1979

A comparison of automata-theoretic and algebraic approaches to tree transduction and use of algebraic tree transducers in semantic-preserving translations

Kathryn Yuhas Kwinn

Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/rtd

Part of the Computer Sciences Commons

Recommended Citation


http://lib.dr.iastate.edu/rtd/6610

This Dissertation is brought to you for free and open access by Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
INFORMATION TO USERS

This was produced from a copy of a document sent to us for microfilming. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help you understand markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure you of complete continuity.

2. When an image on the film is obliterated with a round black mark it is an indication that the film inspector noticed either blurred copy because of movement during exposure, or duplicate copy. Unless we meant to delete copyrighted materials that should not have been filmed, you will find a good image of the page in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed the photographer has followed a definite method in "sectioning" the material. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For any illustrations that cannot be reproduced satisfactorily by xerography, photographic prints can be purchased at additional cost and tipped into your xerographic copy. Requests can be made to our Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases we have filmed the best available copy.
KWINN, KATHRYN YUHAS
A COMPARISON OF AUTOMATA-THEORETIC AND
ALGEBRAIC APPROACHES TO TREE TRANSDUCTION AND
THE USE OF ALGEBRAIC TREE TRANSDUCERS IN
SEMANTIC-PRESERVING TRANSLATIONS.

IOWA STATE UNIVERSITY, PH.D., 1979
A comparison of automata-theoretic and algebraic approaches
to tree transduction and the use of algebraic tree
transducers in semantic-preserving translations

by

Kathryn Yuhas Kwinn

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of
The Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Major: Computer Science

Approved:
Signature was redacted for privacy.
In Charge of Major Work
Signature was redacted for privacy.
In Charge of Major Work
Signature was redacted for privacy.
For the Major Department
Signature was redacted for privacy.
For the Graduate College

Iowa State University
Ames, Iowa
1979
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II. AN AUTOMATA-THEORETIC APPROACH TO TREE TRANSDUCTION</td>
<td>6</td>
</tr>
<tr>
<td>III. AN ALGEBRAIC APPROACH TO TREE TRANSDUCTION</td>
<td>20</td>
</tr>
<tr>
<td>IV. A COMPARISON OF AUTOMATA-THEORETIC AND ALGEBRAIC APPROACHES TO TREE TRANSDUCTION</td>
<td>37</td>
</tr>
<tr>
<td>V. CONSTRUCTION OF AN ALGEBRAIC TREE TRANSDUCER</td>
<td>62</td>
</tr>
<tr>
<td>VI. CONCLUSIONS</td>
<td>80</td>
</tr>
<tr>
<td>VII. REFERENCES</td>
<td>84</td>
</tr>
<tr>
<td>VIII. ACKNOWLEDGMENTS</td>
<td>86</td>
</tr>
<tr>
<td>IX. APPENDIX</td>
<td>87</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Pictorial representation of a(b(ccd)a(dd))</td>
<td>8</td>
</tr>
<tr>
<td>2.</td>
<td>A top-down tree transducer and its effect on an input tree</td>
<td>11</td>
</tr>
<tr>
<td>3.</td>
<td>A bottom-up tree transducer and its effect on an input tree</td>
<td>13</td>
</tr>
<tr>
<td>4.</td>
<td>The grammar APL and its type</td>
<td>22</td>
</tr>
<tr>
<td>5.</td>
<td>The assignment of meanings to strings</td>
<td>25</td>
</tr>
<tr>
<td>6.</td>
<td>The type r' and the representations of its operations in $\overline{r}_1$ and $\overline{r}_2$</td>
<td>30</td>
</tr>
<tr>
<td>7.</td>
<td>The grammar INT and its type</td>
<td>33</td>
</tr>
<tr>
<td>8.</td>
<td>Pictorial representation of an algebraic tree transducer</td>
<td>36</td>
</tr>
<tr>
<td>9.</td>
<td>A translation table for an algebraic tree transducer</td>
<td>41</td>
</tr>
<tr>
<td>10.</td>
<td>A representation of r' in $\overline{r}_1$ and $\overline{r}_2$</td>
<td>42</td>
</tr>
<tr>
<td>11.</td>
<td>Construction of a simple algebraic tree transducer from a bottom-up tree transducer</td>
<td>49</td>
</tr>
<tr>
<td>12.</td>
<td>Construction of a bottom-up tree transducer from a simple algebraic tree transducer</td>
<td>50</td>
</tr>
<tr>
<td>13.</td>
<td>The operation of a product transducer P</td>
<td>55</td>
</tr>
<tr>
<td>14.</td>
<td>Rules for a deterministic top-down tree transducer and its corresponding product transducer</td>
<td>60</td>
</tr>
<tr>
<td>15.</td>
<td>The relationship of classes of tree transducers</td>
<td>61</td>
</tr>
<tr>
<td>16.</td>
<td>Determination of which subsets of N give rise to a finite P'</td>
<td>72</td>
</tr>
<tr>
<td>17.</td>
<td>Types used to illustrate finding all representations of r' in r</td>
<td>74</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

It is not a simple matter to translate programs written in one language into another language, and yet the ability to do so would save much duplication of effort. It is an even more complex task to produce a correct automated translator to do the same job. Automated translators, however, have proved themselves useful in language implementation projects. Languages such as PASCAL [1,2] and APL [3,4] have been translated into high level intermediate code which is then interpreted. The intermediate code which is generated is syntactically much simpler than the source code, and hence easier to interpret.

A reasonable standard for the correctness of a translation is that the set of outputs resulting from a given set of inputs upon execution of a program should be independent of the language in which the program is written. This is a separate issue from whether or not the original program correctly accomplishes its intended purpose.

The translation process requires knowledge of the syntactic rules of both the source and target languages and of how the sets of rules correspond so that the resulting code has the correct form. In addition, it requires a knowledge of the meanings, or semantics, of the constructs in both languages so that the resulting program has the same meaning as the original.

The problem of associating syntactic forms of different languages is simplified when the syntactic rules have been specified by some formal means, such as Backus-Naur form [5]. Formal specifications of syntax also help with the semantic side of the translation problem by providing
a formal way for meanings to be associated with each syntactic structure. Such associations are called syntax-directed semantics. Several diverse approaches to syntax-directed semantics have been proposed. Among them is Scott's lattice-theoretic approach [6]. Other methods are Floyd's flowchart schemata [7] and Hoare and Wirth's axiomatic approach [8]. Another is Knuth's system of synthesized and inherited attributes [9].

In this paper, we will examine two formal approaches to language translation. One is an automata-theoretic approach introduced by Thatcher [10,11] which manipulates trees, rather than strings [12]. The automata-theoretic tree transducers process their inputs either in a top-down (root-to-frontier) or bottom-up (frontier-to-root) fashion. They read a single node of the input tree at a time and produce their output based on this node and the current state information. The second approach is the algebra-theoretic approach of Krishnaswamy and Strawn [13]. It couches earlier work by Krishnaswamy and Buttelmann [14] in algebraic terms and, in so doing, simplifies their methodology of translation. The algebraic approach views sets of trees as many-sorted algebras and uses homomorphisms between algebras to transform input trees to output trees. The algebraic tree transducers produce their outputs based on configurations of nodes, rather than on single nodes, of the input tree. Both approaches have the ability to specify the outputs corresponding to infinite sets of input trees using only a finite sets of transformation rules. Both are suitable for use in language translation schemes because they are able to transform syntax trees [5] of the source language into syntax trees of the target language. Thus both are capable of syntax-directed translation.
The algebraic tree transducers use a more highly developed concept of semantics than the automata-theoretic tree transducers. The automata-theoretic transducers merely assign meanings, or semantics, to input trees by associating output trees with them. The meaning of an input is simply the associated output. No consideration is given to the meaning of the input tree in and of itself and apart from the translation process. In contrast, in the algebraic framework, one semantic algebra is associated with the set of input trees and another with the set of output trees so that both the input trees and the output trees have meanings in their own right, apart from the translation induced by the algebraic tree transducer. These semantic algebras are based on Knuth's synthesized attributes. This approach makes it possible for us to talk about translations which are semantic-preserving.

As we compare the automata-theoretic and algebraic approaches to tree transduction, we will see that bottom-up tree transducers are equivalent to a restricted form of algebraic tree transducers and that top-down tree transducers and algebraic tree transducers are incomparable. We will also introduce a new kind of automata-theoretic tree transducer, the product transducer, which operates in a bottom-up fashion reading one input node at a time and yet is capable of modeling the algebraic tree transducer. It is also capable of modeling effectively deterministic top-down tree transducers but cannot handle a nondeterministic top-down transduction. Of the transducers we will consider, only an algebraic tree transducer with an infinite set of transformation rules is capable of modeling a nondeterministic top-down tree transducer.
We will also discuss what properties are desirable in algebraic tree transducers. We will want them to have domains which are full and uniquely decomposable. That is, underlying sets of the source algebra are either totally contained in or totally excluded from the domain of the transducer, and each element of the domain can be broken down in exactly one way into configurations appearing in the translation table of the transducer. We will also want our transducers to be semantic-preserving. We will discuss when and how it is possible to achieve these desired properties in a transducer.

Before beginning our study of tree transducers, we present some notation which will be used throughout this paper. If $S$ is a set, then $|S|$ is the cardinality, or number of elements, of $S$, and $P(S)$ is the powerset, or set of all subsets of $S$. If $S$ and $T$ are sets, then $T^S$ is the set of all functions from $S$ into $T$.

If we wish to let a variable, say $i$, take on all integer values from $j$ to $k$, we will write, "for $i=j,\ldots,k$." If we wish to let $i$ take on all nonnegative integer values, we will write "for $i<\omega$.

If $S$ is a set and $k>0$ is an integer, then $S^k$ is the set of all strings over $S$ of length $k$. $S^+ = \bigcup_{0<k<\omega} S^k$ is the set of all nonempty strings over $S$, and $S^* = S^+ \cup \{\lambda\}$ where $\lambda$ denotes the empty string. If $s \in S^+$, then $s \in S^k$ for some $k>0$, and $|s| = k$ gives the length of $s$. If $s = \lambda$, then $|s| = 0$.

When we use the term "tree," we will mean a node-labeled ordered tree, as defined by Knuth [15]. We will often induct on the height of a tree $t$, denoted $|t|$. If $t$ consists of a single node, then $|t| = 1.$
If \( t = s(t_1 \ldots t_k) \) where \( s \) is the root of \( t \) and \( t_1, \ldots, t_k \) are trees, then \( |t| = 1 + \max_{1 \leq i \leq k} |t_i| \).

We will use the term \( x^z_y \) to mean \( x \) with all occurrences of \( y \) replaced by \( z \). Similarly, \( x^{z_1 \ldots z_k}_{y_1 \ldots y_k} \) will mean \( x \) with all occurrences of \( y_1 \) replaced by \( z_1 \), all occurrences of \( y_2 \) replaced by \( z_2 \), \ldots, and all occurrences of \( y_k \) replaced by \( z_k \).

With this background, we begin our study of tree transducers by reviewing the automata-theoretic approach.
II. AN AUTOMATA-THEORETIC APPROACH TO TREE TRANSDUCTION

In this chapter we will discuss tree transducers, a class of automata which Thatcher has referred to as "generalized sequential machine maps" [10]. We will use the notation presented by Engelfriet [16]. Generalized sequential machines are, of course, finite automata which have been extended to produce an output string for every character of the input string while at the same time performing the customary function of determining the syntactic correctness of the input string. Tree transducers are essentially generalized sequential machines which have been further generalized to accept tree, rather than string, input. Finite automata and generalized sequential machines are concerned with the recognition and translation of regular languages. Tree transducers handle the larger class of (syntax trees of) context-free languages. Since we will be discussing context-free grammars and languages throughout this paper, we precede our discussion of tree transducers by a brief review of some essential facts about context-free grammars. A more complete discussion may be found in [5,12].

A context-free grammar $G = (N, E, P, S)$ consists of a finite set $N$ of nonterminals, a finite set $E$ of terminals, a finite set $P \subseteq N \times (N \cup E)^+$ of productions, and a set $S \subseteq N$ of start symbols. A production $(n, m) \in P$ is usually written $n \rightarrow m$.

If $abce(N \cup E)^+$ for some $beN$ and if $b \rightarrow deP$ for some $de(N \cup E)^+$, then we write $abc = \Rightarrow ade$ and we say, "$abc$ directly derives $ade$ in $G$." If $G_{s_0, s_1, \ldots, s_k} \in N \cup E)^+$ and $s_i \Rightarrow s_{i+1}$ for $0 < i < k$, then we write $s_0 \Rightarrow_G s_k$, and we say, "$s_0$ derives $s_k$ in $G$." If $s_0 \in N$ and $s_k \in E^+$, then we call
$s_0, s_1, \ldots, s_k$ a derivation sequence for $s_k$. Every derivation sequence has a representation as a tree. If $s_0 \in S$, this tree is called a derivation tree for $s_k$. If $s_0, s_1, \ldots, s_k$ is a derivation sequence and if $s_i \Rightarrow^G s_{i+1}$ for $0 \leq i < k$ by application of a production to the leftmost nonterminal of $s_i$, then $s_0, s_1, \ldots, s_k$ is called a leftmost derivation of $s_k$ from $s_0$.

If $s_0, s_1, \ldots, s_k$ is a derivation sequence for $s_k$ in $G$, then there must exist a sequence of productions $p_1, \ldots, p_k$ such that $s_i \Rightarrow^G s_{i+1}$ for $0 \leq i < k$. The sequence $p_1, \ldots, p_k$ is called a parse for $s_k$ in $G$. If $s_0, \ldots, s_k$ is a leftmost derivation, then $p_1, \ldots, p_k$ is a left parse. A left parse is the preorder traversal [15] of a tree, called a syntax tree of $G$. If the left-hand side of $p_1$ is a start symbol of $G$, then $p_1, \ldots, p_k$ is a proper syntax tree of $G$.

The language $L(G)$ generated by $G$ is given by $\{t \in \Sigma^*: s \Rightarrow^G t$ for some $s \in S\}$ and is called a context-free language. Every string in $L(G)$ has a leftmost derivation from a start symbol of $G$.

We will now turn our attention to a study of the automata-theoretic approach to tree transduction.

An alphabet $\Sigma$ is a set of symbols. An alphabet $\Sigma$ is ranked if there exists a function $r$ mapping $\Sigma$ into the nonnegative integers $\mathbb{Z}_0^+$. For all $k \geq 0$, $\Sigma_k = \{\sigma \in \Sigma: r(\sigma) = k\}$. If $\sigma \in \Sigma_k$, then $\sigma$ is said to have rank $k$.

For a ranked alphabet $\Sigma$, we define the set $T_\Sigma$ inductively, as follows. If $\sigma \in \Sigma_0$, then $\sigma \in T_\Sigma$. If $\sigma \in \Sigma_k$ for some $k > 1$ and if $t_1, \ldots, t_k \in T_\Sigma$, then $\sigma(t_1 \ldots t_k) \in T_\Sigma$. We observe that every element of $T_\Sigma$ corresponds to the preorder traversal of a node-labeled tree, as illustrated by (II.1). For this reason, we call $T_\Sigma$ the set of all trees over $\Sigma$. 
Example. Let $E=\{a,b,c,d\}$ where $\Sigma_0=\{c,d\}$, $\Sigma_2=\{a\}$, and $\Sigma_3=\{b\}$. Then $a(b(ccd)a(dd))\in T_\Sigma$ and is represented pictorially in Figure 1.

From (II.1), we see that the rank $r(\sigma)$ of a node $\sigma$ of a tree is precisely the number of immediate successors or sons of that particular node. This observation enables us to justify defining rank in terms of functions, rather than relations, as Engelfriet [16] has done. A particular occurrence of any $\sigma\in E$ in any $t\in T_\Sigma$ has a unique number of immediate successors, and hence, a unique rank. Thus if $E$ is an alphabet ranked (in the sense of Engelfriet) by the relation $\preceq E \times Z_0^+$, then there is a one-to-one mapping from $T_\Sigma$ into $T_r$ where $r$ is ranked by the function $r': r \to Z_0^+$ defined by $r'((\sigma,z))=z$ whenever $(\sigma,z)\in r$.

For any set $S$, let $T_\Sigma[S]=T_\Delta$ where $\Delta$ is the ranked alphabet such that $\Delta_0=\Sigma_0 \cup S$ and $\Delta_k=\Sigma_k$ for all $k \geq 1$. Let $X = \{x_1,x_2,x_3,\ldots\}$ be a denumerable set of variables. For $k \geq 1$, let $X_k = \{x_1,x_2,\ldots,x_k\}$, and let $X_0 = \phi$. (These variables will serve as placeholders in the transition rules of the transducer we are about to define.) We further define $Q(X) = \{q(x): q \in Q \text{ and } x \in X\}$ and $Q(T_\Sigma[X]) = \{q(t): q \in Q \text{ and } t \in T_\Sigma[X]\}$. The

![Figure 1. Pictorial representation of $a(b(ccd)a(dd))$]
sets $Q(X_k)$ and $Q(T^*_k[X_k])$ are the obvious restrictions of $Q(X)$ and $Q(T^*_k[X])$.

A **finite tree transducer** $M = (\Sigma, \Delta, Q, Q^o, R)$ consists of

- $\Sigma$, a finite ranked alphabet of **input symbols**,
- $\Delta$, a finite ranked alphabet of **output symbols**,
- $Q$, a finite ranked alphabet of **states** (each having rank 1),
- $Q^o \subseteq Q$, a set of designated states, and
- $R$, a finite set of **rules**.

If every rule in $R$ is of the form $q(\sigma) \rightarrow \tau$ for some $q \in Q$, $\sigma \in \Sigma_0$, and $\tau \in \Delta$ or $q(\sigma(x_1 \ldots x_k)) \rightarrow \tau$ for some $q \in Q$, $\sigma \in \Sigma_k$, and $\tau \in \Delta [Q(X_k)]$, then we say that $M$ is a **top-down tree transducer**, and $Q^o$ is a set of **initial states**.

On the other hand, if every rule in $R$ is of the form $\sigma \rightarrow q(\tau)$ for some $\sigma \in \Sigma_0$, $q \in Q$, and $\tau \in \Delta$ or $\sigma(q_1(x_1) \ldots q_k(x_k)) \rightarrow q(\tau)$ for some $\sigma \in \Sigma_k$, $q_1, \ldots, q_k \in Q$, and $\tau \in \Delta [X_k]$, then we say that $M$ is a **bottom-up tree transducer**, and $Q^o$ is a set of **final states**. A top-down transducer begins processing at the root of the input tree and works toward the leaves, while a bottom-up tree transducer begins processing at the leaves of the input tree and proceeds toward the root.

We say that a tree transducer $M$ is **linear** if no variable (element of $X$) occurs more than once on the right-hand side of any rule. Otherwise, $M$ is **nonlinear**. A top-down tree transducer is **deterministic** if for every $k \geq 0$, $q \in Q$, and $\sigma \in \Sigma_k$, there is exactly one rule in $R$ having $q(\sigma(x_1 \ldots x_k))$ (or $q(\sigma)$ if $k = 0$) as its left-hand side. Likewise, a bottom-up tree transducer is **deterministic** if for every $k \geq 0$, $q_1, \ldots, q_k \in Q$, and $\sigma \in \Sigma_k$, there is exactly one rule in $R$ having $\sigma(q_1(x_1) \ldots q_k(x_k))$ as its left-hand side. All other tree transducers are **nondeterministic**.
However, if no two rules of a tree transducer \( M \) have identical left-hand sides, we say that \( M \) is **effectively deterministic**. Our definitions concerning tree transducers will be illustrated by (II.2) and (II.3).

(II.2) **Example.** Let \( M_1 \) be the tree transducer defined in Figure 2. Then \( M_1 \) is an effectively deterministic top-down tree transducer because all rules are of the top-down form and no two rules have identical left-hand sides. \( M_1 \) is not deterministic because, among other state and input symbol combinations, the combination \( q_a(c) \) does not occur on the left-hand side of a rule. In addition, \( M_1 \) is nonlinear because \( x_1 \) occurs twice on the right-hand side of the third rule.

Let us examine how \( M_1 \) operates on the input tree \( a(b(c)c) \), as shown in part (1) of Figure 2. Since \( q_a \) is the only initial state of \( M_1 \), we must begin processing in state \( q_a \). Hence our initial configuration is given by part (2) of the figure. The first rule is the only one which can be applied at this time. In this rule, the variable \( x_1 \) is a placeholder for the first subtree of \( a \), namely \( b(c) \), and \( x_2 \) is a placeholder for the second subtree, \( c \). Applying the rule, we get part (3). Next, we can process either \( q_b(b(c)) \) or \( q_c(c) \). The choice of which we process first will have no bearing on the final output. In order to process \( q_b(b(c)) \), we must use the second rule of \( M_1 \), which produces part (4). Again, we have a choice. We may process either \( q_b(c) \) or \( q_c(c) \). If we choose \( q_b(c) \), then according to the fourth rule, we get part (5). We complete the transduction by processing \( q_c(c) \) according to the fifth rule. The result is given in part (6). No
$M_1 = (\Sigma, \Delta, Q, Q_d, R)$ where

$\Sigma = \{a, b, c\}$ with $\Sigma_0 = \{c\}$, $\Sigma_1 = \{b\}$, and $\Sigma_2 = \{a\}$,

$\Delta = \{A, B, D\}$ with $\Delta_0 = \{D\}$, and $\Delta_2 = \{A, B\}$,

$Q = \{q_a, q_b, q_c\}$, $Q_d = \{q_a\}$, and $R$ contains

$q_a(a(x_1 x_2)) \rightarrow A(q_b(x_1)q_c(x_2)),$

$q_b(b(x_1)) \rightarrow A(B(q_b(x_1)D)D),$

$q_b(a(x_1 x_2)) \rightarrow B(q_a(x_1)q_b(x_1)),$

$q_b(c) \rightarrow B(DD)$, and

$q_c(c) \rightarrow D.$

Figure 2. A top-down tree transducer and its effect on an input tree
states remain in the tree, and so our processing is complete. Part (6) shows the output tree which results from the input tree shown in part (1).

(II.3) Example. Let $M_2$ be the tree transducer defined in Figure 3. Then $M_2$ is a nondeterministic bottom-up transducer. $M_2$ is also linear.

Let us examine the operation of bottom-up tree transducers by tracing the actions taken by $M_2$ when given as input the tree $a(b(b(c))b(c))$ shown in part (1) of Figure 3. By applying its first rule at each leaf node of the input tree, $M_2$ produces the tree shown in part (2) of the figure. Next, $M_2$ can apply the second rule along each branch of the tree, obtaining the tree in part (3). For its next move, $M_2$ has two choices. It may apply either the third or the fourth rule to the remaining $b$, producing part (4). The final move, in either case, is to apply the last rule, giving part (5). Processing is now complete, and since $q_a \in Q_d$, part (6) shows the output trees associated with the input tree of part (1).

If tree $t'$ results from tree $t$ by the application of a single rule of a transducer $M$, we write $t \xrightarrow{M} t'$. We omit the $M$ whenever it is understood. Also, we write $t \xrightarrow{*} t'$ whenever $t'$ results from $t$ by the successive application of zero or more rules of $M$.

We are now able to define $\text{Tr}_M$, the transduction induced by $M$. If $M$ is a top-down tree transducer, then $\text{Tr}_M = \{ (t, t') \in T_T \times T_T : q(t) \xrightarrow{M} t' \}$ for some $q \in Q_d$. If $M$ is a bottom-up tree transducer, then $\text{Tr}_M = \{ (t, t') \in T_T \times T_T : t \xrightarrow{*} q(t') \}$ for some $q \in Q_d$. $\text{Tr}_M$ is a (partial) function
\[ M_2 = (\Sigma', \Delta', Q', Q_d', R') \] where

\[ \Sigma' = \{a, b, c\} \text{ with } \Sigma_0' = \{c\}, \Sigma_1' = \{b\}, \text{ and } \Sigma_2' = \{a\}, \]

\[ \Delta' = \{A, B, C\} \text{ with } \Delta_0' = \{C\}, \Delta_1' = \{B\}, \text{ and } \Delta_2' = \{A\}, \]

\[ Q' = \{q_a, q_b, q_c\}, Q_d' = \{q_a\}, \text{ and } R' \text{ contains} \]

\[ c \rightarrow q_c(C), \]

\[ b(q_c(x_1)) \rightarrow q_b(B(x_1)), \]

\[ b(q_b(x_1)) \rightarrow q_b(x_1), \]

\[ b(q_b(x_1)) \rightarrow q_b(B(x_1)), \text{ and} \]

\[ a(q_b(x_1)q_j(x_2)) \rightarrow q_a(A(x_1Cx_2)). \]

Figure 3. A bottom-up tree transducer and its effect on an input tree.
from $T_L$ to $T_\Delta$ whenever $M$ is an effectively deterministic transducer. Otherwise, $T_M$ is a relation.

We have noted that tree transducers are generalizations of the generalized sequential machines (gsm) of conventional (string) automata theory and that gsm's are basically finite automata which have been extended to produce output. The purpose of a finite automaton is to recognize all the words of a regular language. We shall see that a tree transducer can be used to recognize all the derivation trees or syntax trees of a context-free grammar.

A **tree automaton** $A = (\Sigma, \Sigma, Q, Q_d, R)$ is a tree transducer whose input and output alphabets are identical and whose rules are restricted as stated below. If $A$ is a top-down tree automaton, then every rule in $R$ is either of the form $q(\sigma) \rightarrow a$ if $\sigma \in \Sigma_0$ or $q(\sigma(x_1 \ldots x_k)) \rightarrow \sigma(q_1(x_1) \ldots q_k(x_k))$ if $\sigma \in \Sigma_k$ for some $k \geq 1$. If $A$ is a bottom-up tree automaton, then every rule in $R$ is either of the form $a \rightarrow q(\sigma)$ if $\sigma \in \Sigma_0$ or $\sigma(q_1(x_1) \ldots q_k(x_k)) \rightarrow q(\sigma(x_1 \ldots x_k))$ if $\sigma \in \Sigma_k$ for some $k \geq 1$.

We say that a top-down tree automaton $A$ **recognizes** (or **accepts**) a tree $t \in T_\Sigma$ provided $q(t) \overset{*}{\rightarrow} t$ for some $q \in Q_d$. Likewise, if $A$ is a bottom-up tree automaton, then we say that $A$ recognizes (or accepts) $t \in T_\Sigma$ whenever $t \overset{*}{\rightarrow} q(t)$ for some $q \in Q_d$. We say that a set of trees $S \subseteq T$ is **recognizable** if there exists a tree automaton $A$ such that $S = \{ t : A \text{ recognizes } t \}$.

There are a number of well-known results about recognizers and recognizable sets. We shall mention several of them below.
(II.4) Theorem. The domain of every tree transducer is a recognizable set [16].

(II.5) Theorem. Every recognizable set is recognizable by a deterministic bottom-up tree automaton [11].

In fact, the following three classes of tree automata all recognize precisely the same sets of trees: deterministic bottom-up, nondeterministic bottom-up, and nondeterministic top-down. The deterministic top-down tree automata are, however, less powerful. There are recognizable sets of trees which are not recognizable by any deterministic top-down tree automaton. These results are discussed in [11].

Another of the well-known results demonstrates the relationship between context-free grammars and tree automata. As a direct consequence of this result, which is stated in (II.6), the set of all derivation trees of a context-free grammar is recognizable.

Let \( \Sigma_1 \) and \( \Sigma_2 \) be alphabets ranked by \( r_1 \) and \( r_2 \), respectively. A projection \( \pi: \Sigma_1 \rightarrow \Sigma_2 \) is a rank preserving function. That is, for all \( s \in \Sigma_1 \) we have \( r_2(\pi(s)) = r_1(s) \). The mapping \( \pi \) can be extended in a natural way to \( \pi^*: T_{\Sigma_1} \rightarrow T_{\Sigma_2} \) by defining \( \pi^*(s) = \pi(s) \) whenever \( r_1(s) = 0 \) and \( \pi^*(s(t_1 \ldots t_k)) = \pi(s)(\pi^*(t_1) \ldots \pi^*(t_k)) \) whenever \( r_1(s) = k \) and \( t_i \in \Sigma_2 \) for \( i = 1, \ldots, k \). The extended map \( \pi^* \) is also called a projection.

(II.6) Theorem. Every recognizable set of trees is the image of a projection from the set of derivation trees of a context-free grammar [10,17].
From a practical standpoint, we would like a result which says that syntax trees, rather than derivation trees, of a context-free grammar are recognizable by tree automata. Such a result would enable us to use a tree transducer to do syntax-directed translation of parsed code since a parser, in effect, produces syntax trees from source code. Such a result is an immediate consequence of the following theorem.

(II.7) Theorem. Every recognizable set is the image of a projection from the set of proper syntax trees of a context-free grammar.

Proof. Let \( Z \) be a recognizable subset of \( T_\Sigma \) for some alphabet \( \Sigma \) ranked by \( r \), and let \( A = (\Sigma, \Sigma, Q, Q_d, R) \) be a deterministic bottom-up tree automaton which recognizes \( Z \).

Let \( G = (Q, \Sigma, P, Q_d) \) be a context-free grammar where \( P \) is formed as follows. If \( \sigma \rightarrow q(\sigma) \in R \), then \( q \rightarrow \sigma \in P \). If \( \sigma(q_1(x_1)\ldots q_k(x_k)) \rightarrow q(\sigma(x_1\ldots x_k)) \in R \), then \( q \rightarrow \sigma q_1\ldots q_k \in P \).

We will show that the set of trees recognized by \( A \) is the image of a projection from the set of proper syntax trees of \( G \). First, we define a ranking function on \( P \). Every production in \( P \) is of the form \( q \rightarrow \sigma q_1\ldots q_k \) for some \( \sigma \in \Sigma, q, q_1, \ldots, q_k \in Q \), and \( k \geq 0 \). Let us define \( r'(q \rightarrow \sigma q_1\ldots q_k) = k \). This definition is consistent with our observation that the rank of a node in a tree is the number of immediate successors of that node. We observe that \( r'(q \rightarrow \sigma q_1\ldots q_k) = r(\sigma) \) for all productions in \( P \). Thus we define \( \pi: P \rightarrow \Sigma \) by \( \pi(q \rightarrow \sigma q_1\ldots q_k) = \sigma \) for all \( q \rightarrow \sigma q_1\ldots q_k \in P \) and \( k \geq 0 \). Clearly \( \pi \) is a projection.

In the Appendix we show that \( \pi^* \) maps the set of proper syntax trees of \( G \) precisely onto the set of trees recognized by \( A \).
Several other well-known results about tree transducers deserve mention. First, the classes of transductions induced by top-down and bottom-up tree transducers are incomparable [16,18]. Top-down transducers have the ability to generate multiple copies of a subtree of the input tree and then can process these copies differently. Bottom-up transducers cannot model this behavior. On the other hand, bottom-up transducers can produce multiple copies of or delete subtrees of the output tree. Top-down transducers cannot perform these functions. The following examples adapted from [16] illustrate the incomparability of top-down and bottom-up tree transducers.

(II.8) Example. Let $T = (E, \Delta, Q, Q_d, R)$ be a top-down tree transducer where $E = \{\sigma, a_0, a_1\}$ with $E_0 = \{a_0\}$ and $E_1 = \{\sigma, a_1\}$, $\Delta = \{\sigma, a_0, b_0, a_1, b_1\}$ with $\Delta_0 = \{a_0, b_0\}$, $\Delta_1 = \{a_1, b_1\}$, and $\Delta_2 = \{\sigma\}, Q = Q_d = \{q\}$, and $R$ contains:

- $q(\sigma(x_1)) \rightarrow \sigma(q(x_1)q(x_1)),$
- $q(a_1(x_1)) \rightarrow a_1(q(x_1)),$
- $q(a_1(x_1)) \rightarrow b_1(q(x_1)),$
- $q(a_0) \rightarrow a_0,$ and
- $q(a_0) \rightarrow b_0.$

It is reasonably easy to see why no bottom-up tree transducer can induce $T_{\sigma}$. If $T$ encounters an input tree $\sigma(t)$ where $t = a_1(a_1(...(a_1(a_0))...))$, it first creates a tree $\sigma(q(t)q(t))$ and then processes each copy of $t$ independently. Since $T$ is nondeterministic, it will produce many outputs for the given input, and in many of these, the outputs associated with the two copies of $t$ will be different. A
bottom-up tree transducer would process $t$ before reading $\sigma$. Upon reading $\sigma$ it could, at best, make two identical copies of the output associated with $t$. Thus no bottom-up tree transducer can induce $T_{\sigma}$. 

(II.9) Example. Let $B = (\Sigma, \Delta, Q, Q_d, R')$ be a bottom-up tree transducer where $\Sigma, \Delta, Q$, and $Q_d$ are as in (II.8) and $R'$ contains:

\begin{align*}
a_0 &\rightarrow q(a_0), \\
a_0 &\rightarrow q(b_0), \\
a_1(q(x_1)) &\rightarrow q(a_1(x_1)), \\
a_1(q(x_1)) &\rightarrow q(b_1(x_1)), \\
\sigma(q(x_1)) &\rightarrow q(\sigma(x_1 x_1)).
\end{align*}

Again, it is fairly simple to see why no top-down tree transducer $T$ can induce $T_{\sigma}$. When $B$ encounters the input tree $\sigma(t) = \sigma(a_1(a_1(\ldots(a_1(a_0)\ldots)))$, it first nondeterministically translates $t$ to some $\tau$, and then $\sigma(q(\tau)) \Rightarrow q(\sigma(\tau \tau))$. A top-down transducer would first create $\sigma(q_1(t) q_2(t))$ for some states $q_1$ and $q_2$ and then would need to translate both copies of $t$ to $\tau$. Clearly $T$ would need to be nondeterministic to create $\tau$ from $t$, and hence it would have the ability to translate $t$ in other ways as well. Hence $\sigma(\tau \tau)$ is only one of many possible outputs a top-down transducer would produce from $\sigma(t)$. Thus no top-down tree transducer can induce $T_{\sigma}$. 

Baker [18] and Engelfriet [16] have both shown that every transduction induced by a top-down tree transducer can be induced by two bottom-up tree transducers in composition. Similarly, they have shown that the transduction induced by a bottom-up tree transducer can be performed by two top-down transducers in composition. These results,
together with the incomparability of top-down and bottom-up transducers, show that neither the class of top-down nor bottom-up tree transducers is closed under composition.

Many other facts about tree transducers are known but are not necessary for an understanding of this paper. The approach to tree transduction in this chapter has been distinctly automata-theoretic. We have studied a type of finite state machine which both inputs and outputs trees, and we have seen how these machines relate to both the derivation trees and syntax trees of context-free grammars. In the next chapter, we will look at tree transduction from an algebraic, rather than automata-theoretic, point of view. In Chapter IV, we will examine the relationship of the two approaches.
III. AN ALGEBRAIC APPROACH TO TREE TRANSDUCTION

In this chapter, we will consider sets of trees to be algebras, and we will describe tree transducers in terms of homomorphisms between algebras. In the next chapter we will examine the relationship between the algebraic approach presented here and the automata-theoretic approach presented in the previous chapter. The material discussed in this chapter is due to Krishnaswamy and Strawn [13].

Let $P$ be a set whose elements we will call operation symbols, and let $N$ be a set whose elements we will call sorts. Then a type is a function $r: P \to N \times N^*$. If $P$ is finite, then $r$ is a finite type.

A context-free grammar $G = (N, \Sigma, P, S)$ with sets $N$ of nonterminals, $\Sigma$ of terminals, $P$ of productions and $S$ of start symbols gives rise to a type in a natural way. The type of $G$ is the function $r: P \to N \times N^*$ where for all $p \in P$, if $p$ is $n \to n_1 \sigma_1 \cdots n_k \sigma_k$ with $n, n_1, \ldots, n_k \in N$ and $\sigma_0, \sigma_1, \ldots, \sigma_k \in \Sigma^*$, then $r(p) = (n_1 \cdots n_k, n)$. Henceforth, the term "type $r$" denotes a type $r: P \to N \times N^*$.

A type $r$ gives rise to a class of algebras of type $r$. Each algebra of type $r$ has underlying sets indexed by $N$ and operations indexed by $P$. Formally, an algebra of type $r$ (or $r$-algebra) $A$ is a pair $\{A_n\}_{n \in N}, \{P_p\}_{p \in P}$ where each $A_n$ is an underlying set and each $P_p$ is an operation. Furthermore, if $r(p) = (\alpha, \beta)$, then $P_p: A^\alpha \to A^\beta$ where for all $\alpha \in N^*$, $A^\alpha = \{\phi\}$ if $\alpha = \lambda$ and $A^\alpha = A_{n_1} \times \cdots \times A_{n_k}$ if $\alpha = n_1 \cdots n_k$.

For the purposes of this paper, we will usually restrict ourselves to discussing only those classes of algebras which result from those types $r$ for which $r: P \to N \times (N \cup \{\lambda\})$. Hence if $r(p) = (\alpha, n)$ where
For every type \( r \), there is an important \( r \)-algebra \( W \), called the **word algebra of type \( r \) or the word \( r \)-algebra.** The underlying sets of \( W \) are the smallest sets \( W_n \) for \( n \in \mathbb{N} \) determined by the following rule. If \( p \in P \), \( r(p) = (a,n) \) and \( x \in W^a \), then \( p x \in W^a \) if \( a \neq \lambda \) and \( p \in W^a \) if \( a = \lambda \). The operations of \( W \) are evaluated as follows. If \( p \in P \), \( r(p) = (a,n) \), and \( x \in W^a \), then \( p^W(x) = px \) if \( a \neq \lambda \) and \( p^W(\phi) = p \) if \( a = \lambda \).

The word \( r \)-algebra is important for a number of reasons. For example, the word \( r \)-algebra has a very important interpretation when \( r \) is the type of a context-free grammar \( G = (N, \Sigma, P, S) \). The elements of \( W_n \), for every \( n \in \mathbb{N} \), are precisely the left parses associated with the (leftmost) derivations of terminal strings from the nonterminal \( n \). Thus, when \( P \) is interpreted as a ranked alphabet where the ranking function \( r' \) is defined by \( r'(p) = |a| \) whenever \( r(p) = (a,n) \), then the elements of the underlying sets of \( W \) are the syntax trees of \( G \).

Some of the previous discussion will be illustrated by (III.1), which will be referenced frequently throughout this paper. The grammar we will use is a context-free grammar for a sublanguage of APL [4]. It is a working grammar for lexically analyzed code, rather than for source strings.

(III.1) **Example.** Let \( APL = (N_1, \Sigma_1, P_1, S_1) \) be the context-free grammar shown in Figure 4. The type of \( APL \), \( r_1: P_1 \to N_1^{\ast} \times N_1^{\ast} \), is also shown in Figure 4.
\[ \text{APL} = (N_1, \Sigma_1, P_1, S_1) \text{ where} \]
\[ N_1 = \{\text{Exp, Bas, Exp}'\}, \quad \Sigma_1 = \{\text{Con, Var, Op, (,)}\}, \]
\[ S_1 = \{\text{Exp}\}, \text{ and } P_1 \text{ is given below.} \]

<table>
<thead>
<tr>
<th>Name of production ( p )</th>
<th>Actual production ( p )</th>
<th>( r_1(p) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>( \text{Exp} \to \text{Bas} )</td>
<td>((\text{Bas}, \text{Exp}))</td>
</tr>
<tr>
<td>OE</td>
<td>( \text{Exp} \to \text{Op Exp} )</td>
<td>((\text{Exp}, \text{Exp}))</td>
</tr>
<tr>
<td>VE</td>
<td>( \text{Exp} \to \text{Var Exp} )</td>
<td>((\text{Exp}, \text{Exp}))</td>
</tr>
<tr>
<td>BOE</td>
<td>( \text{Exp} \to \text{Bas Op Exp} )</td>
<td>((\text{Bas Exp}, \text{Exp}))</td>
</tr>
<tr>
<td>BVE</td>
<td>( \text{Exp} \to \text{Bas Var Exp} )</td>
<td>((\text{Bas Exp}, \text{Exp}))</td>
</tr>
<tr>
<td>C</td>
<td>( \text{Bas} \to \text{Con} )</td>
<td>((\lambda, \text{Bas}))</td>
</tr>
<tr>
<td>V</td>
<td>( \text{Bas} \to \text{Var} )</td>
<td>((\lambda, \text{Bas}))</td>
</tr>
<tr>
<td>E'</td>
<td>( \text{Bas} \to (\text{Exp}') )</td>
<td>((\text{Exp}', \text{Bas}))</td>
</tr>
<tr>
<td>E</td>
<td>( \text{Exp}' \to \text{Exp} )</td>
<td>((\text{Exp}, \text{Exp}'))</td>
</tr>
</tbody>
</table>

Figure 4. The grammar APL and its type

The word \( r_1 \)-algebra \( W_1 \) consists of underlying sets \( W_1, \text{Exp}', W_1, \text{Bas}' \) and \( W_1, \text{Exp}' \) and operation symbols \( B_{W_1}, OE_{W_1}, \ldots, E_{W_1} \). Clearly according to the rules of operation evaluation in a word algebra, the expression \( BVE_{W_1} \left( C_{W_1} B_{W_1} \left( V_{W_1} \right) \right) = BVE \text{ C B V} \) is an element of \( W_1, \text{Exp}' \). But \( BVE \text{ C B V} \) is the parse corresponding to the leftmost derivation \( \text{Exp} \Rightarrow \text{Bas Var Exp} \Rightarrow \text{Con Var Exp} \Rightarrow \text{Con Var Bas} \Rightarrow \text{Con Var Var} \). Thus \( BVE \text{ C B V} \) is a syntax tree of the APL grammar.

Another important \( r \)-algebra is the string algebra of type \( r \). Let \( r \) be the type of a context-free grammar \( G = (N, \Sigma, P, S) \). Then the string algebra of type \( r \) (or string \( r \)-algebra) \( ST \) is a pair \((\{ST\}_{n \in \mathbb{N}}, \{p_{ST}\}_{p \in P})\)
where \( \text{ST}_n = \Sigma^* \) for all \( n \in \mathbb{N} \). If \( p \in P \) is \( n = \sigma_0 n_1 \sigma_1 \ldots \sigma_k \) where \( \sigma_i \in \Sigma \) for \( i = 1, \ldots, k \), \( n_i \in \mathbb{N} \), and \( x = (x_1 \ldots x_k) \in \text{ST}_{n_1} \times \ldots \times \text{ST}_{n_k} \), then
\[
p_{\text{ST}}(x) = \sigma_0 \sigma_1 \ldots \sigma_k.
\]
Once we have discussed the concept of homomorphisms between \( r \)-algebras, we will be able to see the connection between the string \( r \)-algebra and \( L(G) \), the set of strings generated by \( G \).

A homomorphism of \( r \)-algebras is a family of operation preserving functions. It is the natural extension of the concept of homomorphism of classical algebras (in which \( \mathbb{N} \) is a singleton set) [19].

Let \( A \) and \( B \) be \( r \)-algebras. A homomorphism \( h: A \rightarrow B \) of \( r \)-algebras is a family of functions \( \{ h_n : A_n \rightarrow B_n \} \) such that for every \( p \in P \), if \( r(p) = (\alpha, n) \), \( x = x_1 \ldots x_k \in \Lambda^\alpha \) where \( k = |\alpha| \), then
\[
h_n(p_A(x)) = p_B(h_{n_1}(x_1) \ldots h_{n_k}(x_k))
\]
and \( h_n(p_A(x)) \in B_n \). Naturally, if \( r(p) = (\lambda, n) \), then \( h_n(p_A(x)) = p_B \in B_n \). Each \( h_n \) is called a component of the homomorphism \( h \).

We will often consider homomorphisms \( h: W \rightarrow B \) where \( W \) is the word \( r \)-algebra and \( B \) is any \( r \)-algebra. Since the underlying sets of \( W \) are always disjoint, we often write \( h(x) \) for \( h_n(x) \) when \( x \in W_n \), since it is understood which component of \( h \) we are applying.

The set of all homomorphisms \( h: A \rightarrow B \) where \( A \) and \( B \) are arbitrary \( r \)-algebras can be arbitrarily large. However, in the case where \( A \) is \( W \), the word \( r \)-algebra, there is a unique homomorphism from \( A \) into \( B \) [20]. This fact is central to the development of the algebraic tree transducer.

(III.2) Theorem. Let \( r: P \rightarrow \mathbb{N}^* \times \mathbb{N}^* \) be a type. Let \( W \) be the word \( r \)-algebra, and let \( A \) be any \( r \)-algebra. Then there is a unique homomorphism \( h: W \rightarrow A \).
The proof of (III.2) can be found in [20].

Let us consider for a moment the homomorphism $h: W \to ST$ where $W$ is the word $r$-algebra, $ST$ is the string $r$-algebra, and $r$ is the type of the context-free grammar $G = (N, \Sigma, P, S)$. We claim that for all $n \in N$ and $w \in W_n$, we have $h(w) = z \in \Sigma^*$ if and only if $w$ is the parse corresponding to the leftmost derivation of the string $w$ from the nonterminal $n$. That is, $h$ associates every syntax tree of $G$ with the terminal string for which it is a parse. Thus $h(W) = \bigcup_{n \in N} h(W_n) = L(G')$ where $G' = (N, \Sigma, P, N)$, and furthermore $\bigcup_{n \in S} h(W_n) = L(G)$.

Our goal for the remainder of this chapter will be to develop "semantic-preserving" tree transducers within the algebraic framework we have described. This requires that we incorporate some notion of semantics into this framework. We will do so via the language definition system.

Let $G$ be a context-free grammar with type $r$, and let $A$ be an arbitrary $r$-algebra. Then a language definition system (LDS) is a pair $D = (G, A)$ whose underlying grammar is $G$ and whose algebraic semantics is $A$.

We intuitively think of semantics as some assignment of meaning to every string in a language. Based on our previous discussions of string algebras and homomorphisms from word algebras, we can easily see how an LDS is consistent with our intuitive notion of semantics. Let $D = (G, A)$ be an LDS where $G = (N, \Sigma, P, S)$ has type $r$, and let $W$ be the word $r$-algebra. Then $W$ contains precisely the set of syntax trees of $G$. Furthermore, there is a unique homomorphism $\sigma$ from $W$ into $ST$, the string $r$-algebra, and $\sigma(W)$ is precisely the set of terminal strings which can
be derived from the nonterminals of G. We call $\sigma$ the **concrete syntax homomorphism** of $D$. There is also a unique homomorphism $\mu: W \rightarrow A$, which we call the **semantic homomorphism** of $D$. These algebras and homomorphisms are depicted in Figure 5.

We now have a rigorous syntax-directed method of assigning a meaning to every string in $\sigma(W)$. Let $sec(W)$. Then $\{\mu(w): \sigma(w) = s\}$ is the set of all meanings associated with $s$. From this point on, we will think of languages as sets of strings together with their meanings, and we will speak of $L(D)$, the **language of $D$**, where $L(D) = \{((\sigma(w)), \mu(w)) : weU, \text{ } neS, \text{ } n\}$. From this definition, it is easy to see that $L(D)$ determines a function if and only if $G$ is an unambiguous grammar.

In order to define algebraic tree transducers, we will also need the notions of "derived" and "represented" types. Derived types are necessary because we will often want our transducers to produce output based on a configuration of nodes in the input tree rather than based on single nodes, as conventional tree transducers do. For example, we will ultimately construct a tree transducer whose input trees are the syntax trees of the grammar APL given in Figure 4. We will want this transducer

![Diagram](image_url)

**Figure 5.** The assignment of meanings to strings
to produce output trees based on the following portions of input trees:
B(C), B(V), B(E'(E(x_1))), OE(x_1), BOE(V x_1), BOE(V x_1), BOE(E'(E(x_1)) x_2),
BVE(C x_1), BVE(V x_1), and BVE(E'(E(x_1)) x_2). We will explain how to
construct the derived type \( \overline{r} \) for any type \( r \), and we will see that the
configurations we have specified correspond to operations in the type
derived from APL. Represented types are necessary to allow us to
translate between languages whose underlying grammars are of different
types.

Let \( r: P \to N^* \times N^* \) be a type. Then the **derived type** of \( r \) is a type
\( \overline{r}: \overline{P} \to N^* \times N^* \) where \( \overline{P} \) is the smallest set such that

1) \( P \subseteq \overline{P} \) and \( \overline{r}(p) = r(p) \) for all \( p \in P \),
2) \( \{ x^\alpha_i : \alpha \in N^* \text{ and } 0 < i < |\alpha| \} \subseteq \overline{P} \) and \( \overline{r}(x^\alpha_i) = (\alpha, \alpha_i) \) if \( i > 0 \) and
   \( \overline{r}(x_0^\alpha) = (\alpha, \lambda) \), and
3) If \( q_0, q_1, \ldots, q_k \in \overline{P} \) with \( r(q_i) = (\beta_i, \alpha_i) \) for \( i = 1, \ldots, k \) and if
   \( \overline{r}(q_0) = (\alpha, n) \), then \( q = q_0[q_1 \ldots q_k] \in \overline{P} \) and \( \overline{r}(q) = (\beta, n) \).

An element \( x^\alpha_i \) of \( \overline{P} \) is called a **projection**. We note that, for all
intents and purposes, \( \overline{P} \) is a subset of the set of trees over the infinite
alphabet which consists of the elements of \( P \) and all the projections.

Since \( \overline{r} \) is a type it gives rise to a class of \( \overline{r} \)-algebras. Those
\( \overline{r} \)-algebras derived from \( r \)-algebras are of special importance. Let
\( r: P \to N^* \times N^* \) be a type, and let \( A \) be an \( r \)-algebra. Then the **derived**
\( \overline{r} \)-**algebra** \( \overline{A} \) is the \( \overline{r} \)-algebra \( (\{ \overline{A} \}_n, \{ p^r_A \}_p) \) where

1) \( \overline{A}_n = A_n \) for all \( n \in N \), and
2) If \( \overline{r}(p) = (\alpha, n) \), then \( p^r_A: A^\alpha \to A_n \), and if \( \overline{r}(p) = (\alpha, \lambda) \), then
   \( p^\alpha_A: A^\alpha \to \{ \phi \} \). Furthermore
   a) \( p^r_A = p_A \) for all \( p \in P \),
b) $X^\alpha_{0,A}(x) = \phi$ for all $\alpha \in N^*$ and $x \in A$,
c) $X^\alpha_{1,A}(x) = x$ for all $\alpha \in N^*$ and $x \in A$, and
d) $q_0[q_1 \ldots q_k]_A(x) = q_0,A(x) \ldots q_k,A(x)$.

We call the operations of $A$ derived operations.

(III.3) Example. Let $r_1$ be the type of the grammar APL given in Figure 4, and let $\overline{r}_1$ be the type derived from $r_1$. Let $W_1$ be the word $r_1$-algebra, and let $\overline{W}_1$ be the derived $\overline{r}_1$-algebra. Then the operations of $\overline{W}_1$ are indexed by $\overline{P}_1$ where $P_1 \subseteq \overline{P}_1$, $\{X^\alpha_{1}: \alpha \in N^* \text{ and } 0 \leq i \leq |\alpha| \} \subseteq \overline{P}_1$, and, among others, such expressions as $B[C]$, $B[V]$, $X^\alpha_1 \text{Exp}$, $X^\alpha_1 \text{Bas} \text{Bas} \text{Exp}$, $B[X^\alpha_1 \text{Bas}]$, and $BVE[X^\alpha_1 \text{Bas} \text{Exp} X^\alpha_1 \text{Bas} \text{Exp}] [E'[E[X^\alpha_1 \text{Exp} \text{Exp}]] X^\alpha_1 \text{Exp} \text{Exp}]$ are elements of $\overline{P}_1$.

In fact, we can show that there is an operation symbol in $\overline{P}_1$ associated with each configuration mentioned previously as a desired input to the transducer we will construct.

We can clearly see that for any type $r$ and set $P$, the resulting set $\overline{P}$ of derived operation symbols is infinite. Furthermore, $\overline{P}$ contains many operation symbols which represent the same derived operation in a derived $\overline{r}$-algebra. For instance, in (III.3) both $X^\alpha_1 \text{Exp}[B[C]]$ and $B[C]$ represent the derived operation $B(C)$. We list below several identities which are true in every derived $\overline{r}$-algebra $\overline{A}$.

1) $X^\alpha_{0}[p_1 \ldots p_k]_A = X^\alpha_{0,A} \overline{p}_1$ whenever $\overline{r}(p_1) = (\beta, \alpha)$ for $1 \leq i \leq k = |\alpha|$.
2) $X^\alpha_{1}[p_1 \ldots p_k]_A = p_1,A \overline{p}_1$.
3) $p_0[p_1 \ldots p_k][q_1 \ldots q_m]_A = p_0[p_1[q_1 \ldots q_m] \ldots p_k[q_0 \ldots q_m]]_A$.
4) $p[X^\alpha_{1} \ldots X^\alpha_{k}]_A = p_A \overline{p}$ whenever $\overline{r}(p) = (\alpha, n)$ for some $n \in N$.

We can use the preceding identities to find a "simplest" element $\overline{q}$ of $\overline{P}$ such that $\overline{q}_A = p_A \overline{p}$ for any given $p \in \overline{P}$. We will say that such a $\overline{q}$
is in "normal form." Let \( r: p \to \mathbb{N}^\times\mathbb{N}^\times \) be a type with derived type \( \overline{r} \). Then \( p \in \mathbb{P} \) is in normal form if either \( p = X_1^a \) for some \( a \in \mathbb{N}^\times \) and \( 0 < i < |a| \) or \( p = p_0[p_1 \ldots p_k] \) where \( p_0 \in \mathbb{P} \) and \( p_i \) is in normal form for \( i = 1, \ldots, k \). That is, operation symbols in normal form have projections along the frontiers of their associated derived operations and elements of \( \mathbb{P} \) at all branching nodes.

(III.4) **Theorem.** Let \( \overline{r} \) be the derived type of \( r: \mathbb{P} \to \mathbb{N}^\times\mathbb{N}^\times \). Then for each \( p \in \mathbb{P} \), there is a unique \( q \in \mathbb{P} \) in normal form such that for every derived \( r \)-algebra \( A \), \( q_A = p_A \).

Another important theorem concerning \( \overline{r} \)-algebras and \( r \)-algebras says that they have the same sets of homomorphisms. As a consequence, there is a unique homomorphism from the derived word \( \overline{r} \)-algebra (i.e., the algebra derived from the word \( r \)-algebra) to any derived \( r \)-algebra \( \overline{A} \).

(III.5) **Theorem.** Let \( r \) be a type with derived type \( \overline{r} \). Let \( A \) and \( B \) be \( r \)-algebras, and let \( \overline{A} \) and \( \overline{B} \) be their derived \( \overline{r} \)-algebras. Then \( h: A \to B \) is a homomorphism of \( r \)-algebras if and only if \( h: \overline{A} \to \overline{B} \) is a homomorphism of \( \overline{r} \)-algebras.

We now turn our attention to representable types and represented algebras. Let \( r': \mathbb{P}' \to \mathbb{N}'^\times\mathbb{N}'^\times \) and \( r: \mathbb{P} \to \mathbb{N}^\times\mathbb{N}^\times \) be types. Let \( \eta: \mathbb{N}' \to \mathbb{N} \) and \( \pi: \mathbb{P}' \to \mathbb{P} \) be functions. Furthermore, let \( \eta \) extend to domain \( \mathbb{N}'^\times \) and codomain \( \mathbb{N}^\times \) by the rules \( \eta(\lambda) = \lambda \) and \( \eta(n_1 \ldots n_k) = \eta(n_1) \ldots \eta(n_k) \). If for each \( p' \in \mathbb{P}' \) with \( r'(p') = (\alpha, \beta) \) we have \( r(\pi(p')) = (\eta(\alpha), \eta(\beta)) \), then we say that \( r' \) is representable in \( r \) via \( \eta \) and \( \pi \).
Given an r-algebra A and a type r' that is representable in r via \( \eta \) and \( \pi \), we can construct a unique r'-algebra \( A' \) which is defined in terms of the underlying sets and operations of A. Let \( r': P' \rightarrow N' \times N' \) be representable in r via \( \eta: N' \rightarrow N \) and \( \pi: P' \rightarrow P \), and let A be an r-algebra. Then we say that the r'-algebra \( A' = \{\{A_{\eta(n)}\}_{n \in N'}, \{\pi(p)_{A}\}_{p \in P}\} \) is represented in A.

(III.6) Example. Let \( r_{1} \) be the type of the grammar APL as given in Figure 4, let \( \overline{r}_{1} \) be the type derived from \( r_{1} \), and let \( r': P' \rightarrow N' \times N' \) be a type where \( N' = \{Y,Z\} \), \( P' = \{1,2,\ldots,22\} \), and \( r' \) is as shown in Figure 6. Let \( \eta_{1}: N' \rightarrow \overline{N}_{1} \) be given by \( \eta_{1}(Y) = \eta_{1}(Z) = \text{Exp} \), and let \( \pi_{1}: P' \rightarrow \overline{P}_{1} \) be shown in Figure 6.

We claim that \( r' \) is representable in \( \overline{r}_{1} \) via \( \eta_{1} \) and \( \pi_{1} \). This is clear because \( \eta_{1} \) is a constant function. Hence, as long as \( \pi_{1}(p) \) and \( p \) are operations having the same number of arguments, we can be sure that the representability condition is met. Consequently the word r'-algebra \( \overline{W}' \) is representable in the word \( \overline{r}_{1} \)-algebra \( \overline{W}_{1} \) by the r'-algebra \( \overline{W}_{1}' = \{\{\overline{W}_{1}(n)\}_{n \in N'}, \{\pi(p)_{\overline{W}_{1}}\}_{p \in P'}\} \). We note that the operations of the represented algebra \( \overline{W}_{1}' \) are precisely the inputs we have said we will want for the transducer we will build.

We are now ready to use the algebraic concepts we have been discussing to formulate a mechanism for performing language translation. The transducer we will define will transform syntax trees of one grammar into syntax trees of another grammar. Translation from source strings to target strings will require only the additional application of the appropriate concrete syntax homomorphisms.
<table>
<thead>
<tr>
<th>p</th>
<th>( r'(p) )</th>
<th>( \pi_1(p) )</th>
<th>( \pi_2(p) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((\lambda, Y))</td>
<td>(B[C])</td>
<td>(C_0)</td>
</tr>
<tr>
<td>2</td>
<td>((\lambda, Z))</td>
<td>(B[C])</td>
<td>(I[C_0])</td>
</tr>
<tr>
<td>3</td>
<td>((\lambda, Y))</td>
<td>(B[V])</td>
<td>(V_0)</td>
</tr>
<tr>
<td>4</td>
<td>((\lambda, Z))</td>
<td>(B[V])</td>
<td>(I[V_0])</td>
</tr>
<tr>
<td>5</td>
<td>((Y, Y))</td>
<td>(B[E'[E[X_1^{Exp}]])</td>
<td>(X_1^{Int})</td>
</tr>
<tr>
<td>6</td>
<td>((Y, Z))</td>
<td>(B[E'[E[X_1^{Exp}]])</td>
<td>(I[X_1^{Int}])</td>
</tr>
<tr>
<td>7</td>
<td>((Y, Y))</td>
<td>(OE[X_1^{Exp}])</td>
<td>(O_1 I[X_1^{Int}])</td>
</tr>
<tr>
<td>8</td>
<td>((Y, Z))</td>
<td>(OE[X_1^{Exp}])</td>
<td>(O_3 I[X_1^{Int}])</td>
</tr>
<tr>
<td>9</td>
<td>((Z, Y))</td>
<td>(VE[X_1^{Exp}])</td>
<td>(V_1 I^<em>[X_1^{Int</em>}])</td>
</tr>
<tr>
<td>10</td>
<td>((Z, Z))</td>
<td>(VE[X_1^{Exp}])</td>
<td>(V_3 I^<em>[X_1^{Int</em>}])</td>
</tr>
<tr>
<td>11</td>
<td>((Y, Y))</td>
<td>(BOE[C X_1^{Exp}])</td>
<td>(O_2 II[C_0 X_1^{Int}])</td>
</tr>
<tr>
<td>12</td>
<td>((Y, Z))</td>
<td>(BOE[C X_1^{Exp}])</td>
<td>(I[O_2 II[C_0 X_1^{Int}]])</td>
</tr>
<tr>
<td>13</td>
<td>((Y, Y))</td>
<td>(BOE[V X_1^{Exp}])</td>
<td>(V_1 I^*[O_3 I[X_1^{Int}]])</td>
</tr>
<tr>
<td>14</td>
<td>((Y, Z))</td>
<td>(BOE[V X_1^{Exp}])</td>
<td>(V_3 I^*[O_3 I[X_1^{Int}]])</td>
</tr>
<tr>
<td>15</td>
<td>((Y, Y))</td>
<td>(BOE[E'[E[X_1^{Exp} Exp]] X_2^{Exp} Exp)</td>
<td>(O_2 II[X_1^{Int} Int X_2^{Int Int}])</td>
</tr>
<tr>
<td>16</td>
<td>((Y, Z))</td>
<td>(BOE[E'[E[X_1^{Exp} Exp]] X_2^{Exp} Exp)</td>
<td>(I[O_2 II[X_1^{Int} Int X_2^{Int Int}]])</td>
</tr>
<tr>
<td>17</td>
<td>((Y, Y))</td>
<td>(BVE[C X_1^{Exp}])</td>
<td>(V_2 II[C_0 X_1^{Int}])</td>
</tr>
<tr>
<td>18</td>
<td>((Y, Z))</td>
<td>(BVE[C X_1^{Exp}])</td>
<td>(I[V_2 II[C_0 X_1^{Int}]])</td>
</tr>
<tr>
<td>19</td>
<td>((Z, Y))</td>
<td>(BVE[V X_1^{Exp}])</td>
<td>(V_1 I^<em>[V_3 I^</em>[X_1^{Int*}]])</td>
</tr>
<tr>
<td>20</td>
<td>((Z, Z))</td>
<td>(BVE[V X_1^{Exp}])</td>
<td>(V_3 I^<em>[V_3 I^</em>[X_1^{Int*}]])</td>
</tr>
<tr>
<td>21</td>
<td>((Y, Y))</td>
<td>(BVE[E'[E[X_1^{Exp} Exp]] X_2^{Exp} Exp)</td>
<td>(V_2 II[X_1^{Int} Int X_2^{Int Int}])</td>
</tr>
<tr>
<td>22</td>
<td>((Y, Z))</td>
<td>(BVE[E'[E[X_1^{Exp} Exp]] X_2^{Exp} Exp)</td>
<td>(I[V_2 II[X_1^{Int} Int X_2^{Int Int}]])</td>
</tr>
</tbody>
</table>

Figure 6. The type \( r' \) and the representations of its operations in \( \overline{r_1} \) and \( \overline{r_2} \)
Let $G_i$ be a context-free grammar of type $r_i: P_i \to N_{i1}^* \times N_{i2}^*$ for $i=1,2$. Let $\sigma_i: W_i \to ST_i$ be the concrete syntax homomorphism from $W_i$ to the string $r_i$-algebra $ST_i$ for $i=1,2$. Then a set $T \subseteq \bigcup_{(n_1, n_2) \in S_1 \times S_2} W_1, n_1 \times W_2, n_2$ for $S \subseteq S_1 \times S_2$ is called a tree transduction. Furthermore, $T$ induces a string transduction $ST_T = \{(\sigma_1(t_1), \sigma_2(t_2)): (t_1, t_2) \in T\}$.

As mentioned before, we are primarily interested in transductions which preserve semantics. We will call such transductions translations. We will say that a transduction $T$ (or $ST_T$) is "semantic-preserving" if for every $(t_1, t_2) \in T$ (or $(s_1, s_2) \in ST_T$), $t_1$ and $t_2$ (or $s_1$ and $s_2$) have a meaning in common. The formal definitions follow.

Let $D_i = (G_i, A_i)$ be an LDS where $G_i = (N_i, \Sigma_i, P_i, S_i)$ has type $r_i$ and word algebra $W_i$ for $i=1,2$. Let $\sigma_i: W_i \to ST_i$ be the concrete syntax homomorphism from $W_i$ into the string $r_i$-algebra $ST_i$, and let $\mu_i: W_i \to A_i$ be the semantic homomorphism from $W_i$ into the semantic algebra $A_i$ for $i=1,2$. Then a tree transduction $T \subseteq \bigcup_{(n_1, n_2) \in S_1 \times S_2} W_1, n_1 \times W_2, n_2$ for $S \subseteq S_1 \times S_2$ is a semantic-preserving tree transduction between $D_1$ and $D_2$, or a tree translation, provided $\mu_1(t_1) = \mu_2(t_2)$ for all $(t_1, t_2) \in T$. Furthermore, the string transduction $ST_T$ induced by a tree translation is said to be a language translation.

We have seen that a tree transduction is merely a relation between two algebras. As yet, we have no means of performing a transduction other than by consulting a table for the entire transduction relation. This clearly is not a satisfactory situation. In most cases it is impractical, if not impossible, to maintain such a table for all possible inputs. We will now see how the notions of derived and represented types and homomorphisms between algebras can be used to define tree transducers.
for which we can evaluate the output associated with a given input by considering the input tree to be composed of derived operations. In many cases, this will enable us to write a finite table, called a "translation table," for a transduction.

Let \( D = (G, A) \) be an LDS where \( G = (N, E, P, S) \) has type \( r: P \rightarrow N^{*} \times N^{*} \) with word algebra \( W \) for \( i = 1, 2 \). Let \( r': P' \rightarrow N'^{*} \times N'^{*} \) be representable in both \( r_1 \) and \( r_2 \), and let \( \overline{W}_1 \) be the \( r' \)-algebra represented in the word \( r_1 \)-algebra \( \overline{W}_1 \) for \( i = 1, 2 \). Also, let \( h_1: W' \rightarrow \overline{W}_1 \) be the unique homomorphism from \( W' \) into \( \overline{W}_1 \) for \( i = 1, 2 \). Then the relation

\[
T = h_2 \circ h_1^{-1} \subseteq \overline{W}_1 \times \overline{W}_2
\]

is called the algebraic tree transducer induced by the representations of \( r' \) in \( r_1 \) and \( r_2 \).

Furthermore, if for some \( S \subseteq N' \) we have \( H_i(s) \in S_i \) for all \( s \in S \) and for \( i = 1, 2 \), then \( T \) is said to induce a tree transduction \( T_i = \{(h_1(w), h_2(w)) : w \in \bigcup_{s \in S} W' \} \).

(III.7) Example. Let us construct an algebraic tree transducer to convert syntax trees of the grammar APL of type \( r_1 \) given in Figure 4 into syntax trees of the grammar INT of type \( r_2 \) given in Figure 7.

First we must find a type \( r' \) which is representable in both \( r_1 \) and \( r_2 \) and specify functions \( H_1, \pi_1, \eta_1, \) and \( \pi_2 \) which induce the representations. Then we need only construct the appropriate homomorphisms, and our work will be done.

We know that the type \( r' \) given in Figure 6 is representable in \( r_1 \) via the functions \( H_1 \) and \( \pi_1 \) where \( H_1(n) = \text{Exp} \) for all \( n \in N' \) and \( \pi_1 \) is as given in Figure 6. We claim that the type \( r' \) is also representable in \( r_2 \) via \( H_2: N' \rightarrow N_2 \) and \( \pi_2: P' \rightarrow P_2 \) where \( H_2(y) = \text{Int} \) and \( \pi_2(z) = \text{Int}^* \) and \( \pi_2 \) is as given in Figure 6.
INT = (N_2, E_2, P_2, S_2) where
N_2 = \{Int, Int^*\}, E_2 = \{Con_0, Var_0, Var_1, Var_2, Var_3, Op_0, Op_1, Op_2, Op_3\},
S_2 = \{Int\}, and P_2 is given below.

<table>
<thead>
<tr>
<th>Name of production p</th>
<th>Actual production p</th>
<th>r_2 (p)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_0</td>
<td>Int \to Con_0</td>
<td>(\lambda, Int)</td>
</tr>
<tr>
<td>V_0</td>
<td>Int \to Var_0</td>
<td>(\lambda, Int)</td>
</tr>
<tr>
<td>O_1I</td>
<td>Int \to Op_1 Int</td>
<td>(Int, Int)</td>
</tr>
<tr>
<td>V_1I*</td>
<td>Int \to Var_1 Int*</td>
<td>(Int^*, Int)</td>
</tr>
<tr>
<td>O_2II</td>
<td>Int \to Op_2 Int Int</td>
<td>(Int Int, Int)</td>
</tr>
<tr>
<td>V_2II</td>
<td>Int \to Var_2 Int Int</td>
<td>(Int Int, Int)</td>
</tr>
<tr>
<td>I</td>
<td>Int^* \to Int</td>
<td>(Int, Int^*)</td>
</tr>
<tr>
<td>O_3I</td>
<td>Int^* \to Op_3 Int</td>
<td>(Int, Int^*)</td>
</tr>
<tr>
<td>V_3I*</td>
<td>Int^* \to Var_3 Int*</td>
<td>(Int^<em>, Int^</em>)</td>
</tr>
</tbody>
</table>

Figure 7. The grammar INT and its type

Let W_i be the word r_i-algebra for i=1,2. Then the homomorphisms we require to define our transducer \tau are h_1: W' \to \overline{W}_1' and h_2: W' \to \overline{W}_2' where W' is the word r'-algebra and \overline{W}_1' and \overline{W}_2' are the r'-algebras represented in the derived word r_1- and r_2-algebras, respectively. We recall that for w = p_{W'}(w_1...w_k)eW', we have h_i(p_{W'}(w_1...w_k)) = \pi_i(p_{\overline{W}_i}(h_1(w_1)...h_1(w_k))) for i=1,2. Thus h_1 and h_2 are readily available from the finite table in Figure 6. (We will call any tabular form of determining the output of an algebraic tree transducer based on components of the input tree a translation table for the transduction.
We note that whenever the common representable type is finite, we can produce a finite translation table for the transduction. Examples of translation tables are given in Figures 6 and 9.) Since \( S_1 = \{ \text{Exp} \} \) and \( S_2 = \{ \text{Int} \} \) while \( \eta_1(Y) = \text{Exp} \) and \( \eta_2(Y) = \text{Int} \), we have \( \tau = h_2 \circ h_1^{-1} \) and \( \text{Tr}_\tau = \{(h_1(w), h_2(w)) : w \in W' \} \).

We will examine the action of \( \tau \) on one element of \( W'_Y \), namely \( 19_{W'}(20_{W'}(2_{W'})) \). Applying the homomorphism \( h_i \) for \( i=1,2 \), we see that

\[
\begin{align*}
\pi_{1}(19) & = (Z,Y), \\
\pi_{1}(20) & = (Z,Z), \\
\pi_{1}(2) & = (\lambda, Z).
\end{align*}
\]

Now, evaluating this expression according to Figure 6 for \( i=1 \), we get

\[
\begin{align*}
\pi_{1}(19) & = (BVE(V BVE(V B(C)))) \\
\pi_{1}(20) & = (BVE(V BVE(V B(C)))) \\
\pi_{1}(2) & = (V I*(V I*(V I*(I(C_0))))).
\end{align*}
\]

For the sake of simplicity, we have made no mention of the semantic algebras associated with either APL or INT, and so we cannot comment on whether or not the algebraic tree transducer \( \tau \) we have specified is semantic-preserving.

Semantic-preserving tree transducers are, of course, necessary for correct translators. In this regard, we would like to be able to determine without exhaustive checking whether an algebraic tree
transducer is semantic-preserving. The following definition and theorem will give us a means of doing this.

Let $D_i = (G_i, A_i)$ be an LDS where $G_i$ has type $r_i$ with word algebra $W_i$ for $i=1,2$. Let $r'$ be representable in both $r_1$ and $r_2$. In addition, let $\mu_i : W_i \rightarrow A_i$ be the semantic homomorphism of $D_i$ for $i=1,2$. Let $\tau$ be the algebraic tree transducer induced by the representations of $r'$ in $r_1$ and $r_2$. Then $\tau$ is operation preserving if

1) $\mu_{1,n}(h_1,(W_n)) = \mu_{2,n}(h_2,(W_n)) = Z_n$ for all $n \in \mathbb{N}$
2) $p_{A_1}^a |Z_0 = p_{A_2}^a |Z_0$ for all $p \in P'$ with $r'(p) = (a,n)$ where $p_{A_i}^a |Z_0$ denotes the restriction of $p_{A_i}^a$ to $Z_0$ for $i=1,2$.

(III.8) Theorem. If $\tau$ is an operation preserving algebraic tree transducer, then $T_x$ is a tree translation and $S_x$ is a language translation.

There do exist algebraic tree transducers which are not operation preserving but which do induce tree translations (i.e., semantic-preserving tree transductions). However, it is not easy to determine, given $\tau$ which is not operation preserving, whether or not $T_x$ is semantic-preserving.

In summary, we have shown how the sets of syntax trees and strings generated by a context-free grammar can be given an algebraic interpretation. We have shown how to use the notions of defined and represented types and homomorphisms between algebras to define tree transductions on syntax trees which also induce string transductions.
between languages. (See Figure 8.) Finally, we have presented a condition under which the transductions induced by algebraic tree transducers are semantic-preserving.

\[ \tau = h_2 \cdot h_1^1 \]

Figure 8. Pictorial representation of an algebraic tree transducer
IV. A COMPARISON OF AUTOMATA-THEORETIC AND ALGEBRAIC APPROACHES TO TREE TRANSDUCTION

In Chapters II and III we have explored both automata-theoretic and algebraic approaches to tree transduction. In this chapter we will examine the relationship between the two approaches.

We saw in Chapter II that neither the class of top-down nor the class of bottom-up tree transducers is closed under composition. In contrast, (IV.1) shows that algebraic tree transducers are closed under composition. Hence algebraic tree transducers are not equivalent to either top-down or bottom-up tree transducers. We shall see, in fact, that algebraic tree transducers are more powerful than either top-down or bottom-up tree transducers.

(IV.1) Theorem. Let $\tau_1$ and $\tau_2$ be algebraic tree transducers. Then there exists an algebraic tree transducer $\tau$ such that $\text{Tr}_\tau$ is precisely the transduction induced by the composition $\tau_2 \circ \tau_1$.

Proof. Let $D_i = (G_i, A_i)$ be an LDS where $G_i$ has type $r_i : P_i \rightarrow N_i \times N_i^*$ with word $r_i$-algebra $W_i$ for $i=1,2,3$. Let $r' : P' \rightarrow N' \times N'$ be representable in both $r_1$ and $r_2$, and let $r'' : P'' \rightarrow N'' \times N''$ be representable in both $r_2$ and $r_3$. Let $W'$ and $W''$ be the word $r'$- and $r''$-algebras, respectively. Let $f_1 : W' \rightarrow \overline{W}_1$, $f_2 : W' \rightarrow \overline{W}_2$, $g_1 : W'' \rightarrow \overline{W}_1$, and $g_2 : W'' \rightarrow \overline{W}_2$ be the indicated unique homomorphisms. Let $\tau_1 = f_2 \circ f_1^{-1}$ and $\tau_2 = g_2 \circ g_1^{-1}$ be the resulting algebraic tree transducers. Then $\text{Tr}_{\tau_2 \circ \tau_1} = \{(t_1, t_3) \in \overline{W}_1 \times \overline{W}_3 : \text{there exists } t_2 \in \overline{W}_2 \cap \overline{W}_2'' \text{ such that } (t_1, t_2) \in \text{Tr}_{\tau_1} \text{ and } (t_2, t_3) \in \text{Tr}_{\tau_2}\}$. 

Let \( \tau^# : P \rightarrow \mathbb{N}^* \times \mathbb{N}^* \) be a type where \( P = \text{Tr} \) and \( N = \mathbb{N} \times \mathbb{N} \). For all \( p = (t_1, t_3) \in P \), let \( \tau^#(p) = \tau^#((t_1, t_3)) = (\lambda, (n_1, n_3)) \) whenever \( t_1 \in \mathbb{W}_{1, n_1}^* \) and \( t_3 \in \mathbb{W}_{3, n_3}^* \). Let us define \( \eta_1 : N \rightarrow N_1 \), \( \pi_1 : P \rightarrow P_1 \), \( \eta_3 : N \rightarrow N_3 \), and \( \pi_3 : P \rightarrow P_3 \) by \( \eta_1((n_1, n_3)) = n_1 \) for all \( (n_1, n_3) \in N \) and \( \eta_3((n_1, n_3)) = n_3 \) for all \( (n_1, n_3) \in N \). Then \( \tau^# \) is representable in \( \tau_1 \) and \( \tau_3 \) because for \( i = 1, 3 \) we have \( \tau_1^i((n_1, n_3)) = \tau_1^i(t_1, t_3) = (\lambda, (n_1, n_3)) \) for all \( (n_1, n_3) \in N \) and \( (t_1, t_3) \in P \). Let \( h_1 : \mathbb{W}^* \rightarrow \mathbb{W}^* \) be the unique homomorphism from the word \( \tau^# \)-algebra \( \mathbb{W}^* \) into the derived word \( \tau^# \)-algebra \( \mathbb{W}^* \) for \( i = 1, 3 \), and let \( \tau = h_3 h_1^{-1} \). Then \( \text{Tr}_\tau = \{(h_1(w), h_3(w)) : w \in \mathbb{W}^* \} \) where \( I = \{ n \in N : \eta_1(n) \in S_1 \text{ and } \eta_3(n) \in S_3 \} \) and \( S_i \) is the set of start symbols of \( G_i \) for \( i = 1, 3 \).

We claim that \( \text{Tr}_\tau = \text{Tr}_{\tau_2 \circ \tau_1} \). Let \( (t_1, t_3) \in \text{Tr}_\tau \). Then \( t_1 = h_1(w) \) and \( t_3 = h_3(w) \) for some \( w \in \mathbb{W}^* \), where \( \eta_1(n) \in S_1 \) and \( \eta_3(n) \in S_3 \). But since \( \mathbb{W}^* \) has type \( \tau^# : P \rightarrow \mathbb{N}^* \times \mathbb{N}^* \) and \( P \) consists only of nullary operation symbols, we must have \( w = p \) for some \( p \in P \). Furthermore since \( h_1(p) = t_1 \), we have \( \eta_1(p) = t_1 \) for \( i = 1, 3 \), and hence \( p = (t_1, t_3) \in \text{Tr}_{\tau_2 \circ \tau_1} \). Thus \( \text{Tr}_\tau \subseteq \text{Tr}_{\tau_2 \circ \tau_1} \).

Conversely, if \( (t_1, t_3) \in \text{Tr}_{\tau_2 \circ \tau_1} \), then \( (t_1, t_3) \in P \) and \( \tau^#((t_1, t_3)) = (\lambda, (n_1, n_3)) \) for some \( n_1 \in S_1 \) and \( n_3 \in S_3 \). Consequently, \( \pi_1((t_1, t_3)) = \pi_1((t_1, t_3)) = t_1 \), \( \pi_3((t_1, t_3)) = t_3 \), so that \( (t_1, t_3) \in \text{Tr}_\tau \). Thus \( \text{Tr}_{\tau_2 \circ \tau_1} \subseteq \text{Tr}_\tau \), and so \( \text{Tr}_\tau = \text{Tr}_{\tau_2 \circ \tau_1} \).

Unfortunately, (IV.1) is solely of theoretical, rather than practical, value because the transducer \( \tau \) which performs the composition \( \tau_2 \circ \tau_1 \) is based on an infinite type \( \tau^# \) even when \( \tau_1, \tau_2, \) and \( \tau_3 \) are all finite types. The resulting transducer \( \tau \) has an infinite translation table which is simply the relation \( \tau \) itself.
As a direct result of (IV.1) and the fact that neither the class of top-down nor bottom-up tree transducers is closed under composition, we have the following theorem.

(IV.2) Theorem. The class of algebraic tree transducers is not equivalent to either the class of top-down or the class of bottom-up tree transducers.

For applications to computing, we are primarily interested in transducers which, like the automata-theoretic transducers we considered in Chapter II, can be implemented using finite translation tables. As we commented in (III.7), an algebraic tree transducer $\tau$ has a finite translation table whenever $\tau = h_2 oh_1^{-1}$ where $h_1$ and $h_2$ are both homomorphisms of $r'$-algebras for some finite type $r'$. For the remainder of this paper, we will use the term algebraic tree transducer to refer to those transducers which have finite translation tables. When we wish to refer to transducers which may be built from homomorphisms of algebras of an infinite type, we will use the term infinite algebraic tree transducer.

We will now examine the relationship between algebraic tree transducers and bottom-up tree transducers. We will see that the class of bottom-up tree transducers is equivalent to a proper subclass of the class of algebraic tree transducers. We will also investigate an automata-theoretic transducer powerful enough to model an algebraic tree transducer.
Example. Let us consider the translation table given in Figure 9. We will first construct an algebraic tree transducer which induces the required transduction.

Let \( r_1 : P_1 \rightarrow N_1^* \times N_1^* \) be a type where \( P_1 = \{a, b, c, d, e\} \), \( N_1 = \{q\} \),
\[ r_1(a) = (q, q), \quad r_1(b) = (qq, q), \quad r_1(c) = r_1(d) = (\lambda, q), \quad \text{and} \quad r_1(e) = (qq, q). \]
Let \( W_1 \) be the word \( r_1 \)-algebra. Then the trees generated by the inputs in Figure 9 are all elements of \( W_1[q] \).

Let \( r_2 : P_2 \rightarrow N_2^* \times N_2^* \) be a second type where \( P_2 = \{A, B, C, D, E\} \),
\[ N_2 = \{q\}, \quad r_2(A) = (qq, q), \quad r_2(b) = (qqq, q), \quad r_2(C) = r_2(D) = (\lambda, q), \quad \text{and} \quad r_2(E) = (qqqq, q). \]
Let \( W_2 \) be the word \( r_2 \)-algebra. Then the trees generated by the outputs in Figure 9 are all elements of \( W_2[q] \).

Let \( r' : P' \rightarrow N'^* \times N'^* \) be yet a third type where \( P' = \{1, 2, 3, 4, 5\} \),
\[ N' = \{q\}, \quad r'(1) = r'(2) = (\lambda, q), \quad r'(3) = r'(4) = (qq, q), \quad \text{and} \quad r'(5) = (qqq, q). \]
Then \( r' \) is representable in both \( r_1 \) and \( r_2 \) via \( \eta_1(q) = \eta_2(q) = q \) and \( \pi_1 \) and \( \pi_2 \) as shown in Figure 10. If we let \( W' \) be the word \( r' \)-algebra, then the unique homomorphisms \( h_1 : W' \rightarrow W_1[q] \) and \( h_2 : W' \rightarrow W_2[q] \) define an algebraic tree transducer \( \tau = h_2 h_1^{-1} \) which induces the transduction generated by the translation table of Figure 9.

We claim that this same transduction cannot be performed by any bottom-up tree transducer. We will try to construct one and see where the difficulties arise. Certainly the first two rows of the translation table can be implemented by the rules \( c \rightarrow q(C) \) and \( d \rightarrow q(D) \). To implement the third row of the table, our first attempt might be to encode the input symbols we read into the state of the transducer until we read the root of the expression which appears in the input column, at which time we may produce the entire output expression. This
Figure 9. A translation table for an algebraic tree transducer
<table>
<thead>
<tr>
<th>p</th>
<th>r'(p)</th>
<th>π₁(p)</th>
<th>π₂(p)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(λ,q)</td>
<td>c</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>(λ,q)</td>
<td>d</td>
<td>D</td>
</tr>
<tr>
<td>3</td>
<td>(qq,q)</td>
<td>a[b[a[x₁^{qq}] a[x₂^{qq}]])</td>
<td>A[x₁^{qq} x₂^{qq}]</td>
</tr>
<tr>
<td>4</td>
<td>(qq,q)</td>
<td>b[a[x₁^{qq}] x₂^{qq}]</td>
<td>B[x₁^{qq} x₂^{qq}]</td>
</tr>
<tr>
<td>5</td>
<td>(qqq,q)</td>
<td>e[b[a[x₁^{qqq}] a[x₂^{qqq}]] b[a[x₃^{qqq}] a[x₄^{qqq}]]</td>
<td>E[x₁^{qqq} x₂^{qqq} x₃^{qqq} x₄^{qqq}]</td>
</tr>
</tbody>
</table>

Figure 10. A representation of r' in \( \bar{r}_1 \) and \( \bar{r}_2 \)
strategy would cause us to add the rule $a(q(x_1)) \rightarrow q_a(x_1)$ to our proposed set of rules. However, using this strategy we encounter difficulty when we read $b(q_a(x_1) q_a(x_2))$. No new output is called for at this point, but without new output we cannot save both the $x_1$ and $x_2$ outputs. Both will be needed for the output expression we are trying to create. The only other reasonable alternative is to introduce nondeterminism at this point, i.e., to guess that the $b$ we are reading is part of an expression which matches the third row, rather than the fifth row of Figure 9. This causes us to add $b(q_a(x_1) q_a(x_2)) \rightarrow q_b(A(x_1 x_2))$ and $a(q_b(x_1)) \rightarrow q(x_1)$ to our set of rules.

Unfortunately, we cannot use the same trick to implement the fifth row of the translation table. We cannot predict that we are reading a portion of the fifth entry since we need the expressions represented by $x_1$, $x_2$, $x_3$, and $x_4$ to create the output. These are not available as common descendants of one node before reading the $e$ at the root of the expression. Furthermore, we cannot pass all four of these expressions up to the $e$ for the same reason that we could not pass the expressions represented by $x_1$ and $x_2$ up to the $a$ at the root of the expression in the third row. Consequently, this transduction cannot be effected by any bottom-up tree transducer. Hence algebraic tree transducers are not equivalent to bottom-up tree transducers.

There is, however, an interesting connection between bottom-up tree transducers and algebraic tree transducers. We will see that bottom-up tree transducers are equivalent to a restricted form of algebraic tree transducers. To this end, we present the following definition and theorem.
Let $\tau = h_2 \circ h_1^{-1}$ be an infinite algebraic tree transducer where $h_1: W' \to \overline{W}_1'$ is a homomorphism of $r'$-algebras, $W_1$ has type $r_1: P_1 \to N_1^* \times N_1^*$, and $r': P' \to N'^* \times N'^*$ is representable in $r_1$ via $\eta_1: N' \to N_1$ and $\pi_1: P' \to \overline{P}_1$ for $i=1,2$. We say that $\tau$ is simple if $r'$ is representable in $r_1$, as well as in $\overline{r}_1$, via $\eta_1$ and $\pi_1$. That is, $\tau$ is simple if $\pi_1(P') \subseteq P_1$.

(IV.4) Theorem. The class of transductions induced by simple algebraic tree transducers is equal to the class of transductions induced by bottom-up tree transducers.

Proof. Given a bottom-up tree transducer, we wish to show that there exists a simple algebraic tree transducer which induces the same transduction.

Let $B = (\Sigma, \Delta, Q, Q^*, R)$ be a bottom-up tree transducer. We recall that the transduction $\text{Tr}_B$ induced by $B$ is given by $\text{Tr}_B = \{(t, t') \in T^* \times T^*_\Delta: t \not\approx_B q(t') \text{ for some } q \in Q^*\}$.

In order to define a simple algebraic tree transducer which induces $\text{Tr}_B$, we must first define language definition systems $D_1$ and $D_2$ whose respective algebras are $T_\Sigma$ and $T_\Delta$. Let $G_1 = (Q_1, \Sigma, P_1, Q_1, R)$ be a context-free grammar where $P_1 = \{q \to s q_1 \ldots q_k: s(q_1(x_1) \ldots q_k(x_k)) \to q(t) \in R \text{ for some } k > 0 \text{ and } t \in T_\Delta[X_k]\} \cup \{q \to s: s = q(t) \in R \text{ for some } t \in T_\Delta\}$. Then the type of $G_1$ is $r_1: P_1 \to Q^* \times Q^*$ where we define $r_1(q \to s q_1 \ldots q_k) = (q_1 \ldots q_k, q)$ and $r_1(q \to s) = (\lambda, q)$ for each production in $P_1$. Whenever its type is clear from the context, we refer to $q \to s q_1 \ldots q_k$ or $q \to s$ simply as $s$. Let $W_1$ be the word $r_1$-algebra, and let $A$ be any $r_1$-algebra. We note that
UW₁ = ∪q∈Q W₁,q = T₂, the set of input trees of B. D₁ = (G₁,A₁) will serve as our LDS for T₂.

In addition, let G₂ = (Z,Δ,P₂,Z) be a context-free grammar where 
P₂ = {Z → dZδ(d) : d∈Δ}, where δ is the ranking function on Δ, and let the name of Z → dZδ(d) be d for each production in P₂. Then the type of G₂ is r₂ : P₂ → {Z}сх{Z}*, where for each d∈P₂, r₂(d) = (Zδ(d),Z) where Z₀ = λ. Let W₂ be the word r₂-algebra, and let A₂ be any r₂-algebra. Then UW₂ = W₂,Z = T₂, the set of output trees of B. D₂ = (G₂,A₂) will serve as our LDS for T₂.

Next we need a type r' which is representable in both r₁ : P₁ → QсхQ* and r₂ : P₂ → {Z}сх{Z}*, the type derived from r₂. The common representable type will be deduced from R, the set of rules of B, since the input and output trees are related through R. We first observe that if ρ∈R is of the form s → q(t) for some s∈Z₀, q∈Q, and t∈T₂, then s∈P₁ since s is the name of q → s. Also t∈T₂, and hence t = d(t₁...tₖ) for some d∈Δ, k>0, and t₁,...,tₖ∈T. Consequently, by our inductive definition of P₂, t = d(t₁...tₖ)∈P₂. Furthermore, if ρ∈R is of the form s(q₁(x₁)...qₖ(xₖ)) → q(t) for some s∈Σₖ, q₁,...,qₖ∈Q, and t∈T₂[Σₖ], then by reasoning similar to that stated above, we find that if α = q₁...qₖ, then s∈P₁ and t = x₁...xₖ∈P₂. Thus we see that for all intents and purposes, the left-hand sides of the rules of R are elements of P₁, while the right-hand sides similarly resemble elements of P₂.

With these observations in mind, we can proceed to define r' and to give the necessary representations.
Let us define \( r': R \rightarrow Q^* \times Q^* \), where \( R \) is the set of rules of \( B \) and \( Q \) the set of states, by \( r'(\rho) = (\lambda, q) \) whenever \( \rho \) is of the form \( s \rightarrow q(t) \) and \( r'(\rho) = (q_1 \ldots q_k, q) \) whenever \( \rho \) is of the form \( s(q_1(x_1) \ldots q_k(x_k)) \rightarrow q(t) \). Furthermore, let \( W' = \{ W'_q \}_{q \in Q}, \{ \rho_{W'} \}_{\rho \in R} \) be the word \( r'-\text{algebra} \). Clearly \( r' \), \( r_1 \), and \( r_2 \) are all finite types since \( R \) is finite.

To achieve the necessary representations of \( r' \) in \( r_1 \) and \( r_2 \), we let \( \eta_1: Q \rightarrow Q \) be the identity function and \( \eta_2: Q \rightarrow \{ Z \} \) be the obvious constant function, and we extend \( \eta_1 \) and \( \eta_2 \) to domain \( Q^* \) in the usual manner. In addition, we define \( \pi_1: R \rightarrow P_1 \) and \( \pi_2: R \rightarrow P_2 \) by \( \pi_1(\rho) = s \) and \( \pi_2(\rho) = t \), whenever \( \rho \) is of the form \( s(q_1(x_1) \ldots q_k(x_k)) \rightarrow q(t) \) and \( x_1 \ldots x_k \). Thus if \( \rho \) is \( s \rightarrow q(t) \), then \( r'(\rho) = (\lambda, q) \) and \( r_1(\pi_1(\rho)) = r_1(s) = (\lambda, q) = (\eta_1(\lambda), \eta_1(q)). \) Also, if \( \rho \) is \( s(q_1(x_1) \ldots q_k(x_k)) \rightarrow q(t) \), then \( r'(\rho) = (q_1 \ldots q_k, q) = (\alpha, q) \) and \( r_1(\pi_1(\rho)) = r_1(s) = (\alpha, q) = (\eta_1(\alpha), \eta_1(q)). \) Consequently \( r' \) is representable in \( r_1 \).

We can similarly show that \( r' \) is representable in \( r_2 \).

Let \( W_1' \) and \( W_2' \) be the \( r' \)-algebras represented in \( W_1 \) and the derived \( r_2 \)-algebra \( W_2' \), respectively, and let \( h_1: W' \rightarrow W_1' \) and \( h_2: W' \rightarrow W_2' \) be the unique homomorphisms from \( W' \) into the represented algebras. Then the relation \( \tau = h_2 \circ h_1^{-1} \subseteq W_1' \times W_2' \) is the simple algebraic tree transducer induced by the representations of \( r' \) in \( r_1 \) and \( r_2 \). Furthermore, the transduction induced by \( \tau \) is \( \text{Tr}_\tau = \{(h_1(w), h_2(w)): w \in U_{q \in \{ q \in Q: \eta_1(q) = q \}} W' \} \) since \( Q_d = \{ q \in Q: \eta_1(q) \in Q_d \} \) and \( \eta_2(q) = Z \).

We claim that \( \text{Tr}_\tau \subseteq \text{Tr}_B \). The details of the proof may be found in the Appendix.
It remains for us to show that for every simple algebraic tree transducer \( \tau \), there exists a bottom-up tree transducer \( B \) such that
\[
\text{Tr}_B = \text{Tr}_\tau.
\]

Let \( D_1 = (G_1, A_1) \) be an LDS, and let \( S_1 \) be the set of start symbols of \( G_1 \) for \( i=1,2 \). Let \( G_1 \) have finite type \( r_1 : P_1 \to N_1^* \times N_1^* \) with word \( r_1 \)-algebra \( W_1 \) for \( i=1,2 \). Let the finite type \( r' : P' \to N' \times N' \) be representable in \( r_1 \) via \( \eta_1 : N' \to N_1 \) and \( \pi_1 : P' \to P_1 \) and in \( r_2 \) via \( \eta_2 : N' \to N_2 \) and \( \pi_2 : P' \to P_2 \). Let \( W' \) be the word \( r' \)-algebra, and let \( W'_1 \) and \( W'_2 \) be the \( r' \)-algebras represented in \( W_1 \) and \( W_2 \) respectively.

Let \( h_1 : W' \to W'_1 \) and \( h_2 : W' \to W'_2 \) be the unique homomorphisms which determine the simple algebraic tree transducer \( \tau = h_2 \circ h_1 \). Let \( I = \{ n \in N' : \eta_1(n) \in S_1 \text{ and } \eta_2(n) \in S_2 \} \). We recall that the transduction \( \text{Tr}_\tau \) induced by \( \tau \) is
\[
\text{Tr}_\tau = \{ (h_1(w), h_2(w)) : w \in W_n \text{ for } n \in I \}.
\]

In order to specify a bottom-up tree transducer \( B \) which imitates \( \tau \), we must provide a set \( R \) of rules which relate \( \pi_1(p) \) and \( \pi_2(p) \) for every \( p \in P' \). Let \( B = (P_1, P_2, N', I, R) \) be a bottom-up tree transducer, as described below. Let \( P_1 \) and \( P_2 \) be ranked by \( \sigma \) and \( \delta \), respectively, where \( \sigma \) and \( \delta \) are defined as follows. For all \( p \in P \), if \( r_1(p) = (\lambda, n) \), then \( \sigma(p) = 0 \), and if \( r_1(p) = (n_1 \ldots n_k, n) \), then \( \sigma(p) = k \). Similarly, for all \( q \in P_2 \), if \( r_2(q) = (\lambda, n) \), then \( \delta(q) = 0 \), and if \( r_2(q) = (n_1 \ldots n_k, n) \), then \( \delta(q) = k \). We obtain the set \( R \) of rules of \( B \) from \( r', \pi_1, \) and \( \pi_2 \).

If \( p \in P_1 \) and \( r'(p) = (\lambda, n) \), then \( \pi_1(p) \to (\pi_2(p)) \) in \( R \). If \( p \in P_1 \) and \( r'(p) = (\lambda, n) \), then \( \pi_1(p) \to n(\pi_2(p)) \) is in \( R \). Then the transduction induced by \( B \) is
\[
\text{Tr}_B = \{(t, t') \in T_{P_1} \times T_{P_2} : \}
\]
t \overset{\star}{\Rightarrow} n(t') for some n \in \mathbb{N}\}. We claim that \text{Tr}_B = \text{Tr}_\tau. The details of the proof are in the Appendix.

Thus we have shown that bottom-up tree transducers are equivalent to simple algebraic tree transducers. We will illustrate (IV.4) with two examples.

(IV.5) Example. Let \( B \) be the bottom-up tree transducer given in (II.9). We will construct a simple algebraic tree transducer \( \tau \) such that \( \text{Tr}_\tau = \text{Tr}_B \).

Let \( G_1 = (\{q\}, \{a_0, a_1, \sigma\}, P_1, \{q\}) \) be a context-free grammar where \( \langle a \rangle \) contains
\[
q \rightarrow a_0 \quad \text{(briefly \( a_0 \))},
q \rightarrow a_1 q \quad \text{(briefly \( a_1 \))}, \text{ and}
q \rightarrow \sigma q \quad \text{(briefly \( \sigma \))}.
\]
Let \( r_1 : P_1 \rightarrow \{q\}^* \times \{q\}^* \) be the type of \( G_1 \). Then \( r_1 \) is finite, \( r_1(a_0) = (\lambda, q) \), \( r_1(a_1) = (q, q) \), and \( r_1(\sigma) = (q, q) \). Let \( \mathcal{W}_1 \) be the word \( r_1 \)-algebra. Then \( \mathcal{UW}_1 = \mathcal{W}_1, q = T_\mu \).

Let \( G_2 = (\{Z\}, \{a_0, b_0, a_1, b_1, \sigma\}, P_2, \{Z\}) \) be a second context-free grammar where \( P_2 \) contains
\[
Z \rightarrow a_0 \quad \text{(briefly \( a_0 \))},
Z \rightarrow b_0 \quad \text{(briefly \( b_0 \))},
Z \rightarrow a_1 Z \quad \text{(briefly \( a_1 \))},
Z \rightarrow b_1 Z \quad \text{(briefly \( b_1 \))}, \text{ and}
Z \rightarrow \sigma Z Z \quad \text{(briefly \( \sigma \))}.
\]
Let \( r_2 : P_2 \rightarrow \{Z\}^* \times \{Z\}^* \). Then \( r_2 \) is also finite, \( r_2(a_0) = r_2(b_0) = (\lambda, Z) \), \( r_2(a_1) = r_2(b_1) = (Z, Z) \), and \( r_2(\sigma) = (ZZ, Z) \). Let \( \mathcal{W}_2 \) be the word
Let \( r'_1: R \to \{q\}^* \times \{q\}^* \) be yet a third finite type where \( r' \) is given in Figure 11. Let \( W' \) be the word \( r'_1 \)-algebra. Then \( r' \) is representable in both \( r_1 \) and \( \bar{r}_2 \) via \( \eta_1(q) = q \) and \( \eta_2(q) = Z \) and \( \pi_1 \) and \( \pi_2 \) as shown in Figure 11. Let \( \tau = h_2 \circ h_1^{-1} \) be the simple algebraic tree transducer formed from the unique homomorphisms \( h_1: W' \to W'_1 \) and \( h_2: W' \to \bar{W}'_2 \). Then \( T_{\tau} = T_{\bar{B}} \).

<table>
<thead>
<tr>
<th>( \rho )</th>
<th>( r'(\rho) )</th>
<th>( \pi_1(\rho) )</th>
<th>( \pi_2(\rho) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_0 \to q(a_0) )</td>
<td>( (\lambda, q) )</td>
<td>( a_0 )</td>
<td>( a_0 )</td>
</tr>
<tr>
<td>( a_0 \to q(b_0) )</td>
<td>( (\lambda, q) )</td>
<td>( a_0 )</td>
<td>( b_0 )</td>
</tr>
<tr>
<td>( a_1(q(x_1)) \to q(a_1(x_1)) )</td>
<td>( (q, q) )</td>
<td>( a_1 )</td>
<td>( a_1[Z_{x_1}] )</td>
</tr>
<tr>
<td>( a_1(q(x_1)) \to q(b_1(x_1)) )</td>
<td>( (q, q) )</td>
<td>( q_1 )</td>
<td>( b_1[Z_{x_1}] )</td>
</tr>
<tr>
<td>( \sigma(q(x_1)) \to q(\sigma(x_1, x_1)) )</td>
<td>( (q, q) )</td>
<td>( \sigma )</td>
<td>( \sigma[Z_{x_1} Z_{x_1}] )</td>
</tr>
</tbody>
</table>

Figure 11. Construction of a simple algebraic tree transducer from a bottom-up tree transducer

(IV.6) **Example.** Let \( r_1, r_2, \) and \( r' \) be the finite types shown in parts (1), (2), and (3) of Figure 12, respectively. Then \( U \) must be the only start symbol of \( G_1 \), the underlying grammar of \( r_1 \). Let us assume that \( V \) is the only start symbol of \( G_2 \) underlying \( r_2 \). Then \( r' \) is representable in \( r_1 \) and \( \bar{r}_2 \) via \( \eta_1(Y) = \eta_1(Z) = U, \eta_2(Y) = V, \eta_2(Z) = W, \)
\( r_1: \{a,b,c\} \rightarrow \{U\}^* \times \{U\}^* \)

\[
\begin{align*}
    r_1(a) &= (\lambda, U) \\
    r_1(b) &= (UU, U) \\
    r_1(c) &= (U, U)
\end{align*}
\]

\( r_2: \{A,B,C,D\} \rightarrow \{V,W\}^* \times \{V,W\}^* \)

\[
\begin{align*}
    r_2(A) &= (VV, V) \\
    r_2(B) &= (\lambda, V) \\
    r_2(C) &= (\lambda, W) \\
    r_2(D) &= (W, W)
\end{align*}
\]

\( r': \{1,2,3,4\} \rightarrow \{Y,Z\}^* \times \{Y,Z\}^* \)

<table>
<thead>
<tr>
<th>p</th>
<th>( r'(p) )</th>
<th>( \pi_1(p) )</th>
<th>( \pi_2(p) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(\lambda, Y)</td>
<td>a</td>
<td>A[B B]</td>
</tr>
<tr>
<td>2</td>
<td>(\lambda, Z)</td>
<td>a</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>(YY, Y)</td>
<td>b</td>
<td>A[A[X_1^W B] X_2^W]</td>
</tr>
<tr>
<td>4</td>
<td>(Z, Z)</td>
<td>c</td>
<td>D[X_1^W]</td>
</tr>
</tbody>
</table>

\( a \rightarrow Y(A(B B)) \)

\( a \rightarrow Z(C) \)

\( b(Y(x_1) Y(x_2)) \rightarrow Y(A(A(x_1 B) x_2)) \)

\( c(Z(x_1)) \rightarrow Z(D(x_1)) \)

Figure 12. Construction of a bottom-up tree transducer from a simple algebraic tree transducer
and \( \pi_1 \) and \( \pi_2 \) as shown in part (3) of Figure 12. Let \( \tau \) be the simple algebraic tree transducer induced by these representations.

Let \( B = (\{a,b,c\}, \{A,B,C,D\}, \{Y,Z\}, \{Y\}, R) \) be the bottom-up tree transducer formed according to the construction used in the proof of (IV.4). Then \( R \) contains the rules shown in part (4) of Figure 12. Furthermore, \( Tr^B = Tr^\tau \).

As we see from (IV.5), simple algebraic tree transducers are capable of imitating nondeterministic bottom-up tree transducers. If we wish to use tree transducers to automate translation from one language to another, we prefer to use deterministic transducers whenever possible because of the extra time complexity and additional storage required to handle nondeterministic processes. Hence we would like to identify the precise relationship between deterministic bottom-up tree transducers and simple finite algebraic tree transducers.

(IV.7) **Theorem.** Let \( B = (\Sigma, \Delta, Q_0, Q, \delta, R) \) be a bottom-up tree transducer and let \( \tau = \text{h}_{\omega_{\overline{1}}}^{-1} \) be the simple algebraic tree transducer such that \( Tr^\tau = Tr^B \). Let \( \pi_1 \) and \( \pi_2 \) be the functions of the representations which relate the operation symbols of the common representable type \( r': P' \to N'^* \times N'^* \) to the operation symbols of \( r_1 \) and \( r_2 \) respectively. Then \( B \) is effectively deterministic if and only if whenever \( p, q \in P' \) are such that \( r'(p) = (a, n) \) and \( r'(q) = (a, n') \) for some \( a \in N'^* \) and \( n, n' \in N' \), it is the case that \( \pi_1(p) \neq \pi_1(q) \).

**Proof.** Suppose \( P' \) contains \( p \) and \( q \) such that \( r'(p) = (a, n) \) and \( r'(q) = (a, n') \) for some \( a = n_1 \ldots n_k \in N'^* \) and \( n, n' \in N' \), and suppose \( \pi_1(p) = \pi_1(q) \). Then \( R \) contains two rules with identical left-hand sides,
namely \[ \pi_1(p)(n_1(x_1) \ldots n_k(x_k)) \rightarrow n(\pi_2(p)_{\alpha_1}^{x_1} \ldots_{\alpha_k}^{x_k}) \text{ and } \pi_1(q)(n_1(x_1) \ldots n_k(x_k)) \rightarrow n(\pi_2(q)_{\alpha_1}^{x_1} \ldots_{\alpha_k}^{x_k}). \] Thus \( B \) is not effectively deterministic.

On the other hand, if \( B \) is not effectively deterministic, then \( B \) contains two rules with identical left-hand sides, say \( s(q_1(x_1) \ldots q_k(x_k)) \rightarrow q(t) \) and \( s(q_1(x_1) \ldots q_k(x_k)) \rightarrow q'(t') \). Corresponding to each of these rules is an operation symbol of \( P' \). That is, there exist \( p, p' \in P' \) such that \( r'(p) = (q_1 \ldots q_k, q) \), \( r'(p') = (q_1 \ldots q_k, q') \), and \( \pi_1(p) = \pi_1(p') = s \). Hence the theorem is proved.

The example in (IV.3) shows that not every algebraic tree transduction can be performed by a bottom-up tree transducer. From both the translation table of Figure 9 and the difficulties which arise as we try to construct an equivalent bottom-up tree transducer, however, we can gain some insight into what sort of automata-theoretic tree transducer is required to do the job. We can see that a transducer in which the rules have complex left-hand, as well as right-hand, sides is adequate. That is, we need rules of the form \( t \rightarrow q(t') \) where \( t \in T_{\Sigma}, q \in Q, \) and \( t' \in T \) or where \( t \in T_{\Sigma}[Q(X_k)] \) and each \( x_k \in X_k \) appears exactly once in \( t \), \( q \in Q, \) and \( t' \in T_{\Sigma}[X_k] \). The most natural implementation of such a transducer involves reading a number of nodes of the input tree simultaneously and verifying that they represent an appropriate derived operation, then generating the correct output. This technique, however, represents a significant departure from customary automata which read one input node at a time. We define below a more conventional automaton which will
allow us to simulate the occurrence of transduction rules with complex left-hand sides.

A bottom-up tree transducer with product output (product transducer)

\( P = (\Sigma, \Delta, Q, Q_d, R) \) consists of

- \( \Sigma \), a finite input alphabet ranked by a function \( r_\Sigma \),
- \( \Delta \), a finite output alphabet ranked by \( r_\Delta \),
- \( Q \), a finite set of states each having rank 1,
- \( Q_d \subseteq Q \), a set of designated (final) states, and
- \( R \), a finite set of rules.

Each rule in \( R \) is either of the form \( s \rightarrow q_1(t_1) \ldots q_k(t_k) \) for some \( s \in \Sigma_0 \), \( k > 0 \), \( q_1, \ldots, q_k \in Q \), and \( t_1, \ldots, t_k \in T \), or \( s(q_1(x_1) \ldots q_k(x_k)) \rightarrow q'_1(t_1) \ldots q'_j(t_j) \) for some \( s \) such that \( r_\Sigma(s) \geq 1 \), \( k > r_\Sigma(s) \), \( j > 0 \), \( q_1, \ldots, q_k, q'_1, \ldots, q'_j \in Q \), and \( t_1, \ldots, t_j \in T \).

We define \( T_p \), the transduction induced by \( P \), to be \( T_p = \{(t, t') \in T_\Sigma \times T_\Delta : t \xrightarrow{p} q_1(t_1) \ldots q_k(t_k) \text{ for some } k > 0, q_1, \ldots, q_k \in Q \text{, and } t_1, \ldots, t_k \in T_\Delta \text{ and } t' = t_i \text{ for some } 1 < i < k \text{ such that } q_i \in Q_d \} \).

We observe that every bottom-up tree transducer \( B = (\Sigma, \Delta, Q, Q_d, R) \) is a product transducer. The rules of \( B \) are of the form \( s \rightarrow q(t) \) where \( s \in \Sigma_0 \), \( q \in Q \), and \( t \in T_\Delta \) or \( s(q_1(x_1) \ldots q_k(x_k)) \rightarrow q(t) \) where \( s \in \Sigma_k \) for some \( k > 1 \), \( q_1, \ldots, q_k, q \in Q \), and \( t \in T_\Delta \), and hence are also rules of a product transducer.

Let us examine the behavior of a product transducer using (IV.8).

(IV.8) Example. Let \( P = (\Sigma, \Delta, Q, Q_d, R) \) where \( \Sigma_0 = \{c, d\} \), \( \Sigma_1 = \{a\} \), \( \Sigma_2 = \{b, e\} \), \( \Delta_0 = \{C, D\} \), \( \Delta_2 = \{A\} \), \( \Delta_3 = \{B\} \), and \( \Delta_4 = \{E\} \). Furthermore, let \( Q = \{q, q_a, q_baa\} \), \( Q_d = \{q\} \), and let \( R \) consist of
(1) \( c \rightarrow q(C), \)
(2) \( d \rightarrow q(D), \)
(3) \( a(q(x_1)) \rightarrow q_a(x_1), \)
(4) \( b(q_a(x_1) q_a(x_2)) \rightarrow q_{baa}(x_1) q_{baa}(x_2), \)
(5) \( a(q_{baa}(x_1) q_{baa}(x_2)) \rightarrow q(A(x_1 x_2)), \)
(6) \( b(q_a(x_1) q(x_2)) \rightarrow q(B(x_1 x_1 x_2)), \) and
(7) \( e(q_{baa}(x_1) q_{baa}(x_2) q_{baa}(x_3) q_{baa}(x_4)) \rightarrow q(E(x_1 x_2 x_3 x_4)). \)

Then \( P \) is a product transducer. It is, in fact, an automata-theoretic transducer which implements the translation table given in Figure 9, a table which cannot be implemented by any conventional bottom-up tree transducer. (See (IV.3).)

As an illustration of the operation of \( P \), let us consider its effect on the tree \( t = a(b(a(c) a(d))) \), as shown in part (1) of Figure 13. First, rules (1) and (2) are applied to the leaves of \( t \) to produce part (2). Next, rule (3) is applied twice, yielding part (3). Then rule (4) is invoked giving part (4). Finally rule (5) is applied, resulting in \( q(A(C D)) = q(t') \), as shown in part (5). Since \( q \in Q_d \), the pair \( (t, t') \in T_{r_p} \).

We will show that for every algebraic tree transducer \( \tau \), there exists a product transducer \( P \) such that \( T_{r_p} = T_{r_\tau} \). First, however, we will introduce some additional notation.

Let \( r: P \rightarrow N^* \times N^* \) be a type, and let \( \bar{r}: \bar{P} \rightarrow N^* \times N^* \) be the type derived from \( r \). For all \( t \in P \) in normal form, we define \( t^- \) as follows. If \( t = p \) for some \( p \in P \), then \( t^- = p \). If \( t = X_1^\alpha \) for some \( \alpha \in N^* \) and \( 1 \leq |\alpha| \), then \( t^- = \lambda \). If \( t = p[t_1 \ldots t_k] \) for some \( p \in P \) and \( t_1, \ldots, t_k \in \bar{P} \), then \( t^- = pt_1^- \ldots t_k^- \).
Let $\tau$ be an algebraic tree transducer. Then there exists a product transducer $P$ such that $T_{r,P} = T_{r,\tau}$.

Proof. We will ignore the underlying structures of the LDS's involved in the definition of $\tau$ as much as possible, since only their types and start symbols are relevant to our proof.

Let $\tau = h_2oh_1^{-1}$ be a finite algebraic tree transducer where $h_i: W_i \rightarrow W'_i$, $W_i$ has type $r_i: P_i \rightarrow N_i^*xN_i^*$, and the underlying grammar for $W_i$ has set of start symbols $S_i$ for $i=1,2$. Let $r': P' \rightarrow N'^*xN'^*$ be
representable in \( \overline{r} \) via \( \pi_1 : P' \to \overline{P}_1 \) and \( \eta_1 : N' \to N_1 \) for \( i=1,2 \). We can assume that for \( i=1,2 \), \( \pi_1 \) maps elements of \( P' \) to \( \overline{P}_1 \) as follows. If \( r'(p) = (\lambda, n) \), then \( \pi_1(p) \epsilon \overline{T}_P \), whereas if \( r'(p) = (n_1 ... n_k, n) \), then \( \pi_1(p) \) is in normal form. We recall that \( \overline{Tr}_1 = \{(h_1(w), h_2(w)) : w \epsilon U \epsilon I^* \} \) where \( I = \{ n \epsilon N' : \eta_1(n) \epsilon S_1 \) and \( \eta_2(n) \epsilon S_2 \} \).

Let \( P = (P_1, P_2 \cup \{d\}, Q, I, R) \) be a product transducer where \( d \) is some symbol not in \( P_2 \), \( Q \) is that finite subset of \( N' \cup P_1^* \) which occurs in the rules of \( R \), and \( R \) is built as follows. For every \( p \epsilon P' \) such that \( r'(p) = (\lambda, n) \), \( \pi_1(p) = t \epsilon \overline{P}_1 \), and \( \pi_2(p) = t' \epsilon \overline{P}_2 \),

(1) if \( t = s(t_1 ... t_k) \) for some \( s \epsilon P_1 \), \( t_1 ..., t_k \epsilon \overline{P}_1 \), and \( k \geq 0 \), and if \( t \) has \( m \) leaves, then \( s(u_1(x_1) ... u_m(x_m)) \to n(t') \epsilon R \) where for \( i=1,...,m \), \( u_1 = t_j^- \) whenever the \( i \)th leaf of \( t \) is in \( t_j \) for some \( 1 \leq j < k \).

(2) if \( s(t_1 ... t_k) \) is a proper subtree of \( t \) for some \( s \epsilon P_1 \), \( t_1 ..., t_k \epsilon \overline{P}_1 \), and \( k \geq 0 \), and if \( s(t_1 ... t_k) \) has \( m \) leaves, then \( s(u_1(x_1) ... u_m(x_m)) \to t_1^- ... t_k^- (x_1) ... t_1^- ... t_k^- (x_m) \epsilon R \) where for \( i=1,...,m \), \( u_1 = t_j^- \) whenever the \( i \)th leaf of \( s(t_1 ... t_k) \) is in \( t_j \) for some \( 1 \leq j < k \).

(3) if \( s \) is a proper subtree of \( t \) for some \( s \epsilon P_1 \), then \( s \to s(d) \epsilon R \).

Thus whenever \( p \epsilon P' \) and \( r'(p) = (\lambda, n) \), we have \( \pi_1(p) \to n(\pi_2(p)) \). In addition, for every \( p \epsilon P' \) such that \( r'(p) = (n_1 ... n_m, n) = (a, n) \) for some \( m \geq 0 \), \( \pi_1(p) = t \), and \( \pi_2(p) = t' \),

(4) if \( t = s \) for some \( s \epsilon P_1 \), then \( s(u_1(x_1) ... u_m(x_m)) \to n(t') \epsilon R \).

(5) if \( t = s(t_1 ... t_k) \) for some \( s \epsilon P_1 \), \( t_1 ... t_k \epsilon \overline{P}_1 \), and \( k \geq 0 \), then \( s(u_1(x_1) ... u_m(x_m)) \to n(t_1^a ... t_k^a) \epsilon R \) where for \( i=1,...,m \), \( u_1 = n_1 \) if \( t_i^- = \lambda \) and \( u_i = t_j^- \) if \( t_j \neq \lambda \) and \( x_i \) appears in \( t_j \) for some \( 1 \leq j < k \).
(6) if \( s(t_1 \ldots t_k) \) is a proper subtree of \( t \) for some \( s \in \mathcal{P}_1 \), \( t_1, \ldots, t_k \in \mathcal{P}_1 \), and \( k > 0 \), then \( s(u_1(x_1) \ldots u_m(x_m)) \rightarrow st_1 \ldots st_k(x_1) \ldots st_1 \ldots t_k(x_m) \in \mathcal{R} \) where for \( i = 1, \ldots, m \), \( u_i \) is as in part (5) above.

(7) if \( s \) is a proper subtree of \( t \) for some \( s \in \mathcal{P} \) such that \( r_1(s) = (\lambda, n) \), then \( s \rightarrow s(d) \in \mathcal{R} \).

Consequently, whenever \( p \in \mathcal{P}' \) and \( r'(p) = (n_1, \ldots, n_k, n) = (A, n) \), we have

\[
\pi_1(p)_{x_1^\alpha \ldots x_m^\alpha} x_1^\alpha \ldots x_m^\alpha \rightarrow n(p_2(p)_{x_1^\alpha \ldots x_m^\alpha} x_1^\alpha \ldots x_m^\alpha)
\]

Let \( I = \{ n \in \mathbb{N} : \eta_1(n) \in S_1 \} \) and \( \eta_2(n) \in S_2 \}. \) Then \( T_{p} = \{ (t, t') \in T_p \times T_p : t \rightarrow n(t') \) for some \( n \in I \} \).

We claim that \( T_{p} = T_{p+} \). The details of the proof may be found in the Appendix.

As we might suspect since the product transducer of (IV.8) does not use the full generality of the definition of the product transducer on the right-hand sides of its rules, there are product transducers whose transductions cannot be induced by algebraic tree transducers. We will see an example of such a product transducer as we study the relationship of top-down tree transducers to algebraic tree transducers and product transducers.

We note that if \( T \) is a top-down tree transducer, then there exist two bottom-up tree transducers, \( B_1 \) and \( B_2 \), such that \( T_{B_2 \circ B_1} = T_T \). Hence there also exist (simple) algebraic tree transducers \( \tau_1 \) and \( \tau_2 \) such that \( T_{\tau_2 \circ \tau_1} = T_T \). Furthermore, since infinite algebraic tree transducers are closed under composition, there exists an infinite algebraic tree transducer \( \tau \) such that \( T_T = T_{\tau_2 \circ \tau_1} = T_T \). Thus every
top-down tree transduction $\text{Tr}_T$ can be performed by an infinite algebraic
tree transducer $\tau$. However, as (IV.10) shows, $\tau$ may not be simple or
even finite.

(IV.10) Example. Let $T$ be the top-down tree transducer given in
(II.8), and let $\tau$ be the infinite algebraic tree transducer such that
$\text{Tr}_\tau = \text{Tr}_T$. We recall that no single bottom-up tree transducer can induce
$\text{Tr}_T$. Thus $\tau$ cannot be a simple algebraic tree transducer. Similarly,
$\tau$ cannot be finite, even if not simple, because a finite $\tau$ would at best
transform input trees of the form $\sigma(t)$ to output trees of the form
$\sigma(t_1 t_2)$ where $t_1$ and $t_2$ would be identical after some point. Thus no
finite $\tau$ can generate the full set of output trees generated by $T$ for
every input. Hence $\tau$ must be infinite.

We see from (IV.10) that algebraic tree transducers cannot induce
all the transductions induced by top-down tree transducers. In
particular, they cannot induce the transductions which are induced by
nonlinear top-down tree transducers. (Engelfriet [16] has shown that
every linear top-down tree transducer $LT$ has a bottom-up counterpart
which induces the same transduction. Thus by (IV.4) there exists a
(simple) algebraic tree transducer $\tau$ such that $\text{Tr}_\tau = \text{Tr}_{LT}$.) Furthermore,
we recall that top-down tree transducers cannot induce all the
transductions which can be induced by bottom-up tree transducers, and
hence by simple algebraic tree transducers. Thus top-down tree
transducers and algebraic tree transducers are in fact incomparable.

Since bottom-up and top-down tree transducers are incomparable, and
since bottom-up tree transducers are a restriction of product transducers,
we see that there are transductions induced by product transducers which cannot be induced by top-down tree transducers. Conversely, product transducers cannot induce the transductions induced by nondeterministic top-down tree transducers. They lack the ability possessed by top-down tree transducers to create copies of unprocessed subtrees of the input tree and then process them differently. Thus product transducers and top-down tree transducers are incomparable. However, product transducers are capable of modeling effectively deterministic top-down tree transducers, as shown by (IV.11). In this way, they are similar to the algebraic model of effectively deterministic top-down tree transducers given by Goguen, Thatcher, Wagner, and Wright [20]. However, they are more powerful than the transducers of Goguen et al. since they can obviously perform all bottom-up tree transductions.

(IV.11) Example. Let us consider the well-known deterministic top-down finite tree transducer which takes derivatives [21]. Let $D = (E,E,\{i,d\},\{d\},R)$ where $E_0 = \{0,1,a\}$, $E_2 = \{+,\times\}$, and $R$ is given in part (1) of Figure 14. We know that $Tr_D$ cannot be implemented by any bottom-up tree transducer. There is a product transducer $D' = (Z,Z,\{i,d\},\{d\},R')$ such that $Tr_{D'} = Tr_D$. The rules of $D'$ are given in part (2) of Figure 14.

The transducer $D'$ in (IV.11) serves as an example of a product transducer which cannot be modeled by an algebraic tree transducer because the output generated upon reading a $\times$ involves both the identity state (i) and derivative state (d) outputs of the arguments of the multiplication. No algebraic tree transducer can provide all the
Figure 14. Rules for a deterministic top-down tree transducer and its corresponding product transducer

required data in a single output tree. Hence, (IV.11) confirms that algebraic tree transducers are strictly weaker than product transducers.

In summary, we have seen that bottom-up tree transducers (B) are equivalent to simple algebraic tree transducers (SA) and strictly weaker than algebraic tree transducers (A). Algebraic tree transducers are strictly weaker than product transducers (P). Effectively deterministic top-down tree transducers (EDT) are strictly weaker than both product
transducers and nondeterministic top-down tree transducers (NT).
Product transducers and nondeterministic top-down tree transducers are
incomparable and are both strictly weaker than infinite algebraic tree
transducers (IA). These results are summarized in Figure 15.

Figure 15. The relationship of classes of tree transducers
V. CONSTRUCTION OF AN ALGEBRAIC TREE TRANSDUCER

We now turn our attention to the matter of constructing algebraic tree transducers to perform semantic-preserving translations. We will assume that we are dealing with source and target languages whose underlying grammars have known algebraic structures. (We saw in Chapter III that every context-free grammar determines an algebraic type. Some other, more complex grammars, e.g., Fischer's IO grammars [22], can be shown to determine algebraic types in a similar fashion.)

One of our main concerns will be how to determine a suitable common representable type for a transducer. The other will be how to pick appropriate sets of derived operations to generate the source and target algebras.

There are several properties we would like our transducers to possess. Suppose \( \tau = h_2 \circ h^{-1}_1 \subseteq W_1' \times W_2' \) is the algebraic tree transducer induced by the representations of \( r': P' \rightarrow N'^* \times N'^* \) in \( r_i: P_i \rightarrow N_i^* \times N_i^* \) for \( i=1,2 \). In addition, suppose \( \eta_i: N' \rightarrow N_i \) and \( \pi_i: P' \rightarrow P_i \) for \( i=1,2 \) are the functions which determine the representation. We would like to guarantee that \( W_1' = W_1 \) for every \( \eta_i \circ (N') \). If this is so, we will say that \( W_1' \) and \( \tau \) are full. In addition, especially for the purposes of ease in implementation and the minimization of ambiguity, we would like to guarantee that for every \( t \) in \( W_1' \), the domain of \( \tau \), there exist a unique \( \eta_i \in N' \) and a unique \( w \in W_1' \) such that \( h_1(w) = t \). We will say that such a \( W_1' \) is uniquely decomposable. We would also like to guarantee that our common representable type \( r' \) induces a semantic-preserving translation.
Let \( r: P \to N^* \times N^* \) be a finite type. Let \( W \) be the word \( r \)-algebra, and let \( U \) be any \( r \)-algebra. Let \( h: W \to U \) be the unique homomorphism from \( W \) onto \( U \). Let \( \overline{r}: \overline{P} \to N^* \times N^* \) be the type derived from \( r \).

Suppose \( p \in P \), \( r(p) = (\lambda, n) \) for some \( n \in N \), and \( u \in U_n \). We say that \( p[X^\lambda] \) is a **parse** for \( u \) provided \( p_u = u \). If \( p[q_1 \ldots q_k] \in \overline{P} \), \( r(p) = (n, \ldots, n) \), and there exists \( u \in U_n \) and \( v_i \in U_n \) such that \( q_i \) is a parse for \( v_i \) for \( i = 1, \ldots, k \), and if \( u = p_u(v_1 \ldots v_k) \), then \( p[q_1 \ldots q_k] \) is a parse for \( u \). Thus if \( \pi \in \overline{P} \) is a parse for \( u \in U_n \) for some \( n \in N \), we have \( \overline{\pi} \in \overline{W_n} \) and \( h(\overline{\pi}) = u \). If for every \( u \in U_n \), there is a unique \( \pi \in \overline{P} \) such that \( \overline{\pi} \) is a parse for \( u \), then \( U \) is uniquely decomposable.

We commented in Chapter III that every element of a word algebra as well as every derived operation can be thought of as a tree. We present below some additional definitions concerning word algebras and trees.

If \( w \in W_n \) for some \( n \in N \), then \( w \) is an **\( n \)-tree**. Similarly, if \( N' \subseteq N \) and \( w \in U_{n \in N'} W_n \), then \( w \) is an **\( N' \)-tree**. If \( w = p_w(w_1 \ldots w_k) \) for some \( p \in P \) and \( w_1, \ldots, w_k \in U_{n \in N} W_n \), then \( v \) is a **subtree** of \( w \) if and only if \( v = w_i \) or \( v \) is a subtree of \( w_i \) for some \( 1 \leq i \leq k \). Thus for any \( n \in N \), if \( v \) is a subtree of \( w \) and \( v \in W_n \), then \( v \) is an **\( n \)-subtree** of \( w \). Also if \( N' \subseteq N \) and \( v \) is an \( n \)-subtree of \( w \) for some \( n \in N' \), then \( v \) is an **\( N' \)-subtree** of \( w \).

If \( v \) is an \( N' \)-subtree of \( w \) and \( v \) is not a subtree of any \( N' \)-subtree of \( w \), then \( v \) is a **maximal \( N' \)-subtree** of \( w \).

If \( p, q \in \overline{P} \) and the normal form representation of \( q \) is a \( (n-, N', \text{maximal } N') \) subtree of the normal form representation of \( p \), then \( q \) is a \( (n-, N', \text{maximal } N') \) **suboperation** of \( p \).
Theorem. Let \( r: \mathbb{P} \rightarrow \mathbb{N}^* \times \mathbb{N}^* \) be a finite type, and let \( W \) be the word \( r \)-algebra. Let \( \overline{W} \) be the derived word \( \overline{r} \)-algebra where \( r: \mathbb{P} \rightarrow \mathbb{N}^* \times \mathbb{N}^* \) is the type derived from \( r \). For every nonempty subset \( \mathbb{N}' \) of \( \mathbb{N} \), there exists a (not necessarily finite) type \( r': \mathbb{P}' \rightarrow \mathbb{N}'^* \times \mathbb{N}'^* \) such that \( r' \) is representable in \( \overline{r} \) and the represented algebra \( \overline{W}' \) is full and uniquely decomposable.

Proof. For all \( p \in \mathbb{P} \), let \( p_{NF} \) denote the normal form representation of \( p \). Let \( \mathbb{P}' = \{ p_{NF} \in \mathbb{P} \mid r(p) \in \mathbb{N}'^* \times \mathbb{N}'^*, p_{NF} \neq \lambda \}, \) and \( p_{NF} \) contains no \( \mathbb{N}' \)-suboperations other than projections on its frontier. Let \( r': \mathbb{P}' \rightarrow \mathbb{N}'^* \times \mathbb{N}'^* \) be given by \( r'(p) = \overline{r}(p) \) for all \( p \in \mathbb{P}' \). Then \( r' \) is clearly representable in \( \overline{r} \) via the identity functions on \( \mathbb{N}' \) and \( \mathbb{P}' \).

Let \( W' \) be the word \( r' \)-algebra, let \( \overline{W}' \) be the represented \( r' \)-algebra, and let \( h': W' \rightarrow \overline{W}' \) be the unique homomorphism. Clearly \( \overline{W}'_n \subseteq \overline{W}_n = W_n \) for all \( n \in \mathbb{N}' \). We wish to show that \( W_n \subseteq \overline{W}'_n \) for all \( n \in \mathbb{N}' \).

If \( w \in W_n \) for some \( n \in \mathbb{N}' \) and \( |w| = 1 \), then there exists \( p \in \overline{P} \) such that \( r(p) = (\lambda, n) \) and \( p_{W} = p = w \). But then \( p_{NF} \in \mathbb{P}' \), so that \( p_{NF} \in W_n \) and \( h(p_{NF}, W') = p_{NF}, W = w \in \overline{W}'_n \).

Suppose that if \( w \in \overline{W}_n \) for some \( n \in \mathbb{N}' \) and \( |w| < m \), then \( w \notin \overline{W}'_n \). If \( |w| = m \) and \( w \in W_n \) for some \( n \in \mathbb{N}' \), then there exist maximal \( \mathbb{N}' \)-subtrees \( w_i \in W_n \) of \( w \) for \( i = 1, \ldots, k \) and \( p_{NF} \in \overline{P} \) such that \( r'(p_{NF}) = (n_1, \ldots, n_k, n) \) and \( w = p_{NF}, \overline{w}(w_1 \ldots w_k) \). Since \( |w_i| < m \), we have \( w_i \in \overline{W}'_n \) and \( w_i = h'(w_i) \) for \( i = 1, \ldots, k \). Also, since \( p_{NF} \) is in normal form, \( p \in \mathbb{P}' \), and so \( p_{W}(w_1 \ldots w_k) \in W_n \) and \( h'(p_{W}(w_1 \ldots w_k)) = p_{W}(w_1 \ldots w_k) = w \in \overline{W}'_n \). Thus \( W_n = \overline{W}'_n \) for all \( n \in \mathbb{N}' \) so that \( \overline{W}' \) is full.

Since \( W \) is a word algebra, it is uniquely decomposable. Thus for all \( n \in \mathbb{N} \) and \( w \in W_n \), there is a unique \( \pi \in \overline{P} \) such that \( \pi \) is a parse for \( w \).
Since \( P' \subseteq P \), there is at most one \( \pi \in P' \) such that \( \pi \) is a parse for \( w \) whenever \( w \in W' \) and \( n \in N' \). Since every parse is in normal form and hence in \( P' \) whenever \( w \in W' \) and \( n \in N' \), there is exactly one \( \pi \in P' \) such that \( \pi \) is a parse for \( w \). Thus \( W' \) is also uniquely decomposable.

(V.2) Example. Let \( r: \{B,OE,VE,BOE,BVE,C,V,E,E'\} \rightarrow N^* \times N^* \times N^* \times N^* \) be the type of the grammar APL as shown in Figure 4. Let \( N' = \{\text{Exp}\} \). From (V.1) we get \( P' = \{B[C[X_0^1]], B[V[X_0^1]], B[E'[E[X_1^Exp]], OE[X_1^Exp], VE[X_1^Exp], BOE[C[X_0^1] X_1^Exp], BOE[V[X_0^1] X_1^Exp], BOE[E'[E[X_1^Exp Exp], Exp Exp], BVE[C[X_0^1] X_1^Exp], BVE[V[X_0^1] X_1^Exp], BVE[E'[E[X_1^Exp Exp], Exp Exp]]\} \). We note that these derived operations are equivalent to those given in Figure 6 and generate the domain of the transducer \( t \) given in (III.7).

For every nonempty subset \( N' \) of \( N \), we have found a set \( P' \subseteq P \) of derived operations which generates a full and uniquely decomposable algebra, i.e., a suitable domain for an algebraic tree transducer whenever \( P' \) is finite. We will now investigate how the choice of arbitrary sets \( P'' \) of derived operations whose domains and codomains are all restricted to \( N' \) influences the fullness and unique decomposability of the algebras they generate.

Let \( r: P \rightarrow N^* \times N^* \) be a finite type, and let \( \overline{r}: \overline{P} \rightarrow N^* \times N^* \) be the type derived from \( r \). Let \( W \) be the word \( r \)-algebra and \( \overline{W} \) be the derived word \( \overline{r} \)-algebra. Let \( \phi \neq N' \subseteq N \), and let \( P' \subseteq \overline{P} \) be the set obtained in (V.1). Let \( P'' \) be any nonempty subset of \( \overline{P} \) such that \( \overline{r}(p) \in N'^* \times N'^* \) for all \( p \in P'' \). Let \( r'' : P'' \rightarrow N'^* \times N'^* \) be given by \( r''(p) = \overline{r}(p) \) for all \( p \in P'' \). Clearly \( r'' \) is representable in \( \overline{r} \) via the identity functions on \( N' \) and \( P'' \). Let \( W'' \) be
the word $r^*$-algebra and $\overline{W}^*$ be the $r^*$-algebra represented in $\overline{W}$. Let $h': W' \to \overline{W}$ and $h'': W'' \to \overline{W}$ be the specified unique homomorphisms.

(V.3) **Theorem**. If $P' \neq P''$, then $\overline{W}''$ is full but not uniquely decomposable.

**Proof.** Since $\overline{W}'$ is full and generated by $P'$ and $\overline{W}''$ is generated by $P''$, a superset of $P'$, $\overline{W}''$ is clearly full.

Since $P''$ is a superset of $P'$, there exists $p \in P''$ such that $p \notin P'$. If $r''(p) = (\lambda, n)$ for some $n \in N'$, then $p \in \overline{W}''_n$ and $h''(p) = p \overline{W}''_n = \overline{W}_n = W_n$.

However since $w \in \overline{W}''_n$ and $p \notin P'$, there must exist $w' \in \overline{W}'_n \subseteq \overline{W}''_n$ such that $h'(w') = w$. Similarly, if $r'(p) = (n_1 \ldots n_k, n)$ for some $n_1 \ldots n_k \in N'^*$, then for any $w_i \in \overline{W}''_{n_i}$ for $i = 1, \ldots, k$, we have $w = p \overline{W}''_{n_1 \ldots n_k} \in \overline{W}''_n$ and $h''(w) = w \overline{W}''_n = \overline{W}_n$. Again there must exist $w' \in \overline{W}'_n$ such that $h'(w') = w$ where $w' = q \overline{W}'_{n_j}$ for some $j \geq 0$ such that $q \neq p$ since $p \notin P'$. Thus in either case, $\overline{W}''$ is not uniquely decomposable.

(V.4) **Theorem**. If $P'' \neq P'$, then $\overline{W}''$ is uniquely decomposable but not full.

**Proof.** Since $P'' \neq P'$ and $P''$ generates $\overline{W}''$, it must also be uniquely decomposable.

Since $P'' \neq P'$, there exists $p \in P'$ such that $p \notin P''$. Suppose $r'(p) = (\lambda, n)$ for some $n \in N'$. Then $p \in \overline{W}'_n$ and $h'(p) = p \overline{W}'_n = W_n$. By the unique decomposability of $\overline{W}'$, there cannot exist $w \in W''_n \neq W'_n$ such that $w \neq p$ and $h''(w) = p$. Suppose, on the other hand, that $r'(p) = (n_1 \ldots n_k, n)$ for some $n_1 \ldots n_k \in N'^*$ and $n \in N'$. Then if $w_i \in \overline{W}'_{n_i}$ for $i = 1, \ldots, k$, we have $w = p \overline{W}'_{n_1 \ldots n_k} \in \overline{W}'_n$ and $h'(w) = w \overline{W}'_n$. Again by the unique decomposability
of \( \overline{W'} \), there cannot exist \( w' \in \overline{W''} \subseteq \overline{W'} \) such that \( h''(w') = w \). Thus \( \overline{W''} \) is not full.

If \( P'' \) is not comparable to \( P' \), the fullness and unique decomposability of \( \overline{W''} \) must be determined on an individual basis. However, the theorem in (V.5) provides a test for determining which subsets \( P'' \) of \( \overline{P} \) may give rise to full and uniquely decomposable algebras.

(V.5) **Theorem.** If \( P'' \neq P' \) gives rise to a full and uniquely decomposable \( \overline{r''} \)-algebra, then \( P'' \) is incomparable to \( P' \) and every element of \( P'' \) is equivalent to an element of \( \overline{P'} \).

**Proof.** \( P'' \) is incomparable to \( P' \) as a direct result of (V.3) and (V.4). We note that if \( p \in P' \), then \( r(p) \notin \overline{N'} \) and \( p \neq q[q_1 \ldots q_k] \) for any \( p \neq q, q_1, \ldots, q_k \in P' \) and \( k > 0 \) since \( p \) contains no \( \overline{N'} \)-suboperations other than projections on its frontier. Furthermore, each \( p \in P' \) appears in the unique parse of some word \( a \) in \( \overline{W''} = \overline{W'} \). Thus every element of \( P' \) is either equivalent to an element of \( P'' \) or a suboperation of an element of \( P'' \).

Suppose there exists \( q \in P'' \) such that \( q \) is not an element of \( \overline{P'} \).

Then \( q_{NF} \) appears in the unique parse of some \( b \in \overline{W''} \) such that \( b \notin \overline{W'} \) for some \( n \in \overline{N'} \). This is impossible since \( \overline{W''} = \overline{W'} \). Hence every element of \( P'' \) is equivalent to an element of \( \overline{P'} \).

The theorem in (V.6) shows that the finiteness of \( P'' \) is dependent upon the finiteness of \( P' \).

(V.6) **Theorem.** If \( P' \) is infinite and \( P'' \) gives rise to a full and uniquely decomposable algebra, then \( P'' \) is also infinite.
Proof. Suppose $P''$ is finite. Let $P'''$ be the set of normal form representations of the maximal $N'$-suboperations of the operations on $P''$. Then by (V.1), the algebra generated by $P'''$ is full since the algebra generated by $P''$ is full and $P''' = (P'')'$ in the construction of the proof of (V.1). But clearly $P''' \subseteq P'$, so that $P'''$ cannot generate a full algebra by (V.4). Hence $P''$ must be infinite.

Thus, in some sense, the set $P'$ we described in the proof of (V.1) is the best subset of $\overline{P}$ which gives rise to a full and uniquely decomposable algebra. We will now discuss the conditions under which $P'$ is a finite set.

(V.7) Theorem. Let $r: P \rightarrow N^* \times N^*$ be a finite type. Let $\psi \neq N' \subseteq N$, and let $r': P' \rightarrow N' \times N'^*$ be the type constructed from $r$ in the proof of (V.1). Then $P'$ (and hence $r'$) is infinite if and only if for some $n \in N'$ there exists $\alpha \in N'^*$, $p \in P'$ such that $r'(p) = (\alpha, n)$, and an $m$-suboperation $p'$ of $p$ for some $m \in N'$ such that $p'$ has an $m$-suboperation $q$.

Proof. Let $p$, $p'$, and $q$ be as stated in the theorem. Let us define $q_0 = p'_{NF}$ and $q_{j+1} = (q_j)_{p'_{NF}}$ for all $j \geq 0$. Thus $\{q_j\}_{j<\omega}$ is an infinite subset of $\overline{P}$ such that $q_i \neq q_j$ for all $i \neq j$. Let $p_0 = p$ and $p_{j+1} = (p_j)^{q_{j+1}}_{p'}$ for each $j \geq 0$. Then $\{p_j\}_{j<\omega}$ is an infinite subset of $P'$.

On the other hand, assume that if $p \in P'$ and $r'(p) = (\alpha, n)$ for some $\alpha \in N'^*$ and $n \in N'$, then $p$ has no $m$-suboperation $p'$ having an $m$-suboperation $q$ for any $m \in N'$. Then $|p| \leq |N'| + 1$ for all $p \in P'$. Furthermore, $P$ is finite and hence the labels of the branching nodes of all $p \in P'$ come from a finite set. Also, each $p \in P'$ has either $X_0^\lambda$, $X_1^\alpha$, $\ldots$, $X_k^\alpha$ along its
frontier where $k = |a|$. Since $N'$ is finite and each $p \in P'$ is bounded in height, the set of projections which appear on the frontiers of elements of $P'$ is finite. Thus $P'$ must be finite, and the theorem is proved.

The following definitions will provide a convenient framework for determining which subsets $N'$ of $N$ give rise to a finite set $P'$ in the proof of (V.1).

Let $\overline{P}_N$ denote the set of trees formed from $\overline{P}$ by replacing the label $p \in P \cup \{x_1^a : a \in N^* \text{ and } 0 \leq i < |a|\}$ of each node of each tree by the second coordinate of $r(p)$. We can assume without loss of generality that for every $n \in N$ there exist $a \in N^*$ and $p \in P$ such that $r(p) = (a, n)$. Thus each $n \in N$ is the beginning of a path through an element of $\overline{P}_N$. Let $C^{(1)} = N$.

If $am \in N^1$ is a partial path of length $i+1$ through an element of $\overline{P}_N$, $a \in C^{(1)}$, and $m \neq \alpha_j$ for $1 \leq j < |a|$, then we will put $am$ in $C^{(i+1)}$. However, if $am$ is a partial path and $m = \alpha_j$ for some $1 \leq j < |a|$, then we will put $am$ in $C^{(i+1)}$. That is, $C^{(i+1)} = \{am : a \in C^{(i)} \text{ and there exist } p \in P \text{ and } \beta \in N^* \text{ such that } r(p) = (\beta, a) \text{ and } m = \beta_j \text{ for some } 1 \leq j < |\beta| \text{ but } m \neq \alpha_k \text{ for } 1 \leq k < |a|\}$ and $C^{(i+1)} = \{am : a \in C^{(i)} \text{ and there exist } p \in P \text{ and } \beta \in N^* \text{ such that } r(p) = (\beta, n), m = \beta_j \text{ for some } 1 \leq j < |\beta|, \text{ and } m = \alpha_k \text{ for some } 1 \leq k < |a|\}$.

Then $C^{(i)}$ is finite since $C^{(1)} \subseteq N^i$ for all $i \geq 1$, and $N$ is finite. Furthermore $C^{(i)} = \emptyset$ whenever $i > |N|$. $C^{(i+1)}$ is also finite since $C^{(i+1)} \subseteq \{am : a \in C^{(i)} \text{ and } m \in N\}$ for all $i \geq 1$ since $C^{(i)}$ and $N$ are both finite, and $C^{(i+1)} = \emptyset$ whenever $i > |N|$. Let $C = \bigcup_{1 \leq i < \omega} C^{(i)}$. Then $C$ is also finite. $C$ is the set of all nontrivial partial paths through elements of $\overline{P}_N$ which terminate at the first repetition. We note that although $C$ is related to the infinite set $\overline{P}_N$, we can easily compute $C$.
using the formal definitions we have given. In (V.8) we will define a function on subsets of $\mathbb{N}$ which uses $C$ to determine which subsets give rise to finite sets $P'$ is the proof of (V.1).

(V.8) **Theorem.** Let $r: \mathbb{P} \to \mathbb{N} \times \mathbb{N}$ be a finite type, and let $C = \{c_1, \ldots, c_k\}$ be the finite subset of $\mathbb{N} \times \mathbb{N}$ described above. Let $f: P(N) \times \phi \to \{0,1\}^k$ be given by $f(M) = b_1 \ldots b_k$ where for $i = 1, \ldots, k$ we determine $b_i$ as follows. Let $b_i = 1$ if $c_i \subseteq M$ and there exist $n$ in $c_i$ and $\alpha \in C$ such that $n \notin M$ and $\alpha$ contains no element of $M$. Let $b_i = 0$ otherwise. Let $N' = \{M: f(M) = 0^k\}$. Then $N'$ gives rise to a finite set $P'$ in the proof of (V.1) if and only if $N' \in V'$.

**Proof.** Suppose $\phi \neq N' \subseteq N$ gives rise to an infinite $P'$. Then for some $n \in N'$ there exist $\alpha \in N'^{\times}$ and $p \in P'$ such that $r'(p) = (\alpha, n)$ and an $m$-suboperation $p'$ of $p$ for some $m \notin N'$ such that $p'$ has an $m$-suboperation $q$. Since $p \in P' \subseteq \overline{P}$, the corresponding element $p_N \in \overline{P}_N$ contains a partial path which begins with $n$ and contains two $m$'s. Furthermore we can find such a $p$ so that this path is in $C$, i.e., contains no other repetitions. In addition, there is a partial path through $q \in N$ beginning and ending with $m$ and containing no elements of $N'$ since $q$ is a suboperation of $p \in P'$. Thus $f(N')$ contains at least one 1, corresponding to the above-mentioned partial path through $p$, so that $N' \in V'$. Hence if $N' \in V'$, the corresponding $P'$ is finite.

If $N' \notin V'$, we wish to show that $N'$ gives rise to an infinite set $P'$. If $N' \notin V'$, then $f(N')$ contains at least one 1. Thus $C$ contains paths $c$ and $c'$ such that $c$ begins with some $n \in N'$ and contains some $m \notin N'$ and $c'$ begins with $m$ and contains no elements of $N'$. Hence there must exist $\alpha \in N^{\times}$
and \( p \in P \) such that \( r(p) = (\alpha, \gamma) \) and an \( m \)-suboperation \( p' \) of \( p \) which has no \( N' \)-suboperations but does have an \( m \)-suboperation \( q \). Furthermore, since \( P \) is the set of all operations derived from \( P \), such a \( p \) exists for which \( p' \) is not a suboperation of any \( N' \)-suboperation of \( p \). Thus, by (V.7), \( P' \) is infinite. Consequently if \( P' \) is finite, we must have \( N' \supseteq N' \), and the theorem is proved.

(V.9) Example. As in (V.2), let \( r \) be the type of the grammar APL as given in Figure 4. Then \( C^{(1)} = N = \{ \text{Bas, Exp, Exp'} \} \) and \( C^{(2)} = \{ \text{Bas Exp'}, \text{Exp Bas, Exp' Exp} \} \) while \( C^{(2)} = \{ \text{Exp Exp} \} \). Next we find that \( C^{(3)} = \{ \text{Bas Exp' Exp, Exp Bas Exp', Exp' Exp Bas} \} \) and \( C^{(3)} = \{ \text{Exp' Exp Exp} \} \). Continuing, we find that \( C^{(4)} = \emptyset \) and hence \( C^{(1)} = \emptyset \) for all \( i > 4 \). Also, \( C^{(4)} = \{ \text{Bas Exp' Exp Bas, Exp Bas Exp', Exp' Exp Bas Exp', Bas Exp' Exp Exp} \} \) and \( C^{(1)} = \emptyset \) for all \( i > 4 \). Hence \( C = \bigcup_{1 < i < \omega} C^{(i)} = \{ \text{Exp Exp, Exp' Exp Exp, Bas Exp' Exp Bas, Exp Bas Exp' Exp, Exp' Exp Bas Exp', Bas Exp' Exp Exp} \} = \{ 1, 2, 3, 4, 5, 6 \} \). The function \( f \) from nonempty subsets of \( N \) to \( \{ 0, 1 \}^6 \) is shown in Figure 16. According to the figure, the sets \( \{ \text{Exp} \}, \{ \text{Bas, Exp} \}, \{ \text{Exp, Exp'} \} \), and \( \{ \text{Bas, Exp, Exp'} \} \) all give rise to finite sets \( P' \) in the proof of (V.1).

We recall that we can construct an algebraic tree transducer \( \tau \) whenever we have language definition systems \( D_1 = (G_1, A_1) \) for the source language and \( D_2 = (G_2, A_2) \) for the target language such that the types \( r_1 \) and \( r_2 \) derived from the types \( r_1 \) of \( G_1 \) and \( r_2 \) of \( G_2 \) have a finite common representable type \( r' \). Furthermore, the transduction \( Tr_{\tau} \) induced by \( \tau \) will be semantic-preserving if for every \( (t, t') \in Tr_{\tau} \), \( t \) and \( t' \) have some meaning in common.
One step toward determining whether two types have a common representable type is to determine whether or not a given type $r'$ is representable in another finite type $r$. We recall that $r': P' \rightarrow N'^\ast \times N'^\ast$ is representable in $r: P \rightarrow N^\ast \times N^\ast$ if and only if there exist functions $\eta: N' \rightarrow N$ (extended in the usual way to $\eta: N'^\ast \rightarrow N^\ast$) and $\pi: P' \rightarrow P$ such that for all $p \in P'$ with $r'(p) = (\alpha, \beta)$, we have $r(\pi(p)) = (\eta(\alpha), \eta(\beta))$.

The algorithm in (V.10) computes all possible representations of $r'$ in $r$.

(V.10) Algorithm. Determine $R$, the set of all representations of $r': P' \rightarrow N'^\ast \times N'^\ast$ in $r: P \rightarrow N^\ast \times N^\ast$ where $P$ and $P'$ are both finite.

<table>
<thead>
<tr>
<th>$N'$</th>
<th>$f(N')$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(1 2 3 4 5 6)</td>
</tr>
<tr>
<td>{Bas}</td>
<td>0 0 1 1 0 0</td>
</tr>
<tr>
<td>{Exp}</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>{Exp'}</td>
<td>0 1 0 0 0 1</td>
</tr>
<tr>
<td>{Bas, Exp}</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>{Bas, Exp'}</td>
<td>0 1 1 1 0 1</td>
</tr>
<tr>
<td>{Exp, Exp'}</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>{Bas, Exp, Exp'}</td>
<td>0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Figure 16. Determination of which subsets of $N$ give rise to a finite $P'$
\[ R = \phi. \]

Repeat for each \( \eta \in N' \):

Repeat for each \( \pi \in P' \):

\[ \text{Rep} = \text{true.} \]

Repeat for each \( p \in P' \):

If \( r'(p) = (\alpha, \beta) \) and \( r(\pi(p)) \neq (\eta(\alpha), \eta(\beta)) \), then \( \text{Rep} = \text{false.} \)

If \( \text{Rep} \), then \( R = R \cup \{ (\pi, \eta) \} \).

Return \( R \).

(V.11) **Theorem.** The algorithm in (V.10) computes the set of all representations of \( r' \) in \( r \).

At first glance the restriction that \( r \) be a finite type appears to make the algorithm in (V.10) useless in the construction of an algebraic tree transducer since we usually want to find a representation in the infinite type \( \overline{r}_1 \) derived from the finite type \( r_1 : P_1 \rightarrow N_1^{\ast \times N_1^{\ast}} \). However, instead of considering all of \( \overline{P}_1 \), we may focus our attention on \( \overline{P}_1' \), as described in (V.1), or on any other finite subset of \( \overline{P}_1 \) which generates a full and uniquely decomposable algebra. This approach is illustrated in (V.12).

(V.12) **Example.** Let \( r_1 \) and \( r' \) be the types of the context-free grammars given in Figure 17, and let \( \overline{r}_1 \) be the type derived from \( r_1 \). Let \( W_1 \) be the word \( r_1 \)-algebra, and suppose we want to construct an algebraic tree transducer whose domain is \( W_1, E \cup W_1, T \). According to (V.1), the set \( P' = \{ 1[X_1^{TE} X_2^{ET}], 2[X_1^{TE}], 3[X_1^{TE} 5[X_2^{TE}]], 3[X_1^{TE} 6], 4[5[X_1^{TE}]], \)
\[ G = (\{E,T,F\}, \{+,\times,a,(,)\}, P, \{E\}) \text{ where } P \text{ contains} \]

<table>
<thead>
<tr>
<th>Production p</th>
<th>Name of p</th>
<th>( r_1(p) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E \rightarrow E+T )</td>
<td>1</td>
<td>((E,T,E))</td>
</tr>
<tr>
<td>( E \rightarrow T )</td>
<td>2</td>
<td>((T,E))</td>
</tr>
<tr>
<td>( T \rightarrow TxF )</td>
<td>3</td>
<td>((TF,T))</td>
</tr>
<tr>
<td>( T \rightarrow F )</td>
<td>4</td>
<td>((F,T))</td>
</tr>
<tr>
<td>( F \rightarrow (E) )</td>
<td>5</td>
<td>((E,F))</td>
</tr>
<tr>
<td>( F \rightarrow a )</td>
<td>6</td>
<td>((\lambda,F))</td>
</tr>
</tbody>
</table>

\[ G' = (\{A\}, \{+,\times,a\}, P', \{A\}) \text{ where } P' \text{ contains} \]

<table>
<thead>
<tr>
<th>Production p</th>
<th>Name of p</th>
<th>( r'(p) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A \rightarrow +AA )</td>
<td>1'</td>
<td>((AA,A))</td>
</tr>
<tr>
<td>( A \rightarrow \times AA )</td>
<td>2'</td>
<td>((AA,A))</td>
</tr>
<tr>
<td>( A \rightarrow a )</td>
<td>3'</td>
<td>((\lambda,A))</td>
</tr>
</tbody>
</table>

Figure 17. Types used to illustrate finding all representations of \( r' \) in \( r \)
4[6]) obtained by letting $N_1' = \{E, T\}$ generates a full and uniquely decomposable algebra.

We can use (V.10) to compute all representations of $r|p_1'$ in $r'$. Applying the algorithm, we find that $R$ consists of a single $(\pi, \eta)$ pair, namely $\pi = \{(1', 3[X_1^{TE} 5[X_2^{TE}])], (2', 3[X_1^{TE} 5[X_2^{TE}]), (3', 4[6])\}$ and $\eta = \{(A, T)\}$.

The task of constructing an algebraic tree transducer to transform one language to another is not as straightforward as we might like. As we have seen, algebraic tree transducers are built by taking into consideration the underlying language definition systems (LDS's) of the source and target languages. It is well-known that many different grammars can generate the same language. One grammar may be better suited to the construction of a full semantic-preserving transducer with a uniquely decomposable domain than another. For example, we may wish to translate from a language which has both character-string and integer-valued identifiers to a language which has only integer values. If both kinds of identifiers are in the same underlying set of the source algebra, we might attempt to find another LDS in which these two types of identifiers are in separate underlying sets so that we might exclude the character-string identifiers from the domain of the transducer and still produce a full transducer. Similarly, languages may have several semantic algebras of the type of each underlying grammar. Our ability to construct an appropriate transducer is thus dependent on our initial choices of LDS's.
Unfortunately, even after the LDS's are specified, other heuristic decisions remain. Once we have decided which underlying sets of the source algebra we wish to translate, we must determine whether or not a finite set of derived operations will allow us to generate these sets completely. If not, we must either expand or reduce the domain of the transducer so that it may be finitely generated. In addition, we have seen that it is possible to find more than one set of derived operations which generate a full and uniquely decomposable source algebra. Some choices may allow us to construct the transducer we seek while others may not.

The procedure in (V.13) gives a method for constructing a full semantic-preserving algebraic tree transducer $\tau$ whose domain is uniquely decomposable assuming that the underlying LDS's have been specified, that the underlying sets of the source algebra which constitute the domain and codomain of the transducer are finitely generated, that sets $Q_1$ and $Q_2$ of derived operations have been specified, and that $Q_1$ generates a full and uniquely decomposable algebra. The procedure begins by trying to construct a common representable type $\tau': P' \to N'*\times N'*$ using the smallest possible $N'$ and increases $N'$ by one element at a time until a type is found which induces a full semantic-preserving transduction. Unfortunately, there is no way to determine an upper bound on the size of $N'$, and our choices of $Q_1$ and $Q_2$ may be inappropriate, so that we cannot guarantee termination of the procedure.

(V.13) Procedure. Let $D_i = (G_i, A_i)$ be an LDS where $G_i$ has type $r_i: P_i \to N_i*\times N_i*$, word $r_i$-algebra $W_i$, and derived type $\overline{r_i}$ for $i=1,2$. 
Let \( \mu_i : W_i \to A_i \) be the semantic homomorphism for \( i = 1,2 \). Construct a full semantic-preserving algebraic tree transducer \( \tau \subseteq U_{i \in M_1} W_i \times U_{j \in M_2} W_j \) where \( M_k \subseteq N_k \) for \( k = 1,2 \) and \( U_{i \in M_1} W_i \) is finitely generated by \( Q_1 \subseteq P_1 \).

1. \( k = |M_1| \).
   \[ m = \max\{|\alpha| : f_1(p) = (\alpha, n) \text{ for some } p \in Q_1 \text{ and } n \in M_1\} \]
   
   Found = false.

2. Repeat until Found:
   \[
   \begin{align*}
   N' &= \{n_1, \ldots, n_k\}. \\
   H &= \{n \in M_1 : n \text{ is onto } M_1\}. \\
   \text{Repeat for each } n \in H \text{ or until Found:} \\
   P'_n &= \emptyset. \\
   \text{Repeat for each } (\alpha, \beta) \in (U_{i=0}^m (N')^i) \times N': \\
   \text{Repeat for each } p \in Q_1: \\
   \text{If } f_1(p) = (n(\alpha), n(\beta)), \text{ then } P'_n = P'_n U\{p\} \text{ and } f'_n(p) = (\alpha, \beta). \\
   \text{(Note: } f'_n : P'_n \to N' \times N' \text{ is representable in } f_1 \text{ via } n \text{ and the identity function on } P'_n \subseteq Q_1 \subseteq P_1.\text{)}
   \]
   If \( P'_n = Q_1 \), then do:
   
   Find \( R \), the set of all representations of \( r' \) in \( f_2 |_{Q_2} \) using \( (V.10) \).
   
   Repeat for each \( r \in R \) or until Found:
   
   Consult an oracle to determine whether the induced \( \tau \) is operation preserving. (Note: undecidable problem \([14]\).)
   
   If \( \tau \) is operation preserving, Found = true.
   
   \( k = k + 1 \). (Note: Add another underlying set and try again.)

(\( V.14 \) Theorem. The procedure in \( (V.13) \) constructs a full operation preserving (and hence semantic-preserving) algebraic tree
transducer which translates $D_1$ into $D_2$ using $Q_1$ and $Q_2$ as the images of the operations of the common representable types provided such a transducer exists.

**Proof.** Let us assume that the desired algebraic tree transducer $\tau$ does exist. Let us suppose that no transducer which induces the desired translation has fewer than $j$ underlying sets in its common representable type, but that some transducer for this translation has precisely $j$ underlying sets.

Step 1 of the procedure is executed only once and always terminates. All of the sets computed in step 2 are finite since $N'$, $M_1$, $M_2$, $Q_1$, and $Q_2$ are all finite. Hence each of the inner loops of step 2 terminates. Only the outermost loop is potentially infinite.

If we find a common representable type for $\bar{r}_1|Q_1$ and $\bar{r}_2|Q_2$ which induces a full transducer and has fewer than $j$ underlying sets, then the transducer must fail to be operation preserving. In this case, or if no such type is found, the procedure attempts to construct a type having $j$ underlying sets.

Let $r: P \rightarrow N^* \times N^*$ be an actual common representable type which leads to a desired $\tau$ and let $n_i: N \rightarrow M_i$ and $\pi_i: P \rightarrow Q_i$ be the functions which give the representation of $r$ in $r_i|Q_i$ for $i=1,2$. Since $\tau$ is full, $n_i$ is onto $M_i$ and hence when $k=j$, we find $n_i \in H$ since $N$ is isomorphic to $N' = \{n_1, \ldots, n_j\}$. Hence when we come to $n_1$ in the loop, "Repeat for each $n_i \in H$ or until Found;" we construct $P' = Q_1$ and $r' = \pi_1 \circ \eta_1$. Since $r'$ is representable in $r_1|Q_1$ via $\eta_1$ and the identity function on $P'$, $\eta_1 = Q_1$, we must have $P$ isomorphic to $P'$. Thus the procedure has, in effect, constructed $r$. 

Since \( r \) is representable in \( \mathbb{Q}_2 \), one of the pairs in \( R \) will constitute the particular representation we seek. When the induced \( \tau \) is submitted to the oracle, it confirms that \( \tau \) is operation preserving. Hence Found will be set to true, and the procedure terminates having determined a suitable translation.

We have presented a number of guidelines to aid in the construction of an algebraic tree transducer. We have seen how to choose a set of derived operations which generate a full and uniquely decomposable domain for an algebraic tree transducer. In addition, we have seen how to decide what underlying sets of the source language's algebra must constitute the domain of the transducer in order for the domain to be finitely generated. Finally, we have seen how to find a common representable type which then induces an algebraic tree transducer once derived operations and underlying sets for both the source and target languages have been chosen. An oracle can determine whether or not this transducer is semantic-preserving. In practice, we would substitute judicious checking for the oracle to minimize the probability of accepting a transducer which is not semantic-preserving.
VI. CONCLUSIONS

We have examined two approaches to tree transduction. The automata-theoretic approach reported by Thatcher [10,11], Engelfriet [16], and others can transform syntax trees of one language into syntax trees of another language. Both bottom-up and top-down tree transformations are defined and have distinct capabilities. This automata-theoretic approach, although easy to implement, does not incorporate a sufficiently sophisticated consideration of semantics to make it useful for language translation.

The algebraic approach developed by Krishnaswamy and Strawn [13] can also perform transformations on syntax trees. It includes consideration of the source and target languages' semantics, and so is useful for developing semantic-preserving translations. An algebraic tree transducer is induced by the representations of some word algebra in algebras which generate the sets of input and output trees. Each representation defines a unique homomorphism from the common representable algebra to each of two algebras represented in the source and target algebras. Homomorphisms are applied from the top down, but are evaluated from the bottom up. We have seen that algebraic tree transducers can induce all the translations induced by bottom-up tree transducers and more, but are unable to model nonlinear top-down tree transducers.

Algebraic tree transducers produce their outputs based on derived operations often consisting of more than a single node of the input tree. In order to allow implementation in a more conventional framework, we
have also introduced a new automata-theoretic tree transducer, the product transducer. The product transducer works in a bottom-up fashion. It can model an algebraic tree transducer by reading a derived operation one node at a time and collecting all of the outputs of its operands in a product. Then when it reads the root of the derived operation, it can produce the appropriate output for that operation and compose it properly with the outputs of its operands. Although the product transducer does not explicitly consider the source and target languages' semantics, if it is used to model a semantic-preserving algebraic tree transducer, the result will certainly be a semantic-preserving translation. The product transducer has the added benefit of being able to model all deterministic top-down tree transducers.

We have also examined the problem of constructing an algebraic tree transducer whose domain is full and uniquely decomposable. The fullness guarantees that all the desired trees are in the domain of the transducer. The unique decomposability guarantees that the algebraic tree transducer can be modeled by a product transducer which recognizes the derived operations deterministically, although it will, of course, translate them nondeterministically if the algebraic transducer does. We have seen under what conditions we can find finite sets of derived operations to act as the image in the source algebra of the common representable algebra's operations so that the resulting transducer will have a full and uniquely decomposable domain. We have also specified the composition of these sets of derived operations. In addition, we have also studied a process for constructing an algebraic tree transducer once the language definition systems and the sets of derived
operations which should appear in the translation table have been specified. The procedure we have presented will find a common representable type whose sorts and operations symbols are mapped to the specified sorts and derived operation symbols of the algebras obtained from the source and target languages whenever such a type exists. It then consults an oracle to determine whether or not the transduction induced by the selected representations is semantic-preserving.

We believe that in many ways, the benefits of using algebras to construct translators are similar to the benefits of using structured programming techniques. Hence we would like to conduct an experiment in which two equally qualified programming teams attempt the same language translation problem, one team using the algebraic structure of the languages to develop their translator and the other team ignoring the algebraic structure. We conjecture that the team using the algebraic techniques would be able to produce a correct translator in less time.

We would also like to determine whether more programming language features can be modeled algebraically so that a wider range of translations may be performed algebraically. This effort might also lead to generalizing our definition of the algebraic tree transducer if we find that more complicated algebras are needed to model the additional language features.

In addition, we would like to see more work, such as that done by Strawn [4], to produce grammars for existing languages whose syntactic rules were not originally specified according to grammars. Along with this, we wish to see more development of algebraic semantics. We would
also like to recommend that the syntax and semantics of new languages be defined algebraically.

We believe that the algebraic specification of programming language syntax and semantics will prove to be of significant benefit in automating language translation. We have seen how an algebraic tree transducer can be modeled by a product transducer, an automaton which should not be difficult to implement. In addition, even when language translations are implemented by other means, the algebraic specifications are useful because they provide a standard against which the translators may be judged.
VII. REFERENCES


VIII. ACKNOWLEDGMENTS

The author is grateful to the members of her committee for their patient guidance and assistance with her research. She is particularly grateful to George Strawn and Ramachandran Krishnaswamy who have spent many hours sharing their understanding of tree transducers, analyzing her ideas, and making suggestions which have improved the presentation of her results. She is also indebted to Mrs. Fred Roberts for typing an exceptionally difficult manuscript quickly and accurately, to her husband, Bill Kwinn, for drawing the figures, and to Mrs. LaDena Bishop and the Iowa State University Thesis Office staff for their helpful suggestions and for their careful checking of the manuscript.

The author also wishes to thank the many people whose contributions to her research efforts have been more indirect than those mentioned above. The Department of Computer Science, the Computation Center, and the Department of Mathematics at Iowa State University have all provided much-needed financial support. Numerous teachers and professors have provided her with background and skills which have been most helpful. Outstanding among these are Bernice VanSickle and Sven Hammar, who insisted that she develop good writing and research skills in high school, and the mathematics and chemistry faculties of Saint Mary's College. Several friends, notably the Hentzels, Kafuras, Lohrs, Stouts, and Tondras, have housed, fed, and transported her during trips to Ames to complete her research. Lastly, the Yuhas and Kwinn families, and Bill especially, have given encouragement, understanding, and love without measure.
IX. APPENDIX

We present below the details of several proofs which were omitted from the text.

(II.7) We will first establish that for all \( q \in Q \), if \( t \) is the syntax tree associated with some derivation sequence \( q, s_1', \ldots, s_k \) in \( G \), then \( \pi^*(t) \xrightarrow{A} q(\pi^*(t)) \).

Suppose \( t \) is the syntax tree associated with the derivation \( q \Rightarrow s \) for some \( s \in \Sigma \). Then the height of \( t \) is 1 (\( |t| = 1 \)), and \( t \) must be the single production \( q \rightarrow s \). Hence \( \pi^*(t) = \pi(t) = s \). Clearly \( s \xrightarrow{A} q(s) \) since we obtained the production \( q \rightarrow s \) of \( P \) from \( s \rightarrow q(s) \) in \( R \).

Suppose that for all \( t \in T^p \) such that \(|t| < k \), it happens that whenever \( t \) is the syntax tree associated with a derivation sequence beginning with \( q \) for some \( q \in Q \), we have \( \pi^*(t) \xrightarrow{A} q(\pi^*(t)) \).

Let \( |t| = k \) and suppose \( t \) is the syntax tree associated with some derivation sequence beginning with \( q \) for some \( q \in Q \). Then \( t = p(t_1' \ldots t_j) \) for some \( p \in P \) and \( t_1', \ldots, t_j \in T^p \) where \(|t_i'| < k \) and \( t_i' \) is the syntax tree associated with a derivation sequence beginning with some \( q_i \in Q \) for \( i = 1, \ldots, j \). Then \( p \) must be \( q \rightarrow sq_1' \ldots q_j' \) for some \( s \in \Sigma \). Furthermore, by our inductive hypothesis, \( \pi^*(t_i') \xrightarrow{A} q_i(\pi^*(t_i')) \) for \( i = 1, \ldots, j \). Since \( p \) is \( q \rightarrow sq_1' \ldots q_j' \), there must be a rule \( s(q_1(x_1) \ldots q_j(x_j)) \rightarrow q(s(x_1 \ldots x_j)) \) in \( R \). Hence \( \pi^*(t) = \pi^*(p(t_1' \ldots t_j)) = \pi(p(\pi^*(t_1') \ldots \pi^*(t_j))) = s(\pi^*(t_1') \ldots \pi^*(t_j)) \xrightarrow{A} s(q_1(\pi^*(t_1')) \ldots q_j(\pi^*(t_j))) \xrightarrow{A} q(s(\pi^*(t_1') \ldots \pi^*(t_j))) = q(\pi^*(t)). \) In short, \( \pi^*(t) \xrightarrow{A} q(\pi^*(t)) \). Thus, if \( t \) is a proper syntax tree of \( G \) (i.e., the tree associated with a derivation sequence beginning with \( q \) for some \( q \in Q_d \)), then \( \pi^*(t) \xrightarrow{A} q(\pi^*(t)) \), and hence \( A \)}
recognizes $t$.

We must still show that $\pi^*$ is onto $Z$. In order to do so, we will show that if $\tau \in T_\Sigma$ and $\tau \xrightarrow{A^*} q(\tau)$, then there exists a derivation sequence in $G$ beginning with $q$ whose associated syntax tree is mapped to $\tau$ under $\pi^*$.

Suppose $\tau \in T_\Sigma$, $|\tau| = 1$, and $\tau \xrightarrow{A^*} q(\tau)$ for some $q \in Q$. Since $|\tau| = 1$, $\tau$ must be $\sigma$ for some $\sigma \in \Sigma$. Hence $\sigma \xrightarrow{A} q(\sigma)$, and so $\sigma \xrightarrow{A} q(\sigma) \in R$.

Consequently $\sigma \xrightarrow{A} q(\sigma) \in R$, and so $q, s$ is a derivation sequence of $G$. Its syntax tree is the tree $t$ whose only node is labeled $q \xrightarrow{A} s$, and $\pi^*(t) = \pi(q \xrightarrow{A} s) = \sigma = \tau$.

Suppose that whenever $\tau \in T_\Sigma$, $|\tau| < k$, and $\tau \xrightarrow{A^*} q(\tau)$, there exists a derivation sequence in $G$ beginning with $q$ and an associated syntax tree $t$ such that $\pi^*(t) = \tau$.

Suppose $\tau \in T_\Sigma$, $|\tau| = k$, and $\tau \xrightarrow{A^*} q(\tau)$. Then $\tau = \sigma(\tau_1 \ldots \tau_j)$ for some $\sigma \in \Sigma_k$ and $\tau_1, \ldots, \tau_j \in T_\Sigma$. Clearly $|\tau_i| < k$ for $i = 1, \ldots, j$, and also $\tau_i \xrightarrow{A} q_i(\tau_i)$ for some $q_i \in Q$. Hence, for each $i$ there exists a derivation sequence in $G$ beginning with $q_i$ and an associated syntax tree $t_i$ such that $\pi^*(t_i) = \tau_i$. Furthermore, $R$ must contain the rule $\sigma(q_1(x_1) \ldots q_j(x_j)) \xrightarrow{A} q(\sigma x_1 \ldots x_j)$, and hence $P$ contains the production $q \xrightarrow{A} \sigma q_1 \ldots q_j$. But then $(q \xrightarrow{A} \sigma q_1 \ldots q_j)(t_1 \ldots t_j)$ is a syntax tree for a derivation sequence in $G$ beginning with $q$, and $\pi^*((q \xrightarrow{A} \sigma q_1 \ldots q_j)(t_1 \ldots t_k)) = \sigma(\pi^*(t_1) \ldots \pi^*(t_j)) = \sigma(\tau_1 \ldots \tau_j) = \tau$. Consequently, $\pi^*$ maps the proper syntax trees of $G$ onto $Z$.

(IV.4) Given $B$, we have constructed $\tau$. In order to show that $Tr_\tau = Tr_B$, we will first show that whenever $w \in W^*$ for any $q \in Q$, then
If $|w| = 1$ and $w \in W_q'$, then $w = \rho$ for some $\rho \in \mathbb{R}$ of the form $s \rightarrow \rho(t)$ for some $\sigma \in \Sigma_0$, $\rho \in \mathbb{Q}$, and $t \in T_\Delta$. Hence $r'(\rho) = (\lambda, q)$. Since $w \in W_q'$, we have $h_1(w) = h_1(w) = h_1(q) = \rho W_1^{-1} = \pi_1(\rho) = s$. Furthermore $h_2(w) = h_2(q) = \rho W_2^{-1} = \pi_2(\rho) = t$. But since $\pi$ is $s \rightarrow q(t)$, we have $s \Rightarrow B q(t)$, and hence $h_1(w) \Rightarrow_B q(h_2(w))$.

Suppose $|w| = n$ and for all $u \in U_{\Sigma_0} W_q' q$ such that $|u| < n$, $h_1(u) \Rightarrow_B q(h_2(u))$ whenever $u \in W_q' q$. Since $|w| = n$, we have $w = \rho(w_1 \ldots w_k)$ for some $\rho \in \mathbb{R}$ and $w_i \in W_q'$ for $i = 1, \ldots, k$. Furthermore, $|w_i| < n$ for $i = 1, \ldots, k$, and so by our inductive hypothesis $h_1(w_i) \Rightarrow_B q(h_2(w_i))$. Also, since $w \in W_q'$ and $w_i \in W_q'$ for $i = 1, \ldots, k$, $\rho$ must be of the form $s(\sigma_1(x_1) \ldots \sigma_k(x_k)) \rightarrow q(t)$ for some $\sigma \in \Sigma$ and $t \in T_\Delta[X_k]$. But then $h_1(w) = h_1(\rho(w_1 \ldots w_k)) = \pi_1(\rho) \cdot h_1(w_1) \ldots h_1(w_k) = s(h_1(w_1) \ldots h_1(w_k)) \Rightarrow_B h_1(w_1) \ldots h_1(w_k) = \sigma_1(x_1) \ldots \sigma_k(x_k)$.

Thus $h_1(w) \Rightarrow_B q(h_2(w))$ where $\alpha = q_1 \ldots q_k$. But then $h_1(w) \Rightarrow_B q(h_2(w))$.

If $q \in Q_d$, then $(h_1(w), h_2(w))) \in Tr_\tau$ and also $(h_1(w), h_2(w))) \in Tr_B$, so that $Tr_\tau - Tr_B$.

Suppose $t \in T_\Sigma$, $t' \in T_\Delta$, and $t \Rightarrow_B q(t')$ for some $q \in Q$. We will show that there exists $w \in W_q'$ such that $h_1(w) = t$ and $h_2(w) = t'$.

If $|t| = 1$, then $t = s$ for some $s \in \Sigma_0$. Also, since $t \Rightarrow_B q(t')$, we have $s \Rightarrow_B q(t')$, and hence $s \rightarrow q(t')$ must be an element, say $\rho$, of $\mathbb{R}$. Consequently $r'(\rho) = (\lambda, q)$ and $\rho \in W_q'$. But then $h_1(\rho) = \pi_1(\rho) = s$ and $h_2(\rho) = \pi_2(\rho) = t'$.

Suppose $|t| = n$ and for all $u \in T_\Sigma$ such that $|u| < n$, if $u \Rightarrow_B q(u')$ for some $q \in Q$ and $u' \in T_\Delta$, then there exists $w \in W_q'$ such that $h_1(w) = u$ and...
\[ h_2(w) = u'. \] Since \(|t| = n\), we must have \(t = s(t_1...t_k)\) for some \(s \in \Sigma_k\) and \(t_1, ..., t_k \in \Sigma^*\) for some \(k > 0\). Furthermore, since \(t = s(t_1...t_k) \triangleright_B q(t')\), we must have \(t_1 \triangleright_B q_1(t'_1)\) for some \(q_1 \in Q\) and \(t'_1 \in T_{\Delta}\) for \(i = 1, ..., k\). But \(|t'_i| < n\) for \(i = 1, ..., k\), and hence there must exist \(w_i \in W_q^e\) such that \(h_1(w_i) = t_1\) and \(h_2(w_i) = t'_1\). Also, there must exist a rule \(\rho \in R\) such that \(\rho\) is \(s(q_1(x_1)...q_k(x_k)) \rightarrow q(v)\) for some \(v \in T_{\Delta}[X_k]\) such that \(t' = v_{x_1...x_k}\). But then \(r'(\rho) = (q_1...q_k,q)\) and so \(\rho(w_1...w_k) \in W_q^e\).

Hence letting \(\alpha = q_1...q_k\), we have \(h_1(\rho(w_1...w_k)) = h_2(\rho(w_1...w_k))\) = \(s(t_1...t_k) = t,\) and \(h_2(\rho(w_1...w_k)) = s(\rho(h_2(w_1)...h_2(w_k))) = x_{1}^{\alpha}...x_{k}^{\alpha}v_{x_1...x_k}(t'_1...t'_k) = t'.\) Thus whenever \(t \triangleright_B q(t')\) for some \(t \in T_{\Sigma}, q \in Q,\) and \(t'_i \in T_{\Delta}\), there exists \(w \in W_q^e\) such that \(h_1(w) = t\) and \(h_2(w) = t'.\) If \(q \in Q_d\), then \((t,t') \in T_{\tau}\) so that \((h_1(w),h_2(w)) = (t,t') \in T_{\tau}.\) Thus \(T_{\tau} = T_{\tau}.\) Consequently \(T_{\tau} = T_{\tau}\).

Given \(\tau\), we have constructed \(B\). In order to show that \(T_{\tau} = T_{\tau}\), we will first establish that \(T_{\tau} = T_{\tau}\) by showing that whenever \(w \in W_q^e_n\) for some \(n \in N_q^e\), it also happens that \(h_1(w) \triangleright_B n(h_2(w)).\)

If \(|w| = 1\) and \(w \in W_q^e_n\), then \(w = p\) for some \(p \in P\) such that \(r'(p) = (\lambda, n)\). By our construction of \(R, \pi_1(p) \rightarrow n(\pi_2(p))\) is in \(R\), and hence \(h_1(w) = \pi_1(p) \triangleright_B n(\pi_2(p)) = n(h_2(w)).\)

Suppose \(|w| = m, w \in W_q^e_n,\) and for all \(u \in U_n\) such that \(|u| < m,\) we have \(h_1(u) \triangleright_B n(h_2(u))\) whenever \(u \in W_q^e_n.\) Since \(|w| = m,\) we have \(w = p(w_1...w_k)\) for some \(w_i \in W_q^e\) for \(i = 1, ..., k\) and \(p \in P\) such that \(r'(p) = (n_1...n_k,n)\). Furthermore, \(|w_i| < m\) for \(i = 1, ..., k,\) and hence
\[ h_1(w_1) \xRightarrow{B} n_1(h_2(w_1)) \]. Also, since \( p \in P' \) and \( r'(p) = (n_1 \ldots n_k, n) = (\alpha, n) \), we know that \( R \) contains a rule \( \pi_1(p) \xRightarrow{\alpha} n_1(x_1) \ldots n_k(x_k) \). Hence \( h_1(w) = \pi_1(p)(h_1(w_1) \ldots h_1(w_k)) \xRightarrow{B} n_1(h_2(w_1) \ldots h_2(w_k)) \).

\[ n(\pi_2(p) x_1 \ldots x_k) = n(h_2(w)). \] Consequently, if \( w \in W' \) for some \( n \in \mathbb{N}' \), then \( h_1(w) \xRightarrow{B} n(h_2(w)) \). In addition, if \( w \in W' \) and \( n \in I \), then

\[ (h_1(w), h_2(w)) \in Tr_B \] and \( (h_1(w), h_2(w)) \in Tr_A \) so that \( Tr_B = Tr_A \).

We must still show that \( Tr_B = Tr_A \). We will do so by establishing that if \( t \xRightarrow{B} n(t') \) for some \( t \in T \), \( n \in \mathbb{N}' \), and \( t' \in T' \), then there exists \( w \in W' \) such that \( h_1(w) = t \) and \( h_2(w) = t' \).

If \( |t| = 1 \), then \( t = p \) for some \( p \in P' \). If \( t \xRightarrow{B} n(t') \), we must have \( p \xRightarrow{B} n(t') \), and hence \( p + n(t') \) must be in \( R \). Consequently, there must exist some \( q \in P' \) such that \( r'(q) = (\lambda, n) \), \( \pi_1(q) = p = t \), and \( \pi_2(q) = t' \). But \( q \in W' \), so that \( h_1(q) = \pi_1(q) = t \) and \( h_2(q) = \pi_2(q) = t' \).

Suppose \( |t| = m \) and for all \( u \in T_p \) such that \( |u| < m \), if \( u \xRightarrow{B} n(u') \) for some \( n \in \mathbb{N}' \) and \( u' \in T_p \), then there exists \( w \in W' \) such that \( h_1(v) = u \) and \( h_2(v) = u' \). Suppose \( |t| = m \) and \( t \xRightarrow{B} n(t') \) for some \( n \in \mathbb{N}' \) and \( t' \in T' \).

Then \( t = p(t_1 \ldots t_k) \) for some \( p \in P \) such that \( r_1(p) = (n_1 \ldots n_k, n) = (\alpha, n) \) for some \( n_1, \ldots, n_k \in \mathbb{N}' \) and \( t_i \in T_p \) for \( i = 1, \ldots, k \). Furthermore, since \( t = p(t_1 \ldots t_k) \xRightarrow{B} n(t') \), we must have \( t_i \xRightarrow{B} n(t'_i) \) for some \( t'_i \in T_p \) for \( i = 1, \ldots, k \). But \( |t_i| < m \), so that there exists \( w_i \in W' \) such that \( h_1(w_i) = t_i \) and \( h_2(w_i) = t'_i \) for \( i = 1, \ldots, k \). In addition, \( R \) must contain a rule \( p \xRightarrow{\alpha} n(u_1 \ldots u_k) \) for some \( u \in P' \) such that \( p \xRightarrow{\alpha} n(x_1 \ldots x_k) \).
\( t' = u_{x_k^\alpha}^{t_k' \ldots t_1'} \). Hence we must have \( q \in P' \) such that \( \pi_1(q) = p \) and \( \pi_2(q) = u \). But then \( h_1(q(w_1 \ldots w_k)) = \pi_1(q)(h_1(w_1) \ldots h_1(w_k)) = p(t_1 \ldots t_k) = t \), and \( h_2(q(w_1 \ldots w_k)) = \pi_2(q)(h_2(w_1) \ldots h_2(w_k)) = t' \).

Consequently, for all \( t \in T_{ek} \), there exists \( \nu \in W^\prime_{ek} \) such that \( h_1(\nu) = t \) and \( h_2(\nu) = t' \). Hence whenever \( \nu \in \mathcal{U} \), we have \( (t, t') \in \mathcal{T}_{ek} \) and \( (h_1(\nu), h_2(\nu)) = (t, t') \in \mathcal{T}_{ek} \), so that \( \mathcal{T}_{ek} - \mathcal{T}_{ek} \). Thus \( \mathcal{T}_{ek} = \mathcal{T}_{ek} \), and the theorem is proved.

(IV.9) Given \( t \) we have constructed \( P \). We wish to show that \( \mathcal{T}_{ek} - \mathcal{T}_{ek} \). In order to do so, we first prove that if \( t \not\preceq_{ek} n(t') \) for some \( \nu \in N' \), then there exists \( \nu \in W^\prime_{ek} \) such that \( h_1(\nu) = t \) and \( h_2(\nu) = t' \).

Suppose \( t \not\preceq_{ek} n(t') \) for some \( \nu \in N' \). Then either there exist \( \nu_1, \ldots, \nu_k \in T_{ek} \), \( \alpha \in T_{ek} \), and \( \nu_1, \ldots, \nu_k \in N' \) such that \( t = \alpha \nu_1 \ldots \nu_k \) and \( \nu_i \not\preceq_{ek} n(t') \) for \( i = 1, \ldots, k \), or not.

Suppose not. If \( |t| = 1 \), then \( t = s \) for some \( s \in P_{ek} \), and since \( t \not\preceq_{ek} n(t') \), we must have \( t = s \not\triangleright n(t') \) in \( R \). From the construction of \( R \), we know that there exists \( p \in P' \) such that \( r'(p) = (\lambda, n) \), and \( \pi_1(p) = s \) while \( \pi_2(p) = t' \). But then \( \nu \in W^\prime_{ek} \), and \( h_1(p) = s \) while \( h_2(p) = t' \), as desired. If \( |t| > 1 \), then since \( t \not\preceq_{ek} n(t') \) and \( t = s(t_1 \ldots t_k) \) for some \( s \in P_{ek} \) and \( t_1, \ldots, t_k \in T_{ek} \), we must have \( s(\nu_1(x_1) \ldots \nu_m(x_m)) = n(t') \) in \( R \) where \( m \) is the number of leaves of \( t \) and where for \( i = 1, \ldots, m \), \( \nu_i = t_j^- \) whenever the \( i^{th} \) leaf of \( t \) is in \( t_j \) for some \( 1 < j < k \). This requires that \( t = s(t_1 \ldots t_k) = \pi_1(p) \) for some \( p \in P' \) such that \( r'(p) = (\lambda, n) \).
Furthermore \( \pi_2(p) = t', \) so that again we have \( p \in W'_n, \) \( h_1(p) = t \) and \( h_2(p) = t', \) as needed.

Suppose, on the other hand, that \( t = a \) and that for \( i = 1, \ldots, k, \) \( t_i \) is a maximal subtree of \( t \) such that \( t_i \overset{p}{\rightarrow} n_i(t'_i). \) We know from the preceding argument that whenever \( t = 1 \) and \( t \overset{p}{\rightarrow} n(t'), \) there exists \( w \in W_n \) such that \( h_1(w) = t \) and \( h_2(w) = t'. \) Let us assume that whenever \( |t| < m \) and \( t \overset{p}{\rightarrow} n(t') \) for any \( n \in N', \) there exists \( w \in W_n \) such that \( h_1(w) = t \) and \( h_2(w) = t'. \) Suppose \( |t| = m \) and \( t \overset{p}{\rightarrow} n(t') \). We need only show that if \( t = a_{x_1 \ldots x_k} \) for some \( a \in T_{p_1}[X_k] \) and some maximal \( t_1, \ldots, t_k \in T_{p_1} \) such that \( t_i \overset{p}{\rightarrow} n_i(t'_i) \) for \( i = 1, \ldots, k, \) then there exists \( w \in W_n \) such that \( h_1(w) = t \) and \( h_2(w) = t'. \) Certainly \( |t_i| < m, \) so that there exists \( w \in W_n \) such that \( h_1(w_i) = t_i \) and \( h_2(w_i) = t'_i. \) Furthermore, it must be the case that \( t'_i \) is a subtree of \( t' \) for \( i = 1, \ldots, k. \) Also there must exist \( v \in T_{\Delta}[X_k] \) such that \( t' = v_{x_1 \ldots x_k} \) and \( a_{x_1 \ldots x_k}. \) If \( a = s(x_1 \ldots x_k) \) for some \( s \in P_1, \) then \( s(n_1(x_1) \ldots n_k(x_k)) \rightarrow p n(v) \) is in \( R, \) and hence there exists \( p \in P' \) such that \( r'(p) = (n_1 \ldots n_k, n), \)

\( \pi_1(p) = s_{x_1 \ldots x_k}, \) and \( \pi_2(p) = v_{x_1 \ldots x_k} \) where \( a = n_1 \ldots n_k. \) On the other hand, we may have \( a = s(a_1 \ldots a_j) \) for some \( s \in P_1 \) and \( a_1, \ldots, a_j \in T_{p_1}[X_k]. \) Then \( s(u_1(x_1) \ldots u_k(x_k)) \rightarrow n(v) \) is in \( R, \) where for \( i = 1, \ldots, k, \) \( u_i = n_i \) if \( a_i = \lambda \) and \( u_i = a_m \) if \( a_m \neq \lambda \) and \( x_i \) appears in \( a_m. \) Hence again there exists \( p \in P' \) such that \( r'(p) = (n_1 \ldots n_k, n), \) \( \pi_1(p) = a_{x_1 \ldots x_k}, \) and
\[ \pi_2(p) = \frac{X_1^\alpha \cdots X_k^\alpha}{x_1 \cdots x_k}. \]

In either case \( w = p(w_1 \cdots w_k) \in W_n \), \( h_1(w) = \frac{x_1^\alpha \cdots x_k^\alpha}{x_1 \cdots x_k} \).

\[ h_1(p(w_1 \cdots w_k)) = \pi_1(p)(h_1(w_1) \cdots h_1(w_k)) = \frac{x_1^\alpha \cdots x_k^\alpha}{x_1 \cdots x_k}(t_1 \cdots t_k) = t, \]

and

\[ h_2(w) = \pi_2(p)(h_2(w_1) \cdots h_2(w_k)) = \frac{x_1^\alpha \cdots x_k^\alpha}{x_1 \cdots x_k}(t_1' \cdots t_k') = t'. \] Thus if

\[ t \stackrel{p}{\Rightarrow} n(t') \]

for some \( n \in N' \), then there exists \( w \in W_n \) such that \( h_1(w) = t \) and \( h_2(w) = t' \). Furthermore if \( n \in I \), then \((t, t') \in Tr_p\), so that \( Tr_p \subseteq Tr_\tau \).

It remains for us to show that \( Tr_\tau = Tr_p \). We will do so by proving that if \( w \in W_n \), then \( h_1(w) \stackrel{p}{\Rightarrow} n(h_2(w)) \).

If \( |w| = 1 \), and \( w \in W_n \), then \( w = p \) for some \( p \in P \) such that \( r'(p) = (\lambda, n) \). By the construction of \( P \), we have \( \pi_1(p) \stackrel{p}{\Rightarrow} n(\pi_2(p)) \), and hence

\[ h_1(w) = \frac{\pi_1(p)}{p} \stackrel{p}{\Rightarrow} n(\pi_2(p)) = n(h_2(w)). \]

We can assume that if \(|w| < m \) and \( w \in W_n \), then

\[ h_1(w) \stackrel{p}{\Rightarrow} n(h_2(w)). \]

Suppose \( w \in W_n \) and \(|w| = m \). Then \( w = p(w_1 \cdots w_k) \) for some \( p \in P \) such that \( r'(p) = (n_1 \cdots n_k, n) = (\alpha, n) \) for some \( n_1, \ldots, n_k \in N' \).

But then \(|w_i| < m \) for \( i = 1, \ldots, k \), so that \( h_1(w_i) \stackrel{p}{\Rightarrow} n_1(h_2(w_i)) \) by our inductive hypothesis. Furthermore, since \( \pi \in P \), we know by the construction of \( R \) that \( \pi_1(p) \frac{\pi_1(n_1) \cdots \pi_k(n_k)}{x_1 \cdots x_k} \stackrel{p}{\Rightarrow} n(\pi_2(p)) \frac{x_1 \cdots x_k}{x_1 \cdots x_k} \).

\[ h_1(w) = \frac{\pi_1(p)(h_1(w_1) \cdots h_1(w_k))}{x_1 \cdots x_k} \stackrel{p}{\Rightarrow} \pi_1(p) \frac{n_1(h_2(w_1)) \cdots n_k(h_2(w_k))}{x_1 \cdots x_k} \]

\[ n(\pi_2(p) \frac{x_1 \cdots x_k}{x_1 \cdots x_k}) = n(h_2(w)). \] Thus whenever \( w \in W_n \), we have

\[ h_1(w) \stackrel{p}{\Rightarrow} n(h_2(w)). \]

Furthermore, if \( n \in I \) we have \((h_1(w), h_2(w)) \in Tr_\tau \) and \((h_1(w), h_2(w)) \in Tr_p \) so that \( Tr_\tau \subseteq Tr_p \), and in fact \( Tr_p = Tr_\tau \).