}$, any one of which will make (2) a true statement if substituted for Y, the recursive definition (1) implicitly defines a set of partial functions, any one of which will make (1) a true statement (for all relevant values of x). In this example there is just one solution: the factorial function $\{\langle \xi, \eta \rangle \mid \xi! = \eta\}$.

In Chapter 3 we extend the McCarthy calculus by allowing a choice between two classes of variables for the x's in definitions like (1). The choice will be the same as the choice between call-by-value and call-by-name in ALGOL 60 [38, §4.7.3]. McCarthy's algorithmic explanation is formalized by subtree replacement systems and it is shown that recursive definitions specify singlevalued partial functions despite the nondeterminism of the evaluation algorithm. (As with any nondeterministic algorithm, it is still possible that one attempt to compute $f(\xi)$ will succeed while another goes into an infinite computation.)

A recursive definition defines one partial function under the algorithmic explanation and a set of partial functions that solve its equations under the semantic explanation. Are there any solutions? Is the algorithmically defined function one of the solutions? Which one? Morris [37, Chap. 3, Thm. 2] conjectured that the algorithm specifies the minimal solution: the partial function which solves the equations and is extended by every other solution. (Strictly speaking, we must introduce a new datum ∞ and work with total functions, but the preceding approximation to the conjecture is accurate enough for this preliminary sketch.) In Chapter 3 we verify this conjecture. When restricted to definitions that call all their parameters by value, our Validity Theorem (8.4) is closely related to the "first recursion theorem" [26, §66, Thm. 66]: any partial recursive functional $\mathcal{F}$ has a unique minimal fixed point defined by formal calculations using the set of equations that specifies $\mathcal{F}$.

Chapter 4 applies the theory to the full lambda calculus, including eta and delta rules as defined by Curry and Feys [14, Chap. 3]. We add a new operator symbol γ to represent application of one lambda expression to another, so that a lambda expression such as $\lambda x.x(yz)$ corresponds to the tree $\overline{\lambda}(\overline{x}, \overline{\gamma}(\overline{x}, \overline{\gamma}(\overline{y}, \overline{z})))$, wherein the root is labelled by λ, the first son of the root is labelled by x, the second son of the root is labelled by γ, and so on. The beta, eta, and delta rules are expressed as rules in a subtree replacement system.

The classical Church-Rosser theorem [14, Chap. 4] asserts that the lambda calculus is Church-Rosser _modulo_ a certain equivalence relation: if S and S′ can both be derived from a lambda expression R, then there are equivalent lambda expressions T and T′ such that T can be derived from S and T′ can be derived from S′. We must be content with $T \neq T'$ in many examples. The equivalence relation involved is essentially the same as the relation between $\int_0^\pi \cos(x+y)\,dy$ and $\int_0^\pi \cos(x+z)\,dz$: equivalent expressions are the same except for the arbitrary names of dummy variables. This relation is usually defined by saying that R is equivalent to S if R can be transformed to S by a sequence of "alpha conversions," but this simple definition is quite awkward in actual use. In Chapter 4 we introduce a new definition that permits a higher degree of mathematical rigor than the usual one. (In Appendix B we prove that our equivalence relation is indeed the same as the one used in [14].) For the sake of simplicity in completing this introductory sketch, we will ignore the difference between equivalence and equality of lambda expressions. The actual technical exposition in Chapter 4 will be extremely careful about this distinction.

The main result of Chapter 4 is a relatively simple proof of the classical Church-Rosser theorem using a divide-and-conquer strategy. We consider the whole lambda calculus as a "union" of two parts: one defined by beta rules and the other by eta and delta rules. The beta part is divided into two stages by new sets of rules called "gamma" and "sigma." Gamma rules detect subtrees of the form $\overline{\gamma}(\overline{\lambda}(\overline{x}, S), R)$, which correspond to "beta redexes" $(\lambda x.S)R$ in ordinary notation, to which beta rules might be applied. Rather than replace such a subtree by the result $[R/x]S$ of applying a beta rule, a gamma rule replaces $\overline{\gamma}(\overline{\lambda}(\overline{x}, S), R)$ by $\overline{\sigma}(R, \overline{x}, S)$, where σ is a new symbol introduced to mark places where beta substitutions have been "requested." The sigma rules "perform" the substitutions that gamma rules request, deriving $[R/x]S$ from $\overline{\sigma}(R, \overline{x}, S)$. Using results from Chapter 2 and various elementary properties of the lambda calculus, we show that gamma and sigma rules define Church-Rosser subtree replacement systems and that these systems can interact to simulate beta rules. The net result is that the beta part of the lambda calculus is Church-Rosser. By a direct application of the fact that unequivocal closed subtree replacement systems are Church-Rosser, we also show that the eta-delta part is Church-Rosser. To complete the proof we then apply results from Chapter 2 to the effect that appropriately connected Church-Rosser parts form Church-Rosser wholes.

Chapter 5 applies the theory of subtree replacement systems to finite tree transducers and describes the use of these transducers in compiler design. We analyze syntax-directed compilation as a sequence of five processes (which would be implemented as coroutines in practice):

(1) Lexical analysis transduces the source program (a string of characters) to a string of terminal symbols from a context-free grammar.

(2) Context-free parsing assigns a "ranked parse tree" to the string of terminal symbols; this tree displays the way the grammar generates the string.

(3) Lexical filtration verifies that the character strings in the program corresponding to terminal symbols in the ranked parse tree do not violate restrictions such as the ALGOL 60 prohibition of transfers into a block.

(4) Semantic analysis transduces the ranked parse tree, whose nodes are labelled by context-free productions, to a "coding tree" whose

nodes are labelled by code-building operations: operations on machine code that can readily be programmed in assembly language or other languages appropriate for compiler writing.

(5) Code generation performs the operations specified by the coding tree.

There are helpful mathematical models for the difficult aspects of lexical analysis, context-free parsing, and semantic analysis. These three processes provide the lexical filter and the code generator with such explicitly structured data that lexical filtration and code generation are rather straightforward.

Finite tree transducers provide a model for semantic analysis and are discussed in detail. These devices are generalizations of the well-known finite transducers (often called "generalized sequential machines with final states" or "deterministic a-transducers") which provide a model for lexical analysis. To see how the two classes of abstract device are related, we consider string transducers as restricted Turing machines. A finite string transducer has a single tape on which the input is originally written. The tape head begins at the left end of the input and can move only to the right. Whenever the head reads an input symbol, it erases that symbol and writes out a short string of output symbols (creating some new tape squares if necessary) before changing control state and moving on to the next input symbol. In a top-down finite tree transducer the tape head begins at the root and moves downward toward the leaves. (Bottom-up transducers are also considered in Chapter 5 but will be ignored in this preliminary sketch.) As with finite string transducers, there are just finitely many possible control states associated with the tape head, and only finitely many responses to each (state, input symbol) combination are possible.

Whenever the tape head reads an input symbol, it erases that symbol and writes out a small portion of the output tree before changing state and moving down to the next input symbol. Here there is a complication unknown in the case of string transducers: if the current input symbol is on a node with more than one son, then there are more than one "next" input symbols. In order to use one-way tape motion without arbitrarily discarding all but one son, we allow the tape head to _split_ into several independent heads, each with its own control state, so that for each "next" input symbol there is a tape head that moves down to it. Although such behavior is extremely awkward to formalize in the usual style of automata theory (as in [24]), it can be obtained easily from a subtree replacement system that uses additional nodes labelled by states to record the positions and states of tape heads. A finite tree transducer is specified by a subtree replacement system under restrictions appropriate for expressing the way tape heads are to move.

Several important properties of finite tree transducers are considered in Chapter 5. A transducer is _partial deterministic_ if there is at most one response to each (state, input symbol) combination and is _total_ if there is at least one response to each such combination. Transducers that are both partial deterministic and total are _deterministic_. Since the tape heads are not synchronized, tree transduction is still a nondeterministic process with "deterministic" transducers, but we show that the nondeterminism is harmless. Specifically, partial determinism implies that the subtree replacement system is Church-Rosser, so that the output tree is determined by the input tree, while totality implies that there is at least one output for every input.

A transducer is linear if it never makes more than one copy of a portion of the input: whenever a tape head moves down to a son of a node, it does not split into more than one head for that son. Finite string transducers are trivially linear, and theorems on string transducers may require the addition of linearity to the hypothesis before they can be extended to tree transducers.

The transduction computed by a transducer is the correspondence it establishes between input and output trees. Chapter 5 concludes with a study of the question of whether the composition of two transductions of a certain type is also a transduction of that type. After discussing the practical significance of constructive proofs that classes of transductions are closed under composition, we construct the product transducer $\Pi_2 \wedge \Pi_1$ for any top-down finite tree transducers Π_1 and Π_2 such that the output vocabulary of Π_1 is the input vocabulary of Π_2. Elaborating slightly upon theorems of Rounds [46] and Thatcher [52], we note that partial determinism, totality, and linearity are inherited by $\Pi_2 \wedge \Pi_1$ if possessed by Π_1 and Π_2, and that

$$\begin{aligned}&(\text{Transduction computed by } \Pi_2 \wedge \Pi_1) = \\ &\qquad (\text{Transduction computed by } \Pi_2) \circ (\text{Transduction computed by } \Pi_1)\end{aligned}$$

if Π_1 and Π_2 are deterministic. Finally, we show that there is a pair of linear partial deterministic transductions whose composition is not computable by any top-down finite tree transducer, deterministic or nondeterministic.

2. Terminology

Standard mathematical notations are used as much as possible. We sometimes abbreviate "for all x in $\mathbb{N}$" by $(\forall x \in \mathbb{N})$, "there is a y less than z such that" by $(\exists y < z)$, and so on. The abbreviation $\exists !$ is used for "there is exactly one" or "there is a unique"; this symbol is less widely known than $\forall$ and $\exists$.

For any sets X and Y, $X \times Y$ is the set of all ordered pairs $\langle x, y \rangle$ such that $x \in X$ and $y \in Y$. A map or function $F : X \longrightarrow Y$ is any subset of $X \times Y$ such that, for each $x \in X$, there is at most one $y \in Y$ with $\langle x, y \rangle \in F$. The domain Dom F is $\{x \in X \mid (\exists y \in Y)(\langle x, y \rangle \in F)\}$. Thus

$$\text{Dom } F = \{x \in X \mid (\exists ! y \in Y)(\langle x, y \rangle \in F)\}$$

also. If Dom F = X then F is total; otherwise F is partial. For each $x \in \text{Dom } F$, the unique $y \in Y$ such that $\langle x, y \rangle \in F$ is denoted F(x) or Fx, depending on which is more readable in each context.

A total function $F : X \longrightarrow Y$ is injective iff

$$(\forall x, x' \in X)(Fx = Fx' \text{ implies } x = x')$$

and is surjective iff

$$(\forall y \in Y)(\exists X \in X)(Fx = y).$$

A map that is both injective and surjective is <u>bijective</u>. A bijective function is also said to be a <u>bijection</u>.

Given functions $F : X \longrightarrow Y$ and $G : Y \longrightarrow Z$, we set

$$G \circ F = \{\langle x, z\rangle \mid (\exists y \in Y)(\langle x, y\rangle \in F \;\&\; \langle y, z\rangle \in G)\},$$

so that $G \circ F : X \longrightarrow Z$ and

$$\mathrm{Dom}(G \circ F) = \{x \in \mathrm{Dom}\, F \mid Fx \in \mathrm{Dom}\, G\}.$$

The same equation may be written more succinctly as

$$\mathrm{Dom}(G \circ F) = F^{-1}(\mathrm{Dom}\, G)$$

where, for any subset B of Y,

$$F^{-1}(B) = \{x \in \mathrm{Dom}\, F \mid Fx \in B\}.$$

For any $y \in Y$ we also write

$$F^{-1}(y) = \{x \in \mathrm{Dom}\, F \mid Fx = y\}.$$

Any set $\Longrightarrow$ of ordered pairs is a <u>relation</u>. We write $x \Longrightarrow y$ rather than $\langle x, y\rangle \in \Longrightarrow$, but a symbol like $\Longrightarrow$ is still being used as the name of a set. Equations like

$$(\Longrightarrow) = (\underset{1}{\Longrightarrow} \cup \underset{2}{\Longrightarrow})$$

are well-formed and mean exactly what they say about sets of ordered pairs. Round brackets are used liberally to make such equations readable. When $(\Longrightarrow) \subseteq \mathbb{B} \times \mathbb{B}$ we say that $\Longrightarrow$ is a relation <u>on</u> $\mathbb{B}$.

The <u>composite</u> of relations $\underset{1}{\Longrightarrow}$ and $\underset{2}{\Longrightarrow}$ is defined by

$$(\underset{1}{\Longrightarrow}\;\underset{2}{\Longrightarrow}) = \{\langle x, z\rangle \mid (\exists y)(x \underset{1}{\Longrightarrow} y \;\&\; y \underset{2}{\Longrightarrow} z)\}.$$

If F and G are functions then they are also relations and we have

$$(FG) = G \circ F .$$

This reversal is unfortunate but unavoidable so long as we apply functions from the left (as is standard) rather than from the right (as would be more elegant). The best way to have composition operations for both functions and relations is to retain the raised circle $\circ$ for functional composition and pronounce it as "after." Juxtaposition with no $\circ$ always means relational composition rather than functional composition.

The set $\{0, 1, 2, \ldots\}$ of <u>nonnegative</u> integers is denoted $\mathbb{N}$. The letters i, j, k, J, K are often used as variables ranging over $\mathbb{N}$.

The set of all subsets of a set X is denoted 2^X. The set of all subsets of $V_\#$ is then typed as $2^{V}\#$, but this notation <u>is</u> standard.

CHAPTER 2

THEORY OF GENERAL AND SUBTREE REPLACEMENT SYSTEMS

This chapter presents the abstract theory of general replacement systems (§3) and subtree replacement systems (§5). The preliminary results in §3 apply to any situation where changes in the data are made in discrete steps, whether or not the data are trees. We define the Church-Rosser property formally and establish conditions under which the Church-Rosser property for a complex system can be derived from properties of its parts and their interconnections.

Slight extensions of Brainerd's arithmetic of trees [6, §2] are described in §4. We have added many mnemonics and have extended some of the operations from single nodes to certain sets of nodes. Using this formal computational technique in §5, we establish sufficient conditions for the Church-Rosser property in subtree replacement systems. Our main theorem asserts that every "unequivocal" and "closed" subtree replacement system is Church-Rosser. In later chapters it will be fairly easy but sometimes rather tedious to verify that several important systems are unequivocal and closed. In order to avoid burdensome repetition in these verifications, we present part of the argument abstractly in §6.

3. General Replacement Systems

Let $\mathbb{B}$ be a set of "objects" and let there be some means of "replacing" one object by another. We write $R \Longrightarrow S$ and pronounce this as "R can become S." For example, $\mathbb{B}$ might be the set of all possible arithmetic expressions, and $R \Longrightarrow S$ might mean that the expression R can be transformed to the expression S by performing one of the innermost indicated operations. In this and in many other examples the replacement relation $\Longrightarrow$ is not singlevalued, yet the net result of a complete computation should sometimes depend only on the starting configuration, not the specific path chosen. The expression $(14 + 32) \times (93 + 8)$ can be reduced to a single numeral by two additions and then one multiplication. It makes no difference which addition is performed first, since there are no side effects or round-off errors.

If R_0 can become R_1, R_1 can become $R_2, \ldots, R_{K-1}$ can become R_K, let us say that R_0 "can evolve" to R_K. More precisely, the evolution relation $\overset{*}{\Longrightarrow}$ is the reflexive transitive closure of the becoming relation $\Longrightarrow$. When R_0 can evolve to R_K and R_K cannot become anything new, then R_K is to be considered the "value" or "meaning" of R_0. To avoid unwanted connotations, we use the term "normal form" instead. Systems in which normal forms are unique are appropriate for evaluation or translation processes.

(3.1) Definition. Let $\mathbb{B}$ be any set, $\Longrightarrow$ be any binary relation on $\mathbb{B}$. Then $\mathfrak{B} = (\mathbb{B}, \Longrightarrow)$ is a <u>general replacement system</u> (<u>GRS</u>). A member T of $\mathbb{B}$ is <u>irreplaceable</u> iff

(1) $$(\forall S \in \mathbb{B})(T \Longrightarrow S \text{ implies } T = S).$$

Let $R, T \in \mathbb{B}$. Then T is a <u>normal form</u> for R iff

(2) $$R \overset{*}{\Longrightarrow} T \;\&\; T \text{ is irreplaceable}$$

where $\overset{*}{\Longrightarrow}$ is the reflexive transitive closure of $\Longrightarrow$.

When are normal forms unique? The condition that $\Longrightarrow$ itself be a function is much too restrictive. Why not beg the question by <u>adding</u> a sequencing mechanism $\mathfrak{M}$ to a GRS? The mechanism $\mathfrak{M}$ will look at R and the various S, T, . . . that R <u>can</u> become, then determine which one R <u>does</u> become, perhaps after consulting some sort of record of past steps. Suppose one has a mechanism $\mathfrak{M}$ and then is given a very different and complex mechanism $\mathfrak{M}'$ that purports to find the same normal forms more cheaply. Whether we try to verify this claim by a formal proof or by testing it on many "well-chosen" examples, it will help if there is an invariant core of knowledge about the normal forms. More precisely, consider three ways that $\mathfrak{M}'$ might err in seeking the normal form defined by $\mathfrak{M}$ for an object R:

(a) $\mathfrak{M}'$ finds a normal form S′ for R although $\mathfrak{M}$ finds no normal form for R.

(b) $\mathfrak{M}'$ finds a normal form S′ for R although $\mathfrak{M}$ finds a normal form S with $S \neq S'$.

(c) $\mathfrak{M}'$ finds no normal form for R although $\mathfrak{M}$ finds a normal form S.

Error (a) is more properly considered an improvement: $\mathfrak{M}'$ gives output where the original mechanism would enter an endless computation. Error (b) cannot occur if normal forms are unique in the underlying GRS. Uniqueness of normal forms provides no assurance that error (c) does not occur, but the insights gained while proving uniqueness should assist in the analysis of this problem. We therefore seek abstract sufficient conditions for uniqueness of normal forms that do not depend on how the system is implemented by a sequencing mechanism.

(3.2) <u>Definition</u>. A GRS $\mathfrak{B} = (\mathbb{B}, \Longrightarrow)$ is <u>Church-Rosser</u> iff

$$(\forall R, S, S' \in \mathbb{B})[(R \overset{*}{\Longrightarrow} S \;\&\; R \overset{*}{\Longrightarrow} S') \text{ implies } (\exists T \in \mathbb{B})(S \overset{*}{\Longrightarrow} T \;\&\; S' \overset{*}{\Longrightarrow} T)].$$

Normal forms are unique in Church-Rosser systems. The name comes from the work of Church and Rosser [13] on this property for a particular GRS. There are several easy but useful theorems telling how to infer the Church-Rosser property for a complex system from various properties of its parts.

(3.3) <u>Definition</u>. Let $\mathfrak{B}_1 = (\mathbb{B}, \underset{1}{\Longrightarrow})$ and $\mathfrak{B}_2 = (\mathbb{B}, \underset{2}{\Longrightarrow})$ be GRSs. Then $\mathfrak{B}_1$ <u>commutes</u> with $\mathfrak{B}_2$ iff

$$(\forall R, S_1, S_2 \in \mathbb{B})[(R \underset{1}{\overset{*}{\Longrightarrow}} S_1 \;\&\; R \underset{2}{\overset{*}{\Longrightarrow}} S_2) \text{ implies } (\exists T \in \mathbb{B})(S_1 \underset{2}{\overset{*}{\Longrightarrow}} T \;\&\; S_2 \underset{1}{\overset{*}{\Longrightarrow}} T)].$$

It is convenient to diagram the Church-Rosser property and commutativity, as in Figure 2-1. The variables in the universal quantifier $(\forall R, S, S' \in \mathbb{B})$ are represented by filled circles and the variable in the existential quantifier $(\exists T \in \mathbb{B})$ is represented by an open circle. The pairs in the relation $\overset{*}{\Longrightarrow}$ are represented by arrows joining some of the circles.

In the right half of Figure 2-1 we add labels to the arrows so as to indicate which of several relations is being asserted. In general, a diagram asserts that, whenever given objects stand in the relations indicated by the filled circles and the arrows joining them, then further objects indicated by the open circles exist, so as to make the relations indicated by the other arrows true also. As in the "diagram-chasing"

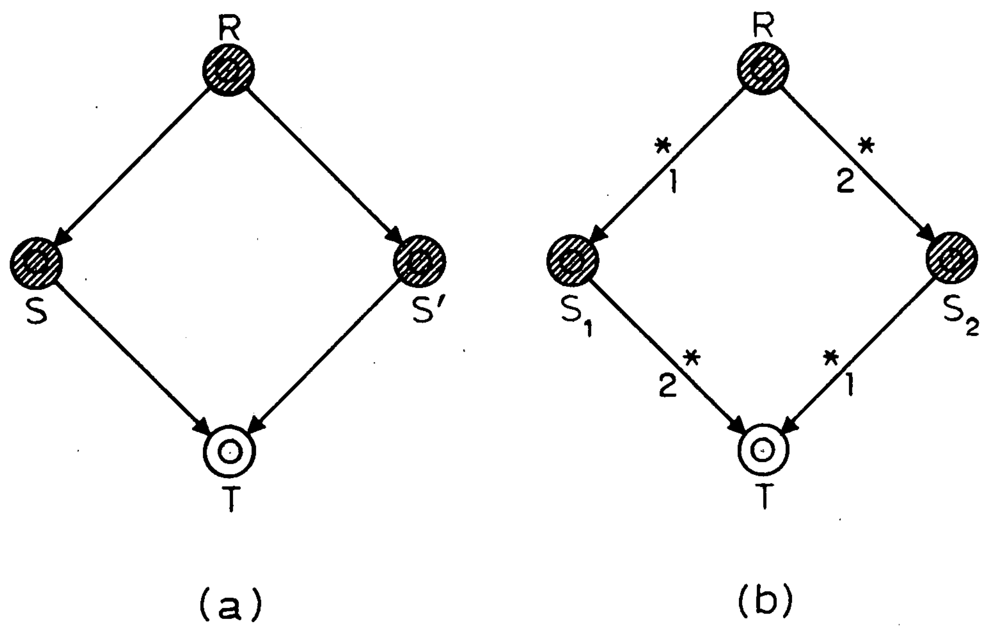


Figure 2-1. Diagrams for the Church-Rosser property (a) and commutativity (b).

arguments in algebraic topology, manipulations of this two-dimensional symbolism are sometimes clearer than the corresponding manipulations of the usual one-dimensional symbolism of mathematics.

(3.4) <u>Lemma</u>. Let $\mathfrak{B} = (\mathbb{B}, \Longrightarrow)$ be a GRS. If there exists a binary relation $\underset{\perp}{\Longrightarrow}$ on $\mathbb{B}$ such that

(1) $$(\overset{*}{\underset{\perp}{\Longrightarrow}}) = (\overset{*}{\Longrightarrow})$$

(2) $$(\forall R, S, S' \in \mathbb{B})[(R \underset{\perp}{\Longrightarrow} S \;\&\; R \underset{\perp}{\Longrightarrow} S') \text{ implies } (\exists T \in \mathbb{B})(S \underset{\perp}{\Longrightarrow} T \;\&\; S' \underset{\perp}{\Longrightarrow} T)],$$

then $\mathfrak{B}$ is Church-Rosser. ∎

(3.5) <u>Commutative Union Theorem</u>. Let $\{\mathfrak{B}_a \mid a \in A\}$ be a family of Church-Rosser GRSs with $\mathfrak{B}_a = (\mathbb{B}, \underset{a}{\Longrightarrow})$ for each $a \in A$. Let $\mathfrak{B}_a$ commute with $\mathfrak{B}_b$ for all $a, b \in A$ with $a \neq b$. Then the <u>union</u> $\mathfrak{B} = (\mathbb{B}, \Longrightarrow)$ is Church-Rosser, where

$$(\mathbb{B}, \Longrightarrow) = (\mathbb{B}, \bigcup_{a \in A} \underset{a}{\Longrightarrow}).$$

<u>Proof:</u> We will use Lemma 3.4. Let $\underset{\perp}{\Longrightarrow}$ be $\bigcup_{a \in A} \overset{*}{\underset{a}{\Longrightarrow}}$, so that (3.4.1) is trivial. To prove (3.4.2), we assume $(R \underset{\perp}{\Longrightarrow} S \;\&\; R \underset{\perp}{\Longrightarrow} S')$ and show that some $T \in \mathbb{B}$ has $(S \underset{\perp}{\Longrightarrow} T \;\&\; S \underset{\perp}{\Longrightarrow} T)$. For some $a, b \in A$ we have $R \overset{*}{\underset{a}{\Longrightarrow}} S$ and $R \overset{*}{\underset{b}{\Longrightarrow}} S'$. By the Church-Rosser property if $a = b$ or by commutativity if $a \neq b$, some $T \in \mathbb{B}$ has $(S \overset{*}{\underset{b}{\Longrightarrow}} T \;\&\; S' \overset{*}{\underset{a}{\Longrightarrow}} T)$. Therefore $(S \underset{\perp}{\Longrightarrow} T \;\&\; S' \underset{\perp}{\Longrightarrow} T)$, as wanted. ∎

The usefulness of this theorem is enhanced by the following sufficient condition for commutativity, which will be proven by diagrammatic reasoning.

(3.6) <u>Commutativity Lemma</u>. Let $\mathfrak{B}_1 = (\mathbb{B}, \underset{1}{\Longrightarrow})$ and $\mathfrak{B}_2 = (\mathbb{B}, \underset{2}{\Longrightarrow})$ be GRSs. Let $\overset{=}{\underset{1}{\Longrightarrow}}$ be the reflexive closure of $\underset{1}{\Longrightarrow}$. If

(1) $$(\forall R, S_1, S_2 \in \mathbb{B})[(R \underset{1}{\Longrightarrow} S_1 \;\&\; R \underset{2}{\Longrightarrow} S_2) \text{ implies } (\exists T \in \mathbb{B})(S_1 \overset{*}{\underset{2}{\Longrightarrow}} T \;\&\; S_2 \overset{=}{\underset{1}{\Longrightarrow}} T)],$$

then $\mathfrak{B}_1$ commutes with $\mathfrak{B}_2$.

<u>Proof:</u> We assume (1) and prove commutativity by two inductions. For each $K \in \mathbb{N}$ we prove that

(2) $$(\forall R, S_1, S_2 \in \mathbb{B})[(R \underset{1}{\Longrightarrow} S_1 \;\&\; R \overset{K}{\underset{2}{\Longrightarrow}} S_2) \text{ implies } (\exists T \in \mathbb{B})(S_1 \overset{*}{\underset{2}{\Longrightarrow}} T \;\&\; S_2 \overset{=}{\underset{1}{\Longrightarrow}} T)].$$

When $K = 0$ we let T be S_1. To pass from K to K+1, suppose (2) holds for K and that $R, S_1 S_2 \in \mathbb{B}$ have $R \underset{1}{\Longrightarrow} S_1$ and $R \overset{K+1}{\underset{2}{\Longrightarrow}} S_2$. Let $P \in \mathbb{B}$ with $R \overset{K}{\underset{2}{\Longrightarrow}} P \underset{2}{\Longrightarrow} S_2$. This situation is diagrammed by the filled circles and arrows between them at the top of Figure 2-2. We call this the <u>working diagram</u>. The procedure for adding to this diagram (with the help of whatever we are assuming or have already proven) will now be explained.

The induction hypothesis is expressed as diagram $\boxed{a}$ in Figure 2-2. (For obvious typographical reasons, we will write (Da) rather than $\boxed{a}$.) Suppose (Da) is cut on a mimeograph <u>stencil</u> and lay this stencil over the working diagram so that each filled circle of (Da) is over a circle in the working diagram and each arrow joining filled circles in (Da) is over a similarly labelled and directed arrow in the working diagram. (The circles R, S_1, P in the working diagram are used.) Now <u>print</u> <u>through</u> the stencil, forming the circle Q and the

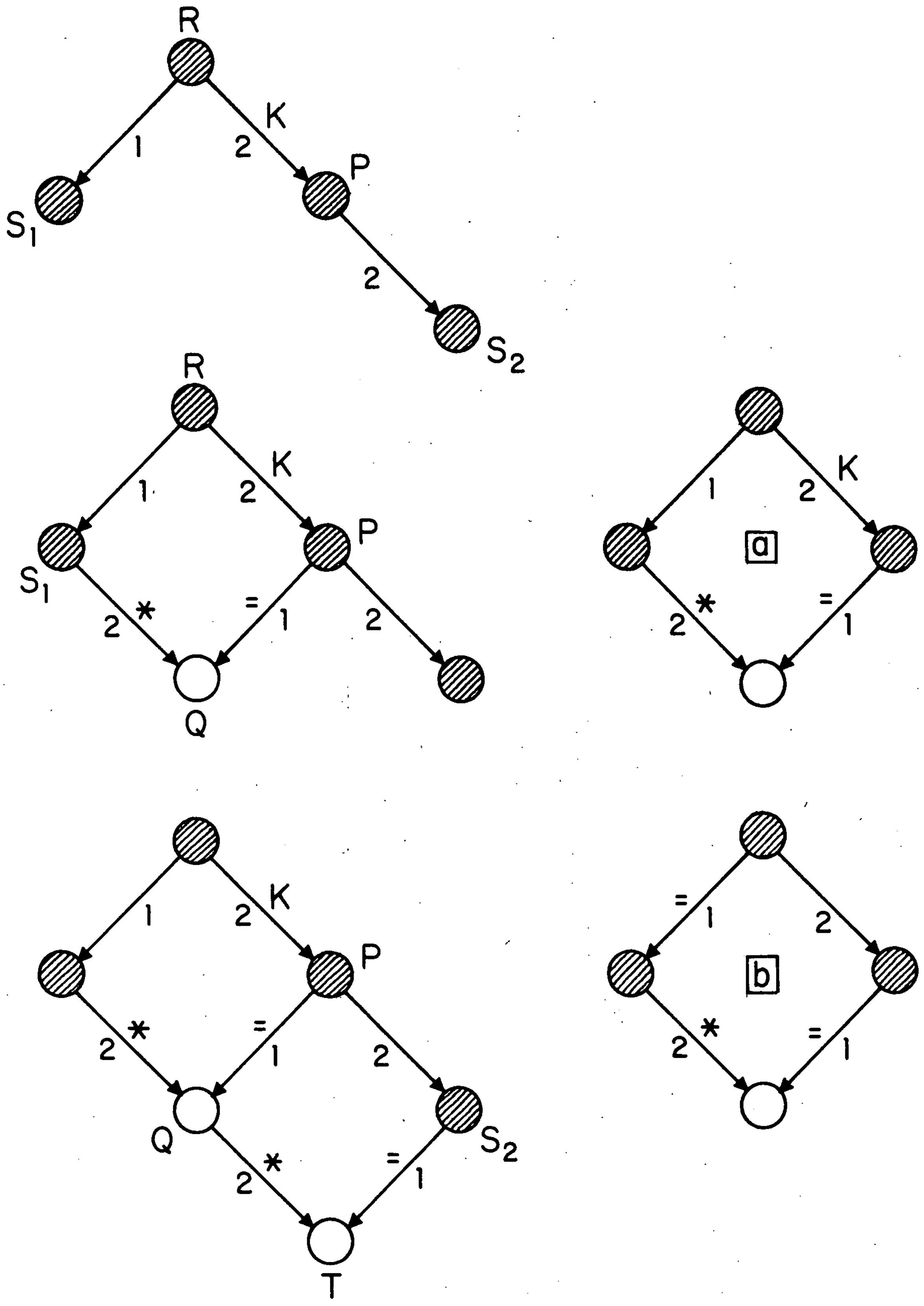


Figure 2-2. Adding to a diagram with the help of two stencils.

arrows from S_1 to Q and from P to Q. This step corresponds to the valid inference

$$\begin{array}{l}
R \xRightarrow[1]{} S_1 \ \& \ R \xRightarrow[2]{K} P \ \& \ P \xRightarrow[2]{} S_2 \\
(\forall\, r, s_1, p \in \mathbb{B}) \, [(r \xRightarrow[1]{} s_1 \ \& \ r \xRightarrow[2]{K} p) \text{ implies} \\
\qquad\qquad (\exists\, Q \in \mathbb{B})(s_1 \xRightarrow[2]{*} Q \ \& \ p \xRightarrow[1]{=} Q)] \\
\hline
R \xRightarrow[1]{} S_1 \ \& \ R \xRightarrow[2]{K} P \ \& \ P \xRightarrow[2]{} S_2 \ \& \\
\qquad\qquad (\exists\, Q \in \mathbb{B})(S_1 \xRightarrow[2]{*} Q \ \& \ P \xRightarrow[1]{=} Q)].
\end{array}$$

The working diagram now consists of circles R, S_1, P, S_2, and Q, together with the arrows joining them.

Next we note that (1) implies the corresponding statement with $R \xRightarrow[1]{=} S_1$ in place of $R \xRightarrow[1]{} S_1$. Indeed, if $R = S_1$, we may simply let T be S_2. We express this trivial extension of (1) as (Db) in Figure 2-2. Consider (Db) as a stencil and lay it over the working diagram so that each filled circle in (Db) is over a circle in the working diagram and each arrow joining filled circles in (Db) is over a similarly labelled and directed arrow in the working diagram. (The circles P, Q, S_2 in the working diagram are used.) Now print through the stencil, forming the circle T and the arrows from Q to T and from S_2 to T. This corresponds to the valid inference

$$\begin{array}{l}
R \xRightarrow[1]{} S_1 \ \& \ R \xRightarrow[2]{K} P \ \& \ P \xRightarrow[2]{} S \ \& \\
\qquad\qquad (\exists\, Q \in \mathbb{B})(S_1 \xRightarrow[2]{*} Q \ \& \ P \xRightarrow[1]{=} Q) \\
(\forall\, p, q, s_2 \in \mathbb{B})[(p \xRightarrow[1]{=} q \ \& \ p \xRightarrow[2]{} s_2) \text{ implies} \\
\qquad\qquad (\exists\, T \in \mathbb{B})(q \xRightarrow[2]{*} T \ \& \ s_2 \xRightarrow[1]{=} T)] \\
\hline
(3) \quad R \xRightarrow[1]{} S_1 \ \& \ R \xRightarrow[2]{K} P \ \& \ P \xRightarrow[2]{} S_2 \ \& \\
\qquad (\exists Q, T \in \mathbb{B})(S_1 \xRightarrow[2]{*} Q \ \& \ P \xRightarrow[1]{=} Q \ \& \ Q \xRightarrow[2]{*} T \ \& \ S_2 \xRightarrow[1]{=} T).
\end{array}$$

In adding open circles and arrows to the working diagram, we have built up a diagram that says

$$(4)\quad (\forall R, S_1, S_2, P \in \mathbb{B})[(R \underset{1}{\Rightarrow} S_1 \;\&\; R \overset{K}{\underset{2}{\Rightarrow}} P \underset{2}{\Rightarrow} S_2) \text{ implies}$$
$$(\exists Q, T \in \mathbb{B})(S_1 \overset{*}{\underset{2}{\Rightarrow}} Q \overset{*}{\underset{2}{\Rightarrow}} T \;\&\; P \overset{=}{\underset{1}{\Rightarrow}} Q \;\&\; S_2 \overset{=}{\underset{1}{\Rightarrow}} T)].$$

Telescoping our two inferences, we can construct a deduction in first order logic of (3) from the premisses (Da), (Db), and

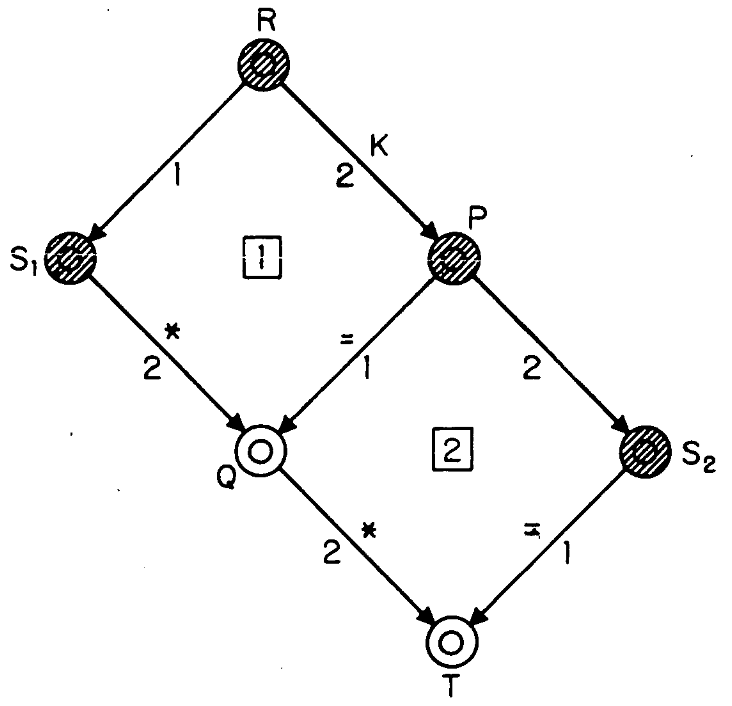
$$R \underset{1}{\Rightarrow} S_1 \;\&\; R \overset{K}{\underset{2}{\Rightarrow}} P \;\&\; P \underset{2}{\Rightarrow} S_2 .$$

Any such deduction can be mechanically transformed into a deduction of (4) from the premisses (Da) and (Db) alone. The diagrammatic manipulations in Figure 2-2 determine a proof of (4).

A more concise way to expound the diagrammatic proof of (4) is shown at the top of Figure 2-3. Here we display only the final state of the working diagram, together with numbers to indicate the order in which portions of the diagram appeared during the process of choosing stencils, laying them on the working diagram, and printing through. To <u>verify</u> a diagram like this is to check that it could indeed be built up from the filled circles and arrows between them alone, using only available stencils.

In situations where it may not be obvious which stencils were used or why they were true, we will add commentary in the text. Here, for example, we can say that (D1) was filled in because of the induction hypothesis and that then (D2) was filled in because of a trivial extension of (1).

Having deduced (4) from the induction hypothesis (2), we note that (4) implies a statement similar to (2) but with K+1 in place of K.



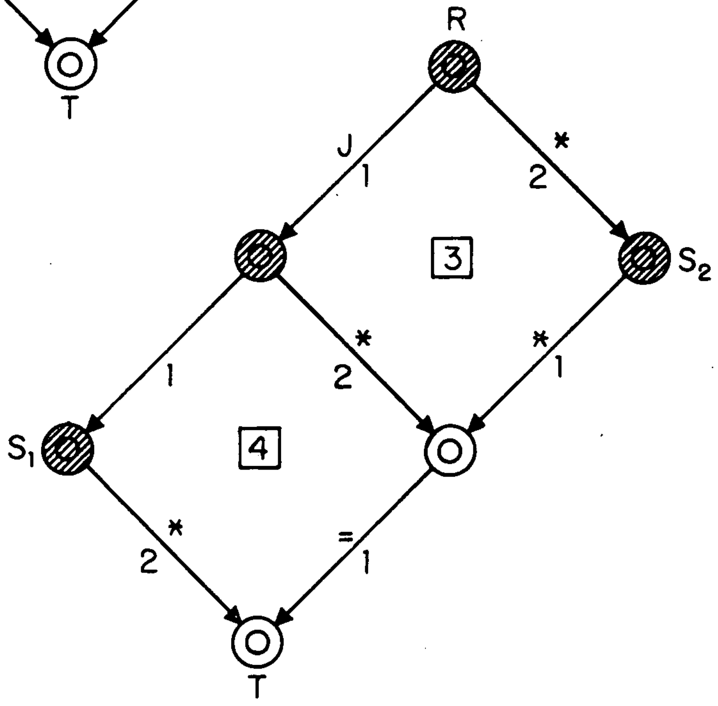


Figure 2-3. Induction steps in the proof of the Commutativity Lemma (3.6).

We have passed from K to K+1, and the inductive proof that every $K \in \mathbb{N}$ satisfies (2) is complete.

We now prove commutativity by proving that each $J \in \mathbb{N}$ satisfies

(5) $$(\forall R, S_1, S_2 \in \mathbb{B})\,[(R \overset{J}{\underset{1}{\Longrightarrow}} S_1 \;\&\; R \overset{*}{\underset{1}{\Longrightarrow}} S_2) \text{ implies } (\exists T \in \mathbb{B})(S_1 \overset{*}{\underset{2}{\Longrightarrow}} T \;\&\; S_2 \overset{*}{\underset{1}{\Longrightarrow}} T)].$$

When J = 0 we may let T be S_2. To pass from J to J+1 we verify the diagram at the bottom of Figure 2-3. First (D3) is filled in by the induction hypothesis and then (D4) is filled in by the fact that (2) holds for all $K \in \mathbb{N}$. ∎

Theorem 3.5 and Lemma 3.6 were discovered by Hindley [21, Chap. 1, Theorem 1.2, Lemma 1.3] and independently by the author.

There is a condition weaker than commutativity which implies that the union of two Church-Rosser GRSs is Church-Rosser.

(3.7) Definition. Let $\mathfrak{B}_1 = (\mathbb{B}, \underset{1}{\Longrightarrow})$ and $\mathfrak{B}_2 = (\mathbb{B}, \underset{2}{\Longrightarrow})$ be GRSs. Then $\mathfrak{B}_1$ requests $\mathfrak{B}_2$ iff

$$(\forall R, S_1, S_2 \in \mathbb{B})[(R \overset{*}{\underset{1}{\Longrightarrow}} S_1 \;\&\; R \overset{*}{\underset{2}{\Longrightarrow}} S_2) \text{ implies } (\exists T \in \mathbb{B})(S_1 \overset{*}{\underset{2}{\Longrightarrow}} T \;\&\; S_2 (\overset{*}{\underset{1}{\Longrightarrow}} \overset{*}{\underset{2}{\Longrightarrow}}) T)].$$

Figure 2-4 illustrates this definition.

(3.8) Theorem. Let $\mathfrak{B}_1 = (\mathbb{B}, \underset{1}{\Longrightarrow})$ and $\mathfrak{B}_2 = (\mathbb{B}, \underset{2}{\Longrightarrow})$ be Church-Rosser GRSs and let $\mathfrak{B}_1$ request $\mathfrak{B}_2$. Then the union $\mathfrak{B} = (\mathbb{B}, \Longrightarrow)$, where

$$(\mathbb{B}, \Longrightarrow) = (\mathbb{B}, \underset{1}{\Longrightarrow} \cup \underset{2}{\Longrightarrow}),$$

is Church-Rosser.

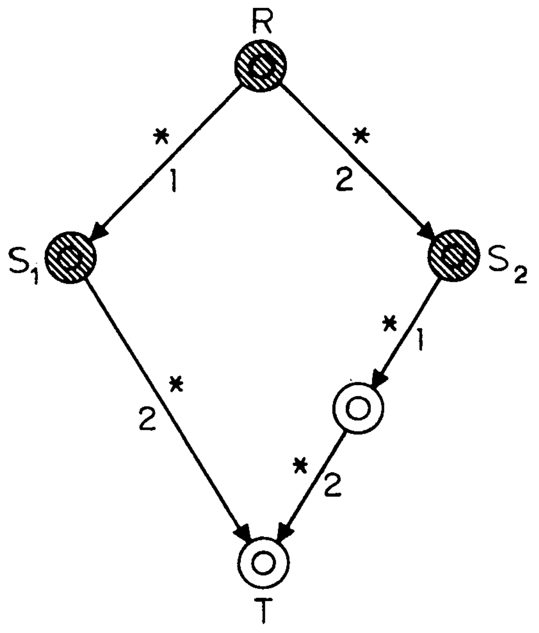


Figure 2-4. The GRS $\mathfrak{B}_1 = (\mathbb{B}, \xRightarrow[1]{})$ requests the GRS $\mathfrak{B}_2 = (\mathbb{B}, \xRightarrow[2]{})$.

Proof: We will use Lemma 3.4. Let $\underset{\perp}{\Longrightarrow}$ be the composite $(\overset{*}{\underset{1}{\Longrightarrow}}\ \overset{*}{\underset{2}{\Longrightarrow}})$. Then $(\overset{*}{\underset{\perp}{\Longrightarrow}}) = (\overset{*}{\Longrightarrow})$ because

$$(\Longrightarrow) \subseteq (\underset{\perp}{\Longrightarrow} \cup \underset{\perp}{\Longrightarrow}) = (\underset{\perp}{\Longrightarrow})\ \&\ (\underset{\perp}{\Longrightarrow}) \subseteq (\overset{*}{\Longrightarrow}\ \overset{*}{\Longrightarrow}) = (\overset{*}{\Longrightarrow}).$$

We must verify (3.4.2):

$$(\forall R, S, S' \in \mathbb{B})[(R \underset{\perp}{\Longrightarrow} S\ \&\ R \underset{\perp}{\Longrightarrow} S') \text{ implies}$$
$$(\exists T \in \mathbb{B})(S \underset{\perp}{\Longrightarrow} T\ \&\ S' \underset{\perp}{\Longrightarrow} T)].$$

We begin with the hypothesis $(R \overset{*}{\underset{1}{\Longrightarrow}} \overset{*}{\underset{2}{\Longrightarrow}} S\ \&\ R \overset{*}{\underset{1}{\Longrightarrow}} \overset{*}{\underset{2}{\Longrightarrow}} S')$ in Figure 2-5. We fill in (D1) because $\mathfrak{B}_1$ is Church-Rosser. We then fill in (D2) and (D3) because $\mathfrak{B}_1$ requests $\mathfrak{B}_2$. Finally, we fill in (D4) because $\mathfrak{B}_2$ is Church-Rosser. Since $\overset{*}{\underset{2}{\Longrightarrow}}$ is transitive, we do have $S \overset{*}{\underset{1}{\Longrightarrow}} \overset{*}{\underset{2}{\Longrightarrow}} T$ and $S' \overset{*}{\underset{1}{\Longrightarrow}} \overset{*}{\underset{2}{\Longrightarrow}} T$. ∎

Unfortunately, there seems to be no analog of Lemma 3.6 for "requests." Only the star in $R \overset{*}{\underset{2}{\Longrightarrow}} S_2$ can be removed.

(3.9) Lemma. Let $\mathfrak{B}_1 = (\mathbb{B}, \underset{1}{\Longrightarrow})$ and $\mathfrak{B}_2 = (\mathbb{B}, \underset{2}{\Longrightarrow})$ be GRSs, $\mathfrak{B}_2$ be Church-Rosser. If

(1) $$(\forall R, S_1, S_2 \in \mathbb{B})[(R \overset{*}{\underset{1}{\Longrightarrow}} S_1\ \&\ R \underset{2}{\Longrightarrow} S_2) \text{ implies}$$
$$(\exists T \in \mathbb{B})(S_1 \overset{*}{\underset{2}{\Longrightarrow}} T\ \&\ S_2(\overset{*}{\underset{1}{\Longrightarrow}}\ \overset{*}{\underset{2}{\Longrightarrow}})T)],$$

then $\mathfrak{B}_1$ requests $\mathfrak{B}_2$.

Proof: We assume (1) and prove that $\mathfrak{B}_1$ requests $\mathfrak{B}_2$ by proving that each $K \in \mathbb{N}$ satisfies

(2) $$(\forall R, S_1, S_2 \in \mathbb{B})[(R \overset{*}{\underset{1}{\Longrightarrow}} S_1\ \&\ R \overset{K}{\underset{2}{\Longrightarrow}} S_2) \text{ implies}$$
$$(\exists T \in \mathbb{B})(S_1 \overset{*}{\underset{2}{\Longrightarrow}} T\ \&\ S_2(\overset{*}{\underset{1}{\Longrightarrow}}\ \overset{*}{\underset{2}{\Longrightarrow}})T)].$$

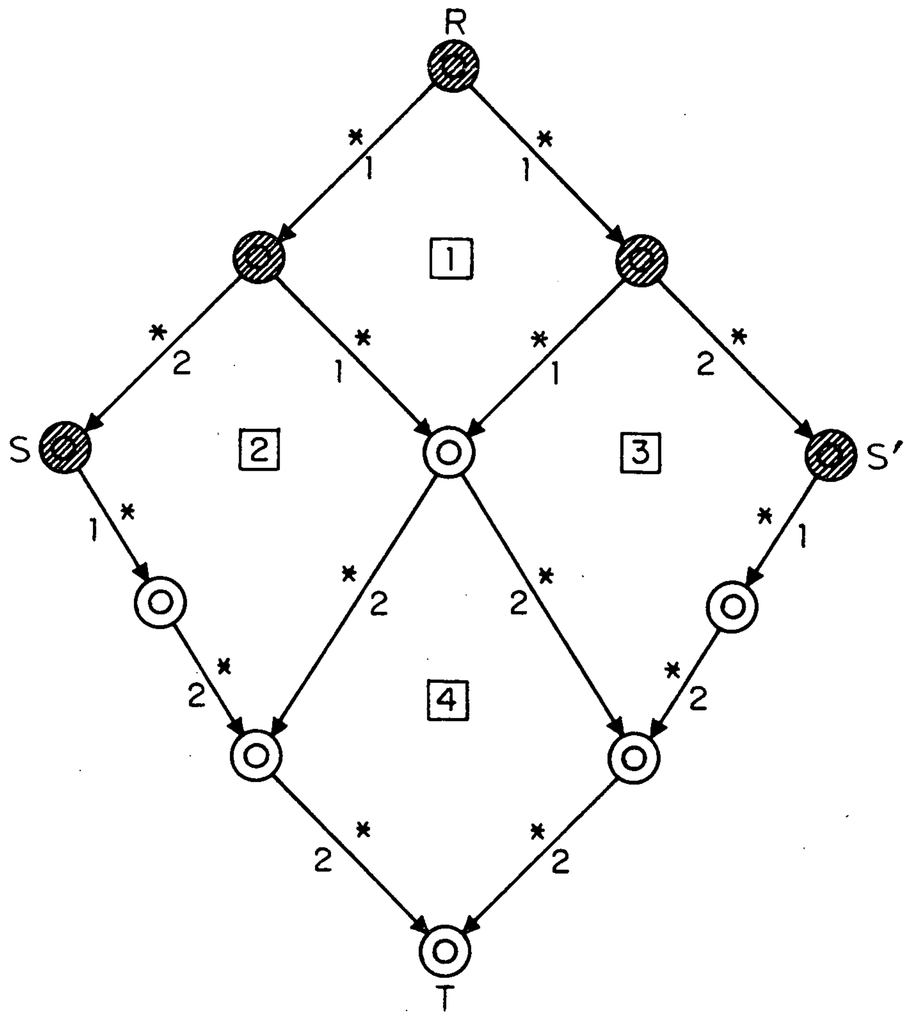


Figure 2-5. Diagram for the proof of Theorem 3.8.

We use induction on K. When K = 0 let T be S_1. We assume (2) holds for K and pass to K+1 by verifying Figure 2-6.

We begin with the hypothesis $(R \overset{*}{\underset{1}{\Longrightarrow}} S_1 \ \& \ R \overset{K+1}{\underset{2}{\Longrightarrow}} S_2)$ in Figure 2-6. We fill in (D1) by the induction hypothesis, (D2) by (1), and (D3) because $\mathfrak{B}_2$ is Church-Rosser. This proves

$$(\forall R, S_1, S_2 \in \mathbb{B})[(R \overset{*}{\underset{1}{\Longrightarrow}} S_1 \ \& \ R \overset{K+1}{\underset{2}{\Longrightarrow}} S_2) \text{ implies}$$
$$(\exists T \in \mathbb{B})(S_1 \overset{*}{\underset{2}{\Longrightarrow}} \overset{*}{\underset{2}{\Longrightarrow}} T \ \& \ S_2 \overset{*}{\underset{1}{\Longrightarrow}} \overset{*}{\underset{2}{\Longrightarrow}} \overset{*}{\underset{2}{\Longrightarrow}} T)],$$

which implies (2) for K+1 because $\overset{*}{\underset{2}{\Longrightarrow}}$ is transitive. ∎

4. Trees and Substitution

This section describes some elementary concepts in the mathematics of trees. It corresponds to arithmetic and basic algebra in numerical mathematics. The system of arithmetic and the basic algebra for trees sketched here are mostly taken from work of Brainerd [6, §2]. We have added convenient notations for string manipulation and many mnemonics. We have also extended some operations on nodes in trees to sets of "independent" nodes.

There are several common mathematical definitions for finite trees with labelled nodes and a left-to-right ordering. The one most appropriate here is often called "Dewey decimal" or "branch numbers." This formalism has also been useful in studying regular sets of trees [6] and syntactic complexity [40]. Strings of integers keep track of relations among a tree's nodes.

For any set A, the set of all strings of members of A is A^*. The members of A need not be "symbols" because strings are defined to be

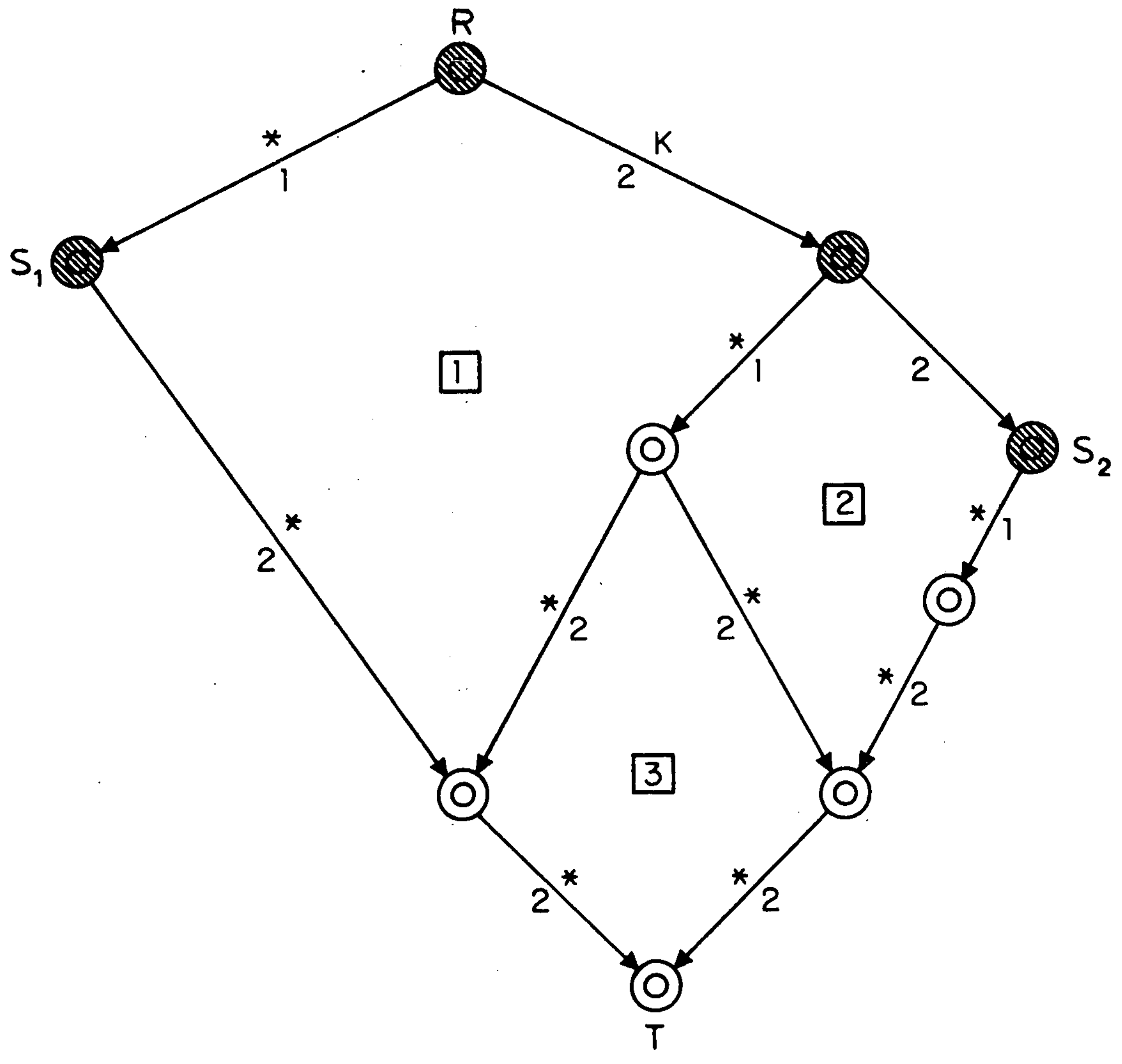


Figure 2-6. Induction step in the proof of Lemma 3.9.

finite sequences. In particular, $\mathbb{N}^*$ is the set of all strings of non-negative integers. We name strings in A^* by listing (within round brackets and separated by commas) the names for the members of A that occur in them. The null string is ( ); the string consisting of just thirty-seven is (37), and the string consisting of three followed by seven is (3, 7). This is already the systematic way to write argument strings for functions of several variables, and it avoids all ambiguity in naming strings of strings.

The length of a string w is denoted $|w|$. The set of all strings of length J in A^* is A^J. In particular, $A^0 = \{()\}$. For any string w and any $K \in \mathbb{N}$, $K{:}w$ is the first K entries in w, in the same order as in w. For $|w| \leq K$ this is w itself. The positions in w are numbered $0, 1, \ldots, |w| - 1$ and we write

$$w = (w_0, \ldots, w_{|w|-1}) = (w_0, \ldots, w_{-1}).$$

When the value of J is obvious or irrelevant, we write -1 rather than $J-1$, as here. The word "last" is a good pronunciation for -1 here.

Concatenation is indicated by a dot:

$$(x_0, \ldots, x_{-1}) \cdot (y_0, \ldots, y_{-1}) = (x_0, \ldots, x_{-1}, y_0, \ldots, y_{-1}).$$

This operation on strings extends to sets of strings in the obvious way. For $w \in A^*$; $X, Y \subseteq A^*$ we have

$$w \cdot X = \{w \cdot x \mid x \in X\} \quad \& \quad X \cdot Y = \{x \cdot y \mid x \in X \ \& \ y \in Y\}.$$

Now we define the father and left brother functions and the ancestor and independence relations on $\mathbb{N}^*$. (The reasons for these names will be clear shortly.)

(4.1) Definition. Let $m, n \in \mathbb{N}^*$. Define the following:

(1) $Fa(n) = (|n| - 1) : n$ (for $n \neq ()$) (father)

(2) $LBr(n) = Fa(n) \cdot (n_{-1} - 1)$ (for $n \neq ()$; $n_{-1} \neq 0$) (left brother)

(3) m anc n iff $|m| : n = m$ (ancestor)

(4) $m \perp n$ iff NOT(m anc n or n anc m) (independence)

(5) $m \underline{\perp} n$ iff ($m \perp n$ or $m = n$)

(4.2) Definition. A tree domain is any finite nonempty subset D of $\mathbb{N}^*$ closed under the father and left brother functions:

$$Fa[D] \subseteq D \quad \& \quad LBr[D] \subseteq D.$$

The way these definitions express the structure of a tree's nodes is illustrated in Figure 2-7. For any $n \in \mathbb{N}^*$ that appears as a node in this tree domain, $Fa(n)$ is indeed the father of n in the usual intuitive sense. Note that the sons of a node n in D are the members of the set $D \cap Fa^{-1}(n)$. From now on, "node" means "string of nonnegative integers."

Once a tree domain is given, only the assignment of labels to nodes is needed to specify a tree. (We are discussing labelled trees.)

(4.3) Definition. Let V be any set and let

$$V_* = \{R \mid R : \mathbb{N}^* \longrightarrow V \text{ (partial) } \& \text{ Dom } R \text{ is a tree domain}\}.$$

Members of V_* are called trees.

(4.4) Definition. Let $R, S \in V_*$; $n \in \text{Dom } R$.

(1) $R/n = \{\langle p, x\rangle \mid \langle n \cdot p, x\rangle \in R\}$ (subtree of R at n)

(2) $R(n \leftarrow S) = \{\langle m, x\rangle \mid \langle m, x\rangle \in R \ \& \ \text{NOT}(n \text{ anc } m)\}$
$\cup \{\langle n \cdot p, y\rangle \mid \langle p, y\rangle \in S\}$ (Replace n by S in R)

Figure 2-8 illustrates a tree R whose domain is the set of nodes shown in Figure 2-7. Now let us extend the idea of independence to *sets* of nodes.

(4.5) *Definition*. Let $M, N \subseteq \mathbb{N}^*$.

(1) $M \perp N$ iff $(\forall m \in M)(\forall n \in N)(m \perp n)$

(2) M is *independent* iff $(\forall m, n \in M)(m \perp n)$

The proofs of the following lemmas are trivial calculations.

(4.6) *Lemma*. Let $m, n \in \mathbb{N}^*$; $M, N \subseteq \mathbb{N}^*$.

(1) $m \ \underline{anc}\ n$ iff $(\exists p \in \mathbb{N}^*)(m \cdot p = n)$

(2) $M \cdot \mathbb{N}^* = \{n \in \mathbb{N}^* \mid (\exists m \in M)(m\ \underline{anc}\ n)\}$

(3) $(\underline{anc}\ \underline{anc}) \subseteq (\underline{anc})$ (transitivity)

(4) $M \perp N$ iff $M \cdot \mathbb{N}^* \cap N \cdot \mathbb{N}^* = \emptyset$

(4.7) *Lemma*. Let $R, S. T \in V_*$; $m, n \in \mathrm{Dom}\ R$; $p \in \mathrm{Dom}\ S$; $m \perp n$.

(1) $R(n \longleftarrow S)/(n \cdot p) = S/p$ (embedding)

(2) $R(n \longleftarrow S(p \longleftarrow T)) = R(n \longleftarrow S)(n \cdot p \longleftarrow T)$ (associativity)

(3) $R(n \longleftarrow S)/m = R/m$ (persistence)

(4) $R(n \longleftarrow S)(m \longleftarrow T) = R(m \longleftarrow T)(n \longleftarrow S)$ (commutativity)

(5) For $M \subseteq \mathrm{Dom}\ R$ an independent set, we may define

$$R(M \longleftarrow S) = R(m_0 \longleftarrow S) \ldots (m_{-1} \longleftarrow S)$$

(just R if $M = \emptyset$), where $(m_0, \ldots, m_{-1})$ is any list of the members of M; the order is irrelevant.

(6) Any set of *leaves* in Dom R is independent, where n is a leaf iff $\mathrm{Dom}\ R \cap Fa^{-1}(n) = \emptyset$.

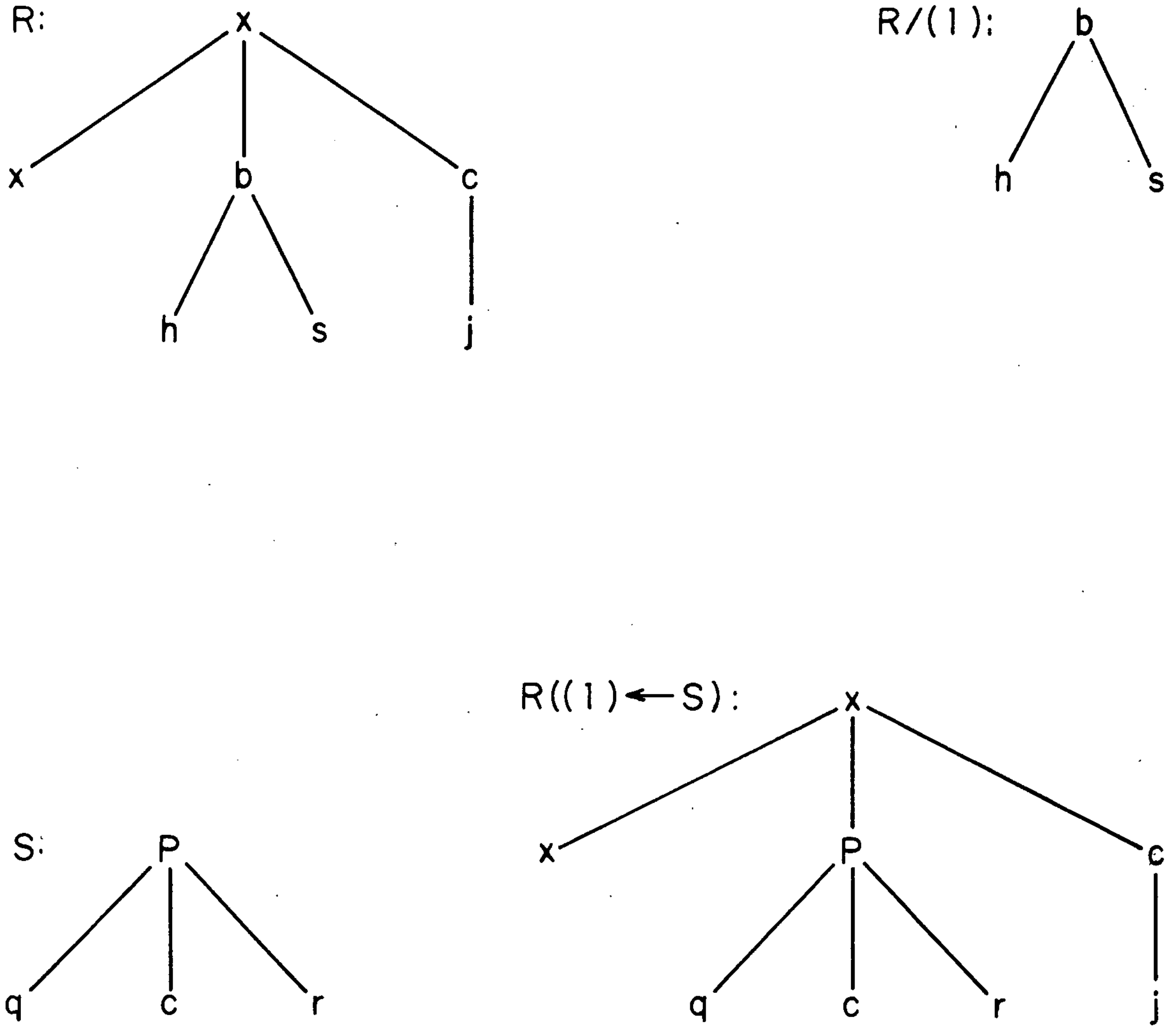


Figure 2-8. The subtree of R at (1) and the result of replacing it by S.

(4.8) Definition. Let $m, n, p \in \mathbb{N}^*$. The quotient of n by m is defined by

$$n/m = p \quad \text{iff} \quad m \cdot p = n .$$

For example, $(3,4,5,6)/(3,4) = (5,6)$ because $(3,4) \cdot (5,6) = (3,4,5,6)$. This notation is taken from [6, §2] without change.

(4.9) Lemma. Let $R, S \in V_*$; $m, n \in \operatorname{Dom} R$; m anc n.

(1) $\quad R/n = (R/m)/(n/m) \quad$ (cancellation)

(2) $\quad R(n \longleftarrow S)/m = (R/m)(n/m \longleftarrow S) \quad$ (distributivity)

The lemmas (4.6), (4.7), (4.9) are used constantly but rarely cited, just as commutativity and distributivity for numbers are used constantly but rarely cited in real analysis.

Diagrams like Figure 2-8 are too cumbersome as names of trees. Tabulations like

$$S = \{\langle (\,), p\rangle, \langle (0), q\rangle, \langle (1), c\rangle, \langle (2), r\rangle\}$$

are even more awkward. The natural algebraic structure of V_* leads to more manageable nomenclature. Each $a \in V$ defines an operation $\bar{a}: (V_*)^* \longrightarrow V_*$. For example, consider $(R, S) \in (V_*)^2$, with R, S from Figure 2-8. The tree $\bar{a}(R, S)$ is shown in Figure 2-9. Here we indicate the value of a function F at an argument w by Fw (with no round brackets because the scope of the argument expression is already clear). The values of trees are indicated similarly: Rm, not $R(m)$.

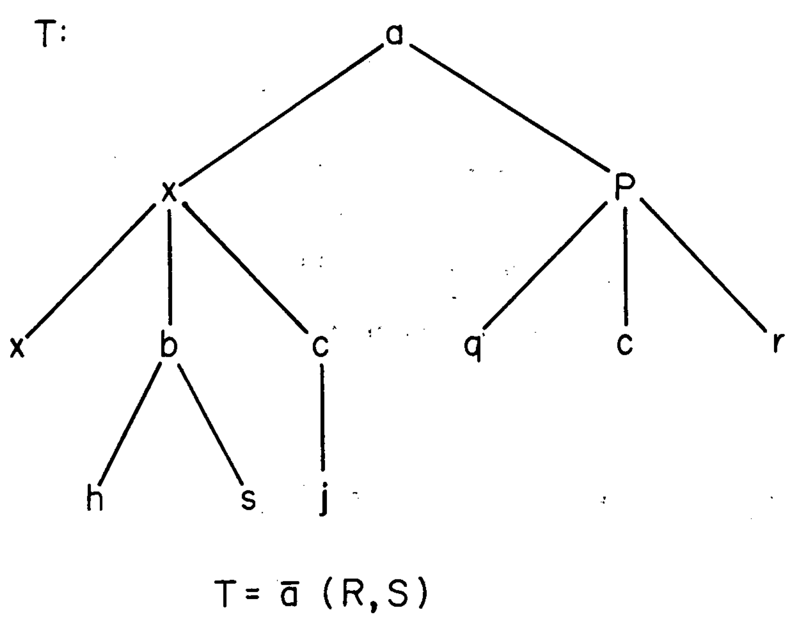


Figure 2-9. The operation $\bar{a}$ is applied to (R,S), where R,S are from Figure 2-8.

(4.10) Definition. Let $a \in V$; $K \in \mathbb{N}$; $(R_0, \ldots, R_{-1}) \in (V_*)^K$. Then

$$\bar{a}(R_0, \ldots, R_{-1}) = \{\langle (), a \rangle\} \cup \bigcup_{k<K} \{\langle (k) \cdot m, y \rangle \mid \langle m, y \rangle \in R_k\}.$$

Applying this definition to R in Figure 2-8, we have

$$R = \bar{x}(\bar{x}(\,), \bar{b}(\bar{h}(\,), \bar{s}(\,)), \bar{c}(\bar{j}(\,)))\,.$$

We will henceforth omit the names of null argument strings when using this definition. When confusion between $a \in V$ and $\bar{a} : (V_*)^* \longrightarrow V_*$ is unlikely, we will even omit the overlines, so that R has two abbreviated names:

$$R = \bar{x}(\bar{x}, \bar{b}(\bar{h}, \bar{s}), \bar{c}(\bar{j})) = x(x, b(h,s), c(j))\,.$$

This amounts to the "pseudoterm" notation [50, §1]. Several well-known notations are essentially the same as this one. Cambridge Polish notation uses (b, h, s) rather than b(h, s), while labelled bracket notation uses $[_b\ h, s\ ]_b$.

In many applications only the ranked trees are of interest: there is a rank function $\rho : V \longrightarrow \mathbb{N}$ and the symbols are thought of as "operators." Each operator x takes $\rho(x)$ operands and so a node labelled x in a tree should have exactly $\rho(x)$ sons. The set of all ranked trees with labels in V is denoted $V_\#$ and defined as follows:

(4.11) Definition. Let $\rho : V \longrightarrow \mathbb{N}$ and set

$$V_\# = V_\#(\rho) = \{R \in V_* \mid (\forall n \in \text{Dom } R)(|\text{Dom } R \cap \text{Fa}^{-1}(n)| = \rho(Rn))\}.$$

The round brackets and commas in our algebraic nomenclature can be omitted without any logical ambiguity for ranked trees. The result is the well-known Lukasiewicz notation.

5. Subtree Replacement Systems

Subtree replacement systems generalize the main idea in rewriting systems, where a replacement relation $\Longrightarrow$ is defined on strings. For strings X and Y, $X \Longrightarrow Y$ iff some substring of X matches the left side of a rule and Y is the result of substituting the right side of the rule in place of that substring in X.

(5.1) Definition. A subtree replacement system (SRS) is any 4-tuple $\mathfrak{C} = (V, \mathbb{F}, \Longrightarrow, \mathbb{R})$, where

(1) $\mathbb{F} \subseteq V_*$ ($\mathbb{F}$ is a forest.)

(2) $\mathbb{R} \subseteq V_* \times V_*$ (Write $\varphi \longrightarrow \psi$ for $\langle \varphi, \psi \rangle \in \mathbb{R}$.)

(3) $(\forall R \in \mathbb{F})(\forall S \in V_*)$

$$(\ R \Longrightarrow S \text{ iff } (\exists \varphi \longrightarrow \psi \in \mathbb{R})(\exists n \in \text{Dom } R)$$
$$[R/n = \varphi \ \& \ S = R(n \longleftarrow \psi)]) .$$

(4) $(\Longrightarrow) \subseteq \mathbb{F} \times \mathbb{F}$.

This definition generalizes the notion of $\Longrightarrow$ in Brainerd's "regular systems" [6, Def. 3.2]. The crucial difference is that we do not require that $\mathbb{R}$ be finite. The use of infinite sets of rules will permit us to describe many complex tree-manipulating systems within the SRS framework. The complexity of these systems will be broken down into two stages: specification of an infinite set of rules by some finite means, and then the use of rules in this set to evaluate trees.

Note that condition (3) in Definition 5.1 tells how to apply rules to trees in the forest $\mathbb{F}$, while condition (4) requires that the results of such applications be in $\mathbb{F}$ too. (In all our examples (4) will be quite trivial.) The system $(\mathbb{F}, \Longrightarrow)$ is therefore a GRS.

The same letter will often be used for an SRS and for the corresponding GRS, and results from §3 will be used to construct Church-Rosser wholes from Church-Rosser parts. For an initial source of Church-Rosser systems we consider some formal properties an SRS might have.

(5.2) Definition. An SRS $\mathfrak{C} = (V, \mathbb{F}, \Longrightarrow, \mathbb{R})$ is unequivocal iff $\mathbb{R}$ is a partial function: no two rules have the same left side.

Many SRSs are unequivocal but not Church-Rosser. To see how interference between rules can destroy the Church-Rosser property, let us momentarily revert to string rewriting systems. Consider a system with these two rules:

$$AB \longrightarrow CA \qquad\qquad A \longrightarrow a\,.$$

Let R be the string bABc. By applying $AB \longrightarrow CA$ we can form S = bCAc. By applying $A \longrightarrow a$ instead, we can form S′ = baBc. Because the substrings used to form S and S′ overlap in R, the opportunity for applying $AB \longrightarrow CA$ was destroyed in passing to S′. No common string T is derivable from both S and S′. Now construct a third rule by applying $A \longrightarrow a$ to the A in AB and to the A in CA in the rule $AB \longrightarrow CA$:

$$aB \longrightarrow Ca\,.$$

By adding this rule to the system we obtain the Church-Rosser property, since both S and S′ can now evolve to bCac.

In general we need systems "closed" under the operation of applying rules to rules. Whenever a little rule $\varphi \longrightarrow \psi$ is applicable inside the left half of a big rule $\varphi_0 \longrightarrow \psi_0$, then replacing φ by ψ at appropriate places in both φ_0 and ψ_0 leads to another rule $\varphi_1 \longrightarrow \psi_1$.

The net effect of applying $\varphi_0 \longrightarrow \psi_0$ and then $\varphi \longrightarrow \psi$ can be obtained by applying $\varphi \longrightarrow \psi$ and then $\varphi_1 \longrightarrow \psi_1$.

The closure concept for rules that are pairs of trees is illustrated in Figure 2-10. Trees are actually simpler than strings here, since two subtrees cannot overlap in a tree unless one includes the other. A more complicated closure concept is needed to insure the Church-Rosser property in an unequivocal string rewriting system.

In Figure 2-10 the little rule $\varphi \longrightarrow \psi$ is applicable at a node n inside the left half of the big rule $\varphi_0 \longrightarrow \psi_0$. More concisely: $\varphi_0/n = \varphi$, where $n \in \mathrm{Dom}\ \varphi_0$ and $n \neq (\ )$. We suppose that n has "residues" p and q where this same subtree appears after φ_0 has been replaced by ψ_0: $\psi_0/p = \psi_0/q = \varphi_0/n$. Replacing φ by ψ at n in Dom φ_0 and at each residue of n in Dom ψ_0 leads to a pair of trees $\varphi_1 \longrightarrow \psi_1$; we could say that the little rule $\varphi \longrightarrow \psi$ has been applied to the big rule $\varphi_0 \longrightarrow \psi_0$. In a closed system, $\varphi_1 \longrightarrow \psi_1$ is also in the set of rules being used.

The formal definition is conveniently divided into two parts. A "residue map" for $\varphi_0 \longrightarrow \psi_0$ assigns to each node $n \in \mathrm{Dom}\ \varphi_0$ a set r(n) of "residues" in Dom ψ_0. Each node n may have any number (including zero) of residues, but each residue p of n must have $\psi_0/p = \varphi_0/n$. We also require that independent nodes have independent residues: if $p \perp q$ in Dom φ_0 then $r(p) \perp r(q)$, where the independence relation between sets of nodes is as defined in (4.5.1). (This additional requirement is motivated more by technical considerations in the proof of the theorem we are approaching than by the basic intuitive idea.)

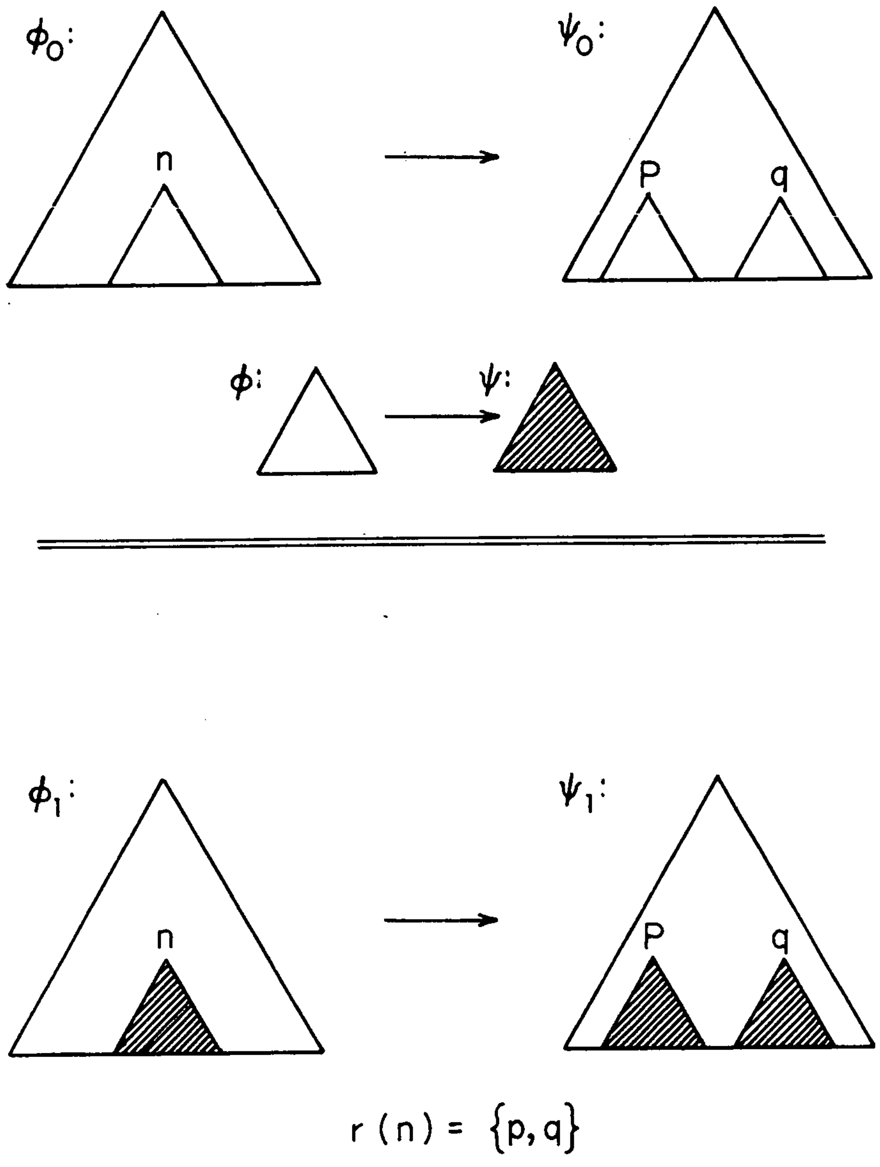


Figure 2-10. The rule $\varphi \longrightarrow \psi$ is applied at n in φ_0 and at each of the residues of n in ψ_0 to form another rule $\varphi_1 \longrightarrow \psi_1$.

(5.3) Definition. Let $\mathfrak{C} = (V, \mathbb{F}, \Rightarrow, \mathbb{R})$ be an SRS; $\varphi_0 \longrightarrow \psi_0 \in \mathbb{R}$. A function r defined on Dom φ_0 is a residue map for this rule iff

(1) $(\forall n \in \text{Dom}\,\varphi_0)[r(n) \subseteq \text{Dom}\,\psi_0 \ \& \ (\forall m \in r(n))(\psi_0/m = \varphi_0/n)]$

(2) $(\forall p, q \in \text{Dom}\,\varphi_0)(p \perp q \text{ implies } r(p) \perp r(q))$.

Now suppose that a residue map $r[\varphi_0, \psi_0]$ has been assigned to each rule $\varphi_0 \longrightarrow \psi_0$ and choose any specific rule $\varphi_0 \longrightarrow \psi_0$. Let r_0 be $r[\varphi_0, \psi_0]$ and let $n \in \text{Dom}\,\varphi_0$. In (1) above, we have $m \perp m'$ whenever $m, m' \in r_0(n)$, so $r_0(n)$ is an independent set of nodes. For any tree ψ we can form $\varphi_1 = \varphi_0(n \longleftarrow \psi)$ and $\psi_1 = \psi_0(r_0(n) \longleftarrow \psi)$. In particular, if some rule $\varphi \longrightarrow \psi$ has $\varphi_0/n = \varphi$, then we can form the pair of trees $\varphi_1 \longrightarrow \psi_1$ by applying $\varphi \longrightarrow \psi$ at n in φ_0 and at each residue of n in ψ_0. In a closed system $\varphi_1 \longrightarrow \psi_1$ must also be one of the rules. We will also require that nodes independent of n in φ_0 be uneffected by all this: if $p \perp n$ in Dom φ_0, then p has the same residues in ψ_1 as in ψ_0:

$$r[\varphi_1, \psi_1](p) = r[\varphi_0, \psi_0](p).$$

(Since $\varphi_1 = \varphi_0(n \longleftarrow \psi)$ and $p \perp n$, p is indeed a node in Dom φ_1.) (This additional requirement is motivated more by technical considerations in the proof of the theorem we are approaching than by the basic intuitive idea.)

(5.4) Definition. Let $\mathfrak{C} = (V, \mathbb{R}, \Rightarrow, \mathbb{R})$ be an SRS. Then $\mathfrak{C}$ is closed iff it is possible to assign a residue map $r[\varphi_0, \psi_0]$ to each rule $\varphi_0 \longrightarrow \psi_0 \in \mathbb{R}$ in such a way that the following holds: whenever $\varphi_0 \longrightarrow \psi_0$, $\varphi \longrightarrow \psi \in \mathbb{R}$ and $n \in \text{Dom}\,\varphi_0$ with $n \neq (\ )$ and $\varphi_0/n = \varphi$, then

(1) $$\varphi_1 \longrightarrow \psi_1 \in \mathbb{R}$$

and

(2) $$(\forall p \in \mathrm{Dom}\ \varphi_0)[p \perp n \text{ implies } r[\varphi_1, \psi_1](p) = r[\varphi_0, \psi_0](p)],$$

where

(3) $$\varphi_1 = \varphi_0(\longleftarrow \psi) \ \& \ \psi_1 = \psi_0(r[\varphi_0, \psi_0](n) \longleftarrow \psi).$$

The evaluation of arithmetic expressions in §1 provides an example of a closed SRS. If $\varphi_0(\ )$ is an ordinary arithmetic operator, as in

$$\overline{\mathsf{X}}(\overline{47}, \overline{23}) \longrightarrow \overline{1081},$$

we set $r[\varphi_0, \psi_0](n) = \emptyset$ for all $n \in \mathrm{Dom}\ \varphi_0$. This is trivially a residue map. If $\varphi_0(\ )$ is the special conditional operator C, then $\varphi_0 \longrightarrow \psi_0$ is either of the form

(*) $$\overline{\mathrm{C}}(\overline{\underset{\sim}{\mathrm{true}}}, \psi_0, \omega) \longrightarrow \psi_0$$

or of the form

(**) $$\overline{\mathrm{C}}(\overline{\underset{\sim}{\mathrm{false}}}, \omega, \psi_0) \longrightarrow \psi_0 .$$

In the first case we set

$$r[*](n) = \begin{cases} \{n/(1)\} & \text{if } (1)\ \underline{\mathrm{anc}}\ n \\ \emptyset & \text{otherwise} \end{cases}$$

for all $n \in \mathrm{Dom}\ \varphi_0$. In the second case we set

$$r[**](n) = \begin{cases} \{n/(2)\} & \text{if } (2)\ \underline{\text{anc}}\ n \\ \emptyset & \text{otherwise} \end{cases}$$

for all $n \in \text{Dom}\ \varphi_0$.

To see that $r[*]$ is indeed a residue map in Definition 5.3, note that (5.3.1) is only nontrivial for a node n with $(1)\ \underline{\text{anc}}\ n$. Since $\varphi_0/(1) = \psi_0$, $n/(1)$ is indeed in Dom ψ_0. Furthermore, the cancellation law (4.9.1) implies that each $p \in r[*](n) = \{n/(1)\}$ has

$$\varphi_0/n = (\varphi_0/(1))/(n/(1)) = \psi_0/p.$$

We also need (5.3.2): independent nodes have independent residues. For $p \perp q$ in Dom φ_0, we may simply note that $\emptyset \perp M$ and $M \perp \emptyset$ for any set M of nodes, unless both $r[*](p)$ and $r[*](q)$ are nonempty. But in that case we have $p/(1) \perp q/(1)$ because $p \perp q$ and $(1)\ \underline{\text{anc}}\ p$ and $(1)\ \underline{\text{anc}}\ q$. Therefore, $r[*](p) \perp r[*](q)$ in all cases. (Similarly, $r[**]$ is also a residue map.)

To see that this system is indeed closed in Definition 5.4, suppose that $\varphi_0 \longrightarrow \psi_0$, $\varphi \longrightarrow \psi \in \mathbb{R}$ and $n \in \text{Dom}\ \varphi_0$ with $n \neq (\ )$ and $\varphi_0/n = \varphi$. Then $\varphi_0 \longrightarrow \psi_0$ cannot be an ordinary arithmetic rule because $\varphi \longrightarrow \psi$ cannot be of the form $\overline{\text{numeral}} \longrightarrow \ldots$. Therefore $\varphi_0 \longrightarrow \psi_0$ is as in (*) or (**). Both cases are treated the same way, so we will assume (*) for the sake of definiteness. As in (5.4.3), we set

$$\varphi_1 = \varphi_0(n \longleftarrow \psi) \ \ \& \ \ \psi_1 = \psi_0(r[\varphi_0, \psi_0] \longleftarrow \psi).$$

We must verify (5.4.1): $\varphi_1 \longrightarrow \psi_1 \in \mathbb{R}$. In $\overline{C}(\underset{\sim}{\text{true}}, \psi_0, \omega) = \varphi_0$, there is only room for φ_0/n to be the left half of a rule if $(1)\ \underline{\text{anc}}\ n$ or $(2)\ \underline{\text{anc}}\ n$. Suppose first that $(1)\ \underline{\text{anc}}\ n$. Then

$$\varphi_1 = \bar{C}(\overline{\underset{\sim}{\text{true}}}, \psi_0, \omega)(n \longleftarrow \psi)$$
$$= \bar{C}(\overline{\underset{\sim}{\text{true}}}, \psi_0(n/(1) \longleftarrow \psi), \omega),$$

while

$$\psi_1 = \psi_0(\{n/(1)\} \longleftarrow \psi) = \psi_0(n/(1) \longleftarrow \psi).$$

Therefore $\varphi_1 \longrightarrow \psi_1$ is another of the rules for conditionals. Now suppose that (2) $\underline{\text{anc}}$ n. Then

$$\varphi_1 = \bar{C}(\overline{\underset{\sim}{\text{true}}}, \psi_0, \omega)(n \longleftarrow \psi)$$
$$= \bar{C}(\overline{\underset{\sim}{\text{true}}}, \psi_0, \omega(n/(2) \longleftarrow \psi))$$

$$\psi_1 = \psi_0(\emptyset \longleftarrow \psi) = \psi_0 .$$

Therefore $\varphi_1 \longrightarrow \psi_1$ is another of the rules for conditionals. In both cases, $\varphi_1 \longrightarrow \psi_1 \in \mathbb{R}$.

We must verify (5.4.2): for $p \perp n$ in Dom φ_0, the new set of residues $r[\varphi_1, \psi_1](n)$ is the same as $r[\varphi_0, \psi_0](p)$. We saw above that $\varphi_1 \longrightarrow \psi_1$ is of the form (*), so $r[\varphi_1, \psi_1](p)$ is defined by the r[*] equation:

$$r[\varphi_1, \psi_1](p) = \begin{cases} \{p/(1)\} & \text{if} \quad (1)\ \underline{\text{anc}}\ p \\ \emptyset & \text{otherwise} \end{cases} .$$

Since $r[\varphi_0, \psi_0](p)$ is also defined by the r[*] equation, we do have

$$r[\varphi_1, \psi_1](p) = r[\varphi_0, \psi_0](p) .$$

In the next section we will show that any system whose rules can be specified in a certain natural way is closed. The argument given above is actually a special case of the argument in the next section; these tedious verifications do not have to be carried out in every concrete situation.

That an unequivocal closed SRS should be Church-Rosser is quite plausible, but neither induction on lengths of derivations nor induction on sizes of trees is appropriate. Induction on the sizes of certain sets of nodes does work. The size of $N \subseteq \mathbb{N}^*$ is its cardinality $|N|$. Given an SRS $\mathfrak{C} = (V, \mathbb{F}, \Longrightarrow, \mathbb{R})$, we define a relation $\underset{N}{\Longrightarrow}$ on $\mathbb{F}$ for every $N \subseteq \mathbb{N}^*$. The union of all the relations $\underset{N}{\Longrightarrow}$ will be called $\underset{\perp}{\Longrightarrow}$. Whenever $R \underset{\perp}{\Longrightarrow} S$, it will be possible to derive S from R by simultaneously applying rules at all the nodes in an independent subset of Dom R.

(5.5) <u>Definition</u>. Let $\mathfrak{C} = (V, \mathbb{F}, \Longrightarrow, \mathbb{R})$ be an SRS. For any $N \subseteq \mathbb{N}^*$, let $(n_0, \ldots, n_{|N|-1})$ be a listing of the members of N and define $\underset{N}{\Longrightarrow}$ to be the set of all $\langle R, S\rangle$ in $\mathbb{F} \times V_*$ such that

(1) N is an independent set of nodes in Dom R

(2) $(\exists\, \varphi, \psi \in (V_*)^{|N|})$

$$[(\forall\, i < |N|)(R/n_i = \varphi_i \;\&\; \varphi_i \longrightarrow \psi_i \in \mathbb{R}) \;\&$$

$$S = R(n_0 \longleftarrow \psi_0) \ldots (n_{-1} \longleftarrow \psi_{-1})].$$

Now define

(3) $$(\underset{\perp}{\Longrightarrow}) = \bigcup_{N \subseteq \mathbb{N}^*} (\underset{N}{\Longrightarrow}).$$

By (1) in this definition, $\underset{N}{\Longrightarrow}$ is $\emptyset$ unless N is actually a finite set of nodes that is independent in the sense of (4.5.2). Suppose that N is indeed an independent finite set of nodes. By (1) and (2), to say that

$R \underset{N}{\Longrightarrow} S$ is to say that S may be formed from R by applying rules at the members of N. The order of the listing $(n_0, \ldots, n_{-1})$ is irrelevant by commutativity (4.7.4). For any $i < |N|$, we can show that

$$R(n_0 \longleftarrow \psi_0) \ldots (n_{i-1} \longleftarrow \psi_{i-1})/n_i = R/n_i$$

by i applications of persistence (4.7.3), so that

$$R(n_0 \longleftarrow \psi_0) \ldots (n_{i-1} \longleftarrow \psi_{i-1}) \Longrightarrow R(n_0 \longleftarrow \psi_0) \ldots (n_i \longleftarrow \psi_i)$$

by application of $\varphi_i \longrightarrow \psi_i$ at n_i. Therefore

$$(\underset{N}{\Longrightarrow}) \subseteq (\overset{|N|}{\Longrightarrow})$$

and $\underset{N}{\Longrightarrow}$ is actually a subset of $\mathbb{F} \times \mathbb{F}$ because $\Longrightarrow$ is a subset of $\mathbb{F} \times \mathbb{F}$.

(5.6) Main Theorem. Any unequivocal closed SRS is Church-Rosser.

Proof: Let $\mathfrak{C} = (V, \mathbb{F}, \Longrightarrow, \mathbb{R})$ be an unequivocal closed SRS. We will use Lemma 3.4 to show that $(\mathbb{F}, \Longrightarrow)$ is Church-Rosser. Let $\underset{\perp}{\Longrightarrow}$ be as in Definition 5.5. Since $(\underset{N}{\Longrightarrow}) \subseteq (\overset{|N|}{\Longrightarrow})$ whenever N is an independent finite subset of $\mathbb{N}^*$ and $\underset{N}{\Longrightarrow} = \emptyset$ otherwise, the union $\underset{\perp}{\Longrightarrow}$ in (5.5.3) is contained in $\overset{*}{\Longrightarrow}$. Therefore $\underset{\perp}{\Longrightarrow}$ is indeed a relation on $\mathbb{F}$.

We need (3.4.1): $(\underset{\perp}{\overset{*}{\Longrightarrow}}) = (\overset{*}{\Longrightarrow})$. We already have $(\underset{\perp}{\Longrightarrow}) \subseteq (\overset{*}{\Longrightarrow})$, so it will suffice to show that $(\Longrightarrow) \subseteq (\underset{\perp}{\Longrightarrow})$. Suppose $R \Longrightarrow S$, so that Definition 5.1 implies that some $n \in \text{Dom } R$ and some $\varphi \longrightarrow \psi \in \mathbb{R}$ have

$$R/n = \varphi \;\;\&\;\; S = R(n \longleftarrow \psi).$$

In Definition 5.5 we have $R \underset{\{n\}}{\Longrightarrow} S$ and therefore $R \underset{\perp}{\Longrightarrow} S$, as desired.

We need (3.4.2):

$$(\forall R,S,S' \in \mathbb{F})[(R \underset{1}{\Longrightarrow} S \;\&\; R \underset{1}{\Longrightarrow} S') \text{ implies}$$
$$(\exists T \in \mathbb{F})(S \underset{1}{\Longrightarrow} T \;\&\; S' \underset{1}{\Longrightarrow} T)].$$

When $R \underset{1}{\Longrightarrow} S$ and $R \underset{1}{\Longrightarrow} S'$ we have $R \underset{N}{\Longrightarrow} S$ and $R \underset{N'}{\Longrightarrow} S$ for some $N, N' \subseteq \text{Dom } R$. The induction will be on $|N \cup N'|$. As often happens in inductions, we must carry along more information than is actually used at the journey's end. The extra information is the fact that S and S′ can both become a common tree T by a single step of $\underset{1}{\Longrightarrow}$ in which the nodes used are descended from nodes in $N \cup N'$. Specifically, we verify (3.4.2) by proving the following statement for each $K \in \mathbb{N}$:

(1) $$(\forall R,S,S' \in \mathbb{F})(\forall N,N' \subseteq \text{Dom } R)$$
$$[(R \underset{N}{\Longrightarrow} S \;\&\; R \underset{N'}{\Longrightarrow} S' \;\&\; |N \cup N'| = K) \quad \text{implies}$$
$$(\exists P \subseteq \text{Dom } S)(\exists P' \subseteq \text{Dom } S')(\exists T \in \mathbb{F})$$
$$(S \underset{P}{\Longrightarrow} T \;\&\; S' \underset{P'}{\Longrightarrow} T \;\&\; P \cup P' \subseteq (N \cup N') \cdot \mathbb{N}^*)].$$

We use the induction schema

$$(\forall K \in \mathbb{N})[(\forall k < K)(\ldots k \ldots) \quad \text{implies} \quad (\ldots K \ldots)]$$
$$\text{implies} \quad (\forall K \in \mathbb{N})(\ldots K \ldots).$$

Let $K \in \mathbb{N}$ and let (1) hold for all $k < K$. Let R, S, S', N, N' be as in (1), with $|N \cup N'| = K$. Let rules $\varphi_i \longrightarrow \psi_i$ be applied at nodes n_i in N to form S, while rules $\varphi'_j \longrightarrow \psi'_j$ are applied at nodes n'_j in N′ to form S′. We show how to choose P, P′, T.

Case 1 $(N \perp N')$ Let $P = N'$; $P' = N$;

$$T = R(n_0 \longleftarrow \psi_0) \ldots (n_{-1} \longleftarrow \psi_{-1})(n'_0 \longleftarrow \psi'_0) \ldots (n'_{-1} \longleftarrow \psi'_{-1}).$$

Then P, P', T have all the desires properties by persistence (4.7.3) and commutativity (4.7.4).

Case 2 (NOT $N \perp N'$) Suppose some member of N is an ancestor of some member of N'. After permuting the listings for N and N' as needed, we may assume that some $J > 0$ has n_0 anc n'_j for all $j < J$ and NOT(n_0 anc n'_j) for all $j \geqslant J$. (The roles of primed and unprimed variables can simply be reversed if no member of N is an ancestor of a member of N'.) Let N_0 be $N - \{n_0\}$ and N'_0 be $N' - \{n'_j \mid j < J\}$. Then $n_0 \perp (N_0 \cup N'_0)$ by independence of N'_0, transitivity (4.6.3), and the definitions (4.1.4), (4.5).

Case 2.1 ($n_0 = n'_j$ for some $j < J$; choose one.) Let R_0 be $R(n_0 \longleftarrow \psi_0)$. Since $\mathfrak{C}$ is unequivocal, $R_0 = R(n'_j \longleftarrow \psi'_j)$ also. We have

$$R_0 \underset{N_0}{\Longrightarrow} S \quad \& \quad R_0 \underset{N' - \{n'_j\}}{\Longrightarrow} S' \quad \& \quad |N_0 \cup N' - \{n'_j\}| = K - 1 .$$

By the induction hypothesis there are P, P', T with $S \underset{P}{\Longrightarrow} T$ and $S' \underset{P'}{\Longrightarrow} T$ and $P \cup P' \subseteq (N_0 \cup N' - \{n'_j\}) \cdot \mathbb{N}^* \subseteq (N \cup N') \cdot \mathbb{N}^*$.

Case 2.2 ($n_0 \neq n'_j$, all $j < J$.) We wish to apply the induction hypothesis to N_0 and N'_0, which do have $|N_0 \cup N'_0| < K$. We therefore construct trees R_0, S_0, S'_0 that are closely related to R, S, S' and that have $R_0 \underset{N_0}{\Longrightarrow} S_0$ and $R_0 \underset{N'_0}{\Longrightarrow} S'_0$. The construction hinges on a rule $\bar{\varphi} \longrightarrow \bar{\psi}$. Intuitively, $\bar{\varphi} \longrightarrow \bar{\psi}$ is the result of applying $\varphi'_0 \longrightarrow \psi'_0, \ldots, \varphi'_{J-1} \longrightarrow \psi'_{J-1}$ to $\varphi_0 \longrightarrow \psi_0$.

Let residue maps be assigned to rules so as to satisfy the conditions in the definition of closed systems (5.4). Let $r[\varphi_0, \psi_0]$ be the residue map for $\varphi_0 \longrightarrow \psi_0$. Since $\{n'_j/n_0 \mid j < J\}$ is an independent subset of Dom φ_0, the sets

$$M_j = r[\varphi_0, \psi_0](n'_j/n_0) \quad \text{for all } \ j < J$$

are independent subsets of Dom ψ_0 with $M_j \perp M_k$ whenever $j \neq k$. The union

$$M = \bigcup_{j<J} M_j$$

is therefore an independent subset of Dom ψ_0.

Define a pair of trees $\bar{\varphi} \longrightarrow \bar{\psi}$ by

(2) $$\begin{aligned} \bar{\varphi} &= \varphi_0(n'_0/n_0 \longleftarrow \psi'_0) \ldots (n'_{J-1}/n_0 \longleftarrow \psi'_{J-1}) \\ \bar{\psi} &= \psi_0(M_0 \longleftarrow \psi'_0) \ldots (M_{J-1} \longleftarrow \psi'_{J-1}) . \end{aligned}$$

We claim that $\bar{\varphi} \longrightarrow \bar{\psi}$ is in $\mathbb{R}$. Suppose not. Closure implies

$$\varphi_0(n'_0/n_0 \longleftarrow \psi'_0) \longrightarrow \psi_0(M_0 \longleftarrow \psi'_0) \in \mathbb{R}$$

by (5.4.1), so we must have $J > 1$ and there must be some $j < J-1$ such that

$$\begin{aligned} &\hat{\varphi} \longrightarrow \hat{\psi} \in \mathbb{R} \\ &\hat{\varphi}(n'_{j+1}/n_0 \longleftarrow \psi'_{j+1}) \longrightarrow \hat{\psi}(M_{j+1} \longleftarrow \psi'_{j+1}) \notin \mathbb{R} \end{aligned}$$

where

$$\begin{aligned} \hat{\varphi} &= \varphi_0(n'_0/n_0 \longleftarrow \psi'_0) \ldots (n'_j/n_0 \longleftarrow \psi'_j) \\ \hat{\psi} &= \psi_0(M_0 \longleftarrow \psi'_0) \ldots (M_j \longleftarrow \psi'_j). \end{aligned}$$

But j applications of (5.4.2) imply that $r[\hat{\varphi}, \hat{\psi}](n'_{j+1}/n_0) = M_{j+1}$ and j applications of persistence (4.7.3) imply that $\hat{\varphi}/(n'_{j+1}/n_0) = \varphi'_{j+1}$.

In (5.4.1) we get a contradiction:

$$\hat{\varphi}(n'_{j+1}/n_0 \longleftarrow \psi'_{j+1}) \longrightarrow \hat{\psi}(M_{j+1} \longleftarrow \psi'_{j+1}) \in \mathbb{R}.$$

Therefore $\overline{\varphi} \longrightarrow \overline{\psi} \in \mathbb{R}$.

Figure 2-11 will be verified next. Let

(3) $$R_0 = R(n_0 \longleftarrow \overline{\psi})$$
$$S_0 = S(n_0 \longleftarrow \overline{\psi})$$
$$S'_0 = S'(n_0 \longleftarrow \overline{\psi}).$$

We will show that $\overline{\varphi} \longrightarrow \overline{\psi}$ may be applied to S' at n_0, and that then

(4) $$S' \underset{\{n_0\}}{\Longrightarrow} S'_0.$$

By J applications of distributivity (4.9.2) to (2),

$$\overline{\varphi} = (R/n_0)(n'_0/n_0 \longleftarrow \psi'_0)\ldots(n'_{J-1}/n_0 \longleftarrow \psi'_{J-1})$$
$$= R(n'_0 \longleftarrow \psi'_0)\ldots(n'_{J-1} \longleftarrow \psi'_{J-1})/n_0.$$

By $n_0 \perp N'_0$ and $|N'_0|$ applications of persistence (4.7.3), this becomes

$$\overline{\varphi} = R(n'_0 \longleftarrow \psi'_0)\ldots(n'_{J-1} \longleftarrow \psi'_{J-1})\ldots(n'_{-1} \longleftarrow \psi'_{-1})/n_0$$
$$= S'/n_0.$$

By $\overline{\varphi} = S'/n_0$, $\overline{\varphi} \longrightarrow \overline{\psi} \in \mathbb{R}$, and $S'_0 = S'(n_0 \longleftarrow \overline{\psi})$ in (3), we have (4).

Because M is an independent subset of Dom ψ_0 and $\psi_0 = S/n_0$, $n_0 \cdot M$ is an independent subset of Dom S. We will show that

(5) $$S \underset{n_0 \cdot M}{\Longrightarrow} S_0,$$

where $\varphi'_j \longrightarrow \psi'_j$ is applied at $n_0 \cdot p$ whenever $j < J$ and $p \in M_j$. Suppose $j < J$ and $p \in M_j$. By $|N| - 1$ applications of persistence (4.7.3) and then

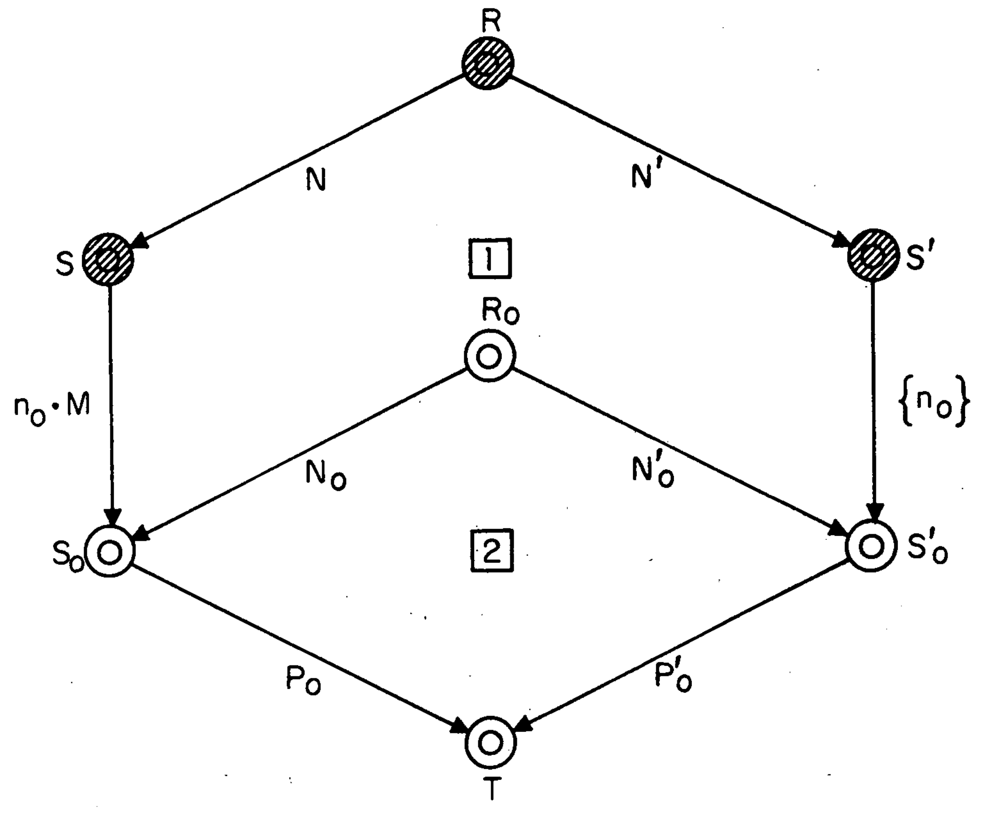


Figure 2-11. Case 2.2 in the proof of the Main Theorem (5.6).

one application of embedding (4.7.1), we have

$$\begin{aligned} S/(n_0 \cdot p) &= R(n_0 \longleftarrow \psi_0) \ldots (n_{-1} \longleftarrow \psi_{-1})/(n_0 \cdot p) \\ &= R(n_0 \longleftarrow \psi_0)/(n_0 \cdot p) \\ &= \psi_0/p . \end{aligned}$$

By (5.3.1) in the definition of residue maps we have

$$\psi_0/p = \varphi_0/(\, n'_j/n_0) = \varphi'_j ,$$

so that $S/(n_0 \cdot p) = \varphi'_j$. By $|M|$ applications of associativity (4.7.2) in (3) and (2),

$$\begin{aligned} S_0 &= S(n_0 \longleftarrow \psi_0(M_0 \longleftarrow \psi'_0) \ldots (M_{J-1} \longleftarrow \psi'_{J-1})) \\ &= S(n_0 \longleftarrow \psi_0)(n_0 \cdot M_0 \longleftarrow \psi'_0) \ldots (n_0 \cdot M_{J-1} \longleftarrow \psi'_{J-1}) . \end{aligned}$$

But $S(n_0 \longleftarrow \psi_0) = S$ because $S/n_0 = \psi_0$, so

$$S_0 = S(n_0 \cdot M_0 \longleftarrow \psi'_0) \ldots (n_0 \cdot M_{J-1} \longleftarrow \psi'_{J-1}) .$$

This completes the proof of (5).

We now claim that

(6) $$R_0 \underset{N_0}{\Longrightarrow} S_0 \quad \& \quad R_0 \underset{N'_0}{\Longrightarrow} S'_0 .$$

First we must show that $R_0 \in \mathbb{F}$. By applying $\varphi'_j \longrightarrow \psi'_j$ at n'_j for each $j < J$, we get

(7) $$R \underset{1}{\Longrightarrow} R(n'_0 \longleftarrow \psi'_0) \ldots (n'_{J-1} \longleftarrow \psi'_{J-1}) .$$

By J applications of distributivity (4.9.2) and then by (2),

$$\begin{aligned}&R(n'_0 \leftarrow \psi'_0)\dots(n'_{J-1} \leftarrow \psi'_{J-1})/n_0\\ &\quad = (R/n_0)(n'_0/n_0 \leftarrow \psi'_0)\dots(n'_{J-1}/n_0 \leftarrow \psi'_{J-1})\\ &\quad = \varphi_0(n'_0/n_0 \leftarrow \psi'_0)\dots(n'_{J-1}/n_0 \leftarrow \psi'_{J-1})\\ &\quad = \overline{\varphi}.\end{aligned}$$

Applying the rule $\overline{\varphi} \longrightarrow \overline{\psi}$ at n_0, we get

$$\begin{aligned}&R(n'_0 \leftarrow \psi'_0)\dots(n'_{J-1} \leftarrow \psi'_{J-1})\\ &\quad \Longrightarrow R(n'_0 \leftarrow \psi'_0)\dots(n'_{J-1} \leftarrow \psi'_{J-1})(n_0 \leftarrow \overline{\psi})\\ &\quad\ = R(n_0 \leftarrow \overline{\psi})\end{aligned}$$

because $n_0 \underline{\text{anc}}\, n'_j$ for all $j < J$. (Recall the definition of substitution (4.4.2).) By (7) and $R(n_0 \leftarrow \overline{\psi}) = R_0$ in (3), we have $R \underset{\text{I}}{\Longrightarrow} \Longrightarrow R_0$. Therefore $R_0 \in \mathbb{F}$.

Now we will show that $R_0 \underset{\overline{N_0}}{\Longrightarrow} S_0$ by applying $\varphi_i \longrightarrow \psi_i$ at n_i for each i with $0 < i < |N|$. Suppose $0 < i < |N|$, so that $n_i \perp n_0$ and persistence (4.7.3) implies that

$$R_0/n_i = R(n_0 \leftarrow \overline{\psi})/n_i = R/n_i = \varphi_i$$

in (3). By $|N_0|$ applications of commutativity (4.7.4) in (3),

$$\begin{aligned}S_0 &= S(n_0 \leftarrow \overline{\psi})\\ &= R(n_0 \leftarrow \psi_0)(n_1 \leftarrow \psi_1)\dots(n_{-1} \leftarrow \psi_{-1})(n_0 \leftarrow \overline{\psi})\\ &= R(n_0 \leftarrow \psi_0)(n_0 \leftarrow \overline{\psi})(n_1 \leftarrow \psi_1)\dots(n_{-1} \leftarrow \psi_{-1}).\end{aligned}$$

By $n_0 \underline{\text{anc}}\, n_0$ we have

$$R(n_0 \leftarrow \psi_0)(n_0 \leftarrow \overline{\psi}) = R(n_0 \leftarrow \overline{\psi})$$

in the definition of substitution (4.4.2), so that

$$\begin{aligned} S_0 &= R(n_0 \longleftarrow \bar{\psi})(n_1 \longleftarrow \psi_1)\ldots(n_{-1} \longleftarrow \psi_{-1}) \\ &= R_0(n_1 \longleftarrow \psi_1)\ldots(n_{-1} \longleftarrow \psi_{-1}). \end{aligned}$$

This proves the first half of (6). Similar reasoning with $n'_J, \ldots, n'_{-1}$ in place of $n_1, \ldots, n_{-1}$ and use of (4.4.2) to show

$$R(n'_0 \longleftarrow \psi'_0)\ldots(n'_{J-1} \longleftarrow \psi'_{J-1})(n_0 \longleftarrow \bar{\psi}) = R(n_0 \longleftarrow \bar{\psi})$$

leads to the second half of (6).

By (4), (5), and (6), the upper portion (D1) of Figure 2-11 is now complete. Because $|N_0 \cup N'_0| = K - 1 - J < K$, the induction hypothesis yields P_0, P'_0, T with $S_0 \underset{P_0}{\Longrightarrow} T$; $S'_0 \underset{P'_0}{\Longrightarrow} T$; $P_0 \cup P'_0 \subseteq (N_0 \cup N'_0) \cdot \mathbb{N}^*$. This completes the lower portion (D2) of Figure 2-11.

Let $P = n_0 \cdot M \cup P_0$ and $P' = \{n_0\} \cup P'_0$. By

(8) $$n_0 \perp (N_0 \cup N'_0) \ \& \ P_0 \cup P'_0 \subseteq (N_0 \cup N'_0) \cdot \mathbb{N}^*,$$

both P and P′ are independent sets of nodes. By (5) and (4),

$$\text{Dom } S_0 - n_0 \cdot \mathbb{N}^* \subseteq \text{Dom } S \ \& \ \text{Dom } S'_0 - n_0 \cdot \mathbb{N}^* \subseteq \text{Dom } S'.$$

But (8) implies that

$$P_0 \subseteq \text{Dom } S_0 - n_0 \cdot \mathbb{N}^* \ \& \ P'_0 \subseteq \text{Dom } S'_0 - n_0 \cdot \mathbb{N}^*.$$

Therefore

$$P_0 \subseteq \text{Dom } S \ \& \ P'_0 \subseteq \text{Dom } S',$$

which implies that

$$P \subseteq \text{Dom } S \ \& \ P' \subseteq \text{Dom } S'.$$

Thus P and P′ are independent subsets of Dom S and Dom S′, respectively. By $S_0 \underset{P_0}{\Longrightarrow} T$ and $|n_0 \cdot M|$ applications of persistence (4.7.3) to

(5), we have $S \underset{P}{\Longrightarrow} T$. By $S'_0 \underset{P'_0}{\Longrightarrow} T$ and one application of persistence (4.7.3) to (4), we have $S' \underset{P'}{\Longrightarrow} T$. Finally,

$$P \cup P' \subseteq n_0 \cdot \mathbb{N}^* \cup (N_0 \cup N'_0) \cdot \mathbb{N}^* \subseteq (N \cup N') \cdot \mathbb{N}^* . \blacksquare$$

As we have remarked, some of the restrictions in the definitions of residue maps and of closed SRSs are based less on the intuitive ideas than on the need to show that $\overline{\varphi} \longrightarrow \overline{\psi} \in \mathbb{R}$ in the proof of the Main Theorem above. In Chapters 3 and 5 these strong definitions are appropriate: all the SRSs are closed in the full technical sense. In Chapter 4 there will be a system that seems to be "essentially" closed but that does not meet all the formal requirements. A complicated special construction will be needed to imitate this SRS by the interaction of two SRSs that do meet the requirements. It would be helpful if the definitions could be weakened so that unequivocal closed SRSs would still be Church-Rosser but more systems would be unequivocal and closed.

The following modification of the closure concept is too weak to support a Church-Rosser theorem, but it does have some uses in Chapter 4.

(5.7) <u>Definition.</u> Let $\mathfrak{C} = (V, \mathbb{F}, \Longrightarrow, \mathbb{R})$ be an SRS; $\mathbb{R}_1, \mathbb{R}_2 \subseteq \mathbb{R}$; $\varphi_0 \longrightarrow \psi_0 \in \mathbb{R}_1$; $n \in \mathrm{Dom}\, \varphi_0$; $n \neq ()$; $M \subseteq \mathrm{Dom}\, \psi_0$ with M independent. Then $\varphi_0 \longrightarrow \psi_0$ is <u>pseudoclosed</u> at (n, M) with respect to $(\mathbb{R}_1, \mathbb{R}_2)$ iff, whenever $\varphi \longrightarrow \psi \in \mathbb{R}_2$ and $\varphi_0/n = \varphi$, then there is some $\tilde{\psi} \in V_*$ such that

(1) $$\varphi_0(n \longleftarrow \psi) \longrightarrow \psi_0(M \longleftarrow \tilde{\psi}) \in \mathbb{R}_1$$

(2) $$(\forall p \in M)(\psi_0/p \longrightarrow \tilde{\psi} \in \mathbb{R}_2).$$

The set M is like the set r(n) of residues of n in a closed system, but here we have only assigned a subset of Dom ψ_0 to one node n in the left half of one rule $\varphi_0 \longrightarrow \psi_0$. We have not assigned a subset of the right half's nodes to every node in the left half of every rule. We do require that M be independent, but ψ_0/p need not be φ_0/n for $p \in M$.

As is appropriate with so weak a method of assigning nodes in ψ_0 to nodes in φ_0, we say that the individual rule $\varphi_0 \longrightarrow \psi_0$ is pseudoclosed at the specific pair (n, M) of a node in φ_0 and "corresponding" nodes in ψ_0. We do not require that every node in the left half of every rule behave a certain way.

We require that some $\tilde{\psi} \in V_*$ have

(a) $$\varphi_0(n \longleftarrow \psi) \longrightarrow \psi_0(M \longleftarrow \tilde{\psi}) \in \mathbb{R}_1$$

rather than the condition

(b) $$\varphi_0(n \longleftarrow \psi) \longrightarrow \psi_0(r[\varphi_0, \psi_0](n) \longleftarrow \psi) \in \mathbb{R}$$

in the definition of closed SRSs (5.4). Since we may not have $\psi_0/p = \varphi$ for all $p \in M$ and we may not have $\tilde{\psi} = \psi$, we must require $\psi_0/p \longrightarrow \tilde{\psi} \in \mathbb{R}_2$ separately.

In one respect we have strengthened the definition: (a) says that the new rule belongs to the same subset $\mathbb{R}_1$ of $\mathbb{R}$ as does $\varphi_0 \longrightarrow \psi_0$, while (b) merely says that the new rule is in $\mathbb{R}$. This additional information will be helpful in Chapter 4, where the rules of the lambda calculus will quite naturally divide into two subsets.

A fragment of the proof of the Main Theorem (5.6) can be extended to the context of pseudoclosed rules. Using the $\underset{N}{\Longrightarrow}$ notation defined by (5.5), we suppose that $R \underset{\{m\}}{\Longrightarrow} S$ and $R \underset{\{m'\}}{\Longrightarrow} S'$, where m $\underline{\text{anc}}$ $m' \neq m$

and the rule applied at m is pseudoclosed at $(m'/m, M)$ for some M. We will show that some T has $S \xRightarrow{|M|} T$ and $S' \Longrightarrow T$. The formal lemma yields additional information about the rules used to derive T.

(5.8) Lemma. Let $\mathfrak{C} = (V, F, \Longrightarrow, \mathbb{R})$ be an SRS; $\mathbb{R}_1, \mathbb{R}_2 \subseteq \mathbb{R}$; $R, S, S' \in \mathbb{F}$; $m, m' \in \operatorname{Dom} R$. Suppose $R \underset{\{m\}}{\Longrightarrow} S$; $R \underset{\{m'\}}{\Longrightarrow} S'$; $m \ \underline{\text{anc}}\ m' \neq m$; $R/m \longrightarrow S/m \in \mathbb{R}_1$; $R/m' \longrightarrow S'/m' \in \mathbb{R}_2$. Suppose $M \subseteq \operatorname{Dom}(S/m)$, independent, with $R/m \longrightarrow S/m$ pseudoclosed at $(m'/m, M)$ with respect to $(\mathbb{R}_1, \mathbb{R}_2)$. Then there is a $T \in \mathbb{F}$ such that

$$S \xRightarrow{|M|} T \qquad \text{(using rules in } \mathbb{R}_2)$$
$$S' \Longrightarrow T \qquad \text{(using a rule in } \mathbb{R}_1).$$

Proof: Let $\varphi_0 \longrightarrow \psi_0$ be $R/m \longrightarrow S/m$; n be m'/m; $\varphi \longrightarrow \psi$ be $R/m' \longrightarrow S'/m'$. Let $\tilde{\psi}$ be as in Definition 5.7 and set $\varphi_1 = \varphi_0(n \longleftarrow \psi)$; $\psi_1 = \psi_0(M \longleftarrow \tilde{\psi})$, so that $\varphi_1 \longrightarrow \psi_1 \in \mathbb{R}_1$ by (5.7.1). Let T be $R(m \longleftarrow \psi_1)$. Then

$$S'/m = (R/m)(m'/m \longleftarrow \psi) = \varphi_0(n \longleftarrow \psi) = \varphi_1$$
$$T = R(m' \longleftarrow \psi)(m \longleftarrow \psi_1) = S'(m \longleftarrow \psi_1).$$

So $S' \Longrightarrow T$ using a rule in $\mathbb{R}_1$. On the other hand, suppose $p \in M$, so that $\psi_0/p \longrightarrow \tilde{\psi} \in \mathbb{R}_2$ by (5.7.2). Then

$$S/(m \cdot p) = (S/m)/p = \psi_0/p$$
$$T = R(m \longleftarrow \psi_0(M \longleftarrow \tilde{\psi})) = S(m \cdot M \longleftarrow \tilde{\psi}).$$

So $S \xRightarrow{|M|} T$ using rules in $\mathbb{R}_2$. ∎

6. Parameters and Rule-Schemata

In order to apply the theorem that every unequivocal closed SRS is Church-Rosser, we must consider ways to specify SRSs and ways to ascertain whether they are unequivocal and closed from such specifications. It is impossible to inspect every rule in an infinite set of rules; it is impractical to continue to specify sets of rules informally as in our example of arithmetic with a conditional operator. We need a flexible general method for specifying sets of rules that can generate important infinite sets from finite specifications and that lets us verify properties of the sets of rules by inspecting these specifications. Such a method is introduced in this section.

In practice it is usually easy to ascertain whether SRSs are unequivocal, so we will concern ourselves only with whether they are closed. In §5 we illustrated the definition by laboriously checking that arithmetic with conditional expressions forms a closed SRS. The crux of the matter was simple: If $\varphi_0 \longrightarrow \psi_0 \in \mathbb{R}$ and $\varphi \longrightarrow \psi \in \mathbb{R}$ is applicable at n in Dom φ_0 with $n \neq (\ )$, then $\varphi_0 \longrightarrow \psi_0$ has the <u>form</u>

(*) $\quad\bar{C}(\underset{\sim\sim}{\overline{\text{true}}}, \psi_0, \omega) \longrightarrow \psi_0$

or

(**) $\quad\bar{C}(\underset{\sim\sim}{\overline{\text{false}}}, \omega, \psi_0) \longrightarrow \psi_0$

and the new rule $\varphi_1 \longrightarrow \psi_1$ has the same form. Since <u>every</u> rule of that form is in $\mathbb{R}$, $\varphi_1 \longrightarrow \psi_1$ is in $\mathbb{R}$.

This situation is represented pictorially in Figure 2-12, where two "schemata" represent the two sets of rules in (*) and (**). The hatched triangles in the figure represent "arbitrary trees": more

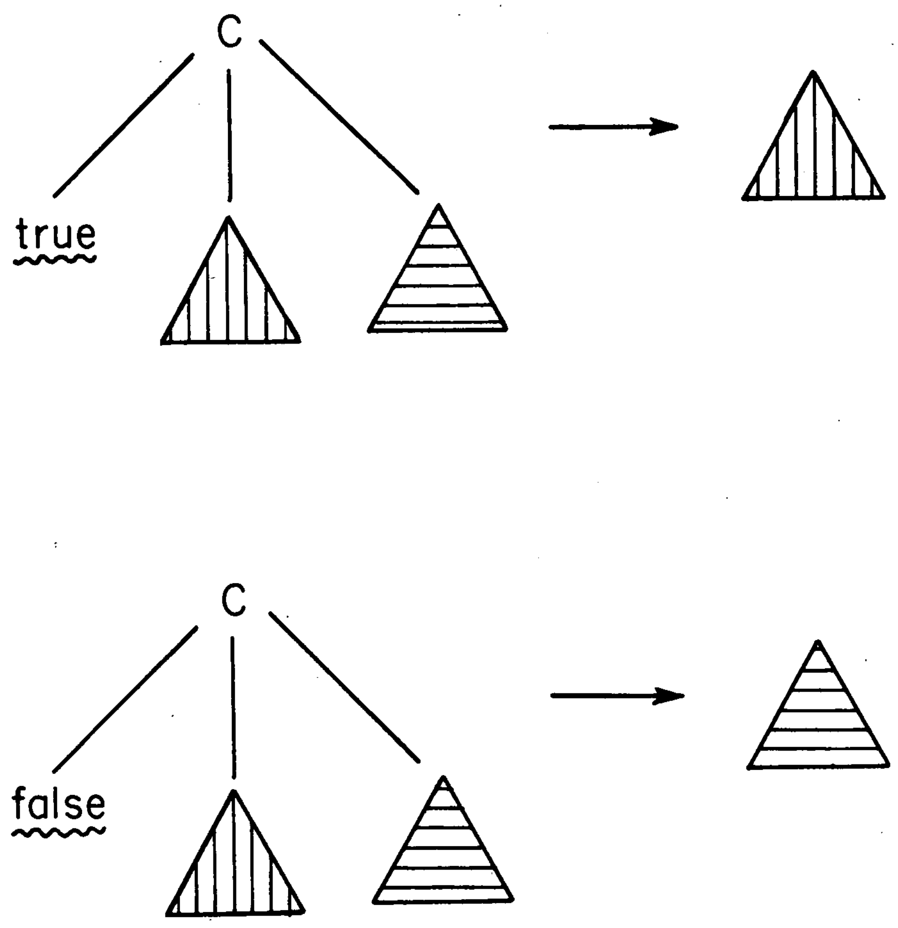


Figure 2-12. Schemata for the rules governing conditional expressions.

precisely, we are to substitute any operator-operand structure for the vertically hatched triangle (wherever it appears) and any operator-operand structure for the horizontally hatched triangle (wherever it appears). The set of all "instances" of a schema obtainable in this way is the set of rules generated by the schema. A set L of schemata generates a set $\mathbb{R}_L$ of rules.

Rather than use pictures as schemata, we will introduce special symbols. Let u and v be new symbols, neither numerals nor arithmetic operators. The pair of trees

(1) $\bar{C}(\overline{\underset{\sim\sim\sim}{\text{true}}}, \bar{u}, \bar{v}) \longrightarrow \bar{u}$

is a schema generating all the rules of form (*). The pair of trees

(2) $\bar{C}(\overline{\underset{\sim\sim\sim}{\text{false}}}, \bar{u}, \bar{v}) \longrightarrow \bar{v}$

is a schema generating all the rules of form (**). The new symbols are "parameters" for which trees may be substituted to form rules. Associated with each parameter is a "domain" – the set of trees that may be substituted for that parameter in schemata.

For the rest of this section, we suppose that sets V and U are given and that to each $u \in U$ there has been assigned a subset D_u of V_*. Members of U will be called parameters; each parameter u has domain D_u. In some examples we will have $U \cap V = \emptyset$, but in others we will have $U \subseteq V$. The results of this section will be stated in such a way that both possibilities (and any other relation between U and V) are allowed.

A schema for generating rules ("rule-schema") is to be a pair of trees in $(V \cup U)_*$. Three of the formal properties of the schemata (1) and (2) will be required of any rule-schema $R \longrightarrow S$:

(a) Parameters occur only at leaves (defined as in (4.7.6) of R and S.

(b) No parameter occurs twice in R.

(c) Every parameter that occurs in S also occurs in R.

(6.1) Definition. A template is any $R \in (V \cup U)_*$ such that $R^{-1}(U)$ is a set of leaves in R. A rule-schema is any pair $R \longrightarrow S$ of templates such that

(1) $$(\forall u \in U)(|R^{-1}(u)| \leqslant 1)$$

(2) $$(\forall u \in U)(S^{-1}(u) \neq \emptyset \text{ implies } R^{-1}(u) \neq \emptyset).$$

To form an "instance" of a rule schema, we must choose a specific tree in the domain of each parameter involved, then replace subtrees of the form $\bar{u}$ for $u \in U$ by the appropriate chosen trees. This is the familiar method of substituting "expressions" for "variables" in algebra and logic.

(6.2) Definition. Let $R \longrightarrow S$ be a rule-schema. If $(u_0, \ldots, u_{K-1})$ with $K \in \mathbb{N}$ is a list of the members of $\{u \in U \mid R^{-1}(u) \neq \emptyset\}$ and $T_k \in D_{u_k}$ for each $k < K$, then $\varphi \longrightarrow \psi$ is an instance of $R \longrightarrow S$, where

(1) $$\varphi = R(R^{-1}(u_0) \longleftarrow T_0) \ldots (R^{-1}(u_{-1}) \longleftarrow T_{-1})$$
$$\psi = S(S^{-1}(u_0) \longleftarrow T_0) \ldots (S^{-1}(u_{-1}) \longleftarrow T_{-1}).$$

If L is a set of rule-schemata, then

(2) $$\mathbb{R}_L = \{\varphi \longrightarrow \psi \in V_* \times V_* \mid (\exists R \longrightarrow S \in L)$$
$$(\varphi \longrightarrow \psi \text{ is an instance of } R \longrightarrow S)\}.$$

Given an instance $\varphi \longrightarrow \psi$ of a rule-schema $R \longrightarrow S$, there is a natural way to define a residue map for $\varphi \longrightarrow \psi$. Let $n \in \mathrm{Dom}\ \varphi$. There is at most one node $m \in R^{-1}(U)$ such that $m\ \underline{\mathrm{anc}}\ n$ because $R^{-1}(U)$ is an independent set of nodes. If there is such a node m let $u = Rm$ and let $T = \varphi/m$, so that $T \in D_u$ in the previous definition. For each node p in S with $Sp = u$, we have $\psi/p = T$ because T was substituded for u at p also. Therefore

$$\psi/(p\cdot(n/m)) \doteq T/(n/m) = (\varphi/m)/(n/m) = \varphi/n$$

by $\psi/p = T$ and two applications of cancellation (4.9.1). Thus $p\cdot(n/m)$ has the basic property of residue nodes: a copy of φ/n appears at $p\cdot(n/m)$ in ψ. All such $p\cdot(n/m)$ will be considered residues of n. (In particular, if n should have no ancestors in $R^{-1}(U)$, then n has no residues in ψ.)

(6.3) Lemma. Let $\varphi \longrightarrow \psi$ be an instance of a rule-schema $R \longrightarrow S$. Define a map from $\mathrm{Dom}\ \varphi$ into the set of subsets of $\mathbb{N}^*$ by

$$r(n) = \{p\cdot(n/m) \mid m \in R^{-1}(U)\ \&\ m\ \underline{\mathrm{anc}}\ n\ \&\ Sp = Rm\}.$$

Then r is a residue map for $\varphi \longrightarrow \psi$.

Proof: Let $n \in \mathrm{Dom}\ \varphi_0$. We must show that $r(n) \subseteq \mathrm{Dom}\ \psi$ and that $\psi/q = \varphi/n$ for each $q \in r(n)$ in order to verify (5.3.1) in the definition of residue maps. If $r(n) = \emptyset$ these are trivial, so we may assume $r(n) \neq \emptyset$. Some $m \in R^{-1}(U)$ has $m\ \underline{\mathrm{anc}}\ n$ and

$$r(n) = \{p\cdot(n/m) \mid Sp = Rm\}.$$

Let $u = Rm$ and $T = \varphi/m$, so that $T \in D_u$ and ψ has the form

$$\psi = S(\ldots)(S^{-1}(u) \longleftarrow T)(\ldots)$$

where (...) represents substitutions at nodes that are independent of $S^{-1}(u)$. Since $n/m \in \text{Dom } T$ by $n \in \text{Dom } \varphi$, each $p \in S^{-1}(u)$ has $p \cdot (n/m) \in \text{Dom } \psi$. By $\psi/p = T$ and two applications of cancellation (4.9.1), we have

$$\psi/(p \cdot (n/m)) = T/(n/m) = (\varphi/m)/(n/m) = \varphi/n .$$

This completes the verification of (5.3.1).

We must verify (5.3.2): independent nodes have independent residues. This is implied by

$$(\forall\, n, n' \in \text{Dom } \varphi)\,(\forall\, q \in r(n))\,(\forall\, q' \in r(n'))\,(q \ \underline{\text{anc}}\ q' \text{ implies } n \ \underline{\text{anc}}\ n'),$$

which we will now prove. Suppose $q \in r(n)$; $q' \in r(n')$; $q \ \underline{\text{anc}}\ q'$. For some $m \ \underline{\text{anc}}\ n$ and some $m' \ \underline{\text{anc}}\ n'$ we have $m, m' \in R^{-1}(U)$. For some $p \in S^{-1}(Rm)$ and some $p' \in S^{-1}(Rm')$ we have $q = p \cdot (n/m)$ and $q' = p' \cdot (n'/m')$. By $q \ \underline{\text{anc}}\ q'$ we have $p \cdot \mathbb{N}^* \cap p' \cdot \mathbb{N}^* \neq \emptyset$, so (4.6.4) implies that NOT($p \perp p'$). But p and p' are leaves in S, so we must have $p = p'$. From $q \ \underline{\text{anc}}\ q'$ it now follows that $n/m \ \underline{\text{anc}}\ n'/m'$. But $p = p'$ also implies that $S^{-1}(Rm) \cap S^{-1}(Rm') \neq \emptyset$, so that Rm and Rm' are the same parameter. In the definition of rule-schemata we have $m = m'$ by (6.1.1), so that $n/m \ \underline{\text{anc}}\ n'/m$. This implies that $n \ \underline{\text{anc}}\ n'$, as desired. ∎

Replacing parts of an instance $\varphi \longrightarrow \psi$ of a schema will often lead to another instance of the same schema. If we replace only parts of φ that were substituted in for parameters, if these replacements do not result in substitutions from outside the domains of parameters, and if the appropriate replacements are made at residues in ψ also, then the result $\overline{\varphi} \longrightarrow \overline{\psi}$ should still be an instance of the schema.

(6.4) Lemma. Let a rule $\varphi \longrightarrow \psi$ be an instance of a rule schema $R \longrightarrow S$. Let N be an independent subset of Dom φ and let $Q[n] \in V_*$ for each $n \in N$. For any listing $(n_0, \ldots, n_{K-1})$ of the members of N, set

(1) $$\overline{\varphi} = \varphi(n_0 \leftarrow Q[n_0]) \ldots (n_{-1} \leftarrow Q[n_{-1}])$$
$$\overline{\psi} = \psi(r(n_0) \leftarrow Q[n_0]) \ldots (r(n_{-1}) \leftarrow Q[n_{-1}]),$$

where r is the residue map from the previous lemma. For each $m \in R^{-1}(U)$, let $(N_0^m, \ldots, N_{K_m-1}^m)$ be a listing of the members of N descended from m. Suppose

(2) $$(\forall k < K)(\exists \mathbf{m} \in R^{-1}(U))\ (m \ \underline{\text{anc}}\ n_k)$$

(3) $$(\forall m \in R^{-1}(U))[(\varphi/m)(N_0^m/m \leftarrow Q[N_0^m]) \ldots (N_{-1}^m/m \leftarrow Q[N_{-1}^m]) \in D_{Rm}].$$

Then $\overline{\varphi} \longrightarrow \overline{\psi}$ is an instance of $R \longrightarrow S$.

Proof: Let $(\mu_0, \ldots, \mu_{J-1})$ be any listing of the members of $R^{-1}(U)$ and let $u_j = R\mu_j$ for each $j < J$. Write N_k^j rather than $N_k^{\mu_j}$ for N_k^m when $m = \mu_j$. For each $j < J$, set

$$\overline{T}_j = (\varphi/\mu_j)(N_0^j/\mu_j \leftarrow Q[N_0^j]) \ldots (N_{-1}^j/\mu_j \leftarrow Q[N_{-1}^j]),$$

so that $\overline{T}_j \in D_{u_j}$ by (3). We therefore have an instance $\tilde{\varphi} \longrightarrow \tilde{\psi}$ of of $R \longrightarrow S$, where

(4) $$\tilde{\varphi} = R(\mu_0 \leftarrow \overline{T}_0) \ldots (\mu_{-1} \leftarrow \overline{T}_{-1})$$
$$\tilde{\psi} = S(S^{-1}(u_0) \leftarrow \overline{T}_0) \ldots (S^{-1}(u_{-1}) \leftarrow \overline{T}_{-1}).$$

We will show that $\tilde{\varphi} = \overline{\varphi}$ and $\tilde{\psi} = \overline{\psi}$.

First we note that (2) establishes a bijection between indices $i<K$ and pairs of indices (j, k) with $j < J$ and $k < K_{m_j}$, such that $n_i = N_k^j$.

The listing

$$(N_0^0, \dots, N_{-1}^0, \dots \dots, N_0^{J-1}, \dots, N_{-1}^{J-1})$$

is a permutation of $(n_0, \dots, n_{K-1})$, and such permutations may be ignored by commutativity (4.7.4). In (1) we have

$$\begin{aligned}\bar{\varphi} &= \varphi(N_0^0 \longleftarrow Q[N_0^0]) \dots (N_{-1}^{J-1} \longleftarrow Q[N_{-1}^{J-1}]) \\ &= [R(\mu_0 \longleftarrow \varphi/\mu_0) \dots (\mu_{J-1} \longleftarrow \varphi/\mu_{J-1})] \\ &\qquad (N_0^0 \longleftarrow Q[N_0^0]) \dots (N_{-1}^{J-1} \longleftarrow Q[N_{-1}^{J-1}]) .\end{aligned}$$

We write each N_k^j as $\mu_j \cdot (N_k^j/\mu_j)$ and use commutativity (4.7.4) and associativity (4.7.2) to rearrange this as

$$\begin{aligned}\bar{\varphi} = R(\mu_0 &\longleftarrow (\varphi/\mu_0)(N_0^0/\mu_0 \longleftarrow Q[N_0^0]) \dots \\ &\qquad (N_{-1}^0/\mu_0 \longleftarrow Q[N_{-1}^0])) \dots \\ (\mu_{J-1} &\longleftarrow (\varphi/\mu_{J-1})(N_0^{J-1}/\mu_{J-1} \longleftarrow Q[N_0^{J-1}]) \dots \\ &\qquad (N_{-1}^{J-1}/\mu_{J-1} \longleftarrow Q[N_{-1}^{J-1}])) .\end{aligned}$$

Comparison with (4) shows that $\tilde{\varphi} = \bar{\varphi}$.

Whenever $n_i = N_k^j$, the residue map from Lemma 6.3 evaluates to

$$r(n_i) = S^{-1}(u_j) \cdot (N_k^j/\mu_j) .$$

Applying this to (1) and rearranging by commutativity (4.7.4), we get

$$\begin{aligned}\bar{\psi} &= \psi(S^{-1}(u_0) \cdot (N_0^0/\mu_0) \longleftarrow Q[N_0^0]) \dots \\ &\qquad (S^{-1}(u_{J-1}) \cdot (N_{-1}^{J-1}/\mu_{J-1}) \longleftarrow Q[N_{-1}^{J-1}]) \\ &= [S(S^{-1}(u_0) \longleftarrow \varphi/\mu_0) \dots (S^{-1}(u_{J-1}) \longleftarrow \varphi/\mu_{J-1})] \\ &\qquad (S^{-1}(u_0) \cdot (N_0^0/\mu_0) \longleftarrow Q[N_0^0]) \dots \\ &\qquad (S^{-1}(u_{J-1}) \cdot (N_{-1}^{J-1}/\mu_{J-1}) \longleftarrow Q[N_{-1}^{J-1}]) .\end{aligned}$$

Letting $S^{-1}(u_j)$ play the role that μ_j played in the proof that $\tilde{\varphi} = \overline{\varphi}$, we can now repeat that reasoning to show that $\tilde{\psi} = \overline{\psi}$. ∎

At last we have our general method for specifying closed SRSs: generate the rules from rule-schemata and use the above lemma to establish closure. More precisely, we have the following theorem:

(6.5) Rule-Schemata Theorem. Let L be a set of rule-schemata and let $\mathfrak{C}$ be an SRS of the form $\mathfrak{C} = (V, \mathbb{F}, \Longrightarrow, \mathbb{R}_L)$. Suppose that each $\varphi_0 \longrightarrow \psi_0 \in \mathbb{R}_L$ has

(1) $(\exists ! R \longrightarrow S \in L)(\varphi_0 \longrightarrow \psi_0$ is an instance of $R \longrightarrow S)$

(2) $(\forall n \in \text{Dom } \varphi_0)(\forall \varphi \longrightarrow \psi \in \mathbb{R}_L)$
$[(n \neq (\ ) \ \& \ \varphi_0/n = \varphi)$ implies
$(\exists m \in R^{-1}(U))(m \ \underline{\text{anc}} \ n \ \& \ (\varphi_0/m)(n/m \longleftarrow \psi) \in D_{Rm})]$,

where R is the left half of the rule-schema $R \longrightarrow S$ with $\varphi_0 \longrightarrow \psi_0$ as an instance. Then $\mathfrak{C}$ is closed.

Proof: Since each rule is an instance of a unique rule-schema, residue maps may be assigned to the rules by Lemma 6.3. We must verify the properties of closed systems in Definition 5.4. Suppose $\varphi_0 \longrightarrow \psi_0$, $\varphi \longrightarrow \psi \in \mathbb{R}_L$; $n \in \text{Dom } \varphi_0$; $n \neq (\ )$; $\varphi_0/n = \varphi$. Set

$$\varphi_1 = \varphi_0(n \longleftarrow \psi)$$
$$\psi_1 = \psi_0(r[\varphi_0, \psi_0](n) \longleftarrow \psi).$$

We will apply Lemma 6.4 with the independent subset $N = \{n\}$ of $\text{Dom } \varphi_0$ and the tree $Q[n] = \psi$ assigned to n. By (2), some $m \in R^{-1}(U)$ has $m \ \underline{\text{anc}} \ n$ and $(\varphi_0/m)(n/m \longleftarrow Q[n]) \in D_{Rm}$, so (6.4.2) and (6.4.3) do hold. Therefore $\varphi_1 \longrightarrow \psi_1$ is an instance of $R \longrightarrow S$ by Lemma 6.4. We do have $\varphi_1 \longrightarrow \psi_1 \in \mathbb{R}_L$, as required by (5.4.1).

We must show (5.4.2): any p with $p \perp n$ in Dom φ_0 has

$$r[\varphi_1, \psi_1](p) = r[\varphi_0, \psi_0](p).$$

In Lemma 6.3 we defined $r[\varphi_1, \psi_1](p)$ by an equation involving a schema $R_1 \longrightarrow S_1$ of which $\varphi_1 \longrightarrow \psi_1$ is an instance. But $R_1 \longrightarrow S_1 = R \longrightarrow S$ by (1), so $r[\varphi_1, \psi_1](p)$ is defined by the same equation as $r[\varphi_0, \psi_o](p)$. ∎

The following chapters will apply the Rule-Schemata Theorem (6.5) and the Main Theorem (5.6) to show that various SRSs are Church-Rosser. The condition (6.5.2) is not very difficult to test in practice. Although only a very special case of Lemma 6.4 has been used so far, the full generality will be needed to establish additional results in the application areas.

One application for this theory that will not be treated in detail by later chapters should be mentioned here. Hindley [personal communication] has found conditions that imply the Church-Rosser property for systems of combinatory logic. When applied to rule-schemata in the special format used in combinatory logic, these conditions insure that the logical system is unequivocal and closed when expressed as an SRS; the proof that this is so is a specialization of the proof of the Rule-Schemata Theorem (6.5). The Church-Rosser property then follows from the Main Theorem (5.6).

Hindley's conditions were inspired by Sanchis' proof of the Church-Rosser property for one specific combinatory system [47, §2, Lemma 1]. Implicit in Sanchis' proof are broad outlines of the proofs of (6.5) and (5.6), as restricted to this one set of rule-schemata and the set of rules it generates. The simplicity of this particular system made it appropriate to ignore all details and to carry out the proofs on an intuitive level, as in our preliminary discussion motivating the closure concept.

CHAPTER 3

APPLICATIONS TO RECURSIVE DEFINITIONS

This chapter applies the theory of subtree replacement systems to recursive definitions. In §7 we extend McCarthy's calculus for recursive definitions by introducing a choice between two types of parameter that corresponds to the choice between call-by-value and call-by-name in ALGOL 60. We show that formal calculations using recursive definitions define singlevalued partial functions despite the nondeterminism of the evaluation algorithm.

Recursive definitions are commonly written as sets of equations, and it is natural to assume that the functions they define are solutions to the equations. In §8 we formulate this idea precisely and prove that it is correct. When restricted to definitions that use only call-by-value, our result is closely related to Kleene's "first recursion theorem" [26, §66, Thm. 26]: any partial recursive functional $\mathfrak{F}$ has a unique minimal fixed point defined by formal calculations using the set of equations that specifies $\mathfrak{F}$. (A detailed comparison is given at the end of §8.)

7. An Algorithmic Explanation

A function may be defined by an equation of the form

$$f(x) := \ldots$$

where the ... is built up from x and constants and previously defined functions. It is not so clear how the factorial is "defined" by

(1) $f(x) := \underset{\sim}{\text{if}}\ x = 0\ \underset{\sim}{\text{then}}\ 1\ \underset{\sim}{\text{else}}\ x \times f(x-1)$.

The circularity of this definition renders it useless without some further explanation. Following Morris [37, Chap. 3], we recognize two sorts of explanation for recursive definitions. An algorithmic explanation tells how to use (1) in formal computations. For each argument ξ there is a procedure for transforming the expression $f(\xi)$ to a numerical value η. A careful algorithmic explanation for recursive definitions with conditional operators is given by Manna and Pnueli [32, §3]. A semantic explanation treats (1) as an equation that may or may not be solved by any particular choice for the function f. The function specified by (1) is defined to be one of the solutions to the equation.

This chapter uses an elaboration of the McCarthy calculus for recursive definitions [34, p. 42] to deal with two basic problems posed by algorithmic and semantic explanations. The Main Theorem (5.6) leads to satisfactory and intimately related solutions for both. We treat the "functionality" problem in this section and the "validity" problem in the next section.

First we review McCarthy's concept of recursive definitions from [34]. A recursive definition is a set of equations of the form

$$f(a_0, \ldots, a_{k-1}) := e ,$$

where f is a "function letter" used to name the function we are defining, and e is any expression built up from function letters, names of given functions that have already been defined, the $\underset{\sim}{\text{if}} \ldots \underset{\sim}{\text{then}} \ldots \underset{\sim}{\text{else}} \ldots$ construction, variables chosen from among $a_0, \ldots, a_{k-1}$, and constants. In [34] there can be only one equation beginning with f for each function letter f, and the arguments $a_0, \ldots, a_{k-1}$ must all be variables. These restrictions will be slightly relaxed below, but they can be imposed here for the sake of simplicity in this preliminary sketch.

Our example (1) is a recursive definition if we consider the data space to be the nonnegative integers and the given functions to include the test for equality (=), multiplication (X), and subtraction (-). The general concept is not restricted to primitive recursion or even to integer data.

Each equation

$$f(a_0, \ldots, a_{k-1}) \ := \ e$$

in a recursive definition generates a set $\mathbb{R}[f]$ of *rules* of the form

$$f(A_0, \ldots, A_{k-1}) \longrightarrow E$$

where each A_i is an expression that may be substituted for a_i and E is the result of performing all these substitutions in e. For each given function g, we also have a set $\mathbb{R}[g]$ consisting of all rules

$$g(\xi_0, \ldots, \xi_{-1}) \longrightarrow \eta$$

where η is a constant representing the value of g for the constants $\xi_0, \ldots, \xi_{-1}$. For conditional expressions we also have the set $\mathbb{R}[C]$ of all rules

$$\underline{\text{if}}\ \underline{\text{true}}\ \underline{\text{then}}\ E\ \underline{\text{else}}\ E' \longrightarrow E$$

$$\underline{\text{if}}\ \underline{\text{false}}\ \underline{\text{then}}\ E\ \underline{\text{else}}\ E' \longrightarrow E'.$$

Expressions are to be evaluated by matching subexpressions against the left sides of rules, then replacing these subexpressions by the right sides of the rules.

We have sketched the essence of the nondeterministic algorithm used by McCarthy for evaluating expressions. A formalization in terms of subtree replacement systems (and a more specific comparison with [34]) will be given later. For the moment it is enough to agree that we can associate with each recursive definition an algorithm for evaluating expressions. Given such an assignment of algorithms to recursive definitions, we have an algorithmic explanation of these definitions: a definition L defines a function Lf, for each function letter f, such that

$$Lf(\xi_0, \ldots, \xi_{-1}) = \eta$$

whenever $\xi_0, \ldots, \xi_{-1}$ and η are constants such that the formal expression $f(\xi_0, \ldots, \xi_{-1})$ evaluates to η. (This means that the algorithm has at least one computation that begins with $f(\xi_0, \ldots, \xi_{-1})$ and halts at η, not that every computation that begins at $f(\xi_0, \ldots, \xi_{-1})$ must halt at η.)

In general, Lf will be only a partial function: for some choices of $\xi_0, \ldots, \xi_{-1}$ there may be no computations by the algorithm that halt at a value η. We do expect Lf to be a function: for each $(\xi_0, \ldots, \xi_{-1})$ there should be at most one η such that $f(\xi_0, \ldots, \xi_{-1})$ can be evaluated to η.

When an algorithmic explanation is nondeterministic (or involves asynchronous parallel processing), the singlevaluedness of the relation between input and output needs to be shown. The general remarks motivating this functionality problem in §3 are applicable here. They can be fleshed out with examples where ingenious implementations of recursive definitions could save a great deal of time [35, §2.2] or space [49].

Now we can proceed with the formal development. For the rest of this chapter, let $\mathbb{D}$ be the data space of objects on which we compute. For each k in the set $\mathbb{P}$ of positive integers, we consider any set G_k of given functions. Each $g \in G_k$ is a total function $g : \mathbb{D}^k \longrightarrow \mathbb{D}$, whose computations are taken for granted. In applications the given functions are the hardware capabilities, library subroutines, and so on that the programmer can use as black boxes. The examples in this section will use $\mathbb{D} = \mathbb{N}$ and given functions from arithmetic. The actual results apply to arbitrary data spaces with at least two members.

In order to write recursive definitions of new functions in terms of old ones and each other, let F_k be an infinite set of function letters for each $k \in \mathbb{P}$. The sets F_k are to be disjoint from each other, from $\mathbb{D}$, and from the sets G_j. Finally, let W and X be infinite sets that are disjoint from each other and from all the previous sets. These sets will supply the "variables" in the intuitive sketch of the McCarthy calculus. For reasons that will be clear shortly, we say that W is the set of call-by-name parameters and X is the set of call-by-value parameters.

The given functions, function letters, parameters, and data combine to form a total vocabulary V with a rank function $\rho : V \longrightarrow \mathbb{N}$, as follows:

$$V = \mathbb{D} \cup W \cup X \cup F \cup G \qquad \text{where} \qquad F = \bigcup_{k \in \mathbb{P}} F_k$$

$$G = \bigcup_{k \in \mathbb{P}} G_k$$

$$\rho(f) = k \quad \text{for} \quad f \in F_k \cup G_k$$

$$\rho(a) = 0 \quad \text{for} \quad a \in \mathbb{D} \cup W \cup X .$$

The forest $V_{\#} = V_{\#}[\rho]$ defined by (4.11) consists of all operator-operand structures for well-formed expressions constructed from this vocabulary. The surface syntax in implementations is of no concern to us here. For example, the pair of trees involved in defining factorials by (1) is shown in Figure 3-1, which assumes $C \in F_3$; $=, \times, \dot{-} \in G_2$; $f \in F_1$; $x \in X$; $\mathbb{D} = \mathbb{N}$. (Subtraction is made total by the usual expedient of setting $\xi \dot{-} \eta = 0$ when $\xi < \eta$.) We will retain the overlines in the algebraic nomenclature (4.10) in order to prevent confusion between the <u>tree</u> $\bar{+}(\bar{4}, \bar{6})$ with three nodes and the <u>value</u> $+(4, 6) = 10$ of the function $+$ at the argument string $(4, 6)$. Thus $+(4, 6) = +(5, 5)$ but $\bar{+}(\bar{4}, \bar{6}) \neq \bar{+}(\bar{5}, \bar{5})$.

The pair of trees in Figure 3-1 is to be a rule-schema in the sense of Definition 6.1. The set $U = W \cup X$ is to be the set of parameters. The domains of the parameters are specified as follows.

(7.1) $(\forall u \in W)(D_u = V_{\#})$ call-by-name

$(\forall x \in X)(D_x = \mathbb{D}_{\#})$ call-by-value

Call-by-name is used in the definition of the conditional operator needed for our definition of factorials. To compute conditionals, we use any two schemata of the form

(2) $\overline{C}(\overline{yes}, \bar{u}, \bar{v}) \longrightarrow \bar{u}$

$\overline{C}(\overline{no}, \bar{u}, \bar{v}) \longrightarrow \bar{v}$

where

(3) $C \in F_3$ & $u, v \in W$ & $yes, no \in \mathbb{D}$ & $u \neq v$ & $yes \neq no$.

In our example with $\mathbb{D} = \mathbb{N}$ it would be natural to use 1 for yes and 0 for no. In any situation with $|\mathbb{D}| \geqslant 2$ we can form two schemata as in (2) such that (3) holds. Note that conditional expressions will be

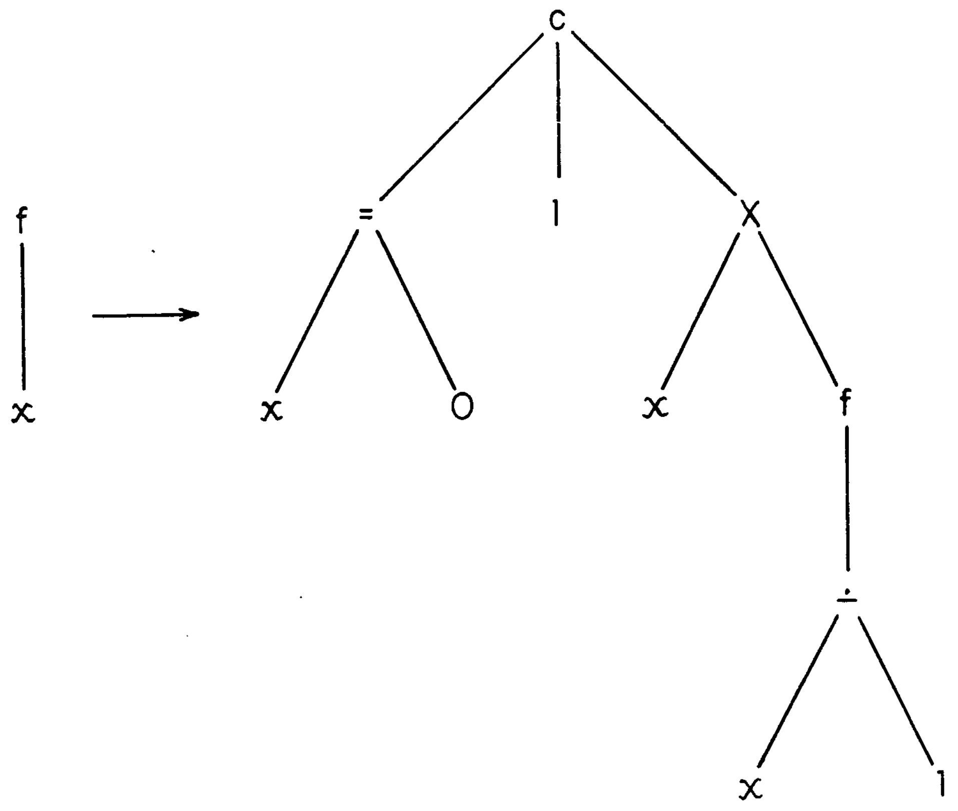


Figure 3-1. Operator-operand structures in the definition of factorials.

evaluated in the proper way by the set of rules generated by (2). To evaluate $\bar{C}(P,A,B)$ we must first evaluate P, and $\bar{C}(P,A,B)$ should have no value unless P can evolve to either $\overline{yes}$ or $\overline{no}$. If P can evolve to $\overline{yes}$ we should forget about B and attempt to evaluate A. The rule

$$\bar{C}(\overline{yes},A,B) \longrightarrow A$$

expresses this. On the other hand, if P can evolve to $\overline{no}$, we should apply

$$\bar{C}(\overline{no},A,B) \longrightarrow B$$

after evaluating P. If we think of (2) as the definition of a "procedure" C and we think of $C(\overline{yes},A,B)$ as a call on C with A as the "actual parameter" in place of the "formal parameter" u, then A and B should be passed <u>unevaluated</u>, as in ALGOL 60 call-by-name [38, §4.7.3.2]. If A has a value but B does not, then it is incorrect as well as inefficient to try to evaluate all subexpressions of $\bar{C}(\overline{yes},A,B)$ before evaluation of the whole expression. In Figure 3-1, on the other hand, there is no need to call the procedure f before evaluating A in $\bar{f}(A)$. Substituting A for x would lead to three independent evaluations of A (one for each occurrence of x in the procedure body) if A does not have the value 0. Because we used a call-by-value parameter x in Figure 3-1, the only way to evaluate $\bar{f}(A)$ is to let A evolve to $\bar{\xi}$ for some $\xi \in \mathbb{D}$ and then use the rule

$$\bar{f}(\bar{\xi}) \longrightarrow \bar{C}(\bar{=}(\bar{\xi},\bar{0}),\bar{1},\bar{\times}(\bar{\xi},\bar{f}(\bar{\dot{-}}(\bar{\xi},\bar{1}))))$$

to enter the procedure f, as in ALGOL 60 call-by-value [38, §4.7.3.1]. If A has no value, then neither does $\bar{f}(A)$.

All instances of Figure 3-1 and (2), together with all the rules like $\bar{=}(\bar{2}, \bar{0}) \longrightarrow \overline{no}$ and $\bar{-}(\bar{2}, \bar{1}) \longrightarrow \bar{1}$ that describe the given functions, define the set $\mathbb{R}$ of rules in an SRS $\mathfrak{C} = (V, V_{\#}, \Longrightarrow, \mathbb{R})$. For any $\xi \in \mathbb{N}$, the tree $\bar{f}(\bar{\xi})$ should have a unique normal form $\bar{\eta}$ such that $\xi\,! = \eta$, even though the only sequencing is that implied by the use of X or $\mathbb{D}$ rather than W at some of the argument positions in the schemata. Like parallel program schemata [25, § 1], $\mathfrak{C}$ has only permissive control: at any point in the computation, the next operation to be performed may be any one of the operations permitted to start next. Whenever a subtree of the working tree has the form $\bar{C}(\bar{\xi}, A, B)$ for $\xi \in \{yes, no\}$, we may apply the appropriate rule to this subtree, or we may apply a rule elsewhere.

It is easy to write a set of rule-schemata that does <u>not</u> define a function, such as $\{\bar{f}(\bar{x}) \longrightarrow \bar{3},\ \bar{f}(\bar{u}) \longrightarrow \bar{4}\}$. We formalize conditions under which sets of schemata could reasonably be expected to define functions. The left half R of each schema $R \rightarrow S$ should be of the form $\bar{f}(\bar{a}_0, \dots, \bar{a}_{k-1})$ with $f \in F_k$ and $a_i \in \mathbb{D} \cup W \cup X$ for all $i < k$. This is equivalent to saying that $R \in V_{\#}$ with $R(\ ) \in F$ and $\mathrm{Dom}\, R \subseteq \mathbb{N}^0 \cup \mathbb{N}^1$. (Recall that $\mathbb{N}^J$ is the set of all strings J long of nonnegative integers.) As in our schemata for conditionals, there may be two schemata $R \longrightarrow S$ with the same root label $R(\ )$, but then there must be differences between constant arguments that will separate the instances of the two schemata.

(7.2) <u>Definition</u>. A <u>recursive definition</u> (<u>RD</u>) is any set L of rule-schemata in $V_{\#} \times V_{\#}$ such that

(1) $(\forall R \longrightarrow S \in L)(R(\ ) \in F \quad \& \quad \mathrm{Dom}\, R \subseteq \mathbb{N}^1 \cup \mathbb{N}^0)$

(2) $(\forall R \longrightarrow S,\ R' \longrightarrow S' \in L)([R \longrightarrow S \neq R' \longrightarrow S' \quad \& \quad R(\ ) = R'(\ )]$

implies $(\exists j \in \mathbb{N})[R(j), R'(j) \in \mathbb{D} \quad \& \quad R(j) \neq R'(j)])$.

In defining rule-schemata in §6, we required that every parameter occurring in S should occur in R when $R \longrightarrow S$ is a rule-schema (6.1.2). Therefore no two rules have the same left half in the set $\mathbb{R}_{\{R\to S\}}$ of instances of any schema $R \to S$, where instances are defined by (6.2). By (2), $\varphi \neq \varphi'$ whenever $\varphi \longrightarrow \psi, \varphi' \longrightarrow \psi'$ are instances of $R\longrightarrow S, R'\longrightarrow S' \in L$ and $R'\longrightarrow S' \neq R\longrightarrow S$. Therefore no two rules in $\mathbb{R}_L$ have the same left half.

(7.3) Definition. Let L be an RD and let $\mathbb{R}_L$ be the set of all instances of members of L. Define an SRS $\mathfrak{C}_L$ by

(1) $\mathbb{R}_{giv} = \{\bar{g}(\bar{\xi}_0, \ldots, \bar{\xi}_{-1}) \longrightarrow \bar{\eta} \mid g \in G \ \& \ g(\xi_0, \ldots, \xi_{-1}) = \eta\}$

(2) $\mathfrak{C}_L = (V, V_{\#}, \Longrightarrow, \mathbb{R}_{giv} \cup \mathbb{R}_L)$.

For each $f \in F$, the set of ordered pairs

$$\Phi = \{\langle \xi, \eta\rangle \in \mathbb{D}^{\rho(f)} \times \mathbb{D} \mid \bar{f}(\bar{\xi}_0, \ldots, \bar{\xi}_{-1}) \overset{*}{\Longrightarrow} \bar{\eta} \text{ in } \mathfrak{C}_L\}$$

will be the partial function represented by f when the definition L is used. Definitions (7.2) and (7.3) extend the McCarthy formalism in two ways. First, they allow members of $\mathbb{D}$ as well as parameters to appear at argument positions in the left sides of schemata, as in the "proper programs" of E.K. Blum [3, §4]. This flexibility renders the original ad hoc treatment of conditionals [34, p. 42, Rule 1] unnecessary. To write an RD using if ... then ... else, we simply add

$$\bar{C}(\overline{yes}, \bar{u}, \bar{v}) \longrightarrow \bar{u}$$

$$\bar{C}(\overline{no}, \bar{u}, \bar{v}) \longrightarrow \bar{v}$$

to the other schemata. Second, our definitions introduce an explicit choice between call-by-value with X and call-by-name with W.

Examples such as [34, p. 37] and the inspiration from LISP suggest that McCarthy meant to use call-by-value exclusively, except of course for the A and B in *if* P *then* A *else* B. Yet his evaluation rules specified call-by-value for given functions [34, p. 42, Rule 2] and call-by-name for function letters [34, p. 42, Rule 3].

The "proper programs" of E.K. Blum [3, §4] in the Herbrand-Gödel-Kleene formalism are a special case of our system. The crucial restriction on "proper programs" is that all parameters are called by value: only numerals may be substituted for variables in formal calculations. Rather than speak of applying rules to trees, Blum speaks of deducing "equations" from sets of "equations." The initial set of "equations" corresponds to our set L of schemata. The rule of "substitution" [3, p. 254] deduces instances of schemata, the rule of "replacement" [3, p. 254] allows our SRS rules to be applied to parts of "equations," and the rule of "counting" [3, p. 254] evaluates given functions. (The assumptions $\mathbb{D} = \mathbb{N}$; $G_1 = \{\text{successor function}\}$; $G_k = \emptyset$ for all $k > 1$ are also made in [3], but they are irrelevant to the functionality problem.)

We have assigned domains to the parameters in (7.1) and defined RDs to be certain sets L of rule-schemata in (7.2). Our algorithmic explanation for RDs is provided by the SRSs $\mathfrak{C}_L$ in (7.3): to arrive at a value η for an expression A by formal calculation is to let A evolve to $\bar{\eta}$ in $\mathfrak{C}_L$. In order to solve the functionality problem we will now apply Chapter 2 to the SRSs $\mathfrak{C}_L$.

For any RD L, the set of rules $\mathbb{R}_{giv} \cup \mathbb{R}_L$ in $\mathfrak{C}_L$ may also be described as

$$\mathbb{R}_{giv} \cup \mathbb{R}_L = \mathbb{R}_{\mathbb{R}_{giv} \cup L}$$

because each $\varphi \longrightarrow \psi \in \mathbb{R}_{giv}$ is a schema with no parameters and with itself as the only instance. The SRS $\mathfrak{C}_L$ does have the form $(V, \mathbb{F}, \Longrightarrow, \mathbb{R}_M)$ for M a set of schemata, so it is appropriate to use the Rule-Schemata Theorem (6.5).

(7.4) Lemma. Let L be an RD. Then $W \cup X$ is a set of parameters and $\mathbb{R}_{giv} \cup L$ is a set of rule-schemata such that

(1) $\mathbb{R}_{giv} \cup \mathbb{R}_L = \mathbb{R}_{\mathbb{R}_{giv} \cup L}$

(2) Each $\varphi_0 \longrightarrow \psi_0 \in \mathbb{R}_{giv} \cup \mathbb{R}_L$ is an instance of just one $R \longrightarrow S \in \mathbb{R}_{giv} \cup L$.

(3) Suppose $\varphi_0 \longrightarrow \psi_0 \in \mathbb{R}_{giv} \cup \mathbb{R}_L$ is an instance of $R \longrightarrow S \in \mathbb{R}_{giv} \cup L$; $n \in \mathrm{Dom}\, \varphi_0$; $n \neq ()$. Then some $m \in R^{-1}(\mathbb{D} \cup W \cup X)$ has $m\ \underline{\mathrm{anc}}\ n$ and either $\varphi_0 m \in \mathbb{D}$ or

$$Rm \in W \ \&\ (\forall \psi \in V_{\#})[(\varphi_0/m)(n/m \longleftarrow \psi) \in D_{Rm}].$$

(4) $\mathfrak{C}_L$ is Church-Rosser.

Proof: For $U = W \cup X$, we have assigned a domain $D_u \subseteq V_{\#}$ to each $u \in U$ by (7.1). We defined L to be a set of schemata in (7.2). The set $\mathbb{R}_{giv}$ was defined in (7.3.1); since no parameters are used in $\mathbb{R}_{giv}$ it is trivially a set of rule-schemata. The set $\mathbb{R}_{\mathbb{R}_{giv} \cup L}$ of instances of members of $\mathbb{R}_{giv} \cup L$ has

$$\mathbb{R}_{\mathbb{R}_{giv} \cup L} = \mathbb{R}_{\mathbb{R}_{giv}} \cup \mathbb{R}_L = \mathbb{R}_{giv} \cup \mathbb{R}_L$$

because the only instance of a member of $\mathbb{R}_{giv}$ is itself. This proves (1).

Suppose $\varphi_0 \longrightarrow \psi_0 \in \mathbb{R}_{giv} \cup \mathbb{R}_L$. If $\varphi_0(\,) \in G$, then $\varphi_0 \longrightarrow \psi_0$ is an instance of itself in $\mathbb{R}_{giv}$ and of no schema in $\mathbb{R}_L$, since $\varphi_0(\,) \notin F$. If $\varphi_0(\,) \in F$, then $\varphi_0 \longrightarrow \psi_0$ is an instance of just one $R \longrightarrow S \in L$ (by (7.2.2) in the definition of RDs) and of no schema in $\mathbb{R}_{giv}$, since $\varphi_0(\,) \notin G$. This proves (2).

Suppose $\varphi_0 \longrightarrow \psi_0 \in \mathbb{R}_{giv} \cup \mathbb{R}_L$ is an instance of $R \longrightarrow S \in L$. Suppose $n \in \text{Dom}\,\varphi_0$ with $n \neq (\,)$, so that n begins with an integer n_0 such that $n_0 < \rho(\varphi_0(\,))$. Set $m = (n_0)$, so that $m \in \mathbb{N}^1$ and $m \; \underline{\text{anc}} \; n$. In the definition of RDs, (7.2.1) says that R has the form $f(a_0, \ldots, a_{k-1})$ with $f = \varphi_0(\,)$, $k = \rho(f)$, and $a_i \in \mathbb{D} \cup W \cup X$ for all $i < k$. Therefore, $Rm \in \mathbb{D} \cup W \cup X$. To complete the proof of (3) we consider two cases:

<u>Case 1</u> $(\varphi_0 m \in \mathbb{D})$ Nothing more needs to be shown.

<u>Case 2</u> $(\varphi_0 m \notin \mathbb{D})$ We must show that $Rm \in W$ and that any $\psi \in V_\#$ has $(\varphi_0/m)(n/m \longleftarrow \psi) \in D_{Rm}$. If $Rm \in \mathbb{D}$ then $\varphi_0/m = R/m$ because all the substitutions are made at nodes independent of m. If $Rm \in X$ then $\varphi_0/m \in \mathbb{D}_\#$ because $D_x = \mathbb{D}_\#$ for all $x \in X$. Both $Rm \in \mathbb{D}$ and $Rm \in X$ would contradict $\varphi_0 m \notin \mathbb{D}$, so we must have $Rm \in W$. But then $D_{Rm} = V_\#$ and $(\varphi_0/m)(n/m \longleftarrow \psi) \in V_\#$, so $(\varphi_0/m)(n/m \longleftarrow \psi) \in D_{Rm}$ and (3) has been proven.

We use the Main Theorem (5.6) to prove (4). As we noted in discussing Definition 7.2, (7.2.2) implies that no two rules in $\mathbb{R}_L$ have the same left half. Since the given functions are singlevalued, no two rules in $\mathbb{R}_{giv}$ have the same left half. Since $F \cap G = \emptyset$, these facts imply that $\mathfrak{C}_L$ is unequivocal. We must show that $\mathfrak{C}_L$ is closed. By (2) and the Rule-Schemata Theorem (6.5), it will suffice to prove (6.5.2): whenever $\varphi_0 \longrightarrow \psi_0$, $\varphi \longrightarrow \psi \in \mathbb{R}_L \cup \mathbb{R}_{giv}$, $\varphi_0 \longrightarrow \psi_0$ is an instance of

$R \longrightarrow S \in \mathbb{R}_{giv} \cup L$, $n \in \mathrm{Dom}\ \varphi_0$, $n \neq (\ )$, and $\varphi_0/n = \varphi$, then some $m \in R^{-1}(W \cup X)$ has m <u>anc</u> n and $(\varphi_0/m)(n/m \longleftarrow \psi) \in D_{Rm}$. Let m be the node from (3). If $\varphi_0 m \in \mathbb{D}$ then $m = n$ and $\varphi \longrightarrow \psi$ has the form $\bar{\xi} \longrightarrow \psi$ for $\xi = \varphi_0 m \in \mathbb{D}$. But there are no such rules in $\mathbb{R}_{giv} \cup \mathbb{R}_L$, so we have $\varphi_0 m \notin \mathbb{D}$. Therefore (3) implies that $R_m \in W$ and $(\varphi_0/m)(n/m \longleftarrow \psi) \in D_{Rm}$, as desired. ∎

The result $\mathrm{Eval}_L\ R$ of <u>evaluating</u> a tree $R \in V_{\#}$ under an RD L will be defined as ξ whenever $\xi \in \mathbb{D}$ and $R \overset{*}{\Longrightarrow} \bar{\xi}$ in $\mathfrak{C}_L$. It will be convenient to make evaluation a total function by letting a new object ∞ be $\mathrm{Eval}_L\ R$ whenever R has no normal form in $\mathbb{D}_{\#}$. Although evaluation is not a bottom-up algorithm, it will be shown that replacing a subtree R/n of a tree R by a tree S that evaluates like R/n does not change the evaluation of R.

(7.5) <u>Evaluation Theorem</u>. Let $\infty \notin \mathbb{D}$ and set $\bar{\mathbb{D}} = \mathbb{D} \cup \{\infty\}$. Let L be any RD. There is a unique surjection

(1) $\mathrm{Eval}_L : V_{\#} \longrightarrow \bar{\mathbb{D}}$ (total)

such that

(2) $(\forall R \in V_{\#})(\forall \xi \in \mathbb{D})(\mathrm{Eval}_L\ R = \xi \text{ iff } R \overset{*}{\Longrightarrow} \bar{\xi} \text{ in } \mathfrak{C}_L)$

(3) $(\forall R, S \in V_{\#})(R \overset{*}{\Longrightarrow} S \text{ in } \mathfrak{C}_L \text{ implies } \mathrm{Eval}_L\ R = \mathrm{Eval}_L\ S)$

(4) $(\forall R, S \in V_{\#})(\forall n \in \mathrm{Dom}\ R)$

$$(\mathrm{Eval}_L\ S = \mathrm{Eval}_L\ R/n \text{ implies } \mathrm{Eval}_L\ R(n \longleftarrow S) = \mathrm{Eval}_L\ R).$$

Proof: If $R \overset{*}{\Longrightarrow} \bar{\xi}$, then $\bar{\xi}$ is a normal form for R in $\mathfrak{C}_L$. By (7.4.4) and uniqueness of normal forms there is just one function Eval_L satisfying (1) and (2). Since $\text{Eval}_L\ \bar{\xi} = \xi$ for each $\xi \in \mathbb{D}$ and $\text{Eval}_L\ \bar{a} = \infty$ for each $a \in W \cup X$, this function is surjective. Now (3) follows from (2) and the Church-Rosser property. Finally, suppose $R, S \in V_\#$; $n \in \text{Dom } R$; $\text{Eval}_L\ S = \text{Eval}_L\ R/n$. We show that $\text{Eval}_L\ R(n \leftarrow S) = \text{Eval}_L\ R$.

Case 1 ($\text{Eval}_L\ S \neq \infty$.) Let $\text{Eval}_L\ S = \xi = \text{Eval}_L\ R/n$. Then

$$R \overset{*}{\Longrightarrow} R(n \longleftarrow \bar{\xi}) \text{ by } R/n \overset{*}{\Longrightarrow} \bar{\xi}$$

$$R(n \longleftarrow S) \overset{*}{\Longrightarrow} R(n \longleftarrow \bar{\xi}) \text{ by } S \overset{*}{\Longrightarrow} \bar{\xi}$$

Therefore $\text{Eval}_L\ R(n \longleftarrow S) = \text{Eval}_L\ R(n \longleftarrow \bar{\xi}) = \text{Eval}_L\ R$ by (3).

Case 2 ($\text{Eval}_L\ S = \infty$.) We may still have a finite value for $\text{Eval}_L\ R$.

Case 2.1 ($\text{Eval } R \neq \infty$.) Let $K \in \mathbb{N}$; $\xi \in \mathbb{D}$; $(R_0, \ldots, R_K) \in (V_\#)^{K+1}$ with

$$R = R_0 \Longrightarrow R_1 \ldots \Longrightarrow R_K = \bar{\xi}.$$

For each $k < K$, let a rule $\varphi_k \longrightarrow \psi_k$ be applied at a node m_k in R_k to form R_{k+1}. Let r_k be the residue map for $\varphi_k \longrightarrow \psi_k$ defined by Lemma 6.3. We will define a subset N_k of Dom R_k for each $k \leq K$. Intuitively, suppose that n has been painted red, that a red node stays red when a rule is applied there, and that residues of red nodes are painted red whenever a rule is applied. Then N_k is the set of all red nodes in R_k. Formally:

$$N_0 = \{n\}$$

$$N_{k+1} = \{p \in N_k \mid \text{NOT } m_k \ \underline{\text{anc}}\ p\} \cup (\{m_k\} \cap N_k) \cup \{m_k \cdot q \mid (\exists p \in N_k)[m_k \ \underline{\text{anc}}\ p \neq m_k \ \&\ q \in r_k(p/m_k)]\}.$$

For each $k \leq K$, N_k is an independent subset of Dom R_k such that

(5) $$(\forall p \in N_k)(\text{Eval}_L\ R_k/p = \infty)$$

by (3) and the fact that $R/n \overset{*}{\Longrightarrow} R_k/p$ whenever $p \in N_k$.

For each $k \leqslant K$ set

(6) $$T_k = R_k(N_k \longleftarrow S)\,.$$

For each $k < K$ we claim that

(7) $$T_k \overset{=}{\Longrightarrow} T_{k+1}\,.$$

If $m_k \in N_k \cdot \mathbb{N}^*$ then $T_k = T_{k+1}$, so we may assume $m_k \notin N_k \cdot \mathbb{N}^*$. Let N_k/m_k be the set $\{p/m_k \mid p \in N_k \;\&\; m_k \underline{\text{anc}}\ p\}$. For each $\ell \in N_k/m_k$, we have $\ell \in$ Dom φ_k with $\ell \neq (\,)$. We may apply the sublemma (7.4.3) to $\varphi_k \longrightarrow \psi_k$ to show that ℓ has an ancestor μ in left half Φ_k of $\varphi_k \longrightarrow \psi_k$'s schema such that either $\varphi_k\mu \in \mathbb{D}$ or

(8) $$\Phi_k\mu \in W \;\&\; (\forall \psi \in V_\#)\,[(\varphi_k/\mu)(\ell/\mu \longleftarrow \psi) \in D_{\Phi_k\mu}]\,.$$

But $\varphi_k\mu \in \mathbb{D}$ would imply $\mu = \ell$ and then $R/m_k \cdot \ell \in \mathbb{D}_\#$, which would contradict (5), so (8) must hold for all $\ell \in N_k/m_k$. We will apply Lemma 6.4 to show that

(9) $$\varphi_k(N_k/m_k \longleftarrow S) \longrightarrow \psi_k(r_k(N_k/m_k) \longleftarrow S) \in \mathbb{R}_{giv} \cup \mathbb{R}_L\,.$$

The left half Φ_k of $\varphi_k \longrightarrow \psi_k$'s schema plays the role of "R" in Lemma 6.4, and we have just shown that each $\ell \in N_k/m_k$ has an ancestor μ in $\Phi_k^{-1}(W \cup X)$, as required by (6.4.1). By (8) and $D_{\Phi_k\mu} = V_\#$ we have

$$(\varphi_k/\mu)(N_0^\mu/\mu \leftarrow S)\ldots(N_{-1}^\mu/\mu \leftarrow S) \in V_\#$$

for each such μ, where $(N_0^\mu, \ldots, N_{-1}^\mu)$ is a listing of the members of

N_k/m_k descended from μ. This is just what (6.4.2) requires, so (9) does follow from Lemma 6.4.

Direct calculations using the elementary properties of trees from §4 show that this rule is applicable to T_k at m_k and that T_{k+1} is the result. First we note that

$$\begin{aligned} T_k/m_k &= R_k(N_k \leftarrow S)/m_k = (R_k/m_k)(N_k/m_k \leftarrow S) \\ &= \varphi_k(N_k/m_k \leftarrow S), \end{aligned}$$

so that T_k/m_k matches the left half of the rule in (9). By $m_k \notin N_k \cdot \mathbb{N}^*$, the definition of N_{k+1} can be rewritten as

$$N_{k+1} = \{p \in N_k \mid m_k \perp p\} \cup [m_k \cdot r_k(N_k/m_k)].$$

Therefore

$$\begin{aligned} T_{k+1} &= R_{k+1}(N_{k+1} \leftarrow S) \\ &= R_k(m_k \leftarrow \psi_k)(\{p \in N_k \mid m_k \perp p\} \leftarrow S) \\ &\qquad (m_k \cdot r_k(N_k/m_k) \leftarrow S) \\ &= R_k(N_k \leftarrow S)(m_k \leftarrow \psi_i)(m_k \cdot r_k(N_k/m_k) \leftarrow S) \\ &= T_k(m_k \leftarrow \psi_k(r_k(N_k/m_k) \leftarrow S)). \end{aligned}$$

This completes the proof of (7).

By (6) and (7) we have

$$R(n \longleftarrow S) = T_0 \overset{*}{\Longrightarrow} T_K = \xi(N_K \longleftarrow S)$$

while (5) implies that $N_K = \emptyset$. Therefore $\text{Eval}_L\ R(n \longleftarrow S) = \xi = \text{Eval}_L\ R$.

<u>Case 2.2</u> ($\text{Eval}_L\ R = \infty$.) Suppose $\text{Eval}_L\ R(n \longleftarrow S) \neq \infty$. Then $\text{Eval}_L\ R(n \longleftarrow S)(n \longleftarrow R/n) \neq \infty$ by the reasoning of Case 2.1. But

$R(n \longleftarrow S)(n \longleftarrow R/n) = R$, so this is absurd. Therefore $\mathrm{Eval}_L\ R(n \longleftarrow S) = \infty = \mathrm{Eval}_L\ R$. ∎

If L is an RD and $f \in F$ then we could define a partial function $\Phi : \mathbb{D}^{\rho(f)} \longrightarrow \mathbb{D}$ by the evaluation process:

$$\Phi(\xi_0, \ldots, \xi_{-1}) = \mathrm{Eval}_L\ \bar{f}(\bar{\xi}_0, \ldots, \bar{\xi}_{-1}).$$

The total function $Lf : \bar{\mathbb{D}}^{\rho(f)} \longrightarrow \bar{\mathbb{D}}$ defined below includes this function and is more convenient. Our ultimate interest is usually in $\Phi = Lf \cap (\mathbb{D}^{\rho(f)} \times \mathbb{D})$ for some one $f \in F$, but the mathematics of $\{Lh \mid h \in V\}$ tells much more about Φ than the mathematics of $\{Lh \cap (\mathbb{D}^{\rho(h)} \times \mathbb{D}) \mid h \in F\}$ alone.

(7.6) Definition. Let L be an RD, T_∞ be any member of $V_\#$ such that $\mathrm{Eval}_L\ T_\infty = \infty$. For each $\theta \in \bar{\mathbb{D}}$ set

(1) $$\tilde{\theta} = \text{if } \theta \in \mathbb{D} \text{ then } \bar{\theta} \text{ else } T_\infty .$$

For each $f \in V$ and all $\theta \in \bar{\mathbb{D}}^{\rho(f)}$ set

(2) $$Lf(\theta_0, \ldots, \theta_{-1}) = \mathrm{Eval}_L\ \bar{f}(\tilde{\theta}_0, \ldots, \tilde{\theta}_{-1}).$$

When $f \in \mathbb{D}$ we have $\mathrm{Dom}\ Lf = \{()\}$ and the only value assumed is $Lf() = f$. When $f \in W \cup X$ we have $Lf() = \infty$. When $f \in G$, Lf is the natural extension of f from a function on $\mathbb{D}^{\rho(f)}$ to a function on $\bar{\mathbb{D}}^{\rho(f)}$: $Lf(\theta_0, \ldots, \theta_{-1})$ is $f(\theta_0, \ldots, \theta_{-1})$ if each $\theta_i \in \mathbb{D}$ and is ∞ otherwise. When $f \in F$ the values of Lf depend on the choice of L and may be finite despite infinite arguments. For example, if L includes the schemata for conditionals, then $LC(\text{yes}, 17, \infty) = 17$.

What have we really gained by adding ∞ to the data space and expressing the theory in terms of total functions? One advantage is that the theory is somewhat richer. If an RD L and function letters $f, f' \in F_k$ are such that

(a) $$Lf \cap (\mathbb{D}^k \times \mathbb{D}) = Lf' \cap (\mathbb{D}^k \times \mathbb{D}),$$

then the theory restricted to partial functions would be unable to distinguish between Lf and Lf', although $Lf \neq Lf'$ is quite consistent with (a). By introducing ∞ we have made it possible to assert more than we could before; in the next section we will find that some of these new assertions are true. Of course, we may still use a definition such as

(b) $$[L]f = Lf \cap (\mathbb{D}^{\rho(f)} \times \mathbb{D})$$

to restrict parts of the discussion to partial functions whenever we wish to do so.

Another advantage is simplicity. Suppose that $\Phi: \mathbb{D}^2 \longrightarrow \mathbb{D}$ and $\Psi: \mathbb{D} \longrightarrow \mathbb{D}$, with both functions partial. An expression such as $\Phi(\xi, \Psi(\eta))$ is quite awkward to work with if we have no assurance that $\eta \in \text{Dom } \Psi$ and that $(\xi, \Psi(\eta)) \in \text{Dom } \Phi$. What does

$$\Phi(\xi, \Psi(\eta)) = \zeta$$

mean? Suppose $\eta \notin \text{Dom } \Psi$ but $\Phi(37, -)$ is a constant function on $\mathbb{D}$. Is it correct to write

$$\Phi(37, \Psi(\eta)) = \Phi(37, 94) ?$$

Kleene [26, §63] introduces special notions of equality and of composition of functions in order to handle such problems consistently. We only need the usual intuitive notions, yet we can make finer distinctions. If $\tilde{\Phi} : \bar{\mathbb{D}}^2 \longrightarrow \bar{\mathbb{D}}$ is a total function and $\tilde{\Phi}(37, -)$ is constant on $\mathbb{D}$, then

$$\widetilde{\Phi}(37, \infty) = \widetilde{\Phi}(37, 94)$$

will be true if $\widetilde{\Phi}(37, -)$ is actually constant on all of $\bar{\mathbb{D}}$ but will be false otherwise.

The introduction of ∞ here is very much like the addition of ∞ to the number system in the study of infinite series: the mathematics becomes somewhat richer and simpler than before, but none of the real difficulties are magically removed. Clever notations can only dissolve the crust of merely notational difficulty that often obscures a real problem. In particular, the set $\mathscr{C}$ of partial functions $\Phi : \mathbb{D} \longrightarrow \mathbb{D}$ computable by finite RDs is the natural generalization of partial recursive functions. We would define this set as

$$\mathscr{C} = \{\Phi \mid (\exists \text{ L, a finite RD})(\exists f \in F_1)[\Phi = Lf \cap (\mathbb{D} \times \mathbb{D})]\}.$$

Someone who prefers not to consider ∞ would write

$$\mathscr{C} = \{\Phi \mid (\exists \text{ L, a finite RD})(\exists f \in F_1) \\ (\Phi = \{\langle \xi, \eta \rangle \in \mathbb{D} \times \mathbb{D} \mid \bar{f}(\bar{\xi}) \overset{*}{\Longrightarrow} \bar{\eta} \text{ in } \mathfrak{C}_L \})\}$$

instead but would specify exactly the same class of partial functions. Unsolvable problems are still unsolvable problems in either case.

We have defined RDs and have explained them algorithmically: for any RD L, the SRS $\mathfrak{C}_L$ provides a nondeterministic algorithm for evaluating expressions, and any function letter f defines a function $Lf : \bar{\mathbb{D}}^{\rho(f)} \longrightarrow \bar{\mathbb{D}}$ that is singlevalued because normal forms are unique in the Church-Rosser system $\mathfrak{C}_L$. The fact that $Lf \cap (\mathbb{D}^{\rho(f)} \times \mathbb{D})$ is singlevalued whenever $f \in F$ and L is an RD without call-by-name has been proven by E.K. Blum [3, §4, Thm. 1], whose "proper programs" are equivalent to this kind of RD. (As we remarked earlier, the additional assumptions on the data space and given functions in [3] are irrelevant to the functionality problem.) Blum's proof depends on the restriction to call-by-value and does not cover our result.

8. A Semantic Explanation

Any recursive definition L defines a set of rules $\mathbb{R}_L$ that may also be viewed as a set of equations. The rule $\varphi \longrightarrow \psi$ corresponds to the statement $\varphi = \psi$. If we interpret the function letters as names of specific functions on $\overline{\mathbb{D}}$, then the assignment of functions to function letters may or may not "solve" the equations by assigning the same value of $\overline{\mathbb{D}}$ to the left half of every rule as to the right half.

For an example we return to the RD for factorials:

$$\overline{f}(\overline{x}) \longrightarrow \overline{C}(\overline{EQ}(\overline{x}, \overline{0}), \overline{1}, \overline{X}(\overline{x}, \overline{f}(\overline{\dot{-}}(\overline{x}, \overline{1}))))$$

$$\overline{C}(\overline{yes}, \overline{u}, \overline{v}) \longrightarrow \overline{u}$$

$$\overline{C}(\overline{no}, \overline{u}, \overline{v}) \longrightarrow \overline{v}\ .$$

Here we have named the given function

$$\{\langle(\xi, \eta), \zeta\rangle \mid \xi, \eta, \zeta \in \mathbb{D}\ \&\ [(\xi = \eta\ \&\ \zeta = \text{yes})\ \text{or}\ (\xi \neq \eta\ \&\ \zeta = \text{no}]\}$$

"EQ" rather than "=" in order to prevent confusion between it and the relation of equality on $\overline{\mathbb{D}}$.

The rule for $\overline{f}(\overline{6})$ may be thought of as an equation

$$(1) \qquad \overline{f}(\overline{6}) = \overline{C}(\overline{EQ}(\overline{6}, \overline{0}), \overline{1}, \overline{X}(\overline{6}, \overline{f}(\overline{\dot{-}}(\overline{6}, \overline{1}))))$$

that may be true or false, depending on which operations are considered to be the meanings of the operators. We are concerned with whether $\overline{f}(\overline{6})$ and $\overline{C}(\ldots)$ have the same <u>value</u> in $\overline{\mathbb{D}}$; we already know they are not the same tree! We therefore write (1) more explicitly as

(2) $\quad \mathrm{Val}_I\, \bar{f}(\bar{6}) = \mathrm{Val}_I\, \bar{C}(\overline{EQ}(\bar{6}, \bar{0}), \bar{1}, \bar{\times}(\bar{6}, \bar{f}(\bar{\dot{-}}(\bar{6}, \bar{1})))),$

where I is an "interpretation" of the function letters that assigns an actual function

$$I_f : \bar{\mathbb{D}}^{\rho(f)} \longrightarrow \bar{\mathbb{D}} \text{ (total)}$$

to each $f \in F$. Whether (2) is true or false will depend on the choice of I and on the precise definition of "values" by a map

$$\mathrm{Val}_I : V_{\#} \longrightarrow \bar{\mathbb{D}} \text{ (total)}$$

for each possible choice of I.

A <u>semantic</u> <u>explanation</u> for RDs is any scheme for assigning value maps Val_I to interpretations I, so that an interpretation I can be said to <u>solve</u> an RD L if

$$\mathrm{Val}_I\, \varphi = \mathrm{Val}_I\, \psi$$

for each rule $\varphi \longrightarrow \psi \in \mathbb{R}_L$. An RD L has a set of <u>solutions</u> I, just as the numerical equation

$$Y^2 + Y - 6 = 0$$

has a set of solutions $\{-3, 2\}$. (Because RDs may have many function letters, we must consider assignments of functions to function letters rather than just single functions as solutions.)

For numerical equations there is little else to say: some equations have no solutions and others have many. The answer to a numerical question is often one of an equation's solutions, but <u>which</u> one is determined by parts of the question not modeled by the equation. Different questions may determine the same equation but require different choices from among its solutions.

For recursive definitions there is much more to say about their solutions. Under the natural semantic explanation we will use, every RD L has solutions. Indeed, the assignment $I_f = Lf$ of functions to function letters defined by (7.6.2) is a solution. This particular solution can also be characterized in a purely semantic way — it is the unique solution that is "extended" by every other solution. (The notion of "extension" to be used here is similar to the familiar relation among partial functions: f extends g if the set f of ordered pairs is a superset of the set g of ordered pairs.) We call this the "canonical" solution.

By showing that $I_f = Lf$ for all $f \in F$ will solve any RD, we will show that the algorithmic explanation from §7 is "valid" relative to our semantic explanation. Each RD L has a canonical solution I, and the algorithmic explanation defines nondeterministic algorithms for computing the functions I_f. The function Lf computed by the algorithm for f is indeed I_f: the program is correct. A rule of inference is said to be valid if all its instances are correct inferences; similarly, we say that the algorithmic explanation in §7 is valid because the algorithms it defines for each L are correct when considered as attempts to specify the functions in the canonical solution for L. The main result of this section will be called the "Validity Theorem."

If we call members of $F \cup G$ "operators" and members of $\mathbb{D} \cup W \cup X$ "individual symbols" then $V_\#$ corresponds to a set of "terms" in logic. A logician would "interpret" operators f of rank k as operations $I_f\colon \bar{\mathbb{D}}^k \longrightarrow \bar{\mathbb{D}}$ and individual symbols a as members $J_a \in \bar{\mathbb{D}}$, in order to interpret terms as denoting members of $\mathbb{D}$.

It will be more convenient to treat "symbols" of rank 0 consistently. We will say that each individual symbol is interpreted as an operation $I_a : \bar{\mathbb{D}}^0 \longrightarrow \bar{\mathbb{D}}$, so that $I_a = \{\langle(\,),\theta\rangle\}$ for some $\theta \in \mathbb{D}$. This will save some unnecessary case analyses without changing the familiar intuitive notion of "interpreting" symbols as members of $\bar{\mathbb{D}}$ or operations on them.

(8.1) <u>Definition</u>. An <u>interpretation</u> of V on $\bar{\mathbb{D}}$ is any function I assigning to each $\alpha \in V$ a total function $I_\alpha : \bar{\mathbb{D}}^{\rho(\alpha)} \longrightarrow \bar{\mathbb{D}}$.

Interpreting members of V-F as well as members of F is a technical convenience. Once an operation I_α has been assigned to each $\alpha \in V$ by an interpretation I, values in $\bar{\mathbb{D}}$ can indeed be assigned to all the "terms" in $V_\#$ by straightforward bottom-up applications of operations, just as in our original example of evaluating arithmetic expressions. More precisely, the following lemma is proven by induction on the sizes of trees.

(8.2) <u>Lemma</u>. Let I be an interpretation of V on $\bar{\mathbb{D}}$. There is a unique function

(1) $\quad \mathrm{Val}_I : V_\# \longrightarrow \bar{\mathbb{D}} \quad$ (total)

such that

(2) $\quad (\forall \alpha \in V)(\forall T \in (V_\#)^{\rho(\alpha)})$

$$[\mathrm{Val}_I\, \bar{\alpha}(T_0, \ldots, T_{-1}) = I_\alpha(\mathrm{Val}_I\, T_0, \ldots, \mathrm{Val}_I\, T_{-1})]. \blacksquare$$

(The members of V of rank 0 are treated consistently and need no different kind of interpretation.)

An interpretation I of V on $\bar{\mathbb{D}}$ <u>solves</u> a recursive definition L if it agrees with the meanings of the constants, parameters, and given functions and turns every instance of L into a true equation. A solution for L that agrees with all other solutions wherever it is finite is a <u>canonical</u> solution.

(8.3) <u>Definition.</u> Let L be an RD. An interpretation I of V on $\bar{\mathbb{D}}$ is a <u>solution</u> for L iff

(1) $$(\forall \xi \in \mathbb{D})(I_\xi = \{\langle (), \xi \rangle\})$$

(2) $$(\forall a \in W \cup X)(I_a = \{\langle (), \infty \rangle\})$$

(3) $$(\forall g \in G)(\forall \theta \in \bar{\mathbb{D}}^{\rho(g)})(I_g(\theta) = \underset{\sim}{\text{if}}\ \ \theta \in \mathbb{D}^{\rho(g)}\ \ \underset{\sim}{\text{then}}\ \ g(\theta)\ \ \underset{\sim}{\text{else}}\ \infty)$$

(4) $$(\forall \varphi \longrightarrow \psi \in \mathbb{R}_L)(\mathrm{Val}_I\, \varphi = \mathrm{Val}_I\, \psi).$$

A solution I for L is <u>canonical</u> iff every solution J for L satisfies

(5) $$(\forall\, \alpha \in V)(\forall\, \theta \in \bar{\mathbb{D}}^{\rho(\alpha)})$$
$$[I_\alpha(\theta) \neq \infty \ \text{ implies }\ J_\alpha(\theta) = I_\alpha(\theta)\,].$$

Thus a canonical solution agrees with every other solution whereever it is finite. An alternative way to characterize canonical solutions is to define a partial ordering $\leqslant$ on the total functions $\Phi : \bar{\mathbb{D}}^k \longrightarrow \bar{\mathbb{D}}$ by

$$\Phi \leqslant \Psi \ \text{ iff }\ (\forall \theta \in \bar{\mathbb{D}}^k)(\Phi(\theta) \neq \infty \text{ implies } \Phi(\theta) = \Psi(\theta))$$
$$\text{iff }\ \Phi \cap (\bar{\mathbb{D}}^k \times \mathbb{D}) \subseteq \Psi \cap (\bar{\mathbb{D}}^k \times \mathbb{D}),$$

so that $\leqslant$ is very much like the relation "is extended by" among partial functions. Then $\leqslant$ can be generalized to interpretations by setting

$$I \leqslant J \ \text{ iff }\ (\forall \alpha \in V)(I_\alpha \leqslant J_\alpha).$$

A solution I for L is canonical iff it is a minimal solution in this partial ordering: $I \leqslant J$ for every solution J.

Now we are ready to show that any RD does have a unique canonical solution and that this solution is the one specified by our algorithmic explanation.

(8.4) <u>Validity Theorem</u>. Let L be an RD and define an interpretation I of V on $\bar{\mathbb{D}}$ by setting $I_\alpha = L\alpha$ for each $\alpha \in V$. Then

(1) $(\forall R \in V_\#)(\mathrm{Val}_I\, R = \mathrm{Eval}_L\, R)$

(2) I is the <u>unique</u> canonical solution for L.

<u>Proof</u>: We prove (1) by induction on the size $|R|$ of R. (This is the cardinality of R as a <u>set</u> of ordered pairs in (4.3).) Suppose R is a tree such that $\mathrm{Val}_I\, T = \mathrm{Eval}_L\, T$ whenever $|T| < |R|$. We show that $\mathrm{Val}_I\, R = \mathrm{Eval}_L\, R$. Let $\alpha = R(\,)$ and $k = \rho(\alpha)$, so that

$$R = \bar{\alpha}(T_0, \dots, T_{-1}) \quad \text{with} \quad (T_0, \dots, T_{-1}) \in (V_\#)^k .$$

Let $\theta_i = \mathrm{Val}_I\, T_i$ for each $i < k$. By Lemma 8.2 and Definition 7.6,

(3) $\mathrm{Val}_I\, R = I_\alpha(\theta_0, \dots, \theta_{-1}) = L_\alpha(\theta_0, \dots, \theta_{-1}) = \mathrm{Eval}_L\, \bar{\alpha}(\tilde{\theta}_0, \dots, \tilde{\theta}_{-1})$.

By the induction hypothesis, $\theta_i = \mathrm{Eval}_L\, T_i$ for each $i < k$. Therefore, by $\mathrm{Eval}_L\, \bar{\xi} = \xi$ for all $\xi \in \mathbb{D}$ in (7.5.2) and by the definition (7.6.1) of $\tilde{\theta}_i$,

$$\mathrm{Eval}_L\, \tilde{\theta}_i = \underset{\sim}{\text{if}}\ \theta_i \in \mathbb{D}\ \underset{\sim}{\text{then}}\ \theta_i\ \underset{\sim}{\text{else}}\ \infty = \theta_i = \mathrm{Eval}_L\, T_i .$$

For each $i < k$ we therefore have

$$\mathrm{Eval}_L\, T_i = \mathrm{Eval}_L[\bar{\alpha}(T_0, \dots, T_{i-1}, \tilde{\theta}_i, \tilde{\theta}_{i+1}, \dots, \tilde{\theta}_{-1})/(i)]$$

and so

$$\mathrm{Eval}_L\, \bar{\alpha}(T_0, \dots, T_{i-1}, T_i, \tilde{\theta}_{i+1}, \dots, \tilde{\theta}_{-1}) =$$

$$\mathrm{Eval}_L\, \bar{\alpha}(T_0, \dots, T_{i-1}, \tilde{\theta}_i, \tilde{\theta}_{i+1}, \dots, \tilde{\theta}_{-1})$$

by part (4) of the Evaluation Theorem (7.5). In k steps we may replace $\tilde{\theta}_0, \dots, \tilde{\theta}_{-1}$ by $T_0, \dots, T_{-1}$ in $\bar{\alpha}(\tilde{\theta}_0, \dots, \tilde{\theta}_{-1})$ without changing the value of Eval_L. Therefore (3) implies that

$$\mathrm{Val}_I\ R = \mathrm{Eval}_L\ \bar{\alpha}(T_0, \dots, T_{-1}) = \mathrm{Eval}_L\ R\,.$$

Now we must prove (2). By the definition of canonical solutions (8.3.5), any pair of canonical solutions H, J must have

$$(\forall \alpha \in V)(\forall \theta \in \bar{\mathbb{D}}^{\rho(\alpha)})$$

$$[(H_\alpha(\theta) \neq \infty \text{ or } J_\alpha(\theta) \neq \infty) \text{ implies } H_\alpha(\theta) = J_\alpha(\theta)].$$

This implies H = J, so there can be _at most one_ canonical solution. Now we show that I is _a_ canonical solution.

To show that I is a solution we must verify conditions (8.3.1) - (8.3.4). For any $\xi \in \mathbb{D}$ we have $\mathrm{Eval}_L\ \bar{\xi} = \xi$ and so $I_\xi(\ ) = L\xi(\ ) = \xi$, as required by (8.3.1). For any $a \in W \cup X$, $\bar{a}$ is irreplaceable in $\mathfrak{C}_L$ and $\bar{a} \notin \mathbb{D}_\#$, so $\mathrm{Eval}_L\ \bar{a} = \infty$. Therefore $I_a(\ ) = La(\ ) = \infty$, as required by (8.3.2). For any $g \in G$ and $\theta \in \bar{\mathbb{D}}^{\rho(g)}$, $I_g(\theta) = Lg(\theta) = \mathrm{Eval}_L\ \bar{g}(\tilde{\theta}_0, \dots, \tilde{\theta}_{-1})$. If $\theta_i \in \mathbb{D}$ for all i then $\bar{g}(\tilde{\theta}_0, \dots, \tilde{\theta}_{-1}) \longrightarrow \bar{\eta}$ is a rule of $\mathfrak{C}_L$ for $\eta = g(\theta)$. If $\theta_i = \infty$ for some i then any $S \in V_\#$ with $\bar{g}(\tilde{\theta}_0, \dots, \tilde{\theta}_{-1}) \overset{*}{\Longrightarrow} S$ has $S(\ ) = g$ and $S/(i) \notin \mathbb{D}_\#$. Therefore $\mathrm{Eval}_L\ \bar{g}(\tilde{\theta}_0, \dots, \tilde{\theta}_{-1})$ is ∞ in this case. In both cases we have

$$I_g(\theta) = \underset{\sim}{\text{if}}\ \theta \in \mathbb{D}^{\rho(g)}\ \underset{\sim}{\text{then}}\ g(\theta)\ \underset{\sim}{\text{else}}\ \infty,$$

as required by (8.3.3). Finally, we must show that $\mathrm{Val}_I\ \varphi = \mathrm{Val}_I\ \psi$ for each $\varphi \longrightarrow \psi \in \mathbb{R}_L$ to verify (8.3.4). To do this we apply (1) to the special case

$$(\forall \varphi \longrightarrow \psi \in \mathbb{R}_L)(\mathrm{Eval}_L\ \varphi = \mathrm{Eval}_L\ \psi)$$

of part (3) in the Evaluation Theorem (7.5). Therefore I is a solution for L. We must show that I is canonical.

Suppose J is also a solution. We verify (8.3.5) by showing that each $K \in \mathbb{N}$ has the property

(4) $$(\forall R \in V_{\#})(\forall \xi \in \mathbb{D})(R \overset{K}{\Longrightarrow} \bar{\xi} \text{ implies } \mathrm{Val}_J\, R = \xi).$$

For $K = 0$ we have $R = \bar{\xi}$ and so

$$\mathrm{Val}_J\, R = J_{\xi}(\,) = \xi$$

by (1) in the definition of solutions (8.3). To pass from K to K+1, suppose (4) holds for K and R, S, ξ are such that

$$R \Longrightarrow S \overset{K}{\Longrightarrow} \bar{\xi}.$$

We show $\mathrm{Val}_J\, R = \xi$. Let $n \in \mathrm{Dom}\, R$ and $\varphi \longrightarrow \psi \in \mathbb{R}_{giv} \cup \mathbb{R}_L$ with

(5) $$R/n = \varphi \;\;\&\;\; S = R(n \longleftarrow \psi).$$

We claim that

(6) $$\mathrm{Val}_J\, \varphi = \mathrm{Val}_J\, \psi.$$

If $\varphi \longrightarrow \psi \in \mathbb{R}_{giv}$ then (6) holds because J, as a solution for L, satisfies (8.3.1) and (8.3.3). If $\varphi \longrightarrow \psi \in \mathbb{R}_L$ then (6) holds because J, as a solution for L, satisfies (8.3.4). By (5) and (6),

$$S = R(n \longleftarrow \psi) \;\&\; \mathrm{Val}_J \psi = \mathrm{Val}_J (R/n).$$

By induction on $|n|$ and the definition of values under interpretations from Lemma 8.2, this implies that $\mathrm{Val}_J\, R = \mathrm{Val}_J\, S$. But $\mathrm{Val}_J\, S = \xi$ by the induction hypothesis, so $\mathrm{Val}_J\, R = \xi$. ∎

Morris [37, Chap. 3, Thm. 2] stated a conjecture that amounts to this theorem when RDs are defined formally in the manner of §7. He suggested that the theorem could be derived from a small extension of a theorem about the lambda calculus [37, Chap. 3, Thm. 7]. Rather than formally relate the McCarthy calculus to the less intuitive lambda calculus, we proved the Validity Theorem directly.

Suppose that H, J are solutions for an RD L with canonical solution I. Let $f \in F$ and let $\Delta_f = \{\theta \in \bar{\mathbb{D}}^{\rho(f)} \mid Lf(\theta) \neq \infty\}$. Then each $\theta \in \Delta_f$ has $I_f(\theta) \neq \infty$ by the Validity Theorem and so $I_f(\theta) = H_f(\theta)$ and $I_f(\theta) = J_f(\theta)$ because I is canonical. Therefore

$$(\forall \theta \in \Delta_f)[H_f(\theta) = J_f(\theta)].$$

This is an extension of McCarthy's principle of "recursion induction" [34, §8], which asserts only that

$$(\forall \theta \in \Delta_f \cap \mathbb{D}^{\rho(f)}).[J_f(\theta) = J_f(\theta)].$$

The Validity Theorem is also related to Kleene's "first recursion theorem" [26, §66, Thm. 26]: any partial recursive functional $\mathcal{F}$ has a unique minimal fixed point defined by formal calculations using the set of equations that specifies $\mathcal{F}$. A functional is a mapping from functions to functions, or from interpretations of V on $\bar{\mathbb{D}}$ to interpretations of V on $\bar{\mathbb{D}}$, when we wish to think in terms of replacing one system of several functions by another. A partial recursive functional is one definable by formal calculations from sets of equations [26, §63]. An interpretation I is a fixed point of a functional $\mathcal{F}$ if $I = \mathcal{F}(I)$. Minimality is defined by the partial ordering $\leqslant$ that we considered earlier:

$$I \leqslant J \text{ iff } (\forall \alpha \in V)\,[I_\alpha \cap (\bar{\mathbb{D}}^{\rho(\alpha)} \times \mathbb{D}) \subseteq J_\alpha \cap (\bar{\mathbb{D}}^{\rho(\alpha)} \times \mathbb{D})].$$

To state the Validity Theorem in terms of functionals, we assign a functional $\mathcal{F}_L$ to each RD L. Suppose that I is an interpretation of V on $\bar{\mathbb{D}}$ which agrees with the intended meanings of constants, parameters, and given functions, but which may not make each rule in $\mathbb{R}_L$ a true equation. Thus we are considering interpretations that satisfy (1) - (3) but not necessarily (4) in Definition 8.3. We define a new interpretation $J = \mathcal{F}_L(I)$. For $\alpha \in V-F$ let $J_\alpha = I_\alpha$. For $f \in F$ consider any $\theta \in \bar{\mathbb{D}}^{\rho(f)}$. There is at most one rule $\varphi \longrightarrow \psi \in \mathbb{R}_L$ of the form

$$\bar{f}(\tilde{\theta}_0, \dots, \tilde{\theta}_{-1}) \longrightarrow \psi .$$

If there is such a rule we set

(1) $$J_f(\theta) = \mathrm{Val}_I\, \psi ;$$

otherwise we set $J_f(\theta) = I_f(\theta)$.

We claim that I is a fixed point of $\mathcal{F}_L$ iff I is a solution for L, so that I is a minimal fixed point of $\mathcal{F}_L$ iff I is a canonical solution for L.

Suppose that I is a fixed point of $\mathcal{F}_L$. If $\varphi \longrightarrow \psi \in \mathbb{R}_L$, we must show that $\mathrm{Val}_I\, \varphi = \mathrm{Val}_I\, \psi$. Let $\varphi = \bar{f}(R_0, \dots, R_{-1})$ with $f \in F$. Set $\theta_i = \mathrm{Val}_I\, R_i$ for each $i < \rho(f)$. Replacing R_i by $\tilde{\theta}_i$ at (i) in Dom φ and at each residue of (i) in Dom ψ, we construct a rule $\tilde{\varphi} \longrightarrow \tilde{\psi}$, where

$$\tilde{\varphi} = \bar{f}(\tilde{\theta}_0, \dots, \tilde{\theta}_{-1})$$

and $\mathrm{Val}_I\, \tilde{\varphi} = \mathrm{Val}_I\, \varphi$ and $\mathrm{Val}_I\, \tilde{\psi} = \mathrm{Val}_I\, \psi$. By (1) and the fixed point property,

$$I_f(\theta_0, \dots, \theta_{-1}) = \mathrm{Val}_I\, \tilde{\psi} .$$

Therefore

$$\mathrm{Val}_I\ \varphi = \mathrm{Val}_I\ \bar{f}(R_0, \ldots, R_{-1}) = I_f(\mathrm{Val}_I\ R_0, \ldots, \mathrm{Val}_I\ R_{-1})$$
$$= I_f(\theta_0, \ldots, \theta_{-1}) = \mathrm{Val}_I\ \tilde{\psi} = \mathrm{Val}_I\ \psi .$$

Now suppose that I is a solution for L. For $f \in F$ and $\theta \in \bar{\mathbb{D}}^{\rho(f)}$ we must show that $J_f(\theta) = I_f(\theta)$ where $J = \mathcal{F}_L(I)$. If there is no rule in $\mathbb{R}_L$ of the form

$$\bar{f}(\tilde{\theta}_0, \ldots, \tilde{\theta}_{-1}) \longrightarrow \psi$$

then $J_f(\theta) = I_f(\theta)$ already, so we may assume there is such a rule. If $\theta_i = \infty$ we may assume that the tree $\tilde{\theta}_i$ has been chosen in (7.6.1) so as to make $\mathrm{Val}_I\ \tilde{\theta}_i = \infty$; we could, for example, take $\tilde{\theta}_i$ to be $\bar{x}$ for any $x \in X$. Therefore we may assume $\mathrm{Val}_I\ \tilde{\theta}_i = \theta_i$ for all $i < \rho(f)$, since $\mathrm{Val}_L\ \tilde{\theta}_i = \theta_i$ for $\theta_i \neq \infty$ also. Therefore, by Lemma 8.2,

$$I_f(\theta) = \mathrm{Val}_I\ \bar{f}(\tilde{\theta}_0, \ldots, \tilde{\theta}_{-1}) .$$

But $\mathrm{Val}_I\ \bar{f}(\tilde{\theta}_0, \ldots, \tilde{\theta}_{-1}) = \mathrm{Val}_I\ \psi$ because I is a solution for L, so $I_f(\theta) = \mathrm{Val}_I\ \psi$. Comparison with (1) yields $J_f(\theta) = I_f(\theta)$.

In light of the equivalence just demonstrated, the Validity Theorem can be restated as follows:

(*) Any functional $\mathcal{F}$ such that $\mathcal{F} = \mathcal{F}_L$ for some RD L has a unique minimal fixed point I, and $I_f = Lf$ for each $f \in F$.

Restating the "first recursion theorem" [26, §66, Thm. 26] for comparison, we have the following statement about functionals that map functions on $\bar{\mathbb{D}}^k$ to functions on $\bar{\mathbb{D}}^k$:

(**) Any functional $\mathcal{F}$ definable by formal calculations from a system of equations E has a unique minimal fixed point Φ, and Φ is defined by formal calculations based on E.

Aside from the minor difference in format between calculations with equations and applications of rules, there are two differences between (*) and (**). The first difference is that only numerals can be substituted for variables in (**) [26, §54], so that only call-by-value is allowed. There are no restrictions on call-by-name in (*). The second difference is that (*) does place syntactic restrictions on the equations that are not required by (**).

An RD corresponds to a set of equations of the form

$$f(a_0, \ldots, a_{-1}) = e$$

where each a_i has rank zero. Further syntactic restrictions in Definition 7.2 guarantee that the SRS will be unequivocal and that single-valued functions will be defined. In [26] the system E may be any system such that, for each choice of f and $\xi_0, \ldots, \xi_{-1}$, at most one equation $f(\xi_0, \ldots, \xi_{-1}) = \eta$ can be derived. Furthermore, this single-valuedness must not depend on the choice of the given functions [26, §63]. For example,

$$f(x, g(x. x)) = 3 \quad \text{where} \quad g \in G; \; f \in F$$

is an acceptable set of equations. For any $\Psi: \bar{\mathbb{D}}^2 \longrightarrow \bar{\mathbb{D}}$, set

$$\Phi(\xi, \eta) = 3$$

whenever $\xi, \eta \neq \infty$ and $\eta = \Psi(\xi, \xi)$, with $\Phi(\xi, \eta) = \infty$ otherwise. Then the functional $\Phi = \mathcal{F}(\Psi)$ is definable by this set of equations: Φ is the set of all $\langle (\xi, \eta), \zeta \rangle$ such that

$$f(\xi, \eta) = \zeta$$

is deducible when g is considered to be Ψ. To define this functional by an RD we could use

$$\bar{f}(\bar{x}, \bar{y}) \longrightarrow \bar{C}(\overline{EQ}(\bar{y}, \bar{f}(\bar{x}, \bar{x})), \bar{3}, \bar{h}(\bar{0}))$$

$$\bar{C}(\overline{yes}, \bar{u}, \bar{v}) \longrightarrow \bar{u}$$

$$\bar{C}(\overline{no}, \bar{n}, \bar{v}) \longrightarrow \bar{v}$$

where $h \in F_1$. (We only need to have $\bar{h}(\bar{0})$ be something with $\mathrm{Eval}_L\ \bar{h}(\bar{0}) = \infty$.) For any interpretation I, the functional $\mathcal{F}_L$ will replace I_f by J_f for $J = \mathcal{F}_L(I)$, where $J_f = \mathcal{F}(I_f)$.

We do not consider the restriction to RDs in (*) to be very serious, since it is well-known that RDs can define all the partial recursive functions in terms of successor and equality [34, §9] and that RDs are generally quite convenient for defining functions in terms of other functions [33][34]. Under the restriction to integer data in [26], it appears likely that RDs suffice for defining all partial recursive functionals, but that has not been demonstrated.

CHAPTER 4
APPLICATIONS TO THE LAMBDA CALCULUS

This chapter applies the theory of general and subtree replacement systems to the full lambda calculus, including eta and delta rules as defined by Curry and Feys [14, Chap. 3]. In §9 we explain the lambda calculus informally and then formalize it in terms of general and subtree replacement systems. Our novel approach to alphabetic equivalence permits a higher degree of mathematical rigor than does the usual approach.

In §10 we prove the classical Church-Rosser theorem [14, Chap. 4] with a divide-and-conquer strategy. The lambda calculus is analyzed as a union of two systems: $\mathfrak{B}_\beta$ (defined by the beta rules) and $\mathfrak{B}_{\eta\delta}$ (defined by the eta and delta rules). New systems $\mathfrak{C}_\gamma$ and $\mathfrak{C}_\sigma$ are defined, so that $\mathfrak{B}_\beta$ can be simulated by the interaction of $\mathfrak{C}_\gamma$ and $\mathfrak{C}_\sigma$, yet these systems by themselves are much simpler than $\mathfrak{B}_\beta$.

The Main Theorem (5.6) implies that $\mathfrak{C}_\gamma$ $\mathfrak{C}_\sigma$, and $\mathfrak{B}_{\eta\delta}$ are all Church-Rosser. We derive the Church-Rosser property for $\mathfrak{B}_\beta$ from the properties of $\mathfrak{C}_\gamma$ and $\mathfrak{C}_\sigma$ by a special argument. After showing that $\mathfrak{B}_\beta$ and $\mathfrak{B}_{\eta\delta}$ are Church-Rosser, we use the Commutative Union Theorem (3.5) to conclude that the full lambda calculus is Church-Rosser.

8. Lambda Calculus

Analogies between the lambda calculus and programming languages have been discussed by many authors. Landin [29] even proposes defining the semantics of a programming language with this calculus. He considers a syntactic description of the usual sort together with a syntax-directed translator that maps programs to lambda expressions. The semantics of lambda expressions are already defined by the nondeterministic evaluation procedure specified by the formal "reduction" rules of Curry and Feys [14, Chap. 3].

Although Landin's paper and the references cited there suffer from some gaps and confusions, the approach does clarify many issues in programming language design, and our interest in the lambda calculus stems partly from this fact. (Wegbreit [53, pp. 132-138] has taken a similar view.) The lambda calculus is also interesting as a mathematical example. Unlike the McCarthy formalism for recursive definitions, the lambda calculus is complex enough to illustrate the whole-part theorems of §3. Unlike English or PL/1, it is also simple enough to be treated in detail without overwhelming the rest of our discussion. We begin with an informal description of the lambda calculus.

The standard mathematical approach to functions is to define them as sets of ordered pairs. The lambda calculus treats functions the way ordinary mathematics treats sets (or classes) -- as primitive entities that satisfy certain axioms but are not definable in terms of other entities. Functions and their arguments are objects on an equal footing. An object R may be "applied" to another object S, forming a third object $\gamma(R, S)$. New objects can also be "abstracted" by using "variables" x, y, z, ... and a special symbol λ. Suppose S is an expression that would name an object if all the occurrences of a variable x were to be replaced by the name of an object. Then the expression $\lambda(x, S)$ names an object T. For each object R, $\gamma(T, R)$ is the object named by S after x has been replaced by a name for R.

To illustrate the notation we suppose for the moment that the objects under consideration include the nonnegative integers and the addition map ADD. Then $\gamma(\mathrm{ADD}, 3)$ corresponds to the function $\{\langle k, 3+k\rangle \mid k \in \mathbb{N}\}$ while $\gamma(\gamma(\mathrm{ADD}, 3), 5)$ is the result of applying this function to 5. The equation

(9.1.1) $\gamma(\gamma(\mathrm{ADD}, 3), 5) = 8$

and an infinity of similar equations are approximately equivalent to

$$\mathrm{ADD} = \{\langle h, \langle k, h+k\rangle\rangle \mid h, k \in \mathbb{N}\}$$

which is only trivially different from the usual

$$\mathrm{ADD} = \{\langle (h, k), h+k\rangle \mid h, k \in \mathbb{N}\}.$$

The function $\{\langle h, h+h\rangle \mid h \in \mathbb{N}\}$ can be described by abstraction as $\lambda[x, \gamma(\gamma(\mathrm{ADD}, x), x)]$. An infinity of equations like

(9.1.2) $\gamma(\lambda[x, \gamma(\gamma(\mathrm{ADD}, x), x)], 3) = \gamma(\gamma(\mathrm{ADD}, 3), 3)$

express the meaning of the abstraction.

Since the variable x does not occur in ADD, the abstraction $\lambda(x, \gamma(\mathrm{ADD}, x))$ is a needlessly long description of ADD. The equation

(9.1.3) $\lambda(x, \gamma(\mathrm{ADD}, x)) = \mathrm{ADD}$

and similar equations collapse such abstractions.

The lambda calculus consists of a precise syntax for application and abstraction plus three sets of axioms. The three equations (9.1) illustrate the three kinds of axiom used. There is one rule of inference: from $R = S$ and $P = Q$, we may infer $R = T$, where T is the result of replacing P by Q somewhere in S. To prove a theorem of the form $R = 7$ amounts to evaluating the expression R and getting the answer 7. The study of this system is facilitated by using trees rather than parenthesized strings and a set of rules in an SRS rather than a set of equations. Our formulation is logically equivalent to that of Curry and Feys [14, Chap. 3] but more amenable to detailed mathematical analysis. Comparisons are given at the end of this section.

We will now begin constructing the SRS which will be used to formalize the lambda calculus. The notation developed in the next few paragraphs will be used throughout the chapter.

The vocabulary consists of a set C of <u>constants</u> denoting "known" objects, an infinite set X of <u>variables</u>, and two special symbols λ, γ.

$$V = C \cup X \cup \{\lambda, \gamma\} .$$

We require that C, X, and $\{\lambda, \gamma\}$ be mutually disjoint and specify a rank function $\rho : V \longrightarrow \mathbb{N}$ by assigning rank 2 to λ and γ, rank 0 to other symbols. The forest $\mathbb{F}$ of interest here is not all of $V_\#$, since $\lambda(R, S)$ is only significant when R consists of a single variable. In the fully abbreviated algebraic nomenclature of (4.10), $\mathbb{F}$ may be defined inductively as follows:

$$(C \cup X)_\# \subseteq \mathbb{F}$$

$$(\forall x \in X)(\forall S \in \mathbb{F})(\lambda(x, S) \in \mathbb{F})$$

$$(\forall R, S \in \mathbb{F})(\gamma(R, S) \in \mathbb{F}) .$$

The following family (9.2) of definitions deals with bound and free variables. The distinction between bound and free occurrences of variables in more familiar notations is discussed by Curry and Feys [14, Chap. 3]. For example, x has a free occurrence and y has a bound occurrence in the expression $\int_0^\pi \cos(x+y)\, dy$, while y has both kinds of occurrence in the expression $\int_0^\pi \cos(x+y)\, dy + y$. For $S \in \mathbb{F}$ and

$x \in X$, the set $S^{-1}(x)$ is divided into bound and free occurrences of x:

(9.2.1) $B_x S = \{n \in S^{-1}(x) \mid (\exists m \ \underline{\text{anc}}\ n)(Sm = \lambda \ \& \ S(m\cdot(0)) = x)\}$ (<u>bound</u> occurrences of x in S)

$F_x S = S^{-1}(x) - B_x S$ (<u>free</u> occurrences of x in S).

For $S \in \mathbb{F}$ and $m, n \in \text{Dom } S$ define

(9.2.2) n <u>is bound to</u> m <u>in</u> S iff $[m \ \underline{\text{anc}}\ n \ \& \ Sm = \lambda \ \& \ S(m\cdot(0)) = Sn \ \&$

$(\forall p \ \underline{\text{anc}}\ n)([Sp = \lambda \ \& \ S(p\cdot(0)) = Sn] \text{ implies } p \ \underline{\text{anc}}\ m)]$

so that n is bound to m when $n \in B_x S$ for some $x \in X$ and m is the closest ancestor of n involved in the definition of $B_x S$.

For example, let $y, z \in X$ with $y \neq z$ and let S be

(a) $\gamma(\lambda(z, \gamma(y, y)); \gamma(y, \lambda(z, \lambda(z, z))))$.

Then $B_y S$ is $\emptyset$ and $B_z S$ is $\{(0,0), (1,1,0), (1,1,1,0), (1,1,1,1)\}$. The node $(1,1,1,1)$ is bound to $(1,1,1)$ rather than to $(1,1)$. In this example, the set of variables that occur bound in S is $\text{BVbl } S = \{z\}$, while the set of variables that occur free in S is $\text{FVbl } S = \{y\}$. For any $S \in \mathbb{F}$ we define

(9.2.3) $\text{BVbl } S = \{x \in X \mid B_x S \neq \emptyset\}$

$\text{FVbl } S = \{x \in X \mid F_x S \neq \emptyset\}$.

For $S \in \mathbb{F}$ such that

(9.2.4) $\text{BVbl } S \cap \text{FVbl } S = \emptyset \ \&$

$(\forall m, n \in S^{-1}(\lambda))(S(m\cdot(0)) = S(n\cdot(0)) \text{ implies } m = n)$

we say that S is <u>alphanormal</u> and set

(9.2.5) $\mathbb{F}_0 = \{S \in \mathbb{F} \mid S \text{ is alphanormal}\}$.

No variable is both free and bound in an alphanormal tree. No two occurrences of any single variable are bound to distinct occurrences of λ in an alphanormal tree. Our example (a) satisfies the first of these criteria but not the second. The reasons for introducing $\mathbb{F}_0$ will emerge gradually, once certain equivalence relations have been defined. One of the properties of $\mathbb{F}_0$ can be stated immediately: subtrees of alphanormal trees are alphanormal. This fact will be used repeatedly in inductive arguments in the next section.

The result of substituting $R \in \mathbb{F}$ for each free occurrence of a variable x in $S \in \mathbb{F}$ will be called $[R/x]S$:

(9.2.6) $[R/x]S = S(F_xS \longleftarrow R)$.

For example, if R is $\gamma(y, z)$ and S is $\gamma(x, \gamma(\lambda(y, x), \lambda(x, z)))$, then

$$[R/x]S = \gamma(\gamma(y, z), \gamma(\lambda(y, \gamma(y, z)), \lambda(x, z))) .$$

Notice that the free variable y in R is "captured" by the occurrence $(1, 0)$ of λ in S, since $(1, 0)$ is an ancestor of the node $(1, 0, 1)$ in F_xS and $(1, 0, 0)$ is an occurrence of y in S. Although (0) is a free occurrence of y in R, the ocrresponding node $(1, 0, 1)\cdot(0)$ in $[R/x]S$ is a bound occurrence of y in $[R/x]S$. We will often wish to assume $\mathrm{FVbl}\ R \cap \mathrm{BVbl}\ S = \emptyset$ in order to avoid such "captures" of free variables.

The use of variables requires arbitrary choices. The expressions $\int_0^\pi \cos(x+y)\, dy$ and $\int_0^\pi \cos(x+z)\, dz$ differ only in an irrelevant alphabetic decision. Even free variables are ultimately used only to require that the same expression be substituted for a free variable at each of its occurrences. Intuitively, $\int_0^\pi \cos(x+y)\,dy$ and $\int_0^\pi \cos(z+y)\,dy$ are only trivially different, although only changes in bound variables are commonly recognized as "trivial" in logic.

Here it will be convenient to recognize two sorts of alphabetic equivalence Trees are "weakly alphaequivalent" if they are the same except for the choices of variables. They are "strongly alphaequivalent" if they are the same except for the choices of bound variables. To aid in defining these relations we first define a relation $\xleftrightarrow[R]{}$ on $\mathbb{N}^*$ for each $R \in \mathbb{F}$.

(9.3.1) $m \xleftrightarrow[R]{} n$ iff

$$[(\exists x \in X)(m, n \in F_x R) \text{ or } (\exists p \in R^{-1}(\lambda))(m, n \text{ are bound to } p \text{ in } R)].$$

The "links" of Bourbaki [5, p. 16] inspired the $\xleftrightarrow[R]{}$ notation. Note that only one of the alternatives in (9.3.1) can hold. Although defined as a relation on $\mathbb{N}^*$, $\xleftrightarrow[R]{}$ is in fact a subset of $R^{-1}(X) \times R^{-1}(X)$. Since every node in $R^{-1}(X)$ is a free or a bound occurrence of the variable that labels it, $\xleftrightarrow[R]{}$ is reflexive. It is obviously symmetric. Suppose $m \xleftrightarrow[R]{} n \xleftrightarrow[R]{} p$. We claim that $m \xleftrightarrow[R]{} p$. There are two cases. If $m, n \in F_x R$ for some $x \in X$, then $n \xleftrightarrow[R]{} p$ implies that $n, p \in F_x R$, so that $m, p \in F_x R$ and $m \xleftrightarrow[R]{} p$. On the other hand, if m and n are bound to q for some q in $R^{-1}(\lambda)$, then $n \xleftrightarrow[R]{} p$ implies that n and p are bound to q, so that m and p are bound to q and $m \xleftrightarrow[R]{} p$. Thus $\xleftrightarrow[R]{}$ is an equivalence relation on $R^{-1}(X)$. Another technical convenience in working with the forest $\mathbb{F}_0$ of alphanormal trees defined in (9.2.5) is the fact that, whenever $R \in \mathbb{F}_0$ and $m, n \in \text{Dom } R$, then

$$m \xleftrightarrow[R]{} n \quad \text{iff} \quad Rm = Rn.$$

For alphanormal trees, nodes labelled by variables are equivalent iff they have the same label.

By a straightforward case analysis (given in Appendix A) we can show that, whenever $R \in \mathbb{F}$ and $m, n, p \in \text{Dom } R$ with m anc n and m anc p, then

(9.3.2) $\quad n \underset{R}{\longleftrightarrow} p$ iff $n/m \underset{R/m}{\longleftrightarrow} p/m$.

Now the two kinds of alphaequivalence can be defined. We use the usual notation $f \restriction A$ for the restriction of a function f to arguments in A.

(9.4) Definition. For $R \in \mathbb{F}$ let the frame Frame R be $R \restriction R^{-1}(V\text{-}X)$. For $R, S \in \mathbb{F}$ define weak alphaequivalence ($\simeq$) and strong alphaequivalence ($\cong$) by

(1) $\quad R \simeq S$ iff $[\text{Frame } R = \text{Frame } S \;\&\; (\underset{R}{\longleftrightarrow}) = (\underset{S}{\longleftrightarrow})]$

(2) $\quad R \cong S$ iff $[R \simeq S \;\&\; (\forall x \in X)(\forall n \in F_x R)(Sn = x)]$.

The technical motivation for introducing weak alphaequivalence lies in the inductive properties anticipated in the lemma (9.3.2). Recall that any tree in $\mathbb{F}$ with more than one node has the form $\lambda(x, S)$ or $\gamma(R, S)$ with $R, S \in \mathbb{F}$ and $x \in X$. We can almost say that $\lambda(x, S) \simeq \lambda(x', S')$ iff $S \simeq S'$ and that $\gamma(R, S) \simeq \gamma(R', S')$ iff $R \simeq S$ and $R' \simeq S'$: complex trees are weakly alphaequivalent iff all the corresponding subtrees are weakly alphaequivalent. A bothersome complication arises from the need to make free variable changes in one subtree consistent with free variable changes in another. This requirement is stated formally by the following lemma.

(9.5) Lemma. Let $R, R', S, S' \in \mathbb{F}$ and $x, x' \in X$. Then

(1) $\lambda(x, S) \simeq \lambda(x', S')$ iff $(S \simeq S' \ \& \ F_x S = F_{x'} S')$

(2) $\gamma(R, S) \simeq \gamma(R', S')$ iff

$$[R \simeq R' \ \& \ S \simeq S' \ \&$$
$$(\forall y, y' \in X)(F_y R = F_{y'} R' \neq \emptyset \text{ implies } F_y S = F_{y'} S')] . \blacksquare$$

The proof is a straightforward application of the definition of weak alphaequivalence (9.4.1) and the lemma (9.3.2). Details are in Appendix A.

At first glance the clause

$$(\forall y, y' \in X)(F_y S = F_{y'} S' \neq \emptyset \text{ implies } F_y R = F_{y'} R')$$

seems to be missing from the right half of (9.5.2), but it follows from the right half as stated.

The following lemma relates alphabetic changes in trees of the form

$$T = S(m_0 \longleftarrow R_0) \ldots (m_{-1} \longleftarrow R_{-1})$$

to alphabetic changes in the appropriate trees $S, R_0, \ldots, R_{-1}$.

(9.6) <u>Lemma.</u> Let $x, x' \in X$; $K \in \mathbb{N}$; $S, S', R_0, R'_0, \ldots, R_{K-1}, R'_{K-1} \in \mathbb{F}$; $m_0, \ldots, m_{K-1} \in \mathbb{N}^*$ (distinct nodes). Suppose

(1) $F_x S = \{m_k | k < K\} = F_{x'} S'$

(2) $(\forall k < K)(\text{FVbl } R_k \cap (\text{BVbl } S \cup \{x\}) = \emptyset)$

(3) $(\forall k < K)(\text{FVbl } R'_k \cap (\text{BVbl } S' \cup \{x'\}) = \emptyset)$.

Then the statement

(4) $S(m_0 \longleftarrow R_0) \ldots (m_{-1} \longleftarrow R_{-1}) \simeq S'(m_0 \longleftarrow R'_0) \ldots (m_{-1} \longleftarrow R'_{-1})$

holds iff the following statements hold:

(5) $S \simeq S'$ & $(\forall k < K)(R_k \simeq R'_k)$

(6) $(\forall y, y' \in X)[F_y S = F_{y'} S' \neq \emptyset$ implies $(\forall k < K)(F_y R_k = F_{y'} R'_k)]$

(7) $(\forall y, y' \in X)(\forall j, k < K)(F_y R_j = F_{y'} R'_j \neq \emptyset$ implies $F_y R_k = F_{y'} R'_k)$.

Proof: Let T, T' be the left and right halves of (4). We assume (4) and derive (5), (6), (7). Intuitively, (6) and (7) say that the changes in free variables made in passing from S, $R_0, \ldots, R_{-1}$ to S', $R'_0, \ldots, R'_{-1}$ are consistent.

Obviously Frame S = Frame S'. Now suppose $m \underset{S}{\longleftrightarrow} n$. We show $m \underset{S'}{\longleftrightarrow} n$. If $m \in F_x S$ then $m, n \in F_{x'} S'$ by (1) and so $m \underset{S'}{\longleftrightarrow} n$. Otherwise we have $m, n \notin F_x S$ and then $m, n \notin F_{x'} S'$ by (1). In the definition (9.3.1), m and n may both be free occurrences of a variable other than x or m and n may both be bound to a single occurrence of λ. Both these conditions are unaffected by substitutions at nodes in $F_x S$, so $m \underset{T}{\longleftrightarrow} n$. Therefore $m \underset{T'}{\longleftrightarrow} n$. But this implies $m \underset{S'}{\longleftrightarrow} n$ because both conditions in the definition (9.3.1) are unaffected by substitutions at nodes in $F_{x'} S'$. We have shown that $(\underset{S}{\longleftrightarrow}) \subseteq (\underset{S'}{\longleftrightarrow})$. Similarly, $(\underset{S'}{\longleftrightarrow}) \subseteq (\underset{S}{\longleftrightarrow})$, so that $S \simeq S'$. For each $k < K$ we show that $R_k \simeq R'_k$ with the help of the lemma (9.3.2). This proves (5).

Suppose $y, y' \in X$ with $F_y S = F_{y'} S' \neq \emptyset$. For $k < K$ we show $F_y R_k = F_{y'} R'_k$. By (1) we have $y = x$ iff $y' = x'$, and in that case $F_y R_k = \emptyset = F_{y'} R'_k$ by (2) and (3). Now we assume $y \neq x$ and $y' \neq x'$. Let $p \in F_y S$, so that $p \in F_y T$ and $p \in F_{y'} T'$ also.

$$
\begin{aligned}
m_k \cdot F_y R_k &= F_y T \cap m_k \cdot \mathbb{N}^* && \text{by (2)} \\
&= \{n \in m_k \cdot \mathbb{N}^* \mid n \xleftrightarrow[T]{} p\} && \text{by def. of } \xleftrightarrow[T]{} \text{ (9.3.1)} \\
&= \{n \in m_k \cdot \mathbb{N}^* \mid n \xleftrightarrow[T']{} p\} && \text{by (4)} \\
&= F_{y'} T' \cap m_k \cdot \mathbb{N}^* && \text{by def. of } \xleftrightarrow[T']{} \text{ (9.3.1)} \\
&= m_k \cdot F_{y'} R'_k && \text{by (3)}
\end{aligned}
$$

This proves (6).

Suppose $y, y' \in X$ and $j, k < K$ with $F_y R_j = F_{y'} R'_j \neq \emptyset$. We show $F_y R_k = F_{y'} R'_k$ by reasoning like that for (6) with a node $m_j \cdot q$ for $q \in F_y R_j$ in the role of p in the proof of (6). This proves (7).

Now we assume (5), (6), (7) and derive (4). From (5) we get Frame T = Frame T'. Now we suppose $m \xleftrightarrow[T]{} n$ and show $m \xleftrightarrow[T']{} n$.

<u>Case 1</u> $(m \in \text{Dom } S - F_x S)$.

<u>Case 1.1</u> $(n \in \text{Dom } S - F_x S)$. Then $m \xleftrightarrow[S]{} n$, so $m \xleftrightarrow[S']{} n$ by (5). Therefore $m \xleftrightarrow[T']{} n$.

<u>Case 1.2</u> $((\exists k < K)(m_k \underline{\text{ anc }} n)$; choose one). Let $y = Sm = Tm$, so that $m, n \in F_y T$ by (2) and the definition (9.3.1). Let $y' = S'm = T'm$, so that $m, n \in F_{y'} T'$ follows from (5) and (6) and then (3). Therefore $m \xleftrightarrow[T']{} n$.

<u>Case 2</u> $((\exists j < K)(m_j \underline{\text{ anc }} m)$; choose one).

<u>Case 2.1</u> $(n \in \text{Dom } S - F_x S)$. Similar to Case 1.2.

<u>Case 2.2</u> $((\exists k < K)(m_k \underline{\text{ anc }} n)$; choose one). If $j = k$ we can show $m \xleftrightarrow[T']{} n$ by (5) and two applications of the lemma (9.3.2), so we may assume $j \neq k$. By (2), some $y \in X$ has $m, n \in F_y T$. Let $y' = T'm$. Then $m, n \in F_{y'} T'$ by (3), (5), and (7).

We have shown that $(\xleftrightarrow[T]{}) \subseteq (\xleftrightarrow[T']{})$. Similarly, $(\xleftrightarrow[T']{}) \subseteq (\xleftrightarrow[T]{})$. Therefore $T \simeq T'$.∎

The lambda calculus can now be defined.

(9.7) <u>Definition</u>. A <u>full lambda calculus</u> is any SRS $\mathfrak{C}_\lambda = (V, \mathbb{F}, \underset{\lambda}{\Longrightarrow}, \mathbb{R}_\lambda)$ such that

(1) $\mathbb{R}_\lambda = \mathbb{R}_\beta \cup \mathbb{R}_\eta \cup \mathbb{R}_\delta$

where

(2) $\mathbb{R}_\beta = \{\gamma(\lambda(x, S), R) \longrightarrow [R/x]S \mid \text{FVbl } R \cap (\text{BVbl } S \cup \{x\}) = \emptyset\}$

(3) $\mathbb{R}_\eta = \{\lambda(x, \gamma(R, x)) \longrightarrow R \mid F_x R = \emptyset\}$

(4) $\mathbb{R}_\delta \subseteq \{\gamma(R, S) \longrightarrow T \mid R(\ ) \neq \lambda \ \& \ \text{FVbl } R = \text{FVbl } S = \text{FVbl } T = \emptyset\}$

(5) $(\forall R, R', S, S' \in \mathbb{F})(\forall n \in \text{Dom } R)$

$([R \longrightarrow S \in \mathbb{R}_\delta \ \& \ R/n \cong R'] \text{ implies}$

$[R' \longrightarrow S' \in \mathbb{R}_\lambda \text{ iff } (n = (\ ) \ \& \ S' = S)])$.

The substitution $[R/x]S$ in (2) above is defined by (9.2.6). By requiring $\text{FVbl } R \cap \text{BVbl } S = \emptyset$, we have prevented captures of free variables. The additional requirement that $\text{FVbl } R \cap \{x\} = \emptyset$ is a minor technical convenience. Notice that $\text{FVbl } R \cap (\text{BVbl } S \cup \{x\}) = \emptyset$ will be true if $\gamma(\lambda(x, S), R)$ is alphanormal in the sense of (9.2.4). Notice also that (2) covers our original intuitive example (9.1.2):

$$\gamma(\lambda[x, \gamma(\gamma(\text{ADD}, x), x)], 3) \longrightarrow \gamma(\gamma(\text{ADD}, 3), 3).$$

Comparison of $\lambda(x, \gamma(x, x))$ with x illustrates why we should only "abbreviate" $\lambda(x, \gamma(R, x))$ as R when $F_x R = \emptyset$. Notice that (3) covers our original intuitive example (9.1.3):

$$\lambda(x, \gamma(\text{ADD}, x)) \longrightarrow \text{ADD}.$$

The differences between one lambda calculus and another are in $\mathbb{R}_\delta$: any choice of $\mathbb{R}_\delta$ that satisfies (4) and (5) is allowed. Since most applications involve just one system $\mathfrak{C}_\lambda$, we will speak of "the" lambda calculus, as is customary. Our original example

$$\gamma(\gamma(\text{ADD}, 3), 5) \longrightarrow 8$$

from (9.1.1) does meet the restrictions. To see the effect of (5), suppose first that $n = (\,)$. Then the choices of bound variables in R are irrelevant to any rule $R \longrightarrow S$ in $\mathbb{R}_\delta$: $R' \longrightarrow S$ is still in $\mathbb{R}_\delta$ if $R \cong R'$, and there is no other rule $R' \longrightarrow S'$ with $S' \neq S$ in $\mathbb{R}_\lambda$. Now suppose $n \neq (\,)$. Then no proper subtree of R in a rule $R \longrightarrow S$ in $\mathbb{R}_\delta$ can be replaceable, even after changes of bound variables. This requirement is similar to the bottom-up evaluation of given functions in Chapter 3.

The sets $\mathbb{R}_\beta$, $\mathbb{R}_\eta$ and $\mathbb{R}_\delta$ are partial functions on trees by (2), (3), and (5). By $\gamma \neq \lambda$, no $\varphi \in V_*$ can be the left half of both a beta and an eta rule or the left half of both a delta and an eta rule. By the condition $R(\,) \neq \lambda$ in (4), no $\varphi \in V_*$ can be the left half of both a beta rule and a delta rule. Therefore $\mathbb{R}_\lambda$ is a union of partial functions with disjoint domains and so $\mathbb{R}_\lambda$ is a partial function. The system $\mathfrak{C}_\lambda$ is unequivocal. Unfortunately $\mathfrak{C}_\lambda$ is not quite the system we wish to use. As the following counterexample shows, $\mathfrak{C}_\lambda$ is not Church-Rosser.

Let x, y, z be distinct variables and set

$$R = \gamma(\lambda(x, \gamma(\lambda(z, \gamma(z, x)), y)), \lambda(y, y))$$

$$S = \gamma(\lambda(z, \gamma(z, \lambda(y, y))), y))$$

$$S' = \gamma(\lambda(x, \gamma(y, x)), \lambda(y, y))$$

$$S'' = \gamma(y, \lambda(y, y)).$$

By applying a rule from $\mathbb{R}_\beta$ at ( ) in R we can show that $R \underset{\lambda}{\Rightarrow} S$. By applying a rule from $\mathbb{R}_\beta$ at (0, 1) in R we can show that $R \underset{\lambda}{\Rightarrow} S'$. By applying a rule from $\mathbb{R}_\beta$ at ( ) in S′ we can show that $S' \underset{\lambda}{\Rightarrow} S''$. Since $R \overset{*}{\underset{\lambda}{\Rightarrow}} S$ and $R \overset{*}{\underset{\lambda}{\Rightarrow}} S''$ while S and S′ are irreplaceable, both are normal forms of R. But $S \neq S'$. Normal forms are not unique in $\mathfrak{C}_\lambda$.

In our counterexample the tree S is irreplaceable because

$$y \in \mathrm{FVbl}[y] \cap (\mathrm{BVbl}[\gamma(z, \lambda(y, y))] \cup \{z\}) .$$

In light of the arbitrary nature of bound variable choices that we discussed in connection with the definitions of $\cong$ and $\simeq$ in (9.4), this difficulty seems rather insubstantial. Consider a new variable w and set

$$T = \gamma(\lambda(z, \gamma(z, \lambda(w, w)), y)) ,$$

so that $S \cong T$. By applying a rule from $\mathbb{R}_\beta$ at ( ) in T, we can show that

$$T \underset{\lambda}{\Rightarrow} \gamma(y, \lambda(w, w)) .$$

By deliberately confusing S with T and $\gamma(y, \lambda(w, w))$ with S″, we can blunt the force of the counterexample. The tree R does have a unique normal form, at least when we ignore differences between strongly alphaequivalent trees. The precise way to "identify" equivalent objects is to pass to equivalence classes. We replace $\mathfrak{C}_\lambda$ by a GRS $\mathfrak{B}_\lambda = (\mathbb{E}, >_\lambda)$, where $\mathbb{E}$ is the set of all strong alphaequivalence classes of trees in the forest $\mathbb{F}$ of lambda expressions. The replacement relation between classes of trees is the natural relation induced by the replacement relation between trees: for all $\mathcal{R}, \mathcal{S} \in \mathbb{E}$,

$$\mathcal{R} >_\lambda \mathcal{S} \quad \text{iff} \quad (\exists R \in \mathcal{R})(\exists S \in \mathcal{S}) \; (R \underset{\lambda}{\Rightarrow} S) .$$

The classical Church-Rosser theorem [14, Chap. 4] asserts only that $\mathfrak{B}_\lambda$ is Church-Rosser. (The fact that $\mathfrak{B}_\lambda$ is indeed the GRS considered in [14] may not be obvious. A proof is given in Appendix B.)

For later convenience we state the formal definition in terms of any pair of relations $\underset{i}{\Longrightarrow}$ and $\mathbb{R}_i$ such that the 4-tuple $\mathfrak{C}_i = (V, \mathbb{F}, \underset{i}{\Longrightarrow}, \mathbb{R}_i)$ is an SRS. The examples of interest to us will involve $\mathbb{R}_i = \mathbb{R}_\lambda$, $\mathbb{R}_i = \mathbb{R}_\beta$, and $\mathbb{R}_i = \mathbb{R}_\eta \cup \mathbb{R}_\delta$.

(9.8) <u>Definition</u>. Let $\mathbb{E}$ be the set of all strong alphaequivalence classes of trees in $\mathbb{F}$. For each pair of relations $\underset{i}{\Longrightarrow}$ and $\mathbb{R}_i$ such that $\mathfrak{C}_i = (V, \mathbb{F}, \underset{i}{\Longrightarrow}, \mathbb{R}_i)$ is an SRS, define a GRS $\mathfrak{B}_i = (\mathbb{E}, >_i)$ by setting, for all $\mathcal{R}, \mathcal{S} \in \mathbb{E}$,

$$\mathcal{R} >_i \mathcal{S} \text{ iff } (\exists R \in \mathcal{R})(\exists S \in \mathcal{S})(R \underset{i}{\Longrightarrow} S).$$

In the next section we will prove that $\mathfrak{B}_\lambda$ is Church-Rosser. In order to represent manipulations of equivalence classes by manipulations of "typical" members of the classes, we will use the alphanormal trees defined by (9.2.4). Every tree is strongly alphaeqivalent to an alphanormal tree, so every class $\mathcal{R} \in \mathbb{E}$ has an alphanormal representative $R \in \mathcal{R} \cap \mathbb{F}_0$. The convenient properties of $\mathbb{F}_0$ will expedite our proofs. To prepare for the next section we will now state an elementary lemma relating $\mathbb{R}_\lambda$ to weak alphaequivalence. (We use <u>weak</u> alphaequivalence in order to facilitate induction arguments.)

Intuitively, the rules of $\mathfrak{C}_\lambda$ are insensitive to changes in bound and free variables, so long as bound and free roles for each variable do not clash. Formally, the following is proven in Appendix A.

(9.9) <u>Lemma</u>. Let $i \in \{\beta, \eta, \delta\}$; $\varphi \longrightarrow \psi \in \mathbb{R}_i$; $\varphi \simeq \varphi'$. If $i = \beta$, let $\varphi' \in \mathbb{F}_0$ also. Then there is $\psi' \in \mathbb{F}$ such that $\psi \simeq \psi'$ and

(1) $\varphi' \longrightarrow \psi' \in \mathbb{R}_i$

(2) $(\forall y, y' \in X)(F_y\psi = F_{y'}\psi' \neq \emptyset$ implies $F_y\varphi = F_{y'}\varphi' \neq \emptyset)$. ■

For each change from a variable x in φ to a variable x' in φ' at a node n, we simply replace x by x' at the appropriate corresponding nodes (if any) in ψ. The verification that the resulting tree ψ' has the desired properties is trivial, once we can say where to make the changes in ψ. To each rule $\varphi \longrightarrow \psi \in \mathbb{R}_\lambda$ we therefore assign a map $r = r[\varphi, \psi]$ that assigns to each $n \in \text{Dom}\ \varphi$ a subset of $\text{Dom}\ \psi$. The three kinds of rules lead to three kinds of pseudoresidue map.

(9.10.1) For $\varphi \longrightarrow \psi = \gamma(R, S) \longrightarrow T \in \mathbb{R}_\delta$: $r(n) = \emptyset$

(9.10.2) For $\varphi \longrightarrow \psi = \lambda(x, \gamma(R, x)) \longrightarrow R \in \mathbb{R}_\eta$:
$r(n) = \underline{\text{if}}\ (1,0)\ \underline{\text{anc}}\ n\ \underline{\text{then}}\ \{n/(1,0)\}\ \underline{\text{else}}\ \emptyset$

(9.10.3) For $\varphi \longrightarrow \psi = \gamma(\lambda(x, S), R) \longrightarrow [R/x]S \in \mathbb{R}_\beta$:
$r(n) = \underline{\text{if}}\ (0,1)\ \underline{\text{anc}}\ n\ \&\ n/(0,1) \notin F_xS\ \underline{\text{then}}\ \{n/(0,1)\}$
$\underline{\text{else}}\ \underline{\text{if}}\ (1)\ \underline{\text{anc}}\ n\ \underline{\text{then}}\ \{p \cdot (n/(1)) \mid p \in F_xS\}$
$\underline{\text{else}}\ \emptyset$

Because independent nodes do not always have independent residues in the last equation, $r[\varphi, \psi]$ is not a genuine residue map when $\varphi \longrightarrow \psi \in \mathbb{R}_\beta$.

We have defined the lambda calculus and stated some of its elementary properties. The "obs" in the system of Curry and Feys [14, Chap. 3] correspond to trees in $\mathbb{F}$ in the obvious way, and it is not hard to show that "α-convertibility" [14, §3D3] corresponds to strong alphaequivalence. When the "reducibility relation $\geq$" [14, §3D3] is lifted to equivalence classes in the manner of Definition 9.8, it corresponds to $\overset{*}{>}_\lambda$. Details are in Appendix B. The reasons for departing from [14] are sketched below.

A rudimentary tree formalism is introduced in [14, §2B] and the morphology of the "obs" is stated abstractly [14, §3C2], but these gestures have no effect on the proof of the Church-Rosser theorem [14, Chap. 4]. No workable methods for calculating with trees and nodes to make arguments flow are provided in [14].

The "α-convertibility" approach to changes of bound variables is simple to define but awkward to use. In practice, writers on lambda calculus must often resort to vagueness and dubious "without loss of generality" claims. Weak and strong alphaequivalence let us say exactly what is being assumed about changes of bound variables. If someone questions these assumptions, proofs can be supplied. The calculations are tedious (as in Lemma 9.6), but no new ideas are required.

Curry and Feys avoid assuming FVbl R $\cap$ BVbl S = $\emptyset$ in defining $\mathbb{R}_\beta$ by means of a definition of substitution [14, §3E1] more complicated than our (9.2.6). Finding no technical advantages in shifting the complexity from $\mathbb{R}_\beta$ to substitution in our treatment, we have left the irritant where it first appeared. Our definitions of substitution for free variables and $\mathbb{R}_\beta$ are very close to the original formulation of Church [12, §§4, 7].

10. Classical Church-Rosser Theorem

We work toward the proof that the system $\mathfrak{B}_\lambda$ defined by (9.7) and (9.8) is Church-Rosser. Recall that $\mathfrak{B}_\lambda = (\mathbb{E}, >_\lambda)$, where $\mathbb{E}$ is the set of all strong alphaequivalence classes of trees in $\mathbb{F}$ and the relation $>_\lambda$

on $\mathbb{E}$ is induced by the relation $\xrightarrow[\lambda]{}$ on $\mathbb{F}$:

$$\mathcal{R} >_\lambda \mathcal{S} \quad \text{iff} \quad (\exists R \in \mathcal{R})(\exists S \in \mathcal{S})(R \xrightarrow[\lambda]{} S).$$

The definitions of $\mathbb{R}_\beta$, $\mathbb{R}_\eta$, and $\mathbb{R}_\delta$ in (9.7) lead to SRSs

$$\mathfrak{C}_\beta = (V, \mathbb{F}, \xrightarrow[\beta]{}, \mathbb{R}_\beta)$$

$$\mathfrak{C}_{\eta\delta} = (V, \mathbb{F}, \Longrightarrow, \mathbb{R}_{\eta\delta}) \quad \text{where} \quad \mathbb{R}_{\eta\delta} = \mathbb{R}_\eta \cup \mathbb{R}_\delta .$$

The corresponding GRSs $\mathfrak{B}_\beta$ and $\mathfrak{B}_{\eta\delta}$ defined by (9.9) form a family of GRSs $\{\mathfrak{B}_a \mid a \in \{\beta, \eta\delta\}\}$ whose union is $\mathfrak{B}_\lambda$. Lemma 10.2 asserts that $\mathfrak{B}_{\eta\delta}$ is Church-Rosser, Lemma 10.3 asserts that $\mathfrak{B}_\beta$ commutes with $\mathfrak{B}_{\eta\delta}$, and Theorem 10.11 asserts that $\mathfrak{B}_\beta$ is Church-Rosser, so the Church-Rosser property for $\mathfrak{B}_\lambda$ follows from the Commutative Union Theorem (3.5).

In order to establish the results mentioned above we will represent relations between strong alphaequivalence classes by relations between certain trees in the classes. The following lemma is the formal basis for these representations.

(10.1) Lemma. Let $i, j \in \{\beta, \eta, \delta\}$; $\mathcal{R}, \mathcal{S}, \mathcal{S}' \in \mathbb{E}$. Suppose $\mathcal{R} >_i \mathcal{S}$ and $\mathcal{R} >_j \mathcal{S}'$. Then there are $R, S, S' \in \mathbb{F}$ such that

$$R \in \mathcal{R} \cap \mathbb{F}_0 \ \& \ S \in \mathcal{S} \ \& \ S' \in \mathcal{S}'$$
$$R \xrightarrow[i]{} S \ \& \ R \xrightarrow[j]{} S'.$$

Proof: By Definition 9.8 there are $R_1, R_1' \in \mathcal{R}$; $S_1 \in \mathcal{S}$; $S_1' \in \mathcal{S}'$ with $R_1 \xrightarrow[i]{} S_1$ and $R_1' \xrightarrow[j]{} S_1'$. Let R be any alphanormal tree in $\mathcal{R}$.

Let a rule $\varphi \longrightarrow \psi$ be applied at a node n in R_1 to form S_1, so that $R_1/n = \varphi$. By $|n|$ applications of Lemma 9.5 in the "only if" direction, R/n is an alphanormal tree φ' such that $\varphi \simeq \varphi'$. By Lemma 9.9 there is a tree $\psi' \in \mathbb{F}$ such that $\psi \simeq \psi'$, $\varphi' \longrightarrow \psi' \in \mathbb{R}_i$, and

(1) $(\forall y, y' \in X)(F_y\psi = F_{y'}\psi' \neq \emptyset$ implies $F_y\varphi = F_{y'}\varphi' \neq \emptyset)$.

Set $S = R(n \longleftarrow \psi')$, so that $R \underset{i}{\Longrightarrow} S$. We wish to show that $S_1 \cong S$. By $S_1 = R_1(n \longleftarrow \psi)$, $S = R(n \longleftarrow \psi')$, $R_1 \simeq R$, and $\psi \simeq \psi'$, we can show that $S_1 \simeq S$ by $|n|$ applications of Lemma 9.5 in the "if" direction. (The consistence of free variable changes is assured by (1) and $R_1 \simeq R$.) Moreover, the changes in free variables between S_1 and S are the same as those between R_1 and R:

$$(\forall y, y' \in X)(F_yS_1 = F_{y'}S \neq \emptyset \text{ implies } F_yR_1 = F_{y'}R \neq \emptyset).$$

By $R_1 \cong R$, $F_yR_1 = F_{y'}R \neq \emptyset$ implies that $y = y'$. Therefore

$$(\forall y, y' \in X)(F_yS_1 = F_{y'}S \neq \emptyset \text{ implies } y = y').$$

Together with $S_1 \simeq S$, this implies that $S_1 \cong S$.

By repeating the above argument with R_1' in place of R_1 and S_1' in place of S_1, we find that some $S' \in \mathbb{F}$ has $R \underset{j}{\Longrightarrow} S'$ and $S_1' \cong S'$. Therefore R, S, and S' have the desired properties. ∎

(10.2) Lemma. Let $\mathbb{R}_{\eta\delta} = \mathbb{R}_\eta \cup \mathbb{R}_\delta$ and let $\mathfrak{C}_{\eta\delta}$ be the SRS $(V, \mathbb{F}, \underset{\eta\delta}{\Longrightarrow}, \mathbb{R}_{\eta\delta})$. The GRS $\mathfrak{B}_{\eta\delta}$ is Church-Rosser.

Proof: First we show that $\mathfrak{C}_{\eta\delta}$ is Church-Rosser. By $\gamma \neq \lambda$ and the definitions of $\mathbb{R}_\eta$ and $\mathbb{R}_\delta$ in (9.7), $\mathbb{R}_{\eta\delta}$ is a partial function. Therefore $\mathfrak{C}_{\eta\delta}$ is unequivocal. We have assigned a map

$$r[\varphi, \psi]: \text{Dom } \varphi \longrightarrow 2^{\text{Dom } \psi}$$

to each $\varphi \longrightarrow \psi \in \mathbb{R}_{\eta\delta}$ by (9.10.1) and (9.10.2). It is easy to check that $r[\varphi, \psi]$ is a residue map for each $\varphi \longrightarrow \psi$ and that $\mathfrak{C}_{\eta\delta}$ is closed in

Definition 5.4. By the Main Theorem (5.6), $\mathfrak{C}_{\eta\delta}$ is Church-Rosser.

Now suppose that $\mathcal{R}, \mathcal{S}, \mathcal{S}' \in \mathbb{E}$ with $\mathcal{R} \overset{*}{>}_{\eta\delta} \mathcal{S}$ and $\mathcal{R} \overset{*}{>}_{\eta\delta} \mathcal{S}'$. By Lemma 10.1 and similar reasoning, there are $R, S, S' \in \mathbb{F}$ such that $R \overset{*}{\underset{\eta\delta}{\Longrightarrow}} S \in \mathcal{S}$ and $R \overset{*}{\underset{\eta\delta}{\Longrightarrow}} S' \in \mathcal{S}'$. Therefore some $T \in \mathbb{F}$ has $S \overset{*}{\underset{\eta\delta}{\Longrightarrow}} T$ and $S' \overset{*}{\underset{\eta\delta}{\Longrightarrow}} T$ because $\mathfrak{C}_{\eta\delta}$ is Church-Rosser. Letting $\mathcal{T}$ be the strong alphaequivalence class of T, we have $\mathcal{S} \overset{*}{>}_{\eta\delta} \mathcal{T}$ and $\mathcal{S}' \overset{*}{>}_{\eta\delta} \mathcal{T}$. ∎

(10.3) <u>Lemma.</u> The GRSs $\mathfrak{B}_\beta$ and $\mathfrak{B}_{\eta\delta}$ commute.

<u>Proof:</u> We use the Commutativity Lemma (3.6). Suppose $\mathcal{R}, \mathcal{S}_1, \mathcal{S}_2 \in \mathbb{E}$ with $\mathcal{R} >_\beta \mathcal{S}_1$ and $\mathcal{R} >_{\eta\delta} \mathcal{S}_2$. We will show that some $\mathcal{T} \in \mathbb{E}$ has $\mathcal{S}_1 \overset{*}{>}_{\eta\delta} \mathcal{T}$ and $\mathcal{S}_2 \overset{=}{>}_\beta \mathcal{T}$. By Lemma 10.1, there are $R \in \mathcal{R} \cap \mathbb{F}_0$; $S_1 \in \mathcal{S}_1$; $S_2 \in \mathcal{S}_2$ with $R \underset{\beta}{\Longrightarrow} S_1$ and $R \underset{\eta\delta}{\Longrightarrow} S_2$. Let rules $\varphi_1 \longrightarrow \psi_1$ and $\varphi_2 \longrightarrow \psi_2$ be applied at nodes n_1, n_2 to derive S_1 and S_2, so that $S_1 = R(n_1 \longleftarrow \psi_1)$ and $S_2 = R(n_2 \longleftarrow \psi_2)$.

<u>Case 1</u> $(n_1 \perp n_2)$. Let $T = R(n_1 \longleftarrow \psi_1)(n_2 \longleftarrow \psi_2)$ and let $\mathcal{T} \in \mathbb{E}$ with $T \in \mathcal{T}$.

<u>Case 2</u> (n_1 <u>anc</u> n_2). By the definition of $\mathbb{R}_\beta$ (9.7.2), there are $x \in X$ and $P, Q \in \mathbb{F}_0$ with

$$\varphi_1 = \gamma(\lambda(x, Q), P) \quad \& \quad \psi_1 = [P/x]Q$$

while, because $R \in \mathbb{F}_0$,

$$\text{FVbl } P \cap (\text{BVbl } Q \cup \{x\}) = \emptyset .$$

<u>Case 2.1</u> $(n_1 \cdot (0) = n_2)$. Then $\varphi_2 \longrightarrow \psi_2 \in \mathbb{R}_\eta$ with

$$\varphi_2 = \lambda(x, Q) = \lambda(x, \gamma(\psi_2, x)) \quad \& \quad F_x \psi_2 = \emptyset .$$

It is easy to compute that $S_1 = S_2$. Let $\mathcal{T} \in \mathbb{E}$ with $S_1 \in \mathcal{T}$.

<u>Case 2.2</u> ($n_1 \cdot (0,1)$ <u>anc</u> n_2). Let $m = n_2/(n_1 \cdot (0,1))$, so that $\varphi_2 = Q/m$ and $r(n_2/n_1) = \{m\}$ in (9.10.3). We claim that $\varphi_1 \longrightarrow \psi_1$ is pseudoclosed at $(n_2/n_1, \{m\})$ with respect to $(\mathbb{R}_\beta, \mathbb{R}_{\eta\delta})$. Let $\tilde{\psi}$ be $[P/x]\psi_2$.
In Definition 5.7 we need

(1) $$\varphi_1(n_2/n_1 \longleftarrow \psi_2) \longrightarrow \psi_1(\{m\} \longleftarrow \tilde{\psi}) \in \mathbb{R}_\beta$$

(2) $$\psi_1/m \longrightarrow \tilde{\psi} \in \mathbb{R}_{\eta\delta} .$$

For $\tilde{Q} = Q(m \longleftarrow \psi_2)$ we have

(3) $$\varphi_1(n_2/n_1 \longleftarrow \psi_2) = \gamma(\lambda(x, \tilde{Q}), P) .$$

But $B_x Q = \emptyset$ by $R \in \mathbb{F}_0$, and so

(4) $$\psi_1(\{m\} \longleftarrow \tilde{\psi}) = ([P/x]Q)(m \longleftarrow [P/x]\psi_2) = [P/x]\tilde{Q} .$$

Now FVbl $P \cap$ (BVbl $\tilde{Q} \cup \{x\}) = \emptyset$ by BVbl $\tilde{Q} \subseteq$ BVbl Q, so (1) follows from (3), (4), and the definition of $\mathbb{R}_\beta$ (9.7.2).

By the definitions of $\mathbb{R}_\eta$ and $\mathbb{R}_\delta$ in (9.7), $[P/x]\varphi_2 \longrightarrow [P/x]\psi_2 \in \mathbb{R}_{\eta\delta}$. Since $B_x Q = \emptyset$ implies that

$$[P/x]\varphi_2 = [P/x](Q/m) = \psi_1/m$$

while $[P/x]\psi_2 = \tilde{\psi}$ by definition, this proves (2).

By Lemma 5.8, some $T \in \mathbb{F}$ has $S_1 \overset{*}{\underset{\eta\delta}{\Longrightarrow}} T$ and $S_2 \underset{\beta}{\Longrightarrow} T$. Let $\mathcal{T} \in \mathbb{E}$ with $T \in \mathcal{T}$.

<u>Case 2.3</u> ($n_1 \cdot (1)$ <u>anc</u> n_2). Similar to Case 2.2, but easier. Letting M be $r(n_2/n_1)$ in (9.10.3) we show that $\varphi_1 \longrightarrow \psi_1$ is pseudoclosed at $(n_2/n_1, M)$ with respect to $(\mathbb{R}_\beta, \mathbb{R}_{\eta\delta})$, using $\tilde{\psi} = \psi_2$ now. We then have $S_1 \overset{*}{\underset{\eta\delta}{\Longrightarrow}} T$ and $S_2 \underset{\beta}{\Longrightarrow} T$ for some $T \in \mathbb{F}$ by Lemma 5.8.

Case 3 (n_2 anc n_1). By the restriction (9.7.5) on $\mathbb{R}_\delta$, $\varphi_2 \longrightarrow \psi_2 \in \mathbb{R}_\eta$ and so some $x \in X$ has

$$\varphi_2 = \lambda(x, \gamma(\psi_2, x)) \ \& \ F_x\psi_2 = \emptyset .$$

Case 3.1 ($n_2 \cdot (1) = n_1$). There are $y \in X$ and $Q \in \mathbb{F}$ such that $\psi_2 = \lambda(y, Q)$ and $\psi_1 = [x/y]Q$. Therefore

$$S_1 = R(n_2 \longleftarrow \lambda(x, [x/y]Q)) \ \& \ S_2 = R(n_2 \longleftarrow \lambda(y, Q)) .$$

Since $Q^{-1}(x) = \emptyset$, it is easy to show that $\lambda(x, [x/y]Q) \cong \lambda(y, Q)$ and hence that $S_1 \cong S_2$. Let $\mathcal{T} \in \mathbb{E}$ with $S_1, S_2 \in \mathcal{T}$.

Case 3.2 ($n_2 \cdot (1,0)$ anc n_1). Letting $M = r(n_1/n_2)$ in (9.10.2), we check that $\varphi_2 \longrightarrow \psi_2$ is pseudoclosed at $(n_1/n_2, M)$ with respect to $(\mathbb{R}_\eta, \mathbb{R}_\beta)$, using $\tilde{\psi} = \psi_1$ in Definition 5.7. By Lemma 5.8, there is a $T \in \mathbb{F}$ such that

$$S_2 \overset{|M|}{\underset{\beta}{\Longrightarrow}} T \ \& \ S_1 \underset{\eta}{\Longrightarrow} T .$$

By $|M| = 1$ we have $S_1 \overset{*}{\underset{\eta\delta}{\Longrightarrow}} T$ and $S_2 \overset{=}{\underset{\beta}{\Longrightarrow}} T$. Let $\mathcal{T} \in \mathbb{E}$ with $T \in \mathcal{T}$. ∎

We cannot show that $\mathfrak{B}_\beta$ is Church-Rosser as simply as in Lemma 10.2 because $\mathfrak{C}_\beta$ is not closed. Instead we analyze $\mathfrak{B}_\beta$ as a combination of "requests" for substitutions and "performances" of substitutions. Rather than apply a rule from $\mathbb{R}_\beta$ to a tree in $\mathbb{F}$ in one step, we first apply a rule of the form $\gamma(\lambda(x, S), R) \longrightarrow \sigma(R, x, S)$ where σ is a new symbol. The set of such rules will be called $\mathbb{R}_\gamma$ and will be defined later. These requests for substitutions form trees in a larger forest $\mathbb{H}$ that includes $\mathbb{F}$ as well as trees of the form $\sigma(R, x, S)$ with $R, S \in \mathbb{H}$ and $x \in X$. Substitutions will be performed by a set $\mathbb{R}_\sigma$ of rules that move σ downward. A single step $R \underset{\beta}{\Longrightarrow} S$ will be simulated by a sequence of steps $R(\underset{\gamma}{\Longrightarrow} \overset{*}{\underset{\sigma}{\Longrightarrow}})S$ with the intermediate trees in $\mathbb{H} - \mathbb{F}$. Let σ be a new symbol (pronounced "substitute") and let

(10.4.1) $\quad W = V \cup \{\sigma\}$.

Assign rank 3 to σ and define $\mathbb{H} \subseteq W_{\#}$ inductively:

(10.4.2) $\quad (C \cup X)_{\#} \subseteq \mathbb{H}$

$(\forall x \in X)(\forall S \in \mathbb{H})(\lambda(x, S) \in \mathbb{H})$

$(\forall R, S \in \mathbb{H})(\gamma(R, S) \in \mathbb{H})$

$(\forall x \in X)(\forall R, S \in \mathbb{H})(\sigma(R, x, S) \in \mathbb{H})$.

We expect $\sigma(R, x, S)$ to have normal form $[R/x]S$ whenever $R, S \in \mathbb{F}$. Let $\mathbb{R}_\sigma$ be the set of rules defined by

(10.4.3) $\quad \sigma(R, x, \gamma(S, T)) \longrightarrow \gamma(\sigma(R, x, S), \sigma(R, x, T))$

$\sigma(R, x, \lambda(x, S)) \longrightarrow \lambda(x, S)$

$\sigma(R, x, \lambda(y, S)) \longrightarrow \lambda(y, \sigma(R, x, S))$

$\sigma(R, x, x) \longrightarrow R \qquad\qquad \sigma(R, x, a) \longrightarrow a$

whenever $R, S, T \in \mathbb{H}$; $x, y \in X$; $a \in C \cup X$ with $x \neq y$ and $x \neq a$. The SRS $\mathfrak{C}_\sigma = (W, \mathbb{H}, \underset{\sigma}{\Longrightarrow}, \mathbb{R}_\sigma)$ will perform substitutions. The actual letters R, S, T and a letter $a = a_x$ for each choice of x above may be considered as parameters with domains

(10.4.4) $\quad D_R = D_S = D_T = \mathbb{H}$ & $D_{a_x} = (C \cup X - \{x\})_{\#}$.

Then (10.4.3) defines an infinite set of rule-schemata as x and y vary.

The substitution $[R/x]$ of R for the free occurrences of x in S was defined by (9.2.6):

$$[R/x]S = S(F_x S \longleftarrow R) .$$

The following lemma shows that $\mathfrak{C}_\sigma$ can compute substitutions by finding normal forms for trees in $\mathbb{H}$. It also establishes useful inductive properties of these normal form computations.

(10.5) <u>Lemma.</u>

(1) $(\forall x \in X)(\forall R, S \in \mathbb{F})(\sigma(R, x, S) \overset{*}{\underset{\sigma}{\Longrightarrow}} [R/x]S)$

(2) Each $P \in \mathbb{H}$ has a unique normal form $\Sigma P \in \mathbb{F}$ under $\mathfrak{C}_\sigma$.

(3) $(\forall x \in X)(\forall R, S \in \mathbb{H})(\Sigma\lambda(x, S) = \lambda(x, \Sigma S)$ &

$\Sigma\gamma(R, S) = \gamma(\Sigma R, \Sigma S)$ & $\Sigma\sigma(R, x, S) = [\Sigma R/x]\Sigma S)$.

<u>Proof:</u> Comparing $[R/x]S$ from (9.2.6) with the rules from (10.4.3) leads to a proof of (1) by induction on the size of S. By the inductive definition of $\mathbb{H}$ in (10.4.2), we can show that each $P \in \mathbb{H}$ has at least one normal form in $\mathbb{F}$, where (1) is used in the only nontrivial case $[P(\ ) = \sigma]$ in the induction. To show that each $P \in \mathbb{H}$ has at most one normal form, we will show that $\mathfrak{C}_\sigma$ is Church-Rosser. The rules in (10.4.3) form a partial function on trees, so $\mathfrak{C}_\sigma$ is unequivocal. By the Rule-Schemata Theorem (6.5), $\mathfrak{C}_\sigma$ is closed. Therefore $\mathfrak{C}_\sigma$ is Church-Rosser by the Main Theorem (5.6). This proves (2), and (3) follows from (2) and the definition of $\mathbb{R}_\sigma$ in (10.4.3). ∎

Let $\mathbb{R}_\gamma$ be the set of rules defined by

(10.6.1) $\gamma(\lambda(x, S), R) \longrightarrow \sigma(R, x, S)$

whenever $x \in X$; $R, S \in \mathbb{H}$. The SRS $\mathfrak{C}_\gamma = (W, \mathbb{H}, \underset{\gamma}{\Longrightarrow}, \mathbb{R}_\gamma)$ will request substitutions. The actual letters R, S above may be considered as parameters with domains

(10.6.2) $\quad D_R = D_S = \mathbb{H}$.

In order to relate $\mathfrak{C}_\sigma$ and $\mathfrak{C}_\gamma$ to $\mathfrak{B}_\beta$ we set

(10.6.3) $\quad (\underset{1}{\Longrightarrow}) = (\overset{*}{\underset{\gamma}{\Longrightarrow}} \overset{*}{\underset{\sigma}{\Longrightarrow}} \cong) \cap \mathbb{F}_0 \times \mathbb{F}_0$.

We will show that $(\mathbb{F}_0, \underset{1}{\Longrightarrow})$ is a Church-Rosser GRS and that this implies the Church-Rosser property for $\mathfrak{B}_\beta$. The first two steps are easy lemmas.

(10.7) <u>Lemma</u>. Let $S \in \mathbb{F}$; $\overline{S} \in \mathbb{H}$. Suppose $S \overset{*}{\underset{\gamma}{\Longrightarrow}} \overline{S}$. Then $\text{BVbl}\, \Sigma\overline{S} \subseteq \text{BVbl}\, S$ and $\text{FVbl}\, \Sigma\overline{S} \subseteq \text{FVbl}\, S$.

<u>Proof:</u> We prove both assertions simultaneously by induction on the size of S. The only nontrivial case is when $S(\ ) = \gamma$ and $\overline{S}(\ ) = \sigma$. Suppose this happens. Then some $x \in X$ and $P, Q \in \mathbb{F}$ have $S = \gamma(\lambda(x, Q), P)$ while some $\overline{P}, \overline{Q} \in \mathbb{H}$ have $\overline{S} = \sigma(\overline{P}, x, \overline{Q})$ and $P \overset{*}{\underset{\gamma}{\Longrightarrow}} \overline{P}$ and $Q \overset{*}{\underset{\gamma}{\Longrightarrow}} \overline{Q}$. By the induction hypothesis and (3) in Lemma 10.5,

$$\begin{aligned} \text{BVbl}\, \Sigma\overline{S} = \text{BVbl}\, [\Sigma\overline{P}/x]\Sigma\overline{Q} &\subseteq \text{BVbl}\, \Sigma\overline{P} \cup \text{BVbl}\, \Sigma\overline{Q} \\ &\subseteq \text{BVbl}\, P \cup \text{BVbl}\, Q \subseteq \text{BVbl}\, S . \end{aligned}$$

Also by the induction hypothesis,

$$\begin{aligned} \text{FVbl}\, \Sigma\overline{S} &\subseteq \text{FVbl}\, \Sigma\overline{P} \cup (\text{FVbl}\, \Sigma\overline{Q} - \{x\}) \\ &\subseteq \text{FVbl}\, P \cup (\text{FVbl}\, Q - \{x\}) = \text{FVbl}\, S . \blacksquare \end{aligned}$$

(10.8) <u>Lemma</u>. $\quad (\underset{1}{\Longrightarrow}) \subseteq (\overset{*}{\underset{\beta}{\Longrightarrow}} \cong)$.

<u>Proof:</u> It suffices to show that each $R \in \mathbb{F}_0$ has the property

(1) $\quad (\forall S \in \mathbb{H})(R \overset{*}{\underset{\gamma}{\Longrightarrow}} S \text{ implies } R \overset{*}{\underset{\beta}{\Longrightarrow}} \Sigma S)$,

since $(\overset{*}{\underset{\sigma}{\Longrightarrow}} \cong) = (\Sigma \cong)$ when the map $\Sigma : \mathbb{H} \longrightarrow \mathbb{F}$ is viewed as a relation.

Suppose that (1) holds for all trees smaller than R and that $R \overset{*}{\underset{\gamma}{\Rightarrow}} S$. The proof that $R \overset{*}{\underset{\beta}{\Rightarrow}} \Sigma S$ is straightforward, using (3) in Lemma 10.5, except in the case where R( ) = γ and S( ) = σ. Suppose this happens, so that R has the form $\gamma(\lambda(x, Q), P)$ with FVbl $P \cap (\text{BVbl } Q \cup \{x\}) = \emptyset$. There are $\overline{P}, \overline{Q} \in \mathbb{H}$ such that $S = \sigma(\overline{P}, x, \overline{Q})$ and $P \overset{*}{\underset{\gamma}{\Rightarrow}} \overline{P}$ and $Q \overset{*}{\underset{\gamma}{\Rightarrow}} \overline{Q}$. By the induction hypothesis, $P \overset{*}{\underset{\beta}{\Rightarrow}} \Sigma\overline{P}$ and $Q \overset{*}{\underset{\beta}{\Rightarrow}} \Sigma\overline{Q}$. By (3) in Lemma 10.5 and FVbl $\Sigma\overline{P} \cap (\text{BVbl } \Sigma\overline{Q} \cup \{x\}) = \emptyset$ from Lemma 10.7,

$$R = \gamma(\lambda(x, Q), P) \overset{*}{\underset{\beta}{\Rightarrow}} \gamma(\lambda(x, \Sigma\overline{Q}), \Sigma\overline{P}) \underset{\beta}{\Rightarrow} [\Sigma\overline{P}/x]\Sigma\overline{Q} = \Sigma S. \blacksquare$$

Suppose $R, S \in \mathbb{F}$; $\overline{R}, \overline{S} \in \mathbb{H}$; $R \overset{*}{\underset{\gamma}{\Rightarrow}} \overline{R}$; $S \overset{*}{\underset{\gamma}{\Rightarrow}} \overline{S}$. It is reasonable to expect that $[R/x]S \overset{*}{\underset{\gamma}{\Rightarrow}} \overset{*}{\underset{\sigma}{\Rightarrow}} [\Sigma\overline{R}/x]\Sigma\overline{S}$, at least when appropriate restrictions on variables being both bound and free are imposed. The next lemma carries out this idea and allows a different choice of R at each node in $F_x S$.

(10.9) <u>Lemma</u>. Let $S \in \mathbb{F}$; $\overline{S} \in \mathbb{H}$. Let $(m_0, \ldots, m_{K-1})$ be a listing of $F_x S$ for some $x \in X$ and $K \in \mathbb{N}$. Let $R_k \in \mathbb{F}$ and $\overline{R}_k \in \mathbb{H}$ for each $k < K$. Suppose

(1) $$B_x S = \emptyset \ \& \ (\forall k < K)(\text{FVbl } R_k \cap \text{BVbl } S = \emptyset)$$

(2) $$S \overset{*}{\underset{\gamma}{\Rightarrow}} \overline{S} \ \& \ (\forall k < K)(R_k \overset{*}{\underset{\gamma}{\Rightarrow}} \overline{R}_k).$$

Then there are mutually independent sets $N_0, \ldots, N_{K-1}$ of independent nodes in $\Sigma\overline{S}$ such that

(3) $$F_x \Sigma\overline{S} = \bigcup_{k<K} N_k$$

(4) $$S(m_0 \longleftarrow R_0)\ldots(m_{-1} \longleftarrow R_{-1}) (\overset{*}{\underset{\gamma}{\Rightarrow}} \overset{*}{\underset{\sigma}{\Rightarrow}}) \Sigma\overline{S}(N_0 \longleftarrow \Sigma\overline{R}_0)\ldots(N_{-1} \longleftarrow \Sigma\overline{R}_{-1}).$$

<u>Proof</u>: For some $J \in \mathbb{N}$ there are trees $S_0, \ldots, S_J$ with

$$S = S_0 \underset{\gamma}{\Rightarrow} S_1 \ldots \underset{\gamma}{\Rightarrow} S_J = \overline{S}.$$

For each $j < J$, let a rule $\varphi_j \longrightarrow \psi_j$ be applied at a node n_j to derive S_{j+1} from S_j. Each $\varphi_j \longrightarrow \psi_j$ is an instance of a rule-schema in the definitions (10.6.1) and (10.6.2). Let r_j be the residue map assigned to $\varphi_j \longrightarrow \psi_j$ by Lemma 6.3. For each $k < K$ define $M_k^0, \ldots, M_k^J$ by $M_k^0 = \{m_k\}$ and

$$M_k^{j+1} = \{m \in M_k^j \mid m \perp n_j\} \cup$$
$$\{n_j \cdot p \mid (\exists m \in M_k^j)(n_j \; \underline{\text{anc}} \; m \;\; \& \;\; p \in r_j(m/n_j))\}.$$

By induction on j, we can show that $S_j^{-1}(x) = \bigcup_{k<K} M_k^j$, and no $m \in M_k^j$ has the form $q \cdot (0)$ with $S_j q = \lambda$ or $q \cdot (1)$ with $S_j q = \sigma$. By Lemma 6.4 we can then show that each $j < J$ has

(5) $$S_j(M_0^j \longleftarrow R_0) \ldots (M_{-1}^j \longleftarrow R_{-1}) \underset{\gamma}{\Rightarrow}$$
$$S_{j+1}(M_0^{j+1} \longleftarrow R_0) \ldots (M_{-1}^{j+1} \longleftarrow R_{-1}).$$

In (2) we have

(6) $$S(m_0 \longleftarrow R_0) \ldots (m_{-1} \longleftarrow R_{-1}) (\overset{*}{\underset{\gamma}{\Rightarrow}} \overset{*}{\underset{\sigma}{\Rightarrow}}) \overline{S}(M_0^J \longleftarrow \Sigma\overline{R}_0) \ldots (M_{-1}^J \longleftarrow \Sigma\overline{R}_{-1}).$$

For some $\overline{J} \in \mathbb{N}$ there are trees $\overline{S}_0, \ldots, \overline{S}_{\overline{J}}$ with

$$\overline{S} = \overline{S}_0 \underset{\sigma}{\Rightarrow} \overline{S}_1 \ldots \underset{\sigma}{\Rightarrow} \overline{S}_{\overline{J}} = \Sigma\overline{S}.$$

For each $j < \overline{J}$, let a rule $\overline{\varphi}_j \longrightarrow \overline{\psi}_j \in \mathbb{R}_\sigma$ be applied at a node $\overline{n}_j$ to derive $\overline{S}_{j+1}$ from $\overline{S}_j$. Each $\overline{\varphi}_j \longrightarrow \overline{\psi}_j$ is an instance of a rule-schema in the definitions (10.4.3) and (10.4.4). Let $\overline{r}_j$ be the residue map assigned to $\overline{\varphi}_j \longrightarrow \overline{\psi}_j$ by Lemma 6.3. For each $k < K$ define

$\bar{M}_k^0, \ldots, \bar{M}_k^{\bar{J}}$ by $\bar{M}_k^0 = M_k^J$ and

$$\bar{M}_k^{j+1} = \{m \in \bar{M}_k^j \mid m \perp \bar{n}_j\} \cup$$
$$\{\bar{n}_j \cdot p \mid (\exists m \in \bar{M}_k^j)(\bar{n}_j \ \underline{\text{anc}}\ m \ \& \ p \in \bar{r}_j(m/\bar{n}_j))\} .$$

By induction on j, we show that $\bar{S}_j^{-1}(x) = \bigcup_{k<K} \bar{M}_k^j$ and no $m \in \bar{M}_k^j$ has the form $q\cdot(0)$ with $\bar{S}_j q = \lambda$ or $q\cdot(1)$ with $\bar{S}_j q = \sigma$. Setting

(7) $$(\forall k < K)(N_k = \bar{M}_k^{\bar{J}})$$

yields (3). Now we will show that each $j<J$ has

(8) $$\bar{S}_j(\bar{M}_0^j \longleftarrow \Sigma\bar{R}_0)\ldots(\bar{M}_{-1}^j \longleftarrow \Sigma\bar{R}_{-1}) \overset{*}{\underset{\sigma}{\Longrightarrow}}$$
$$\bar{S}_{j+1}(\bar{M}_0^{j+1} \longleftarrow \Sigma\bar{R}_0)\ldots(\bar{M}_{-1}^{j+1} \longleftarrow \Sigma\bar{R}_{-1}) .$$

In all but one case we can prove (8) by means of Lemma 6.4, as in (5). The odd case occurs when $\bar{\varphi}_j \longrightarrow \bar{\psi}_j$ is of the form $\sigma(P, y, x) \longrightarrow x$ for $y \in X$ with $y \neq x$. Assume this happens. There is no rule

$$\sigma(P, y, \Sigma\bar{R}_k) \longrightarrow \Sigma\bar{R}_k$$

in $\mathbb{R}_\sigma$, but several steps can achieve the same effect. We claim that

$$\sigma(P, y, \Sigma\bar{R}_k) \overset{*}{\underset{\sigma}{\Longrightarrow}} \Sigma\bar{R}_k .$$

By $S \in \mathbb{F}$ we have $y \in \mathrm{BVbl}\, S$, so $F_y \Sigma\bar{R}_k = \emptyset$ by (1). Therefore $[\Sigma P/y]\Sigma\bar{R}_k = \Sigma\bar{R}_k$, and we do have $\sigma(P, y, \Sigma\bar{R}_k) \Longrightarrow [\Sigma P/y]\Sigma\bar{R}_k$ by (3) in Lemma 10.5. This completes the proof of (8). Combining (8) with (6) and (7) yields (4).∎

The next lemma will show that the assertion diagrammed by Figure 4-1 is correct. The reason for proving this complicated statement is simply that we wish to use it as a stencil later, in verifying a diagram by the technique introduced in the proof of the Commutativity Lemma (3.6). We discuss the content of Figure 4-1 before proceeding with the proof.

Suppose we have $R \in \mathbb{F}_0$ and we apply some gamma rules to derive a tree $R' \in \mathbb{H}$. We can now apply more gamma rules to form $R'' \in \mathbb{H}$, but we can also perform the substitutions presently requested in R' instead. By applying sigma rules to R' we can derive the normal form $\Sigma R'$, which is actually in $\mathbb{F}$ and is therefore strongly alphaequivalent to some $S \in \mathbb{F}_0$. Can we apply gamma rules to S so as to parallel the derivation of R'' from R'? Can we obtain $S \overset{*}{\underset{\gamma}{\Longrightarrow}} S'$ where S' is essentially the same as R'', at least after we perform the substitutions and allow for changes of bound variables? Figure 4-1 asserts that this is always possible.

(10.10) <u>Lemma</u>. Let $R, S \in \mathbb{F}_0$; $R', R'' \in \mathbb{H}$. Suppose $R \overset{*}{\underset{\gamma}{\Longrightarrow}} R' \overset{*}{\underset{\gamma}{\Longrightarrow}} R''$ and $\Sigma R' \cong S$. Then $S(\overset{*}{\underset{\gamma}{\Longrightarrow}} \overset{*}{\underset{\sigma}{\Longrightarrow}} \cong)\, \Sigma R''$.

<u>Proof</u>: Consider three statements involving indeterminate trees R, R', R'', S, S':

(1) $R \overset{*}{\underset{\gamma}{\Longrightarrow}} R' \overset{*}{\underset{\gamma}{\Longrightarrow}} R''$ & $\Sigma R' \simeq S$

(2) $S \overset{*}{\underset{\gamma}{\Longrightarrow}} S'$ & $\Sigma R'' \simeq \Sigma S'$

(3) $(\forall y, z \in X)(F_y \Sigma R'' = F_z \Sigma S' \neq \emptyset$ implies $F_y \Sigma R' = F_z S)$.

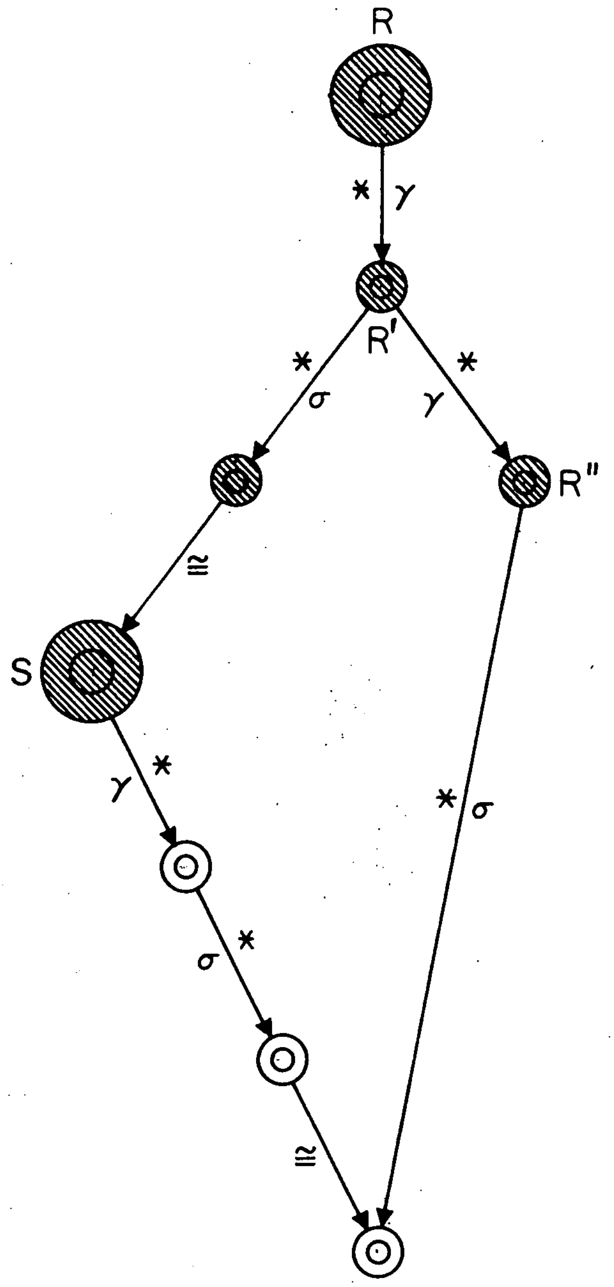


Figure 4-1. Stencil for use in the proof that ($\mathbb{F}_0, \xRightarrow[1]{}$) is Church-Rosser. Small circles represent members of $\mathbb{H}$; large circles represent members of $\mathbb{F}_0$.

It will suffice to prove that each $R \in \mathbb{F}_0$ satisfies

(4) $(\forall S \in \mathbb{F}_0)(\forall R', R'' \in \mathbb{H})((1) \text{ implies } (\exists S' \in \mathbb{H})[(2) \mathbin{\&} (3)])$.

Suppose (4) holds for all trees smaller than R in $\mathbb{F}_0$ and suppose R, S, R', R'' satisfy (1). We will show there is an $S' \in \mathbb{H}$ such that (2) and (3) hold. There are three basic cases: $R(\,) \in C \cup X$ or $R(\,) = \lambda$ or $R(\,) = \gamma$. The case where $R(\,) = \gamma$ has two subcases: $R''(\,) = \gamma$ or $R''(\,) = \sigma$. When $R''(\,) = \sigma$ there are also two subcases to consider: $R'(\,) = \gamma$ or $R'(\,) = \sigma$. Except for the trivial case $[R(\,) \in C \cup X]$, all the cases are treated by applying the induction hypothesis to various subtrees of R. Various trees play the role of S' in (4) for each of these subtrees of R, and we must combine them into a tree S' that satisfies (2) and (3). The leaves $[R(\,) = \gamma;\ R''(\,) = \sigma;\ R'(\,) = \gamma]$ and $[R(\,) = \gamma;\ R''(\,) = \sigma;\ R'(\,) = \sigma]$ in the hierarchy of cases require some extremely tedious verifications, but the strategy is exactly the same for them as for $[R(\,) = \lambda]$. The leaves in the hierarchy of cases are doubly underlined in the detailed argument below.

Case 1 $(R(\,) \in C \cup X)$. Then some $a \in C \cup X$ has

$$R = R' = R'' = \Sigma R' = \Sigma R'' = a$$

and some $b \in C \cup X$ has $S = b$. (If $a \in C$ then $b = a$. If $a \in X$ then $b \in X$ but perhaps $b \neq a$, since only weak alphaequivalence is assumed in (1).) Let $S' = S$, so that $\Sigma S' = b$ and (2) and (3) are trivial.

Case 2 $(R(\,) = \lambda)$. There are $x, \bar{x} \in X$ and $Q, \bar{Q} \in \mathbb{F}_0$ and $Q', Q'' \in \mathbb{H}$ with

$$R = \lambda(x, Q) \mathbin{\&} R' = \lambda(x, Q') \mathbin{\&} R'' = \lambda(x, Q'') \mathbin{\&} S = \lambda(\bar{x}, \bar{Q})\,.$$

By the definition of $\mathbb{R}_\gamma$ in (10.6) and the inductive lemma (9.5.1) for $\simeq$,

$$Q \overset{*}{\underset{\gamma}{\Longrightarrow}} Q' \overset{*}{\underset{\gamma}{\Longrightarrow}} Q'' \ \& \ \Sigma Q' \simeq \bar{Q} \ \& \ F_x \Sigma Q' = F_{\bar{x}} \bar{Q} .$$

By the induction hypothesis, some $Q' \in \mathbb{H}$ has

$$\bar{Q} \overset{*}{\underset{\gamma}{\Longrightarrow}} \bar{Q}' \ \& \ \Sigma Q'' \simeq \Sigma \bar{Q}' \ \&$$
$$(\forall y, z \in X)(F_y \Sigma Q'' = F_z \Sigma \bar{Q}' \neq \emptyset \text{ implies } F_y \Sigma Q' = F_z \bar{Q}) .$$

Setting $S' = \lambda(\bar{x}, \bar{Q}')$, we find that (2) is trivial and (3) follows from direct calculations based on equations like

$$F_y \lambda(x, Q) = \underset{\sim}{\text{if}}\ y = x\ \underset{\sim}{\text{then}}\ \emptyset\ \underset{\sim}{\text{else}}\ (1) \cdot F_y Q .$$

<u>Case 3</u> $(R() = \gamma)$.

<u>Case 3.1</u> $(R''() = \gamma)$. Similar to Case 2.

<u>Case 3.2</u> $(R''() = \sigma)$. There are $x \in X$ and $P, Q \in \mathbb{F}_0$ and $P'', Q'' \in \mathbb{H}$ with

$$R = \gamma(\lambda(x, Q), P) \ \& \ R'' = \sigma(P'', x, Q'') .$$

<u>Case 3.2.1</u> $(R'() = \gamma)$. There are $\bar{x} \in X$ and $\bar{P}, \bar{Q} \in \mathbb{F}_0$ and $P', Q' \in \mathbb{H}$ with

$$R' = \gamma(\lambda(x, Q'), P') \ \& \ S = \gamma(\lambda(\bar{x}, \bar{Q}), \bar{P}) .$$

By the definition of $\mathbb{R}_\gamma$ in (10.6) and the inductive lemma (9.5) for $\simeq$,

(5) $\quad P \overset{*}{\underset{\gamma}{\Longrightarrow}} P' \overset{*}{\underset{\gamma}{\Longrightarrow}} P'' \ \& \ \Sigma P' \simeq \bar{P}$

(6) $\quad Q \overset{*}{\underset{\gamma}{\Longrightarrow}} Q' \overset{*}{\underset{\gamma}{\Longrightarrow}} Q'' \ \& \ \Sigma Q' \simeq \bar{Q}$

$$(\forall y, z \in X)(F_y \lambda(x, \Sigma Q') = F_z \lambda(\bar{x}, \bar{Q}) \neq \emptyset \text{ implies } F_y \Sigma P' = F_z \bar{P}).$$

We will show that this last fact and (6) imply

(7) $(\forall y, z \in X)(F_y \Sigma P' = F_z \overline{P} \neq \emptyset$ implies $F_y \Sigma Q' = F_z \overline{Q})$.

Suppose $F_y \Sigma P' = F_z \overline{P} \neq \emptyset$, so that $y \neq x$ and $z \neq \bar{x}$. Consider two cases.

<u>Case a</u> $(F_y \Sigma Q' \neq \emptyset)$ By (6), some $w \in X$ has $F_y \Sigma Q' = F_w \overline{Q} \neq \emptyset$. We have $w \neq \bar{x}$ because $y \neq x$, so $Fy\lambda(x, \Sigma Q') = F_w \lambda(\bar{x}, \overline{Q}) \neq \emptyset$. Therefore $F_y \Sigma P' = F_w \overline{P}$. But $F_y \Sigma P' = F_z \overline{P} \neq \emptyset$, so $z = w$. The equation $F_y \Sigma Q' = F_w \overline{Q}$ becomes $F_y \Sigma Q' = F_z \overline{Q}$, as desired.

<u>Case b</u> $(F_y \Sigma Q' = \emptyset)$ Suppose $F_z \overline{Q} \neq \emptyset$. By (6), some $w \in X$ has $F_w \Sigma Q' = F_z \overline{Q} \neq \emptyset$. We have $w \neq x$ because $z \neq \bar{x}$, so $F_w \lambda(x, \Sigma Q') = F_z \lambda(\bar{x}, \overline{Q}) \neq \emptyset$. Therefore $F_w \Sigma P' = F_z \overline{P}$. But $F_y \Sigma P' = F_z \overline{P} \neq \emptyset$, so $y = w$. The equation $F_w \Sigma Q' = F_z \overline{Q} \neq \emptyset$ becomes $F_y \Sigma Q' = F_z \overline{Q} \neq \emptyset$. But this contradicts $F_y \Sigma Q' = \emptyset$, so the supposition is false and $F_z \overline{Q} = \emptyset = F_y \Sigma Q'$. This proves (7).

By (5) and the induction hypothesis, some $\overline{P}' \in \mathbb{H}$ has

(8) $\overline{P} \overset{*}{\underset{\gamma}{\Longrightarrow}} \overline{P}'$ & $\Sigma P'' \simeq \Sigma \overline{P}'$

(9) $(\forall y, z \in X)(F_y \Sigma P'' = F_z \Sigma \overline{P}' \neq \emptyset$ implies $F_y \Sigma P' = F_z \overline{P})$.

By (6) and the induction hypothesis, some $\overline{Q}' \in \mathbb{H}$ has

(10) $\overline{Q} \overset{*}{\underset{\gamma}{\Longrightarrow}} \overline{Q}'$ & $\Sigma Q'' \simeq \Sigma \overline{Q}'$

(11) $(\forall y, z \in X)(F_y \Sigma Q'' = F_z \Sigma \overline{Q}' \neq \emptyset$ implies $F_y \Sigma Q' = F_z \overline{Q})$.

Let S' be $\sigma(\overline{P}', \bar{x}, \overline{Q}')$, so that (8) and (10) imply $S \overset{*}{\underset{\gamma}{\Longrightarrow}} S'$. To complete the proof of (2), we will use Lemma 9.6 to show that

$$\Sigma R'' = [\Sigma P''/x]\Sigma Q'' \simeq [\Sigma\overline{P}'/\bar{x}]\Sigma\overline{Q}' = \Sigma S'.$$

We must verify (9.6.1)--(9.6.3) and (9.6.5)--(9.6.7). For (9.6.1) we must show that $F_x\Sigma Q'' = F_{\bar{x}}\Sigma\overline{Q}'$. There are two cases to consider.

<u>Case a</u> $(F_x\Sigma Q'' \neq \emptyset)$ By (10), some $w \in X$ has

(12) $$F_x\Sigma Q'' = F_w\Sigma\overline{Q}' \neq \emptyset.$$

By (11) and Lemma 10.7, this implies that

$$F_x\Sigma Q' = F_w\overline{Q} \neq \emptyset.$$

But $F_x\Sigma Q' = F_{\bar{x}}\overline{Q}$ by the inductive lemma (9.5) for $\simeq$ and $\Sigma R' \simeq S$, so this yields $w = \bar{x}$. In (12) we have $F_x\Sigma Q'' = F_{\bar{x}}\Sigma\overline{Q}'$.

<u>Case b</u> $(F_x\Sigma Q'' = \emptyset)$ Suppose $F_{\bar{x}}\Sigma\overline{Q}' \neq \emptyset$. Reasoning as in Case a, we find that some $w \in X$ has $F_w\Sigma Q'' = F_{\bar{x}}\Sigma\overline{Q}' \neq \emptyset$ and then that $w = x$. Therefore $F_x\Sigma Q'' \neq \emptyset$, a contradiction. We must actually have $F_{\bar{x}}\Sigma\overline{Q}' = \emptyset = F_x\Sigma Q''$.

The only other condition not trivial to verify is (9.6.6):

$$(\forall y, z \in X)(F_y\Sigma Q'' = F_z\Sigma\overline{Q}' \neq \emptyset \text{ implies } F_y\Sigma P'' = F_z\Sigma\overline{P}').$$

Suppose $F_y\Sigma Q'' = F_z\Sigma\overline{Q}' \neq \emptyset$, so that (11) and Lemma 10.7 imply

(13) $$F_y\Sigma Q' = F_z\overline{Q} \neq \emptyset.$$

<u>Case a</u> $(F_y\Sigma P'' \neq \emptyset)$ By (8), some $w \in X$ has

(14) $$F_y\Sigma P'' = F_w\Sigma\overline{P}' \neq \emptyset.$$

To this we apply (9), then Lemma 10.7, and then (7), so that

$$F_y\Sigma Q' = F_w\overline{Q}.$$

Comparing with (13), we find that $w = z$ and so (14) implies that $F_y\Sigma P'' = F_z\Sigma\overline{P}'$, as desired.

Case b ($F_y \Sigma P'' = \emptyset$). Suppose that $F_z \Sigma \bar{P}' \neq \emptyset$. Reasoning as in Case a, we find that some $w \in X$ has $F_w \Sigma P'' = F_z \Sigma \bar{P}' \neq \emptyset$ and then that $w = y$. Therefore $F_y \Sigma P'' \neq \emptyset$, a contradiction. We must actually have $F_z \Sigma \bar{P}' = \emptyset = F_y \Sigma P''$. This completes the proof of (2).

To show that our $S' = \sigma(\bar{P}', x, \bar{Q}')$ satisfies (3), we suppose that $F_y \Sigma R'' = F_z \Sigma S' \neq \emptyset$ and show that $F_y \Sigma R' = F_z S$. By Lemma 10.7 and $R, S \in \mathbb{F}_0$

$$F_y \Sigma R'' = F_x \Sigma Q'' \cdot F_y \Sigma P'' \cup (F_y \Sigma Q'' — F_x \Sigma Q'')$$

$$F_z \Sigma S' = F_{\bar{x}} \Sigma \bar{Q}' \cdot F_z \Sigma \bar{P}' \cup (F_z \Sigma \bar{Q}' — F_{\bar{x}} \Sigma \bar{Q}'),$$

so that our supposition implies

(15) $$F_x \Sigma Q'' \cdot F_y \Sigma P'' = F_{\bar{x}} \Sigma \bar{Q}' \cdot F_z \Sigma \bar{P}'$$

(16) $$F_y \Sigma Q'' — F_x \Sigma Q'' = F_z \Sigma \bar{Q}' — F_{\bar{x}} \Sigma \bar{Q}',$$

where both equations cannot be $\emptyset = \emptyset$.

Case a ($F_x \Sigma Q'' \cdot F_y \Sigma P'' = \emptyset$) Then (16) cannot be $\emptyset = \emptyset$, so $y \neq x$ and $z \neq \bar{x}$. By (16), (11), and Lemma 10.7:

$$F_y \Sigma Q' = F_z \bar{Q} \neq \emptyset \ \& \ y \neq x \ \& \ z \neq \bar{x}.$$

By $\Sigma R' \simeq S$, this implies that $F_y \Sigma R' = F_z S$.

Case b ($F_x \Sigma Q'' \cdot F_y \Sigma P'' \neq \emptyset$) Then (15) implies that

$$F_y \Sigma P'' = F_z \Sigma \bar{P}' \neq \emptyset.$$

By (9) and Lemma 10.7 we have $F_y \Sigma P' = F_z \bar{P} \neq \emptyset$. By $\Sigma R' \simeq S$, this implies that $F_y \Sigma R' = F_z S$. The proof of (3) is complete.

Case 3.2.2 $(R'(\ ) = \sigma)$ There are $P', Q' \in \mathbb{H}$ such that

(17) $R' = \sigma(P', x, Q')$ & $\Sigma R' = [\Sigma P'/x]\Sigma Q'$

(18) $P \overset{*}{\underset{\gamma}{\Longrightarrow}} P' \overset{*}{\underset{\gamma}{\Longrightarrow}} P''$ & $Q \overset{*}{\underset{\gamma}{\Longrightarrow}} Q' \overset{*}{\underset{\gamma}{\Longrightarrow}} Q''$.

By Frame $\Sigma R'$ = Frame S we have $F_x\Sigma Q' \subseteq$ Dom S in (17). Set

$\overline{Q} = S(F_x\Sigma Q' \longleftarrow \overline{x})$ where $\overline{x}$ is any new variable.

Let $(m_0, \ldots, m_{K-1})$ for $K = |F_x\Sigma Q'|$ be any listing of $F_x\Sigma Q'$ and let $\overline{P}_k$ be S/m_k for each $k < K$, so that (17) and the definition of $\overline{Q}$ imply

(19) $\Sigma R' = \Sigma Q'(m_0 \longleftarrow \Sigma P') \ldots (m_{-1} \longleftarrow \Sigma P')$
$S = \overline{Q}(m_0 \longleftarrow \overline{P}_0) \ldots (m_{-1} \longleftarrow \overline{P}_{-1})$.

By Lemma 9.6, $\Sigma R' \simeq S$ implies that

(20) $\Sigma Q' \simeq \overline{Q}$ & $(\forall k < K)(\Sigma P' \simeq \overline{P}_k)$

(21) $(\forall y, z \in X)[F_y\Sigma Q' = F_z\overline{Q} \neq \emptyset$ implies $(\forall k < K)(F_y\Sigma P' = F_z P_k)]$

(22) $(\forall y, z \in X)(\forall j, k < K)(F_y\Sigma P' = F_z\overline{P}_j \neq \emptyset$ implies $F_y\Sigma P' = F_z\overline{P}_k)$.

From (20) and (21) it follows that

(23) $(\forall y, z \in X)(\forall k < K)(F_y\Sigma P' = F_z\overline{P}_k \neq \emptyset$ implies $F_y\Sigma Q' = F_z\overline{Q})$.

Applying the induction hypothesis to Q in (18) and (20), we find that some $\overline{Q}' \in \mathbb{H}$ has

(24) $\overline{Q} \overset{*}{\underset{\gamma}{\Longrightarrow}} \overline{Q}'$ & $\Sigma Q'' \simeq \Sigma\overline{Q}'$

(25) $(\forall y, z \in X)(F_y\Sigma Q'' = F_z\Sigma\overline{Q}' \neq \emptyset$ implies $F_y\Sigma Q' = F_z\overline{Q})$.

Applying the induction hypothesis to P in (18) and (20), we find that some $(\overline{P}'_0, \ldots, \overline{P}'_{-1}) \in \mathbb{H}^K$ has

(26) $\overline{P}_k \overset{*}{\underset{\gamma}{\Longrightarrow}} \overline{P}'_k \;\&\; \Sigma P'' \simeq \Sigma\overline{P}'_k$

(27) $(\forall y, z \in X)(F_y\Sigma P'' = F_z\Sigma\overline{P}'_k \neq \emptyset$ implies $F_y\Sigma P' = F_z\overline{P}_k)$

for all $k < K$.

The listing $(m_0, \ldots, m_{K-1})$ of $F_x\Sigma Q'$ is also a listing of $F_{\bar{x}}\overline{Q}$. By (24) and (26), Lemma 10.9 is applicable with $\overline{Q}$ in the role of S and $\overline{P}_k$ in the role of R_k in Lemma 10.9. Therefore there are mutually independent sets $N_0, \ldots, N_{K-1}$ of independent nodes in $\Sigma\overline{Q}'$ such that

(28) $F_{\bar{x}}\Sigma\overline{Q}' = \bigcup_{k<K} N_k$

and

$$\overline{Q}(m_0 \longleftarrow \overline{P}_0)\ldots(m_{-1} \longleftarrow \overline{P}_{-1}) \overset{*}{\underset{\gamma}{\Longrightarrow}} \overset{*}{\underset{\sigma}{\Longrightarrow}}$$
$$\Sigma\overline{Q}'(N_0 \longleftarrow \Sigma\overline{P}'_0)\ldots(N_{-1} \longleftarrow \Sigma\overline{P}'_{-1}).$$

Comparing this last fact with (19), we find that some $S' \in \mathbb{H}$ has

(29) $S \overset{*}{\underset{\gamma}{\Longrightarrow}} S' \;\&\; \Sigma S' = \Sigma\overline{Q}'(N_0 \longleftarrow \Sigma\overline{P}'_0)\ldots(N_{-1} \longleftarrow \Sigma\overline{P}'_{-1})$.

We must show that $\Sigma R'' \simeq \Sigma S'$ to complete the proof of (2). First we will show that $F_x\Sigma Q'' = F_{\bar{x}}\Sigma\overline{Q}'$.

Case a ($F_x\Sigma Q'' \neq \emptyset$) By (24), some $w \in X$ has

(30) $F_x\Sigma Q'' = F_w\Sigma\overline{Q}' \neq \emptyset$.

By (25) and Lemma 10.7, this implies that

$$F_x\Sigma Q' = F_w\overline{Q} \neq \emptyset.$$

But $F_x\Sigma Q' = F_{\bar{x}}\overline{Q}$ by definition of $\overline{Q}$, so this yields $w = \bar{x}$. In (30) we have $F_x\Sigma Q'' = F_{\bar{x}}\Sigma\overline{Q}'$.

Case b ($F_x\Sigma Q'' = \emptyset$) Suppose $F_{\bar{x}}\Sigma\bar{Q}' \neq \emptyset$. Reasoning as in Case a, we find that some $w \in X$ has $F_w\Sigma Q'' = F_{\bar{x}}\Sigma\bar{Q}' \neq \emptyset$ and then that $w = x$. Therefore $F_x\Sigma Q'' \neq \emptyset$, a contradiction. We must actually have $F_{\bar{x}}\Sigma\bar{Q}' = \emptyset = F_x\Sigma Q''$.

By $F_x\Sigma Q'' = F_{\bar{x}}\Sigma\bar{Q}'$ and (28):

$$\Sigma R'' = [\Sigma P''/x]\Sigma Q'' = \Sigma Q''(N_0 \longleftarrow \Sigma P'') \ldots (N_{-1} \longleftarrow \Sigma P'').$$

Comparing this with (29), we find that we have most of the conditions needed to establish $\Sigma R'' \simeq \Sigma S'$ with the help of Lemma 9.6. The conditions still to be verified are (9.6.6) and (9.6.7):

(31) $(\forall y, z \in X)[F_y\Sigma Q'' = F_z\Sigma\bar{Q}' \neq \emptyset \text{ implies } (\forall k < K)(F_y\Sigma P'' = F_z\Sigma\bar{P}'_k)]$

(32) $(\forall y, z \in X)(\forall j, k < K)(F_y\Sigma P'' = F_z\Sigma\bar{P}'_j \neq \emptyset \text{ implies } F_y\Sigma P'' = F_z\Sigma\bar{P}'_k)$

Suppose $F_y\Sigma Q'' = F_z\Sigma\bar{Q}' \neq \emptyset$, so that (25) and Lemma 10.7 imply

(33) $F_y\Sigma Q' = F_z\bar{Q} \neq \emptyset$.

For any $k < K$ there are two cases to consider.

Case a ($F_y\Sigma P'' \neq \emptyset$) By (26), some $w \in X$ has

$$F_y\Sigma P'' = F_w\Sigma\bar{P}'_k \neq \emptyset.$$

To this we apply (27), then Lemma 10.7, and then (23), so that

$$F_y\Sigma Q' = F_w\bar{Q}.$$

But then $w = z$ by (33), so $F_y\Sigma P'' = F_z\Sigma\bar{P}'_k$.

Case b ($F_y\Sigma P'' = \emptyset$) Suppose $F_z\Sigma\bar{P}'_k \neq \emptyset$. Reasoning as in Case a, we find that some $w \in X$ has $F_w\Sigma P'' = F_z\Sigma\bar{P}'_k \neq \emptyset$ and then that $w = y$. Therefore $F_y\Sigma P'' \neq \emptyset$, a contradiction. We must actually have $F_z\Sigma\bar{P}'_k = \emptyset = F_y\Sigma P''$. This completes the proof of (31).

Now suppose that $F_y \Sigma P'' = F_z \Sigma \bar{P}'_j \neq \emptyset$. To this we apply (27), then Lemma 10.7, and then (22), so that

(34) $$F_y \Sigma P' = F_z \bar{P}_k.$$

By $F_y \Sigma P'' \neq \emptyset$ and $\Sigma P'' \simeq \Sigma \bar{P}'_k$, some $w \in X$ has

$$F_y \Sigma P'' = F_w \Sigma \bar{P}'_k \neq \emptyset.$$

By (27) and Lemma 10.7, this implies $F_y \Sigma P' = F_w \bar{P}_k \neq \emptyset$. Comparison with (34) yields $w = z$, so $F_y \Sigma P'' = F_z \Sigma \bar{P}'_k$. This proves (32).

Finally, we must prove (3). Suppose that $F_y \Sigma R'' = F_z \Sigma S'$. For any $k < K$ we have

$$F_y \Sigma R'' = F_x \Sigma Q'' \cdot F_y \Sigma P'' \cup (F_y \Sigma Q'' - F_x \Sigma Q'')$$

$$F_z \Sigma S' = F_{\bar{x}} \Sigma \bar{Q}' \cdot F_z \Sigma \bar{P}'_k \cup (F_z \Sigma \bar{Q}' - F_{\bar{x}} \Sigma \bar{Q}'),$$

and so

(35) $$F_x \Sigma Q'' \cdot F_y \Sigma P'' = F_{\bar{x}} \Sigma \bar{Q}' \cdot F_z \Sigma \bar{P}'_k$$

(36) $$F_y \Sigma Q'' - F_x \Sigma Q'' = F_z \Sigma \bar{Q}' - F_{\bar{x}} \Sigma \bar{Q}',$$

where both equations cannot be $\emptyset = \emptyset$.

<u>Case a</u> $(F_x \Sigma Q'' \cdot F_y \Sigma P'' = \emptyset)$ Then (36) cannot be $\emptyset = \emptyset$, so $y \neq x$ and $z \neq \bar{x}$. Applying (25) and Lemma 10.7 to (36) yields

$$F_y \Sigma Q' = F_z \bar{Q} \neq \emptyset \;\&\; y \neq x \;\&\; z \neq \bar{x}.$$

By $\Sigma R' \simeq S$ and the equations (19), this implies $F_y \Sigma R' = F_z S$.

<u>Case b</u> $(F_x \Sigma Q'' \cdot F_y \Sigma P'' \neq \emptyset)$ Then $K \neq 0$ and (35) implies that

$$F_y \Sigma P'' = F_z \Sigma \bar{P}'_0 \neq \emptyset.$$

By (27) and Lemma 10.7 we have

$$F_y \Sigma P' = F_z \overline{P}_0 \neq \phi.$$

By $\Sigma R' \simeq S$ and $K \neq 0$ in (19), this implies $F_y \Sigma R' = F_z S$. ∎

(10.11) <u>Theorem</u>. The GRS $\mathfrak{B}_\beta$ is Church-Rosser.

<u>Proof</u>: First we will show that $(\mathbb{F}_0, \underset{1}{\Longrightarrow})$ is Church-Rosser, where $\underset{1}{\Longrightarrow}$ is from (10.6.3):

$$(\underset{1}{\Longrightarrow}) = (\overset{*}{\underset{\gamma}{\Longrightarrow}} \overset{*}{\underset{\sigma}{\Longrightarrow}} \cong) \cap \mathbb{F}_0 \times \mathbb{F}_0 .$$

By Lemma 3.4 and transitivity of $\cong$, it will suffice to verify Figure 4-2.

The SRS $\mathfrak{C}_\gamma$ defined by (10.6) is unequivocal and is closed because of the Rule-Schemata Theorem (6.5). By the Main Theorem (5.6), $\mathfrak{C}_\gamma$ is Church-Rosser. Therefore we fill in (D1). Figure 4-1 is available as a stencil because of Lemma 10.10, so we fill in (D2) and (D3). Finally, we fill in (D4) because normal forms are unique in $\mathfrak{C}_\sigma$ and any tree in $\mathbb{F}$ can be alphanormalized by changing bound variables.

Now suppose $\mathcal{R}, \mathcal{S}, \mathcal{S}' \in \mathbb{E}$ with $\mathcal{R} \overset{*}{>}_\beta \mathcal{S}$ and $\mathcal{R} \overset{*}{>}_\beta \mathcal{S}'$. We will show there is a $\mathcal{T} \in \mathbb{E}$ such that $\mathcal{S} \overset{*}{>}_\beta \mathcal{T}$ and $\mathcal{S}' \overset{*}{>}_\beta \mathcal{T}$. Let $\mathcal{R} \overset{J}{>}_\beta \mathcal{S}$ and $\mathcal{R} \overset{K}{>}_\beta \mathcal{S}'$. For J = 0 or K = 0 the choice of $\mathcal{T}$ is trivial, so we may assume $J \neq 0$ and $K \neq 0$. Let

$$(\underset{0}{\Longrightarrow}) = (\underset{\beta}{\Longrightarrow} \cong) \cap \mathbb{F}_0 \times \mathbb{F}_0 .$$

By $1+(J-1)+(K-1)$ applications of Lemma 10.1, there are $R \in \mathcal{R}$ and $S \in \mathcal{S}$ and $S' \in \mathcal{S}'$ with

(1) $$R \overset{J}{\underset{0}{\Longrightarrow}} S \ \& \ R \overset{K}{\underset{0}{\Longrightarrow}} S' .$$

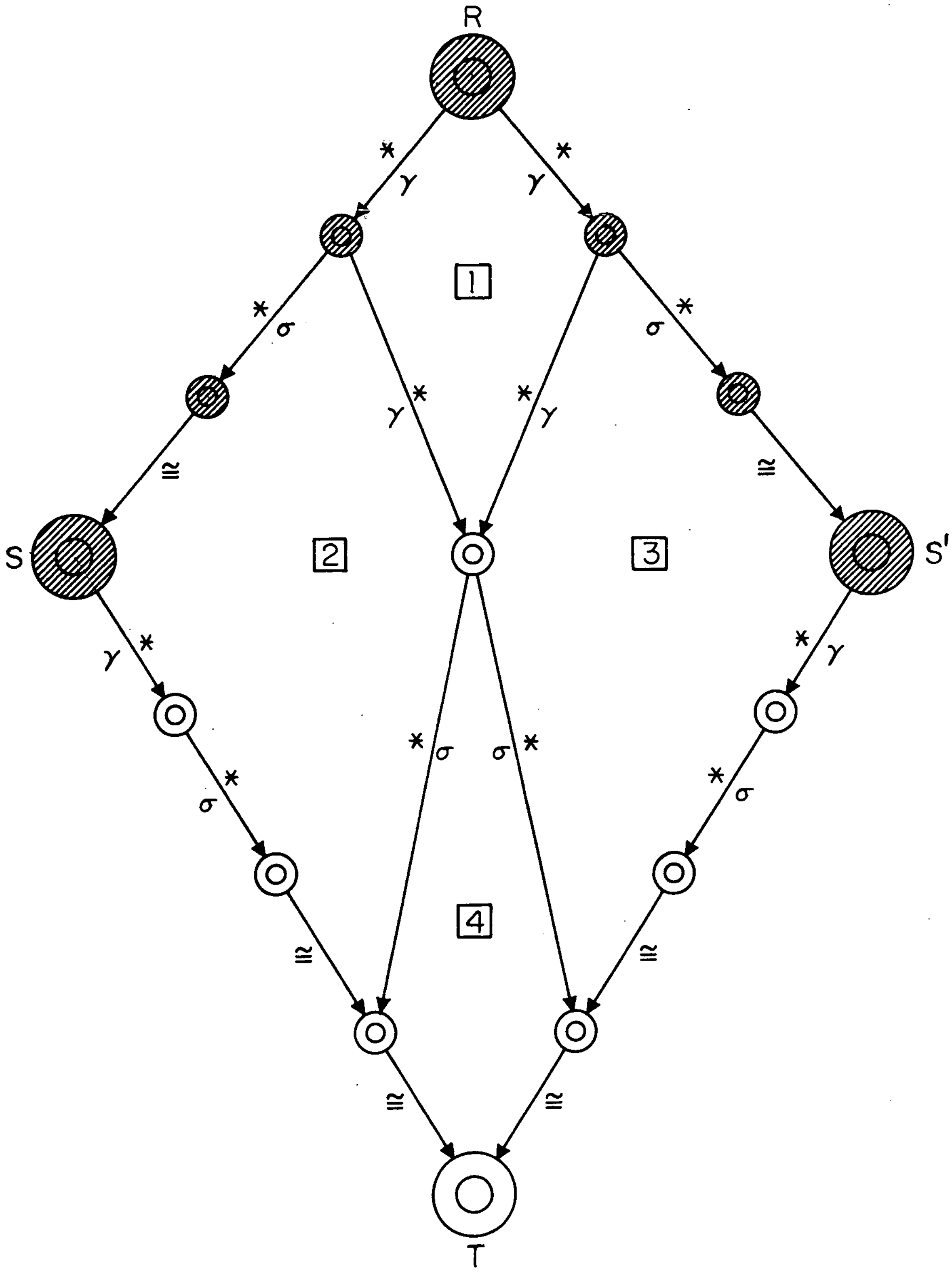


Figure 4-2. Diagram for the proof that ($\mathbb{F}_0, \xrightarrow[1]{}$) is Chruch-Rosser. Small circles represent members of $\mathbb{H}$; large circles represent members of $\mathbb{F}_0$.

By the definition of $\mathfrak{C}_\gamma$ in (10.6) and the ability of $\mathfrak{C}_\sigma$ to perform substitutions (established by (1) in Lemma 10.5), we have $(\underset{\beta}{\Longrightarrow}) \subseteq (\underset{\gamma}{\Longrightarrow} \overset{*}{\underset{\sigma}{\Longrightarrow}})$, so that $(\underset{0}{\Longrightarrow}) \subseteq (\underset{1}{\Longrightarrow})$. By (1) and the Church-Rosser property for $(\mathbb{F}_0, \underset{1}{\Longrightarrow})$, there is a $T \in \mathbb{F}_0$ such that $S \overset{*}{\underset{1}{\Longrightarrow}} T$ and $S' \overset{*}{\underset{1}{\Longrightarrow}} T$. By Lemma 10.8, this implies that $\mathcal{S} \overset{*}{>}_\beta \mathcal{T}$ and $\mathcal{S}' \overset{*}{>}_\beta \mathcal{T}$ for $\mathcal{T} \in \mathbb{E}$ with $T \in \mathcal{T}$.∎

(10.12) <u>Theorem</u>. The GRS $\mathfrak{B}_\lambda$ is Church-Rosser.

<u>Proof</u>: We use the Commutative Union Theorem (3.5). By Lemma 10.2, Lemma 10.3, and Theorem 10.11, $\{\mathfrak{B}_a \mid a \in \{\beta, \eta\delta\}\}$ is a family of Church-Rosser GRSs $\mathfrak{B}_a = (\mathbb{E}, >_a)$ that commute with each other. Therefore the union $(\mathbb{E}, >_\beta \cup >_{\eta\delta})$ is Church-Rosser. But

$$\mathfrak{B}_\lambda = (\mathbb{E}, >_\lambda) = (\mathbb{E}, >_\beta \cup >_{\eta\delta})$$

because $(\underset{\lambda}{\Longrightarrow}) = (\underset{\beta}{\Longrightarrow}) \cup (\underset{\eta\delta}{\Longrightarrow})$ and each $>_i$ is induced by the corresponding $\underset{i}{\Longrightarrow}$ in Definition 9.8. Therefore $\mathfrak{B}_\lambda$ is Church-Rosser.∎

Curry and Feys [14, §$\underset{\sim}{4}$S] review the early work on the Church-Rosser theorem. Theirs is the first proof that $\mathfrak{B}_\lambda$ is Church-Rosser and the first correct proof that $\mathfrak{B}_\beta$ is Church-Rosser. They show that a complex array of abstract postulates implies the Church-Rosser property [14, §$\underset{\sim}{4}$C2 (Thm. 2), §$\underset{\sim}{4}$A4 (Thm. 5), §$\underset{\sim}{4}$A3 (Thm. 4)] and that these postulates hold for $\mathfrak{B}_\beta$ [14, §$\underset{\sim}{4}$B3 (Thm. 3), §$\underset{\sim}{4}$C1 (Thm. 1)]. Hindley [21] and Schroer [48] also proved that $\mathfrak{B}_\beta$ is Church-Rosser by arguments of this form. Hindley's proof that his postulates imply the Church-Rosser property appears in [22] and his proof that $\mathfrak{B}_\beta$ satisfies his postulates will appear in [23]. The three arguments are all quite difficult and dissimilar to our proof.

Mitschke [36, §§2-4] independently proved that $\mathfrak{B}_\beta$ is Church-Rosser with a construction similar to our $\mathfrak{C}_\gamma$. Aside from minor notational differences, the only difference between the constructions is that he defines substitution for free variables and $\mathbb{R}_\beta$ [36, §2] in the manner of Curry and Feys. As was remarked at the end of §9, we have almost returned to Church's original formulation of these notions.

Mitschke's argument relating his $\mathfrak{C}_\gamma$ to $\mathfrak{B}_\beta$ is organized differently but resembles ours in that both are shallow analyses of $\mathfrak{B}_\beta$ and $\mathfrak{C}_\gamma$, in contrast to the deep analyses of $\mathfrak{B}_\beta$ alone in earlier proofs. We have been completely scrupulous about alphabetic changes. A similarly detailed exposition would lengthen Mitschke's argument (particularly in Claim II in the proof of [36, §4, Thm. 4.2]), but it could still be somewhat shorter than ours.

The idea of applying the Commutative Union Theorem (3.5) to $\mathfrak{B}_\beta$ and $\mathfrak{B}_{\eta\delta}$ was discovered by Hindley [21] and independently by the author [41, §5]. Hindley's exposition of the idea will appear in [23]. Curry and Feys used a complicated special argument [14, §$\underset{\sim}{4}$D].

The proofs of Theorems 10.11 and 10.12 use general theorems about SRSs to establish intermediate results such as the Church-Rosser property for $\mathfrak{C}_{\eta\delta}$ and $\mathfrak{C}_\gamma$ and (3.6.1) in the proof that $\mathfrak{B}_\beta$ and $\mathfrak{B}_{\eta\delta}$ commute (Lemma 10.3). In this respect, our treatment appears to be unique. For us the classical Church-Rosser theorem is an example, not a goal.

CHAPTER 5

APPLICATIONS TO TREE TRANSDUCERS

This chapter studies tree transducers in terms of subtree replacement systems. We develop basic mathematical properties of tree transducers and explain how these properties can assist in constructing syntax-directed compilers.

In §11 we review some well-known topics in formal language theory that are clearly relevant to compiler construction. We also note some practical difficulties which prevent these ideas themselves from being a satisfactory theory of compilers. A model for syntax-directed compilation that permits the use of formal language theory without ignoring these difficulties is described in §12. The model consists of five devices connected in sequence. The first device reads in a source program and the last device writes out machine code. For each of the five abstract machines used, there is a significant body of theoretical results or practical experience (or both) to assist in designing and implementing the specific device to be used in each particular compiler. The use of coroutine linkages to compress the five stages into a single pass process is discussed.

One of the five components of our model for compilers is a <u>tree transducer</u>: a device that maps trees to trees in a manner reminiscent of the operation of finite transducers on strings. The basic theory of tree transducers is developed in §13. This section is mostly exegesis on the work of Rounds [45][46] and Thatcher [51][52]. We obtain some very modest extensions of some of their results and we indicate the significance of this work for compiler construction.

One of the problems in this area is complex enough to be discussed in a section by itself. In §14 we consider the question of whether the composition of two maps, both definable by a certain type of transducer, is also definable by a transducer of the same type. After explaining how such closure-under-composition theorems could be helpful in constructing compilers, we elaborate slightly upon theorems of Rounds [46] and Thatcher [52] to the effect that certain important classes of transductions are closed under composition. (A transduction is the map defined by a transducer.) We also present a negative result: there is a pair of "linear" and "partial deterministic" transductions whose composition is not computable by any finite tree transducer, deterministic or nondeterministic, that reads input trees from the top down.

11. Formal Language Theory and Compiling

In this section we review some well-known topics from formal language theory in the context of compiling. We consider context-free grammars, finite transducers, syntax-directed transductions, and generalized syntax-directed translations.

Consider arithmetic expressions of the form $a+(a+a)\times a$, where the four a's represent arbitrary constants or variables. Such expressions may be evaluated by performing an addition, then a multiplication, and then another addition, as shown in the operator-operand structure displayed in Figure 5-1. It seems rather easy to produce machine code that evaluates $a+(a+a)\times a$ _if_ the tree structure is available.

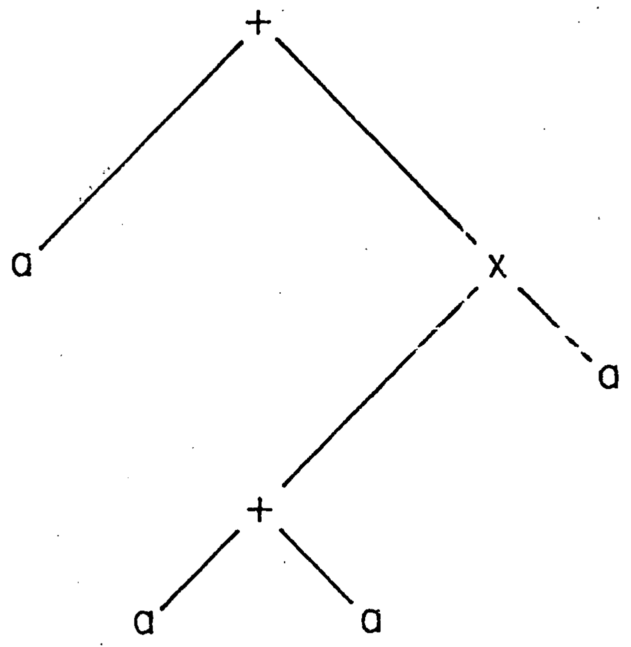


Figure 5-1. Operator-operand structure for a+(a+a)Xa.

But how is the operator-operand structure to be recovered from the string of characters that constitutes a computer program? Many constructions in programming languages do not even have an obvious meaning in terms of operator-operand structures. For example, the sequence of operations performed on the data by an ALGOL 60 **for** statement [38, §4.6] is only determined at run time.

Context-free grammars provide a convenient way to define sets of strings and assign tree structures to the strings in such a way that the structure of a string can often be recovered from the string itself. They can assign structures to **for** statements as naturally as to arithmetic expressions.

(11.1) Definition. A context-free grammar is any 4-tuple $G = (N,T,P,X)$, where N and T are disjoint finite nonempty sets, $X \in N$, and P is a finite subset of

$$\{\langle A, \omega\rangle \mid A \in N \ \&\ \omega \in (N \cup T)^* \ \&\ \omega \neq ()\}.$$

Members of N are called nonterminals while members of T are called terminals. The designated member X of N is the initial symbol. Members of P are called productions and are written $A \longrightarrow \omega$ rather than $\langle A, \omega\rangle$ (just as rules are written $\varphi \longrightarrow \psi$ rather than $\langle \varphi, \psi\rangle$).

There are several variants of this definition in use. We have followed Hopcroft and Ullman [24, §2.3] because this is a very accessible source with references to many of the others. The main difference between this definition and several others is that we require $\omega \neq ()$ in each production $A \longrightarrow \omega$. Programming languages never include the null string as a complete program, and there is a systematic procedure for eliminating all productions of the form $A \longrightarrow ()$ from context-free grammars for such languages [24, §4.6]. Since the problem of finding structures of strings is less complicated when no productions have the form $A \longrightarrow ()$, we prefer the narrower definition.

In order to explain how a context-free grammar assigns tree structures to strings we first define a function yield that maps trees to strings. The yield of a tree is the string formed by the labels on the leaves when listed in the natural left-to-right order. Using the algebraic nomenclature defined in (4.10), we define yield as follows.

(11.2) Definition. Let V be any set and let $\mathbb{P}$ be the set of all positive integers. Then yield : $V_* \longrightarrow V^*$ is the unique total function such that

(1) $(\forall \alpha \in V)(\underline{\text{yield}}(\bar{\alpha}) = (\alpha))$

(2) $(\forall \alpha \in V)(\forall K \in \mathbb{P})(\forall R \in (V_*)^K)$
$(\underline{\text{yield}}(\bar{\alpha}(R_0, \ldots, R_{-1})) = \underline{\text{yield}}(R_0) \cdot \underline{\text{yield}}(R_1) \cdot \ldots \underline{\text{yield}}(R_{-1}))$.

(Existence and uniqueness may be demonstrated by induction on sizes of trees.)

A context-free grammar $G = (N,T,P,X)$ defines a forest of "phrase structure" trees: each tree R displays one way to derive a string of terminals from the initial nonterminal X by applying productions.

(11.3) Definition. Let $G = (N,T,P,X)$ be a context-free grammar and let $V = N \cup T$. A tree $R \in V_*$ is a phrase structure generated by G iff $R() = X$, each leaf in Dom R is labelled by a terminal, and, whenever $n \in$ Dom R and n has sons $n \cdot (0), \ldots n \cdot (J-1)$ with $J \neq 0$ in Dom R, then there is a product $A \longrightarrow \omega \in P$ such that

$A = Rn \; \& \; |\omega| = J \; \& \; (\forall j < J)(\omega_j = R(n \cdot (j)))$.

Phrase structures are the "derivation trees" of [24, §2.6]. We prefer the term borrowed from linguistics [8, §3][9, Chap. 4] because it prevents confusion between phrase structures and another type of tree structure to be considered later in this section.

The language generated by a grammar is the set of all yields of phrase structures generated by the grammar. If each string generated is the yield of just one phrase structure, then the grammar is said to be unambiguous.

(11.4) Definition. Let $G = (N,T,P,X)$ be a context-free grammar and let $\mathcal{L}(G)$ be the set of all phrase structures generated by G. The language generated by G is

$$L(G) = \{\underline{\text{yield}}(R) \mid R \in \mathcal{L}(G)\}.$$

If

$$(\forall w \in L(G))(\exists ! \; R \in \mathcal{L}(G))(w = \underline{\text{yield}}(R))$$

then G is unambiguous.

A grammar for a programming language should be unambiguous. Unfortunately, there is no decision procedure for this property [24, Thm. 14.7].

For an example of a context-free grammar we return to $a+(a+a)\times a$ and similar forms for arithmetic expressions. Let

$$N = \{e,t,f\} \; \& \; T = \{(,),+,\times,a\}$$

$$
\begin{aligned}
P = \{ e &\longrightarrow t \\
e &\longrightarrow e+t \\
t &\longrightarrow f \\
t &\longrightarrow t \times f \\
f &\longrightarrow a \\
f &\longrightarrow (e) \}.
\end{aligned}
$$

The grammar $G_0 = (N,T,P,e)$ assigns to $a+(a+a)\times a$ the phrase structure shown in Figure 5-2. A direct proof that G_0 is unambiguous would be too laborious for consideration here. A more practical way to demonstrate unambiguity is to verify a stronger but decidable property, such as the LR(k) property [24, §12.5].

Suppose $G = (N,T,P,X)$ is an unambiguous grammar and $w \in L(G)$. To _parse_ w is to find the one tree $R \in \mathcal{L}(G)$ such that $w = \underline{\text{yield}}(R)$. There are several algorithms, each correct for a broad class of grammars, that read w from left to right and construct R with little or no backtracking. Some build R from the top down (i.e., from the root to the leaves), as in [16] or [44], while others build R from the bottom up, as in [15] or [19]. Rather than survey this area here, we will simply note that there _are_ efficient methods for context-free parsing.

Unfortunately, many programming languages cannot be generated by context-free grammars. Floyd [17] has shown that certain very natural constraints, such as the requirement that all identifiers used be declared, will remove a language from the context-free family. On the other hand, a programming language may be "almost" context-free in that it can be specified by a context-free grammar G together with an

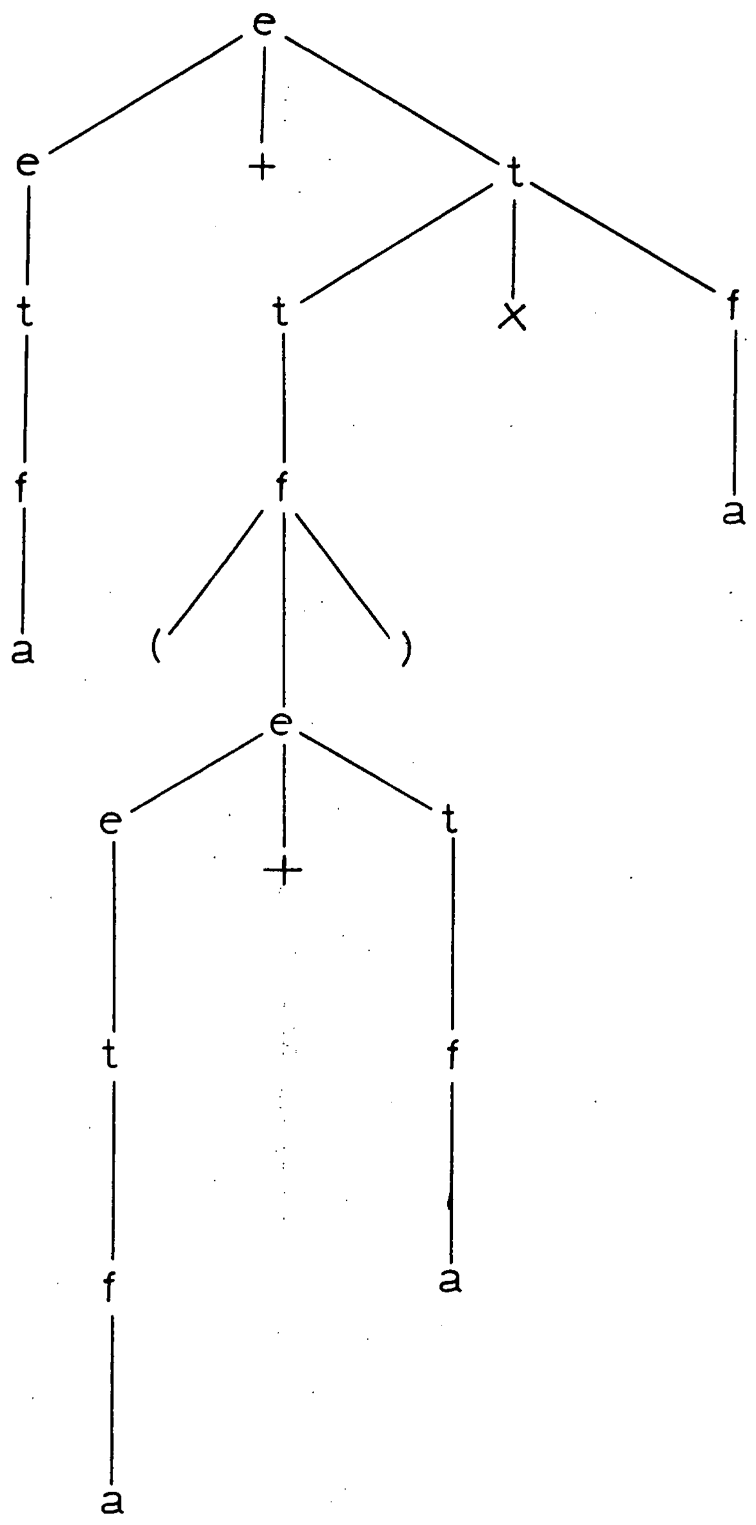


Figure 5-2. Phrase structure for a+(a+a)Xa.

informal discussion of the restrictions on allowable members of L(G), as in the definition of ALGOL 60 [38]. Since the theory and practice of context-free parsing are much more advanced than parsing techniques for more complex forms of grammar, there is good reason to use context-free grammars even for languages that are not quite context-free. There is even a systematic way to do this with a two-stage generative process consisting of a context-free grammar and a device that we will call a "lexical synthesizer." Using ALGOL 60 as an example, we will now sketch the two-stage method. (For the sake of simplicity, we delete the production $\langle\text{label}\rangle \longrightarrow \langle\text{unsigned integer}\rangle$, so that $\langle\text{label}\rangle \longrightarrow \langle\text{identifier}\rangle$ is the only production for $\langle\text{label}\rangle$ that remains in [38, §3.5.1].) The following discussion is based on remarks by Cheatham [7, pp. III.B.11, III.B.12, IX.A.1, IX.A.2].

Reserved words like begin are single terminal symbols in the ALGOL 60 grammar [38, §2, fn. 6], so some additional device that converts the symbol begin to the string of five characters BEGIN is needed. By simplifying the grammar and complicating this spelling device, we can specify the ALGOL 60 character strings formally. First we replace the productions for the nonterminal ⟨identifier⟩ by a new production

$$\langle\text{identifier}\rangle \longrightarrow \{\text{identifier}\}$$

where {identifier} is a new terminal. Unlike the terminals begin, +, and so on, this new terminal can be spelled as a character string in more than one way. We call it a <u>lexical variable</u>. In general, a programming language might need several lexical variables, as in {real identifier} and {integer identifier} in FORTRAN.

Let G be the modified ALGOL 60 grammar. In order to generate a program we first generate a phrase structure $R \in \mathcal{L}(G)$. The lexical synthesizer assigns a string ξ_k of characters to each position $k < |w|$ in $w = \underline{\text{yield}}(R)$. The concatenation $\xi_0 \cdot \xi_1 \cdot \ldots \xi_{|w|-1}$ of all these strings of characters is then a well-formed ALGOL 60 program. The way each $k < |w|$ is treated depends on the terminal w_k at that position. If w_k is an ordinary terminal like begin, then ξ_k is simply the spelling of w_k. In this case the lexical synthesizer acts like a homomorphism from strings of terminals to strings of characters. If w_k is a lexical variable like $\langle$identifier$\rangle$, however, then ξ_k is one of the character strings that the old grammar could generate from the nonterminal corresponding to w_k. In choosing ξ_k the lexical synthesizer does not merely apply the old rules. It also inspects relevant portions of the tree R and relevant ξ_j's for $j \neq k$, so as to obey the restrictions that prevent ALGOL 60 from being context-free. For example, actual label identifiers cannot be assigned to occurrences of $\langle$identifier$\rangle$ in such a way that a jump statement transfers into a block [38, §4.3.4]. The restriction on jumps, the requirement that each identifier used be declared, and several other indisputably well-motivated restrictions can be formalized readily because the lexical synthesizer has access to the tree R as well as the string w.

In order to invert the two-stage generative process with the help of context-free parsing techniques, we consider three processes in sequence: lexical analysis, context-free parsing, and lexical filtration.

The input to the lexical analyzer is a string Ξ of characters that is presumably a source program. The analyzer finds a string w of terminals and a string ξ_k of characters for each $k < |w|$ such that

$$\Xi = \xi_0 \cdot \xi_1 \cdot \ldots \xi_{w-1}$$

and each ξ_k is a possible spelling for the terminal w_k. The output of the lexical analyzer is w together with spelling information for each $k < |w|$ such that w_k is a lexical variable. One natural format for this output is the augmented terminal string

$$(\langle w_0, \beta_0\rangle, \langle w_1, \beta_1\rangle, \ldots, \langle w_{-1}, \beta_{-1}\rangle)$$

where each β_k is a pointer. If w_k is an ordinary terminal then β_k can be any "don't care" pointer. If w_k is a lexical variable then β_k is a true pointer to a symbol table location where the actual spelling ξ_k used in the program may be found. (Other data, such as which block an identifier belongs to, may be accumulated in the symbol table later.)

The input to the context-free parser is the augmented terminal string produced by the lexical analyzer. The parser treats each $\langle a, \beta\rangle$ like the terminal a, ignoring the pointer β, and finds the phrase structure $R \in \mathcal{L}(G)$ of $w \in L(G)$. (An error message results if $w \notin L(G)$.) Each leaf in R is actually labelled by a pair $\langle a, \beta\rangle$: the pointers are ignored but not erased. To be more precise, we say that the parser produces an augmented phrase structure.

Finally, the lexical filter determines whether the lexical variables at leaves of R could be generated by the synthesizer in accord with the way they are actually spelled in the program Ξ, as indicated by the symbol table entries pointed to by the augmented terminals. The output of the filter is R if the actual spellings could be generated by the synthesizer and an error message otherwise. This is the stage that detects undeclared identifiers and similar faults in programs that are well-formed in terms of the context-free portion of the syntax. The lexical filter inspects structures that the lexical analyzer and context-free parser have already provided. This is not a very difficult task, so we will not consider formal models for lexical filtration.

We have sketched a three-stage method for context-free parsing of languages that are not quite context-free. Lexical filtration is not very difficult and several efficient context-free parsing methods are available, as in [15] [16] [19] [44], but we have said nothing about how lexical analysis is to be performed. Our next task is to define a concept from formal language theory that is useful as a mathematical model for the most difficult part of lexical analysis.

The source program Ξ must be analyzed as $\xi_0 \cdot \ldots \xi_{|w|-1}$ for some string w of terminals such that w_k can be spelled as ξ_k for each $k < K$. As Ξ is read from left to right, the analyzer should sometimes decide that the end of a ξ_k has been reached and should output the appropriate $\langle w_k, \beta_k \rangle$ as the next symbol of the augmented terminal string. The symbol table pointers β_k can be assigned by well-known methods for searching and updating tables (such as hash coding), so we will only concern ourselves with the mapping from Ξ to w. We wish to define a class of abstract devices that are easy to implement and that can map strings to strings efficiently.

(11.5) Definition. Let M be any 7-tuple

$$M = (K, \Sigma, \Delta, \delta, \lambda, s, F)$$

where K, Σ, Δ are finite sets, $s \in K$, $F \subseteq K$, and

$$\delta : K \times \Sigma \longrightarrow K$$

$$\lambda : K \times \Sigma \longrightarrow \Delta^*.$$

Then M is a deterministic finite string transducer with set K of states, input vocabulary Σ, output vocabulary Δ, transition map δ, output map λ, starting state s, and set F of final states.

The adjective "finite" modifies "transducer," not "string." Strings are defined to be finite already. The deterministic finite string transducers are often called "generalized sequential machines with final states" or "deterministic a-transducers."

(11.6) Definition. Let $M = (K, \Sigma, \Delta, \delta, \lambda, s, F)$ be a deterministic finite string transducer. The extended transition map $\bar{\delta} : K \times \Sigma^* \longrightarrow K$ is defined by

(1) $\bar{\delta}(q, ()) = q$

and

(2) $\bar{\delta}(q, (a)\cdot x) = \bar{\delta}(\delta(q, a), x)$

for all $q \in K$; $x \in \Sigma^*$; $a \in \Sigma$. The extended output map $\bar{\lambda}: K \times \Sigma^* \longrightarrow \Delta^*$ is defined by

(3) $\bar{\lambda}(q, ()) = ()$

and

(4) $\bar{\lambda}(q, (a)\cdot x) = \lambda(q, a) \cdot \bar{\lambda}(\delta(q, a), x)$

for all $q \in K$; $x \in \Sigma^*$; $a \in \Sigma$. The transduction computed by M is the partial function $\langle M \rangle : \Sigma^* \longrightarrow \Delta^*$ defined by

(5) $\langle M \rangle = \{ \langle x, \bar{\lambda}(s, x) \rangle \mid x \in \Sigma^* \ \& \ \bar{\delta}(s, x) \in F \}.$

In our intended application to compiling, the input vocabulary Σ is the set of characters available for writing programs. The output vocabulary Δ is the set T of terminals in a context-free grammar (N, T, P, X). The set F of final states may be used to reject certain character strings without the expense of an attempt at parsing. If M is not in a final state after processing the input string then an error message results. If no such test seems to be appropriate, we may let $F = K$, so that all character strings are transduced to terminal strings and sent to the parser. In this case it is customary to omit the reference to F and call the 6-tuple $(K, \Sigma, \Delta, \delta, \lambda, s)$ a "generalized sequential machine." (Hopcroft and Ullman call M a "generalized sequential machine" even when $F \neq K$ [24, §9.3], but this is a departure from the more usual nomenclature.)

Many mathematical properties of finite transducers have been established, as can be seen by consulting [24] and the references cited there. In §14 we will consider the uses of one of these properties: the class of transductions defined by deterministic finite string transducers is closed under composition. These transducers also appear as parts of an important LR(k) parsing system [15].

Finite transducers are reasonably straightforward to implement. Whenever a character is read the lexical analyzer consults a $|K|$ by $|\Sigma|$ matrix of entries $(\delta(q, a), \lambda(q, a))$ for $q \in K$ and $a \in \Sigma$. Unless a matrix with $|K|\,|\Sigma|$ entries is unmanageably large, implementation of a finite transducer is quite easy. Finally, the deterministic finite string transducers appear to be powerful enough to perform lexical analysis without unreasonably restricting the programming language designer [7, Chap. IX].

We have sketched the role of context-free grammars and finite transducers in assigning tree structures to computer programs. After lexical analysis, parsing, and lexical filtration, a source program has been transformed to a phrase structure tree augmented by pointers to symbol table entries at leaves labelled by lexical variables. Concepts and results from formal language theory are helpful in designing and evaluating lexical analyzers and context-free parsers, while lexical filters are rather straightforward to design because of the explicitly structured data available to them. But how can augmented tree structures be used to generate machine code?

Before sketching two relatively well-known mathematical proposals for generating code from augmented tree structures and the practical shortcomings of these proposals, we must introduce a slight complication. Phrase structures are simple to define and intuitively natural, but a somewhat less transparent class of trees has several technical advantages in the detailed consideration of parsing or code generating algorithms. The "ranked parse trees" defined below have nodes labelled by productions rather than by terminals and nonterminals. Recall the definition of ranked trees (4.11).

(11.7) Definition. Let $G = (N, T, P, X)$ be a context-free grammar. For each $\pi \in P$ let $\rho(\pi)$ be the unique $K \in \mathbb{N}$ such that π has the form

(1) $A \longrightarrow x_0 \cdot (B_0) \cdot \ldots x_{K-1} \cdot (B_{K-1}) \cdot x_K$

where $A, B_0, \ldots, B_{K-1} \in N$ and $x_0, \ldots, x_K \in T^*$, so that $\rho : P \longrightarrow \mathbb{N}$. A tree $S \in P_\#$ is a ranked parse tree generated by G iff $S(\,)$ is a production $X \longrightarrow \chi$ for the initial symbol X and, whenever $n \in \text{Dom } S$ and

$k < K$ for $K = \rho(Sn)$, then $S(n\cdot(k))$ is a production $B_k \longrightarrow \omega$ for the nonterminal B_k when Sn is expressed in the form (1).

Aho and Ullman call such structures "parse trees" [2, §2]. We have added the word "ranked" to prevent confusion between these trees and phrase structures, which are also called "parse trees" by some authors.

For an example, we return to the grammar $G_0 = (N, T, P, e)$ for forms of arithmetic expressions. We have $N = \{e,t,f\}$ and $T = \{(\,,\,), \times, a\}$. The productions are

$$\begin{array}{ll}
\pi_0 = e \longrightarrow t & \text{with } \rho(\pi_0) = 1 \\
\pi_1 = e \longrightarrow e+t & \text{with } \rho(\pi_1) = 2 \\
\pi_2 = t \longrightarrow f & \text{with } \rho(\pi_2) = 1 \\
\pi_3 = t \longrightarrow t\times f & \text{with } \rho(\pi_3) = 2 \\
\pi_4 = f \longrightarrow a & \text{with } \rho(\pi_4) = 0 \\
\pi_5 = f \longrightarrow (e) & \text{with } \rho(\pi_5) = 1 .
\end{array}$$

The ranked parse tree shown in Figure 5-3 conveys the same information about $a+(a+a)\times a$ as the phrase structure tree shown in Figure 5-4.

By comparing the definition of phrase structures in (11.3) with the definition of ranked parse trees in (11.7), we can show that there is a bijection between the two forests for any context-free grammar. A ranked parse tree is essentially a phrase structure tree where each node labelled by a nonterminal A has been replaced with a node labelled by the appropriate production $A \longrightarrow \omega$.

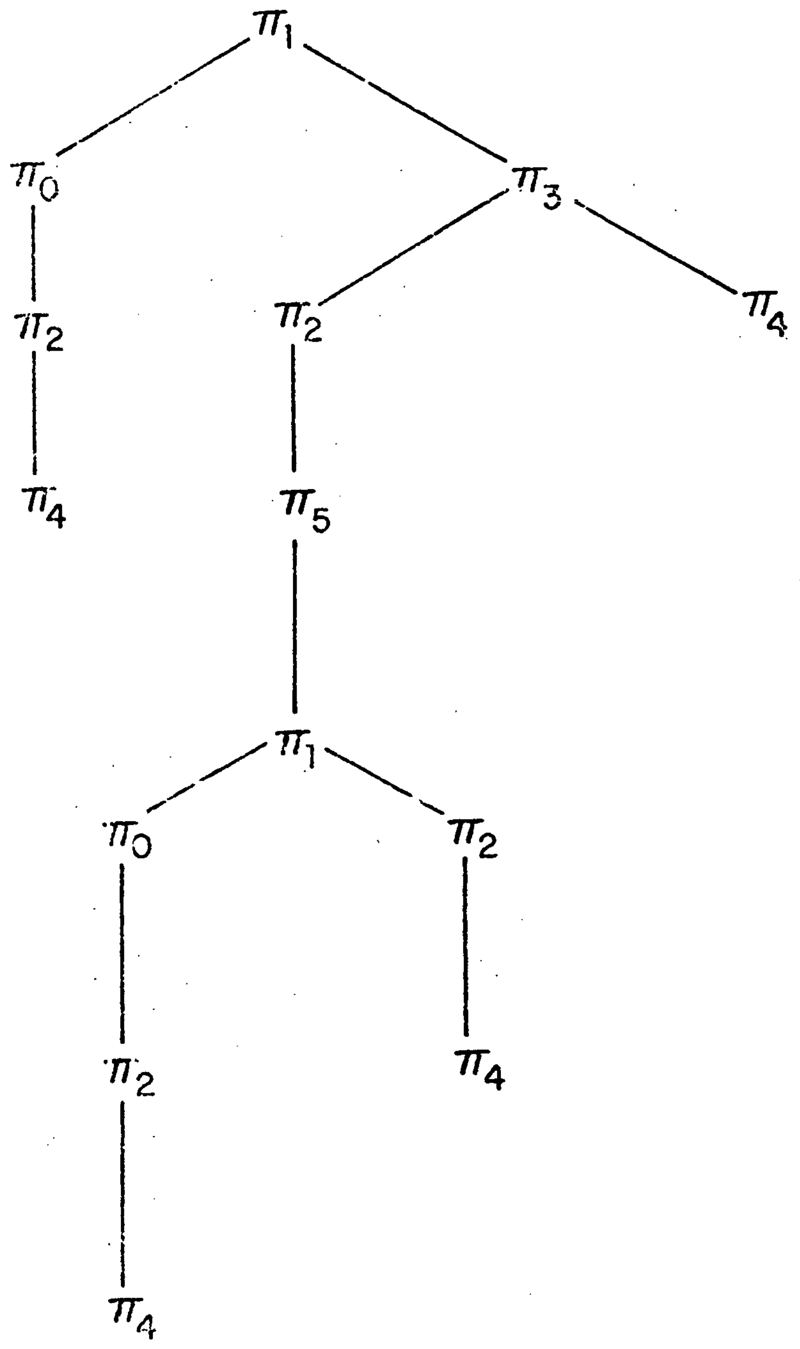


Figure 5-3. Ranked parse tree for a+(a+a)Xa.

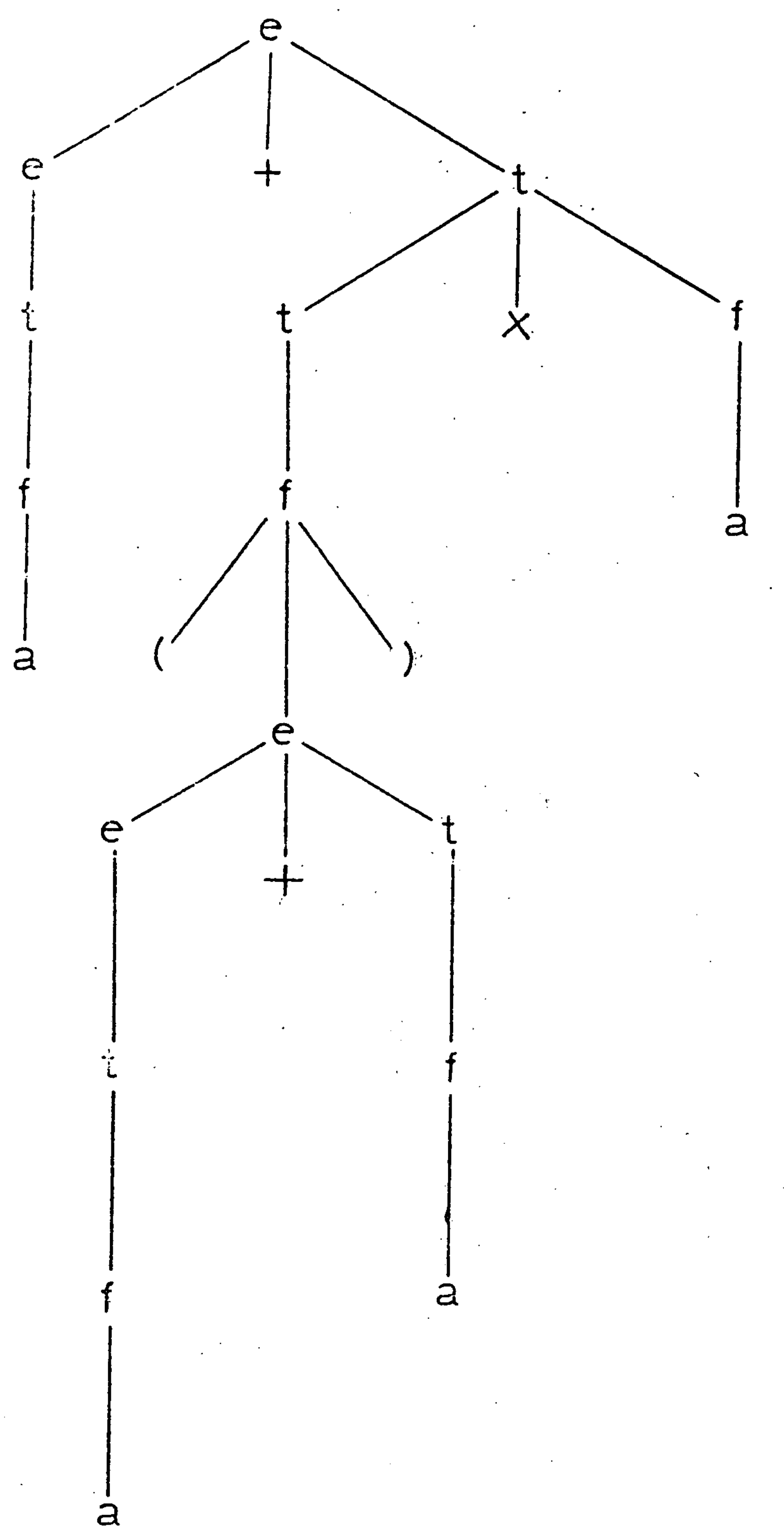


Figure 5-4. Phrase structure for a+(a+a)Xa.

In the intended application to compilers, lexical variables will only occur in productions of the form

$$\langle\text{syntactic category}\rangle \longrightarrow \{\text{syntactic category}\},$$

so that each ω in a production $A \longrightarrow \omega$ is free of lexical variables or is of the form (a) for some lexical variable a. Corresponding to augmented phrase structures, we have augmented ranked parse trees whose leaves are labelled by pairs $\langle\pi, \beta\rangle$ where π is a production of rank zero and β is a pointer. If π is $A \longrightarrow \omega$ with ω free of lexical variables then β is a "don't care" pointer, but if π is $A \longrightarrow$ (a) for some lexical variable a then β points to a symbol table entry.

We will assume that the code to be generated from an augmented ranked parse tree depends mainly on the basic tree, without the pointers. The symbol table is consulted for the actual address fields of machine instructions and perhaps for some details such as the choice between real number and integer arithmetic. The basic ranked parse tree should suffice to determine the machine language program except for some blank spaces to be filled in by following pointers to the symbol table in a straightforward way. This is the semantic aspect of the idea that programming languages are almost context-free. Therefore we consider the problem of generating code from a ranked parse tree without regard to the pointers that are attached to the leaves in practice.

Let MLR be the set of all machine language routines definable by phrases in programs in the language being compiled, where a phrase is a segment of a program that can be traced back to a single node in the phrase structure or ranked parse tree. Let P be the set of productions in the grammar, with a rank function ρ in Definition 11.7. We might assign to each $\pi \in P$ an operation

$$I_\pi : \mathrm{MLR}^{\rho(\pi)} \longrightarrow \mathrm{MLR}$$

so that, if a tree in $P_\#$ has the form $\pi(R_0, \dots, R_{-1})$ and each R_k determines a machine language routine $\mathfrak{R}_k \in \mathrm{MLR}$, then $\pi(R_0, \dots, R_{-1})$ determines the routine $I_\pi(\mathfrak{R}_0, \dots, \mathfrak{R}_{-1})$. By interpreting π as the operation I_π on machine code, we can assign a machine language program to each ranked parse tree. This is exactly the same process we used for Lemma 8.2 in Chapter 3, where each operator-operand tree could be assigned a value in the extended data space $\bar{\mathbb{D}}$ after each operator had been interpreted as an operation on $\bar{\mathbb{D}}$. In the nomenclature of Brooker and Morris [43, §§1, 4, 6], each I_π is a "format routine" macro-instruction. In the nomenclature of Knuth [27], we are using a single "synthesized attribute" whose range of values is MLR.

The scheme just sketched is too flexible, rather as if we had allowed an arbitrary Turing machine to perform lexical analysis. There do not appear to be many theorems that could assist someone who wishes to construct a compiler along these lines. We will now consider some restrictions on MLR and the I_π's that lead to a richer theory.

In defining "syntax-directed transductions," Lewis and Stearns [30] assume that MLR is a subset of Γ^* for some finite set Γ and that each I_π has the form

(1) $$(\forall y_0, \dots, y_{K-1} \in \mathrm{MLR})(I_\pi(y_0, \dots, y_{K-1}) = x_0 \cdot y_{f(0)} \cdot \dots x_{K-1} \cdot y_{f(K-1)} \cdot x_K)$$

where $K = \rho(\pi)$ and $x_0, \dots, x_{K-1} \in \Gamma^*$ and $f : \{0, \dots, K-1\} \longrightarrow \{0, \dots, K-1\}$ is a bijection. These assumptions hold in their example [30, p. 466], where the source language consists of simple arithmetic expressions and the target machine has a hardware stack, but not in some other important situations. As Aho and Ullman point out in the case of for statements [2, p. 94], the routine $I_\pi(y_0, \ldots, y_{K-1})$ might require several copies of some of the y_k's. The desired flexibility can be obtained by replacing (1) with

$$(2) \quad (\forall y_0, \ldots, y_{K-1} \in \mathrm{MLR})(I_\pi(y_0, \ldots, y_{K-1}) = x_0 \cdot y_{f(0)} \cdot \ldots x_{J-1} \cdot y_{f(J-1)} \cdot x_J)$$

where $K = \rho(\pi)$ and $J \in \mathbb{N}$ and $x_0, \ldots, x_{J-1} \in \Gamma^*$ and $f : \{0, \ldots, J-1\} \longrightarrow \{0, \ldots, K-1\}$. We simply omit the assumption that $J = K$ and f is a bijection. The "generalized syntax-directed translations" proposed in [2] are yet more flexible than this. A tree in $P_\#$ may define several machine language routines, one for each of several "translation symbols" [2, §3] associated with the left half of the production at the root. To simplify the notation we suppose that all nonterminals have the same number $H \neq 0$ of translation symbols, so that a tree in $P_\#$ determines a sequence of H machine language routines. The code defined by a ranked parse tree is defined to be the first component of the member of MLR^H determined by the tree as a member of $P_\#$. Instead of (2) we have $I_\pi : (\mathrm{MLR}^H)^K \longrightarrow \mathrm{MLR}^H$, and each $h < H$ has

$$(3) \quad (\forall Y^0, \ldots, Y^{K-1} \in \mathrm{MLR}^H)$$
$$([I_\pi(Y^0, \ldots, Y^{K-1})]_h = x_0 \cdot Y^{f(0)}{}_{g(0)} \cdot \ldots x_{J-1} \cdot Y^{f(J-1)}{}_{g(J-1)} \cdot x_J)$$

for some $x_0, \ldots, x_J \in \Gamma^*$ and $f : \{0, \ldots, J-1\} \longrightarrow \{0, \ldots, K-1\}$ and $g : \{0, \ldots, J-1\} \longrightarrow \{0, \ldots, H-1\}$. The choices of J, $x_0, \ldots, x_J$, f, and g depend on h.

The concept of "generalized syntax-directed translations" summarized by (3) shares a fundamental restriction with narrower theory of "syntax-directed transductions" summarized by (1): machine code consists of strings of symbols and the only way to combine several machine language routines is to concatenate them.

To concatenate routines is to link them in a linear sequence. This is not a sufficiently powerful way to manipulate code, since it prohibits many common uses of branch instructions. For example, suppose a statement in an ALGOL 60 program has the form

$$\text{IF } \mathcal{P} \text{ THEN } \mathcal{A} \text{ ELSE } \mathcal{B}$$

and that the character strings $\mathcal{P}$, $\mathcal{A}$, $\mathcal{B}$ determine machine language routines $\mathfrak{P}$, $\mathfrak{A}$, $\mathfrak{B}$. Suppose that executing $\mathfrak{P}$ would place the value of $\mathcal{P}$ in some register X whose contents can be tested by a branch instruction. Then $\mathfrak{P}$, $\mathfrak{A}$, $\mathfrak{B}$, and the branch instruction should be linked as shown in Figure 5-5.

Fortunately, the results in [2] deal with the size of the compiled program as a function of the size of the source program, without really using the linear linkage assumption. It is enough to assume that size is measured in such a way that the size of each combination of machine language routines is a constant plus the sum of the sizes of the components. For example, suppose that the size of a routine is the number of machine instructions in it, and that the normal flow of control is from each instruction to its successor in a sequence of instructions. Then the size of the routine shown in Figure 5-5 is

$$2 + \text{size}(\mathfrak{P}) + \text{size}(\mathfrak{A}) + \text{size}(\mathfrak{B}),$$

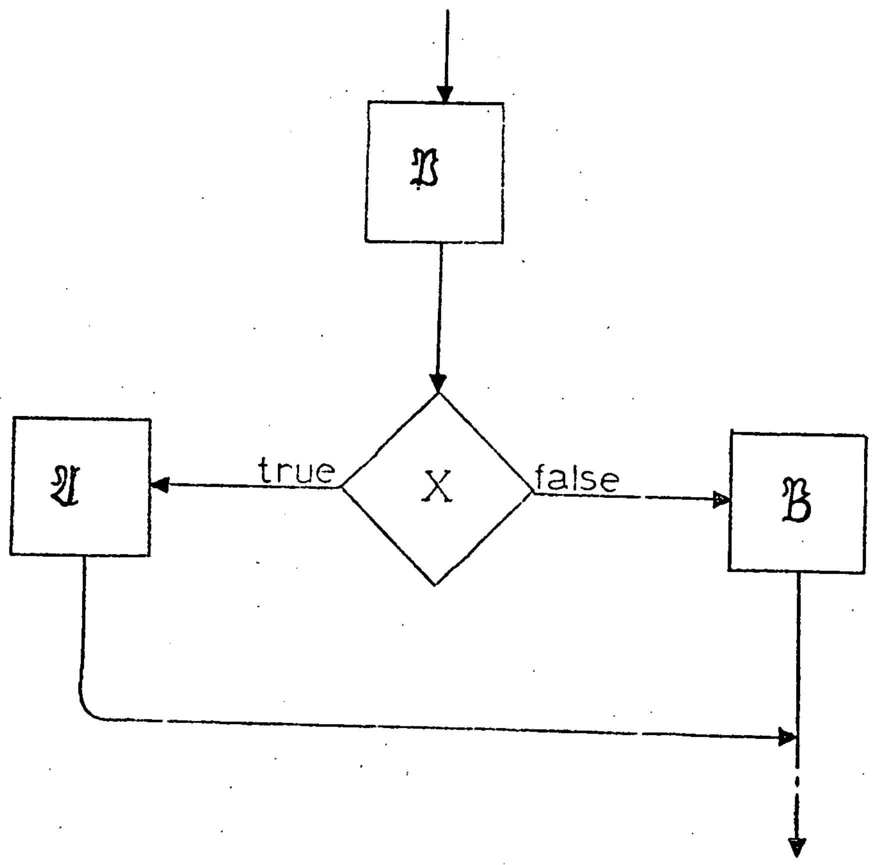


Figure 5-5. Nonlinear linkage in the code for a conditional statement.

where the branch instruction and one transfer instruction (the bend in the link down from $\mathfrak{A}$) account for the 2.

For further study of context-free semantics we wish to avoid the unrealistic linear linkage assumption, but we also wish to assume somewhat more about machine code than do Aho, Hopcroft, and Ullman [1] or Knuth [27]. We seek a model that allows nonlinear linkage but still has enough structure to support a theory comparable to that of finite transducers or context-free parsers. Such a model is proposed in the next section.

12. Tree Transducers and Compiling

In this section we sketch a model for syntax-directed compilation that allows the use of formal language theory without requiring unrealistic assumptions about the method of generating code from ranked parse trees. The model presupposes the existence of "deterministic finite tree transducers" similar to deterministic finite string transducers such that all states are final states. (The basic theory of such tree transducers will be presented in the next section.) This section concludes with remarks on the use of coroutine linkages to implement the five processing stages of our model without impractically large intermediate storage requirements.

We return to the problem of nonlinear linkage that arose in the previous section. Consider an ALGOL 60 conditional statement

IF $\mathscr{P}$ THEN $\mathscr{a}$ ELSE $\mathscr{B}$

where the character strings $\mathcal{P}$, $\mathcal{A}$, $\mathcal{B}$ determine machine language routines $\mathfrak{P}$, $\mathfrak{A}$, $\mathfrak{B}$. Suppose that executing $\mathfrak{P}$ would place the value of $\mathcal{P}$ in some register X whose contents can be tested by a branch instruction.

Two binary operations on machine language routines are of interest here. For all routines $\mathfrak{R}$ and $\mathfrak{S}$, let SEQ($\mathfrak{R}$,$\mathfrak{S}$) be the result of linking $\mathfrak{R}$ and $\mathfrak{S}$ sequentially, so that the exit from $\mathfrak{R}$ becomes the entry to $\mathfrak{S}$. For all routines $\mathfrak{R}$ and $\mathfrak{S}$, let TESTX($\mathfrak{R}$,$\mathfrak{S}$) be the result of merging the exits from $\mathfrak{R}$ and $\mathfrak{S}$ and forming a branch instruction that tests X and branches to $\mathfrak{R}$ on true and to $\mathfrak{S}$ on false. Thus TESTX($\mathfrak{A}$,$\mathfrak{B}$) is the routine displayed below the dotted line in Figure 5-6 while

(1) SEQ($\mathfrak{P}$, TESTX($\mathfrak{A}$, $\mathfrak{B}$))

is the entire routine displayed in Figure 5-6. In (1) we have a description of the code for IF $\mathcal{P}$ THEN $\mathcal{A}$ ELSE $\mathcal{B}$ in terms of given routines and elementary operations on code that could easily be programmed in assembly language, AMBIT/G [11] [20], or any other language at all suitable for compiler writing. The routines $\mathfrak{P}$, $\mathfrak{A}$, and $\mathfrak{B}$ may be quite complex, but they are also built up from smaller routines by basic <u>code-building operations.</u> The set Q of available operations is a ranked alphabet including SEQ and TESTX (both with rank two). The routines $\mathfrak{P}$, $\mathfrak{A}$, and $\mathfrak{B}$ can be described by trees $\hat{\mathfrak{P}}$, $\hat{\mathfrak{A}}$, and $\hat{\mathfrak{B}}$ in $Q_{\#}$. The code for IF $\mathcal{P}$ THEN $\mathcal{A}$ ELSE $\mathcal{B}$ in (1) can then be described by the tree

(2) $\overline{\mathrm{SEQ}}(\hat{\mathfrak{P}}, \overline{\mathrm{TESTX}}(\hat{\mathfrak{A}}, \hat{\mathfrak{B}}))$

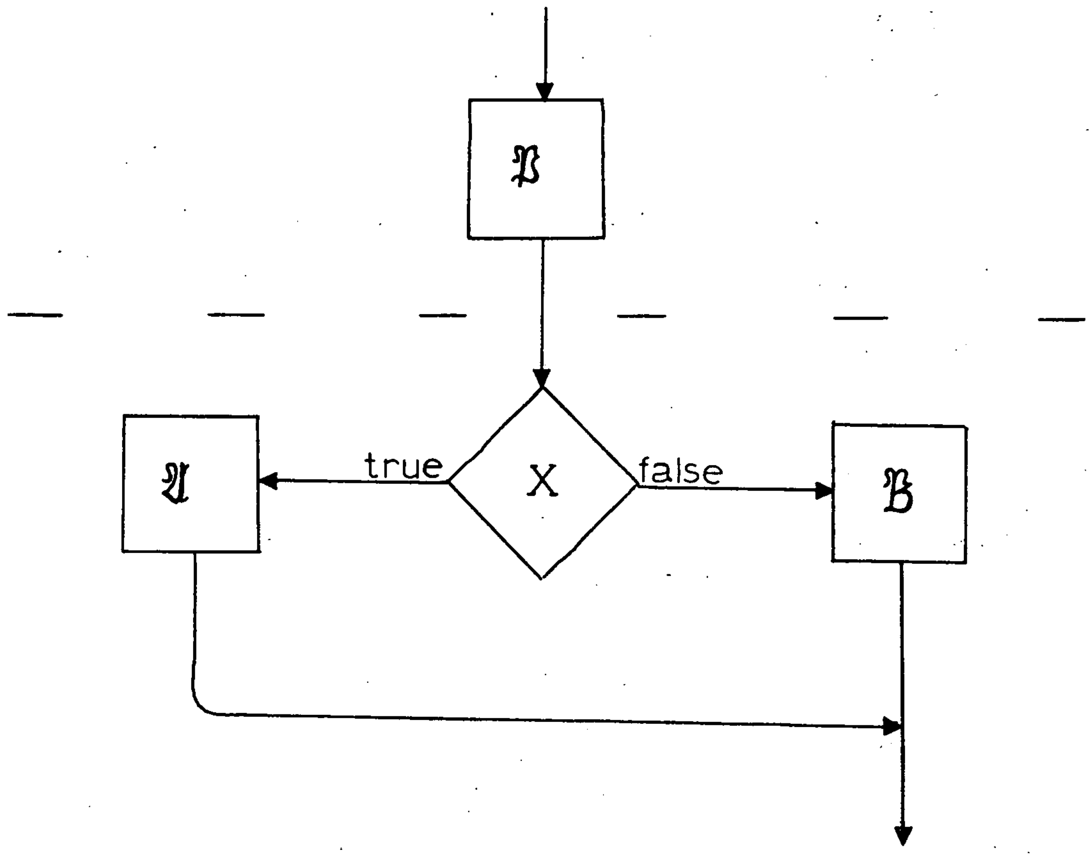


Figure 5-6. The conditional IF $\mathcal{P}$ THEN $\mathcal{A}$ ELSE $\mathcal{B}$ is coded as SEQ($\mathfrak{P}$,TESTX($\mathfrak{A}$,$\mathfrak{B}$)) when $\mathcal{P}$, $\mathcal{A}$, $\mathcal{B}$ are coded as $\mathfrak{P}$, $\mathfrak{A}$, $\mathfrak{B}$.

where $\overline{\text{SEQ}}$ and $\overline{\text{TESTX}}$ are the operations on $Q_{\#}$ defined by (4.10).

By hypothesis, Q consists of names for readily programmed operations on machine code. To pass from the tree (2) to the routine (1) is simply to perform the indicated operations. More generally, we may assume that there is a code generator which converts ranked trees to machine code. (The actual input to the code generator will, of course, be an augmented tree, with pointers attached to the leaves as in the previous section.) The code generator may perform ordinary bottom-up evaluations as in (1). It may also perform operations with side effects on an environment that includes the symbol table, which must eventually be expanded to include addresses as well as spellings. We will not attempt to model code generation in detail. In each specific application Q is to be a selection from operations that are already well understood and hence not in urgent need of a mathematical model.

In order to avoid dubious assumptions about the mathematical structure of machine code, we have postulated a code generator that maps trees to machine language routines. The input to the code generator is a coding tree: a ranked tree augmented by pointers to symbol table entries as well as ordinary labels at the leaves. The ranked alphabet Q represents whatever basic code-building operations are available; it has no simple relation to the ranked alphabet P defined by the context-free grammar $G = (N, T, P, X)$ used in generating the source language. (Recall that ranks are assigned to productions by Definition 11.7.)

There is a gap in the method of compiler construction we have developed. Lexical analysis, context-free parsing, and lexical filtration map a character string to an augmented ranked parse tree, as we indicated in the previous section. Code generation maps a coding tree to machine code. A fifth process is needed to map augmented ranked parse trees to coding trees. We will call this process semantic analysis.

The semantic analysis of an augmented parse tree is to be determined mainly by the corresponding ranked parse tree without the pointers: the source language is almost context-free semantically as well as syntactically. We will assume that the semantic analyzer can map trees labelled by productions to trees labelled by code-building operations with the help of a "deterministic finite tree transducer" analogous to the type of string transducer defined in (11.5). The semantic analyzer ignores but does not erase the pointers on leaves of the augmented ranked parse tree. The pointers are simply carried along so as to appear where needed in the coding tree.

The practical significance of lexical analysis as a first stage in compilation is enhanced by the existence of a fruitful mathematical model for string transduction. There is a similarly fruitful model for tree transduction in semantic analysis, but the greater complexity of trees as opposed to strings forces us to postpone formalization until the next section. For the moment, it will suffice to anticipate that tree transducers will be generalizations of string transducers and that they can perform the macroexpansions for "format routine" macroinstructions [43, §§1, 4, 6].

Our model for compilation is summarized in Figure 5-7, where the boxes represent processes and the arrows represent data flow. As the two-headed arrow indicates, the lexical filter might modify the

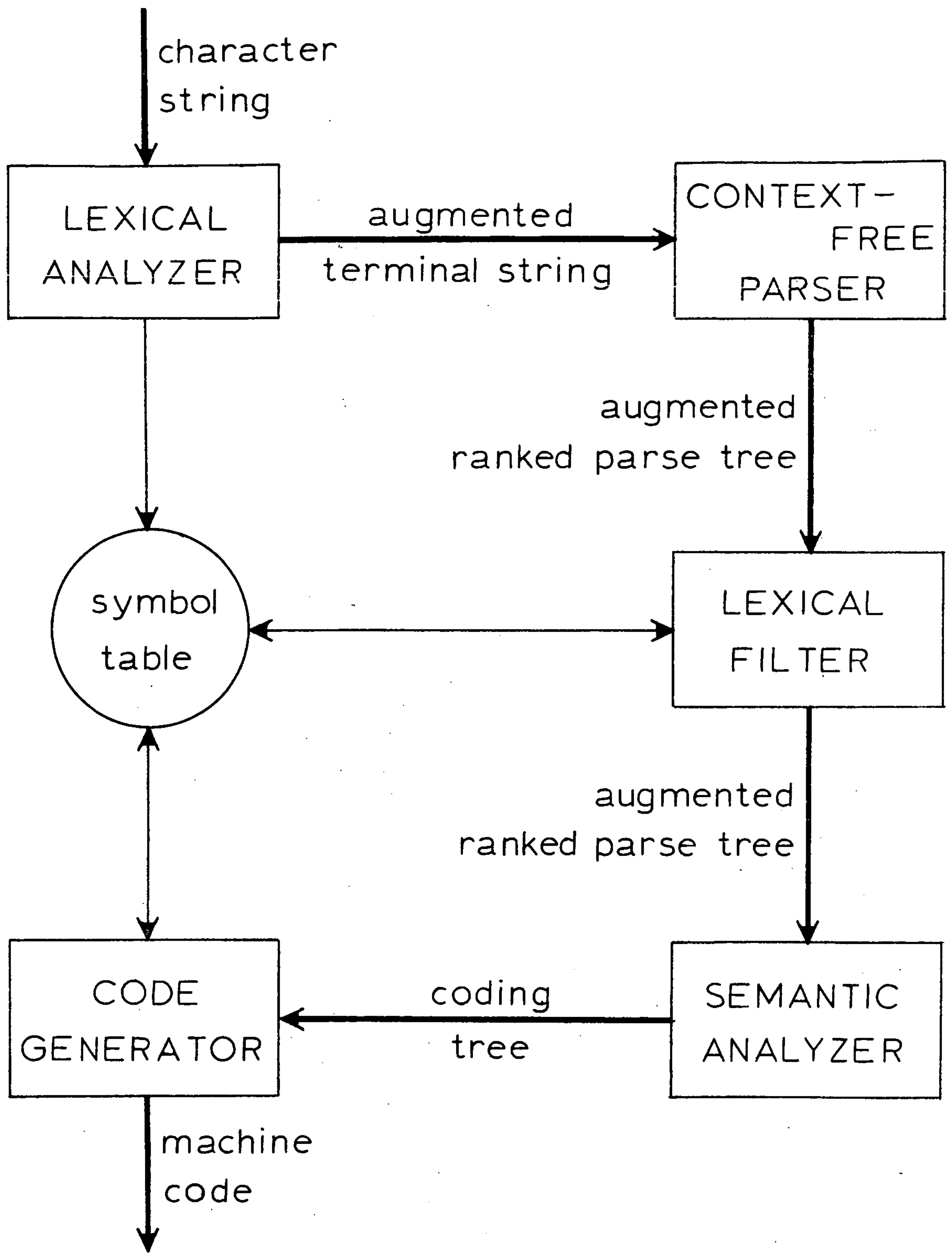


Figure 5-7. A model for syntax-directed compilation.

symbol table as well as consult it. For example, spelling of identifiers might be supplemented by pairs (j, k) where j indicates the block where the identifier is declared and k indicates where the identifier falls in a list of that block's identifiers. The code generator might also modify the symbol table, perhaps by adding addresses that will be needed later in the generative process.

This view of syntax-directed compiling is the natural result of combining two traditions in computer science. Formal language theory (if understood in a broad sense that includes the recent introduction of tree transducers) provides the mathematical background. More concrete discussions of compiler design problems [7] [43] [54] provide the general shape of Figure 5-7 and the specific practical techniques for table searching, linking routines, and so on that we take for granted here. Without proceeding to a complete mathematical formalization, we have separated some of the general ideas stated or implied in these discussions from each other and from considerations of more specific problems.

In contrast with the attempt to model the entire compiler formally by a single abstract automaton as proposed by Lewis and Stearns [30, p. 467], we have attempted to divide and conquer in Figure 5-7. We consider syntax-directed compilation to be a complex task; different formal models are appropriate to different aspects of the task. As we suggested in the previous section, deterministic finite string transducers are appropriate for lexical analysis, several algorithms in the formal language theory literature are appropriate for context-free parsing, and lexical filtration is a fairly straightforward inspection of structures already available. We claim that deterministic finite tree transducers are appropriate for semantic analysis, and some evidence will be presented in the next two sections. By definition, the coding tree can be readily transformed to code by the code generator.

Dividing the task of compilation into several stages has two other advantages regardless of the availability of theoretical models. The algorithms to be used at each stage can be specified and implemented separately, and this will facilitate debugging and documentation. Another advantage is that transportability is enhanced: a compiler for a language on one target machine can be converted to a compiler for the same language on another machine with a minimum of reprogramming. The code generator may need extensive changes, but the other stages should be useable with hardly any changes if written in a higher level compiler writing language such as AMBIT/G [11] [20].

Any model of compilation that involves several stages faces a severe practical difficulty when large programs are considered: there may not be enough primary storage to hold the entire ranked parse tree or coding tree. Even the augmented terminal string may be too large in some situations. Fortunately, there is a programming technique that often permits us to have the advantages of a multistage process without excessive intermediate storage requirements. We conclude this section with a discussion of this "coroutine linkage" [28, §1.4.2] technique and the constraints it imposes on the separate stages.

Suppose that the five stages in Figure 5-7 have been implemented by programs LA, CFP, LF, SA, and CG in an appropriate language, so that the compiler is the sequentially linked program

(1) LA; CFP; LF; SA; CG

with burdensome intermediate storage requirements. We modify the five programs slightly to form <u>coroutines</u> LA′, CFG′, LF′, SA′, and CG′ that can transfer control back and forth as shown below:

(2) $LA' \rightleftarrows CFP' \rightleftarrows LF' \rightleftarrows SA' \rightleftarrows CG'$.

Whenever control passes from one coroutine C to another coroutine D, C pauses until control returns from D. Then C continues from whatever place in C's control sequence transferred to D. Except for this linkage mechanism, LA′ is just like LA, CFP′ is just like CFP, and so on.

The general idea of coroutine linkage is applied here in order to save intermediate storage. The rules by which the coroutines call on each other are simple:

(a) Each of LA′, CFP′, LF′, and SA′ computes only as long as is necessary to determine some further portion of the string or tree that it should output. It then passes control and the new portion of its output to its successor in the sequence (2).

(b) Whenever one of CFP′, LF′, SA′, and CG′ must examine some further portion of its input in order to continue, it passes control to its predecessor in the sequence (2).

(c) Whenever LA′ must examine some further portion of its input in order to continue, it reads in more characters from the input buffer for the whole system.

This is the mode of operation used by Cheatham [7] and Wirth and Weber [54]. If transfers of type (a) are fairly frequent compared to transfers of types (b) or (c), then only fairly small portions of the augmented terminal string, augmented ranked parse tree, and coding tree must be retained in primary storage at any one time. The time saved by avoiding secondary storage may well be more than enough to compensate for the time spent in transferring control back and forth among the coroutines.

Coroutine linkage is only useful here if the original routines LA, . . . , CG are such that transfers of type (a) are fairly frequent compared to transfers of types (b) or (c). Since LA can only produce a prefix of the augmented terminal string after reading a prefix of the character string, CFP should read its input from left to right, remembering only what it might need to know later about previous portions of the terminal string. Furthermore, CFP should be able to output part of the augmented ranked parse tree after reading only part of the augmented terminal string. If CFP builds the tree from the top down, then LF and SA should read it from the top down, and so on. The intended use of coroutine linkages puts constraints on the design of LA, . . . , CG. In particular, SA should read the augmented ranked parse tree in the order that CFP builds it and should output the coding tree in the order that CG reads it. Once the compiler designer has chosen the order in which some chains of events occur, he must follow suit with other chains or risk exhausting the available primary storage. As we will see in §14, this practical consideration leads to many open questions in the theory of tree transducers.

13. Finite Tree Transducers

In this section we define several types of tree transducers in terms of subtree replacement systems. By expressing each type of transducer as a special case of a general model, we hope to facilitate the development of a unified theory with applications to tree transduction in linguistics as well as compiler design. The actual results presented here are only very modest extensions of results of Rounds [45] [46] and Thatcher [51] [52]. In addition to establishing some basic properties of finite tree transducers, we indicate the significance of these properties for compiler construction.

First we set up some notation that will be used throughout this section and the next.

(13.1) Notation. Let $\mathfrak{M}$ and U be disjoint infinite sets. Members of $\mathfrak{M}$ are markers; members of U are parameters. The letters r, s and the symbols @, \$, ¢ are used as variables ranging over $\mathfrak{M}$. The parameters are listed in a sequence $u_0, u_1, u_2, \ldots$ and subscripted u's always refer to this sequence.

Parameters will be used to form rule-schemata, as in §6. Markers carry two kinds of information. Intuitively, if a tree R has a node n with $Rn \in \mathfrak{M}$, then a tape head of an automaton is positioned at node n and can read the labels on n and some nearby nodes. (Imagine an automaton for strings whose heads can read all of several adjacent tape squares at once.) In addition to this information about a tape head's position, a node labelled in $\mathfrak{M}$ encodes a partial record of previous computations in the specific marker chosen. A member of $\mathcal{M}$ acts like a state of an automaton's control unit, but here there is a separate control unit for each head and the units are only partially synchronized. Another anomaly is that tape heads will often split into several heads, especially when moving down from a node with more than one son. They will coalesce into one when moving back up. Sometimes tape heads will even vanish. Such activities are extremely awkward to formalize in the usual style of describing automata (as in [24]), but they can be obtained easily from an SRS. After two definitions we will be able to give examples.

Our transducers will manipulate trees whose nodes are labelled by markers, parameters, and the actual labels occurring on nodes of input and output trees. (To avoid confusion, these labels should be distinct from the markers and parameters.) We will deal with ranked input and output trees because this is technically convenient and because the augmented ranked parse trees and coding trees considered in the previous section actually are ranked trees. It will be convenient to assume that any symbols shared by the input and output vocabularies have the same ranks in both vocabularies.

(13.2) <u>Definition</u>. Let Σ and Δ be finite sets with rank functions $\sigma : \Sigma \longrightarrow \mathbb{N}$ and $\delta : \Delta \longrightarrow \mathbb{N}$. Then (Σ, Δ) is a <u>proper pair</u> of ranked alphabets iff

(1) $$(\Sigma \cup \Delta) \cap (\mathcal{M} \cup U) = \emptyset$$

and

(2) $$(\forall a \in \Sigma \cap \Delta)(\sigma(a) = \delta(a)).$$

If (Σ, Δ) is proper, then the transducer vocabulary

(3) $\quad V = \Sigma \cup \Delta \cup \mathfrak{m} \cup U$

is assigned the rank function

(4) $\quad \rho = \sigma \cup \delta \cup \{\langle r, 1\rangle \mid r \in \mathfrak{m}\} \cup \{\langle w, 0\rangle \mid w \in U\}$.

For the rest of this chapter, the letters V and ρ will be used as in (3) and (4) above. The relevant proper pair (Σ, Δ) will be clear in context, as in the following definition of SRS transducers. We omit the overlines in the algebraic nomenclature defined in (4.10).

(13.3) Definition. Let Π be any 4-tuple $(\Sigma, \Delta, @, \mathfrak{T})$, where $@ \in \mathfrak{m}$, (Σ, Δ) is a proper pair of ranked alphabets, and $\mathfrak{T}$ is an SRS of the form $(V, \mathbb{F}, \Longrightarrow, \mathbb{R})$ with

(1) $\quad \mathbb{F} = \{S \in V_{\#} \mid (\exists R \in (\Sigma \cup U)_{\#})(@(R) \overset{*}{\Longrightarrow} S)\}$.

Then Π is an SRS transducer with input vocabulary Σ, output vocabulary Δ, and root marker @. The map $\langle \Pi \rangle$ from $2^{\Sigma_{\#}}$ into $2^{\Delta_{\#}}$ defined by

(2) $\quad \langle \Pi \rangle \mathcal{R} = \{T \in \Delta_{\#} \mid (\exists R \in \mathcal{R})(@(R) \overset{*}{\Longrightarrow} T)\}$

for all $\mathcal{R} \subseteq \Sigma_{\#}$ is the transduction specified by Π.

The root marker @ is analogous to the left endmarker $\vdash$ in work with strings. We do not need an analog for the right endmarker $\dashv$ because the subtree of a tree at a node already includes everything at or below the node.

The condition (1) in Definition 13.3 means that $\mathfrak{T}$ is determined by $\mathbb{R}$ alone, since the transducer vocabulary was already fixed as

$\Sigma \cup \Delta \cup \mathfrak{m} \cup U$ in (13.2.3). The forest $\mathbb{F}$ consists of all trees derivable in $\mathfrak{C}$ from trees of the form @(R) where @ is the root marker and R is labelled by input symbols and parameters.

By using $2^{\Sigma}\#$ and $2^{\Delta}\#$ in (2) above, we made the transduction $\langle \Pi \rangle$ a total function even though the relation

$$(*) \qquad \{\langle R, S\rangle \in \Sigma_{\#} \times \Delta_{\#} \mid @(R) \overset{*}{\Longrightarrow} S\}$$

may be neither total nor singlevalued. Conditions under which (*) is a total function will be considered later. Theorem 13.16 asserts that $\langle \Pi \rangle$ is a total function if Π is a "deterministic finite tree transducer."

For an example we return to the grammar $G_0 = (N, T, P, e)$ from §11 for forms of arithmetic expressions. We have $N = \{e, t, f\}$ and $T = \{(,), +, \times, a\}$. The productions are

$$\pi_0 = e \longrightarrow t \qquad \text{with } \rho(\pi_0) = 1$$

$$\pi_1 = e \longrightarrow e+t \qquad \text{with } \rho(\pi_1) = 2$$

$$\pi_2 = t \longrightarrow f \qquad \text{with } \rho(\pi_2) = 1$$

$$\pi_3 = t \longrightarrow t \times f \qquad \text{with } \rho(\pi_3) = 2$$

$$\pi_4 = f \longrightarrow a \qquad \text{with } \rho(\pi_4) = 0$$

$$\pi_5 = f \longrightarrow (e) \qquad \text{with } \rho(\pi_5) = 1 .$$

The ranked parse tree shown in Figure 5-8 conveys the same information about $a+(a+a)\times a$ as the operator-operand structure shown in Figure 5-9. We can map the tree in Figure 5-8 to the tree in Figure 5-9 by means of an SRS transducer with input vocabulary $\Sigma = P$ and output

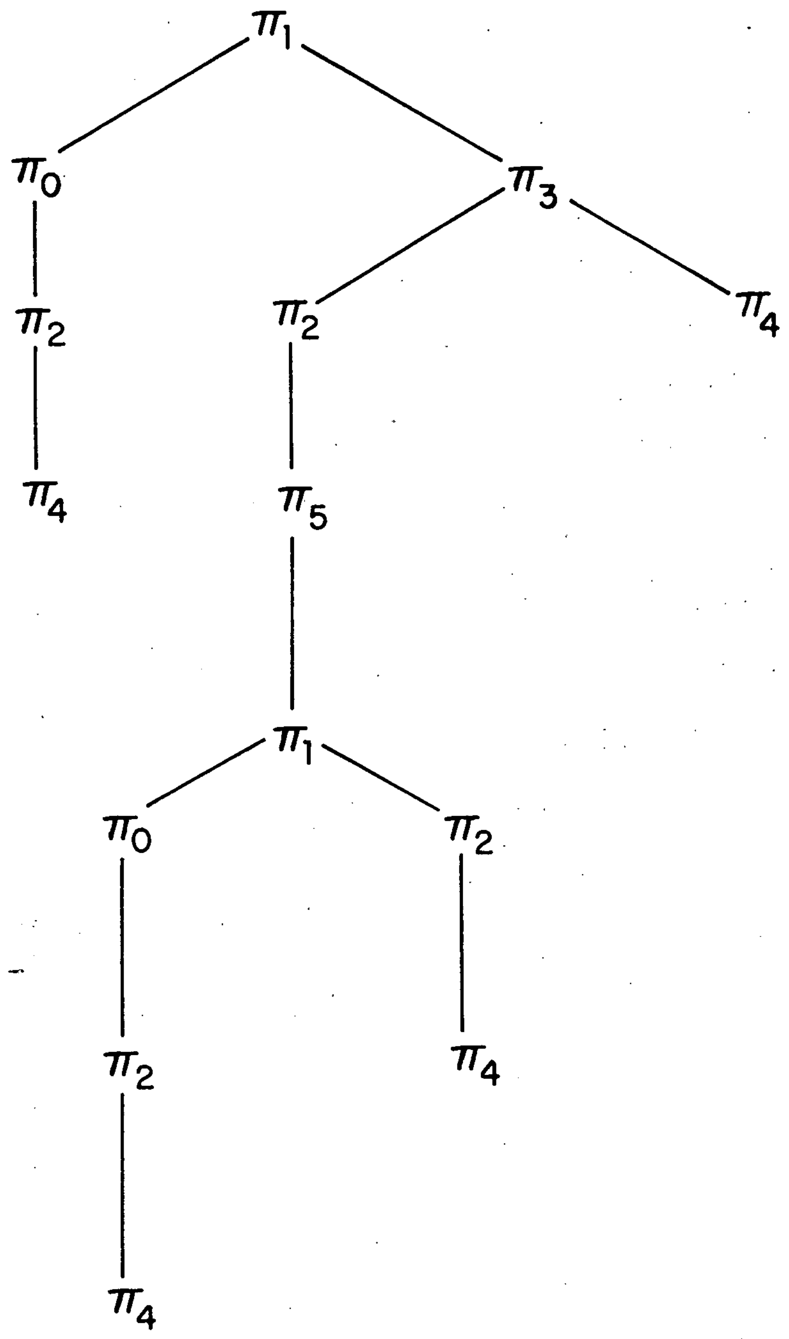


Figure 5-8. Ranked parse tree for a+(a+a)Xa.

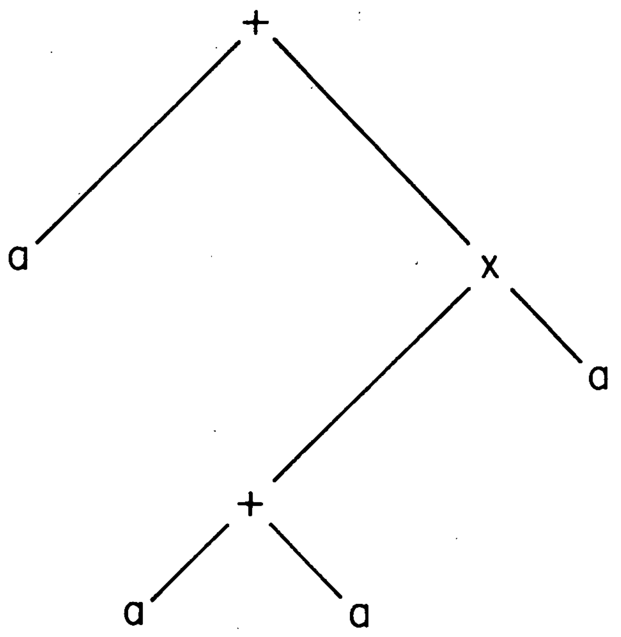


Figure 5-9. Operator-operand structure for a+(a+a)Xa.

vocabulary $\Delta = \{+, \times, a\}$. We choose any $@ \in \mathfrak{M}$ as the root marker and consider the set L of pairs of trees shown in Figure 5-10. Let each $w \in U$ be assigned the domain

$$D_w = (\Sigma \cup U)_{\#} \subseteq V_{\#},$$

so that L is a set of rule-schemata in the sense of Definition 6.1.

Let $\mathbb{R}_L$ be the set of all instances of members of L, as in Definition 6.2. Then there is a unique SRS

$$\mathfrak{C} = (V, \mathbb{F}, \Longrightarrow, \mathbb{R}_L)$$

such that

$$\mathbb{F} = \{T \in V_{\#} \mid (\exists R \in (\Sigma \cup U)_{\#})\, @(R) \overset{*}{\Longrightarrow} T)\}.$$

Thus we have specified an SRS transducer

$$\Pi_0 = (\Sigma, \Delta, @, \mathfrak{C}).$$

If R is the tree in Figure 5-8 and S is the tree in Figure 5-9, then it is easy to verify that

$$\langle \Pi_0 \rangle \{R\} = \{S\}.$$

Once Σ, Δ, and @ have been specified, an SRS transducer may always be defined by assigning domains to the parameters and defining a set L of rule-schemata in the sense of Definition 6.1. The rules of the transducer's SRS are the instances of members of L (in the sense of Definition 6.2).

(13.4) Definition. A (finite) presentation of an SRS transducer $\Pi = (\Sigma, \Delta, @, \mathfrak{C})$, with $\mathfrak{C} = (V, \mathbb{F}, \Longrightarrow, \mathbb{R})$, is any pair $\langle \mathbb{D}, L \rangle$ such that

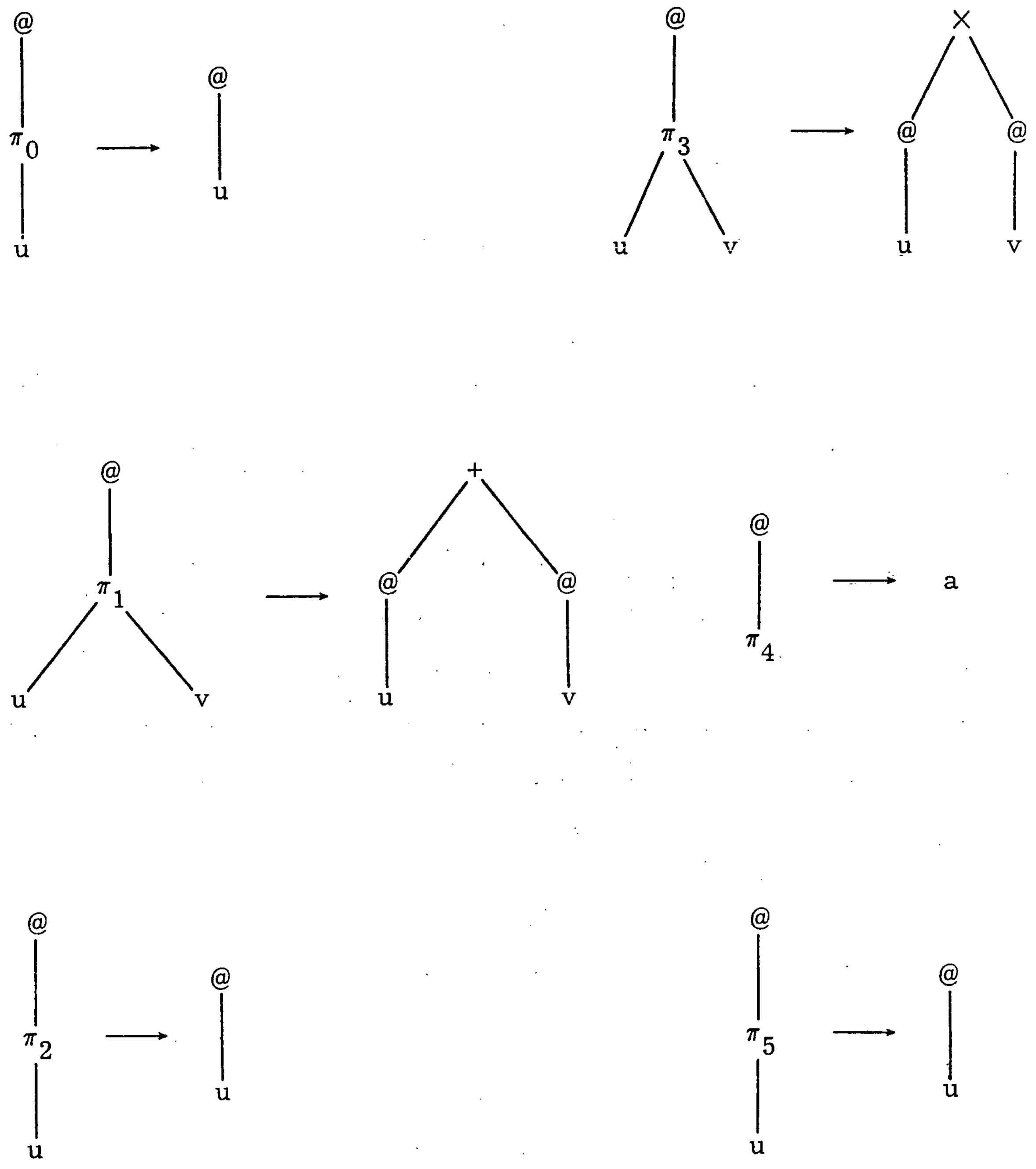


Figure 5-10. Rule-schemata for an SRS transducer. The parameters u_0 and u_1 are written as u and v for legibility.

$\mathbb{D} \subseteq V_{\#}$ and L is a (finite) set of rule schemata such that $\mathbb{R} = \mathbb{R}_L$ when each $w \in U$ is assigned the domain $D_w = \mathbb{D}$.

We will be concerned with finite presentations using very simple domains:

$$\mathbb{D} = (\Sigma \cup U)_{\#} \quad \text{or} \quad \mathbb{D} = (\Delta \cup U)_{\#}.$$

A finitely presented SRS transducer is still only finite in a very weak sense: it takes only finitely much information to specify exactly which transducer we are considering. (Note that Turing machines are finite in this sense.) The definitions given so far allow markers to move up or down as freely as a Turing machine's heads move left or right. We wish to restrict such motions so as to obtain the stronger finiteness associated with "finite automaton" and similar phrases. By requiring that tape heads always move downward or always move upward, we will obtain the desired strong finiteness properties in Lemma 13.13. Downward motion will be considered first.

Our transducer Π_0 that maps ranked parse trees to operator-operand structures is defined by a set L of rule-schemata in Figure 5-10. Imagining that nodes labelled by markers are positions of tape heads, we consider how tape head motion is restricted by the forms of the rule-schemata. For any $\Phi \longrightarrow \Psi \in L$, Φ has the form

$$s(a(u_0, \ldots, u_{-1}))$$

where $s \in \mathfrak{M}$ and $a \in \Sigma$, so that the tape head reads only the label immediately below it. The list $(u_0, \ldots, u_{-1})$ of the first $\rho(a)$ parameters indicates that the trees $T_0, \ldots, T_{-1}$ in any instance $\varphi \longrightarrow \psi$

(1) $s(a(T_0, \ldots, T_{-1})) \longrightarrow \Psi(\Psi^{-1}(u_0) \longleftarrow T_0) \ldots (\Psi^{-1}(u_{-1}) \longleftarrow T_{-1})$

of $\Phi \longrightarrow \Psi$ will vanish or be carried over to ψ without change. Furthermore, the tape head will move downward (perhaps splitting into several heads) so as to be able to read the root labels of $T_0, \ldots, T_{-1}$ next. The tree Ψ is ranked and is labelled by output symbols, markers, and parameters. Each node labelled by a marker has its one son labelled by a parameter u_j: the tape head will move down to T_j in (1). Each node labelled by a parameter is the one son of a node labelled by a marker: no T_j can be simply written out having a tape head pass over it. The output symbols occur above the tape heads and are not processed further.

The schemata in Figure 5-10 satisfy many other interesting restrictions, but the ones already mentioned suffice to make the tape heads move down from the root in trees @(R) with $R \in (\Sigma \cup U)_\#$. The downward moving tape heads may change markers (although not in this example) and may split into several heads at each node read. The heads will always have output symbols above them and input symbols or parameters below them.

(13.5) Definition. An SRS transducer $\Pi = (\Sigma, \Delta, @, \mathfrak{C})$ is a top-down finite tree transducer (TDFTT) iff there is a finite presentation $\langle (\Sigma \cup U)_\#, L \rangle$ for Π such that

(1) $(\forall \Phi \longrightarrow \Psi \in L)(\exists s \in \mathcal{M})(\exists a \in \Sigma)[\Phi = s(a(u_0, \ldots, u_{\rho(a)-1}))]$

(2) $(\forall \Phi \longrightarrow \Psi \in L)[\Psi \in (\Delta \cup \mathcal{M} \cup U)_\# \;\&\; \Psi^{-1}(\mathcal{M}) \cdot (0) = \Psi^{-1}(U)]$.

(13.6) <u>Lemma.</u> Let $\Pi = (\Sigma, \Delta, @, \mathfrak{C})$ be a TDFTT. There is a unique set L of rule-schemata such that $\langle (\Sigma \cup U)_{\#}, L \rangle$ is a presentation for Π satisfying conditions (1) and (2) in Definition 13.5.

<u>Proof:</u> Let $\mathcal{L}$ be the class of all sets L of rule-schemata such that $\langle (\Sigma \cup U)_{\#}, L \rangle$ is a presentation for Π satisfying (13.5.1) and (13.5.2). We must show that $\mathcal{L}$ is a singleton: $(\exists L)(\mathcal{L} = \{L\})$.

Suppose $L, L' \in \mathcal{L}$ and $\Phi \longrightarrow \Psi \in L$. We will show that $\Phi \longrightarrow \Psi \in L'$. Note that $\Phi \longrightarrow \Psi$ is an instance of itself, so that $\Phi \longrightarrow \Psi \in \mathbb{R}_L$. But $\mathbb{R}_L = \mathbb{R}_{L'}$, because $\langle (\Sigma \cup U)_{\#}, L \rangle$ and $\langle (\Sigma \cup U)_{\#}, L' \rangle$ are both presentations for Π, so $\Phi \longrightarrow \Psi$ is an instance of some $\Phi' \longrightarrow \Psi' \in L$. For $s = \Phi()$ and $a = \Phi(0)$, (13.5.1) for L and L′ implies that

$$\Phi = s(a(u_0, \ldots, u_{-1})) = \Phi' .$$

Let $N_k = \Psi'^{-1}(u_k)$ for each $k < \rho(a)$. Then

$$\Psi = \Psi'(N_0 \longleftarrow u_0) \ldots (N_{-1} \longleftarrow u_{-1}) = \Psi' .$$

Therefore $\Phi \longrightarrow \Psi = \Phi' \longrightarrow \Psi'$ and $\Phi \longrightarrow \Psi \in L'$.

We have shown that $L \subseteq L'$ whenever $L, L' \in \mathcal{L}$. Therefore $\mathcal{L}$ is empty or a singleton. But $\mathcal{L} \neq \emptyset$ because Π is a TDFTT, so $\mathcal{L}$ is a singleton. ∎

The earlier definitions by Rounds and Thatcher will not be reproduced here, but we note that transducers equivalent to TDFTTs have been called "nondeterministic finite state transformations" and abbreviated "NDFST" [46, §II.1] or "NFST" [52, §8]. They have also been called "nondeterministic root-to-frontier automata with output" and abbreviated "nondeterministic RFAO" [51, Def. 7′, p. 272]. The next paragraph explains why we introduced "TDFTT" rather than choose among the names previously used.

"Transformation" could lead to confusion between the transduc<u>er</u> and the transduc<u>tion</u>, while "automaton with output" is awkward and suggests more similarity to ordinary (string) automata (as in [24]) than is actually present. The word "tree" should be included because string transducers and tree transducers must be discussed together but not confused under the model for syntax-directed compilation sketched in §12. Hence we say "tree transducer." Finiteness is important while the word "state" after "finite" would add nothing, so we say "finite tree transducer." Some indication of how the markers move is necessary because other directions of movement will be considered shortly. "Top-down" is more common (and more graceful) than "root-to-frontier," so the result is "top-down finite tree transducer" as a concise and natural way to name the class of transducers in Definition 13.5 without inviting confusion later.

As we remarked at the end of §12, the tree transducer in a semantic analyzer should read the ranked parse tree in the same direction that the parsing algorithm builds it. The TDFTTs are appropriate when a top-down parsing algorithm is used, but several major parsing algorithms are bottom-up, as in [15] or [19]. We therefore wish to define tree transducers that move markers upward from the leaves toward the root.

For an example, we return to the grammar $G_0 = (N, T, P, e)$ for forms of arithmetic expressions. We have $N = \{e, t, f\}$ and

$T = \{(,), +, \times, a\}$. The productions are

$$\begin{array}{lll}
\pi_0 = e \longrightarrow t & & \text{with } \rho(\pi_0) = 1 \\
\pi_1 = e \longrightarrow e+t & & \text{with } \rho(\pi_1) = 2 \\
\pi_2 = t \longrightarrow f & & \text{with } \rho(\pi_2) = 1 \\
\pi_3 = t \longrightarrow t\times f & & \text{with } \rho(\pi_3) = 2 \\
\pi_4 = f \longrightarrow a & & \text{with } \rho(\pi_4) = 0 \\
\pi_5 = f \longrightarrow (e) & & \text{with } \rho(\pi_5) = 1 .
\end{array}$$

We can map the ranked parse tree for $a+(a+a)\times a$ shown in Figure 5-11 to the operator-operand structure shown in Figure 5-12, using an SRS transducer that moves markers upward.

Let the input vocabulary Σ be P and let the output vocabulary Δ be $\{+, \times, a\}$. We choose any $@ \in \mathfrak{M}$ as the root marker and any $\$ \in \mathfrak{M}$ with $\$ \neq @$ as a marker to be moved upward. Consider the set L of pairs of trees shown in Figure 5-13. Letting $\mathbb{D} = (\Delta \cup \mathrm{U})_{\#}$, we have a finite presentation $\langle \mathbb{D}, \mathrm{L} \rangle$ for an SRS transducer $\Pi_0 = (\Sigma, \Delta, @, \mathfrak{C})$ with $\mathfrak{C}$ of the form $(\mathrm{V}, \mathbb{F}, \Longrightarrow, \mathbb{R}_{\mathrm{L}})$. (Recall the definition of presentations (13.4).) This transducer meets the requirements of the following definition.

(13.7) <u>Definition</u>. An SRS transducer $\Pi = (\Sigma, \Delta, @, \mathfrak{C})$ is a <u>bottom-up finite tree transducer</u> (<u>BUFTT</u>) iff $\Sigma \cap \Delta = \emptyset$ and there is a finite presentation $\langle (\Delta \cup \mathrm{U})_{\#}, \mathrm{L} \rangle$ for Π such that $\mathrm{L} \neq \emptyset$ and

(1) $(\forall \Phi \longrightarrow \Psi \in \mathrm{L})(\exists a \in \Sigma \cup \{@\})(\exists s \in (\mathfrak{M} - \{@\})^{\rho(a)})$

$$[\Phi = a(s_0(u_0), \ldots, s_{-1}(u_{-1}))]$$

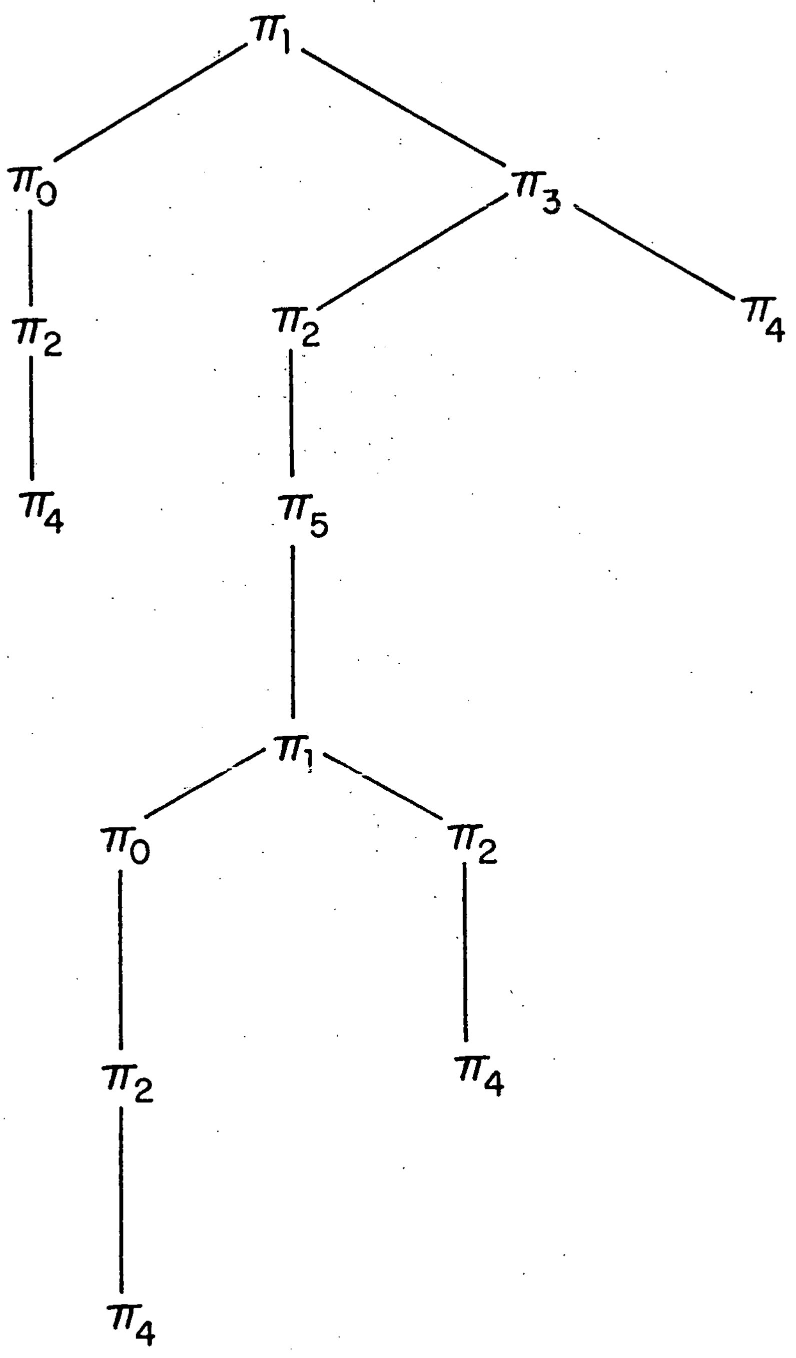


Figure 5-11. Ranked parse tree for a+(a+a)Xa.

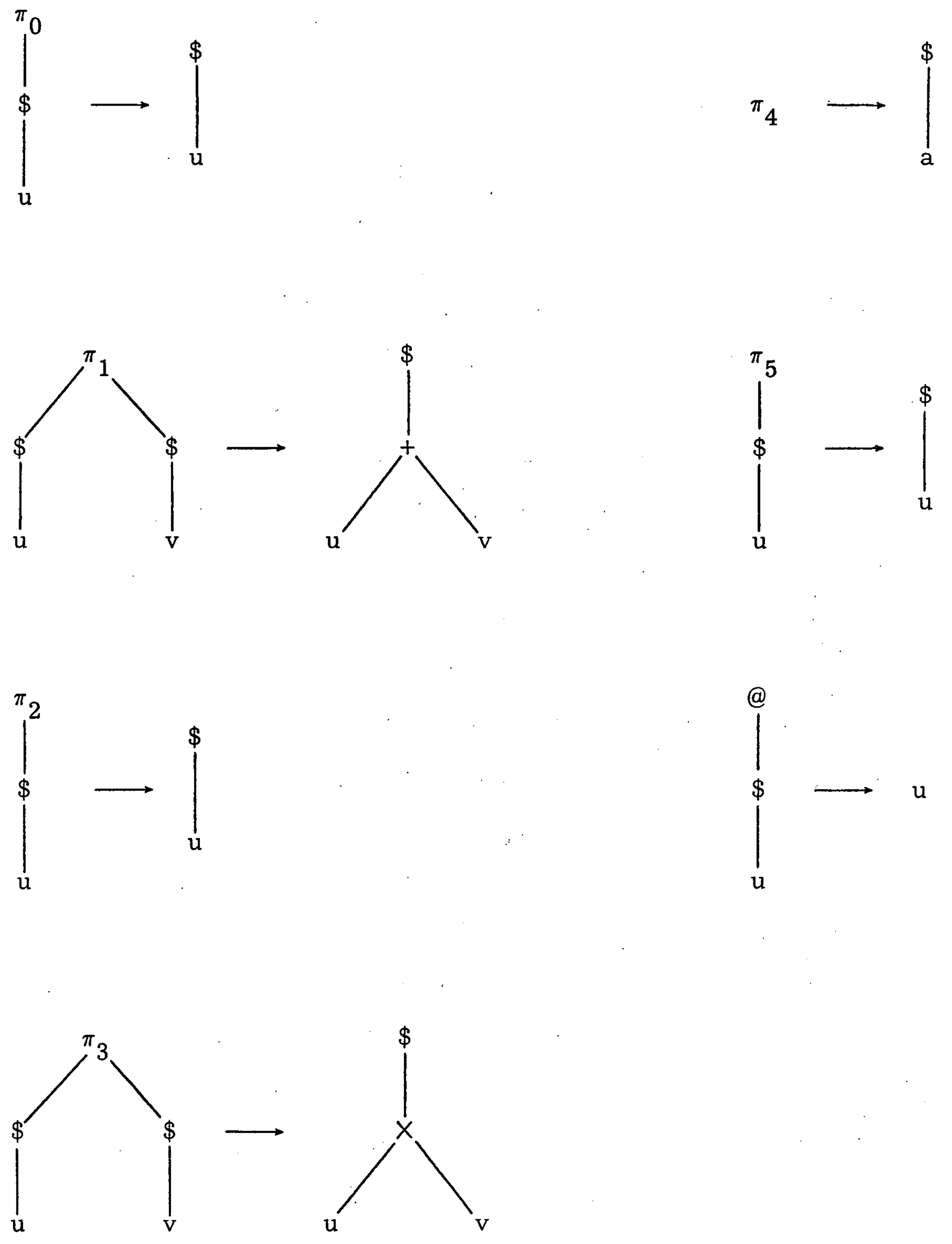


Figure 5-13. Rule-schemata for an SRS transducer. The parameters u_0 and u_1 are written as u and v for legibility.

(2) $(\forall \Phi \longrightarrow \Psi \in L)(\Phi(\,) \in \Sigma$ implies

$$[\Psi(\,) \in \mathfrak{M} - \{@\} \;\&\; \Psi/(0) \in (\Delta \cup U)_{\#}])$$

(3) $(\forall \Phi \longrightarrow \Psi \in L)(\Phi(\,) = @$ implies $\Psi = \{\langle (\,), u_0 \rangle\})$.

The rootmarker is only used to mark the root here, while other markers may be interpreted as states for tape heads that move upward. The requirements $\Sigma \cap \Delta = \emptyset$ and $L \neq \emptyset$ are technical conveniences. Condition (1) says that a head can only move upward from a node n if there are heads at all the brothers of n also. The label $a \in \Sigma \cup \{@\}$ on Fa(n) and the actual string $s \in (\mathfrak{M} - \{@\})^{\rho(a)}$ of markers involved are scanned in choosing the upward motion. As the use of parameters indicates, the trees in $(\Delta \cup U)_{\#}$ below the tape heads must vanish or be carried along without being processed further. Conditions (2) and (3) specify just how the tape heads can coalesce and move upward. If $\Phi(\,) \in \Sigma$ then Ψ has the form $r(\Psi/(0))$ where $r \in \mathfrak{M} - \{@\}$: the heads have coalesced to a single head in the state r. The subtree $\Psi/(0)$ specifies the output produced at this stage and where the previous output trees are to be attached. Thus $\Psi/(0) \in (\Delta \cup U)_{\#}$. On the other hand, if $\Phi(\,) = @$, then Φ has the form $@(s(u_0))$ with $s \in \mathfrak{M} - \{@\}$. The tape heads have travelled all the way up to the root and whatever is below the one remaining head is the complete output tree. Thus Ψ consists of one node labelled by u_0: the markers vanish and the complete output tree is left.

Reasoning just as in the proof of Lemma 13.6, we can demonstrate the following fact.

(13.8) Lemma. Let $\Pi = (\Sigma, \Delta, @, \mathfrak{C})$ be a BUFTT. Then there is a unique nonempty set L of rule-schemata such that $\langle (\Delta \cup U)_{\#}, L \rangle$ is a presentation for Π satisfying conditions (1)--(3) in Definition 13.7. ∎

Transducers equivalent to BUFTTs have been called "nondeterministic frontier-to-root automata with output" and abbreviated "nondeterministic FRAO" [51, Def. 8, p. 270]. The reasons for our nomenclature are similar to the reasons already advanced for "TDFTT."

An SRS transducer that is either a TDFTT or a BUFTT is a finite tree transducer (FTT). A map form $2^{\Sigma_\#}$ to $2^{\Delta_\#}$ expressible as $\langle \Pi \rangle$ for some transducer Π is a transduction of the same kind: top-down finite tree transduction, bottom-up finite tree transduction, and so on.

In general an SRS transducer may have many presentations, but an FTT has just one natural presentation $\langle \mathbb{D}, L \rangle$ which exhibits the fact that the transducer is an FTT by satisfying conditions (1) and (2) in Definitions 13.5 or conditions (1) -- (3) in Definition 13.7. Indeed, suppose Π is an FTT with a set $\mathbb{R}$ of rules. If Π is top-down, then each $\varphi \longrightarrow \psi \in \mathbb{R}$ has $\varphi(0) \in \Sigma$. If Π is bottom-up, then $\mathbb{R} \neq \emptyset$ and each $\varphi \longrightarrow \psi \in \mathbb{R}$ has $\varphi(0) \in \mathfrak{M}$ if $(0) \in \operatorname{Dom} \varphi$. Therefore Π cannot be both top-down and bottom-up. The uniqueness of $\langle \mathbb{D}, L \rangle$ now follows from Lemmas 13.6 and 13.8.

The definitions of TDFTTs and BUFTTs given by Rounds [45] [46] and Thatcher [51] [52] are essentially definitions of the natural presentations of such transducers. Repeated ad hoc for each class of transducers are definitions equivalent to our definitions of instances of rule-schemata (6.2.1), of $\Longrightarrow$ in an SRS (5.1.3), and of transductions (13.3.2).

An important property that FTTs may have is linearity [46, p. 272]. Linear transducers do not make extra copies of portions of their inputs or outputs.

(13.9) Definition. Let Π be an FTT with natural presentation $\langle \mathbb{D}, L \rangle$. Then Π is linear iff

$$(\forall \Phi \longrightarrow \Psi \in L)(\forall u \in U)\,(|\Psi^{-1}(u)| \leqslant 1).$$

In languages with linked storage manipulation facilities (such as AMBIT/G), it is especially easy to implement linear FTTs. Each rule-schema can be represented by a simple local operation on the data graph. There is no need to use an expensive copying process or an ingenious trick that avoids copying but may be difficult to debug or applicable only in special cases.

There is another important property of FTTs that can be formally defined for TDFTTs and BUFTTs simultaneously. Recall that a deterministic automaton has exactly one response to each combination of control state and currently scanned input and storage symbols. Rounds [46, p. 264] has considered the weaker notion of "partial determinism": there is at most one response to each combination of control state and currently scanned input and storage symbols. The discussion in [46] is limited to TDFTTs, but the same definition applies to BUFTTs.

(13.10) Definition. Let Π be an FTT with natural presentation $\langle \mathbb{D}, L \rangle$. Then Π is partial deterministic iff L is a partial function on $V_{\#}$: no two rule-schemata have the same left half.

(13.11) Lemma. Let $\Pi = (\Sigma, \Delta, @, \mathfrak{C})$ be a partial deterministic FTT. Then $\langle \Pi \rangle$ represents a partial function from $\Sigma_{\#}$ into $\Delta_{\#}$:

(1) $$(\forall R \in \Sigma_{\#})(|\langle \Pi \rangle\{R\}| \leqslant 1).$$

Proof: Let $\langle \mathbb{D}, L\rangle$ be the natural presentation for Π, so that $\mathfrak{C} = (V, \mathbb{F}, \Rightarrow, \mathbb{R}_L)$. We will show that $\mathfrak{C}$ is Church-Rosser and that this implies (1).

We claim that $\mathfrak{C}$ is unequivocal: no two rules have the same left half. Suppose $\varphi \longrightarrow \psi$, $\varphi' \longrightarrow \psi' \in \mathbb{R}_L$ with $\varphi = \varphi'$. Then there are $\Phi \longrightarrow \Psi$, $\Phi' \longrightarrow \Psi' \in L$ such that $\varphi \longrightarrow \psi$ is an instance of $\Phi \longrightarrow \Psi$ and $\varphi' \longrightarrow \psi'$ is an instance of $\Phi' \longrightarrow \Psi'$. By the restrictions (13.5.1) and (13.7.1) on the left halves of schemata in FTTs, $\varphi = \varphi'$ implies that $\Phi = \Phi'$. Therefore $\Psi = \Psi'$ by partial determinism. Since $\varphi = \varphi'$, the tree substituted for each parameter in Ψ to form ψ is the same tree substituted for that parameter in Ψ' to form ψ', so $\Psi = \Psi'$ implies $\psi = \psi'$. This proves that $\mathfrak{C}$ is unequivocal.

We claim that $\mathfrak{C}$ is closed in the sense of Definition 5.4. Let each $\varphi \longrightarrow \psi \in \mathbb{R}_L$ be assigned the trivial residue map: no residues for each n in Dom φ. The conditions in Definition 5.4 will be trivially true if, whenever $\varphi_0 \longrightarrow \psi_0 \in \mathbb{R}_L$ and $n \in$ Dom φ_0 with $n \neq ()$, then φ_0/n is not the left half of a rule. To show this we consider two cases. If Π is top-down, then $\varphi_0 n \notin \mathcal{M}$ and so $(\varphi_0/n)() \notin \mathcal{M}$. If Π is bottom-up, then $\varphi_0 n \notin (\Sigma \cup \{@\})$ and so $(\varphi_0/n)() \notin (\Sigma \cup \{@\})$. In both cases the label at the root of φ_0/n is outside the set of possible labels $\varphi()$ for $\varphi \to \psi \in \mathbb{R}_L$, so φ_0/n is not the left half of a rule.

By the Main Theorem 5.6, $\mathfrak{C}$ is Church-Rosser. For all $R \in \Sigma_{\#}$ and $T \in \Delta_{\#}$, we have

$$\begin{aligned} T \in \langle \Pi \rangle \{R\} \quad & \text{iff} \quad @(R) \overset{*}{\Longrightarrow} T \text{ in } \mathfrak{C} \\ & \text{iff} \quad T \text{ is a normal form for } @(R) \text{ in } \mathfrak{C} \end{aligned}$$

because trees in $\Delta_{\#}$ are irreplaceable in TDFTTs or BUFTTs. This implies (1) because normal forms are unique in Church-Rosser systems. ∎

If Π is partial deterministic, then the relation

$$\{\langle R, T\rangle \mid R \in \Sigma_{\#} \ \& \ T \in \langle \Pi \rangle \{R\}\}$$

is a partial function from $\Sigma_{\#}$ into $\Delta_{\#}$. It is convenient to denote this function $\langle \Pi \rangle$ also, using an expression like

$$\langle \Pi \rangle : \Sigma_{\#} \longrightarrow \Delta_{\#}$$

to remove ambiguity.

The definitions of linearity (13.9) and partial determinism (13.10) are the same for TDFTTs and BUFTTs. The other FTT concepts require slightly different definitions for top-down and bottom-up transducers. Perhaps a more unified formulation will emerge in the course of further investigation. The treatment given below is as unified as is feasible at present.

Although the set $\mathfrak{M}$ of markers is infinite, only finitely many markers can actually be used as states of tape heads in each FTT. In a TDFTT the root marker acts as the initial state for the tape head that begins reading the input tree at the root. In a BUFTT the root marker merely marks the root for tape heads that use other markers as states.

(13.12) Definition. Let $\Pi = (\Sigma, \Delta, @, \mathfrak{C})$ be an FTT with natural presentation $\langle \mathbb{D}, L\rangle$. The set Q of states of Π is defined by

(1) $Q = \{s \in \mathfrak{M} \mid (\exists \Phi \longrightarrow \Psi \in L)(\Psi^{-1}(s) \neq \emptyset)\} \cup \{@\}$

if Π is top-down and by

(2) $Q = \{s \in \mathfrak{M} - \{@\} \mid (\exists \Phi \longrightarrow \Psi \in L)(\Psi^{-1}(s) \neq \emptyset)\}$

if Π is bottom-up. When $|Q| = K$, then Π is said to be a K-state transducer.

An FTT has only finitely many states because the natural presentation is finite. Finiteness of the state set is not by itself very helpful: even Turing machines have only finitely many control states. The reason we consider FTTs to be "finite" is that important questions about them can be answered effectively from the finite amount of information in their natural presentations. The following lemma illustrates this idea.

(13.13) Lemma. Let $\Pi = (\Sigma, \Delta, @, \mathfrak{C})$ be an FTT with $\mathfrak{C} = (V, \mathbb{F}, \Longrightarrow, \mathbb{R})$. Let $\langle \mathbb{D}, L\rangle$ be the natural presentation for Π. For any $R \in \mathbb{F}$, the set of normal forms of R under $\mathfrak{C}$ is a finite nonempty subset of $\mathbb{F}$ that is effectively computable from R and L.

Proof: Let $R \in \mathbb{F}$. Consider the sequence of subsets of $\mathbb{F}$ defined by

$$\mathcal{R}_0 = \{R\}$$

$$\mathcal{R}_{i+1} = \mathcal{R}_i \cup \{T \mid (\exists S \in \mathcal{R}_i)(S \Longrightarrow T)\}.$$

For each $i \in \mathbb{N}$ and $S \in \mathcal{R}_i$, the finiteness of S and L implies that $\{T \mid S \Longrightarrow T\}$ is finite and effectively computable from S and L. By induction on i, each $\mathcal{R}_i$ is therefore finite and effectively computable from R and L.

Suppose that some $K \in \mathbb{N}$ has $\mathcal{R}_{K+1} = \mathcal{R}_K$. Then

$$\mathcal{R}_K = \{T \in \mathbb{F} \mid R \overset{*}{\Longrightarrow} T\}$$

and every normal form for R is in $\mathcal{R}_K$. By computing $\mathcal{R}_0, \mathcal{R}_1, \ldots$ and checking whether $\mathcal{R}_i = \mathcal{R}_{i+1}$ at each step, we can find $\mathcal{R}_K$. Since L is finite, we can test each member of $_K$ for irreplaceability, and the irreplaceable members of $\mathcal{R}_K$ are exactly the normal forms for R. Thus the set of normal forms for R is finite and effectively computable from R and L.

We must show that $\mathcal{R}_{K+1} = \mathcal{R}_K$ for some $K \in \mathbb{N}$ and that the set of normal forms for R is nonempty. It will suffice to show that a "weight" $w(S) \in \mathbb{N}$ can be assigned to each $S \in \mathbb{F}$ in such a way that

$$(\forall S, T \in \mathbb{F})(S \Longrightarrow T \text{ implies } w(S) > w(T)).$$

Case 1 (Π is top-down) Define the height

$$h(T) = \text{Max}\,\{|n| \mid n \in \text{Dom } T\}$$

for any tree T. Using the natural presentation $\langle \mathbb{D}, L\rangle$ and the set Q of states for Π, set

$$M = \text{Max}\,\{|\Psi^{-1}(Q)| \mid (\exists\, \Phi \in V_{\#})(\Phi \longrightarrow \Psi \in L)\}$$

and

$$w(S) = \sum_{n \in S^{-1}(Q)} (M+1)^{h(S/n)}$$

for all $S \in \mathbb{F}$. If $S \Longrightarrow T$ by application of a rule at a node n in Dom S, let $h = h(S/n)$ and note that

$$w(T) - w(S) \leq M(M+1)^{h-1} - (M+1)^{h} = -(M+1)^{h-1} < 0,$$

so $w(S) > w(T)$.

Case 2 (Π is bottom-up) Set $w(S) = |S^{-1}(\Sigma \cup \{@\})|$ for all $S \in \mathbb{F}$. If $S \Longrightarrow T$ then $w(T) = w(S) - 1$, so $w(S) > w(T)$. ∎

If $\Pi = (\Sigma, \Delta, @, \mathfrak{C})$ is an SRS transducer in Definition 13.3, then has the form $(V, \mathbb{F}, \Longrightarrow, \mathbb{R})$ with

$$V = \Sigma \cup \Delta \cup \mathfrak{M} \cup U$$

and

$$\mathbb{F} = \{S \in V_{\#} \mid (\exists R \in (\Sigma \cup U)_{\#})(@(R) \overset{*}{\Longrightarrow} S)\}.$$

If Π is an FTT then the restrictions on the forms of the rules lead to restrictions on the forms of trees in $\mathbb{F}$. These restrictions formalize the idea of one-way tape motion. For example, suppose that Π is a TDFTT in Definition 13.5 with a set Q of states from Definition 13.12. Let $S \in \mathbb{F}$. Then $S^{-1}(Q)$ is the set of all positions of tape heads on S, and Dom $S \cap (S^{-1}(Q) \cdot (0) \cdot \mathbb{N}^*)$ is the set of nodes in S that have yet to be scanned by tape heads. These nodes must be labelled by input symbols or parameters:

$$\text{Dom } S \cap (S^{-1}(Q) \cdot (0) \cdot \mathbb{N}^*) \subseteq S^{-1}(\Sigma \cup U).$$

On the other hand, Dom S - $(S^{-1}(Q)\cdot\mathbb{N}^*)$ is the set of nodes in S that have already been formed by the actions of downward moving tape heads:

$$\mathrm{Dom}\ S - (S^{-1}(Q)\cdot\mathbb{N}^*) \subseteq S^{-1}(\Delta).$$

We also have

$$\begin{aligned} S^{-1}(Q) = \emptyset \ &\text{iff all tape heads have vanished} \\ &\text{iff processing is complete} \\ &\text{iff } S \in \Delta_{\#}. \end{aligned}$$

Similar remarks apply to BUFTTs from Definition 13.7, but now the nodes below tape heads display previously formed portions of the output and the nodes above tape heads have yet to be read. The following lemma formalizes these considerations.

(13.14) Lemma. Let $\Pi = (\Sigma, \Delta, @, \mathfrak{C})$ be an FTT with $\mathfrak{C} = (V, \mathbb{F}, \Longrightarrow, \mathbb{R})$. Let $\langle \mathbb{D}, L\rangle$ be the natural presentation and let Q be the set of states. If Π is top-down then, for all $S \in \mathbb{F}$,

(1) $\mathrm{Dom}\ S \cap (S^{-1}(Q)\cdot(0)\cdot\mathbb{N}^*) \subseteq S^{-1}(\Sigma \cup U)$

(2) $\mathrm{Dom}\ S - (S^{-1}(Q)\cdot\mathbb{N}^*) \subseteq S^{-1}(\Delta)$

(3) $S^{-1}(Q) = \emptyset$ iff $S \in \Delta_{\#}$.

If Π is bottom-up then, for all $S \in \mathbb{F}$,

(4) $\mathrm{Dom}\ S \cap (S^{-1}(Q)\cdot(0)\cdot\mathbb{N}^*) \subseteq S^{-1}(\Delta)$

(5) $S() \neq @$ or $\mathrm{Dom}\ S - (S^{-1}(Q)\cdot\mathbb{N}^*) \subseteq S^{-1}(\Sigma \cup \{@\} \cup U)$

(6) $S() \neq @$ iff $S \in \Delta_{\#}$.

Proof: The forest $\mathbb{F}$ is defined to be

$$\{S \in V_\# \mid (\exists R \in (\Sigma \cup U)_\#)(@(R) \overset{*}{\Longrightarrow} S)\}$$

by (13.3.1), so it will suffice to show that trees of the form @(R) for $R \in (\Sigma \cup U)_\#$ have the desired properties and that these properties are preserved by applications of rules. For $S = @(R)$ with $R \in (\Sigma \cup U)_\#$, (1)--(3) (if Π is top-down) or (4)--(6) (if Π is bottom-up) are trivial. Now let $S \in \mathbb{F}$ with the desired properties. Suppose $S \Longrightarrow S'$ by application of a rule $\varphi \longrightarrow \psi$ at $n \in \text{Dom } S$.

Case 1 (Π is top-down) The restrictions (13.5.1) and (13.5.2) on the rule-schemata in L assure that S′ inherits (1) and (2) from S. Since (2) implies that

$$S^{-1}(Q) = \emptyset \text{ iff } \text{Dom } S \subseteq S^{-1}(\Delta) \text{ iff } S \in \Delta_\#,$$

(3) follows from (2).

Case 2 (Π is bottom-up) The restrictions (13.7.1)--(13.7.3) on the rule-schemata in L assure that S′ inherits (4) from S. Now we prove (5). Suppose first that $\varphi \longrightarrow \psi$ is of the form

$$a(s_0(T_0), \ldots, s_{-1}(T_{-1})) \longrightarrow \Psi(\Psi^{-1}(u_0) \leftarrow T_0) \ldots (\Psi^{-1}(u_{-1}) \leftarrow T_{-1})$$

with $a \in \Sigma$ and

$$a(s_0(u_0), \ldots, s_{-1}(u_{-1})) \longrightarrow \Psi \in L.$$

Then $S \notin \Delta_\#$ because $a \notin \Delta$ and so $S(\,) = @$ by (6) for S. By (5) for S,

$$\text{Dom } S - (S^{-1}(Q) \cdot \mathbb{N}^*) \subseteq S^{-1}(\Sigma \cup \{@\} \cup U).$$

By $\Psi() \in Q$ in the restriction (13.7.2) on members of L, S′ inherits this property from S. On the other hand, suppose $\varphi \longrightarrow \psi$ is of the form

$$@(s(T)) \longrightarrow T \quad \text{with} \quad T \in (\Delta \cup U)_{\#}.$$

Then $S = @(s(T))$ and $S' = T$. We have $T() \neq @$ and so (5) holds for S′ in any case.

Finally, we prove (6) by showing that both sides have the same truth value. Suppose first that $\varphi \longrightarrow \psi$ is an instance of a schema

$$a(s_0(u_0), \ldots, s_{-1}(u_{-1})) \longrightarrow \Psi \in L$$

with $a \in \Sigma$. Then $S'() = @$ because $S() = @$ while $S' \notin \Delta_{\#}$ because $\Psi() \in \mathfrak{M}$. Both sides of (6) are false. On the other hand, suppose $\varphi \longrightarrow \psi$ is of the form

$$@(s(T)) \longrightarrow T \quad \text{with} \quad T \in (\Delta \cup U)_{\#}.$$

By (4) for S, T is actually in $\Delta_{\#}$ and so $S' \in \Delta_{\#}$. Both sides of (6) are true. ∎

The state set of an FTT permits us to complete the definition of determinism for FTTs. The idea that there is at most one response to every combination of control state and currently scanned input symbol has been formalized in the definition of partial determinism (13.10). The idea that there is at least one response to each such combination can now be formalized.

(13.15) Definition. Let $\Pi = (\Sigma, \Delta, @, \mathfrak{C})$ be an FTT with natural presentation $\langle \mathbb{D}, L \rangle$ and set Q of states. Then Π is total iff either Π is top-down and

(1) $$(\forall s \in Q)(\forall a \in \Sigma)(\exists \Psi \in V_{\#})$$
$$[\, s(a(u_0, \ldots, u_{-1})) \longrightarrow \Psi \in L\,]$$

or Π is bottom-up and

(2) $$(\forall a \in \Sigma \cup \{@\})(\forall s \in Q^{\rho(a)})(\exists \Psi \in V_{\#})$$
$$[\, a(s_0(u_0), \ldots, s_{-1}(u_{-1})) \longrightarrow \Psi \in L\,].$$

An FTT that is both partial deterministic and total is <u>deterministic</u>. This agrees with the concepts of determinism used by Rounds [45] [46] and Thatcher [51] [52].

(13.16) <u>Theorem</u>. Let $\Pi = (\Sigma, \Delta, @, \mathfrak{C})$ be a deterministic FTT. Then $\langle \Pi \rangle$ represents a total function from $\Sigma_{\#}$ into $\Delta_{\#}$:

$$(\forall R \in \Sigma_{\#})(\exists T \in \Delta_{\#})(\langle \Pi \rangle \{R\} = \{T\}).$$

<u>Proof</u>: Using the assumption that Π is total, we will show that

(1) $$(\forall S \in \mathbb{F})[S^{-1}(U) = \emptyset \text{ implies } (\exists T \in \Delta_{\#})(S \overset{*}{\Longrightarrow} T)],$$

where $\mathfrak{C} = (V, \mathbb{F}, \Longrightarrow, \mathbb{R})$. Assume this for the moment.

Let $R \in \Sigma_{\#}$. Condition (2) in the definition of SRS transducers (13.3) specifies that

$$\langle \Pi \rangle \{R\} = \{T \in \Delta_{\#} \mid @(R) \overset{*}{\Longrightarrow} T\}.$$

By (1) with $S = @(R)$, this implies that

$$(\exists T \in \Delta_{\#})(T \in \langle \Pi \rangle \{R\}).$$

But $|\langle \Pi \rangle \{R\}| \leq 1$ by Lemma 13.10 and the partial determinism of Π, so this implies that some $T \in \Delta_{\#}$ has $\{T\} = \langle \Pi \rangle \{R\}$, as desired. We must prove (1).

By Lemma 13.13, each $S \in \mathbb{F}$ has at least one normal form $T \in \mathbb{F}$, and $S^{-1}(U) = \emptyset$ implies that $T^{-1}(U) = \emptyset$ since applications of rules cannot introduce parameters. It will therefore suffice to prove

$$(\forall T \in \mathbb{F})((T \text{ irreplaceable } \& \ T^{-1}(U) = \emptyset) \text{ implies } T \in \Delta_{\#}),$$

which is implied by

(2) $$(\forall T \in \mathbb{F})[(T \notin \Delta_{\#} \ \& \ T^{-1}(U) = \emptyset) \text{ implies } (\exists n \in \operatorname{Dom} T)(\exists \varphi \longrightarrow \psi \in \mathbb{R})(T/n = \varphi)].$$

Suppose $T \notin \Delta_{\#}$ and $T^{-1}(U) = \emptyset$.

<u>Case 1</u> (Π is top-down) By $\operatorname{Dom} T - (T^{-1}(Q)\cdot\mathbb{N}^*) \subseteq T^{-1}(\Delta)$ (from (2) in Lemma 13.4) and $T^{-1}(\Delta) \neq \operatorname{Dom} T$, there is a node $n \in \operatorname{Dom} T$ with $Tn \in Q$. By $\operatorname{Dom} T \cap (T^{-1}(Q)\cdot(0)\cdot\mathbb{N}^*) \subseteq T^{-1}(\Sigma \cup U)$ (from (1) in Lemma 13.14) and $T^{-1}(U) = \emptyset$, $T(n\cdot(0)) \in \Sigma$. Let s be Tn and let a be $T(n\cdot(0))$. Since Π is total there is a rule-schema

$$s(a(u_0, \ldots, u_{-1})) \longrightarrow \Psi \in L$$

with an instance

(3) $$s(a(P_0, \ldots, P_{-1})) \longrightarrow \Psi(\Psi^{-1}(u_0) \longleftarrow P_0) \ldots (\Psi^{-1}(u_{-1}) \longleftarrow P_{-1})$$

in $\mathbb{R}$ for any $P_0, \ldots, P_{-1} \in (\Sigma \cup U)_{\#}$. In particular, let $P_k = T/(n\cdot(0,k))$ for all $k < \rho(a)$, so that $\operatorname{Dom} T \cap (T^{-1}(Q)\cdot(0)\cdot\mathbb{N}^*) \subseteq T^{-1}(\Sigma \cup U)$ (from (1) in Lemma 13.14) implies $P_k \in (\Sigma \cup U)_{\#}$. The rule (3) is applicable at n in T.

<u>Case 2</u> (Π is bottom-up) By $T \notin \Delta_{\#}$ and part (6) of Lemma 13.14, $T(\,) = @$.

<u>Case 2.1</u> ($T(0) \in Q$) Let s be $T(0)$. Since Π is total there is a rule-schema

$$@(s(u_0)) \longrightarrow u_0 \in L$$

with an instance

(4) $$@(s(P)) \longrightarrow P$$

in $\mathbb{R}$ for any $P \in (\Delta \cup U)_{\#}$. In particular, let $P = T/(0, 0)$, so that $\mathrm{Dom}\ T \cap (T^{-1}(Q)\cdot(0)\cdot\mathbb{N}^*) \subseteq T^{-1}(\Delta)$ (from (4) in Lemma 13.14) implies $P \in \Delta_{\#}$. The rule (4) is applicable at ( ) in T.

<u>Case 2.2</u> ($T(0) \notin Q$). By $T(\,) = @$ and part (5) of Lemma 13.14, $T(0) \in \Sigma \cup \{@\} \cup U$. By induction on J in $@(R) \overset{J}{\Longrightarrow} T$ for some $R \in (\Sigma \cup U)_{\#}$, we can show that $T^{-1}(@) \subseteq \{(\,)\}$. Therefore $T(0) \in \Sigma \cup U$. By $T^{-1}(U) = \emptyset$, $T(0) \in \Sigma$. Therefore $T^{-1}(\Sigma) \neq \emptyset$.

Let $n \in T^{-1}(\Sigma)$ with $|n|$ maximal. Let a be Tn. We claim that $T(n\cdot(k)) \in Q$ for each $k < \rho(a)$. By $\Sigma \cap \Delta = \emptyset$ and $\mathrm{Dom}\ T \cap (T^{-1}(Q)\cdot(0)\cdot\mathbb{N}^*) \subseteq T^{-1}(\Delta)$ (from (4) in Lemma 13.14), $n \notin T^{-1}(Q)\cdot(0)\cdot\mathbb{N}^*$. But $n \notin T^{-1}(Q)$ by $\Sigma \cap Q = \emptyset$, so $n \notin T^{-1}(Q)\cdot\mathbb{N}^*$. Therefore $n\cdot(k) \notin T^{-1}(Q)\cdot\mathbb{N}^*\cdot\mathbb{N}^1$, so

$$n\cdot(k) \in T^{-1}(Q) \cup (\mathrm{Dom}\ T-(T^{-1}(Q)\cdot\mathbb{N}^*)).$$

But $n\cdot(k) \in \mathrm{Dom}\ T-(T^{-1}(Q)\cdot\mathbb{N}^*)$ would imply $T(n\cdot(k)) \in \Sigma$ by the same argument that showed $T(0) \in \Sigma$, so $n\cdot(k) \in T^{-1}(Q)$, as desired.

Let s_k be $T(n\cdot(k))$ for each $k < \rho(a)$. Since Π is total there is a rule-schema

$$a(s_0(u_0), \ldots, s_{-1}(u_{-1})) \longrightarrow \Psi \in L$$

with an instance

$$(5) \quad a(s_0(P_0), \ldots, s_{-1}(P_{-1})) \longrightarrow \Psi(\Psi^{-1}(u_0) \longleftarrow P_0) \ldots (\Psi^{-1}(u_{-1}) \longleftarrow P_{-1})$$

in $\mathbb{R}$ for any $P_0, \ldots, P_{-1} \in (\Delta \cup U)_{\#}$. In particular, let $P_k = T/(n\cdot(0,k))$ for all $k < \rho(a)$, so that $\mathrm{Dom}\, T \cap (T^{-1}(Q)\cdot(0)\cdot\mathbb{N}^*) \subseteq T^{-1}(\Delta)$ (from (4) in Lemma 13.14) implies $P_k \in \Delta_{\#}$. The rule (5) is applicable at n in T. This completes the proof of (2). ∎

The semantic analyzer in a compiler should assign exactly one coding tree to each augmented ranked parse tree; we have just shown that deterministic FTTs can do this. The 1-state deterministic FTTs have an additional convenient property: they can be considered to be either top-down or bottom-up, whichever is appropriate for the use of coroutine linkages as described at the end of §12. As Thatcher [51, p. 271] points out, 1-state TDFTTs can be effectively transformed into 1-state BUFTTs (and *vice versa*) without changing the transductions they compute. Unfortunately, multistate FTTs appear to have no similar theorem, and Thatcher has shown that some deterministic TDFTT transductions cannot be computed by 1-state deterministic TDFTTs [52, Thm. 6.13].

14. Closure Under Composition

Several kinds of tree transducer are now available: 1-state FTTs, linear TDFTTs, and so on. These transducers define classes of transductions which map sets of trees to sets of trees. In this section we show that some of the classes are closed under composition of functions. Only top-down transducers are considered here, but we conjecture that bottom-up transducers have similar closure properties. We also show that the class of partial deterministic linear TDFTT transductions is not closed under composition.

Closure under composition is a natural mathematical question to raise about any class of maps, and positive answers are both common and valuable. Continuous maps, recursive functions, algebraic morphisms, and many other major classes are closed. When proven constructively, closure theorems can have direct practical significance as well. We begin by discussing the computational uses of constructive closure theorems.

Suppose we are given a class $\mathfrak{X}$ of objects and that to each $X \in \mathfrak{X}$ we can assign two sets, In(X) and Out(X), and a (partial) map

(1) $\langle X\rangle : \mathrm{In}(X) \longrightarrow \mathrm{Out}(X)$.

In particular, In(X) could be $2^{\Sigma_\#}$ and Out(X) could be $2^{\Delta_\#}$ if X is a tree transducer with input vocabulary Σ and output vocabulary Δ. Now suppose that to each $X \in \mathfrak{X}$ we can assign a computer program Prog(X) that computes $\langle X\rangle$. One strategy for computing a map

(2) $h : A \longrightarrow C$

would be to seek an object $Z \in \mathfrak{X}$ such that

(3) $\mathrm{In}(Z) = A \ \&\ \mathrm{Out}(Z) = C \ \&\ \langle Z \rangle = h.$

If such a Z can be found, then Prog(Z) is a program for h with no bugs. In complicated situations, however, it will not be obvious how to choose Z. We could attempt to divide and conquer by expressing h as a composition of simpler maps, say

(4) $h = g \circ f \quad \text{where} \quad f : A \longrightarrow B \ \&\ g : B \longrightarrow C.$

It might then be relatively easy to find objects X and Y such that

(5) $\mathrm{In}(X) = A \ \&\ \mathrm{Out}(X) = B \ \&\ \langle X \rangle = f$

(6) $\mathrm{In}(Y) = B \ \&\ \mathrm{Out}(Y) = C \ \&\ \langle Y \rangle = g.$

One way to compute h would be to link Prog(X) and then Prog(Y) in sequence, but this could require a great deal of intermediate storage. A more efficient way would be to form a third program Prog(Y) + Prog(X) in which Prog(Y) calls upon Prog(X) for input and Prog(X) returns control to Prog(Y) whenever it outputs a portion of Prog(Y)'s input. We sketched the use of coroutine linkages for this purpose at the end of §12. Note that adding these linkages to Prog(Y) and Prog(X) so as to form Prog(Y) + Prog(X) is not an automatic operation at present: the new program must be written by hand from the texts of Prog(Y) and Prog(X). This process may introduce bugs.

If a closure theorem can be proven constructively, then there is another method with moderate intermediate storage requirements.

This method cannot introduce bugs and will generally lead to a program faster than Prog(Y) + Prog(X).

From (5) and (6) it follows that

(*) $\quad \mathrm{Out}(X) = \mathrm{In}(Y)$.

An effective closure theorem would construct a "product" object $Y \wedge X$ on the basis of this fact such that

(**) $\quad \mathrm{In}(Y \wedge X) = \mathrm{In}(X)\ \&\ \mathrm{Out}(Y \wedge X) = \mathrm{Out}(Y)\ \&\ \langle Y \wedge X \rangle = \langle Y \rangle \circ \langle X \rangle$.

In (5) and (6) this implies that

$$\mathrm{In}(Y \wedge X) = A\ \&\ \mathrm{Out}(Y \wedge X) = C\ \&\ \langle Y \wedge X \rangle = g \circ f.$$

By (4), choosing $Z = Y \wedge X$ will make (3) true. Assuming, of course, that the closure theorem is correct, then Prog(Z) will compute h in one stage with no bugs. In effect we form the coroutine linkages between the objects X and Y themselves, then pass to a program. The product operation $\wedge : \mathfrak{X}^2 \longrightarrow \mathfrak{X}$ is a precise mathematical function, while the linkage operation + is only a programming heuristic.

We claim that Prog($Y \wedge X$) will generally be faster than Prog(Y) + Prog(X). To support this assertion in the abstract would require a more formal treatment of coroutines operating on structured data than was given in §12. It will be simpler here to consider an example that supports the claim adequately for present purposes and that displays the general principle clearly.

Let $\mathfrak{X}$ be the class of all deterministic finite string transducers as defined in (11.5). For any $M = (K, \Sigma, \Delta, \delta, \lambda, s, F)$ in $\mathfrak{X}$, we have

$\mathrm{In}(M) = \Sigma^*$ and $\mathrm{Out}(M) = \Delta^*$. The transduction $\langle M\rangle$ defined by (11.6.5) is a partial map from In(M) to Out(M). To mimic the action of M when reading symbol a in state r, Prog(M) finds $(\delta(r,a), \lambda(r,a))$ in a $|K|$ by $|\Sigma|$ matrix, updates the row pointer of the matrix to $\delta(r,a)$, and writes $\lambda(r,a)$. This takes $\kappa + |\lambda(r,a)|$ units of time, where κ represents the accessing and updating time for the matrix. Unless $|K|\,|\Sigma|$ is so large that the matrix must be stored in an unusual way, κ does not depend on M.

The well-known Cartesian product construction defines a new transducer $M_2 \wedge M_1$ whenever the output vocabulary of M_1 is the input vocabulary of M_2. Recall the definitions of the extended output and transition maps in (11.6).

(14.1) Definition. For $i = 1, 2$ let $M_i = (K_i, \Sigma_i, \Delta_i, \delta_i, \lambda_i, s_i, F_i)$ be a deterministic finite string transducer. Suppose $\Delta_1 = \Sigma_2$. Then

(1) $$M_2 \wedge M_1 = (K_2 \times K_1, \Sigma_1, \Delta_2, \delta_3, \lambda_3, \langle s_2, s_1\rangle, F_2 \times F_1)$$

where, for all $r_2 \in K_2$, $r_1 \in K_1$, and $a \in \Sigma_1$,

(2) $$\delta_3(\langle r_2, r_1\rangle, a) = \langle \overline{\delta}_2(r_2, \lambda_1(r_1, a)), \delta_1(r_1, a)\rangle$$

(3) $$\lambda_3(\langle r_2, r_1\rangle, a) = \overline{\lambda}_2(r_2, \lambda_1(r_1, a)).$$

By induction on lengths of strings, we can show that all $r_2 \in K_2$, $r_1 \in K_1$, and $x \in \Sigma_1{}^*$ have

$$\overline{\delta}_3(\langle r_2, r_1\rangle, x) = \langle \overline{\delta}_2(r_2, \overline{\lambda}_1(r_1, x)), \overline{\delta}_1(r_1, x)\rangle$$

and

$$\overline{\lambda}_3(\langle r_2, r_1\rangle, x) = \overline{\lambda}_2(r_2, \overline{\lambda}_1(r_1, x)).$$

By the definition of $\langle M\rangle$ in (11.6.5), we therefore have

$$\begin{aligned}\langle M_2 \wedge M_1\rangle &= \{\langle x, \overline{\lambda}_3(\langle s_2, s_1\rangle, x)\rangle \mid x \in \Sigma_1^* \ \& \\ &\qquad\qquad \overline{\delta}_3(\langle s_2, s_1\rangle, x) \in F_2 \times F_1\} \\ &= \{\langle x, \overline{\lambda}_2(s_2, \overline{\lambda}_1(s_1, x))\rangle \mid x \in \Sigma_1^* \ \& \ \overline{\delta}_1(s_1, x) \in F_1 \ \& \\ &\qquad\qquad \overline{\delta}_2(s_2, \overline{\lambda}_1(s_1, x)) \in F_2 \quad \} \\ &= \langle M_2\rangle \circ \langle M_1\rangle .\end{aligned}$$

Thus the product $\wedge$ provides an effective closure theorem for deterministic finite string transducers.

Assuming that a $|K_2|\,|K_1|$ by $|\Sigma_1|$ matrix is not too large, we compare the time required by Prog($M_2 \wedge M_1$) to process input a from state $\langle r_2, r_1\rangle$ with the time required by Prog(M_2) + Prog(M_1) to process a from state r_1 on M_1 and then process $\lambda_1(r_1, a)$ from state r_2 on M_2. If τ is the time spent in coroutine transfers, then the times are

$$\begin{aligned}&\text{Prog}(M_2) + \text{Prog}(M_1) : \kappa + (1+\kappa)\,|\lambda_1(r_1, a)| + |\overline{\lambda}_2(r_2, \lambda_1(r_1, a))| + \tau \\ &\text{Prog}(M_2 \wedge M_1) : \kappa + |\overline{\lambda}_2(r_2, \lambda_1(r_1, a))| \\ \hline &\text{Difference} : (1+\kappa)\,|\lambda_1(r_1, a)| + \tau\end{aligned}$$

A similar saving will be apparent in our product construction for TDFTTs. The construction itself will be similar to the one just given, but the inductive verification will no longer be trivial.

Recall from Definition 13.3 that an SRS transducer is a 4-tuple

$\Pi = (\Sigma, \Delta, @, \mathfrak{C})$ where $\mathfrak{C}$ is an SRS of the form $(V, \mathbb{F}, \Longrightarrow, \mathbb{R})$ and

$$V = \Sigma \cup \Delta \cup \mathfrak{M} \cup U$$

$$\mathbb{F} = \{S \in V_{\#} \mid (\exists R \in (\Sigma \cup U)_{\#}(@(R) \overset{*}{\Longrightarrow} S)\},$$

so that $\mathfrak{C}$ is actually determined by $\mathbb{R}$. If there is a finite set L of rule-schemata such that $\mathbb{R} = \mathbb{R}_L$ when each parameter is assigned the domain $(\Sigma \cup U)_{\#}$, and such that each $\Phi \longrightarrow \Psi \in L$ satisfies two constraints, then Π is a TDFTT. The constraints are (13.5.1):

$$(\exists a \in \Sigma)(\exists s \in \mathfrak{M})[\Phi = s(a(u_0, \ldots, u_{-1}))],$$

and (13.5.2):

$$\Psi \in (\Delta \cup \mathfrak{M} \cup U)_{\#} \;\&\; \Psi^{-1}(\mathfrak{M}) \cdot (0) = \Psi^{-1}(U).$$

As a prelude to the discussion of the product construction for TDFTTs, we consider a formal statement of the idea that TDFTTs generalize finite string transducers. The set $\mathfrak{M}$ of markers is infinite, so we may assume that the state set K of any string transducer is a subset of $\mathfrak{M}$. We may also assume that the input and output vocabularies contain no markers or parameters, since this can always be achieved by renaming.

(14.2) Theorem. Let $M = (K, \Sigma_1, \Delta_1, \delta, \lambda, @, F)$ be any deterministic finite string transducer with $K \subseteq \mathfrak{M}$ and $(\Sigma_1 \cup \Delta_1) \cap (\mathfrak{M} \cup U) = \emptyset$. There exist a proper pair (Σ, Δ) of ranked alphabets, bijections

$$\mu : \Sigma_1^* \longrightarrow \Sigma_{\#} \;\&\; \nu : \Delta_1^* \longrightarrow \Delta_{\#},$$

and a linear partial deterministic TDFTT $\Pi_M = (\Sigma, \Delta, @, \mathfrak{C})$ such that the tree transduction $\langle \Pi_M \rangle : \Sigma_\# \longrightarrow \Delta_\#$ corresponds to the string transduction $\langle M \rangle : \Sigma_1{}^* \longrightarrow \Delta_1{}^*$:

$$\langle \Pi_M \rangle = \nu \circ \langle M \rangle \circ \mu^{-1}.$$

Proof: Let $\dashv$ be a new symbol and set $\Sigma = \Sigma_1 \cup \{\dashv\}$ and $\Delta = \Delta_1 \cup \{\dashv\}$. Let $\dashv$ have rank zero and let members of $\Sigma_1 \cup \Delta_1$ have rank one, so that (Σ, Δ) is a proper pair of ranked alphabets in Definition 13.2. The bijection μ is defined by induction on lengths of strings: each $x \in \Sigma_1{}^*$ has

$$\mu x = \mu(x_0, \ldots, x_{-1}) = x_0(x_1(\ldots x_{-1}(\dashv)\ldots)).$$

(Thus $\mu()$ is the tree with one node labelled $\dashv$.) The definition of ν is similar.

Let L_1 be the set of all rule-schemata

$$r(a(u_0)) \longrightarrow b_0(b_1(\ldots b_{-1}(s(u_0))\ldots))$$

such that $(b_0, \ldots, b_{-1}) = \lambda(r, a)$ and $s = \delta(r, a)$. Let L_2 be the set of all rule-schemata

$$r(\dashv) \longrightarrow \dashv$$

such that $r \in F$. Let $L_M = L_1 \cup L_2$, so that $\langle (\Sigma \cup U)_\#, L_M \rangle$ is the natural presentation of a TDFTT $\Pi_M = (\Sigma, \Delta, @, \mathfrak{C})$ with $\mathfrak{C} = (V, \mathbb{F}, \Longrightarrow, \mathbb{R})$ whose root marker is the starting state @ of M. This transducer is linear in the sense of Definition 13.9 and partial deterministic in the sense of Definition 13.10. By Lemma 13.11, $\langle \Pi_M \rangle$ may be treated as a partial function $\langle \Pi_M \rangle : \Sigma_\# \longrightarrow \Delta_\#$.

To show that $\langle \Pi_M \rangle = \nu \circ \langle M \rangle \circ \mu^{-1}$ is to show that

(1) $$(\forall x \in \Sigma_1^*)(\forall y \in \Delta_1^*)$$
$$[@(\mu x) \overset{*}{\Longrightarrow} \nu y \text{ iff } (\bar{\lambda}(@, x) = y \ \& \ \bar{\delta}(@, x) \in F)].$$

To facilitate an inductive argument we will replace $\mathbb{F}$ by a larger forest $\mathbb{H}$. Let Q be the set of states of Π_M. Let $\mathbb{H}$ be the set of all trees of the form

$$y_0(y_1(\dots y_{-1}(s(x_0(\dots x_{-1}(e)\dots)))\dots)) \text{ or}$$
$$y_0(y_1(\dots y_{-1}(e)\dots))$$

where $x \in \Sigma_1^*$, $s \in Q$, $y \in \Delta_1^*$, and $e \in U \cup \{\dashv\}$, so that $\mathbb{F} \subseteq \mathbb{H}$. Trees in $\mathbb{H}$ need not be derived from trees of the form @(R) with $R \in (\Sigma \cup U)_\#$. The relation on $V_\#$ defined by

$$R \gg S \text{ iff } (R \in \mathbb{H} \ \& \ (\exists \varphi \longrightarrow \psi \in \mathbb{R})(\exists n \in \text{Dom } R)$$
$$[R/n = \varphi \ \& \ S = R(n \longleftarrow \psi)])$$

is actually a relation on $\mathbb{H}$, so we have an SRS

$$\mathfrak{S}^+ = (V, \mathbb{H}, \gg, \mathbb{R}).$$

Note that $(\Longrightarrow) = (\gg) \cap (\mathbb{F} \times \mathbb{H})$. By induction on lengths of strings we will show that each $x \in \Sigma_1^*$ has

(2) $$(\forall r \in Q)(\forall y \in \Delta_1^*)(\forall t \in Q)$$
$$[r(\mu x) \overset{*}{\gg} y_0(\dots y_{-1}(t(\dashv))\dots) \text{ iff } (\bar{\lambda}(r, x) = y \ \& \ \bar{\delta}(r, x) = t)].$$

For x = ( ) we wish to show that

$$r(\dashv) \overset{*}{\gg} y_0(\dots y_{-1}(t(\dashv))\dots) \text{ iff } ((\,) = y \ \& \ r = t).$$

By the construction of the sets L_1 and L_2 of rule-schemata, the only rules of the form $r(\dashv) \longrightarrow \psi$ have $\psi^{-1}(Q) = \emptyset$. Therefore $r(\dashv) \overset{*}{\gg} S$ with $S^{-1}(Q) \neq \emptyset$ can only be true if $S = r(\dashv)$. We have

$$\begin{aligned} r(\dashv) \overset{*}{\gg} y_0(\ldots y_{-1}(t(\dashv))\ldots) \text{ iff } & y_0(\ldots y_{-1}(t(\dashv))\ldots) = r(\dashv) \\ \text{iff } & (y = (\,) \;\&\; t = r), \end{aligned}$$

as desired.

To pass from x to $(a)\cdot x$ for $x \in \Sigma_1^*$, let $r \in Q$, $y \in \Delta_1^*$, $t \in Q$. We wish to show that

$$r(a(\mu x)) \overset{*}{\gg} y_0(\ldots y_{-1}(t(\dashv))\ldots) \text{ iff}$$

$$(\lambda(r,a)\cdot\overline{\lambda}(\delta(r,a), x) = y \;\&\; \overline{\delta}(\delta(r,a), x) = t).$$

Let $(b_0, \ldots, b_{-1}) = \lambda(r,a)$ and let $s = \delta(r,a)$. By the construction of the sets L_1 and L_2 of rule-schemata, we have

$$r(a(\mu x)) \overset{*}{\gg} y_0(\ldots y_{-1}(t(\dashv))\ldots) \text{ iff}$$

$$r(a(\mu x)) \gg b_0(\ldots b_{-1}(s(\mu x))\ldots) \overset{*}{\gg} y_0(\ldots y_{-1}(t(\dashv))\ldots)$$

$$\text{iff}$$

$$|\lambda(r,a)| \leq |y| \;\&\; (\forall j < |\lambda(r,a)|)(b_j = y_j) \;\&$$
$$s(\mu x) \overset{*}{\gg} y_{|\lambda(r,a)|}(\ldots y_{-1}(t(\dashv))\ldots)$$

By the induction hypothesis we therefore have

$$r(a(\mu x)) \overset{*}{\gg} y_0(\ldots y_{-1}(t(\dashv))\ldots) \text{ iff}$$

$$|\lambda(r,a)| \leq |y| \;\&\; (\forall j < |\lambda(r,a)|)(b_j = y_j) \;\&\; \overline{\lambda}(s,x) = (y_{|\lambda(r,a)|}, \ldots, y_{-1}) \;\&\; \overline{\delta}(s,x) = t$$

iff

$$\lambda(r,a)\cdot\overline{\lambda}(s,x) = y \;\&\; \overline{\delta}(s,x) = t,$$

as desired.

We have shown that each $x \in \Sigma_1{}^*$ satisfies (2). By the construction of the sets L_1 and L_2 of rule-schemata, (2) implies

(3) $(\forall r \in Q)(\forall y \in \Delta_1{}^*)$

$$[r(\mu x) \overset{*}{\gg} y_0(\ldots y_{-1}(\dashv)\ldots) \text{ iff } (\overline{\lambda}(r,x) = y \;\&\; \overline{\delta}(r,x) \in F)].$$

Applying (3) with $r = @$ to every $x \in \Sigma_1{}^*$ yields (1).∎

Following Thatcher [52, §6], we have phrased inductions on lengths of strings in terms of passing from x to (a)·x rather than in terms of passing from x to x·(a) as is more common in formal language theory. This change is an attempt to smooth the transition from string transducers to tree transducers. Arguments by induction on sizes of trees pass from $R_0, \ldots, R_{-1}$ to $a(R_0, \ldots, R_{-1})$.

Another feature of the preceding proof that will appear again later is the use of a forest $\mathbb{H}$ that includes $\mathbb{F}$ but is better suited to inductive arguments. The relation $\Longrightarrow$ on $\mathbb{F}$ in each TDFTT is extended to the relation $\gg$ on $\mathbb{H}$ defined by applications of the same set of rules. In the general case, $\mathbb{H}$ is defined by two formal properties: nodes properly descended from nodes labelled by states are labelled by input symbols or parameters, while nodes not descended from nodes labelled by states are labelled by output symbols. The ad hoc introduction of $\mathbb{H}$ in the previous proof is a specific example.

(14.3) Definition. Let $\Pi = (\Sigma, \Delta, @, \mathfrak{C})$ with $\mathfrak{C} = (V, \mathbb{F}, \Longrightarrow, \mathbb{R})$ be a TDFTT. Let Q be the set of states. Define $\mathbb{H}$ to be the set of all $S \in V_{\#}$ such that

(1) $$\operatorname{Dom} S \cap (S^{-1}(Q)\cdot(0)\cdot\mathbb{N}^*) \subseteq S^{-1}(\Sigma \cup U)$$

and

(2) $$\operatorname{Dom} S - (S^{-1}(Q)\cdot\mathbb{N}^*) \subseteq S^{-1}(\Delta)\,.$$

Let $\gg$ be the set of all $\langle R, S\rangle \in V_{\#} \times V_{\#}$ such that

$$R \in \mathbb{H} \;\&\; (\exists \varphi \longrightarrow \psi \in \mathbb{R})(\exists n \in \operatorname{Dom} R)\,[R/n = \varphi \;\&\; S = R(n \longleftarrow \psi)]$$

and set

(3) $$\mathfrak{C}^+ = (V, \mathbb{H}, \gg, \mathbb{R})\,.$$

Define $\langle \Pi\rangle^+ : 2^{\mathbb{H}} \longrightarrow 2^{\mathbb{H}}$ by

(4) $$\langle \Pi\rangle^+ \mathcal{R} = \{T \in \mathbb{H} \,|\, (\exists R \in \mathcal{R})(R \overset{*}{\gg} T) \;\&\; T^{-1}(Q)\cdot(0) = T^{-1}(U)\}.$$

By parts (1) and (2) of Lemma 13.14, we do have $\mathbb{F} \subseteq \mathbb{H}$. By the arguments used for the induction steps in parts (1) and (2) of Lemma 13.14, $\gg$ is actually a relation on $\mathbb{H}$: the result of applying a rule at a node in a tree $S \in \mathbb{H}$ is another tree $S' \in \mathbb{H}$. Thus (3) above defines an SRS that is essentially the same as $\mathfrak{C}$ but is somewhat more amenable to inductive proofs. Note that $(\Longrightarrow) = (\gg) \cap (\mathbb{F} \times \mathbb{H})$. The definition of $\langle \Pi \rangle^+$ in (4) above resembles the definition of $\langle \Pi \rangle$ in (13.3.2):

$$\langle \Pi \rangle \mathcal{R} = \{T \in \Delta_{\#} \mid (\exists R \in \mathcal{R})(@(R) \overset{*}{\Longrightarrow} T)\}.$$

In (13.3.2) we consider normal forms T for @(R) that are irreplaceable because $T \in \Delta_{\#}$. In (4) above we consider normal forms T for R that are irreplaceable because the set $T^{-1}(Q)$ of tape head positions corresponds to the set of all fathers of nodes labelled by parameters. The heads have moved downward as far as they can before specific input trees are substituted for the parameters.

The string transduction equation $\langle M \rangle x = \overline{\lambda}(s, x)$ for $x \in \Sigma^*$ with $\overline{\delta}(s, x) \in F$ has the analogue $\langle \Pi \rangle \{R\} = \langle \Pi \rangle^+ \{@(R)\}$ for $R \in \Sigma_{\#}$. The following lemma establishes this and other useful technical properties of $\langle \Pi \rangle^+$.

(14.4) Lemma. Let $\Pi = (\Sigma, \Delta, @, \mathfrak{C})$ be a TDFTT with natural presentation $\langle (\Sigma \cup U)_{\#}, L \rangle$.

(1) $(\forall R \in \Sigma_{\#})(\langle \Pi \rangle \{R\} = \langle \Pi \rangle^+ \{ @(R)\})$

(2) $(\forall R \in \mathbb{H})(\langle \Pi \rangle^+ \{R\}$ is finite and effectively computable from R and L)

(3) (Π partial deterministic) implies $(\forall R \in \mathbb{H})(|\langle \Pi \rangle^+ \{R\}| \leq 1)$

(4) (Π total) implies $(\forall R \in \mathbb{H})(\langle \Pi \rangle^+ \{R\} \neq \emptyset)$.

Finally, suppose Π is linear and set

$$\mathbb{L} = \{S \in \mathbb{H} \mid (\forall w \in U)(|S^{-1}(w)| \leq 1)\}.$$

Then

(5) $$(\forall R \in \mathbb{L})(\langle \Pi \rangle^{+}\{R\} \subseteq \mathbb{L}).$$

<u>Proof:</u> For any $R \in \Sigma_{\#}$ we have

$$\begin{aligned}\langle \Pi \rangle \{R\} &= \{T \in \Delta_{\#} \mid @(R) \overset{*}{\Longrightarrow} T\} \\ &= \{T \in \mathbb{H} \mid T \in \Delta_{\#} \ \& \ @(R) \overset{*}{\gg} T\} \\ &= \{T \in \mathbb{H} \mid T^{-1}(Q) = \emptyset \ \& \ @(R) \overset{*}{\gg} T\},\end{aligned}$$

where $T^{-1}(Q) = \emptyset$ implies $T \in \Delta_{\#}$ by (2) in Definition 14.3. Since no parameters appear in R, each T with $@(R) \overset{*}{\gg} T$ has $T^{-1}(U) = \emptyset$ and so $T^{-1}(Q) = \emptyset$ iff $T^{-1}(Q)\cdot(0) = T^{-1}(U)$. The equation becomes

$$\langle \Pi \rangle \{R\} = \{T \in \mathbb{H} \mid T^{-1}(Q)\cdot(0) = T^{-1}(U) \ \& \ @(R) \overset{*}{\gg} T\} = \langle \Pi \rangle^{+}\{@(R)\}.$$

This proves (1).

By repeating the proof of Lemma 13.13 for TDFTTs with $\mathbb{H}$ in place of $\mathbb{F}$ and with $\gg$ in place of $\Longrightarrow$, we can show that each $R \in \mathbb{H}$ has a finite nonempty set $\mathfrak{N}$ of normal forms in $\mathfrak{C}^{+}$ and that this set is effectively computable from R and L. To compute $\langle \Pi \rangle^{+}\{R\}$ we have but to select out $\{T \in \mathfrak{N} \mid T^{-1}(Q)\cdot(0) = T^{-1}(U)\}$. This proves (2).

Now suppose Π is partial deterministic. By the same argument used in proving Lemma 13.11 for TDFTTs, $\mathfrak{C}^{+}$ is Church-Rosser. Each $R \in \mathbb{H}$ has $\langle \Pi \rangle^{+}\{R\} \subseteq \mathfrak{N}$ with $|\mathfrak{N}| \leq 1$, where $\mathfrak{N}$ is the set of normal forms for R in $\mathfrak{C}^{+}$. This proves (3).

Now suppose Π is total. The set $\mathfrak{N}$ of normal forms for R in $\mathfrak{C}^{+}$ is nonempty, so (4) will follow from

$$(\forall T \in \mathcal{N})(T^{-1}(Q)\cdot(0) = T^{-1}(U)).$$

Suppose $T \in \mathcal{N}$ but $T^{-1}(Q)\cdot(0) \neq T^{-1}(U)$. By (2) in Definition 14.3, $T^{-1}(U) \subseteq T^{-1}(Q)\cdot\mathbb{N}^*$, so that this supposition implies that some $n \in T^{-1}(Q)$ has $T(n\cdot(0)) \notin U$ and therefore $T(n\cdot(0)) \in \Sigma$ by (1) in Definition 14.3. Let $s = Tn$ and $a = T(n\cdot(0))$. Since Π is total there is a rule-schema

$$s(a(u_0, \ldots, u_{-1})) \longrightarrow \Psi \in L$$

with an instance

(6) $\quad s(a(P_0, \ldots, P_{-1})) \longrightarrow \Psi(\Psi^{-1}(u_0) \longleftarrow P_0)\ldots(\Psi^{-1}(u_{-1}) \longleftarrow P_{-1})$

in $\mathbb{R}$ for any $P_0, \ldots, P_{-1} \in (\Sigma \cup U)_\#$. In particular, let $P_k = T/(n\cdot(0,k))$ for all $k < \rho(a)$, so that $\mathrm{Dom}\, T \cap (T^{-1}(Q)\cdot(0)\cdot\mathbb{N}^*) \subseteq T^{-1}(\Sigma \cup U)$ (from (1) in Definition 14.3) implies $P_k \in (\Sigma \cup U)_\#$. The rule (6) is applicable at n in T, contradicting irreplaceability. This completes the proof of (4).

Now suppose Π is linear. To prove (5) it will suffice to show that, whenever $S \gg T$ and $S \in \mathbb{L}$, then $T \in \mathbb{L}$. Let $S \gg T$ by application of a rule $\varphi \longrightarrow \psi$ at a node n in S.

By $\varphi = S/n$, $\varphi \in \mathbb{L}$. From $\varphi \in \mathbb{L}$ and the definition of linearity (13.9), it follows that $\psi \in \mathbb{L}$. For any $w \in U$ we show that $|T^{-1}(w)| \leqslant 1$.

<u>Case 1</u> $(\psi^{-1}(w) \neq \emptyset)$. Then $\varphi^{-1}(w) \neq \emptyset$, so $\{p \in S^{-1}(w) \mid p \perp n\} = \emptyset$ by $S \in \mathbb{L}$ and $n \cdot \varphi^{-1}(w) \subseteq S^{-1}(w)$. Therefore

$$\begin{aligned} T^{-1}(w) &= n \cdot \psi^{-1}(w) \cup \{p \in T^{-1}(w) \mid p \perp n\} \\ &= n \cdot \psi^{-1}(w) \cup \{p \in S^{-1}(w) \mid p \perp n\} \\ &= n \cdot \psi^{-1}(w) \end{aligned}$$

and $|T^{-1}(w)| = |\psi^{-1}(w)| \leqslant 1$.

<u>Case 2</u> ($\psi^{-1}(w) = \emptyset$). Then

$$\begin{aligned} T^{-1}(w) &= \{p \in T^{-1}(w) \,|\, p \perp n\} \\ &= \{p \in S^{-1}(w) \,|\, p \perp n\} \subseteq S^{-1}(w) \end{aligned}$$

and $|T^{-1}(w)| \leqslant |S^{-1}(w)| \leqslant 1$.∎

The basic apparatus for studying closure under composition for classes of TDFTT transductions is now at hand. The next step is to generalize Definition 14.1 to a product construction for TDFTTs. To simplify the statement of the following definition, let us suppose we are given TDFTTs Π_1 and Π_2 where

$$\Pi_i = (\Sigma_i, \Delta_i, @_i, \mathfrak{C}_i) \quad \& \quad \mathfrak{C}_i = (V_i, \mathbb{F}_i, \underset{i}{\Longrightarrow}, \mathbb{R}_i).$$

Let these transducers have state sets Q_i and natural presentations $\langle \mathbb{D}_i, L_i \rangle$. Suppose also that $\Delta_1 = \Sigma_2$, so that $2^{\Delta_1 \#} = 2^{\Sigma_2 \#}$ and it is reasonable to try to define a product transducer $\Pi_2 \wedge \Pi_1$. Since the states of $\Pi_2 \wedge \Pi_1$ must be markers in our formalism, but they should correspond to pairs of states by analogy with Definition 14.1, we choose any bijection

$$\text{Codepair: } \mathcal{M} \times \mathcal{M} \longrightarrow \mathcal{M}$$

and abbreviate Codepair($\langle s_2, s_1 \rangle$) as $[s_2, s_1]$. Thus $[s_2, s_1]$ is a marker that represents the pair $\langle s_2, s_1 \rangle$ of markers.

For $r_2 \in Q_2$ and $r_1 \in Q_1$ and $a \in \Sigma_1$, we must define rule-schemata with the left half $[r_2, r_1](a(u_0, \ldots, u_{-1}))$. As with string transducers, we first apply Π_1 with state r_1 to $a(u_0, \dots, u_{-1})$, getting the set of all Ψ such that $r_1(a(u_0, \dots, u_{-1})) \longrightarrow \Psi \in L_1$. This set can be succinctly described as $\langle \Pi_1 \rangle^+ \{r_1(a(u_0, \dots, u_{-1}))\}$. Given any Ψ in this set, we apply Π_2 with state r_2 to Ψ. Since Ψ may include nodes labelled in Q_1 as well as Σ_2 and U, this is not strictly possible. But if we simply relabel Ψ by replacing each $s_1 \in Q_1$ by a new symbol $\hat{s}_1$ of rank 1, then the resulting tree $\hat{\Psi}$ can be processed by $\hat{\Pi}_2$, a transducer similar to Π_2 whose input vocabulary consists of Σ_2 together with all the new symbols. In addition to the rule-schemata in L_2, we let $\hat{\Pi}_2$ have all schemata

$$s_2(\hat{s}_1(u_0)) \longrightarrow [s_2, s_1](u_0)$$

for $s_2 \in Q_2$ and $s_1 \in Q_1$, so that $\langle \hat{\Pi}_2 \rangle^+ \{r_2(\hat{\Psi})\}$ is a set of trees $\Omega \in \hat{\mathbb{H}}_2$ where $\hat{\mathbb{H}}_2$ is assigned to $\hat{\Pi}_2$ by Definition 14.3. Each

$$[r_2, r_1](a(u_0, \dots, u_{-1})) \longrightarrow \Omega$$

formed in this way is a rule-schema because parameters occur only at leaves in Ω and the only parameters that might occur are $u_0, \dots, u_{-1}$. The set L_3 of all such schemata specifies an SRS transducer Π_3, which will prove to be a TDFTT. We will define $\Pi_2 \wedge \Pi_1$ to be this TDFTT.

(14.5) Definition. Letting $\hat{s}_1$ be a new symbol of rank 1 for each $s_1 \in Q_1$, and $\hat{Q}_1 = \{\hat{s}_1 \mid s_1 \in Q_1\}$, set

(1) $\hat{L}_2 = L_2 \cup \{s_2(\hat{s}_1(u_0)) \longrightarrow [s_2, s_1](u_0) \mid s_2 \in Q_2 \;\&\; s_1 \in Q_1\}$

and let $\hat{\Pi}_2$ be the unique TDFTT of the form

(2) $\hat{\Pi}_2 = (\Sigma_2 \cup \hat{Q}_1, \Delta_2, @_2, \hat{\mathfrak{C}}_2)$

with natural presentation $\langle(\Sigma_2 \cup \hat{Q}_1 \cup U)_\#, \hat{L}_2\rangle$. For any $\Psi \in (\Delta_1 \cup Q_1 \cup U)_\#$, let $\hat{\Psi}$ result from replacing each label in Q_1 by the corresponding label in $\hat{Q}_1$. Now set

$$(3) \quad L_3 = \{[r_2, r_1](a(u_0, \ldots, u_{-1})) \longrightarrow \Omega \mid r_2 \in Q_2 \ \&\ r_1 \in Q_1 \ \&\ a \in \Sigma_1 \ \& \\ (\exists \Psi \in \langle \Pi_1\rangle^+\{r_1(a(u_0, \ldots, u_{-1}))\})(\Omega \in \langle \hat{\Pi}_2\rangle^+\{r_2(\hat{\Psi})\})\}$$

and let $\Pi_2 \wedge \Pi_1$ be the SRS transducer

$$(4) \quad \Pi_2 \wedge \Pi_1 = (\Sigma_1, \Delta_2, [@_2, @_1], \mathfrak{T}_3)$$

where the set of rules specifying $\mathfrak{T}_3$ is the set of all instances of members of L_3 (with domain $D_w = (\Sigma_1 \cup U)_\#$ for each $w \in U$).

(14.6) <u>Example</u>. Let $\Sigma = \{a, b, c\}$ be a ranked alphabet with $\rho(a) = 2$; $\rho(b) = 1$; $\rho(c) = 0$. Let @, \$, ¢ be three markers. Writing u_0, u_1 as u, v for legibility, set

$$L_1 = \{ \begin{array}{ll} @(a(u,v)) \longrightarrow a(\$(u), ¢(v)) & , \\ \$(a(u,v)) \longrightarrow a(\$(u), \$(v)) & , \\ \$(b(u)) \longrightarrow b(\$(u)) & , \\ \$(c) \longrightarrow c & , \\ ¢(c) \longrightarrow c & \} \end{array}$$

and

$$L_2 = \{ \begin{array}{ll} @(a(u,v)) \longrightarrow \$(u) & , \\ \$(a(u,v)) \longrightarrow a(\$(u), \$(v)) & , \\ \$(b(u)) \longrightarrow b(\$(u)) & , \\ \$(c) \longrightarrow c & \}. \end{array}$$

For $i = 1, 2$ let $\Pi_i = (\Sigma, \Sigma, @, \mathfrak{C}_i)$, with $\mathfrak{C}_i = (V, \mathbb{F}_i, \underset{i}{\Rightarrow}, \mathbb{R}_i)$, be the TDFTT with natural presentation $\langle (\Sigma \cup U)_{\#}, L_i \rangle$.

We apply Definition 14.5 to this example. The new input symbols for $\hat{\Pi}_2$ are $\hat{@}$, $\hat{\$}$, $\hat{¢}$, so that (14.5.1) becomes

$$\begin{aligned} \hat{L}_2 = L_2 \cup \{ & @(\hat{@}(u)) \longrightarrow [@, @](u), \\ & \$(\hat{@}(u)) \longrightarrow [\$, @](u), \\ & \quad \vdots \\ & \$(\hat{¢}(u)) \longrightarrow [\$, ¢](u) \ \}. \end{aligned}$$

The TDFTT from (14.5.2) is

$$\hat{\Pi}_2 = (\Sigma \cup \{\hat{@}, \hat{\$}, \hat{¢}\}, \Sigma, @, \hat{\mathfrak{C}}_2)$$

with natural presentation $\langle (\Sigma \cup \{\hat{@}, \hat{\$}, \hat{¢}\} \cup U)_{\#}, \hat{L}_2 \rangle$.

In (14.5.3) the set L_3 is the union of 18 sets of schemata, one set for each $[r_2, r_1](f(u_0, \ldots, u_{-1}))$ with $r_2 \in \{@, \$\}$; $r_1 \in \{@, \$, ¢\}$; $f \in \{a, b, c\}$. For $[\$, @](a(u, v))$ the set of all relevant Ω is

$$\begin{aligned} \bigcup_{\Psi \in \langle \Pi_1 \rangle^+ \{ @(a(u,v)) \}} \langle \hat{\Pi}_2 \rangle^+ \{ \$(\hat{\Psi}) \} &= \langle \hat{\Pi}_2 \rangle^+ \{ \$(a(\hat{\$}(u), \hat{¢}(v))) \} \\ &= \langle \hat{\Pi}_2 \rangle^+ \{ a(\$(\hat{\$}(u)), \$(\hat{¢}(v))) \} \\ &= \{ a([\$, \$](u), [\$, ¢](v)) \}. \end{aligned}$$

Thus we have a rule-schema

$$[\$, @](a(u, v)) \longrightarrow a([\$, \$](u), [\$, ¢](v))$$

in L_3. As in this example, each $[r_2, r_1](f(u_0, \ldots, u_{-1}))$ leads to at most one schema. The complete list follows.

$$\begin{array}{lll}
[@,@](a(u,v)) & \longrightarrow & [\$,\$](u) \\
[\$,@](a(u,v)) & \longrightarrow & a([\$,\$](u),[\$,¢](v)) \\
[@,\$](a(u,v)) & \longrightarrow & [\$,\$](u) \\
[\$,\$](a(u,v)) & \longrightarrow & a([\$,\$](u),[\$,\$](v)) \\
[\$,\$](b(u)) & \longrightarrow & b([\$,\$](u)) \\
[\$,\$](c) & \longrightarrow & c \\
[\$,¢](c) & \longrightarrow & c
\end{array}$$

The set Q_3 of $\Pi_2 \wedge \Pi_1$ is defined by (13.12.1) as

$$\begin{aligned} Q_3 &= \{s \in \mathcal{M} \mid (\exists\, \Phi \longrightarrow \Psi \in L)(\Psi^{-1}(s) \neq \emptyset)\} \cup \{[@,@]\} \\ &= \{[\$,\$], [\$,¢], [@,@]\}. \end{aligned}$$

Thus Q_3 corresponds to a proper subset of $Q_2 \times Q_1$ in this example.

The SRS transducer $\Pi_2 \wedge \Pi_1$ inherits many properties from Π_2 and Π_1. It is a TDFTT, is linear, and is partial deterministic. Although one may be tempted to add that $\langle \Pi_2 \wedge \Pi_1 \rangle = \langle \Pi_2 \rangle \circ \langle \Pi_1 \rangle$ "by the obvious induction," that temptation must be resisted. As the following theorem shows, $\langle \Pi_2 \rangle \circ \langle \Pi_1 \rangle$ is not computable by $\Pi_2 \wedge \Pi_1$ or any other TDFTT!

(14.7) Theorem. Let Σ be the ranked alphabet $\{a, b, c\}$ with $\rho(a) = 2$; $\rho(b) = 1$; $\rho(c) = 0$. For $i = 1, 2$ there are linear partial deterministic TDFTTs $\Pi_i = (\Sigma, \Sigma, @, \mathfrak{C}_i)$ such that, for each $R \in \Sigma_\#$,

(1) $\langle \Pi_2 \rangle (\langle \Pi_1 \rangle \{R\}) = \underset{\sim}{\text{if}}\ R(\,) = a \ \&\ R(1) = c\ \underset{\sim}{\text{then}}\ \{R/(0)\}\ \underset{\sim}{\text{else}}\ \emptyset$.

However, there is no TDFTT Π_3 such that, for each $R \in \Sigma_\#$,

(2) $\langle \Pi_3 \rangle \{R\} = \underset{\sim}{\text{if}}\ R(\,) = a\ \&\ R(1) = c\ \underset{\sim}{\text{then}}\ \{R/(0)\}\ \underset{\sim}{\text{else}}\ \emptyset.$

<u>Proof:</u> Let Π_1 and Π_2 be as in Example 14.6. For all $R \in \Sigma_\#$,

$$\langle \Pi_1 \rangle \{R\} = \underset{\sim}{\text{if}}\ R(\,) = a\ \&\ R(1) = c\ \underset{\sim}{\text{then}}\ \{R\}\ \underset{\sim}{\text{else}}\ \emptyset.$$

$$\langle \Pi_2 \rangle \{R\} = \underset{\sim}{\text{if}}\ R(\,) = a\ \underset{\sim}{\text{then}}\ \{R/(0)\}\ \underset{\sim}{\text{else}}\ \emptyset.$$

Combining these two equations yields (1).

Now suppose Π_3 is a TDFTT with root marker $@_3$ that satisfies (2) for all $R \in \Sigma_\#$. Let $\langle (\Sigma \cup U)_\#, L_3 \rangle$ be the natural presentation. Since $\{b, c\}_\#$ is infinite and L_3 is finite, it is possible to choose $T, T' \in \{b, c\}_\#$ and a rule-schema $@_3(a(u, v)) \longrightarrow \Omega \in L_3$ such that $T \neq T'$ and

(3) $\langle \Pi_3 \rangle \{a(T, c)\} = \{T\} = \langle \Pi_3 \rangle^+ \{\Omega(\Omega^{-1}(u) \longleftarrow T)(\Omega^{-1}(v) \longleftarrow c)\}$

as well as

$$\langle \Pi_3 \rangle \{a(T', c)\} = \{T'\} = \langle \Pi_3 \rangle^+ \{\Omega(\Omega^{-1}(u) \longleftarrow T')(\Omega^{-1}(v) \longleftarrow c)\}.$$

Therefore $\Omega(\Omega^{-1}(u) \longleftarrow T) \neq \Omega(\Omega^{-1}(u) \longleftarrow T')$ and so

(4) $\Omega^{-1}(u) \neq \emptyset.$

On the other hand,

$$\langle \Pi_3 \rangle \{a(T, b(c))\} = \emptyset = \langle \Pi_3 \rangle^+ \{\Omega(\Omega^{-1}(u) \longleftarrow T)(\Omega^{-1}(v) \longleftarrow b(c))\}.$$

Comparing this with (3) leads to

(5) $\Omega^{-1}(v) \neq \emptyset.$

By (4) and (5) and $u \neq v$, some node $n \in \text{Dom}\ \Omega$ has at least two sons.

But then $n \in \text{Dom } T$ in (3), and this contradicts $T \in \{b, c\}_{\#}$. ∎

Rounds [46, p. 267] discusses an example by Ogden of two linear partial deterministic TDFTT transductions whose composition is not a partial deterministic TDFTT transduction. He suggests that this composition is not even computable by a nondeterministic TDFTT [46, p. 272], but a proof would require much deeper insight than is presently available. Our Π_1 and Π_2 were inspired by Ogden's example but were chosen in such a way that a simple argument would show that $\langle \Pi_2 \rangle \circ \langle \Pi_1 \rangle$ is not a TDFTT transduction.

Definition 14.5 defines an SRS transducer $\Pi_2 \wedge \Pi_1$ whenever Π_1 and Π_2 are TDFTTs and the input vocabulary of Π_2 is the output vocabulary of Π_1. Theorem 14.7 shows that this product construction does not lead to closure under composition for linear partial deterministic TDFTT transductions, although linear partial deterministic TDFTTs are a natural generalization of deterministic finite string transducers, as was shown in Theorem 14.2. Fortunately, several other important classes of TDFTT transductions are closed under composition. Elaborating slightly upon [52, Lemma 6.9] and [46, pp. 266, 275], we have the following theorem.

(14.8) Composition Theorem. For $i = 1, 2$ let $\Pi_i = (\Sigma_i, \Delta_i, @_i, \mathfrak{C}_i)$ be TDFTTs with state sets Q_i. Suppose $\Delta_1 = \Sigma_2$, so that there is a product transducer $\Pi_2 \wedge \Pi_1$. Then $\Pi_2 \wedge \Pi_1$ is a TDFTT with a state set Q_3 such that

(1) $$|Q_3| \leq |Q_2| |Q_1|$$

and

(2) $(\Pi_1, \Pi_2$ linear) implies $(\Pi_2 \wedge \Pi_1$ linear)

(3) $(\Pi_1, \Pi_2$ partial deterministic) implies

$(\Pi_2 \wedge \Pi_1$ partial deterministic)

(4) $(\Pi_1, \Pi_2$ total) implies $(\Pi_2 \wedge \Pi_1$ total)

(5) $(\Pi_1, \Pi_2$ deterministic) implies $\langle \Pi_2 \wedge \Pi_1 \rangle = \langle \Pi_2 \rangle \circ \langle \Pi_1 \rangle$.

Proof: To show that $\Pi_2 \wedge \Pi_1$ is a TDFTT we will show that the set L_3 of rule-schemata in part (3) of Definition 14.5 is finite and satisfies the conditions (1) and (2) in Definition 13.5. For (13.5.1) we need

$$(\forall \Phi \longrightarrow \Omega \in L_3)(\exists s \in \mathcal{M})(\exists a \in \Sigma_1)[\Phi = s(a(u_0, \ldots, u_{-1}))].$$

But L_3 is a set of schemata

(6) $[r_2, r_1](a(u_0, \ldots, u_{-1})) \longrightarrow \Omega$

for $r_2 \in Q_2$, $r_1 \in Q_1$, and $a \in \Sigma_1$. Each $[r_2, r_1]$ is a marker, so (13.5.1) holds.

For (13.5.2) we must show that each Ω in (6) has

(7) $\Omega \in (\Delta_2 \cup \mathcal{M} \cup U)_\# \ \& \ \Omega^{-1}(\mathcal{M}) \cdot (0) = \Omega^{-1}(U)$.

Let $r_2 \in Q_2$, $r_1 \in Q_1$, and $a \in \Sigma_1$. By the definition of L_3 (14.5.3), we must show that (7) holds for any Ω such that

(8) $(\exists \Psi \in \langle \Pi_1 \rangle^+ \{r_1(a(u_0, \ldots, u_{-1}))\})(\Omega \in \langle \hat{\Pi}_2 \rangle^+ \{r_2(\hat{\Psi})\})$.

Suppose Ω satisfies (8). Let $\hat{\mathbb{H}}_2$ be the subset of $(\Sigma_2 \cup \hat{Q}_1 \cup \Delta_2 \cup \mathcal{M} \cup U)_\#$ assigned to $\hat{\Pi}_2$ by Definition 14.3, so that (8) and (14.3.4) imply $\Omega \in \hat{\mathbb{H}}_2$. By the condition (14.3.2) on $\hat{\mathbb{H}}_2$,

(9) $\quad \mathrm{Dom}\,\Omega - (\Omega^{-1}(\hat{Q}_2)\cdot\mathbb{N}^*) \subseteq \Omega^{-1}(\Delta_2)$

where $\hat{Q}_2$ is the state set for $\hat{\Pi}_2$. But (8) and (14.3.4) also imply that $\Omega^{-1}(\hat{Q}_2)\cdot(0) = \Omega^{-1}(U)$. Therefore (9) implies that $\Omega \in (\Delta_2 \cup \hat{Q}_2 \cup U)_\#$, while

$$\Omega^{-1}(\mathcal{M})\cdot(0) = \Omega^{-1}(\hat{Q}_2)\cdot(0) = \Omega^{-1}(U)\,.$$

This proves (7). The product transducer is indeed a TDFTT.

Suppose Ω satisfies (8). From $\Psi^{-1}(Q_1)\cdot(0) = \Psi^{-1}(U)$ and $\Omega^{-1}(\mathcal{M})\cdot(0) = \Omega^{-1}(U)$, it follows that, whenever a node $n \in$ Dom Ω has $\Omega n = w \in U$, then a rule of the form

$$s_2(\hat{s}_1(w)) \longrightarrow [s_2, s_1](w)$$

must have been applied at Fa(n) in deriving Ω from $r_2(\hat{\Psi})$. Therefore

$$(\forall s_3 \in \mathcal{M})[\Omega^{-1}(s_3) \neq \emptyset \text{ implies } (\exists s_2 \in Q_2)(\exists s_1 \in Q_1)(s_3 = [s_2, s_1])]\,.$$

This proves (1).

Suppose Π_1 and Π_2 are linear, so that $\hat{\Pi}_2$ is linear also. For each Ω that satisfies (8), we have

$$(\forall w \in U)(|\hat{\Psi}^{-1}(w)| \leq 1)$$

by linearity of Π_1 and then

$$(\forall w \in U)(|\Omega^{-1}(w)| \leq 1)$$

by linearity of $\hat{\Pi}_2$ and (5) in Lemma 14.4. This proves (2).

Suppose Π_1 and Π_2 are partial deterministic, so that $\hat{\Pi}_2$ is partial deterministic also. For each choice of r_2, r_1, and a, part (3) in Lemma 14.4 implies that there is at most one Ψ in $\langle \Pi_1 \rangle^+ \{r_1(a(u_0,\ldots,u_{-1}))\}$ and then that there is at most one Ω in $\langle \hat{\Pi}_2 \rangle^+ \{r_2(\hat{\Psi})\}$. Therefore at most one Ω has

$$[r_2, r_1](a(u_0, \ldots, u_{-1})) \longrightarrow \Omega \in L_3 .$$

This proves (3). Similarly, part (4) in Lemma 14.4 implies (4).

The crux of the matter is (5), which is Lemma 6.9 of [52] expressed in our terminology. There is no need to repeat the very clear inductive proof in [52] here.∎

We noted in §13 that linear or 1-state deterministic TDFTTs are especially easy to implement. Thus the following special case of this theorem is of interest:

(14.9) <u>Corollary</u>. Let Π_1 and Π_2 be deterministic TDFTTs such that the output vocabulary of Π_1 is the input vocabulary of Π_2. Then $\Pi_2 \wedge \Pi_1$ is a deterministic TDFTT with $\langle \Pi_2 \wedge \Pi_1 \rangle = \langle \Pi_2 \rangle \circ \langle \Pi_1 \rangle$ and

(1) (Π_1, Π_2 are 1-state TDFTTs) implies ($\Pi_2 \wedge \Pi_1$ is a 1-state TDFTT)

(2) (Π_1, Π_2 linear) implies ($\Pi_2 \wedge \Pi_1$ linear).∎

The Composition Theorem (14.8) and its corollary (14.9) are restricted to top-down transducers. We conjecture that the product construction (14.5) can be turned upside down for bottom-up transducers in such a way that analogues of these results will hold.

The restriction to deterministic FTTs in $\langle \Pi_2 \wedge \Pi_1 \rangle = \langle \Pi_2 \rangle \circ \langle \Pi_1 \rangle$ does not appear serious for the application to compiling that we described in §12. Once an augmented ranked parse tree has passed through the lexical filter, it should be converted into just one coding tree by the semantic analyzer, with no further testing. The tree transducer should act like a string transducer in which every state is a final state. There appears to be no need to use a nontotal transducer in order to reject certain inputs.

Since the input to the semantic analyzer's FTT is a ranked parse tree, not an arbitrary ranked tree with nodes labelled by productions, there may well be many (state, symbol) combinations that never arise in semantic analysis. The corresponding schemata may be deleted from a deterministic FTT in order to replace it by a more compactly presented FTT without altering the transduction it specifies. Comparison of the negative theorem (14.7) with the Composition Theorem (14.8) suggests that such optimization should be postponed until the very end of the FTT design process.

Although the Composition Theorem (14.8) (and its conjectured bottom-up analogue) suffice for compiler design purposes, additional closure results would be of mathematical interest and might have applications to more complex tree-manipulating systems implicit in the use of natural language. Total linear TDFTT transductions appear to be closed under composition, regardless of determinism.

(14.10) <u>Conjecture</u>. If Π_1 and Π_2 are total linear TDFTTs and the output vocabulary of Π_1 is the input vocabulary of Π_2, then

$$\langle \Pi_2 \wedge \Pi_1 \rangle = \langle \Pi_2 \rangle \circ \langle \Pi_1 \rangle . \blacksquare$$

The author has written a proof of this conjecture similar to the reasoning in [52, Lemma 6.9] or [46, p. 275], but the argument in its present form is more tedious than the result warrants. The proof is most naturally written in the elegant algebraic style of Thatcher [52]. Unfortunately, his formalism seems essentially limited to FTTs, perhaps even to TDFTTs. As the definition of "production systems" [52, §9] illustrates, it is extremely difficult to deal with SRSs and rule-schemata in general without switching to our formalism. Our more explicit and intuitive notation from §4 can deal with all the SRSs discussed in this report and with the infinite transducers introduced by Rounds [45, p. 112] [46, §II.5], as well as with TDFTTs, but the resulting treatment of TDFTTs becomes awkward in the detailed proofs for theorems like the above or part (5) in the Composition Theorem (14.8). It should be possible to mix the arithmetic of trees and substitution in [52] with our §4; at present we can only say that anyone interested in the mathematics of trees would do well to learn both systems.

CHAPTER 6

EPILOGUE

15. Summary

Subtree replacement systems form a broad class of tree-manipulating systems that includes many of the special cases from logic, linguistics, and automata theory. Systems with the Church-Rosser property are appropriate for evaluation or translation processes: the end result of a complete sequence of applications of the rules does not depend on the order in which the rules were used. We derived sufficient conditions for the Church-Rosser property and applied them to two important tree-manipulating systems: the McCarthy calculus for recursive definitions and the full lambda calculus.

In Chapter 2 we defined two classes of abstract mathematical systems: general replacement systems and subtree replacement systems. We defined normal forms and indicated why uniqueness of normal forms is desirable in systems intended to define "values" or "meanings" for the objects they manipulate. We defined the Church-Rosser property and remarked that normal forms are unique in Church-Rosser systems. The theorems in Chapter 2 established sufficient conditions for the Church-Rosser property. In particular, the Main Theorem (5.6) asserted that every unequivocal closed subtree replacement system is Church-Rosser. The abstract approach in Chapter 2 avoided assumptions about the specific forms (or syntax) of the rules, so that the results were not restricted to one system or to a narrow class of systems.

In Chapter 3 we applied the theory of subtree replacement systems to recursive definitions. We proved that recursively defined functions are singlevalued despite the nondeterminism of the evaluation algorithm. The Evaluation Theorem (7.5) asserted that any two successful attempts to evaluate an expression must produce the same result, and that this result is invariant under replacement of subexpressions by new expressions that evaluate the same way as the subexpressions they replace. The Validity Theorem (8.4) asserted that the function specified by a recursive definition solves the definition (considered as a set of equations) and is the canonical solution: the solution that agrees with every other solution wherever it is finite. By using the theory from Chapter 2, we were able to derive these results without the severe restrictions on call-by-name that were imposed in the previously known special cases.

The classical Church-Rosser theorem for the full lambda calculus was proven in Chapter 4. We defined a subtree replacement system $\mathfrak{C}_\lambda$ that manipulated (tree structures of) lambda expressions. By lifting the replacement relation $\underset{\lambda}{\Longrightarrow}$ from $\mathfrak{C}_\lambda$ to a relation $>_\lambda$ between equivalence classes of lambda expressions, we defined a general replacement system $\mathfrak{B}_\lambda$ that performed lambda reductions without distinguishing between lambda expressions that are strongly alphaequivalent: alike except for the choices of bound variables. We expressed $\mathfrak{B}_\lambda$ as the union of two systems, $\mathfrak{B}_\beta$ and $\mathfrak{B}_{\eta\delta}$. We used the Commutativity Lemma (3.6) to show that $\mathfrak{B}_\beta$ commutes with $\mathfrak{B}_{\eta\delta}$.

Once $\mathfrak{B}_{\eta\delta}$ and $\mathfrak{B}_{\beta}$ had been shown to be Church-Rosser, we concluded that $\mathfrak{B}_{\lambda}$ was Church-Rosser by the Commutative Union Theorem (3.5). That $\mathfrak{B}_{\eta\delta}$ was Church-Rosser followed almost immediately from the Main Theorem (5.6). That $\mathfrak{B}_{\beta}$ was Church-Rosser could not be shown so easily. We introduced a new symbol σ and new subtree replacement systems $\mathfrak{C}_{\gamma}$, $\mathfrak{C}_{\sigma}$. We used the Main Theorem (5.6) to show that $\mathfrak{C}_{\gamma}$ and $\mathfrak{C}_{\sigma}$ were Church-Rosser. Then we showed that $\mathfrak{C}_{\gamma}$ and $\mathfrak{C}_{\sigma}$ could interact to simulate $\mathfrak{B}_{\beta}$.

In Chapter 5 we applied the theory of subtree replacement systems to tree transducers and described the role of these transducers in syntax-directed compiling. We analyzed compilation as a sequence of processes: lexical analysis, context-free parsing, lexical filtration, semantic analysis, and code generation. Finite tree transducers were defined in order to perform semantic analysis by converting trees with nodes labelled by context-free productions to trees with nodes labelled by code-building operations. We showed that a deterministic finite tree transducer maps each input tree to exactly one output tree. We described the practical significance of closure-under-composition theorems and elaborated slightly upon the known theorem that the maps defined by deterministic top-down finite tree transducers are closed under composition. Finally, we proved that the maps defined by partial deterministic linear top-down finite tree transducers are not closed under composition.

16. Further Research

This section surveys some possible directions for further research in the study of general and subtree replacement systems and their applications. We begin with problems that are stated or implied in the previous chapters. We conclude with a discussion of possible applications in the study of asynchronous parallel processing.

We have studied the Church-Rosser property as a means of obtaining uniqueness of normal forms. As we remarked in §1, another desirable property of general replacement systems is the finiteness of every sequence $R_0, R_1, \ldots$ such that $R_i \Longrightarrow R_{i+1}$ for all i. Are there any helpful abstract sufficient conditions for this property? In systems without this property, are there theorems that help ascertain whether a sequencing mechanism omits any normal forms?

Another problem in the general theory is to weaken the definitions of residue maps and closed subtree replacement systems. As we remarked in §5, some of the restrictions in the definitions are motivated less by the intuitive ideas than by technical considerations in the proof of the Main Theorem (5.6). Can the definitions be weakened in such a way that unequivocal closed systems will still be Church-Rosser, yet more systems will be closed?

The recursive definitions considered in Chapter 3 suggest many questions for further investigation. As is usual in the theory of computing, our study of recursion has no provision for "side effects" or for "self-modifying" programs where the data at one stage become the instructions executed at another stage. Side effects and self-modifying programs are so common in the practice of computing that it would surely be worthwhile to introduce them here. With or without these complications, there are several promising directions for further work.

Efficient evaluation of the functions specified by recursive definitions requires more than random application of rules when some parameters are called by name. For example, consider the rule-schemata

(1) $$\bar{f}(\bar{u}) \longrightarrow \bar{C}(\bar{=}(\bar{u}, \bar{0}) , \bar{1} , \bar{\times}(\bar{u}, \tilde{f}(\bar{\dot{-}}(\bar{u}, \bar{1}))))$$

and

(2) $$\bar{C}(\overline{yes}, \bar{u}, \bar{v}) \longrightarrow \bar{u}$$
$$\bar{C}(\overline{no}, \bar{u}, \bar{v}) \longrightarrow \bar{v}\,.$$

The recursive definition formed by (1) and (2) is almost the same as our example, the natural definition of the factorial, but we have used a call-by-name parameter in (1). This is obviously a poor choice, and an analysis of just why it is a poor choice may assist in making better choices in more complicated situations. In general one should evaluate subexpressions before passing them to procedures if the whole expression being evaluated will not have a finite value unless the subexpressions do. On the other hand, the branches of conditionals should be passed unevaluated, as in (2). A multitude of schemes for computing recursively defined functions can be imagined; further development of the theory in Chapter 3 should assist in formulating such schemes, establishing their validity, and analyzing their computational complexity.

The abstract theory of computational complexity for recursive functions may also have fruitful interactions with Chapter 3, at least in the case where the data space is the nonnegative integers and the given functions are successor and the test for equality. If we identify recursive definitions that differ only in the function letters and parameters used (just as we might identify ALGOL 60 programs that differ only in the names chosen for the identifiers and labels), then a model for the axioms of M. Blum [4] can be obtained. We measure the <u>size</u> of a recursive definition L by the number of nodes that appear in it:

$$\|L\| = \sum_{R \longrightarrow S \in L} (|R| + |S|) .$$

Under our convention of identifying definitions that are alphabetic variants of each other, there are only finitely many recursive definitions of each size, and they can be effectively enumerated. These are the axioms on sizes of machines [4, p. 258]. By enumerating pairs of the form (L, f) where L is a recursive definition and f is a function letter of rank 1 involved in L, we can define an "acceptable" Gödel indexing $\varphi_0, \varphi_1, \ldots$ of the partial recursive functions, in the sense of [4, p. 258]. If (L, f) is the k-th pair in the enumeration, let $\Phi_k(\xi)$ be η whenever η is the smallest integer such that

$$\overline{f(\overline{\xi})} \overset{\eta}{\Longrightarrow} \overline{\varphi_k(\xi)}$$

in the subtree replacement system $\mathfrak{C}_L$. We now have a Blum complexity measure [4, p. 261]. What other properties do these size and complexity measures have?

In Chapter 4 we showed that beta reduction in the lambda calculus is Church-Rosser with the help of two subtree replacement systems $\mathfrak{C}_\gamma$ and $\mathfrak{C}_\sigma$. For both of these systems it is quite easy to show that every sequence of applications of rules is finite and to compute normal forms. For any tree R in the forest $\mathbb{H}$ of lambda expressions augmented by σ, let ΓR be the normal form of R in $\mathfrak{C}_\gamma$ and let ΣR be the normal form of R in $\mathfrak{C}_\sigma$. For any tree R in the forest $\mathbb{F}$ of lambda expressions, let AR be an alphanormal tree that is strongly equivalent to R. The following algorithm seeks the normal form of any tree R under beta reduction:

```
(1)        R := AR
(2)        S := ΓR
(3)        if R = S then HALT else go to (4)
(4)        R := ΣS
(5)        go to (1).
```

(We omit various time-saving tricks, such as checking whether R = ΓR while computing ΓR, in order to keep the basic strategy clear.) The results of Chapter 4 imply that the final value of R is the unique (up to strong alphaequivalence) beta normal form of the initial value of R whenever the algorithm does halt. We conjecture that the algorithm omits no normal forms: it does halt eventually whenever the initial tree has a beta normal form.

In Chapter 5 we defined both top-down and bottom-up tree transducers, but the major theorems here and in our references were restricted to top-down transducers. This is unfortunate for the application to syntax-directed compilers, since parsers and code generators are frequently bottom-up. Do the properties of top-down transducers have bottom-up analogs?

By relaxing some of the restrictions on the forms of rules in finite tree transducers, it may be possible to build useful models for tree-manipulating processes in natural languages. We will remark on two such processes: transformational grammars and semantic interpreters.

Transformational grammars have been described informally by Chomsky [8] [9] [10] and formally by Ginsburg and Partee [18], who also present an extensive bibliography on the subject. A transformational grammar assigns two tree structures to each sentence it generates: a "deep structure" and a "surface structure" derived from the deep structure by "transformations" that map trees to trees. Some generalizations of top-down finite tree transducers that could be relevant to the study of transformational grammars have been discussed by Rounds [46, pp. 280-281].

Woods [55] [56] has proposed a theory of computational semantics wherein the deep structures of natural language sentences are translated into a "query language" whose sentences have operator-operand structures. The operators represent whatever subroutines are available for changing the system's data base, applying various functions and predicates to portions of the data base, and reporting the results of such computations. The "semantic rules" in this theory specify a transducer mapping natural language deep structures to query language operator-operand structures. The tree transducer used here is somewhat more complex than the transducers implicit in syntax-directed compilers for programming languages, but the intuitive ideas are similar. The theory of tree transducers sketched in Chapter 5 should eventually be extended to deal with Woods' tree transducer as well.

The whole-part theorems in §3 may be useful in showing the singlevaluedness of functions defined by algorithms or computer systems that allow asynchronous parallel processing. These systems appear to be representable by the "parallel program schemata" of Karp and Miller [25, §1]. After sketching this model for parallel computation, we will indicate how the theorems of §3 may be useful.

A parallel program schema consists of a "memory" M, a set A of "operators" together with functions $D : A \longrightarrow 2^M$ and $R : A \longrightarrow 2^M$ called "domain" and "range" maps, an "alphabet" Σ, and a "control automaton" $\mathfrak{C}$. The memory M is a finite set of "cells" imagined to contain data being manipulated. No assumptions about the size or shape of cells are made – anything from a flip-flop to a disk file might be involved. The set A of operators represents the processors that have access to the data stored in the memory. For each $a \in A$, the domain Da is the set of cells from which a fetches data and the range Ra is the set of cells where a stores results. The alphabet Σ has symbols $a_0, a_1, \ldots, a_{K_a}$ (with $K_a \geq 1$) for each operator a. We say that a has initiator a_0 and K_a terminators $a_1, \ldots, a_{K_a}$. The control automaton $\mathfrak{C}$ is a set Q of states together with a special starting state $s \in Q$ and a transition function

$$\delta : Q \times \Sigma \longrightarrow Q \quad \text{(partial)}$$

such that $\delta(q, a_i)$ is defined whenever a_i is a terminator. Intuitively, the operators a such that $\delta(q, a_0)$ is defined are the ones permitted to begin computing when control is in state q. Any process that has been initiated previously may terminate (unpredictably) at any time and send a terminator to the control. The terminator chosen permits the process to give the control a little information about the computation just completed.

An <u>interpreted</u> schema is a parallel program schema together with interpretations for the operators. We consider a set $\mathbb{D}$ of possible data to be stored in cells. To each $a \in A$ we assign a map F_a from possible assignments $c : Da \longrightarrow \mathbb{D}$ of values to the domain cells for a to possible assignments $d : Ra \longrightarrow \mathbb{D}$ of values to the range cells for a. To each $a \in A$ we also assign a map G_a from possible assignments $c : Da \longrightarrow \mathbb{D}$ of values to the domain cells for a to terminators for a.

A <u>configuration</u> for an interpreted schema is any triple $\alpha = (c,q,\mu)$ where $c : M \longrightarrow \mathbb{D}$ (the current contents of the memory cells), $q \in Q$ (the current state of the control), and μ is a map assigning to each $a \in A$ an <u>input queue</u>: a string of maps from Da into $\mathbb{D}$. An <u>initial</u> configuration is one where q is the starting state s and $\mu(a)$ is the empty queue for each $a \in A$. Computations are sequences of transitions from one configuration to another.

Let $\mathbb{B}$ be the set of all possible configurations for an interpreted schema. We will set up a general replacement system $\mathfrak{B} = (\mathbb{B}, \Longrightarrow)$ to represent the transitions between configurations. Karp and Miller define a partial function mapping $\mathbb{B} \times \Sigma$ into $\mathbb{B}$ (whose value at $\langle \alpha, \sigma \rangle$ is denoted $\alpha \cdot \sigma$). Restating the definition [25, Def. 1.5] here, we define $\alpha \cdot \sigma$ to be the unique $\beta \in \mathbb{B}$, if any, such that the following conditions hold:

Case 1 ($\sigma = a_0$ for some $a \in A$) Then $\alpha = (c, q, \mu)$ with $\delta(q, a_0)$ defined. We have $\beta = (c, \delta(q, a_0), \nu)$, where the queue map ν has $\nu(b) = \mu(b)$ for all $b \in A$ with $b \neq a$ and

$$\nu(a) = \mu(a) \cdot (c \upharpoonright Da),$$

so that the current contents of a's domain cells are added to the tail of a's previous input queue $\mu(a)$.

Case 2 ($\sigma = a_k$ for some $a \in A$ and $k > 0$) Then $\alpha = (c, q, \mu)$ with $\mu(a) = (\kappa) \cdot \eta$ for some $\kappa : Da \longrightarrow \mathbb{D}$ (a nonempty queue headed by κ) and $a_k = G_a(\kappa)$. We have $\beta = (d, \delta(q, a_k), \nu)$ where $d(m) = c(m)$ for all $m \in M - Ra$ and $d \upharpoonright Ra = F_a(\kappa)$, while $\nu(b) = \mu(b)$ for all $b \in A$ with $b \neq a$ and $\nu(a) = \eta$, so that the head of a's previous input queue $\mu(a)$ has been serviced and is now deleted.

(Various restrictions on the model are of interest in practical situations. For example, if $|\mathbb{D}|$ is large then only configurations with very short queues are reasonable. The general model is a convenient framework for expressing and comparing more restricted models.)

Each $\sigma \in \Sigma$ defines a relation $\underset{\sigma}{\Longrightarrow}$ on $\mathbb{B}$ by

$$\alpha \underset{\sigma}{\Longrightarrow} \beta \text{ iff } \alpha \cdot \sigma = \beta,$$

and so each $\sigma \in \Sigma$ defines a general replacement system $(\mathbb{B}, \underset{\sigma}{\Longrightarrow})$. One way to show that the final configuration after a halting computation is determined by the initial configuration is to show that the general replacement system

$$(\mathbb{B}, \Longrightarrow) = (\mathbb{B}, \bigcup_{\sigma \in \Sigma} \underset{\sigma}{\Longrightarrow})$$

is Church-Rosser, since there is a computation that starts at α and halts at β iff α has normal form β in $(\mathbb{B}, \Rightarrow)$. The Church-Rosser property for $(\mathbb{B}, \Rightarrow)$ is much weaker than "determinacy" as defined in [25, Def. 1.9], but it is enough to insure uniqueness of normal forms.

Since each $\underset{\sigma}{\Rightarrow}$ is a partial function on $\mathbb{B}$, each $(\mathbb{B}, \underset{\sigma}{\Rightarrow})$ is a Church-Rosser system. The union process is associative and commutative, so there is a multitude of ways to analyze $(\mathbb{B}, \Rightarrow)$ as a hierarchy of simpler systems, with the systems on each level being unions of systems on the level below. For each such analysis, Theorems 3.5 and 3.8 tell us that appropriately connected Church-Rosser parts form Church-Rosser wholes, while Lemmas 3.6 and 3.9 assist in showing that the parts are indeed appropriately connected.

We can therefore approach each specific interpreted schema with general tools applicable to any union of general replacement systems. Perhaps we can also use the general tools to show that any interpreted schema satisfying certain conditions leads to a Church-Rosser system, where the conditions are reasonably easy to verify in many practical situations. No such results are known at present.

APPENDIX A

Elementary Properties of the Lambda Calculus

By straightforward applications of the definitions in §9 and use of the basic algebraic identities from §4, we can verify a multitude of obvious properties of bound and free variables. For example,

$$F_x\lambda(y, S) = \underset{\sim}{\text{if}}\ x = y\ \underset{\sim}{\text{then}}\ \emptyset\ \underset{\sim}{\text{else}}\ (1)\cdot F_x S,$$

$$m\cdot B_x(R/m) \subseteq B_x R \cap m\cdot \mathbb{N}^*,$$

and so on. This appendix establishes some less obvious but still quite elementary properties of the lambda calculus.

First we verify the assertion (9.3.2) with

(A.1) <u>Lemma</u>. Let $R \in \mathbb{F}$; $m, n, p \in \text{Dom } R$; $m\ \underline{\text{anc}}\ n$; $m\ \underline{\text{anc}}\ p$. Then

$$n \underset{R}{\longleftrightarrow} p \text{ iff } n/m \underset{R/m}{\longleftrightarrow} p/m.$$

<u>Proof:</u> Suppose that $n \underset{R}{\longleftrightarrow} p$ and let $x = Rn = Rp$.

<u>Case 1</u> ($n, p \in F_x R$) Then n/m, $p/m \in F_x(R/m)$, so $n/m \underset{R/m}{\longleftrightarrow} p/m$.

<u>Case 2</u> (n, p are bound to some $q \in \text{Dom } R$)

<u>Case 2.1</u> ($m\ \underline{\text{anc}}\ q$) Then n/m and p/m are bound to q/m in R/m, so $n/m \underset{R/m}{\longleftrightarrow} p/m$.

<u>Case 2.2</u> (NOT $m\ \underline{\text{anc}}\ q$). If n/m were bound to a node ℓ in R/m, then n would be bound to $m\cdot\ell$ in R rather than to q. Therefore $n/m \in F_x(R/m)$. Similarly, $p/m \in F_x(R/m)$, and so $n/m \underset{R/m}{\longleftrightarrow} p/m$.

Now suppose that $n/m \xleftrightarrow[R/m]{} p/m$. Let $x = (R/m)(n/m) = (R/m)(p/m)$, so that $x = Rn = Rp$ also.

Case 1 $(n/m,\ p/m \in F_x(R/m))$

Case 1.1 $((\exists \ell \text{ anc } m)(R\ell = \lambda \ \&\ R(\ell\cdot(0)) = x))$ Let ℓ with ℓ anc m and $R\ell = \lambda$ and $R(\ell\cdot(0)) = x$ be chosen with $|\ell|$ maximal. Then n and p are bound to ℓ in R and so $n \xleftrightarrow[R]{} p$.

Case 1.2 $((\forall \ell \text{ anc } m)(R\ell \neq \lambda \text{ or } R(\ell\cdot(0)) \neq x))$ Then $n, p \in F_x R$ and so $n \xleftrightarrow[R]{} p$.

Case 2 (n/m and p/m are bound to some q in R/m) Then n and p are bound to $m\cdot q$ in R and so $n \xleftrightarrow[R]{} p$. ∎

Next we verify Lemma 9.5.

(A.2) Lemma. Let $R, R', S, S' \in \mathbb{F}$; $x, x' \in X$. Then

(1) $\lambda(x, S) \simeq \lambda(x', S')$ iff $(S \simeq S' \ \&\ F_x S = F_{x'} S')$

(2) $\gamma(R, S) \simeq \gamma(R', S')$ iff

$$[R \simeq R' \ \&\ S \simeq S' \ \&$$
$$(\forall y, y' \in X)(F_y R = F_{y'} R' \neq \emptyset \text{ implies } F_y S = F_{y'} S')].$$

Proof: Suppose $\lambda(x, S) \simeq \lambda(x', S')$, so that

$$\begin{aligned}(1)\cdot F_x S &= \{q \in \text{Dom}\ \lambda(x, S) \mid q \text{ is bound to ( ) in } \lambda(x, S)\}\\ &= \{q \in \mathbb{N}^* \mid q \xleftrightarrow[\lambda(x, S)]{} (0)\}\\ &= \{q \in \mathbb{N}^* \mid q \xleftrightarrow[\lambda(x', S')]{} (0)\} = (1)\cdot F_{x'} S'.\end{aligned}$$

Therefore $F_x S = F_{x'} S'$. We also have Frame S = Frame S′ because Frame $\lambda(x, S)$ = Frame $\lambda(x', S')$. We must show that $(\xleftrightarrow[S]{}) = (\xleftrightarrow[S']{})$.

Let $m, n \in \mathbb{N}^*$. By $\lambda(x, S) \simeq \lambda(x', S')$ and two applications of Lemma A.1:

$$\begin{aligned} m \xleftrightarrow[S]{} n \quad &\text{iff} \quad (1)\cdot m \xleftrightarrow[\lambda(x,S)]{} (1)\cdot n \\ &\text{iff} \quad (1)\cdot m \xleftrightarrow[\lambda(x',S')]{} (1)\cdot n \\ &\text{iff} \quad m \xleftrightarrow[S']{} n. \end{aligned}$$

Now suppose that $S \simeq S'$ and $F_x S = F_{x'} S'$. We have Frame $\lambda(x, S)$ = Frame $\lambda(x', S')$ because Frame S = Frame S'. We must show that $(\xleftrightarrow[\lambda(x,S)]{}) = (\xleftrightarrow[\lambda(x',S')]{})$. By symmetry, it will suffice to show that $(\xleftrightarrow[\lambda(x,S)]{}) \subseteq (\xleftrightarrow[\lambda(x',S')]{})$. Suppose $m \xleftrightarrow[\lambda(x,S)]{} n$.

<u>Case 1</u> ($m, n \in F_y \lambda(x, S)$ for some $y \in X$) Then $y \neq x$ and $m/(1), n/(1) \in F_y S$. By $S \simeq S'$ and two applications of Lemma A.1 we have $m \xleftrightarrow[\lambda(x',S')]{} n$.

<u>Case 2</u> (m, n are bound to some p in $\lambda(x, S)$)

<u>Case 2.1</u> ($p = (\,)$) Then $m/(1)$, $n/(1) \in F_x S$ and so $m/(1)$, $n/(1) \in F_{x'} S'$. Therefore m, n are bound to $(\,)$ in $\lambda(x', S')$ and $m \xleftrightarrow[\lambda(x',S')]{} n$.

<u>Case 2.2</u> ($p \neq (\,)$) Then (1) <u>anc</u> p and $m/(1), n/(1)$ are bound to $p/(1)$ in S. By $S \simeq S'$ and two applications of Lemma A.1 we have $m \xleftrightarrow[\lambda(x',S')]{} n$.

This proves (1).

Now suppose $\gamma(R, S) \simeq \gamma(R', S')$, so that Frame R = Frame R' and Frame S = Frame S' because Frame $\gamma(R, S)$ = Frame $\gamma(R', S')$. Using Lemma A.1 and $(\xleftrightarrow[\gamma(R,S)]{}) = (\xleftrightarrow[\gamma(R',S')]{})$, we can show that $(\xleftrightarrow[R]{}) = (\xleftrightarrow[R']{})$ and $(\xleftrightarrow[S]{}) = (\xleftrightarrow[S']{})$ just as in the proof of (1). We now have $R \simeq R'$ and

$S \simeq S'$. Suppose $y, y' \in X$ with $F_y R = F_{y'} R' \neq \emptyset$. Let $m \in F_y R$. Then

$$F_y S = \{n \in \mathrm{Dom}\, S \mid (1)\cdot n \xleftrightarrow[\gamma(R,S)]{} (0)\cdot m\}$$

$$= \{n \in \mathrm{Dom}\, S' \mid (1)\cdot n \xleftrightarrow[\gamma(R',S')]{} (0)\cdot m\} = F_{y'} S'.$$

Now suppose that $R \simeq R'$, $S \simeq S'$, and $F_y S = F_{y'} S'$ whenever $F_y R = F_{y'} R' \neq \emptyset$. Then Frame $\gamma(R, S)$ = Frame $\gamma(R', S')$ because Frame R = Frame R' and Frame S = Frame S'. We must show that $(\xleftrightarrow[\gamma(R,S)]{}) = (\xleftrightarrow[\gamma(R',S')]{})$. By symmetry, it will suffice to show $(\xleftrightarrow[\gamma(R,S)]{}) \subseteq (\xleftrightarrow[\gamma(R',S')]{})$. Suppose $m, n \in \mathbb{N}^*$ with $m \xleftrightarrow[\gamma(R,S)]{} n$. Then (0) <u>anc</u> m or (1) <u>anc</u> m, and similarly for n.

<u>Case 1</u> ((0) <u>anc</u> m)

<u>Case 1.1</u> ((0) <u>anc</u> n) By $R \simeq R'$ and two applications of Lemma A.1, $m \xleftrightarrow[\gamma(R',S')]{} n$.

<u>Case 1.2</u> ((1) <u>anc</u> n) Since ( ) is the only common ancestor of m and n and $\gamma(R, S)(\,) \neq \lambda$, we must have $m, n \in F_y \gamma(R, S)$ for some $y \in X$. Let $y' = R'(m/(0))$. Then $m/(0) \in F_{y'} R'$ by $R \simeq R'$ and $m/(0) \in F_y R$. Since $F_y R = F_{y'} R' \neq \emptyset$, we have $F_y S = F_{y'} S'$ and so $n/(1) \in F_{y'} S'$ because $n/(1) \in F_y S$. Thus

$$m/(0) \in F_{y'} R' \;\&\; n/(1) \in F_{y'} S'$$

$$m \in F_{y'} \gamma(R', S') \;\&\; n \in F_{y'} \gamma(R', S')$$

$$m \xleftrightarrow[\gamma(R',S')]{} n.$$

<u>Case 2</u> ((1) <u>anc</u> m)

Case 2.1 ((0) anc n) We have m $\xleftrightarrow[\gamma(R',S')]{}$ n by Case 1.2 and the symmetry of both $\xleftrightarrow[\cdot\cdot]{}$ relations.

Case 2.2 ((1) anc n) By $S \simeq S'$ and two applications of Lemma A.1, m $\xleftrightarrow[\gamma(R',S')]{}$ n. ∎

Finally, we must prove Lemma 9.9.

(A.3) Lemma. Let $i \in \{\beta, \eta, \delta\}$; $\varphi \longrightarrow \psi \in \mathbb{R}_i$; $\varphi \simeq \varphi'$. If $i = \beta$. let $\varphi' \in \mathbb{F}_0$ also. Then there is $\psi' \in \mathbb{F}$ such that $\psi \simeq \psi'$ and

(1) $\varphi' \longrightarrow \psi' \in \mathbb{R}_i$

(2) $(\forall y, y' \in X)(F_y\psi = F_{y'}\psi' \neq \emptyset$ implies $F_y\varphi = F_{y'}\varphi' \neq \emptyset)$.

Proof: Suppose first that $i = \delta$. Recall that $\mathbb{R}_\delta$ satisfies restrictions (4) and (5) in Definition 9.7. Then $\varphi \cong \varphi'$ because FVbl $\varphi = \emptyset$ in (9.7.4). Letting $\psi' = \psi$, we have (1) by (9.7.5). Since FVbl $\psi = \emptyset$ in (9.7.4), (2) is trivial.

Now suppose that $i = \eta$, so that some $R \in \mathbb{F}$ and $x \in X$ have

$$\varphi = \lambda(x, \gamma(R, x)) \ \& \ \psi = R \ \& \ F_xR = \emptyset$$

by the definition of $\mathbb{R}_\eta$ (9.7.3). By $\varphi \simeq \varphi'$ there are $R' \in \mathbb{F}$ and $x' \in X$ with

$$\varphi' = \lambda(x', \gamma(R', x')),$$

while Lemma A.2 implies that

$$R \simeq R' \ \& \ F_{x'}R' = F_xR = \emptyset.$$

Letting $\psi' = R'$ yields $\psi \simeq \psi'$ and (1). Now let $y, y' \in X$ and suppose $F_y\psi = F_{y'}\psi' \neq \emptyset$. Then $y \neq x$ and $y' \neq x'$, so

$$F_y\varphi = (1,0)\cdot F_yR = (1,0)\cdot F_{y'}R' = F_{y'}\varphi' \neq \emptyset.$$

Finally, suppose $i = \beta$, so that some $R, S \in \mathbb{F}$ and $x \in X$ have

$$\varphi = \gamma(\lambda(x, S), R) \;\&\; \psi = [R/x]S \;\&\; \mathrm{FVbl}\ R \cap (\mathrm{BVbl}\ S \cup \{x\}) = \emptyset$$

by the definition of $\mathbb{R}_\beta$ (9.7.2). By $\varphi \simeq \varphi'$ there are $R', S' \in \mathbb{F}$ and $x' \in X$ such that

$$\varphi' = \gamma(\lambda(x', S'), R')).$$

Set $\psi' = [R'/x']S'$. By $\varphi' \in \mathbb{F}_0$ we have $\mathrm{FVbl}\ R' \cap (\mathrm{BVbl}\ S' \cup \{x'\}) = \emptyset$ and (1) follows. Let $y.y' \in X$ and suppose $F_y\psi = F_{y'}\psi' \neq \emptyset$. We must show that $F_y\varphi = F_{y'}\varphi' \neq \emptyset$. Since $\mathrm{FVbl}\ R \cap \mathrm{BVbl}\ S = \mathrm{FVbl} R' \cap \mathrm{BVbl} S' = \emptyset$, we have

$$F_y\psi = (F_yS - F_xS) \cup F_xS\cdot F_yR$$

$$F_{y'}\psi' = (F_{y'}S' - F_{x'}S') \cup F_{x'}S' \cdot F_{y'}R'.$$

Combining these with $F_y\psi = F_{y'}\psi'$ yields

(3) $\quad (F_yS - F_xS) \cup F_xS\cdot F_yR = (F_{y'}S' - F_{x'}S') \cup F_{x'}S' \cdot F_{y'}R'.$

Since $F_xS = F_{x'}S'$ while $S^{-1}(X)$ and $S'^{-1}(X)$ are both independent,

$$F_yS - F_xS \perp F_{x'}S' \;\&\; F_{y'}S' - F_{x'}S' \perp F_xS.$$

Therefore

$$(F_yS - F_xS) \cap F_{x'}S'\cdot F_{y'}R' = \emptyset = (F_{y'}S' - F_{x'}S') \cap F_xS\cdot F_yR,$$

and (3) becomes

(4) $\quad F_yS - F_xS = F_{y'}S' - F_{x'}S' \;\&\; F_xS\cdot F_yR = F_{x'}S'\cdot F_{y'}R',$

where both equations cannot be $\emptyset = \emptyset$ because $F_y\psi \neq \emptyset$. There are two cases to consider.

<u>Case 1</u> ($F_xS \cdot F_yR = \emptyset$) Then (4) implies that $F_yS - F_xS = F_{y'}S' - F_{x'}S' \neq \emptyset$. Applying this to

$$F_y\varphi = (0, 1)\cdot(F_yS - F_xS) \cup (1)\cdot F_yR$$

and to the corresponding equation for $F_{y'}\varphi'$, we find that some $p \in \text{Dom } S$ has

$$(0, 1)\cdot p \in F_y\varphi \cap F_{y'}\varphi'.$$

But this implies that $F_y\varphi = F_{y'}\varphi' \neq \emptyset$ by $\varphi \simeq \varphi'$.

<u>Case 2</u> ($F_xS \cdot F_yR \neq \emptyset$) Then F_xS is a nonempty independent set of nodes with $F_xS = F_{x'}S'$. Any equation $M\cdot N = M\cdot P$ for $M, N, P \subseteq \mathbb{N}^*$ with M nonempty and independent implies $N = P$, so (4) implies that $F_yR = F_{y'}R'$. By $F_xS\cdot F_yR \neq \emptyset$,

$$F_yR = F_{y'}R' \neq \emptyset.$$

Applying this and (4) to

$$F_y\varphi = (0, 1)\cdot(F_yS - F_xS) \cup (1)\cdot F_yR$$

and to the corresponding equation for $F_{y'}\varphi'$, we find that $F_y\varphi = F_{y'}\varphi' \neq \emptyset$. This proves (2).

We must still show that $\psi \simeq \psi'$ in the first place! Let $K = |F_xS|$ and let $(m_0, \ldots, m_{K-1})$ be a listing of F_xS. We will use Lemma 9.6 to prove

(5) $\quad S(m_0 \longleftarrow R)\ldots(m_{-1} \longleftarrow R) \simeq S'(m_0 \longleftarrow R')\ldots(m_{-1} \longleftarrow R')$.

Comparing (5) with (9.6.4), we see that it suffices to show

$$F_x S = F_{x'} S'$$

$$\text{FVbl } R \cap (\text{BVbl } S \cup \{x\}) = \emptyset$$

$$\text{FVbl } R' \cap (\text{BVbl } S' \cup \{x'\}) = \emptyset$$

$$S \simeq S' \ \& \ R \simeq R'$$

(6) $(\forall y, y' \in X)(F_y S = F_{y'} S' \neq \emptyset$ implies $F_y R = F_{y'} R')$

$(\forall y, y' \in X)(F_y R = F_{y'} R' \neq \emptyset$ implies $F_y R = F_{y'} R')$.

Only (6) is neither trivial nor already proven. Suppose $F_y S = F_{y'} S' \neq \emptyset$. If $x = y$ then $F_{y'} S' = F_{x'} S' \neq \emptyset$, and so $x' = y'$. We have $F_y R = \emptyset = F_{y'} R'$. If $x \neq y$ then $x' \neq y'$ too and $F_y R = F_{y'} R'$ by Lemma A.2 and $\varphi \simeq \varphi'$. This proves (6).∎

APPENDIX B

Equivalence of Two Versions of the Lambda Calculus

As we noted at the end of §9, our formalization of the lambda calculus differs somewhat from that of Curry and Feys. Even after setting up the obvious correspondence between "obs" [14, Chap. 3] and trees in our forest $\mathbb{F}$, it may not be completely obvious that our $\mathfrak{B}_\lambda$ is the same GRS that Curry and Feys show is Church-Rosser [14, Chap. 4]. This appendix defines a GRS $\mathfrak{B}_\mu$ such that the statement

$$\mathfrak{B}_\mu \text{ is Church-Rosser}$$

<u>is</u> obviously equivalent to the classical Church-Rosser theorem [14, Chap. 4]. Next we show that $\mathfrak{B}_\mu = \mathfrak{B}_\lambda$, so that Theorem 10.12 does cover the classical theorem.

First we define a substitution operation that corresponds to [14, §$\underset{\sim}{3}$E1]:

(B.1) <u>Definition</u>. Choose any map Newvbl: $2^X - \{X\} \longrightarrow X$ such that

(1) $(\forall Y \subset X)(\text{Newvbl}(Y) \notin Y)$.

For all $R \in \mathbb{F}$ and $x \in X$, define

(2) $\{R/x\}: \mathbb{F} \longrightarrow \mathbb{F}$

inductively by setting

(3) $\{R/x\}\gamma(S, T) = \gamma\{R/x\}S, \{R/x\}T)$

(4) $\{R/x\}\lambda(x, S) = \lambda(x, S)$

(5) $\{R/x\}\lambda(y, S) = \underset{\sim}{\text{if}}\ F_x S = \emptyset$ or $F_y R = \emptyset$ $\underset{\sim}{\text{then}}\ \lambda(y, \{R/x\}S)$

$\underset{\sim}{\text{else}}\ \lambda(z, \{R/x\}\{z/y\}S)$

(6) $\{R/x\}x = R$ & $\{R/x\}a = a$

for all $S, T \in \mathbb{F}$; $y, z \in X$; $a \in C \cup X$ such that $y \neq x$ and $a \neq x$ and

$z = \text{Newvbl}(\text{FVbl}\ R \cup \text{FVbl}\ S \cup \{x\})$.

Note the similarity to the definition of $\mathbb{R}_\sigma$ (10.4.3), except in (5) above, where occurrences of y are changed to occurrences of z as needed to prevent captures of free variables. This complication in the definition of substitution for free variables permits a simple definition of beta-reduction [14, §$\underset{\sim}{3}$D3]:

(B.2.1) $\mathbb{R}_\theta = \{\gamma(\lambda(x, S), R) \longrightarrow \{R/x\}S \mid R, S \in \mathbb{F}\ \&\ x \in X\}$,

where we use "θ" rather than "β" to prevent confusion between this set of rules and our own $\mathbb{R}_\beta$. Letting

(B.2.2) $\mathbb{R}_\mu = \mathbb{R}_\theta \cup \mathbb{R}_\eta \cup \mathbb{R}_\delta$,

we have an SRS of the form

(B.2.d) $\mathfrak{C}_\mu = (V, \mathbb{F}, \underset{\mu}{\Longrightarrow}, \mathbb{R}_\mu)$

that carries out "reductions" just as in [14, Chap. 3].

The Church-Rosser property is only sought modulo an equivalence relation defined by yet another set of rules [14, §$\underset{\sim}{3}$D3]:

(B.3.1) $\mathbb{R}_\alpha = \{\lambda(x, S) \longrightarrow \lambda(y, \{y/x\}S) \mid S \in \mathbb{F}\ \&\ x, y \in X\ \&\ F_y S = \emptyset\}$.

These rules define an SRS of the form

(B.3.2) $\mathfrak{C}_\alpha = (V, \mathbb{F}, \underset{\alpha}{\Longrightarrow}, \mathbb{R}_\alpha)$

where $\overset{*}{\underset{\alpha}{\Longrightarrow}}$ corresponds to the relation of alpha-convertibility among "obs." Curry and Feys assert that $\overset{*}{\underset{\alpha}{\Longrightarrow}}$ is an equivalence relation because $\mathbb{R}_\alpha$ is symmetric [14, §3D3]. This is an error. If $x, y \in X$ with $x \neq y$ and if $z = \text{Newvbl}\,\{x, y\}$, then

$$\lambda(x, \lambda(y, x)) \longrightarrow \lambda(y, \lambda(z, y)) \in \mathbb{R}_\alpha$$

but

$$\lambda(y, \lambda(z, y)) \longrightarrow \lambda(x, \lambda(y, x)) \notin \mathbb{R}_\alpha .$$

The relation $\overset{*}{\underset{\alpha}{\Longrightarrow}}$ _is_ symmetric, as we will soon show by proving that it is actually the same as our strong alphaequivalence relation $\cong$. Thus the assertion in [14] is correct although the reason given for it is not. Using the fact that $\overset{*}{\underset{\alpha}{\Longrightarrow}}$ is an equivalence, we define Curry and Feys' GRS:

(B.4) _Definition._ Let $\mathbb{E}_\alpha$ be the set of all equivalence classes of trees in $\mathbb{F}$ under the relation $\overset{*}{\underset{\alpha}{\Longrightarrow}}$. Define a GRS $\mathfrak{B}_\mu = (\mathbb{E}_\alpha, \underset{\mu}{>})$ by setting, for all $\mathcal{R}, \mathcal{S} \in \mathbb{E}_\alpha$,

$$\mathcal{R} \underset{\mu}{>} \mathcal{S} \quad \text{iff} \quad (\exists R \in \mathcal{R})(\exists S \in \mathcal{S})(R \underset{\mu}{\Longrightarrow} S).$$

(B.5) _Lemma._ $(\overset{*}{\underset{\alpha}{\Longrightarrow}}) = (\cong)$.

Proof: First we will show

(1) $(\underset{\alpha}{\Longrightarrow}) \subseteq (\cong)$.

Suppose $R \underset{\alpha}{\Longrightarrow} S$. For some $Q \in \mathbb{F}$; $x, y \in X$; $m \in \text{Dom}\, R$ we have $F_y Q = \emptyset$ and

(2) $R/m = \lambda(x, Q)$

$S = R(m \longleftarrow \lambda(y, \{y/x\}Q))$.

By a straightforward induction on $|Q|$ in the definition (B.1) for $\{y/x\}Q$, we can show that $Q \simeq \{y/x\}Q$ and that

$$(\forall z \in X)(\forall n \in F_z Q)\,[(z = x \;\&\; (\{y/x\}Q)n = y) \text{ or } (z \neq x \;\&\; (\{y/x\}Q)n = z)\;].$$

By the definitions of $\simeq$ and $\cong$ (9.4) and by (1) in Lemma 9.5, these facts imply that

$$\lambda(x, Q) \cong \lambda(y, \{y/x\}Q).$$

Applying this to (2) and using Lemma 9.5 $|m|$ times, we find that $R \cong S$. This proves (1).

In order to show that $(\cong) \subseteq (\overset{*}{\underset{\alpha}{\Longrightarrow}})$, we will show that every $P \in \mathbb{F}$ satisfies

(3) $\quad (\forall P' \in \mathbb{F})(P \cong P' \text{ implies } P \overset{*}{\underset{\alpha}{\Longrightarrow}} P').$

Let $P \in \mathbb{F}$ and suppose all smaller trees in $\mathbb{F}$ satisfy (3). Suppose $P \cong P'$.

<u>Case 1</u> $(P(\;) \in C \cup X)$ Then $P = P'$ and $P \overset{*}{\underset{\alpha}{\Longrightarrow}} P'$.

<u>Case 2</u> $(P(\;) = \lambda)$ Then $P = \lambda(x, S)$ for some $S \in \mathbb{F}$; $x \in X$. Since Frame P = Frame P', we have $P' = \lambda(x', S')$ for some $S' \in \mathbb{F}$; $x' \in X$. Let $y \in X$ with $S^{-1}(y) = S'^{-1}(y) = \emptyset$, so that $P \cong P'$ implies that

$$[y/x]S \cong [y/x']S'$$

by the definition of $\cong$ (9.4.2) and by (1) in Lemma 9.5. By the induction hypothesis and the definition of $\mathfrak{C}_\alpha$ as an SRS,

(4) $\quad \lambda(y, [y/x]S) \overset{*}{\underset{\alpha}{\Longrightarrow}} \lambda(y, [y/x']S').$

By $S^{-1}(y) = \emptyset$ we have $[y/x]S = \{y/x\}S$ and $F_y S = \emptyset$, so that

(5) $$\lambda(x, S) \underset{\alpha}{\Longrightarrow} \lambda(y, [y/x]S).$$

Now consider any subtree of the form $\lambda(z, T)$ in $[y/x']S'$. If $z = x'$ then $T^{-1}(y) = \emptyset$ because $S'^{-1}(y) = \emptyset$ and $[y/x']S' = S'(F_{x'}S' \longleftarrow y)$. Therefore either $F_z x' = \emptyset$ or $F_y T = \emptyset$ and so

$$\{x'/y\}\lambda(z, T) = \lambda(z, \{x'/y\}T)$$

in (5) from Definition B.1. Since this is true for all subtrees of the form $\lambda(z, T)$ in $[y/x']S'$ and since replacing (B.1.5) with

$$\{R/x\}\lambda(y, S) = \lambda(y, \{R/x\}S)$$

would turn (B.1) into a characterization of our $[R/x]$ substitution, we have

(6) $$\{x'/y\}[y/x']S' = [x'/y][y/x']S'.$$

But $F_{x'}[y/x']S' = \emptyset$ while $[x'/y][y/x']S' = S'$ by $S'^{-1}(y) = \emptyset$, so (6) yields

$$\lambda(y, [y/x']S') \underset{\alpha}{\Longrightarrow} \lambda(x', S').$$

Combining this with (4) and (5), we get

$$P = \lambda(x, S) \overset{*}{\underset{\alpha}{\Longrightarrow}} \lambda(x', S') = P'.$$

<u>Case 3</u> $(P(\ ) = \gamma)$ Then $P = \gamma(R, S)$ and $P' = \gamma(R', S')$ for some $R, R', S, S' \in \mathbb{F}$ with $R \cong R'$ and $S \cong S'$ by (2) in Lemma 9.5. We apply the induction hypothesis twice to get $P \overset{*}{\underset{\alpha}{\Longrightarrow}} P'$. ∎

Lemma B.5 implies that the set $\mathbb{E}_\alpha$ in the definition (B.4) of $\mathfrak{B}_\mu$ is the same set $\mathbb{E}$ used to define $\mathfrak{B}_\lambda$. To complete the proof that

$\mathfrak{B}_\mu = \mathfrak{B}_\lambda$ we must show that $(\underset{\mu}{>}) = (\underset{\lambda}{>})$ on this set $\mathbb{E}$.

(B.6) <u>Theorem.</u> $\mathfrak{B}_\mu = \mathfrak{B}_\lambda$.

<u>Proof:</u> The proof of Lemma 9.6 can be adapted to show that

$$\{R/x\}S \simeq \{R'/x'\}S'$$

whenever $R \simeq R'$, $S \simeq S'$, $F_x S = F_{x'} S'$, and

$$(\forall y, y' \in X')(F_y S = F_{y'} S' \neq \emptyset \text{ implies } F_y R = F_{y'} R').$$

Therefore the proof of Lemma 9.9 given in (A.3) can be carried out with θ in place of β. Using Lemma 9.9 with θ in place of β, we may prove Lemma 10.1 with θ in place of β. Thus the corollary

(1) $(\forall \mathcal{R}, \mathcal{S} \in \mathbb{E})[\mathcal{R} \underset{\lambda}{>} \mathcal{S} \text{ iff } (\exists R \in \mathcal{R} \cap \mathbb{F}_0)(\exists S \in \mathcal{S})(R \underset{\lambda}{\Longrightarrow} S)]$

of Lemma 10.1 has the analog

(2) $(\forall \mathcal{R}, \mathcal{S} \in \mathbb{E})[\mathcal{R} \underset{\mu}{>} \mathcal{S} \text{ iff } (\exists R \in \mathcal{R} \cap \mathbb{F}_0)(\exists S \in \mathcal{S})(R \underset{\mu}{\Longrightarrow} S)]$

for $\mathfrak{B}_\mu$.

For all $R, S \in \mathbb{F}$ and $x \in X$ with $\text{FVbl}\, R \cap (\text{BVbl}\, S \cup \{x\}) = \emptyset$,

$$[R/x]S = \{R/x\}S,$$

and therefore

$$(\underset{\theta}{\Longrightarrow}) \cap (\mathbb{F}_0 \times \mathbb{F}) = (\underset{\beta}{\Longrightarrow}) \cap (\mathbb{F}_0 \times \mathbb{F}).$$

Combining this with the definitions of $\mathbb{R}_\lambda$ (9.7.1) and $\mathbb{R}_\mu$ (B.2.2) yields

$$(\underset{\mu}{\Longrightarrow}) \cap (\mathbb{F}_0 \times \mathbb{F}) = (\underset{\lambda}{\Longrightarrow}) \cap (\mathbb{F}_0 \times \mathbb{F}).$$

Therefore (1) and (2) imply that $(\underset{\mu}{>}) = (\underset{\lambda}{>})$. ∎

REFERENCES

1. Aho, A.V, Hopcroft, J.E., and Ullman, J.D. A general theory of translation. Math. Systems Theory 3 (1969), 193-221.

2. Aho, A.V., and Ullman, J.D. Translations on a context-free grammar. ACM Symp. on Theory of Computing (1969), 93-112.

3. Blum, E.K. Towards a theory of semantics and compilers for programming languages. J. Computer and System Sci. 3 (1969), 248-275.

4. Blum, M. On the size of machines. Inform. and Control 11 (1967), 257-265.

5. Bourbaki, N. Theory of Sets. Addison-Wesley, Reading, Mass., 1968.

6. Brainerd, W.S. Tree generating regular systems. Inform. and Control 14 (1969), 217-231.

7. Cheatham, T.E., Jr. Theory and Construction of Compilers, Second ed. Massachusetts Computer Associates, Inc., Wakefield, Mass., 1967.

8. Chomsky, N. Three models for the description of language. IRE Trans. on Inform. Theory IT-2 (1956), 113-124.

9. -------. Syntactic Structures. Mouton, The Hague, 1957.

10. -------. Aspects of the Theory of Syntax. M.I.T. Press, Cambridge, Mass., 1965.

11. Christensen, C. An example of the manipulation of directed graphs in the AMBIT/G programming language. In Klerer, M., and Reinfelds, J. (Eds.), Interactive Systems for Experimental Applied Mathematics. Academic Press, New York, 1968, pp. 423-435.

12. Church, A. A set of postulates for the foundation of logic. Ann. of Math. (2) 33 (1932), 346-366.

13. ------- and Rosser, J.B. Some properties of conversion. Trans. Amer. Math. Soc. 39 (1936), 472-482.

14. Curry, H.B., and Feys, R. Combinatory Logic. North-Holland, Amsterdam, 1958.

15. De Remer, F.L. Practical translators for LR(k) languages. MAC-TR-65, M.I.T. Project MAC, Cambridge, Mass., 1969.

16. Earley, J. An efficient context-free parsing algorithm. Comm. ACM 13 (1970), 94-102.

17. Floyd, R.W. On the nonexistence of a phrase-structure grammar for ALGOL 60. Comm. ACM 5 (1962), 483-484.

18. Ginsburg, S., and Partee, B.H. A mathematical model of transformational grammars. Inform. and Control 15 (1969), 297-334.

19. Gray, J.N., and Harrison, M.A. Single pass precedence analysis. Tenth Annual IEEE Symp. on Switching and Automata Theory (1969), 106-117.

20. Henderson, D.A. Description and definition of simple AMBIT/G. Report CA-6904-2811, Massachusetts Computer Associates, Inc., Wakefield, Mass., 1969.

21. Hindley, R. The Church-Rosser property and a result in combinatory logic. PhD. Thesis, U. of Newcastle-upon-Tyne, 1964.

22. -------. An abstract form of the Church-Rosser theorem, Part I. J. Symbolic Logic 34 (1969), 545-560.

23. -------. An abstract form of the Church-Rosser theorem, Part II. In preparation.

24. Hopcroft, J.E., and Ullman, J.D. Formal Languages and Their Relation to Automata. Addison-Wesley, Reading, Mass., 1969.

25. Karp, R.M., and Miller, R.E. Parallel program schemata. J. Computer and System Sci. 3 (1969), 147-195.

26. Kleene, S.C. Introduction to Metamathematics. Van Nostrand, New York, 1952.

27. Knuth, D.E. Semantics of context-free languages. Math. Systems Theory 2 (1968), 127-145.

28. -------. The Art of Computer Programming, Vol. 1: Fundamental Algorithms. Addison-Wesley, Reading, Mass., 1968.

29. Landin, P.J. A formal description of ALGOL 60. In Steel, T.B., Jr. (Ed.), Formal Language Description Languages for Computer Programming. North-Holland, Amsterdam, 1966, pp. 266-294.

30. Lewis, P.M., II, and Stearns, R.E. Syntax-directed transduction. J. ACM 15 (1968), 465-488.

31. Manna, Z., and McCarthy, J. Properties of programs and partial function logic. In Meltzer, B., and Michie, D. (Eds.), Machine Intelligence 5. American Elsevier, New York, 1970, pp. 27-38.

32. Manna, Z., and Pneuli, A. Formalization of properties of functional programs. J. ACM 17 (1970), 555-590.

33. McCarthy, J. Recursive functions of symbolic expressions and their computation by machine. Comm. ACM 3 (1960), 184-195.

34. -------. Basis for a mathematical theory of computation. In Braffort, P., and Hirschberg, D. (Eds.), Computer Programming and Formal Systems. North-Holland, 1963, pp. 33-70.

35. Minsky, M. Form and content in computer science. J. ACM 17 (1970), 197-215.

36. Mitschke, G. Eine algebraische Behandlung von λ-K-Kalkül und Kombinatorischer Logik. PhD. Thesis, Rheinischen Friedrich-Wilhelms Universität, Bonn, 1970.

37. Morris, J.H., Jr. Lambda-calculus models of programming languages. MAC-TR-57, M.I.T. Project MAC, Cambridge, Mass., 1968.

38. Naur, P. (Ed.) Revised report on the algorithmic language ALGOL 60. Comm. ACM 6 (1963), 1-17.

39. Rosen, B.K. Context-sensitive syntax analysis, Part II: Generative power. Math. Ling. and Autom. Translation NSF-18 (1967), VI-1 – VI-59. Harvard Computation Laboratory, Cambridge, Mass.

40. -------. Syntactic complexity and finite automata. Computer Res. Lab. Memo. RC-T-068, NASA Electronics Research Center, Cambridge, Mass., 1969. Submitted for publication.

41. -------. Tree-manipulating systems and Church-Rosser theorems. Second Annual ACM Symp. on Theory of Computing (1970), 117-127.

42. -------. Tree-manipulating systems and Church-Rosser theorems. To appear in J. ACM.

43. Rosen, S. A compiler-building system developed by Brooker and Morris. In Rosen, S. (Ed.), Programming Systems and Languages. McGraw-Hill, New York, 1967, pp. 306-331.

44. Rosencrantz, D.J., and Stearns, R.E. Properties of deterministic top-down grammars. Inform. and Control 17 (1970), 226-256.

45. Rounds, W.C. Tree-oriented proofs of some theorems on context-free and indexed languages. Second Annual ACM Symp. on Theory of Computing (1970), 109-116.

46. -------. Mappings and grammars on trees. Math. Systems Theory 4 (1970), 257-287.

47. Sanchis, L.E. Functionals defined by recursion. Notre Dame J. Formal Logic 8 (1967), 161-174.

48. Schroer, D.E. The Church-Rosser theorem. PhD. Thesis, Cornell U., Ithaca, N.Y., 1965.

49. Strong, H.R., Jr. Translating recursion equations into flowcharts. Second Annual ACM Symp. on Theory of Computing (1970), 184-197.

50. Thatcher, J.W. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. J. Computer and System Sci. 1 (1967), 317-322.

51. -------. There's a lot more to finite automata theory than you would have thought. Fourth Annual Princeton Conf. on Inform. Sci. and Systems (1970), 263-276.

52. -------. Generalized^2 sequential machine maps. J. Computer and System Sci. 4 (1970), 339-367.

53. Wegbreit, B. Studies in extensible programming languages. PhD. Thesis, Harvard U., Cambridge, Mass., 1970.

54. Wirth, N., and Weber, H. EULER: a generalization of ALGOL, and its formal definition. Comm. ACM 9 (1966), 13-25 and 89-99.

55. Woods, W.A. Semantics for a question-answering system. Math. Ling. and Autom. Translation NSF-19 (1967). Harvard Computation Laboratory, Cambridge, Mass.

56. -------. Procedural semantics for a question-answering machine. Proc. AFIPS 1968 FJCC, Vol. 33, Pt. 1. MDI Publications, Wayne, Penn., 1968, pp. 457-471.