INFORMATION TO USERS

This material was produced from a microfilm copy of the original document. 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 original submitted.

The following explanation of techniques is provided to help you understand markings or patterns 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 thru an image and duplicating adjacent pages to insure you complete continuity.

2. When an image on the film is obliterated with a large round black mark, it is an indication that the photographer suspected that the copy may have moved during exposure and thus cause a blurred image. You will find a good image of the page in the adjacent frame.

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

4. The majority of users indicate that the textual content is of greatest value, however, a somewhat higher quality reproduction could be made from “photographs” if essential to the understanding of the dissertation. Silver prints of “photographs” may be ordered at additional charge by writing the Order Department, giving the catalog number, title, author and specific pages you wish reproduced.

5. PLEASE NOTE: Some pages may have indistinct print. Filmed as received.

Xerox University Microfilms
300 North Zeeb Road
Ann Arbor, Michigan 48106
PYSTER, Arthur Bruce, 1950-
FORMAL TRANSLATION OF PHRASE-STRUCTURE LANGUAGES.
The Ohio State University, Ph.D., 1975
Computer Science

© Copyright by
Arthur Bruce Pyster
1975

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED.
FORMAL TRANSLATION
OF
PHRASE-STRUCTURE LANGUAGES

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate
School of The Ohio State University

By

Arthur Bruce Pyster, B.S., M.S.

* * * * *

The Ohio State University

1975

Reading Committee

Prof. H.W. Buttelmann
Prof. D.J. Moore
Prof. L.J. White

Approved By

[Signature]
Adviser
Department of Computer
and Information Science
To my parents, and to
Rich Erlich, and
Louis Lipton
—without them,
I wouldn't be me
ACKNOWLEDGEMENTS

Bill Buttelmann has been far more to me than just an academic advisor. He has also been a good friend. I am grateful to Bill for all the time he has taken and the concern he has shown over the past three years in supervising this thesis.

After Bill, Kris has had the greatest influence on this research. I can't think of a better project-mate.

Finally, I have to thank the staff of the DECsystem-10 who frequently cajoled the PDP-10 into cooperating with the editing and printing of this thesis.

Research sponsored by The Air Force Office of Scientific Research, Air Force Systems Command, USAF, under Grant No. AFOSR-75-2811. The United States government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.
VITA

December 10, 1950

Born - Chicago, Illinois

1968-1971

General Motors Scholar, University of Illinois, Urbana, Illinois

August, 1971

B.S., Mathematics and Computer Science (with honors), University of Illinois, Urbana, Illinois

1972

Active Duty for Training, U.S. Army Reserves

1972-1975

University Fellow, Ohio State University, Columbus, Ohio

June, 1973

M.S., Computer and Information Science, Ohio State University, Columbus, Ohio

1973-1974

Teaching Associate, Ohio State University, Columbus, Ohio

1975

Research Associate, Ohio State University, Columbus, Ohio

FIELDS OF STUDY

Major Field: Computer and Information Science

Studies in Formal Languages and Automata Theory - Prof. H.W. Buttelmann

Studies in Systems Programming - Prof. D.K. Hsiao

Studies in Programming Languages - Prof. L.J. White
TABLE OF CONTENTS

ACKNOWLEDGEMENTS

VITA

LIST OF FIGURES

Chapter

I. INTRODUCTION

1.1 The "Translation Problem"

1.2 Natural Language Translation

1.3 Programming Language Translation

1.4 Syntax-Directed vs. Semantic-Directed Translation

1.5 History of Syntax-Directed Translation

1.7 Tree Transducers

1.7 Formal Models of Semantics

1.8 Models of Translation

1.9 Key Issues Addressed

II. BASIC TERMS AND DEFINITIONS

2.1 Abbreviations

2.2 Subscripts and Character Set

2.3 Functions and Computation

2.4 Trees

2.5 Grammars

2.6 Time Complexity

III. PHRASE-STRUCTURE SYSTEMS

3.1 Introduction

3.2 Basic Definitions

3.3 Ambiguity

3.4 Restrictions

3.5 Basic Properties

3.6 Table PSS

3.7 Complexity

3.8 Decidability Questions

3.9 Summary

iv
L. Semantic-Directed Translator: sdt-2 243
M. List of Theorems 246
INDEX 253
<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Syntax-Directed Translation</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Semantic-Directed Translation</td>
<td>4</td>
</tr>
<tr>
<td>1.3 XPL Structure</td>
<td>6</td>
</tr>
<tr>
<td>2.1 Drawing of 1&lt;</td>
<td>E&lt;</td>
</tr>
<tr>
<td>2.2 The Box Notation</td>
<td>23</td>
</tr>
<tr>
<td>3.1 Attributed Grammar</td>
<td>29</td>
</tr>
<tr>
<td>3.2 t € phr(UG(PAE))</td>
<td>37</td>
</tr>
<tr>
<td>3.3 Evaluated t € phr(UG(PAE))</td>
<td>38</td>
</tr>
<tr>
<td>3.4 Two Parses of The Same String - &quot;</td>
<td>I&quot;</td>
</tr>
<tr>
<td>4.1 Source Parse of 2!*</td>
<td>59</td>
</tr>
<tr>
<td>4.2 Target Parse of (2)*((1))</td>
<td>59</td>
</tr>
<tr>
<td>5.1 Transducer with Deviance Filter</td>
<td>83</td>
</tr>
<tr>
<td>5.2 Lengthening Chains in r and r'</td>
<td>86</td>
</tr>
<tr>
<td>6.1 Parse of &quot;2 2. 1 * +&quot;</td>
<td>109</td>
</tr>
<tr>
<td>8.1 Complete Phrase-Structure of AG</td>
<td>141</td>
</tr>
<tr>
<td>9.1 Runaway Recursion in W-TRANS</td>
<td>160</td>
</tr>
<tr>
<td>10.1 Relationships Between Translation Systems</td>
<td>179</td>
</tr>
</tbody>
</table>
Chapter I - Introduction

"What we have here is a failure to communicate."

Cool Hand Luke

The "Translation Problem"

The "Translation Problem" can be stated informally as:

Given two languages A and B, specify a relation $T:A \rightarrow B$ such that $v \in T(w)$ implies $w$ and $v$ share a common meaning.

This problem has been investigated extensively both by linguists concerned with natural language translation and by computer scientists interested in programming language translation. Some of the earlier workers in the area examined the Translation Problem within a formal framework. However, in most studies, ad hoc techniques were employed. In this thesis we investigate the Translation Problem in a formal mathematical framework. The thrust of our work is two-fold: (i) use the formally defined meanings of a sentence to partially direct the translator; and (ii) handle more complex syntactic relationships between source and target languages than previously done.

Natural Language Translation

The mechanical translation of natural languages was first proposed in a memorandum by Warren Weaver in 1949. During the next ten years, work in the area centered chiefly around mechanized dictionaries and word-for-word substitution schemes [EdmuH61]. Chomsky's work in formal syntax added a new dimension to the problem by giving linguists the mathematical tools to formally express the grammars of the source and target languages [56,57,59,65]. This intensified research into formal parsing methods for handling natural languages [GettA61]. While some progress was reported [BootA67], none of the early efforts at
mechanical translation were entirely satisfactory. Bar-Hillel [62] attributed this in part to the lack of a formal theory of semantics available at that time. He encouraged linguists to begin developing a satisfactory model of semantics before attempting any further to solve the Translation Problem. Recognizing that much basic research into semantics was necessary before any further attempts at building translators should be made, most linguists turned from translator construction to more theoretical issues in translation. Recently, Wilks [75] suggested that linguists should renew their efforts in natural language translation. Although far more remains to be discovered about semantics than is known, he feels that enough progress has been made to warrant another round of investigation into the Translation Problem.

Programming Language Translation

Attempts at programming language translation have been more successful than those in natural languages. Every compiler or assembler is in fact a translation system. There are many programs today which will convert from, for example, COBOL to FORTRAN or Honeywell FORTRAN to IBM FORTRAN [LedgH69, LapiR67]. However, there has only been a relatively small amount of theoretical investigation into formal semantics and formal properties of translation. As a result, translator writers have been forced into writing ad hoc programs to translate without the insights which formal investigations and formal tools could provide. More details about programming language translation are found in the historical sections of this chapter which follow.

Syntax-Directed vs. Semantic-Directed Translation.

It is convenient to think of a translator as a "black box" which accepts as input a string in the source language and outputs a set of "equivalent" strings in the target language, strings which have the same content as the source. There are two extreme views on the roles of syntax and semantics in the inner workings of the translator:

1) The syntax of both source and target languages are specified formally. The semantics of the source and target systems should be used in building the translator; i.e., in establishing relationships between the syntactic
INTRODUCTION
SYNTAX-DIRECTED VS. SEMANTIC-DIRECTED TRANSLATION

forms of the source and target languages. However, this is the only role of semantics in the translation process - to establish the relationship. Once the translator has been built, the semantics is no longer needed. The established syntactic relationships between source and target grammars can be used to drive the translator. Such a translation mechanism is normally called "syntax-directed". (see Figure 1.1)

2) The syntax and semantics of both source and target languages are expressed formally. The syntax of the source language is only a vehicle for expressing the intended meaning. Once this meaning has been computed, the original source text is no longer important to the translation process. The translation should be determined by the relationship between semantic descriptions of the source program and the target grammar. Thus, semantics plays the critical role at translation time. Such a translation mechanism we call "semantic-directed". (see Figure 1.2)

Figure 1.1. Syntax-Directed Translation
INTRODUCTION
SYNTAX-DIRECTED VS. SEMANTIC-DIRECTED TRANSLATION

The advantage of the first view is its simplicity. Semantic information for even relatively simple programming languages can become quite complex. Syntactic information, however, when represented as a parse tree, is very "clean" and is to a certain degree independent of the complexity of the semantics. (The syntactic description of APL is not really much more complex than that of FORTRAN.) Indeed, a vast amount of processing may be required to establish the desired syntactic relationships between source and target systems. But once built, a syntax-directed translator, as traditionally conceived, runs very quickly. Perhaps the greatest promise for syntax-directed translation is in translating between closely related languages such as CDC FORTRAN and IBM FORTRAN. It is more likely that the necessary syntactic correspondences between variants of the same language or between two similar languages can be found than between such structurally different languages as COBOL and LISP.

The advantage of the second view is its generality. The more disparate the source and target syntactic primitives and control structures, the more difficult it is to establish the necessary relationships between syntactic forms. Differences between source and target syntax are not important for semantic-directed translation. What is important are the "real" differences in semantic power, not how they are expressed in terms of the syntactic primitives.
INTRODUCTION
SYNTAX-DIRECTED VS. SEMANTIC-DIRECTED TRANSLATION

For example, the fact that PL/I has a character data-type and FORTRAN does not is much more important in terms of the tools necessary to translate between them than the fact that in PL/I the declaration statement would end with a semi-colon and in FORTRAN it would end with the 72nd column of a card. But generality, while being a strength is also a weakness. One of the greatest strengths of syntax-directed translation is that it takes advantage of similarities between language structures. Semantic-directed translation would necessarily treat translation between two very similar languages on the same basis as two quite different ones.

Translation mechanisms can either be syntax-directed, semantic-directed or a combination of both. In this thesis, we define and study translators which will fall into all three of these categories.

History of Syntax-Directed Translation

The concept of syntax-directed translation stems from early activity in compiler design. In 1960 Backus proposed a metalanguage for describing programming language syntax. This metalanguage, the now familiar "Backus-Naur Form" (BNF), was used in the ALGOL60 report [NaurP60]. Irons [61,63] recognized the implications of BNF in compiler design. A BNF description of a language could be used to drive a parser. By associating specific actions (which came to be known as the semantics) with each production of the grammar, a highly structured compiler results. For example, with the arithmetic production "E—>EE+" might be associated routines which produce machine code to add the two operands and store the result in a temporary location. Because the structure of the language was input to the compiler in a form equivalent to a formal grammar, rather than built into the code of the compiler, this method came to be called syntax-directed. The rationale of using the phrase "syntax-directed translation" is addressed more fully in Cheatham and Sattley [64].

Syntax-directed translation revolutionized the theory of translation by enabling the notion of a fixed, well-defined structure for compilers. Brooker [63] recognized that by taking advantage of this structure much of the overlapping work which had previously been forced upon all compiler writers could be handled automatically. In particular, a Compiler-Compiler or Translator-Writing System (TWS) could be developed which would accept a syntactic and semantic specification for a compiler and
output the actual compiler. The structure of XPL, one particular TWS, is shown in Figure 1.3. Skeletons of common routines, such as a lexical scanner or parser, found in all compilers, are part of the TWS. Using the syntactic specification of the source language, the TWS "completes" the scanner and parser and inserts them into the generated compiler, relieving the compiler writer of the burden of writing them himself. In addition, the TWS will often have a library of special routines which the writer may call upon in specifying the "semantics" of the language: i.e., in specifying what machine code the generated compiler should produce. Feldman and Gries [66] have written an excellent survey article on the state of the Translator-Writing System in 1966.

Figure 1.3. XPL Structure

Until 1967 no one examined the formal properties of syntax-directed translation. Research had concentrated exclusively on the pragmatics of building better TWS. Lewis and Stearns [67] developed a mathematical model of translation—the "pushdown transducer"—a pushdown automaton with output. Two years later Aho and Ullman published the first in a series of papers on the formal properties of syntax-directed translation [69a, 69b, 71, 72]. They demonstrated formally some of the inherent limitations of the method as traditionally conceived; i.e., that a
syntax-directed translation, as defined by Aho and Ullman, could only be specified between two languages if certain relationships held between them.

**Tree Transducers**

During the middle and late Sixties, Doner [65], and Thatcher and Wright [ThatW65] investigated abstract automata which accept trees as input. A tree automaton is an extension of the classical string automaton such as the finite state machine. A tree automaton will either accept or reject a tree which is input to it depending upon the structural properties of the tree. Part of their work has had an impact on the theory of translation. In addition to studying tree automata as recognizers, they also studied them as transducers, machines which not only accept trees as input, but also output trees as well. Their work in syntax-directed translation was the first to specify a formal structure for the target language. Before their studies, whatever the translator outputted was in the target language. The source had a formal syntactic definition in BNF, but the target had none. Because target languages were typically machine or assembly languages with rather trivial syntax, this did not present major difficulties. However, the advent of tree automata pointed strongly to the formalization of the target language with the same rigor as the source. With a formal definition of the target language, a tree automaton could be designed to translate to a target language with more interesting and complex syntactic structures in the target than is found in machine or assembly languages. Furthermore, with a formal definition for the target language, formal properties of the translation process itself could be investigated. Despite this fact, no major work was actually done in this direction at that time.

**Formal Models of Semantics**

Programming language semantics can roughly be divided into two categories: (i) operational or interpretive; and (ii) descriptive. Operational semantics defines the meaning of the program as a sequence of snapshots of its execution. Descriptive semantics defines the meaning of a program as a specification of the program in some other (hopefully more transparent) notation. The model of semantics which is used in this thesis is descriptive in nature. This is because of the particular requirements of a study of translation. When
one talks of translating a program, it is usually without regard to the execution of the program on a particular data set. Ideally, a translator should produce a target program which has the same input/output behavior on all data sets as the source. The flow of control within a program is normally highly data dependent. There is often an infinite number of data sets upon which the program should potentially be able to execute. Furthermore, the execution time of a program may be quite long, and in fact, the program may not halt at all on certain data sets. All of these factors lead us to conclude that translators should not rely upon information obtained from specific executions of a program. We therefore, exclude operational semantics as a model on which our language definition system will be based.

A number of researchers have attempted to formalize programming language semantics. Among the earliest was McCarthy [60] in his efforts related to LISP. Using the lambda calculus of Church [41], he formally defined the behavior of a LISP interpreter. The IBM Vienna Laboratory developed another interpretive formalism for programming language semantics— the Vienna Definition Language [LucaLS68]. This model of semantics, which uses trees to represent the state of execution of a program, has been used extensively to investigate the formal properties of interpreters and in the definition of such complex languages as PL/I [LucaM69].

The primary efforts in descriptive models of programming language semantics have come from Benson [74], Buttelmann [74], and Knuth [63b,71]. All of these authors incorporated a descriptive semantics into a context-free grammar.

**Models of Translation**

In a syntax-directed translation scheme, three approaches towards defining the source and target language are commonly followed:

1) The syntactic specifications of the source and target languages are defined **formally**. The semantics of both the source and target languages are defined **informally**.
INTRODUCTION

MODELS OF TRANSLATION

2) The syntactic specification of the source language is defined \textit{formally}. The target language is defined as the set of \textit{outputs} of the translator. No formal specification for either source or target semantics is given. The target language may be treated as the set of \textit{meanings} of the source language. The output of the translator for a particular source sentence would be its meaning. Because of this, it may be more appropriate to think of this model of translation as a model of semantics instead.

3) The source and target languages have both a \textit{formal} syntactic and \textit{formal} semantic specification.

Aho and Ullman examined both viewpoints (1) and (2) in developing their models of translation. They never specifically addressed problems of \textit{meaning} and semantic representation. Knuth in his work on attributed grammars took the second viewpoint. He appeared to view his model as a means of specifying the \textit{meaning} of a sentence instead of its translation. The disadvantage of the first two methods is that by not giving both languages a formal syntactic and semantic specification it is impossible to investigate directly the "correctness" of the translator. Under the second philosophy, if a formal specification of the target syntax and semantics were given independently in some other notation, it would be necessary to verify that the set of "meanings" of the source sentences and the independent specification of the target language were equivalent. Furthermore, in the second approach, the source and target languages are defined using different notational schemes. Consequently, the definition of a translator from language A to language B gives little insight into how to construct an inverse translator from language B to language A. Because of these and related problems, both Benson and Buttelmann have chosen the third philosophy - a formal specification of both syntax and semantics for source and target languages. Benson investigated syntax-directed translation from a category theory perspective. He showed that under certain constraints the correctness of a translator may be \textit{proven}. Benson based his work on that of Thatcher, Aho and Ullman, and Lewis and Stearns. He provided a category theoretical model of the tree automaton of Thatcher. Buttelmann defined a model of language, complete with a formal grammar and
formal semantics with the specific intent of investigating formal models of translation. Quoting Buttelmann [74],

The definition [of a phrase-structure semantics] is a model based on the notion that it is phrases which have meaning and that the meaning of a phrase is a function of its syntactic structure and of the meanings of its constituents.

Normally, the set of sentences of a language is the set of frontiers of the parse trees of a grammar. A string is or is not a sentence of the language depending on whether or not it can be parsed with respect to the given grammar. Buttelmann was the first person to use a formally defined semantics to discard, on semantic grounds, strings which would from syntactic considerations alone be sentences of the language. In his language model he associated a partial recursive function and arguments for that function with each parse tree. A sentence was said to have a meaning if the function associated with a parse of the sentence was defined when evaluated on the associated arguments. The language was simply that set of strings which had at least one meaning. In this manner, he was able to investigate the recursively enumerable sets in a framework which had the structural properties of the context-free grammar. A formal semantics for both source and target systems provides information about which mapping between languages will induce a translation. Using this fact, Buttelmann devised a procedure for automatically constructing a syntax-directed translator given only the formal descriptions of the source and target languages. He discovered sufficient conditions to guarantee the correctness of his translators and proved that all translators generated by his procedure satisfy those conditions.

**Key Issues Addressed**

There are many basic questions about formal models of translation which remain unanswered. Two of the most important questions are

a) How can the meanings of sentences be used effectively by a translator, and at what cost?

b) What translation mechanisms can be devised to handle complex syntactic relationships between source and target languages in a coherent manner,
and at what cost?

This thesis is an attempt to investigate these broad research problems. We do so by addressing seven particular questions.

1) How can the semantics of a language be defined with enough mathematical rigor to be useful in the study of formal properties of translation?

2) In what ways can the meanings of a sentence, when formally defined, be used by a syntax-directed translator in conjunction with syntactic properties of the sentence to direct the translation process?

3) Does the use of semantic information by a syntax-directed translator increase the class of language pairs which are translatable? If so, how can we characterize this larger set?

4) How does the use of semantic information by a translator affect its execution time? Can we determine a least upper bound on the execution time of a hybridized translator?

5) What are sufficient conditions to ensure that a translator is correct; i.e., that a source sentence and each of its translations share a common meaning?

6) How can we modify the tree transducer of the syntax-directed translator to induce more complex syntactic relationships between source and target grammars than is possible by using the standard tree transducer?

7) Can we provide a precise mathematical characterization of a semantic-directed translator? If so, what are its behavioral properties? What is an upper bound on its execution time?

In this thesis we attempt to answer these seven questions. In chapter III we present the model of language upon which all of our studies of translation are based. The language definition system is called a phrase-structure system. It has both a formal syntactic and a formal semantic component.
INTRODUCTION

KEY ISSUES ADDRESSED

The syntactic component is a context-free grammar. The semantic component is a formalization of Knuth's attributed grammar [68b] along the lines of Buttelmann's "phrase-structure semantics" [74]. We investigate the properties of the languages generated by these systems, which we call phrase-structure languages. The language definition systems are constrained to generate languages with certain properties which are desirable for our purpose - the study of translation. For example, each sentence has a finite number of effectively computable meanings. The importance of this fact becomes apparent when the reader recalls that the meanings of sentences are to be used to direct the translators which we design.

In chapters iv through vii we define and analyze several hybridized syntax-directed translators; i.e., translators which use both syntactic and semantic information to induce the translation. We call this family of translators phrase-structure translators. Syntactic relationships between the source and target systems give the phrase-structure translator its overall character. This prompts the use of the term "phrase-structure". These translators also use semantic information as well as syntactic to drive the translator. Consequently, we refrain from using the term "syntax-directed" to describe this method of translation. In these chapters we examine the class of language pairs translatable by each translation model, under what conditions the translators are correct, and what their execution times are. We also examine several other properties of these translators which help to provide a more complete characterization of their behavior. The various members of the family use semantic and syntactic information in different ways. Their behavioral properties vary accordingly. This leads to a better understanding of the tradeoff between increasing the syntactic "power" of the translator or increasing its semantic "power".

In chapters viii and ix we define and study a pure semantic-directed translator, TRANS. The basic definitions and theoretical foundations of the semantic-directed translator are given in chapter viii. In chapter ix we investigate the time complexity of TRANS. If a theory of translation is to lead to the construction of actual translators, it must model translators which are in some sense "efficient". In chapter ix we establish a set of efficiency criteria against which we measure the performance of TRANS. Appropriate modifications are made to TRANS to enable it to meet these standards.
Finally, in chapter x we summarize how well we have done in trying to answer the seven major questions which we address in this thesis and what their significance is in terms of the general problem of language translation.
Chapter II — Basic Terms and Definitions

You should print what I mean, not what I say.

Richard J. Daley

In this chapter we present some of the basic terminology which we will frequently use in the remainder of the dissertation.

Abbreviations

Defn D.2.1. The following abbreviations are used throughout the text.

- \( \mathbb{N} \) for integer \( n \geq 1 \), \( (1, \ldots, n) \)
- \( \emptyset \) the null set
- \( \text{INT} \emptyset \) set of integers \( \geq 0 \)
- \( \text{INT}1 \) set of integers \( \geq 1 \)
- \( \varepsilon \) empty string
- \( V^{**k} \) \((v_1 \ldots v_k \mid v_i \in \text{alphabet set} \ V \text{ for positive integer } k)\)
- \( T^+ \) \(\varepsilon\)-free concatenation closure of \( T \)
- \( \Rightarrow \) implies
- \( \text{powerset}(A) \) set of all subsets of set \( A \)
- \( x^{**y} \) for integers \( x \) and \( y \), \( x \) raised to the \( y \)-th power; for string \( x \) and integer \( y \), \( x \) concatenated to itself

14
BASIC TERMS
ABBREVIATIONS

y times

|wl| length of string w; or
cardinality of set w

x || y concatenation of strings
x and y

f(A) for relation f and set
A = domain(f), \{(f(a) \mid a \in A\}

Subscripts and Character Set

This dissertation is typed on the line printer of a
DECsystem-10. Therefore, the character set is more limited
than would otherwise be true. Furthermore, we are forced to
adopt somewhat unusual subscripting conventions. The i-th
element of an ordered set of elements, X, will be denoted
either "x(i)" or "X(i)". This notation is standard in
programming language syntax (e.g., FORTRAN and PL/I).
Multiple subscripts, when used, will be separated by commas.
For example, the (ij)-th member of X will be written as
x(i,j). When no confusion will result, x(i) will be written
without parentheses as "xi". Whenever the subscript is an
expression such as "i-1", or is a multiple subscript such as
"i,j", the subscript will always be separated from the set
name (in the case of our example "x") by parentheses.

Functions and Computation

Defn D.2.2. For integers k \geq 1 and z \geq 1, a k-ary
relation, R, is a non-empty subset of D1x..xDkxR1x..xRz
where each Di and Rj is a set. We define the domain of R to be

\{(d1,..,dk) \mid (d1,..,dk,r1,..,rz) \in R \text{ for }
\text{ } (r1,..,rz) \in R1x..xRz\}

The range of R is

\{(r1,..,rz) \mid (d1,..,dk,r1,..,rz) \in R \text{ for }
\text{ } (d1,..,dk) \in D1x..xDk\}

If domain(R) = D1x..xDk, then R is total. If range(R) is
R1x..xRz, then R is onto. If whenever (d1,..,dk,r1,..,rz)
and (c1,..,ck,r1,..,rz) belong to R,
\text{ } (c1,..,ck) = (d1,..,dk), then R is one-to-one (1-1). If
whenever \((d_1, \ldots, d_k, r_1, \ldots, r_z)\) and \((d_1, \ldots, d_k, q_1, \ldots, q_z)\) belong to \(R\), \((r_1, \ldots, r_z) = (q_1, \ldots, q_z)\) then \(R\) is a function. We often write \(R\) as

\[ R : D_1 \times \cdots \times D_k \rightarrow R_1 \times \cdots \times R_z \]

or as

\[ R(x_1, \ldots, x_k) : D_1 \times \cdots \times D_k \rightarrow R_1 \times \cdots \times R_z \]

[\*]

**Defn D.2.3.** Suppose \(f_2 : C_1 \times \cdots \times C_z \rightarrow D_1 \times \cdots \times D_y\) and \(f_1 : B_1 \times \cdots \times B_k \rightarrow E_1 \times \cdots \times E_w\) are two relations where \(E_1 \times \cdots \times E_w\) is a subset of \(C_1 \times \cdots \times C_j\) for some \(i, j \in \mathbb{Z}\), \(j = i + w - 1\). The composition of relation \(f_1\) and \(f_2\) is defined by:

\[
f_2(x_1, \ldots, x_{i-1}, f_1(x_i, \ldots, x_j), x_{j+1}, \ldots, x_z) : \]

\[
C_1 \times \cdots \times C_(i-1) \times B_1 \times \cdots \times B_k \times C_(j+1) \times \cdots \times C_z \rightarrow D_1 \times \cdots \times D_y
\]

where if \(f_1(b_1, \ldots, b_k) = \text{set } E\) of \(w\)-tuples, then

\[
f_2(c_1, \ldots, c_{(i-1)}, f_1(b_1, \ldots, b_k), c_{(j+1)}, \ldots, c_z) =
\]

\[
\{(d_1, \ldots, d_y) \mid (d_1, \ldots, d_y) \in f_2(c_1, \ldots, e_1, \ldots, e_w, \ldots, c_z) \text{ for } (e_1, \ldots, e_w) \in f_1(b_1, \ldots, b_k) \text{ for } (b_1, \ldots, b_k) \in B_1 \times \cdots \times B_k\}
\]

As stated in D.2.3, the composition of two or more functions is equivalent to the common mathematical definition of functional composition.

For each member, \((d_1, \ldots, d_y)\) in range\((f_2(x_1, \ldots, f_1, \ldots, x_z))\), there is one or more sequences of triples,

\[
(b_1, \ldots, b_k), (c_1, \ldots, e_1, \ldots, e_w, \ldots, c_z), (d_1, \ldots, d_y)
\]

such that \((d_1, \ldots, d_y) \in f_2(c_1, \ldots, e_1, \ldots, e_w, \ldots, c_z)\), and \((e_1, \ldots, e_w) \in f_1(b_1, \ldots, b_k)\).

We call each such sequence a history of \((d_1, \ldots, d_y)\). In the case of a function, there is only one history of \((d_1, \ldots, d_y) = f_2(x_1, \ldots, f_1(b_1, \ldots, b_k), x_z)\) for each \((b_1, \ldots, b_k)\) in \(\text{domain}(f_1)\), and \((c_1, \ldots, c_{(i-1)}, c_{(j+1)}, \ldots, c_z)\) in \(C_1 \times \cdots \times C_(i-1) \times C_(j+1) \times \cdots \times C_z\).

As a notational convenience we often write the composition of \(k\) functions whose domains and ranges are 1-tuples as:

\[ f_1 \# f_2 \# \ldots \# f_k : \text{domain}(f_k) \rightarrow \text{range}(f_1) \]

where \(f_1 \# \ldots \# f_k(x)\) is by definition \(f_1(\ldots(f_k)(x)\ldots)\) for \(x \in \text{domain}(f_k)\).
Defn D.2.4. It is sometimes necessary to consider a particular restriction of a relation. In particular, suppose \( f: D_1 \times \ldots \times D_n \rightarrow R_1 \times \ldots \times R_z \) is a relation. Let \( D'_1 \times \ldots \times D'_n \) be a subset of \( D_1 \times \ldots \times D_n \) in which each \( d'_i \) is either (1) a proper subset of \( D_i \) with just a single member; or (ii) simply \( D_i \) itself. If \( d'_i \) satisfies the first condition, then we say that the \( i \)-th argument of \( f \) is fixed. Otherwise, it is free. We write the restriction of \( f \) to \( D'_1 \times \ldots \times D'_n \) as

\[
f(x'_1, \ldots, x'_n): D'_1 \times \ldots \times D'_n \rightarrow R_1 \times \ldots \times R_z
\]

where \( x'_i \) is '*' if the \( i \)-th argument of \( f \) is free and \( x'_i \) is the sole member of \( D_i \) in \( D'_i \) otherwise.

For example, if the function is addition, \( \text{add}(x_1, x_2): \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z} \), then \( \text{add}(*, 2): \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z} \) is that restriction of "add" in which the first argument ranges over \( \mathbb{Z} \) and the second argument is always the integer 2.

Defn D.2.5. Let \( f: D_1 \times \ldots \times D_k \rightarrow R_1 \times \ldots \times R_z \) be a relation. The inverse relation of \( f \), \( f^{-\text{inv}} \), is

\[
\{(r_1, \ldots, r_z, d_1, \ldots, d_k) | (d_1, \ldots, d_k, r_1, \ldots, r_z) \in f\}
\]

A procedure is a set of operations which when suitably encoded is a Turing machine. A process which may be encoded as a procedure is said to be procedural. A relation \( f: D_1 \times \ldots \times D_n \rightarrow C_1 \times \ldots \times C_z \) is partial recursive if there is a procedure which enumerates \( f(d_1, \ldots, d_n) \) for each \( (d_1, \ldots, d_n) \in D_1 \times \ldots \times D_n \). An algorithm is a procedure whose Turing machine encoding is guaranteed to halt after a finite amount of computation, for all possible inputs. A procedural process which may be encoded as an algorithm is said to be algorithmic. A relation \( f: D_1 \times \ldots \times D_n \rightarrow C_1 \times \ldots \times C_z \) is total recursive if there is an algorithm which enumerates \( f(d_1, \ldots, d_n) \) for each \( (d_1, \ldots, d_n) \in D_1 \times \ldots \times D_n \). As a consequence of this definition, if \( f \) is total recursive, then for each \( (d_1, \ldots, d_n) \in D_1 \times \ldots \times D_n \), \( f(d_1, \ldots, d_n) \) must be a finite set.

The reader should note that the word "algorithmic" is being used in a technical sense in this thesis. Some
authors use the word "algorithmic" to describe a process which we would call "procedural".

**Trees**

**Defn D.2.6.** Let \( NAL \) be a finite non-empty node alphabet set. If \( NODES \), the set of nodes over \( NAL \) is \( NAL^+ \). An ordered tree is a character string which is defined inductively by:

1) \( c \in NODES \) is an ordered tree; or
2) \( c < t_1..tn> \) is an ordered tree if \( c, t_1, ..., tn \) are ordered trees, and \( c, t_1, ..., tn \) have no nodes in common.

Let \( LAL \) be a finite non-empty label alphabet set. Define a mapping, "label"

\[ \text{label}: NODES \rightarrow LAL \]

such that there are an infinite number of nodes which map to each member of \( LAL \).

A labeled ordered tree over nodeset \( NODES \) and labelset \( LAL \) is a character string which is defined inductively by:

1) \( (c, \text{label}(c)) \) is a labeled ordered tree if \( c \in NODES \).
2) \( (c, \text{label}(c)) < t_1..tn> \) is a labeled ordered tree if \( c \in NODES \) and labeled ordered trees \( t_1, ..., tn \) all have no common nodes.

The set of all labeled ordered trees over \( NODES \) and \( LAL \) and the mapping "label", is denoted \( \text{TREE}(NODES, LAL, \text{label}) \).

Two labeled ordered trees are **equal** if they differ only by the nodes which form them. Formally, labeled ordered trees, \( p \) and \( q \), are equal if

1) \( p = (c, \text{label}(c)) \); and
2) \( q = (d, \text{label}(d)) \); and
3) \( \text{label}(c) = \text{label}(d) \)

or if
4) \( p = (c, \text{label}(c)) < t_1..tn> \); and
5) \( q = (d, \text{label}(d)) < r | \ldots | r n > \); and

6) \( \text{label}(c) = \text{label}(d) \); and

7) \( t_i = r_i \) for \( i \in n \).

**Example E.2.1.** Suppose \( \text{NAL} = (a, b, c) \) and \( \text{LAL} = (E, A, 1, +) \). Further suppose that \( \text{label} : \text{NODES} \rightarrow \text{LAL} \) is defined so that

\[
\begin{align*}
\text{label}(a) &= E, \\
\text{label}(b) &= A, \\
\text{label}(c) &= 1, \\
\text{label}(aa) &= 1, \\
\text{label}(ab) &= E, \\
\text{label}(ac) &= +.
\end{align*}
\]

Then \( \text{TREE} (\text{NODES}, \text{LAL}, \text{label}) \) includes:

\[
(a, E) \quad (a, E) < (ac, +) > \quad (c, 1) < (a, E) < (b, A) < (aa, 1) > (ab, E) >
\]

For the purposes of this dissertation, the particular nodes which define a tree are often not important, while the labels of those nodes are. For this reason we will often write just the label of a node-label pair. For example, the three trees given above would be abbreviated

\[
E \quad E < + > \quad 1 < E < A > 1 > E >
\]

Throughout the thesis, trees will be given pictorially whenever doing so is pedagogically helpful. The last tree above, \( 1 < E < A > E > \), is pictured in Figure 2.1. Note that again, only the labels of the nodes are explicitly stated.

![Figure 2.1](image)

**Figure 2.1.** Drawing of \( 1 < E < A > E > \).

Trees as defined in D.2.6 are similar to the standard linear representation for trees used in many texts. We therefore feel free to use such standard terms as height, weight, frontier, and root of a tree without further explanation. For convenience, the frontier of tree \( t \), \( \text{frontier}(t) \), shall sometimes be abbreviated \( \text{fr}(t) \). Similarly, height(\( t \)), weight(\( t \)), and root(\( t \)) will sometimes be abbreviated \( \text{ht}(t) \), \( \text{wt}(t) \), and \( \text{rt}(t) \), respectively. We
shall also refer to the common relationships between nodes of a tree—child, parent, descendent and ancestor without formal definition. Hereafter, unless otherwise noted, all trees considered will be labeled ordered trees. If the node or label alphabets are understood, they will not be explicitly mentioned.

At this point, it will prove convenient to establish certain conventions about symbols used in the body of the thesis. Unfortunately, because of the limited character set available, these notational conventions cannot be uniformly applied, but unless otherwise noted,

1) The letters p, q, r, t, and u stand for trees.
2) The letters f, g, and h stand for relations.
3) The letters b, c, and d stand for nodes.
4) The letters i, j, k are either subscript variables or integers.
5) The letters m, n, x, y, z are integers.
6) The letters v and w are strings.
7) For the sake of readability, whenever the range over which an identifier is being varied is clear from the context, the range will not be explicitly stated.
8) frontier(t, i) denotes the i-th member of frontier(t).
9) Upper case letters and words will normally denote sets. For example, "T" would normally denote a set of trees, since "t" stands for a tree.
10) We often find it convenient to impose an ordering relation on sets so that we may reference the i-th member of the set. Unless specified, the ordering is arbitrary. We denote the i-th element of ordered set A by either A(i) or a(i).

* As a concession to the feminist movement, we will use the neutral terms "child" and "parent" rather than the more traditional "son" and "father".
The usage conventions for letters of the alphabet established above also apply to letters which have been primed ('), double-primed ("'), or subscripted.

**Defn D.2.7.** We define a partial function which specifies how one tree is composed to another. The function has three arguments, the first two of which are trees, the third an integer. The second tree is composed to the i-th frontier position of the first tree, provided that the i-th node of the frontier of the first tree and the root of the second tree have the same label. Formally, let 
\[ TR = \text{TREE}(\text{NODES}, \text{LAL}, \text{label}). \]
The partial function \( \text{compose} : \text{TR} \times \text{TR} \times \text{INT} \rightarrow \text{TR} \) is defined inductively by:

1) \( \text{compose}(c, t', m) \) is defined iff
   \[ \text{label}(\text{rt}(t')) = \text{label}(c), \ c \text{ is not in } \text{nodes}(t'), \text{ and } m = 1. \]
   In that case, \( \text{compose}(c, t', m) = t' \).

2) For \( t = c < t_1 \ldots t_n > \), \( \text{compose}(t, t', m) \) is defined iff
   \[ 0 < m \leq |\text{fr}(T)| \]
   \[ \text{label}(\text{rt}(t')) = \text{label}(\text{fr}(t, m)) \]
   In that case, there exist unique non-negative integers \( x, y, z \) such that
   \[ |\text{fr}(t_1) \ldots \text{fr}(t(x-1))| = y; \text{ and} \]
   \[ |\text{fr}(t_x)| = z; \text{ and} \]
   \[ y < m < y+z. \]
   Then, \( \text{compose}(t, t', m) = c < t_1 \ldots t(x-1) \text{ compose}(t_x, t', m-y) \ t(x+1) \ldots t_n > \)

As a convention, we will always assume that two trees which are to be composed have disjoint sets of nodes.

**Example E.2.2.** Let \( t = C, t' = C<DE>, \text{ and } t'' = E<10>. \)
\( \text{compose}(t, t', 1) = C<DE> \)
\( \text{compose}(t', t'', 2) = C<DE<10>> \)
\( \text{compose}(t', t'', 1) \) is undefined because \( \text{label}(\text{fr}(t', 1)) = D \neq E = \text{label}(\text{rt}(t'')) \)
\( \text{compose}(t', t'', 3) \) is undefined because \( |\text{fr}(t'')| < 3 \)
Defn D.2.8. This definition introduces a notation for describing trees which was first used by Buttelmann [74]. Using it, any subtree of a tree may be "isolated".

If \( t = \) compose\((*,t_k,n_k)\#...#compose\((*,t_l,n_l)\)\(t_0\))

where

1) \( n_l = 1 \); and
2) \( n_i = |fr(t_l)\ldots fr(t(i-1))| \), for \( 2 \leq i \leq k \); and
3) \( |fr(t_0)| = k \).

then \( t = t_0[t_1...t_k] \). This notation for trees is called box bracketing. \( t_0 \) is called the supertree and \( t_1,...,t_k \) are called the subtrees of \( t \). Note that there is an infinite number of box bracketings of any tree. This is shown more clearly in E.2.3.

Example E.2.3. Pictured in Figure 2.2 are the two box bracketings of \( A < B < C > D < E > \) given below. The supertree is distinguished with horizontal lines and the subtrees are distinguished by vertical and diagonal lines. The first tree is the "grafting" of \( B < C > \) and \( D < E > \) onto \( A < B D > \). The second tree is the "grafting" of \( B < C > \) and the single node \( E \) onto \( A < B D < E > \). Thus, both box bracketings break up \( A < B < C > D < E > \) into three sub-trees, with only one sub-tree in common, namely \( B < C > \).

\[
A < B < C > D < E > = A < B D > [B < C > D < E >] = A < B D < E > [B < C > E]
\]

\( A < B C > \) has an infinite number of box bracketings.

\[
\]
BASIC TERMS
TREES

Figure 2.2. Two Box Bracketings of A<B<C>D<E>.

[*]

Grammars

Defn D.2.9. A context-free grammar (cfg) is a 4-tuple (Vn, Vt, Ax, Pr) where

1) Vn is the finite non-empty nonterminal vocabulary.
2) Vt is the finite non-empty terminal vocabulary.
3) Ax ∈ Vn is the start symbol or axiom.
4) Pr, the production set, is an finite ordered non-empty set of trees of unit height such that the root node is labeled from Vn and the frontier nodes are labeled from Vn ∪ Vt. Ax does not label a frontier node of any member of Pr.

The union of Vn and Vt is denoted V for vocabulary.

[*]

D.2.9 excludes productions involving the empty string. They are excluded because structurally, derivations involving such productions are difficult to represent and they can be eliminated without restricting the class of sentences generable. Hereafter, the word "grammar" and "context-free grammar" will be used interchangeably unless otherwise noted.

Defn D.2.10. Let Tr be a non-empty set of trees. The closure of Tr under composition is denoted gen(Tr) (for "generation of Tr"). Tr is also called a generating set for gen(Tr). The members of Tr are sometimes called elementary trees.
Let $G = (V_n, V_t, A_x, Pr)$ be a CFG. The closure of $Pr$ under tree composition is denoted $\text{phr}(Pr)$ (for "phrase-structures"). The following sets are also defined:

1) ("terminated phrase-structures") $\text{phrt}(Pr) = \{ t \in \text{phr}(Pr) | \text{label}(c) \in V_t \text{ for } c \in \text{fr}(t) \}$.

2) ("complete phrase-structures") $\text{phrc}(Pr) = \{ t \in \text{phrt}(Pr) | \text{label}(\text{rt}(t)) = A_x \}$.

3) ("axiomatic phrase-structures") $\text{phra}(Pr) = \{ t \in \text{phr}(Pr) | \text{label}(\text{rt}(t)) = A_x \}$.

[*]

**Defn D.2.11.** The phrase-structures of grammar $G$ are the phrase-structures of its production set $Pr$. Similarly for terminated, axiomatic, and complete phrase-structures.

[*]

**Defn D.2.12.** The sentences of a CFG $G$, $\text{sen}(G)$, are

$\{ w | w = \text{fr}(t), t \in \text{phrc}(G) \}$

The parse or derivation trees of $w$, $\text{parse}(w)$, are

$\{ t | t \in \text{phrc}(G) \text{ and fr}(t) = w \}$

[*]

D.2.12 defines as the sentences of a grammar what would be called the language of the grammar in classic formal language studies. We reserve the word "language" for a somewhat different concept, involving meaning, introduced later.

**Defn D.2.13.** This definition states a precise uniform method for describing the sequence of compositions used to form a tree in $\text{phr}(Tr)$ from the elementary trees in $Tr$.

Let $Tr$ be an ordered non-empty set of trees. A leftmost derivation (lmd) of $t \in \text{phr}(Tr)$ is a list of triples, $Z$,

$(p_1, n_1, m_1), (p_2, n_2, m_2), \ldots, (p_z, n_z, m_z)$

where each triple is a member of $(Tr) \times \mathbb{N} \times \mathbb{N}$ and
1) \( p_i = Tr(m_i) \).
2) \( t = compose(*,pz,nz)# . .#compose(*,p2,n2)(p1) \).
3) \( i > j \implies n_i > n_j \).

**DER-T** is the set of leftmost derivations of \( t \).

We say subtree \( t' \) of tree \( t \) is a sub-structure of \( t \) according to leftmost derivation, \( Z \), just in case \( (t',n_j,m_j) \) is a triple of \( Z \) for some integer \( j \). If \( Tr \) is the set of productions of a cfg, then each tree in \( Tr \) has unit height. It is commonly known that each phrase-structure of a cfg has a unique leftmost derivation [EveyR63]. For cfgs, it follows immediately that if \( t' \) is a subtree of \( t \), then \( t' \) is also a sub-structure of \( t \). Let us say two lmds, \( Z \) and \( Z' \) are isomorphic if (i) \( Z \) and \( Z' \) have the same number of triples; and (ii) for \( (p_i,n_i,m_i) \) and \( (p'_i,n'_i,m'_i) \), it is true that \( p_i = p'_i \) and \( n_i = n'_i \). If two isomorphic trees are distinct and one is used in the \( i \)-th triple of lmd \( Z \), the other in the \( i \)-th triple of lmd \( Z' \), then two non-identical, but isomorphic leftmost derivations of the same tree result. For generating sets which have trees of arbitrary height, a tree may have several non-isomorphic derivations. For example, if \( Tr \) contained the five trees:

\[
S<A> \quad A<B> \quad S<A<B>> \quad B<C<D>> \quad D<E>
\]

then (a) and (b) are sub-structures of

\[
t = S<A<B<C<D<E>>>
\]

but (c) is not, even though there is a box bracketing of \( t \) which isolates \( B<C> \), namely \( S<A<B>>[B<C>C<D<E>]> \).

\[
a) \quad S<A<B>> \\
b) \quad B<C>D<E>> \\
c) \quad B<C>
\]

In the event that a tree has more than one leftmost derivation, a particular leftmost derivation must be explicitly referenced in order to claim that some subtree \( t' \) of \( t \) is also a sub-structure of \( t \).

It is sometimes useful to be able to distinguish between the weight of a tree, which is the number of subtrees of unit height which it contains, and the pseudo-weight of a tree, which is the number of subtrees in
a particular lmd of that tree.

**Defn D.2.14.** The *pseudo-weight* of tree $t$, $\text{pwt}(t)$, with respect to a leftmost derivation $Z$, is the number of triples in $Z$.

[*]

If every leftmost derivation of $t$ has triples all of whose first components have unit weight, then the pseudo-weight of that tree is unique, equaling its weight. However, in general, this need not be true. For example, the pseudo-weight of $S\langle A\langle B\rangle\rangle$ from (a) above is equal to one for leftmost derivation

$$(S\langle A\langle B\rangle\rangle, 1, 3)$$

but it is equal to two for leftmost derivation

$$(S\langle A\rangle, 1, 1), (A\langle B\rangle, 1, 2)$$

An additional concept which we will use later is the notion of a leftmost "partial" derivation of a tree $t$. A leftmost partial derivation of $t$ is the leftmost derivation of a supertree of $t$ with respect to some box bracketing of $t$.

**Defn D.2.15.** Let $Tr$ be a leftmost derivation of $t_0 \in \text{phr}(Tr)$

$$(p_1, n_1, m_1), \ldots, (p_z, n_z, m_z)$$

We say that $Z$ is a *leftmost partial derivation of* $t$ (lmpd) if $t_0[t_1 \ldots t_n]$ is a box bracketing of $t$.

$Z$ is a *maximal leftmost partial derivation of* $t$ (mlmpd) if

1) $t_0[t_1 \ldots t_n]$ is a box bracketing of $t$; and

2) For $i \in n$, there is no lmpd of any $t_i$.

[*]

It is possible to have in the set of productions of a grammar, *useless productions*. A production of cfg $G$ is useless just in case it can never occur as a sub-structure of a parse of some sentence of $G$. Since a useless production does not contribute to the generative power of the grammar to which it belongs, we prefer to restrict the
production set to those productions which are useful in generating sentences. This is captured formally in D.2.16.

**Defn D.2.16.** A context-free grammar is reduced iff for each tree \( t \in \text{Pr} \) there exists a tree \( t' \) in \( \text{phrc}(G) \) such that \( t \) is a sub-structure of \( t' \). 

In the remainder of the paper, it is always assumed that context-free grammars are reduced. Bar-Hillel, Perles and Shamir [63] showed that any context-free grammar can be reduced algorithmically (see also Ginsberg [66], page 21).

**Time Complexity**

In order to analyze the execution time of the procedures developed in this thesis, we introduce a small number of related concepts and notations. For a detailed examination of complexity measures, the reader should see Blum [67], Hartmanis and Hopcroft [71] or Aho, Hopcroft and Ullman [74].

The model of computation we use is a pseudo-PL/I machine. We choose, without loss of generality, this high-level machine rather than the more traditional Turing machine model because of the convenience high-level constructs offer in concisely expressing our procedures. As a general rule, the execution time of an instruction in a procedure is equal to one time step. The only major exception to this rule is the subroutine call. The execution time of a subroutine call is the execution time of the subroutine itself. We measure the time complexity of a procedure in terms of the length of the input to it. Procedure \( M \) is said to execute or operate within time bound

\[ \text{TIME-M} : \text{INT} \rightarrow \text{INT} \]

where

1) \( \text{TIME-M} \) is a function which is defined on \( |w| \) iff \( M \) halts on input \( w \).

2) \( \text{TIME-M}(|w|) = n \Rightarrow \) the execution time of \( M \) on input \( w \leq n \).
We denote by PTIME the class of deterministic procedures which operate in a time bounded by a polynomial function.
"Linguistic description minus grammar equals semantics"

Fodor/Katz

Introduction

In this chapter we introduce the model of language which will be used to develop translators. We call the language model, written as a variant of Knuth's [68b,71] attributed grammar, a "phrase-structure system". The phrase-structure system has two components, a context-free grammar and a formal semantics. Meanings are assigned to a sentence indirectly by assigning them to the parses of the sentence. The assignment process is illustrated more clearly in Figure 3.1.

Figure 3.1. Attributed Grammar.
Figure 3.1a is a parse tree of a sentence. The model of language is such that at each node of the tree is "stored" a "chunk" of semantic information which represents the "contribution" of that node to the overall meaning of the parse. Each "chunk" is a member of the semantic "universe". Several items of information can be stored at the same node. In order to distinguish between them we give each chunk a name in the same way we use variable names in programming. These names are called "attributes". In Figure 3.1b we show the parse tree of figure 3.1a with the associated attributes. Included in the semantic specification of the language are relations which assign values to the attributes of the tree in terms of the values of other attributes. The tree complete with attributes and their values is shown in Figure 3.1c. In order to keep the assignment process from being circular, the attributes of terminal symbols are given initial values. The meaning is "summarized" at the root of the tree. The root has a single attribute whose value is the meaning of the tree.

**Basic Definitions**

**Defn D.3.1.** A phrase-structure system (pss) is an ordered pair \( Q = (G, \text{SEM}) \) where

1) \( G = (V_n, V_t, A_x, Pr) \) is a reduced cfg called the underlying grammar of \( Q \), \( UG(Q) \).

2) \( \text{SEM} \) is a 6-tuple \((\text{SYN-ATT}, \text{INH-ATT}, \text{att}, U, \text{att-val}, Ru)\) where

a) \( \text{SYN-ATT} \) and \( \text{INH-ATT} \) are finite non-empty disjoint sets of character strings. The members of \( \text{SYN-ATT} \) are called synthesized attributes, and the members of \( \text{INH-ATT} \) are called inherited attributes. Their union is called "ATT". Each member of ATT is called an attribute.

b) \( \text{att} \) is a partial function

\[
\text{att} : V \rightarrow \text{powerset}(\text{ATT}) \rightarrow Q
\]

such that \( \text{att}(Ax) \) contains no inherited attributes and exactly one synthesized attribute. From \( \text{att} \) we define two other functions,

\[
\text{syn-att} : V \rightarrow \text{powerset}(\text{SYN-ATT})
\]

\[
\text{inh-att} : V \rightarrow \text{powerset}(\text{INH-ATT})
\]
where
\[
\text{syn-att}(v) = \{ \text{atti} \in \text{att}(v) \mid \text{atti} \in \text{SYN-ATT} \}
\]
\[
\text{inh-att}(v) = \{ \text{atti} \in \text{att}(v) \mid \text{atti} \in \text{INH-ATT} \}
\]

Each member of \text{att}(v) is an attribute of \textit{v}; each member of \text{syn-att}(v) is a synthesized attribute of \textit{v}; and each member of \text{inh-att}(v) is an inherited attribute of \textit{v}.

c) \textit{U} is a recursively enumerable (r.e.) set of objects, the \textit{universe of discourse}.

d) \text{att-val} is a partial function
\[
\text{att-val}: \mathbb{V} \times \text{ATT} \rightarrow \text{powerset}(\text{U})
\]
such that for each \textit{v} \in \mathbb{V}, \text{att-val}(v,\text{attu}) is defined iff \text{attu} \in \text{att}(v), and \text{att-val}(v,\text{attu}) is a non-empty r.e. set whenever defined. We call \text{att-val}(v,\text{attu}) the set of possible values of \textit{v}.

e) \textit{Ru} is a finite non-empty set of ordered pairs. Each member of \textit{Ru}
\[
(p, \text{rules}(p))
\]
is called a \textit{production-rule pair}, where \textit{p} is a production of \textit{Pr}, and \text{rules}(p) is a set of ordered triples called \textit{semantic rules}.

Before we can specify the members of \text{rules}(p), for each \textit{t} \in \text{phr}(\textit{Pr}) we must define three partial functions:
\[
\text{att-t: nodes}(t) \rightarrow \text{powerset(ATT)}
\]
\[
\text{inh-att-t: nodes}(t) \rightarrow \text{powerset(INH-ATT)}
\]
\[
\text{syn-att-t: nodes}(t) \rightarrow \text{powerset(SYN-ATT)}
\]

where
\[
\text{att-t}(c) = \text{att}(\text{label}(c))
\]
\[
\text{inh-att-t}(c) = \text{inh-att}(\text{label}(c))
\]
\[
\text{syn-att-t}(c) = \text{syn-att}(\text{label}(c))
\]

We often write \((c, \text{attu}) \in \text{att-t}\) as "\text{attu}(c)"", and speak of it as the "occurrence of \text{attu} at node \textit{c}". We do so because conceptually, it is convenient to think of each node of \textit{t} as having its own set of attributes, namely those of its label, "hanging off" from it.
Suppose \( b \in \text{nodes}(p) \) and \( \text{label}(b) = B \). If (i) \( B \in \mathcal{V}_t \), \( \text{att}u \in \text{inh-att}(B) \); or (ii) \( B \in \mathcal{V}_n \) and \( \text{att}u \in \text{att}(B) \); then there is exactly one semantic rule in \( \text{rules}(p) \)

\[
(\text{att}u(b), f : D_1 x \ldots x D_y \rightarrow C, \\
\left( (\text{att}u_1(b_1), \ldots, \text{att}u_y(b_y)) \right)
\]

where

1) \( \text{att}uj \in \text{att}_p(bj) \).
2) \( D_j = \text{att-val}(\text{label}(bj), \text{att}u) \).
3) \( C \) is a subset of \( \text{att-val}(B, \text{att}u) \).
4) \( f : D_1 x \ldots x D_y \rightarrow C \) is a total recursive relation.
5) \( \text{att}u \in \text{SYN-ATT} \implies \) each \( b_i \) is a child of \( b \).
6) \( \text{att}u \in \text{INH-ATT} \implies \) each \( b_i \) is the parent of \( b \).

For convenience, we sometimes write a semantic rule as

\[
\text{att}u(b) = f(\text{att}u_1(b_1), \ldots, \text{att}u_y(b_y)) ; \\
D_1 x \ldots x D_y \rightarrow C
\]

\([*]\)

**Example E.3.1.** The following defines a phrase-structure system for postfix arithmetic expressions:

A phrase-structure system for Postfix Arithmetic Expressions

\[
\text{PAE} = (G, \text{SEM}) \text{ where } \\
G = ((S,E), (1,2,+,*), S, Pr) \text{ where the productions are given below.}
\]

synthesized attribute is "val''.

\[
\text{att}(S) = \text{att}(E) = \text{att}(1) = \text{att}(2) = "\text{val}"
\]

\[
\text{att-val}(E, \text{val}) = \text{att-val}(S, \text{val}) = \text{INIT} \\
\text{att-val}(1, \text{val}) = 1; \text{att-val}(2, \text{val}) = 2
\]
We will use a uniform notation for referring to particular nodes of a tree of height one. For such a tree, "x0" is its root and xi is the i-th frontier node.

**Productions**

- **S<E>**  \( \text{val}(x0) = \text{val}(x1) \)
- **E<EE+>**  \( \text{val}(x0) = \text{plus}(\text{val}(x1), \text{val}(x2)) \)
- **E<EE*>**  \( \text{val}(x0) = \text{mult}(\text{val}(x1), \text{val}(x2)) \)
- **E<1>**  \( \text{val}(x0) = 1 \)
- **E<2>**  \( \text{val}(x0) = 2 \)

"plus" and "mult" are the common addition and multiplication functions over the integers.

[*]

As a convention, an uppercase Q (primed, double-primed, or subscripted) will always stand for a phrase-structure system.

If \( y = f(x_1, \ldots, x_n) \), then it is customary to say that the variable \( y \) "depends" on each of the variables \( x_1 \) through \( x_n \). Dependence is a transitive relation; i.e., if \( y \) is dependent upon \( x \), and \( x \) is dependent upon \( z \), then \( y \) is also dependent upon \( z \). We apply the concept of dependence to the semantic rules of \( \text{Ru} \) in a phrase-structure system.

**Defn D.3.2.** Suppose \( t \in \text{phr}(G) \) has leftmost derivation \( Z \), \( c \) and \( d \) belong to \( \text{nodes}(t) \), \( \text{attu} \in \text{att-}t(c) \) and that \( \text{attv} \in \text{att-}t(d) \). \( \text{attu}(c) \) is dependent on \( \text{attv}(d) \) if

1) \( (t', n, m) \) is a triple of \( Z \); and
2) \( c, d \in \text{nodes}(t') \); and
3) rules \( (t') \) of the m-th prp of \( \text{Ru} \) contains

\[
\text{attu}(c) = f(\ldots, \text{attv}(d), \ldots);
\]

\[\ldots x(\text{att-val(label}(d), \text{attv}))x \ldots \rightarrow C \]
where C is a subset of att-val(label(c), attu) 
or if

4) \( b \in \text{nodes}(t), \ attw \in \text{att-t}(b); \) and
5) \( \text{attu}(c) \leq d \ \text{attw}(b); \) and
6) \( \text{attw}(b) \leq d \ \text{attv}(d). \)

If \( \text{attu}(c) \) and \( \text{attv}(d) \) satisfy conditions (1)-(3), then \( \text{attu}(c) \) is directly dependent upon \( \text{attv}(d). \) If \( \text{attu}(c) \) and \( \text{attv}(d) \) satisfy conditions (4)-(6) then \( \text{attu}(c) \) is indirectly dependent upon \( \text{attv}(d). \)

If \( c \) is a node of tree \( t, \) then contingent upon the definition of \( Q, \) \( \text{attu}(c) \) could depend upon the values of attributes of any other node of \( t. \) This dependence can lead to a circular definition in which the value of an attribute is dependent upon itself. Systems defined in this way are called improper. Knuth showed that it is possible to algorithmically check for circular definition. We will always assume that any pss under consideration is proper; i.e., is not circular.

The motivation for the terms "inherited" and "synthesized" should now be clear. An attribute of some node \( n \) is inherited if the attributes upon which it is directly dependent belong to nodes which are ancestral to \( n. \) An attribute is synthesized if the attributes upon which it directly depends belong to nodes which are descendents of \( n. \)

Suppose \( t' \) is a sub-structure of \( t \in \text{phr}(G). \) The "entrance" attributes of \( t' \) are those attributes of \( t' \) which depend directly upon the values of attributes of nodes in \( (\text{nodes}(t) - \text{nodes}(t')) \). The "exit" attributes of \( t' \) are those attributes of \( t' \) on which attributes of nodes in \( (\text{nodes}(t) - \text{nodes}(t')) \) may directly depend. In essence, the exit and entrance attributes of \( t' \) are those attributes of \( t' \) which "interface" with the attributes of the rest of \( t. \) The significance of the entrance and exit attributes will become more apparent in the next chapter where they will be fundamental in proving that a stated condition is sufficient to guarantee that a translator is semantic-preserving.

Defn D.3.3. Let \( t \in \text{phr}(G), \) and let \( c \) be either a frontier or root node of \( t. \) The \textbf{exit} attributes of \( c \) are \( \text{syn-att-t}(c) \) if \( c = \text{rt}(t) \) and \( \text{inh-att-t}(c) \) if \( c \in \text{fr}(t). \)
The entrance attributes of c are syn-att-t(c) if c ∈ fr(t) and inh-att-t(c) if c = root(t).

The next definition defines a special closure of the semantic rules of a phrase-structure system under composition. It is similar in concept to the closure of a set of relations under composition.

**Defn D.3.4.** Suppose t ∈ phr(G), c ∈ nodes(t), v = label(c), attu ∈ att(v). Define relation

\[
\text{rule-**NODESxATTx(phr(Pr))—> (NODESxATT)xRELx(NODESxATT)+}
\]

where REL is the set of all total recursive relations, and

1) If attu(c) is not directly dependent upon the attributes of any node of t, then rule-*(c, attu, t) is

\[
( (c, attu), \text{id} \cdot \text{att-val}(v, attu) \rightarrow \text{att-val}(v, attu), ((c, attu)) )
\]

where "id" is the identity function.

2) If attu(c) is directly dependent upon \(\{\text{attu}(cl), \ldots, \text{attu}(cn)\}\), then c belongs to some sub-structure, t', of t such that

\[
t' \quad \text{rules}(t')
\]

is a production-rule pair of Ru and

\[
\text{attu}(c) = f(\text{attu}(cl), \ldots, \text{attu}(cn));
\text{Dlx..xDn—>B}
\]

belongs to rules(t'). Define rule-*(c, attu, t) as

\[
( (c, attu), f(g1, \ldots, gn);B1x..xBz—>B,
(\text{arg}(1), \ldots, \text{arg}(n)) )
\]

where

a) gi is the second component in rule-*(ci, attui, t); and
b) If the third component of \( \text{rule-}^* (c_1, \text{attu}_1, t) \) is 
\( ((b_l, \text{attv}_l), \ldots, (b_y, \text{attv}_y)) \) then \( \text{arg}(i) = (b_l, \text{attv}_l), \ldots, (b_y, \text{attv}_y) \).

For convenience, we sometimes denote \( \text{rule-}^* (c, \text{attu}, t) \) as
\[ \text{attu}(c) = \text{rule-}^* (\text{arg}(1), \ldots, \text{arg}(n)) : D_1 \ldots D_n \rightarrow B \]

We also sometimes write \( \text{rule-}^* (c, \text{attu}, t) \) as "\( \text{rule-}^* (\text{attu}(c), t) \)".

The second component of \( \text{rule-}^* (\text{attu}(c), t) \) is the composition of total recursive relations. Therefore, it also is a total recursive relation.

We define \( \text{rule-}^* (t) \) as
\[ \{ \text{rule-}^* (\text{attu}(c), t) \mid \text{attu}(c) \text{ is an exit attribute of } t \} \]

[\(*\]

We wish to assign values to the attributes occurring in \( t \). These values will be the basis on which meanings are assigned to parses. To do so, we evaluate the relation of \( \text{rule-}^* (\text{attu}(c), t) \) for each attribute \( \text{attu} \) of each node of \( c \).

**Defn D.3.5.** Suppose \( t \in \text{phrc}(G) \), \( c \in \text{nodes}(t) \), and \( \text{attu} \in \text{att-t}(c) \). If \( \text{label}(c) \in \mathcal{V}t \), \( \text{attu} \in \text{syn-att-t}(c) \), then the set of values of \( \text{attu}(c) \), \( v(\text{attu}(c)) \), is simply \( \text{att-val}(\text{label}(c), \text{attu}) \).

If \( \text{rule-}^* (\text{attu}(c), t) = \)
\[ \text{attu}(c) = f(\text{attu}(c_1), \ldots, \text{attu}(c_n)) : D_1 \ldots D_n \rightarrow B \]

then the set of value of \( \text{attu}(c) \), \( v(\text{attu}(c)) \), is
\[ (f(v_1, \ldots, v_n) \mid v_i \in v(\text{attu}(c_i))) \]

Since \( f \) is a total recursive relation, then the set of values assigned to each attribute of each node of \( t \) is both finite and effectively computable in a finite time.

We wish to distinguish between the individual histories of how each member of \( v(\text{attu}(c)) \) is computed. \( \text{att-t} \) may be partitioned into a finite number of equivalence classes \( E_1, \ldots, E_m \) of equivalence relation \( E \), where \( \text{attu}(c) \in \text{attv}(d) \) if and only if \( (i) \in \text{attu}(c) \leq_d \text{attv}(d) \); (ii)
attv(d) ≤ d attu(c); or (iii) c is d and attu is attv. If attu(c) is not equivalent to attv(d) then each history of u ∈ v(attu(c)) is independent of every history of v ∈ v(attv(c)). Let (attui(d1),...,attuk(dk)) be a set of representative elements of E. One history of each ui ∈ v(attui(d1)) for each i ∈ m determines a unique value for each attribute occurrence in att-t. Call such a set of m histories, a history of t. Let HISTORY-T be the set of all such histories. Since the set of histories of each ui is finite in number, then so must be HISTORY-T. For each member of HISTORY-T, a total function "val-t" is defined:

\[ \text{val-t: att-t} \rightarrow U \]

where \( \text{val-t(attu(c))} \) is the value of \( \text{attu(c)} \) assigned by that history to \( \text{attu(c)} \).

We say that \( \text{val-t} \) specifies an evaluation of t.

The set of all \( \text{val-t} \) functions for \( t \in \text{phr(G)} \) is denoted \( \text{VAL-T} \). \( |\text{VAL-T}| \) is a positive finite integer for \( t \in \text{phr(G)} \) and \( \text{VAL-T} = 0 \) for \( t \) not in \( \text{phr(G)} \).

[\(*\) Example E.3.2. The parse tree shown in Figure 3.2 is in phrc(UG(PAE)).

![Parse Tree](image)

**Figure 3.2.** \( t \in \text{phr(UG(PAE))} \).

Using the production-rule pairs of PAE, the attribute, "val", of each node is assigned a value as shown in Figure 3.3.
Although the grammar of a phrase-structure system is context-free, the semantics is powerful enough to define the many context-sensitive features of programming languages. This is done by using a context-free grammar to generate a superset of the desired trees and then using semantic constraints to "discard" the undesirable trees from consideration. To accomplish this, a special value — "err" — is assumed to be in the semantic universe, $U$. Let $c$ be any node in $t \in \text{parse}(w)$. Assume that $a t t u \in a t t - t(c)$. If for some evaluation of $t$, $v a l - t(a t t u(c)) = \text{err}$, then that evaluation of $t$ is said to be deviant (in the sense of Chomsky [57,65]). Otherwise, the evaluation is non-deviant. The set of non-deviant parse trees of $w$ is that set of parse trees which have a non-deviant evaluation. We let $\text{NDVAL-T}(w)$ (for "non-deviant" VAL-T) denote that subset of VAL-T which do not assign "err" to any attribute of $t$.

Example E.3.3. If extended to include division, phrase-structure system PAE might contain a rule:

$$E < EE > \quad \text{if val}(x_2) \neq 0 \text{ then }$$

\[ \text{val}(x_2) = \text{"err"} \]

\[ \text{else } \text{val}(x_2) = \text{div}(\text{val}(x_1), \text{val}(x_2)) \]

[$*$]

Defn D.3.6. $\text{phr}(Q) = \text{phr}(G)$, $\text{phra}(Q) = \text{phra}(G)$, $\text{phrt}(Q) = \text{phrt}(G)$, and $\text{phrc}(Q) = \text{phrc}(G)$.
(t | t ∈ phrc(G), NDVAL-T ≠ Q)
If no evaluation of any parse tree is deviant, then Q is complete, otherwise it is incomplete. [*]

Defn D.3.7. Let Q be a phrase-structure system. The sentences of Q are

\[ \text{sen}(Q) = \{ \text{fr}(t) \mid t \in \text{phrc}(Q) \} \].

[*]

Example E.3.5. One sentence of pss PAE is:

1 2 1 * +

[*]

The meanings of a sentence are defined indirectly by associating meanings with the parses of the sentence.

Defn D.3.8. Let t ∈ parse(w). The meaning of t with respect to val-t ∈ NDVAL-T, is defined as

\[ \text{mean}(t, \text{val-t}) = \text{val-t}(\text{attu}(\text{rt}(t))) \]

where attu is the lone synthesized attribute of Ax, the label of rt(t).

The set of meanings of t is

\[ \text{mean}(t) = \{ \text{mean}(t, \text{val-t}) \mid \text{val-t} \in \text{NDVAL-T} \} \]

The set of meanings of w, for w ∈ sen(G) is

\[ \text{mean}(w) = \{ \text{mean}(t) \mid t \in \text{parse}(w) \} \]

[*]

Defn D.3.9. The phrase-structure language of Q, L(Q) =

\[ \{ (w, u) \mid w \in \text{sen}(G), u \in \text{mean}(w), \text{mean}(w) \neq Q \} \]

[*]
Example E.3.5. The meaning of sentence "I 2 I * +" from phrase-structure system PAE is:

3

The definitions of "sentence" and "language" distinguish between strings which have meaning and strings together with their meanings. This distinction is drawn so that in the following chapters, it will be easy to differentiate between transducers (which relate sentences per se) and translators (which induce semantic-preserving relations between languages).

Ambiguity

The notion of ambiguity is commonplace in classical formal language theory. The traditional notion of ambiguity is syntactic in nature. A grammar is said to be ambiguous if there is a sentence which has more than one parse tree. This one concept of ambiguity is adequate for purely syntactic studies. However, in studying phrase-structure systems, it is necessary to distinguish between several related notions of "ambiguity" which are either syntactic or semantic in nature. These are presented in the next few definitions.

Defn D.3.13. If in sen(G) there is a sentence w which has two distinct parse trees, then w is syntactically ambiguous and G is syntactically ambiguous.

Example E.3.6. If for some grammar G, the trees pictured in Figure 3.4 were in phrc(G), then G would be syntactically ambiguous because the sentence "I" would have two parses.

\[
\begin{align*}
S & \rightarrow T S \\
T & \rightarrow I \\
S & \rightarrow I
\end{align*}
\]

Figure 3.4. Two Parses of The Same String - "I".
Defn D.3.11. 0 is semantically ambiguous if and only if

1) there is a relation in some semantic rule in Ru which is not a function; or

2) for some terminal symbol v, with synthesized attribute attu, att-val(v) has more than one member.

Otherwise, 0 is semantically unambiguous.

[*]

D.3.10 is the traditional definition of syntactic ambiguity of a grammar. D.3.11 is the analogous definition for semantic ambiguity of a phrase-structure system.

Example E.3.7. A phrase-structure system which contained the following production-rule pair would be semantically ambiguous because f is a relation which is not a function:

\[ E<?> \text{ val}(x0) = f(\text{val}(x1)) \]

where att-val(?,val) = 1 and f(1) = (1,2).

If att-val(?,val) were (1,2), then the phrase-structure system would be semantically ambiguous because a synthesized attribute of a terminal symbol has more than one possible value.

The reason we use the term "semantically ambiguous" to describe such systems, is that if a system is semantically ambiguous, then a tree, t, may have more than one val-t function defined for it.

[*]

Defn D.3.12. If a sentence has more than one meaning, then the sentence is semantically ambiguous. If one or more sentences of a language is semantically ambiguous, then the language is semantically ambiguous.

[*]

Example E.3.8. In a phrase-structure system whose language is the union of base-2 and base-3 numbers, the following would have two meanings:

101 means 5 in base-2 and 10 in base-3.
111 means 7 in base-2 and 13 in base-3.
Note the difference between D.3.11 and D.3.12. The system is semantically ambiguous if there is more than one set of attribute values which can be assigned to a single complete tree. But semantic ambiguity of the system is neither necessary nor sufficient to ensure that a sentence of the language is semantically ambiguous. It is not necessary because a sentence could have two meanings by having two distinct parses each assigned a different meaning. It is not sufficient because the attributes of the root of a complete tree might not depend upon the attributes associated with the semantically ambiguous phrase-structure. We therefore, speak of the ambiguity of a language apart from the ambiguity of the system defining it.

Restrictions

The number of distinct meanings of \( t \) is less than or equal to \( |\text{VAL-T}| \). From the discussion in D.3.5, in which \( \text{VAL-T} \) is defined, it is clear that \( |\text{VAL-T}| \) is finite. Hence, a parse tree of string \( w \) may have only a finite number of meanings. If some parse tree \( t \) of \( w \) has a sub-structure whose frontier is just a single node whose label is the same as the label of its root, then \( w \) has an infinite number of parses. Call such a sub-structure, linear recursive (lrss). If a sentence has a parse with a lrss, then it has an infinite number of parses. For a particular \( \text{VAL-T} \), if the value of an entrance attribute of a linear recursive sub-structure did not necessarily equal the value of the corresponding exit attribute, then each parse could have a different set of meanings. Consequently, a sentence could have an infinite number of meanings. We wish to use the meanings of a sentence to partially direct a translator. Furthermore, whether or not a string has a non-deviant parse is used as the criterion for deciding whether or not that string is a sentence of the language. Therefore, it would be very awkward computationally to permit sentences to have an infinite number of meanings. So we restrict the definition of a phrase-structure system so that any evaluation of a tree containing a linear recursive sub-structure, the values of corresponding entrance and exit attributes must be equal. Therefore, even though a sentence may have an infinite number of parses, it will have only a finite number of meanings. The set of parses of sentence \( w \) whose meanings include all of the meanings of \( w \) is simply that subset of parse\((w)\) which does not include any linear recursive sub-structures. We find it convenient to give
that subset of parse\((w)\) whose members have no linear recursive sub-structures a special name, \(\text{nlrss-parse}(w)\). This finite set of parses may be effectively constructed using Earley's algorithm [68].

**Basic Properties**

Having developed the terminology and notations necessary to examine phrase-structure systems, we now present several of their basic properties.

**Thm T.3.1.** A set \(W\) is recursive iff it is the set of sentences of a pss.

**Proof:** (if) A string \(w\) is a sentence of some pss iff

(i) \(w = f_r(t)\) for some \(t \in \text{phrc}(G)\); and (ii) \(t\) is non-deviant. Given \(w\) and \(G\), the set of parses of \(w\) is effectively constructable. If there are no parses, then \(w\) is not in \(\text{sen}(Q)\). Because of the restriction on lrss in the definition of phrase-structure systems given above,

\[\text{mean}(w) = \{\text{mean}(t) \mid t \in \text{nlrss-parse}(w)\}\]

\(\text{nlrss-parse}(w)\) is a finite set effectively computable using Earley's parsing algorithm. We noted above that for any \(t \in \text{phr}(G)\), \(\text{VAL-T}\) is a finite effectively computable set. Hence the set of evaluations of all trees in \(\text{nlrss-parse}(w)\) is a finite effectively computable set. An evaluation is deviant iff it assigns some attribute the value "err". Hence, for each \(t \in \text{nlrss-parse}(w)\), \(\text{NDVAL-T}\) is a finite effectively computable set. \(w \in \text{sen}(Q)\) iff \(\text{NDVAL-T} \neq \emptyset\).

(only if) Let \(W\) be any recursive set. There is a total recursive function \(f\), the characteristic function of \(W\), with the property that \(f(w) = 1\) if \(w \in W\) and \(f(w) = 0\) otherwise. Define \(Q\) as follows:

1) \(V_n = (A_x, E)\).

2) \(V_t = \text{the alphabet over which } W \text{ is defined.}\)

3) \(P_r = (A_x<E>, E<a>E>, E<a>)\).

4) \(\text{syn-att}(A_x) = \text{syn-att}(E) = "value".}\)
5) Ru is the following production-rule pairs:

\[
\begin{align*}
A x<E> & \quad \text{if } f(\text{value}(x1) = 1 \text{ then} \\
& \quad \text{value}(x3) = 1 \\
& \quad \text{else} \\
& \quad \text{value}(x0) = "\text{err}" \\
E<aE> & \quad \text{value}(x0) = a \mid \text{value}(x2) \\
E<a> & \quad \text{value}(x0) = a
\end{align*}
\]

for all \(a \in \mathcal{V}_t\).

The rules pass the frontier of the parse tree up to the phrase-structure \(A x<E>\). The meaning of any string in \(L(G)\) is 1 if \(w \in W\) and "err" otherwise. Hence, \(w \in \text{sen}(Q)\) iff \(w \in W\).

\[\star\]

**Thm T.3.2.** A set \(W\) is recursively enumerable iff it is the set of meanings of the sentences of a pss.

**Proof:**

(if) We showed in the proof of T.3.1 that the set of meanings of any sentence is a finite effectively computable set. The sentences of \(G\) are simply a context-free language and hence are recursively enumerable. Therefore, the set of meanings of the sentences of \(Q\) are effectively enumerable.

(only if) Every r.e. set is the range of some total recursive function. Construct a pss \(Q\) identical to the one constructed in the proof of T.3.1, except that \(R_u(1)\) is

\[
AX<E> \quad \text{val}(x0) = g(\text{val}(x1))
\]

where \(g\) is a total recursive function whose range is r.e. set \(W\).

\[\star\]

**Defn D.3.13.** Two phrase-structure systems are equivalent if and only if their languages are equal.

\[\star\]

The axiom of a phrase-structure system can only be used as the root of a complete tree. If a phrase-structure system is to have an infinite language, it must have at
least two nonterminals. The following theorem shows that only two are needed in order to define all languages generable from any phrase-structure system. Intuitively, the "power" of the syntax is transferred almost totally into the semantics.

**Thm T.3.3.** If $L$ is the language of $Q$ where the cardinality of $V_n > 2$, then $L$ is also the language of some $Q'$ where the cardinality of $V_{n'} = 2$.

**Proof:** $Q'$ has the following rules:

\[
\begin{align*}
Ax' < E' > & \quad mn(x0) = f(mn(x1)) \\
E' < a \ E' > & \quad mn(x0) = a \| mn(x2) \\
E' < a > & \quad mn(x0) = a
\end{align*}
\]

for all $a \in V_t$.

These rules pass $w = fr(t)$ to the phrase-structure $Ax' < E' >$. The relation $f$ parses $w$ according to the underlying grammar of $Q$, producing set $nlrss-parse(w)$. It then computes $NDVAL-T$ for each $t \in nlrss-parse(w)$, and, finally, using $NDVAL-T$ to compute the meanings of $w$, sets $f(w)$ equal to $mean(w)$ if $INDVAL-T > 0$ and sets $f(w)$ equal to "err" otherwise.

Then $L(Q) = L(Q')$.

[*]

We distinguish those phrase-structure systems which use only synthesized attributes from those which use a mixture of synthesized and inherited attributes.

**Defn D.3.14.** If all attributes used in $R_u$ are synthesized, then $Q$ is a **synthesized phrase-structure system**, denoted syn-pss. Otherwise, it is a **general phrase-structure system**, gen-pss.

[*]

The next result, originally shown by Knuth [68b], follows immediately from the construction of $Q'$ in the proof of T.3.3.
Corollary C.3.1. For each phrase-structure system which uses both synthesized and inherited attributes there is an equivalent phrase-structure system which uses only synthesized attributes.

Proof: The phrase-structure system \( Q' \) constructed in T.3.3 uses only synthesized attributes.

Table PSS

We now introduce a third type of attribute, one not discussed by Knuth — the table attribute. The value of an attribute at a specific node may be thought of as information stored at that node. The values are dependent upon one another. Therefore, the assigning of attribute values proceeds in an ordered manner. So evaluation in an attributed grammar may be described in terms of information flow. The value of an attribute at one node "flows" into its neighbor nodes. There it may be used as an argument to compute the values of its dependent neighbors. Suppose the datatype of a variable is declared in a program and that this information is computed at node \( c \) as the value of attribute "type". It may be necessary to pass the data-type to many other points in the parse tree. For example, the data-type may be needed in order to detect necessary conversions in arithmetic expressions involving the declared identifier. The datatype is constant. It seems cumbersome to have to specify every flowpath which is necessary to send the data-type to each node which depends upon its value. To alleviate the burden of having to explicitly define all of the flow paths for attributes with constant values, we introduce the table attribute. A table attribute may be thought of as a SNOBOL table. Semantic rules add and reference information in the table.

Defn D.3.15. A table phrase-structure system (tab-pss) is a gen-pss with the following modifications:

1) ATT is partitioned into three disjoint subsets: SYN-ATT, INH-ATT, and TAB-ATT. Each member of TAB-ATT is called a table attribute.

2) In addition to the two functions syn-att and inh-att, we define a third function

\[ \text{tab-att: } V \rightarrow \text{powerset(TAB-ATT)} \]
such that \( \text{tab-att}(Ax) = Q \). For each \( v \in V \)
\( \text{tab-att}(v) \) is called the set of \textit{table attributes}
of \( v \) and \( \text{att}(v) = \text{syn-att}(v) \cup \text{inh-att}(v) \cup \text{tab-att}(v) \) is called the set of \textit{attributes} of \( v \).

3) \( \text{att-val}(v, \text{attu}) = \text{att-val}(w, \text{attu}) \) for \( v, w \in V, \text{attu} \in \text{TAB-ATT} \). Hence we write \( \text{att-val}(v, \text{attu}) \) as 
"\text{att-val}(\text{attu})".

4) The production-rule pairs in \( R_u \) are modified. For each \( p \in Pr \), there is one \( prp: \)

\[ p \quad \text{rules}(p) \]

For each member of \( \text{attu}(c) \) of \( \text{att-t}(p) \), there is one semantic rule in \text{rules}(p)

\[ \text{attu}(x) = f(\text{attu}(x_1), \ldots, \text{attu}(x_y)) : \]

\[ \text{D} | x_1 \ldots x_y \rightarrow B \]

where

a) if \( \text{attu} \in \text{SYN-ATT} \) or \( \text{INH-ATT} \) then (i) \( x = c_i \) (ii) \( B \) is a subset of \( \text{att-val}(\text{label}(c), \text{attu}) \). Otherwise, (i) \( x = \text{attv}(c) \in \text{syn-att-t}(c) \cup \text{inh-att-t}(c) \); (ii) \( B \) is a subset of \( \text{att-val}(\text{attu}) \).

b) if \( \text{attu}_i \in \text{SYN-ATT} \) or \( \text{INH-ATT} \) then (i) \( x_i \in \text{nodes}(p) \); and (ii) \( D_i \) is equal to \( \text{att-val}(\text{label}(x_i), \text{attu}_i) \). Otherwise, (i) \( x_i = \text{attv}(c_i) \in \text{syn-att-p}(c_i) \cup \text{inh-att-p}(c_i) \); and (ii) \( D_i \) is equal to \( \text{att-val}(\text{label}(x_i), \text{attv}) \).

c) if \( \text{attu} \in \text{SYN-ATT} \) then

\[ x_i \in \text{nodes}(p) \Rightarrow \]

\[ x_i \text{ is a child of } c \]

\[ x_i = \text{attv}(c_i) \Rightarrow \]

\[ c_i \text{ is a child of } c \]

d) if \( \text{attu} \in \text{INH-ATT} \) then

\[ x_i \in \text{nodes}(p) \Rightarrow \]

\[ x_i \text{ is the parent of } c \]

e) if \( \text{attu} \in \text{TAB-ATT} \) then

\[ x_i = \text{attv}(c_i) \Rightarrow c_i = c \]

[*]
The dependence of one attribute occurrence upon another is extended to tab-pss in the natural way. Similarly for "rule-*". Suppose there are two distinct occurrences, \(\text{attu}(c)\) and \(\text{attu}(d)\), of table attribute \(\text{attu}\) in \(t\). For definiteness, consider some particular evaluation of \(t\). Suppose rule-\(\text{*}(\text{attu}\text{attu}(c),t)\) and rule-\(\text{*}(\text{attu}\text{attu}(d),t)\) are the rules used to assign values to \(\text{attu}\), and that \(v(\text{attu}(d)) = v(\text{attu}(c))\). The values of \(\text{attu}(\text{attu}(d))\) and \(\text{attu}(\text{attu}(c))\) are equal. They are the union of the two values obtained by evaluating rule-\(\text{*}(\text{attu}\text{attu}(d),t)\) and rule-\(\text{*}(\text{attu}\text{attu}(c),t)\) individually.

Example 6.3.9. The sentences of the tab-pss given below are programs with declarations and assignment statements. The declarations dictate the data-type of the identifiers used in the assignment statements.

Synthesized attributes are "val", "junk", "etype", "mn", and "name".

Table attribute is "type".

\(\text{att}(S) = \text{att}(P) = \text{att}(\text{ASN}) = "mn".\)

\(\text{att}(\text{DCL}) = \varnothing.\)

\(\text{att}(\text{ID}) = "name", "junk", "type".\)

\(\text{att}(E) = "val", "type".\)

\(\text{att}(l.) = \text{att}(l) = "val", "type".\)

<table>
<thead>
<tr>
<th>Productions</th>
<th>Semantic Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S&lt;P&gt;)</td>
<td>(mn(x0)=mn(x1))</td>
</tr>
<tr>
<td>(P&lt;S\ P&gt;)</td>
<td>(mn(x0)=mn(x1) | mn(x2))</td>
</tr>
<tr>
<td>(P&lt;S&gt;)</td>
<td>(mn(x0)=mn(x1))</td>
</tr>
<tr>
<td>(S&lt;\text{DCL}&gt;)</td>
<td>(mn(x0)=\varepsilon)</td>
</tr>
<tr>
<td>(S&lt;\text{ASN}&gt;)</td>
<td>(mn(x0)=mn(x1))</td>
</tr>
<tr>
<td>(\text{DCL}&lt;\text{REAL},\text{ID}&gt;)</td>
<td>(\text{type}(\text{name}(x2))=&quot;\text{real}&quot;.)</td>
</tr>
</tbody>
</table>
if ltype(name(x2)) = 2 then
  junk(x2) = "err".

DCL<INT ID>
  type(name(x2))="int".

if ltype(name(x2)) = 2 then
  junk(x2) = "err".

ID<A>
  name(x1)="A"

ID<B>
  name(x1)="B"

ID<C>
  name(x1)="C"

ASN<ID = E>
  if type(name(x1))=etype(x3) then
    mn(x0)=(name(x1),type(x1),val(x3))
  else
    if type(name(x1))="real" then
      mn(x0)=(name(x1),"real",1.)
    else
      mn(x0)=(name(x1),"int",1)
  end

E<1>
  etype(x0)="int"
  val(x0)=1

E<1.>
  etype(x0)="real"
  val(x0)=1.

A sample sentence in this language is:

INT A
REAL C
REAL B
C = 1.
B = 1.
A = 1.

Its meaning is:
\[(C, "real", 1.)(B, "real", 1.)(A, "int", 1)\]

In E.3.9, "junk" is used to check for programs with multiple declarations of a single variable, ID. Every time a variable is declared, the datatype of the declaration is inserted into \(\text{type(name(ID))}\). If the cardinality of \(\text{type(name(ID))}\) is greater than one, then the variable has been multiply declared. Programs with this property are marked deviant in accordance with normal programming conventions. Also note how values are assigned to occurrences of the table attribute "type". The argument of "type" on the left-hand side of the semantic rule is not a reference to a node, "x2", but to an attribute of a node; i.e., \(\text{name(x2)}\). If \(\text{name(x2)}\) were "A" in an actual occurrence of the phrase-structure \(\text{DCL<REAL ID>}\), then \(\text{type(A)}\) would be set to "real". References to the type of \(A\) could be made from any point in the complete tree regardless of where that reference is with respect to the phrase-structure which assigned \(\text{type(A)}\) the value "real". Thus, the references are associative.

**Thm T.3.4.** For every table phrase-structure system there is an equivalent phrase-structure system without table attributes.

**Proof:** A synthesized attribute, "coll", collects values from the nodes of the tree and passes them up to a node labeled S were \(Ax<S>\) is the top-most sub-structure of each parse. The node labeled S then has the complete table in "coll". This is passed down to the tree nodes in its entirety as an inherited attribute. The nodes reference this inherited attribute instead of the tabular one. This is equivalent to the notion of a table attribute.

[\*]

**Complexity**

Assume that any function in a semantic rule executes in one time step. For what corresponds to the special case where the attributed grammar is semantically unambiguous, Lewis, Rosenkrantz, and Stearns [74] have shown that the attributes of any complete tree of an attributed grammar may
be evaluated in

$$O(W)$$  \hspace{1cm} \text{(3.1)}

time, where $W$ is the weight of the tree. The evaluation algorithm first performs a topological sort \cite{Knuth1968} on the attribute occurrences of the complete tree. The ordering relationship is "$\leq_d$". This takes $O(A)$ time where $A$ is the number of attributes in the complete tree. Once the sort is finished, the attributes are evaluated in their sorted order. The meaning of a tree may be computed in less than or equal to

$$O(A)$$

time steps. The number of occurrences of attributes in a tree is less than or equal to some constant times the weight of the tree. This gives time bound 3.1. We will generalize their analysis to the case where the execution time of each function $f_i$ in a semantic rule, is bounded by a polynomial function $F$ of the length of its input and such that for all $w$ in the domain of $f$, $|f(w)| \leq |w|$. If

$$f = f_1#f_2#\ldots#f_z(w)$$

is the composition of $z$ such functions, then the execution time of $f$ is less than or equal to

$$F_1(n)+F_2(n)+\ldots+F_z(n)$$

The main distinction between the attributed grammar studied by Lewis, Rosenkrantz and Stearns and the phrase-structure system is that the relations in the production-rule pairs of the system need not be functions. We analyze the simpler case where the system is semantically unambiguous.

The set of parses of $w$ without linear recursive sub-structures may be obtained using Earley's algorithm in $O(n^{**3} + pn)$ time, where $n$ is the length of $w$ and $p$ is the number of parses. Then the total number of time steps is less than or equal to

$$O(n^{**3} + pW + pW^*F(n))$$  \hspace{1cm} \text{(3.2)}

where $W$ is the maximum number of productions in any parse and $F$ is the maximum complexity of any function in a semantic rule of $Ru$. The weight of any parse tree which has no linear recursive sub-structures is bounded by $k*n$ for some constant $k$. Time bound 3.2 may then be rewritten as
For the syn-pss, the second term of time bound 3.3 is superfluous since no sorting of the attribute dependencies is necessary. Otherwise, the time bound is identical. Therefore, time bound 3.3 without the second term is an upper time bound for syn-pss. If the table references in a tab-pss are truely associative, then no additional time is involved in using them over normal synthesized and inherited attributes. Thus, time bound 3.3 is a valid upper bound on the time required for computing meanings in a tab-pss.

We conclude that if (i) the number of parses of a sentence is polynomially bound; (ii) each function in the set of semantic rules of R is in PTIME; and (iii) for each function in a semantic rule, |f(w)| ≤ |wl|; then the time required to evaluate the set of parses in nlrss-parse(w) is polynomially bound as a function of |wl|.

If any total recursive relation can compute a meaning of w in time $T(|wl|)$, then there is a phrase-structure system which computes that meaning of w in time $T(|wl|) + |wl|^{*2}$.

**Thm T.3.5.** Let L be any language such that its sentences, $sen(L)$, are a recursive set; each sentence has a finite number of meanings; and the set of meanings of its sentences, $mean(L)$, is an r.e. set. If there is a total recursive relation f such that $f(w) = mean(w)$ for $w \in sen(L)$ and $f(w) = "err"$ for $w$ not in $sen(L)$ and which executes in time $F(|wl|)$, then there is a pss Q such that $L(Q) = L$ and each member of $mean(w)$ is computable in time $F(|wl|) + O(|wl|^{*2})$.

**Proof:** Let Q be the pss constructed in the second part of the proof of T.3.1. The concatenation operation is executed $|wl|-1$ times. The sum of the lengths of the arguments of each call of the concatenation function are 2, 3, ..., |wl|. Because of the simple nature of the concatenation operation, we assume that the execution time is directly proportional to the sum of the lengths of the arguments. Therefore, the total execution time necessary to compute the argument of $f(val(x1))$ is $O(|wl|^{*2})$. The time necessary to compute each member of $f(w)$ is by hypothesis bounded by $F(|wl|)$. Therefore, the total time necessary to compute any meaning of w is

$$F(|wl|) + O(|wl|^{*2})$$
Decidability Questions

It is well-known that it is undecidable whether or not a context-free grammar is syntactically ambiguous (see [CantD62] or [FloyR62]). In the next theorem we show that the answer to a similar question about the semantic ambiguity of a language is also undecidable.

Thm T.3.6. It is undecidable whether the language of a phrase-structure system is semantically unambiguous.

Proof: We reduce the question of deciding whether an arbitrary context-free grammar is syntactically ambiguous to this problem. Let G be a cfg. For the semantics of the grammar associate with the root of a parse tree the structure of the tree itself; i.e., the meaning of a parse tree is the parse tree itself. L(Q) is semantically ambiguous iff L(G) is syntactically ambiguous. It is undecidable whether L(G) is syntactically ambiguous. Therefore, it is undecidable whether or not L(Q) is semantically ambiguous.

[*]

For an arbitrary context-free grammar, G, it is decidable whether L(G) is empty, finite, or infinite. The following theorem shows that these properties are no longer decidable for L(Q) even though the syntactic framework of the phrase-structure system is a context-free grammar.

Thm T.3.7. It is undecidable whether the language of a phrase-structure system is empty, finite, or infinite for an arbitrary phrase-structure system.

Proof: A context-dependent grammar (cdg) is a generative grammar all of whose productions have the form

\[ xAy \rightarrow xzy \]

where x and y belong to \( V^+ \cup \{ \epsilon \} \), \( z \in V^+ \), and \( A \in V_n \) [ChomN59]. It is known that the emptiness (finiteness, infinity) problem for context-dependent grammars is recursively unsolvable [ ]. We reduce the question of deciding whether the language of an arbitrary cdg is empty (finite, infinite) to the problem of solving the emptiness (finiteness, infinity) problem for phrase-structure systems. Let G be a context-dependent grammar and let \( G' \) be its context-free "super" grammar; i.e., if \( G = (V_n, V_t, A_x, P_r) \) then \( G' = (V_n, V_t, A_x, P'_r) \)
where if 
\[ xAy \rightarrow xzy \]

belongs to \( Pr \), then 
\[ A<z> \]

belongs to \( Pr' \). Modify \( Pr' \) so that it has a new axiom 
\[ Ax' \] and \( Pr' \) has one more production 
\[ Ax'<Ax> \]

Define a phrase-structure system \( Q \) with underlying grammar \( G' \). The semantic rules pass the derivation tree's structure up to the node labeled \( Ax \). The semantic rule associated with the production \( Ax'<Ax> \) checks whether the syntax tree satisfies the context-dependent constraints of \( G \). If so then the meaning of the tree is set to "ok". Otherwise, the meaning is set to "err". \( \text{sen}(Q) = \text{sen}(G) \). Therefore, 
\( L(Q) \) is empty (finite, infinite) iff \( \text{sen}(G) \) is empty (finite, infinite). If it were decidable whether \( L(Q) \) is empty (finite, infinite), then it would be decidable whether \( \text{sen}(G) \) is empty (finite, infinite).

\[ (*) \]

In general, undecidable properties of recursive sets are undecidable for phrase-structure systems as well.

**Summary**

In this chapter we have defined and analyzed a language definition system called the phrase-structure system. This grammar model has both a formal syntax and a formal semantics. The syntactic component is a context-free grammar, while the semantics is a formalization of Knuth's attributed grammar along the lines of Buttelmann's phrase-structure semantics. We have (i) analyzed the class of languages defined by phrase-structure systems; (ii) introduced several related concepts of syntactic and semantic ambiguity for both phrase-structure systems and their languages; (iii) determined the order of time required to compute the set of meanings of a sentence; and (iv) examined several decision problems for the languages of phrase-structure systems.
Chapter IV - Phrase-Structure Translation.

The automatic translation of languages is essentially a problem of sign-substitutions, that is, formulation and programming of a substitution procedure permitting the signs of the target language to be so selected and arranged as to convey the information contained in the signs of the source language.

Georgetown University
Machine Translation
Project-1963

Introduction

In this chapter we define and analyze the properties of the complete phrase-structure transducer. The complete transducer is an abstract machine which is used to induce a transduction from one language to another. It is called "complete" because the source and target systems which are two of the parameters of the machine are complete phrase-structure systems; i.e., systems which do not generate semantically deviant parses. Syntactic relationships between the source and target systems give the transducer its overall structure. This prompts the use of the name "phrase-structure transducer". Other variants of the phrase-structure transducer, introduced in later chapters, will use semantic information as well as syntactic to drive the transducer. Because of this, we refrain from using the term "syntax-directed" to describe the method of transduction.

Basic Definitions

Defn D.4.1. A complete phrase-structure transducer (cpst) is a 4-tuple \( M = (Q_1, Q_2, \text{TABLE}, \text{ASSOC}) \) where:
PHRASE-STRUCTURE TRANSLATION
BASIC DEFINITIONS

1) \( Q_1 \) is a complete syn-pss \((G_1, SEM_1)\), the source system.

2) \( Q_2 \) is a complete syn-pss \((G_2, SEM_2)\), the target system.

3) \( ASSOC: Vn_1 \rightarrow Vn_2 \) is a partial function such that \( ASSOC(Ax_1) = Ax_2 \), and \( att(v) = att(ASSOC(v)) \) for \( v \in Vn_1 \).

4) \( TABL \) is a finite non-empty table. The i-th row has three entries:

\[
t \quad t' \quad x
\]

where

a) \( t \in phr(UG(Q_1)) \).

b) \( t' \in phr(UG(Q_2)) \), denoted \( t-TABL(i) \). \( t' \) is said to correspond to \( t \). We write this as \( corr(t', i) = t \).

c) \( ASSOC(label(rt(t))) = label(rt(t')) \).

d) \( x = x_1, \ldots, x_n \) is a string of integers called the x-vector. If \( fr(t) \) is \( h_1, \ldots, h_m \) and \( fr(t') \) is \( g_1, \ldots, g_n \) then

i) \( label(g_i) \in Vn_2 \Rightarrow x_i = 0 \).

ii) \( label(g_i) \in Vn_2 \Rightarrow x_i = j \)

where \( label(h_j) \in Vn_1 \) and \( ASSOC(label(h_j)) = label(g_i) \).

The x-vector is also called the association or index vector, denoted \( x-TABL(i) \).

The union of all source \( t \in TABL \) is called \( s-TABL \), and the union of all target \( t' \in TABL \) is called \( t-TABL \). The i-th row of \( TABL \) is denoted \( TABL(i) \). The \( s-TABL \) and \( t-TABL \) entries of the i-th row are \( s-TABL(i) \) and \( t-TABL(i) \), respectively.

[∗]

As a tree transducer per se, the complete transducer closely resembles the syntax-directed translation schema defined by Aho and Ullman [69]. Their formalism does not, however, include a TABL and an ASSOC mapping as given here. Replacing the three entries in a single row of the table,
they have a lone entry of the form

\[ A \rightarrow w, v \] (4.1)

where \( A \) is a nonterminal and \( w \) and \( v \) are strings over the vocabulary alphabet. Each nonterminal in \( w \) is associated with one nonterminal in \( v \) and conversely. This is indicated by subscripting the associated nonterminals of \( w \) and \( v \) with the same integer. In our notation, production 4.1 would look like

\[ A^{w} \rightarrow A^{v} \rightarrow x_{1}, \ldots, x_{k} \]

where \( k = \|v\| \), the number of nonterminals labeling nodes of \( w \) is equal to the number of nonterminals labeling nodes of \( v \), and \( x_{i} = x_{j} > 0 \Rightarrow i = j \). Thus, the syntax-directed translation schema is a restriction of the complete phrase-structure transducer because (i) the source and target entries in the table all have unit height; (ii) the association mapping is a total identity function; and (iii) each target nonterminal is associated with exactly one source nonterminal and conversely.

The next definition states how the cpst induces transductions. If \( t \) and a leftmost partial derivation of \( t \) are given to \( A.4.1 \), then the output tree, \( t' \), is called the "phrase-transform" of \( t \). Intuitively, the algorithm proceeds by "breaking up" \( t \) into sub-trees all of which are members of \( s\text{-TABL} \). Proceeding in a top-down manner, one or more occurrences of a \( t\text{-TABL} \) member is substituted for each of the source sub-trees. If a source sub-tree is \( s\text{-TABL}(i) \), then \( t\text{-TABL}(i) \) is substituted for it. The index vectors determine both the number and position of the \( t\text{-TABL} \) entries in \( t' \).

As a preliminary to D.4.2, we denote the set of all leftmost derivations of trees in \( \text{phr}(G) \) by \( \text{LMD}(G) \).

**Defn D.4.2.** Let \( M \) be a cpst. Let \( p \) be in \( \text{phr}(Q1) \). Suppose \( Z \) is a leftmost partial derivation of \( p \) with respect to \( s\text{-TABL} \):

\[ (p_{1}, n_{1}, m_{1}), \ldots, (p_{z}, n_{z}, m_{z}) \]

The phrase-transform, \( \text{IFP} \text{phr}(Q1) \times \text{LMD}(s\text{-TABL}) \rightarrow \text{phr}(Q2) \), is a partial relation computed by A.4.1.
**Algorithm A.4.1.**

**Purpose:** Compute the phrase-transform of cpst M.

**Input:** p ∈ phr(Q1) and Z, a leftmost partial derivation of p with respect to s-TABL.

**Output:** A finite set of trees in phr(Q2).

**Steps:**

1) Initially, $TFP(p, Z) = p'$, a node whose label is $\text{ASSOC}(\text{label}(\text{rt}(p)))$. $p'$ is said to be "associated" with $\text{rt}(p)$. We write this as $\text{assoc}(p') = \text{rt}(p)$.

2) If $p' = TFP(p, Z)$,
   
   * $q \in \text{tabl}$
   * $q'$ is a row of TABL, for some $i \in Z$, $q = p_i$, $(c_1, ..., c_m)$ are nodes on $\text{fr}(p', y_1), ..., \text{fr}(p', y_m)$, respectively, each associated with $\text{rt}(p_i)$, where $j > k \Rightarrow y_j > y_k$, then $TFP(p, Z) = \text{compose}(*, q', y_1) \ldots \text{compose}(*, q', y_m)(p')$

   If $x_i = j > 0$, then for each occurrence of $q'$ composed to $p'$, $\text{assoc}(\text{fr}(q', i))$ is $\text{fr}(p_i, j)$.

$TFP(p) = \{TFP(p, Z) \mid Z \text{ is a 1mpd of } p\}$

We impose an arbitrary ordering on $TFP(t)$ so that we may reference its $i$-th member. If during some step of A.4.1, $\text{assoc}(d) = c$ for $c \in \text{nodes}(t)$, $d \in \text{nodes}(t')$, $t' \in TFP(t)$, it follows from the definition of $\text{ASSOC}$ that $\text{ASSOC}(\text{label}(c)) = \text{label}(d)$. Therefore, the composition indicated in step 2 of A.4.1 is well-defined, and $\text{att}(\text{label}(c)) = \text{att}(\text{label}(d))$. For each $i \in Z$, we say that $\text{att}_i$ of $c$ is associated with $\text{att}_i$ of $d$, and write this as $\text{assoc}(\text{att}_i(d)) = \text{att}_i(c)$.

For $p' = TFP(p, Z)$, there is a 1md $Z'$ of $p'$ with respect to t-TABL:

$$(p'_{1, n'_{1, m'}_{1}}), \ldots, (p'_{y, n'y, m'y})$$

where for each $j \in y$, there is an $i \in Z$ such that (i) $\text{assoc}(\text{rt}(p'_j)) = \text{rt}(p_i)$; and (ii) some row of TABL:
PHRASE-STRUCTURE TRANSLATION
BASIC DEFINITIONS

\[(p', l, n', l, m', l), \ldots, (p', y, n', y, m', y)\]

where for each \(j \in y\), there is an \(i \in z\) such that (i) \(\text{assoc}(rt(p'j)) = rt(pi)\); and (ii) some row of \(\text{TABL}:\)

\[q \quad q' \quad x\]

with \(q = pi\) and \(q' = p'j\), was applied in forming \(p'\). \(Z'\) is said to be the \textit{Imed} of \(p'\) induced by \((p, Z)\) and \(p'j\) is said to be the \textit{sub-structure} of \(Z'\) induced by \((pi, Z)\).

**Example E.4.1.** Using \(\text{pst}-1\) of Appendix A, the phrase-structure pictured in Figure 4.1 transduces to the infix arithmetic expression pictured in Figure 4.2.

![Figure 4.1: Source Parse of 21+](image)

![Figure 4.2: Target Parse of (2)+(1)](image)
In their work on syntax-directed translation, Aho and Ullman only defined tables whose entries have unit height. This is because they did not think of the translation table as being constructed from two independently defined source and target systems. Rather, they viewed the table itself as being the formal specification of the source and target grammars. (Recall that Aho and Ullman never specifically addressed questions of semantics.) We prefer to follow Buttelmann [74] and establish the source and target systems independently of the transducer. This allows the definition of a language to be used in several different transducers without modification. One of the major disadvantages of having one language defined by several unrelated grammars is that it will in general be algorithmically impossible to verify that the grammars define the same language. An example (pst-1) of a complete phrase-structure translator is given in Appendix A. This example also demonstrates the significance of having TABL entries with height greater than one. Without the use of trees with greater than unit height, no complete transducer which uses the source and target systems of pst-1 could induce the same translation as does pst-1.

The sentences of a complete phrase-structure system form what is classically called a context-free language. Since the input parameters of a cpst are complete source and target systems, many interesting transductions cannot be performed simply because the languages themselves cannot be input to the transducer. In the next chapter we study transducers whose source and target systems are general phrase-structure systems. This changes the class of sentences which can be transduced from the context-free languages to the recursive sets.

**Defn D.4.3.** Let $M$ be a cpst. If $w \in \text{sen}(Q_1)$, then the **string-transform**, $\text{TFS}: \text{sen}(Q_1) \rightarrow \text{sen}(Q_2)$, is a relation defined by $\text{TFS}(w) =$

\[
(\text{fr}(t') \mid t \in \text{parse}(w), \ t \in \text{phrc}(Q_1), \ t' \in \text{TFP}(t), \ t' \in \text{phrc}(Q_2))
\]

$M$ is said to **transduce** $\text{L}(Q_1)$ to $\text{L}(Q_2)$ and $\text{sen}(Q_1)$ to $\text{sen}(Q_2)$ according to $\text{TFS}$. Next we turn to examining conditions under which a cpst induces a translation as well.
Translation

In this section we state a condition on a cpst, which when satisfied, is sufficient to ensure that the induced transduction is semantic-preserving.

In previous chapters we have used "translation" as it is commonly used in the literature. Aho and Ullman [71] define a translation as a "set of pairs of strings". Benson [74] calls it a "relation from L1 [source language] to L2 [target language]". He then examines "semantic-preserving" translations. We prefer to restrict the use of "translation" to this latter sense; i.e., to those mappings on languages which preserve semantics. (The definition of "semantic-preserving" is of course dependent upon the particular model of language used. We provide a formal definition as D.4.4.) We shall use the more general term transduction to describe the purely syntactic operation of mapping strings to strings. Thus, translation is a special form of transduction. This distinction is maintained throughout the rest of the thesis. Consistent with this view, we shall in what follows, refer to Aho and Ullman's system as a syntax-directed transducer.

Defn D.4.4. A relation $f$ from $sen(Q1)$ to $sen(Q2)$ is a translation or a semantic preserving relation iff $v \in f(w)$ implies $\text{mean}(w) \cap \text{mean}(v) \neq \emptyset$.

Although our orientation is towards programming languages, we are attempting to develop a general model of translation. We therefore permit ambiguous (either syntactically or semantically) sentences even though in programming language design, this is usually not allowed. If each sentence has only one meaning, then clearly $f$ should be called "semantic preserving" only if sentences $w$ and $v$ have exactly the same meaning. However, for ambiguous sentences, there appear to be two reasonable ways to define "semantic-preserving". Transduction $\text{TRANS}: sen(Q1) \rightarrow sen(Q2)$ can be said to be semantic-preserving if either

1) $v \in \text{TRANS}(w) \implies \text{mean}(w) = \text{mean}(v)$; or

2) $v \in \text{TRANS}(w) \implies \text{mean}(w) \cap \text{mean}(v) \neq \emptyset$

Criterion (1) is a very strong constraint, stronger than we care to impose. Following Buttelmann [74], we instead opt for criterion (2) and offer a natural language analogy to
support our choice. In translating from English to German (for example), source and target sentences usually do not have exactly the same set of interpretations. An example of this is the translation from English to German of an idiom which has both its idiomatic and its literal interpretation. Linguists do not in general conclude that it is impossible to translate from English to German. They instead insist only that the German translation could reflect the meaning of its English counterpart. This is precisely what criterion (2) does. Note that a string in isolation may be semantically ambiguous, but in a larger context, this string could become semantically unambiguous.

As a consequence of this definition of translation, we have the following theorem which shows that the composition of two semantic-preserving maps need not be semantic-preserving.

**Theorem T.4.1.** There are two semantic-preserving cpst, $M$ and $M'$, such that the target language of $M$ is the source language of $M'$, but the composition of their induced translations is not semantic-preserving.

**Proof:** Suppose sentence $w \in Q1'$ has meanings $m_1$ and $m_2$ and that $TFS'(w) = w' \in \text{sen}(Q2')$. Further suppose $\text{meaning}(w') = m_2$ and $m_3$. If $TFS(w') = v \in \text{sen}(Q2)$ has meanings $m_3$ and $m_4$, then $\text{mean}(w) \cap \text{mean}(v)$ is empty. Therefore, $v$ is not a translation of $w$.

[$\ast$]

Theorem T.4.1 is true only because the three languages involved are semantically ambiguous.

**Corollary C.4.1.** If $M$ and $M'$ are two semantic-preserving cpst such that the target language of $M$ is the source language of $M'$, and all of the languages are semantically unambiguous, then the composition of their induced translations is semantic-preserving.

**Proof:** The cardinality of the set of meanings of any sentence of a semantically unambiguous language is one. If $w \in TFS'(v)$ and $z \in TFS(w)$ then $\text{mean}(w) \cap \text{mean}(v) \neq \emptyset$ and $\text{mean}(z) \cap \text{mean}(w) \neq \emptyset$. If $u = \text{mean}(v)$ then $u = \text{mean}(w)$ and hence $u = \text{mean}(z)$. Therefore, $v$ and $z$ have a common meaning and the composition of induced translations is semantic-preserving.

[$\ast$]
Suppose $M$ is a cpst, $t \in \text{phr}(G)$ and $t'$ is a transform of $t$. The value of the attribute of the root of $t$ is determined by evaluating the relation of $\text{rule}^{-\star}(\text{attu}(rt(t)), t)$. If we could state a condition on $M$ so that (i) the relation of $\text{rule}^{-\star}(\text{attu}(rt(t)), t)$ is a restriction of the relation of $\text{rule}^{-\star}(\text{attu}(rt(t')), t')$; and (ii) the set of arguments of the first relation is always a subset of the set of arguments of the second relation; then the set of values of $\text{attu}(rt(t))$ would always be a subset of the set of values of $\text{attu}(rt(t'))$. Consequently, $M$ would be semantic-preserving. D.4.5 states such a condition.

**Defn D.4.5.** Let $M$ be a cpst. Suppose $t \in \text{phr}(G)$, $Z$ is a lm of $t$, $t' = TFP(t, Z)$, $c = rt(t)$ or $c \in fr(t)$, and $\text{assoc}(d) = c$. By definition of $\text{ASSOC}$, $\text{att}(\text{label}(c)) = \text{att}(\text{label}(d))$. Let $\text{attu}$ be an exit attribute of both $c$ and $d$. Let $\text{rule1}$ and $\text{rule2}$ either be equal to $\text{rule}^{-\star}(\text{attu}(c), t)$ and $\text{rule}^{-\star}(\text{attu}(d), t')$, respectively, or be equal to $\text{rule}(\text{attu}(c), t)$ and $\text{rule}(\text{attu}(d), t')$, respectively. Suppose

\begin{align*}
\text{rule1}: & \quad \text{attu}(c) = f(\text{attu}(c_1), \ldots, \text{attu}(c_n)):
& \quad Clx_{1} \ldots xCn \rightarrow C \\
\text{rule2}: & \quad \text{attu}(d) = g(\text{attu}(d_1), \ldots, \text{attu}(d_z)):
& \quad Dlx_{1} \ldots xDz \rightarrow D
\end{align*}

$\text{rule1}$ is a **restriction of $\text{rule2}$** ($\text{rule1} \leq \text{rule2}$) if

1) $f: Clx_{1} \ldots xCn \rightarrow C$ is a restriction of $g: Dlx_{1} \ldots xDz \rightarrow D$; and

2) either $\text{assoc}(\text{attu}(d_i))$ is equal to $\text{attu}(c_i)$; or $\text{assoc}(\text{attu}(d_i))$ is undefined.

If $S$ and $U$ are sets of semantic rules or their closure such that there is a bijection from the members of $S$ to the members of $U$ such that $\text{biject}(s) = u$ implies $s$ is a restriction of $u$, then $S$ is a **restriction of $U$** ($S \leq U$).

[*]

If $\text{rule}^{-\star}(t) \leq \text{rule}^{-\star}(t')$ or $\text{rule}(t) \leq \text{rule}(t')$ then there is exactly one attribute occurrence of $t$ associated with any given attribute of a node of $t'$. This comes from the fact that there is a 1-1 onto function from $\text{rule}^{-\star}(t)$ to $\text{rule}^{-\star}(t')$ or from $\text{rule}(t)$ to $\text{rule}(t')$. 
Thm T.4.2. Let $M = (Q_1, Q_2, TABL, ASSOC)$ be a cpst. If for each row of $TABL$, rule $\ast (t) \leq_r rule \ast (t')$, then $M$ induces a translation.

Proof: This theorem is a special case of T.5.1 and hence the proof will not be given here.

[*]

Corollary C.4.2. Let $M = (Q_1, Q_2, TABL, ASSOC)$ be a cpst. If for each row of $TABL$:

\[ t \rightarrow t' \rightarrow x \]

rule$(t) \leq_r rule(t')$, then $M$ induces a translation.

Proof:

\[ rule(t) \leq_r rule(t') \]
\[ \Rightarrow \]
\[ rule \ast (t) \leq_r rule \ast (t') \]

[*]

**Basic Properties**

We will now develop several theorems which help characterize the power of the cpst. The first theorem shows that any cpst can be put into a normal form which is useful in proving later results. For each variant of a phrase-structure transducer, there is an analogous normal form. Before stating the normal form theorem, we will define when two phrase-structure transducers are equivalent.

The string transform, $TFS: sen(Q_1) \rightarrow sen(Q_2)$, induced by a cpst need not be a total onto mapping. The strings in $(sen(Q_2) - range(TFS))$ are not of interest because they are not the transformation of any sentence of $Q_1$, and those strings in $(sen(Q_1) - domain(TFS))$ are not of interest because they are not transduced. We use this observation to define when two cpst are equivalent.

**Defn D.4.6.** Two cpst, $M$ and $M'$, which induce $TFS$ and $TFS'$, respectively, are weakly equivalent iff

1) domain$(TFS) = domain(TFS')$; and

2) $w \in domain(TFS) \Rightarrow TFS(w) = TFS'(w)$; and
3) \( w \in \text{domain(TFS)} \implies \text{mean}(w) \) with respect to \( Q_1 = \text{mean}(w) \) with respect to \( Q_1' \) and

4) \( v \in \text{range(TFS)} \implies \text{mean}(v) \) with respect to \( Q_2 = \text{mean}(v) \) with respect to \( Q_2' \).

If in addition, \( Q_1 = Q_1' \) and \( Q_2 = Q_2' \), then \( M \) and \( M' \) are strongly equivalent.

We now introduce a normal form for cpst. The next theorem shows we can always convert a cpst into a weakly equivalent one whose TABL entries all have unit height.

**Thm T. 4.3.** Let \( M = (Q_1,Q_2,TABL,ASSOC) \) be a cpst. There is a weakly equivalent cpst \( M' = (Q_1',Q_2',TABL',ASSOC') \) such that all trees in \( s-TABL' \) and \( t-TABL' \) have unit height.

**Proof:**

**Construction:**

\( M' \) is constructed from \( M \) as follows:

1) Suppose \( TABL \) has \( z \) rows. \( TABL' \) will also have \( z \) rows. For each \( i \in \mathbb{Z} \), if \( TABL(i) \) is

\[
t \quad t' \quad x
\]

then \( TABL'(i) \) is

\[
\text{mod}(t) \quad \text{mod}(t') \quad x
\]

where \( \text{mod}(t) \) is \( \text{rt}(t) < \text{fr}(t) > \) and \( \text{mod}(t') \) is \( \text{rt}(t') < \text{fr}(t') > \).

2) \( Q_1' = (G,SEM) \) where

a) \( \text{Pr}1' = \text{Pr}1 \cup s-TABL' \).

b) \( \text{Ru}1' = \text{Ru}1 \cup B1 \) where \( |B1| = z \), and if \( t = s-TABL(i) \) then \( B1(i) = \text{mod}(t) \text{ rules} \ast(t) \)

3) \( Q_2' = (G',SEM') \) where

a) \( \text{Pr}2' = \text{Pr}2 \cup t-TABL' \).
b) \( \text{Ru}_2' = \text{Ru}_2 \cdot B_2 \) where \( |B_2| = z \), and if \( t = t - \text{TABL}(i) \), then \( B_2(i) \) is

\[ \text{mod}(t) \text{ rule-} * (t) \]

**End of Construction.**

\( \text{mod}(t) \) and \( \text{mod}(t') \) have unit height independent of the height of \( t \) and \( t' \).

Since \( \text{Ru}_1 \) is a subset of \( \text{Ru}_1' \), and \( \text{Ru}_2 \) is a subset of \( \text{Ru}_2' \), then \( L(Q_1) \) is a subset of \( L(Q_1') \) and \( L(Q_2) \) is a subset of \( L(Q_2') \).

For each \( p \in \text{parse}(w) \) with respect to \( Q_1' \) of which \( \text{mod}(t) \) is a phrase-structure, there is a \( p \in \text{parse}(w) \) with respect to \( Q_1 \) in which \( t \) occurs in place of \( \text{mod}(t) \). By the definition of \( \text{rule-} * (t) \), \( t \) and \( \text{mod}(t) \) have the same set of exit attribute values whenever they are given the same entrance attribute values. The situation is analogous for \( \text{Ru}_2 \) and \( \text{Ru}_2' \). Consequently, \( L(Q_1') \) is a subset of \( L(Q_1) \), and \( L(Q_2') \) is a subset of \( L(Q_2) \). Therefore, \( L(Q_1) = L(Q_1') \) and \( L(Q_2) = L(Q_2') \).

Let \( p \in \text{domain}(\text{TFP}) \) and let \( Z \) be a maximal leftmost partial derivation with respect to \( s-\text{TABL} \). There is a \( q \in \text{domain}(\text{TFP}') \) which has a maximal leftmost partial derivation \( Z' \) such that if \( p_i \) is the \( i \)-th production in a triple of \( Z \), then \( \text{mod}(p_i) \) is the \( i \)-th production in a triple of \( Z' \). Similarly for \( p' = \text{TFP}(p,Z) \).

\[ \text{mean}(p) = \text{mean}(q) \text{ and } \text{mean}(\text{TFP}(p,Z)) = \text{mean}(\text{TFP}'(q,Z')). \]

\( \{ w \mid w = \text{fr}(\text{TFP}(p,Z)), p \in \text{parse}(w) \text{ with respect to } Q_1, Z \text{ is a mlpd of } p \} = \)

\( \{ w \mid w = \text{fr}(\text{TFP}'(q,Z')), q \in \text{parse}(w) \text{ with respect to } Q_1', Z' \text{ is a mlpd of } q \} \)

Therefore, \( M \) is weakly equivalent to \( M' \).

\([*]\)

**Defn d.4.7.** Let \( M \) be a cpst, \( t \in s-\text{TABL} \), and suppose

\[ t \; t' \; x \]

is a row of \( \text{TABL} \).

1) All \( M \) are, by definition, exponential. \( M \) is said to be exponential because T.4.4 shows that the
pseudo-weight of \( TFP(t, Z) \) is bounded by an exponential function of the pseudo-weight of \( t \).

2) \( M \) is **multi-linear** if \( x_i = x_j > 0 \implies \) either \( i = j \) or \( i \neq j \) and \( t \) is not a descendent phrase of \( \text{fr}(t, x_i) \) in any tree of \( \text{phr}(Prl) \). \( M \) is said to be multi-linear because T.4.5 shows that the pseudo-weight of \( TFP(t, Z) \) is bounded by an integer constant times the pseudo-weight of \( t \).

3) \( M \) is **linear** if \( x_i = x_j > 0 \implies i = j \). \( M \) is said to be linear because T.4.5 shows that the pseudo-weight of \( TFP(t, Z) \) is bounded by the pseudo-weight of \( t \).

4) \( M \) is **strong** if \( \text{fr}(t, i) \in V_{nl} \implies \) there is some \( x_j = i \). \( M \) is said to be strong because each node of \( t \) which is labeled by a nonterminal is associated with at least one node of \( TFP(t, Z) \).

**Thm T.4.4.** For any \( M \) there is an integer constant \( k \) such that if \( t \in \text{domain}(TFP) \), \( Z \) a leftmost partial derivation of \( t \), and \( t' = TFP(t, Z) \), then

\[
k^**\text{pwt}(t) \geq \text{pwt}(t')
\]

**Proof:** Let \( k \) be the maximum of \( 2 \) and the greatest number of equal numbers greater than zero in the \( x \)-vector of any row of TABL. We assume without loss of generality that \( M \) is in normal form.

Suppose \( r_0 [r_1 \ldots r_n] \) is a box bracketing of \( t; Z \), equal to

\[(p_1, n_1, m_1), \ldots, (p_z, n_z, m_z)\]

is a leftmost derivation of \( r_0 \) with respect to \( s\text{-TABL} \), and \( t' = TFP(t, Z) \). The proof is by induction on the weight of \( r_0 \).

**Base Step:** \( \text{wt}(r_0) = 1 \)

There is a row of TABL

\[
q \quad q' \quad x
\]

where \( q = p_1 \). Then \( t' = q' \), and the weight of \( t' \) is 1 which is less than \( k \). The number of nodes on the
frontier of $t'$ which are associated with any single node of $t$ is less than or equal to $k^{\ast\ast}h(t)$. The weight of $t'$ is less than or equal to

$$1 + k + k^{\ast\ast}2 + \ldots + k^{\ast\ast}(w(t) - 1)$$

**Induction Step:** Let $w(t) = n + 1$

Assume that the theorem is true for $r_0$ with weight less than or equal to $n$. The greater the distance between the root of $r_0$ and a node, $b$, of $r_0$, the greater the number of nodes in $t'$ which can be associated with $b$. At each step in the construction of $t'$, if $c \in fr(t')$, assoc$(b) = c$, and $b$ is not on the frontier of $r_0$, then a $t$-ABL entry is composed to $t'$ at node $c$. Therefore, the weight of the transform of $t$ will be greatest if the frontier of each $p_i$ in $\mathbb{Z}$ has either zero or one nodes labeled by nonterminals in it. In that case, if $q_0 f_l \ldots f_i q f(i+2) \ldots f_n$ is a box bracketing of $r_0$, where $f_y$ belongs to $V_t$ for $y \in \mathbb{Z}$, then $w(t_q)$ is equal to $h(t_q)$, and $w(t_{ql})$ is equal to $h(t_{ql})$.

Consider the particular box bracketing, $s_0 s l$, of $r_0$, where $w(t_{s0}) = n$, and $w(t_{sl}) = l$.

By hypothesis, $s_0$ satisfies the theorem. Therefore, the weight of $s'0$, the sub-structure of $t'$ induced by $(s_0, \mathbb{Z})$, is less than or equal to

$$1 + k + k^{\ast\ast}2 + \ldots + k^{\ast\ast}(w(t_{s0}) - 1)$$

which is equal to

$$1 + k + k^{\ast\ast}2 + \ldots + k^{\ast\ast}(n - 1)$$

which is less than or equal to $k^{\ast\ast}n$. The number of nodes on $fr(s_{s0})$ which are associated with a single node of $fr(s_0)$ is less than or equal to $k^{\ast\ast}h(s_0)$ which is $k^{\ast\ast}n$. If $r_{s0}$ is the sub-structure of $t'$ induced by $(r_0, \mathbb{Z})$, then

$$w(t_{s0}) = w(t_{s'}) + k^{\ast\ast}n = k^{\ast\ast}n + k^{\ast\ast}n$$

which is less than or equal to $k^{\ast\ast}(n + 1) = k^{\ast\ast}w(t_{s0}) \leq k^{\ast\ast}w(t)$

For each node on the frontier of $s_0$ associated with the single frontier node of $s_0$ which is labeled by a nonterminal, no more than $k$ frontier nodes of $s_{s}'$ can be associated with the single frontier node of $s_{s}'$ which is labeled by a nonterminal. Consequently, the
number of nodes on the frontier of $r_0$ which can be associated with a single node of $t$ is less than or equal to $k^*k^{**ht(s)}$, which is $k^{**ht(r_0)}$.

[⋆]

Thm T.4.5. For any multi-linear $M$, there are integer constants $k$ and $z$ such that if $t \in \text{domain}(\text{TFP})$, $Z$ is a leftmost partial derivation of $t$, and $t' = \text{TFP}(t, Z)$, then

$$(k^{**z})pwt(t) \geq pwt(t')$$

Furthermore, if $M$ is linear, then

$$pwt(t) \geq pwt(t')$$

Proof: Let $k$ be the greatest number of equal numbers greater than 0 in the x-vector of any row of $\text{TABL}$. $z$ is the number of rows in $\text{TABL}$. We will assume without loss of generality that $M$ is in normal form.

Suppose $r_0[l..r_n]$ is a box bracketing of $t$: $Z$, equal to

$$(p_1,n_1,m_1), \ldots, (p_z,n_z,m_z)$$

is a leftmost derivation of $r_0$ with respect to $s-\text{TABL}$, and $t' = \text{TFP}(t, Z)$. The proof is by induction on the weight of $r_0$.

Base Step: $wt(r_0) = 1$

There is a row of $\text{TABL}$

$$q \cdot q' \cdot x$$

where $q = p_1$. Then $t' = q'$ and the weight of $t'$ is 1 which is less than or equal to $(k^{**z})w(t)$. The number of nodes on the frontier of $t'$ which can be associated with any single node of $t$ is less than or equal to $k^{**z}$.

Induction Step: $wt(r_0) = j+1$

Assume that the theorem is true for $r_0$ with weight less than or equal to $j$. The same reasoning which we used in the proof of T.4.4 to conclude that if each sub-structure of $t$ has a single node labeled by a non-terminal, then $t'$ will have the maximum weight, can be applied here.
Consider the particular box bracketing, \( s_0(s_1) \), of \( r_0 \), where \( \text{wt}(s_0) = j \), and \( \text{wt}(s_1) = 1 \).

By hypothesis, \( s_0 \) satisfies the theorem. Therefore,

\[
\text{wt}(s_0') \leq (k^{**2}) \cdot \text{wt}(s_0)
\]

where \( s_0' \) is the sub-structure of \( t' \) induced by \((s_0, Z)\). Furthermore, the number of nodes on \( \text{fr}(s_0') \) which are associated with any node of \( s_0 \) must be less than or equal to \( k^{**2} \). Therefore, the number of nodes which can be associated with the root of \( s_1 \) is less than or equal to \( k^{**2} \), and if \( r_0 \) is the sub-structure of \( t' \) induced by \((r_0, Z)\) then

\[
\text{wt}(r_0') \leq (k^{**2}) \cdot \text{wt}(s_0) + k^{**2} \]

\[
(k^{**2}) \cdot (\text{wt}(s_0) + 1) = (k^{**2}) \cdot (\text{wt}(r_0))
\]

Suppose that the number of nodes which can be associated with a single node of \( t \) is greater than \( k^{**2} \). Then there is a path of length greater than \( z \) such that at least two equal trees are the \( s\)-TABLE entries of rows of TABL whose \( x \)-vector has at least two distinct components \( x_i \) and \( x_j \) which are both greater than zero and equal. But this violates the definition of what it means for a cpst to be multi-linear. Therefore, the number of nodes which can be associated with any node of \( t \) is less than or equal to \( k^{**2} \).

For the case where \( M \) is a linear cpst, note that \( k \) is always one. Therefore, \( k^{**2} \) is one. This leads to the immediate conclusion that

\[
\text{wt}(t') \leq \text{wt}(t)
\]

\([*]\)

**Thm 4.9.** For strong \( M \), if \( t \in \text{domain}(TFP) \), \( Z \) is a leftmost partial derivation of \( t \), \( t' = TFP(t, Z) \), and \( t' \in \text{phrc}(Pr2) \), then

\[
\text{pwt}(t) \leq \text{pwt}(t')
\]

**Proof:** Suppose \( Z \) is a lmd of \( r_0 \), a sub-structure of \( t \). Each node on the frontier of a sub-structure of \( r_0 \) is associated with a node in \( t' \) because \( M \) is strong. Therefore, if a sub-structure, \( q' \), of \( r_0 \) induces a terminated sub-structure \( q \) of \( t' \), then \( q \), itself, must be terminated. Therefore, \( r_0 \) must be a terminated phrase-structure. Since the root of \( t_0 \) is also the
root of t, then r0 must be all of t. Hence, pwt(t') must be greater than or equal to pwt(t).

[*]

The next two results are "pumping theorems" on cpst in much the same spirit as the "uvwxy" theorem for cfl [BarPS61]. This permits us to prove certain transformations may not be induced by any linear or multi-linear cpst.

Lemma L.4.1. Let M be a linear cpst such that TFS induced by M is a 1-1 function. Then there are constants m and n depending only on TFS such that if v \in TFS(w), |w| > m and |vl > m then

1) w may be written as $\alpha \alpha \gamma \delta$, where $|\gamma| < n$ and $\gamma$ and $\delta$ are not both the empty string.

2) v may be written as $\pi \gamma \gamma \gamma \pi$, where $|\gamma| < n$ and $\gamma$ and $\delta$ are not both the empty string.

3) For all $i > 0$, $(\alpha \alpha \gamma \delta \pi \gamma \gamma \gamma \pi) \in TFS$.

Proof: Assume without loss of generality that M is in normal form. Let $\text{maxw} = \text{maximum}(|f(t)| \mid t \in \text{TABL} \cup \text{t-TABL})$. Let z be the number of rows in TABL. Then

$$m = \text{maxw} \quad \text{and} \quad n = \text{maxw}$$

For $t \in \text{phrc}(Q1)$ or $\text{phrc}(Q2)$, if $ht(t) \leq j$ then

$$|f(t)| \leq \text{maxw}$$

Let v \in TFS(w) such that $|w| > m$, $|vl > m$, Z is a 1md of t \in parse(w) with respect to Pr1, Y is a mmpd of t with respect to s-TABL, and $Y'$ is the 1md of $t' = TFP(t', Y)$ induced by $(t, Y)$. Since M is in normal form, both t and $t'$ contain a path of length > z. Therefore, there is some box bracketing $t\theta[t1..tk]$ of t such that for some $j \in k$, $tj$ has a box bracketing

$$p\theta[p1..p\alpha[q1..px[r1..rb]..qc]..pd]$$

(4.2)

where

1) $p\theta = px$. 

$\pi \gamma \gamma \gamma \pi$
2) $p_0$, $p_a$, and $p_x$ are all phrase-structures of $Y$.
3) $b > 1$ (or else $|w|_1$ would be $< m$).
4) $|r_l .. r_b| < n$.

Then there is a box bracketing $t_0'[t_1'...t'(k')]$ of $t'$ such that for some $j$, $t_j$ has a box bracketing

$$p'0[p'1..p'a'[q'1..p'x'[r'1..r'b']..q'c']..p'd']$$  \(4.3\)

where

5) $p'0 = p'x'$.
6) $p'0$, $p'a'$, and $p'x'$ are induced by $p_0$, $p_a$, and $p_x$, respectively.
7) $b' > 1$ (or else TFS would not be $1$-1).
8) $|r'1..r'b'| < n$.

Suppose $fr(p_0) = f_1..fd$, $fr(p'0) = f'_1..fd'$, $fr(p_a) = g_1..gc$, and $fr(p'a') = g'_1..gc'$. Then $P' =

$$p_0[f_1..p_a[g_1..p_x..g_c]..fd]$$

is in phr(s-TABL) and is a sub-structure of $t$. $P' =

$$p'0[f'_1..p'a'[g'_1..p'x'..g'_c]..fd']$$

is in phr(t-TABL) and is a sub-structure of $t'$. Suppose that $rt(px)$ is the $j$-th frontier symbol of $p_0[f_1..p_a[g_1..rt(px)..g_c]..fd]$, and the $k$-th frontier symbol of $p_a$, while $rt(p'x')$ is the $j'$-th frontier symbol of $p'0[f'_1..p'a'[g'_1..rt(p'x')..g'_c]..fd']$, and the $k'$-th frontier symbol of $p'a'$.

For $i > 0$, the phrase-transform of tree 4.4 with respect to mlmpd $Y$ is tree 4.5.

$$\text{compose}(*,P,j+i*k)\#..\#$$  \(4.4\)

$$\text{compose}(*,P,j+k)\#$$

$$\text{compose}(*,P,j)(P)$$

$$\text{compose}(*,P',j'+i*k')\#..\#$$  \(4.5\)

$$\text{compose}(*,P',j'+k')\#$$
compose(*,P',j')(P')

Substituting tree 4.4 for px in t and tree 4.5 for p'x' in t' gives the desired result.

[∗]

**Thm T.4.7.** Let M be a multi-linear cpst such that TFS induced by M is a 1-1 function. Then there are integer constants x, m and n depending only on TFS such that if v ∈ TFS(w), |w| ≥ m and |v| ≥ x*m then

1) w may be written as ϕψvβ, where |ψvβ| < n and ϕ and β are not both the empty string.

2) v may be written as π(ψ)ψrzeγ(γ) for some integer y ≤ x where |π(ψ)ψrzeγ(γ)| < n and ψ and γ are not both the empty string.

3) For all i > 0,

\[\{\phi^*\psi^*\gamma^*\} \cup \{\pi(\psi)\psi^*\gamma^*\} \cup \{\pi(\gamma)\psi^*\gamma^*\}\]

belongs to TFS.

**Proof:** Assume without loss of generality that M is in normal form. Let mxwd be the maximum of the lengths of the frontiers of the trees in s-TABL t-TABL. Let z be the number of rows in TABL. Then x is k**z where k is the maximum number of positive equal x-vector components for any x-vector in TABL, and

\[m = mxwd**z \quad n = mxwd**z+1\]

If t ∈ phrc(Pri) or t ∈ phrc(Pr2) and ht(t) ≤ j then

\[|fr(t)| ≤ mxwd**j\]

Let v ∈ TFS(w), |w| ≥ m, |v| ≥ m, Z be a lmd of t ∈ parse(w) with respect to Pri, Y be a lmpd of t with respect to s-TABL, and Y' be the lmd of t' = TFP(t,Y). Since M is in normal form, then both t and t' contain a path of length > z. The situation here is analogous to that of the linear cpst in L.4.1. The critical difference is that several nodes in t' may be associated with a single node in t. In T.4.5 the maximum number of such nodes was shown to be x. If nodes c'/1,...,c'/k' in t' are associated with a single node c in t, then each of the terminated sub-structures of t' rooted by c'/1,...,c'/k' are equal. Call these sub-structures, p'/1,...,p'/k', and let p denote the
sub-structure of $t$ rooted by $c$ which induces each of the $p'$. Suppose $p$ is the recursive sub-structure of $t$ isolated as tree 4.2 in the proof of L.4.1. Then no two or more nodes on the frontier of any $p'$ can be associated with a single node on the frontier of $p$. Otherwise, since $p$ is a recursive sub-structure and each $p'$ is a recursive sub-structure, then $M$ would not be multi-linear, contrary to hypothesis. Trees 4.4 and 4.5 can be substituted for trees 4.2 and 4.3 in exactly the same manner as we did in L.4.1.

As a consequence of this theorem, we can show that certain transductions cannot be induced by either a multi-linear or a linear cpst. The next example shows that no linear cpst can induce a translation from binary to decimal numbers.

**Example E.4.3** There is no linear cpst which induces a translation from base 2 numbers to base 10 numbers.

**Proof:** Let $Q_1$ and $Q_2$ be any complete pss whose languages are the binary and decimal numbers, respectively. Suppose there is a cpst $M$ which induces a translation from $sen(Q_1)$ onto $sen(Q_2)$. Clearly, $TFS(M)$ is a 1-1 onto function. Then there are constants $m$ and $n$ which satisfy L.4.1. Let $w = w(1) \ldots w(y)$ be a binary number such that $y > m$. The meaning of $w$ is

$$mn(w) = w(y) + 2 \times w(y-1) + 2 \times w(y-2)$$

$$+ \ldots + 2 \times w(1)$$

suppose the translation of $w$ is $v = v(1) \ldots v(z)$, a decimal number such that $z > m$. The meaning of $v$ is

$$mn(v) = v(z) + 10 \times v(z-1) + 10 \times v(z-2)$$

$$+ \ldots + 10 \times v(1)$$

Since $v$ is the translation of $w$, $mn(w) = mn(v)$. Hence,

$$w(y) + \ldots + 2 \times w(1) = v(z) + \ldots + 10 \times v(1)$$
By the pumping theorem, \( w \) and \( v \) can be broken into pieces such that \( w = abcde \), \( v = klmno \) and

\[
(a b c d e, k 1 m n o)
\]
is in TFS for each \( i \geq 0 \). Corresponding to the division of \( w \) and \( v \) we have \( mn(w) \) and \( mn(v) \) divided into \( wa \), \( wb \), \( wc \), \( wd \), \( we \) and \( vk \), \( vl \), \( vm \), \( vn \), \( vo \), respectively.

Then \( mn-w-i = mn(a b c d e) = \)

\[
\begin{align*}
&|I|d| \\
&we + wd + wd*2 \\
&(i-1)|d| \\
&+ .. + wd*2 \\
&(i-1)|d| + wc*2 + wb*2 \\
&(i-1)|d| + wb*2 \\
&(i-1)|bd| + wa*2
\end{align*}
\]

and \( mn-v-i = mn(k l m n o) = \)

\[
\begin{align*}
&|I|n| \\
&vo + vn + vn*10 \\
&(i-1)|n| \\
&+ .. + vn*10 \\
&(i-1)|n| + vm*10 + vl*10 \\
&(i-1)|n| + vl*10 \\
&(i-1)|n| + vl*10 \\
&(i-1)|n| \\
&+ .. + vl*10 \\
&(i-1)|n| \\
&+ vk*10
\end{align*}
\]
If \( i = x+1 \) then \( mn-w-i = \)

\[
\begin{align*}
    mn-(w) + \\
    ldlx + ld + lbdlx \\
    wd*2 + wc*2 + wb*2 \\
    lbd + wa*2 - wa -wc
\end{align*}
\]

and \( mn-v-i = \)

\[
\begin{align*}
    mn(v) + \\
    lnx + ln + lnx \\
    vn*10 + vm*10 + vl*10 \\
    ln + vk*10 - vk - vm
\end{align*}
\]

We can select \( v \) so that all of its digits are non-zero. Then \( v_k, \ldots, v_0 \) are all non-zero. For any particular \( w \) and \( v \), the numbers \( wa, wc, vm \) and \( vk \) are constant. As \( i \) grows \( (vm + vn)*(n**lni) \) becomes much larger than \( (wc + wd)*(2**ldli) \) and \( (vk + vl)*(n**lni) \) becomes much larger than \( (wa + wb)*(2**lbdli) \). Hence, as \( i \) grows, \( mn(a b**i c d**i e) \) becomes much larger than \( mn(k l**i m n**i o) \).

Therefore, no linear cpst exists which can translate from binary to decimal.

[\( * \)]

We commented earlier that the syntax-directed transduction schema as defined by Aho and Ullman is a restriction of the cpst. Note that their transduction schema as described earlier in the chapter is a form of strong linear cpst. The final theorem of this chapter demonstrates that these restrictions constrain the class of language pairs between which the syntax-directed transduction schema can translate.

**Thm T.4.8.** The set of language pairs translatable using a syntax-directed transduction schema as defined by Aho and Ullman [69] is strictly less than the set of language pairs translatable using a complete phrase-structure transducer.

**Proof:** The syntax-directed transduction schema of Aho and Ullman is a special form of cpst. Furthermore, Aho
and Ullman showed that their system cannot induce a translation from \( \{ (n,2^n) \mid n \in \text{INT} \} \) to \( \{ (n,n) \mid n \in \text{INT} \} \). This translation is induced by the cpst in Appendix B.

[\star]

**Complexity**

In the earlier sections of this chapter, we studied the cpst as a way in which to define a relation between two languages. In this section we analyze the cpst as a procedure whose execution enumerates the set of translations of input string \( w \). Since the number of translations of a source sentence is not in general bounded or even necessarily finite, we will focus our attention on the amount of execution time required to output a single translation. We analyze the execution time of a subclass of the complete phrase-structure translators. The particular subclass we study are called **honest translators**. A cpst is said to be honest if for each \( t \in \text{phrc}(Q_1) \) and \( t' \in \text{TFP}(t) \),

1) \( t' \in \text{phrc}(Q_2) \); and

2) \( \text{mean}(t) \not\subseteq \text{mean}(t') \neq \emptyset \).

Recall that a transducer is semantic-preserving if for each \( w \in \text{sen}(Q_1) \), \( v \in \text{TFS}(w) \)

\[
\text{mean}(w) \not\subseteq \text{mean}(v) \neq \emptyset
\]

Therefore, honesty implies semantic-preservation. We motivate the use of the term "honesty" by the following example. Suppose \( M \) is a cpst which induces a mapping on the single sentence \( w \) of \( Q_1 \) which has two meanings \( u_1 \) and \( u_2 \), and three parses \( t_1, t_2, \) and \( t_3 \); that for each \( 1 \leq i \leq 3 \), \( \text{TFP}(t_i) = t'_i \in \text{phr}(Q_2) \); that \( \text{fr}(t_1) = \text{fr}(t_2) = v \in \text{sen}(Q_2) \); and that \( t_3 \) is not complete. Further suppose that

- \( \text{mean}(t_1) = u_1 \)
- \( \text{mean}(t'_1) = u_2 \)
- \( \text{mean}(t_2) = u_2 \)
- \( \text{mean}(t'_2) = \emptyset \)
- \( \text{mean}(t_3) = \emptyset \)
- \( \text{mean}(t'_3) \) is undefined

Then \( M \) is semantic-preserving since \( \text{mean}(w) \not\subseteq \text{mean}(v) \) is \( u_2 \). However, by examining the transformation of each parse of \( w \) individually, it is not at all apparent that \( M \) is semantic-preserving. No parse and its transformation share a meaning. We describe \( M \) as being "honest" just in case it is possible to tell from examining the transformations individually whether or not \( M \) is a translator.
Before analyzing the time required for an honest translator to execute, we first note that honest translators are algorithmic.

**Theorem 4.2.** If $M$ is an honest complete phrase-structure translator, then $M$ is algorithmic.

**Proof:** If $M$ is honest, then any non-deviant parse of the source sentence must induce a non-deviant parse of a target sentence. All parses are non-deviant and effectively computable.

We next determine the time required to produce one translation of a sentence using an honest cost. Let $M$ be an honest cost. Let $w \in \text{sen}(Q)$ and for convenience assume $M$ is in normal form.

The parses of $w$ may be obtained in a "factored form" (see Earley [68, 70]) in

$$O(n^{**3})$$  \hspace{1cm} (4.6)

time steps where $n = |w|$. Given the set of parses in this form, it takes

$$O(wt(t))$$  \hspace{1cm} (4.7)

time to construct parse tree $t$. Since all parses are valid and since $M$ is honest, any parse of $w$ input to the transformer will result in the generation of a parse of a translation of $w$. We choose $t$ so that it has no linear recursive sub-structures. In that case, $wt(t) < kn$ for some integer $k$, which is constant for $M$. Then $t$ may be obtained in

$$O(n^{**3} + n)$$  \hspace{1cm} (4.8)

time.

Let $t' \in \text{TFP}(t)$. Count as one unit, the time necessary to compose a single $t$-TABL entry onto $t'$ in the execution of A.4.1 which builds $t'$. Since $M$ is in normal form, $\text{pwt}(p) = wt(p)$ for any $p \in \text{phr}(s$-TABL) or $p \in \text{phr}(t$-TABL). By T.4.4, if $M$ is exponential, then the maximum weight of $t'$ is $k^{**wt(t)}$ for constant $k$. If $M$ is linear then the maximum weight of $t'$ is $wt(t)$. If $M$ is multi-linear, then there is an integer constant $k$ such that $wt(t') \leq k*wt(t)$. The time necessary to compute $fr(t')$ is proportional to the weight of
t'. This gives a total translation time of

\[ O(n^{*3} + n + k^{*n} + k^{*n}) \]  \hspace{1cm} (4.9)

for exponential \( M \) and

\[ O(n^{*3} + n) \]  \hspace{1cm} (4.10)

for linear or multi-linear \( M \). If the source language is syntactically unambiguous, then there is only one parse of each source sentence. In this case the time necessary to construct the single parse tree of \( w \) reduces to \( O(n) \). Consequently, times 4.9 and 4.10 reduce to 4.11 and 4.12, respectively.

\[ O(k^{*n}) \]  \hspace{1cm} (4.11)

\[ O(n^{*3}) \]  \hspace{1cm} (4.12)

For an honest linear or multi-linear cpst whose source language is syntactically unambiguous, the order of time needed to translate \( w \) is of the same order as the time required to parse \( w \).

**Limitations of CPST**

The reader should note three very important properties of the cpst which help to characterize it as a model of translation:

1) The source and target systems are all complete phrase-structure systems. Hence, any language whose sentences are not a context-free language is not translatable. Of course, most interesting programming languages have context-sensitive features such as the common restriction to one occurrence of the same label in a program.

2) At no time in the translation process is semantic information ever used. The translation process is entirely syntax-driven.

3) The table always has one source entry and one target entry per row. The phrase-structures are transduced on a one for one basis. It is never the case that several source phrase-structures are transduced simultaneously into several target phrase-structures.
These facts about complete translators play a strong role in limiting the class of language pairs and system pairs between which they can translate. There are many interesting translations that cannot be induced using cpst. For example, no complete phrase-structure system can define that subset of FORTAN which has assignment statements; integer and real variables and constants; and declaration statements which indicate the data-type of the variables used in a program. This mini-language is not context-free. Hence, it cannot be either the source or target language of any complete translator. In the next three chapters these restrictions are relaxed. The basic model of phrase-structure translation is modified three times to produce more "powerful" translators. This leads to the development of a family of translators capable of inducing the partial translation between any two phrase-structure languages.

Summary

Some of the notations and formalisms necessary to examine formal properties of translation were developed. A specific transducer called the complete phrase-structure transducer was defined and analyzed. It closely resembles the traditional syntax-directed transduction schema in operational philosophy, being entirely syntax-directed. We stated two sufficient conditions for a complete transducer to be semantic preserving, and analyzed the computational complexity of a subclass of the complete translators. Various subclasses of complete transducers were defined, namely the "exponential", "multi-linear", "linear" and "strong" transducers. Transducers fall into one of several categories depending upon what the induced relationship is between the weight of the source parse and the weight of the target parse. A "pumping" theorem similar in spirit to the "uvwxy" theorem for context-free languages was proven. Using it we showed that there is no transducer of a certain type which can induce a translation from binary to decimal numbers. We then showed that the transduction schema defined by Aho and Ullman is a restriction of the complete transducer, being able to induce only a proper subset of the translations which the complete translator can.
Chapter V - Incomplete Phrase-Structure Translators

The meaning of Stonehenge in Tralfamadorian, when viewed from above is: "Replacement part being rushed with all possible speed".

Kurt Vonnegut, Jr.

Introduction

In this chapter we modify the complete phrase-structure transducer given in chapter iv so that the source and target languages may be defined by general phrase-structure systems. This change is quite significant because we prove in T.5.4 that this modified version of a phrase-structure transducer is capable of inducing the partial translation defined between any two phrase-structure languages. This is in marked contrast to the rather limited set of language pairs between which complete translators can translate.

Corollary C.3.1 states that any general phrase-structure system may be algorithmically converted into an equivalent system which uses only synthesized attributes. Because of this, the fact that we will now examine transducers whose source and target systems use inherited attributes may not at first seem significant. However, the method given for doing this conversion in C.3.1 in effect transfers all of the real semantic processing onto the topmost phrase-structure of any parse. All other sub-structures in a complete tree have a rather trivial semantics which does not reflect the original assignment of attribute values at all. This is very unsatisfying in terms of language definition since we would like to have an indication of the intended meaning of a sub-structure visible "locally" - at the sub-structure whose "meaning" is being defined. Furthermore, in terms of the method of conversion used in C.3.1, reducing the seemingly more complex gen-pss to an equivalent syn-pss is actually of no help in proving that a transducer is semantic-preserving. In order to verify that a phrase-structure transducer induces a translation, it would be necessary to tell whether the "super" relation, "f", associated with the top
INCOMPLETE PHRASE-STRUCTURE TRANSLATORS
INTRODUCTION

sub-structure of the source tree is a relational restriction of the "super" relation, "f'" , associated with the top sub-structure of the target tree. This task is no less difficult than deciding whether or not the semantic rules of the original version of the source system are restrictions of the corresponding semantic rules of the original target system. Since there are many instances in which the use of inherited attributes is the most direct way to define the languages under consideration, we should have techniques for verifying that a transducer using gen-pss induces a translation. In this chapter, we state conditions which are sufficient to guarantee the correctness of a phrase-structure translator whose arguments are general phrase-structure systems. Furthermore, we investigate the class of language pairs translatable using incomplete translators. We also study under what conditions a translator may be "inverted" so that instead of translating from language A to B, it translates from language B to A.

Basic Definitions

Defn D.5.1. An incomplete phrase-structure transducer (ipst) is a 4-tuple $M = (Q_1, Q_2, TABL, ASSOC)$ where:

1) $TABL$ and $ASSOC$ are as in a cpst.

2) $Q_1$, the source system is a general pss.

3) $Q_2$, the target system is a general pss.

[*]

The phrase-transform of an ipst is defined in the same way as is the phrase-transform of a cpst. However, in the ipst, the incomplete source and target systems are used to filter out deviant source and target parses that enter and leave the tree transformer.

Defn D.5.2. Let $M$ be an ipst, and $p$ be in $\text{phr}(Q_1)$. The phrase-transform of $p$, $\text{TFP}(p)$, is computed by A.4.1.

[*]

Defn D.5.3. Let $M$ be an ipst. If $w \in \text{sen}(Q_1)$, then the string-transform, $\text{TFS}\text{sen}(Q_1) \rightarrow \text{sen}(Q_2)$, is a relation defined by $\text{TFS}(w) = \ldots$
Buttelmann investigated the notion of deviance which plays so important a role in this thesis. In his work on formal semantics, Buttelmann devised a model of language in which a partial recursive function and a set of arguments for that function are associated with each parse of a sentence. The meaning of the sentence is the value of the function evaluated on its associated arguments whenever the function is defined. A string must have at least one meaning to be a sentence of the language even if it has one or more parses. Despite the similarity between the concept of deviance defined by Buttelmann and that given here, he did not make use of that notion in the same way in his transduction mechanism as we do here. Buttelmann did not filter parses that entered or left his transformer. The difference is shown more clearly in Figure 5.1.

Figure 5.1. Transducer With Deviance Filters

Suppose a sentence is syntactically ambiguous. If a deviant parse of a sentence were input to Buttelmann's transduction mechanism, his tree transformer would produce one or more target trees from it. Each output tree might or might not have a non-deviant evaluation. However, if the ipst were used instead, then a tree which had only deviant evaluations would not be transformed. The greater significance of these differing approaches will be made more explicit in chapter x.

There is a corresponding normal form for ipst in which all trees in TABL have unit height. The proof is essentially the same as that given for cpst in T.4.3 and so
is omitted here.

Translation

Theorem T.5.1 will state a condition which is sufficient to ensure that a transduction is semantic-preserving.

**Lemma L.5.1.** Suppose \( f: B_1 \times \ldots \times B_n \to B, \) \( f': C_1 \times \ldots \times C_n \to C, \) \( g: D_1 \times \ldots \times D_m \to D, \) and \( g': E_1 \times \ldots \times E_m \to E \) are four relations such that \( f \) is a restriction of \( f' \) and \( g \) is a restriction of \( g' \). For some \( i \in n, \) if \( E \) is a subset of \( C_i, \) then

\[
f(b_1, \ldots, b_{i-1}, g(d_1, \ldots, d_m), b_{i+1}, \ldots, b_n) = B_1 \times \ldots \times B_{i-1} \times D_1 \times \ldots \times D_m \times B_{i+1} \times \ldots \times B_n \to B
\]

is a restriction of

\[
f'(c_1, \ldots, c_{i-1}, g'(e_1, \ldots, e_m), \ldots, c_n); \quad C_1 \times \ldots \times C_{i-1} \times E_1 \times \ldots \times E_m \times C_{i+1} \times \ldots \times C_n \to C
\]

**Proof:** Suppose \( (d_1, \ldots, d_m) \) belongs to the domain of \( g. \) Then \( (d_1, \ldots, d_m) \) also belongs to the domain of \( g' \) and by definition of "restriction", \( g(d_1, \ldots, d_m) \) is a subset of \( g'(d_1, \ldots, d_m). \) Suppose \( d \) is any member of \( g(d_1, \ldots, d_m), \) and \( (b_1, \ldots, b_n) \) is any member of \( \text{domain}(f) \) where the \( i \)-th argument is \( d. \) Since \( f \) is a restriction of \( f' \), then \( (b_1, \ldots, b_n) \) is also in \( \text{domain}(f') \), and \( f(b_1, \ldots, d, \ldots, b_n) \) is a subset of \( f'(b_1, \ldots, d, \ldots, b_n). \)

[**]  

**Lemma L.5.2.** If \( \text{rule-*(t)} \preceq_r \text{rule-*(t')}, \) \( \text{rule-*(q)} \preceq_r \text{rule-*(q')}, \) \( \text{label}(rt(q)) = \text{label}(fr(t, z)), \) and \( \text{assoc-inv}(fr(t, z)) = fr(t', z') \) then

\[
\text{rule-*(compose(t, q, i))} \preceq_r \text{rule-*(compose(t', q', j))}
\]

**Proof:** Let \( r = \text{compose}(t, q, i) \) and \( r' = \text{compose}(t', q', j). \) Suppose \( \text{attu}(c) \) is an exit attribute of \( r, \) and \( \text{attu}(c') \) is the associated exit attribute of \( r'. \) The case where \( c \in \text{nodes}(t) \) is symmetric to the case where \( c \in \text{nodes}(q), \) and so a proof of just the former is given below.

Suppose \( c \in \text{nodes}(t). \) Let us say that, in \( t, \) \( \text{attu}(c) \) is **ultimately** dependent upon the second components of \( \text{rule-*(attu}(c), t). \) Similarly, we say
that, in \( t' \), \( \text{attu}(c') \) is ultimately dependent upon the second components of rule-\( \ast \)(\( \text{attu}(c'), t' \)). If \( \text{attu}(c) \) is ultimately dependent upon \( \text{attu}(ci) \), we write this as \( \text{attu}(c) \leq \text{attu}(ci) \).

If \( \text{attu}(c) \) is not ultimately dependent upon some attribute of \( \text{fr}(t,i) \), then rule-\( \ast \)(\( \text{attu}(c), r \)) is equal to rule-\( \ast \)(\( \text{attu}(c), t \)). Similarly for \( \text{attu}(c') \).

Let \( d = \text{fr}(t,i) \) and \( d' = \text{fr}(t', j) \). These are the points on the frontiers of \( t \) and \( t' \), respectively, where \( q \) is composed to \( t \) and \( q' \) is composed to \( t' \). Assume that, in \( t \), \( \text{attu}(c) \leq \text{attv}(d) \). Then, in \( t' \), \( \text{attu}(c') \leq \text{attv}(d') \). \( \text{attv}(d) \) is an exit attribute of \( q \), and \( \text{attv}(d') \) is an exit attribute of \( q' \). Consequently, rule-\( \ast \)(\( \text{attv}(d), q \)) \( \preceq \) rule-\( \ast \)(\( \text{attv}(d'), q' \)).

Suppose that \( \text{attv}(d) \leq \{ \text{attwl}(b_1), \ldots, \text{attwy}(b_y) \} \) in \( q \). Then \( \text{attv}(d') \leq \{ \text{attwl}(b'_1), \ldots, \text{attwy}(b'_y) \} \) in \( q' \). From this it follows that, in \( r \) and \( r' \) respectively, \( \text{attu}(c) \leq \text{attwl}(b_i) \) and \( \text{attu}(c') \leq \text{attwl}(b'_i) \). If no \( b_i \) is \( d \), then from L.5.1, we conclude that rule-\( \ast \)(\( \text{attu}(c), r \)) \( \preceq \) rule-\( \ast \)(\( \text{attu}(c'), r' \)). If some \( b_i \) is \( d \), then \( \text{attwl}(b_i) \) is an exit attribute of \( t \). The same logic can be extended indefinitely as the chains of dependent attributes become longer and longer. Since we are assuming that the source and target systems are proper, the chains must eventually terminate. At that point, either the corresponding second components of rule-\( \ast \)(\( \text{attu}(c), r \)) and rule-\( \ast \)(\( \text{attu}(c'), r' \)) are associated, or \( \text{attu}(c') \) is the attribute of a terminal symbol. In the later case, the set of possible values of the corresponding member of the second component of rule-\( \ast \)(\( \text{attu}(c), r \)) is a subset of the possible values of the attribute of the terminal symbol. From this and L.5.1, we conclude that rule-\( \ast \)(\( r \)) \( \preceq \) rule-\( \ast \)(\( r' \)).

A picture of the lengthening chains is given in Figure 5.2.

[\(*\)
Figure 5.2. Lengthening Chains in \( r \) and \( r' \).

**Thm L.5.1.** Let \( M = (Q_1, Q_2, \text{TABLE}, \text{ASSOC}) \) be an ipst. If for each row of \( \text{TABLE} \)

\[
t \rightarrow t' \rightarrow x
\]

rule-\( * \)(t) is a restriction of rule-\( * \)(t'), then \( M \) induces a translation.

**Proof:** Suppose \( w \in \text{sen}(Q_1), p \in \text{parse}(w), p \) has non-deviant evaluation \( \text{val}-p \), \( Z \) is a mlmpd of \( p \) with respect to \( S-\text{TABLE} \), \( p' \in \text{phrase}(Q_2) \) and \( p' = \text{TFP}(p, Z) \).

From L.5.2 it follows that rule-\( * \)(p) \( \leq_r \) rule-\( * \)(p'). Because \( p' \) is a complete phrase-structure of \( Pr2 \), all entrance attributes of \( p' \) are attributes of terminal symbols. If the i-th member of the second component of rule-\( * \)(p') is \( \text{attui}(c'i) \), then \( \text{att-val}(\text{label}(c'i), \text{attui}) \) is a fixed finite set. By definition of restriction, the i-th member of the second component of rule-\( * \)(q), \( \text{attui}(ci) \), must have as
its set of possible values, a subset of att-val(label(c'i),attui). Since the relation of rule-*(q) is by definition a restriction of the relation of rule-*(p'), then the set of values of each exit attribute of rt(p) must be a subset of the set of values of the same exit attribute of rt(p').

Corollary C.5.1. Let M be a ips t. If for each row of TABL

\[ t \cdot t' \cdot x \]

rule(t) is a restriction of rule(t'), then M induces a translation.

Proof:

\[ \text{rule}(t) \leq \text{rule}(t') \]
\[ \implies \]
\[ \text{rule-}^*(t) \leq \text{rule-}^*(t'). \]

Ip s t Stronger Than Cpo s t

Ipst are more "powerful" than cpst in that the ipst can translate between more language pairs than can the cpst. Suppose instead that the arguments of a cpst were incomplete systems which used synthesized attributes only. This extended version of a cpst would be as powerful as an ipst even though it did not use inherited attributes. This follows immediately from two facts: (i) every gen-pss can be effectively converted into a syn-pss by changing only the semantics of the system, not the syntax; and (ii) neither extended cpst nor ipst use semantic information in computing TFP. Consequently, the conversion from ipst to extended cpst simply requires modifying the semantics of the source and target systems so that they are synthesized systems. The rest of the transducer is left unchanged.

Lemma L.5.3. For every incomplete phrase-structure transducer, there is an effectively constructable equivalent extended complete phrase-structure transducer.

Proof: Let M = (Q1,Q2,TABL,ASSOC) be an ipst. Define M' to be (Q1',Q2',TABL,ASSOC) where Q1' and Q1 have the same underlying grammar for i = 1 or 2. The semantics of both Q1' and Q2' are changed so that all of the productions pass up the syntactic structure of the parse tree to the sub-structure of unit height whose root is labeled by the axiom Ax. The semantic relation
INCOMPLETE PHRASE-STRUCTURE TRANSLATORS

IPST STRONGER THAN CPST

associated with this sub-structure sets the meaning of the parse tree to be equal to whatever the original system's semantics would.

[*]

We capture the notion of relative strengths between transduction models with the next two definitions.

**Defn D.5.2.** A model of translation \( M_1 \), is strongly less than or equal to \((\leq s)\) a second model of translation \( M_2 \), if whenever \( M_1 \) can induce translation, \( \text{TRANS}: \text{sen}(Q_1) \rightarrow \text{sen}(Q_2) \), then \( M_2 \) can induce an extension of \( \text{TRANS} \) using the same systems \( Q_1 \) and \( Q_2 \).

We also define the concept of strongly equal \((=s)\) and strongly less than \((<s)\).

\[ M_1 \leq s M_2 \text{ and } M_2 \leq s M_1 \implies M_1 = s M_2 \]
\[ M_1 \leq s M_2 \text{ but not } M_2 \leq s M_1 \implies M_1 < s M_2 \]

[*]

**Defn D.5.3.** A model of translation \( M_1 \), is weakly less than or equal to \((\leq w)\) a second model of translation \( M_2 \), if whenever \( M_1 \) can induce translation, \( \text{TRANS}: \text{sen}(Q_1) \rightarrow \text{sen}(Q_2) \), then \( M_2 \) can induce an extension of \( \text{TRANS} \) using some \( Q_3 \) and \( Q_4 \) such that \( L(Q_1) = L(Q_3) \) and \( L(Q_2) = L(Q_4) \).

We also define weakly equal \((=w)\) and weakly less than \((<w)\).

\[ M_1 \leq w M_2 \text{ and } M_2 \leq w M_1 \implies M_1 = w M_2 \]
\[ M_1 \leq w M_2 \text{ but not } M_2 \leq w M_1 \implies M_1 < w M_2 \]

[*]

In terms of these definitions and L.5.3, we say that the extended \( \text{cpst} \) is as powerful as the \( \text{ipst} \), but that the \( \text{cpst} \) is strictly less powerful than the \( \text{ipst} \).

**Thm T.5.2.** As models of translation

\[ \text{cpst} < s \text{ extended cpst} < s \text{ ipst} \]
\[ \text{cpst} < w \text{ extended cpst} = w \text{ ipst} \]

*Proof:* By definition, \( \text{cpst} < s \text{ extended cpst} \). By definition \( \text{cpst} < s \text{ ipst} \). By definition \( \text{cpst} < w \text{ extended cpst} \). By L.5.3, \( \text{extended cpst} = w \text{ ipst} \).

[*]
Inverse IPST

Suppose $M$ induces transform $TFS: \text{sen}(Q_1) \longrightarrow \text{sen}(Q_2)$. In general, $\text{range}(TFS)$ is a subset of $\text{sen}(Q_2)$. We would like to know whether or not there is an IPST $M^*$ which induces the inverse transformation $TFS^*$ from $\text{range}(TFS)$ onto $\text{sen}(Q_1)$ such that $v \in TFS(w)$ if and only if $w \in TFS^*(v)$. The major result of this section (I.5.3) states a sufficient condition under which such an inverse IPST may be effectively constructed.

Defn D.5.4. An IPST $M$ is simple if $V_{n1} = V_{n2}$ and ASSOC is a total identity map from $V_{n1}$ onto $V_{n2}$.

Lemma L.5.4. For every IPST $M = (Q_1, Q_2, \text{TABL}, \text{ASSOC})$, there is an equivalent simple-IPST $M' = (Q_1', Q_2', \text{TABL}', \text{ASSOC}')$.

Proof: If ASSOC is already a total identity mapping from $V_{n1}$ onto $V_{n2}$ then the theorem follows immediately. So assume not. Construct $M'$ as follows:

1) Assume without loss of generality that $M$ is in normal form and $V_{n1} \cap V_{n2} = \emptyset$.

2) $Q_1' = Q_1$.

3) ASSOC' is a total identity mapping.

4) If $A<B_1..B_n> C<D_1..D_m> x_1,..,x_m$ is in TABL then $A<B_1..B_n> \text{trans}(C<D_1..D_m>) x_1,..,x_m$ is in TABL' where $\text{trans}(C<D_1..D_m>) = A<E_1..E_m>$ such that if $x_i = \emptyset$ then $E_i = D_i \in V_{t2}$ and if $x_i = j > \emptyset$ then $E_i = B_j \in V_{n1}$.

5) $Q_2' = (G_2', \text{SEM}_2')$ where $G_2' = (V_{n1}, V_{t2}, A_{x1}, P_{r'})$ and if $C<D_1..D_m> \text{rules}(C<D_1..D_m>)$ is in $\text{Ru}$ of $Q_2$, then
trans(C<D1..Dm>) rules(C<D1..Dm>)
is in Ru' of Q2'. Since att-t(Dj) = att-t(trans(Dj)) (by definition of ASSOC) the semantics is well-defined. Clearly, there is a natural homomorphism, h, from the phrase-structures of t-TABL to those of t-TABL' which preserves semantics. If TFP(t) contains t' then TFP'(t) contains h(t'). Therefore, M and M' are equivalent.

Lemma L.5.5. If M is a simple linear ipst, then phr(t-TABL) = range(TFP).
Proof: Range(TFP) is a subset of phr(t-TABL). That phr(t-TABL) is a subset of range(TFP) follows from two facts:
1) ASSOC is a total bijection => ASSOC-INV is a total bijection.
2) Let
\[ t \cdot t' \cdot x \]
be a row of TABL. M is a linear ipst => for each node c' on the frontier of t' whose label is in Vt2, there is a unique node c on the frontier of t whose label is in Vt1 with which it is associated.

Therefore, for each p' \in phr(t-TABL), there is a corresponding p \in phr(s-TABL) such that TFP(p) contains p'.

Lemma L.5.6. If M = (Q1,Q2,TABL,ASSOC) is a simple linear ipst, then there is an equivalent simple linear ipst M' such that
1) Q1' = Q1
2) TABL' = TABL
3) ASSOC' = ASSOC
4) phrc(Q2') is a subset of phrc(t-TABL).
Proof: Modify Q2 so that the productions in R2' are the same, but the semantics discards as deviant any complete tree in phrc(Q2) that is not in phrc(t-TABL).

Lemma L.5.7. For every strong linear ipst M = (Q1, Q2, TABL, ASSOC), there is an effectively constructable inverse ipst IM = (IQ1, IQ2, ITABL, IASSOC) such that TFS(w) contains v iff ITFS(v) contains w.

Proof: From L.5.3 and L.5.4 we know that a simple-ipst M' equivalent to M may be constructed such that phrc(Q2') is a subset of phrc(t-TABL'), where phrc(t-TABL') equals range(TFP'(M)) by L.5.6 Construct IM from M' as follows: If

\[ t \ t' \ x \]

is a row of M', then IM contains a row

\[ t'' \ t' \ x'' \]

where if \( x_i = k > 0 \) then \( x_k'' = 1 \).

Since M is strong, so are M' and IM. Let Z be a mlmpd of p according to s-TABL. Since M is strong, Z is also a lmd of p. Therefore, for each sub-structure, \( r \), of Z, there is a unique sub-structure, \( r' \), in \( p' = TFP(p, Z) \) which is associated with \( r \), and conversely.

Let Z' be the mlmpd of \( p' \) induced by \((p, Z)\). It is also a lmd of \( p' \). Since IM is strong, for each sub-structure, \( r' \), of \( Z' \), there is a unique sub-structure, \( r \), in \( p = ITFP(p', Z') \) which is associated with \( r' \), and conversely. Therefore, if \( p = ITFP(p', Z') \), then \( p' = TFP(p, Z) \). From L.5.5 and L.5.6 we know that phrc(Q2') is a subset of phr(t-TABL) = range(TFP'). Therefore, TFS'(w) contains v = \( \Rightarrow \) there is a parse \( p' \) of v in range(TFP') = \( \Rightarrow p' \in \text{domain}(ITFP) \Rightarrow ITFP(p') \) contains p a parse of w.

Thm T.5.3. For every strong ipst M = (Q1, Q2, TABL, ASSOC), there is an inverse ipst M' = (Q1', Q2', TABL', ASSOC') such that TFS(w) contains v iff TFS'(v) contains w.

Proof: If L.5.5 could be strengthened so that it were true for exponential ipst, the result would follow immediately using reasoning similar to that used to prove L.5.7. For exponential ipst, range(TFP) is, in general, a proper subset of phr(t-TABL). Suppose row z of TABL is used in the transformation of some phrase-structure p where
is that row of $M$, a simple ipst, and $x_i = x_j > 0$ for distinct $i, j$. The sub-structures which are composed at $fr(t', i)$ and $fr(t', j)$ in the steps of the transformation following the application of row $z$ must be the same. In general, no such constraint is imposed on $phr(t\text{-TABL})$. It is only necessary to insure that the set of complete phrase-structures of $phr(t\text{-TABL})$ which are non-deviant with respect to the semantics of $Q_2$ is equal to the set of complete phrase-structures of $range(TFP)$ with respect to the semantics of $Q_2$. This is done by changing the semantics of $Q_2$ so that $att\text{-}t(rt(t'))$ contains an attribute "proper" which is assigned a value of "ok" if the trees rooted by $fr(t', i)$ and $fr(t', j)$ are identical, and is assigned a value of "err" if the trees rooted by $fr(t', i)$ and $fr(t', j)$ are not identical. The trees which are the arguments of the function which is used to compute the value of "proper" are passed up to the function using a synthesized attribute.

[∗]

**Complexity**

In this section, as in the section on complexity in the last chapter, we will examine the translator as a procedure. We again restrict ourselves to the case of the honest translator. The definition of an honest cpst given in chapter iv can be extended without modification to the ipst, and it follows immediately that an honest incomplete translator is algorithmic. This is immediate from the fact that if any parse of a string is meaningful, then a member of the finite effectively computable set of parses without linear recursive sub-structures is meaningful. In this section we analyze the execution time of an honest ipst. Let $M$ be such a translator in normal form and suppose $w \in sen(Q_1)$ has length $n$.

The parses of $w$ may be computed in factored form in

$$O(n^{**3})$$ (5.1)

time. Given the set of parses in this form, it takes

$$O(wt(t))$$ (5.2)

time to construct parse tree $t$. Let
\[ \text{VAL} = \{ \text{VAL-T} \mid t \in \text{parse}(w) \} \]

If \( Q_1 \) is semantically unambiguous, then \( |\text{parse}(w)| = |\text{VAL}| \). In general, any number of parses of \( w \) may be deviant. Because \( M \) is honest, no matter which non-deviant parse is input to the transformer, the parse of a valid translation will be output. The parses of \( w \) must be evaluated in some well-defined manner until either all parses have been eliminated as deviant or the first non-deviant parse is found. If all parses are eliminated as deviant, then \( w \) is not a source sentence, and has no translation. Otherwise, the first non-deviant parse found is given to the transformer. In the worst case, only one parse of \( w \) will be non-deviant and, assuming that \( \text{parse}(w) \) is ordered, it will be the last member. In the best case, the first member of \( \text{parse}(w) \) will be non-deviant. Suppose each function of a semantic rule in \( \text{Rul} \) is in \( \text{PTIME} \) and the length of its output is less than or equal to the length of its input. Then the analysis done in chapter iii on the time necessary to compute \( \text{VAL-T} \) is applicable here. In chapter iii this was shown to be

\[ O(n^{**3} + nF(n)) \]  \hspace{1cm} (5.3)

where \( F(n) \) is the most worst time bound for any semantic function in and semantic rule of \( \text{Rul} \). The analysis of chapter iv on the time necessary to transform a parse tree \( t \) is applicable here without modification. In the best case, the time necessary to translate \( w \) is

\[ O(n^{**3} + nF(n)) \]  \hspace{1cm} (5.4)

for the multi-linear and linear ipst and

\[ O(k**n + nF(n)) \]  \hspace{1cm} (5.5)

for the exponential ipst. In the worst case, the time is

\[ O(n^{**3} + pn + pnF(n)) \]  \hspace{1cm} (5.6)

where \( p = |\text{parse}(w)| \) for linear and multi-linear ipst and

\[ O(k**n + pn + pnF(n)) \]  \hspace{1cm} (5.7)

for the exponential ipst.

If the number of parses of \( w \) is bounded by a polynomial function of the length of \( w \), then the linear and multi-linear honest translators are in \( \text{PTIME} \).
For the special case where Q is both semantically and syntactically unambiguous, times 5.6 and 5.7 reduce to times 5.4 and 5.5, respectively, since \( |\text{parse}(w)| \) is then equal to one.

**IPST Equivalent to Turing Translator**

The reader familiar with syntax-directed methods found in the literature realizes that the traditional syntax-directed transduction schema can only translate between a relatively "small" class of language pairs. One of the most general models of syntax-directed transduction previously studied is that of Buttelmann [74], which he called a "finitely specified transducer". It subsumes the syntax-directed transduction schema of Aho and Ullman [69]. However, no finitely specified transducer can translate between languages in which the ratio of the length of the target sentence to the length of the source sentence is greater than exponential. One of the goals of our research has been to find means to expand the power of syntax-directed methods, so that, for example, we could design a syntax-directed transducer to translate between such languages. In this we have been successful. The ipst has all of the power of a Turing translator, which is defined in D.5.5. The main result of this section (T.5.4) states that the ipst is capable of inducing whatever partial recursive translation is defined between two phrase-structure languages.

**Definition D.5.5.** A Turing translator is a Turing machine with output. When an encoding of the source and target phrase-structure systems are placed on its input tape, as well as a source string which is to be translated, the machine will enumerate on its output tape all of the translations of the source string.

[\(*\)]

**Lemma L.5.8.** Let \( L_1 \) and \( L_2 \) be two phrase-structure languages. There is an ipst which induces the partial translation \( TRANS \), from \( L_1 \) to \( L_2 \), such that \( TRANS(w) = \)

\[
\{ v \mid |w| \leq |v|, \text{mean}(w) \cap \text{mean}(v) \neq \emptyset, \\
v \text{is a sentence of } L_2 \}
\]

**Proof:**

**Constructions:**
Suppose $L_1 = L(Q_1)$, $L_2 = L(Q_2)$, $V_{t1} = (a_1, \ldots, a_y)$, and $V_{t2} = (b_1, \ldots, b_z)$.

Construct $Q$ as follows:

$$Q = (G, \text{SEM})$$

where $G = (\langle S, F, E \rangle, V_{t1}, S, \text{Pr})$ and $\text{Ru}$ is the following production-rule pairs:

<table>
<thead>
<tr>
<th>Productions</th>
<th>Semantic Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) $S&lt;E&gt;$</td>
<td>$\text{val}(x_0) = f(\text{val}(x_1))$</td>
</tr>
<tr>
<td>2) $E&lt;E&gt;$</td>
<td>$\text{val}(x_0) = \text{val}(x_1)$</td>
</tr>
<tr>
<td>3) $E&lt;F&gt;$</td>
<td>$\text{val}(x_0) = \text{val}(x_1)$</td>
</tr>
<tr>
<td>4.1) $F&lt;al F&gt;$</td>
<td>$\text{val}(x_0) = al \</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>4.y) $F&lt;ay F&gt;$</td>
<td>$\text{val}(x_0) = ay \</td>
</tr>
<tr>
<td>5.1) $F&lt;al&gt;$</td>
<td>$\text{val}(x_0) = al$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>5.z) $F&lt;ay&gt;$</td>
<td>$\text{val}(x_0) = ay$</td>
</tr>
</tbody>
</table>

where $f$ is the "super" relation of T.3.3 for $Q_1$.

Construct $Q'$ as follows:

$$Q' = (G', \text{SEM'})$$

where $G' = (\langle S', F', E', A_1, \ldots, A_y \rangle, V_{t2}, S', \text{Pr'})$ and $\text{Ru}'$ is the following production-rule pairs:

<table>
<thead>
<tr>
<th>Productions</th>
<th>Semantic Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1') $S'&lt;E'&gt;$</td>
<td>$\text{val}(x_0) = h(s(x_1), r(x_1))$</td>
</tr>
<tr>
<td>2.1') $E'&lt;b_1 E'&gt;$</td>
<td>$s(x_0) = s(x_2)$; $r(x_0) = b_1 \</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2.z') $E'&lt;b_z E'&gt;$</td>
<td>$s(x_0) = s(x_2)$; $r(x_0) = b_z \</td>
</tr>
</tbody>
</table>
INCOMPLETE PHRASE-STRUCTURE TRANSLATORS
IPST EQUIVALENT TO TURING TRANSLATOR

3.) \( E' < F' \) 
   \[ s(x_0) = s(x_1); \]
   \[ r(x_0) = r(x_1) \]

4.1.) \( F < A_1 \) 
   \[ s(x_0) = a_1 \parallel s(x_1); \]
   \[ r(x_0) = r(x_1) \]

... ... ...

4.4.) \( F < A_y \) 
   \[ s(x_0) = a_y \parallel s(x_1); \]
   \[ r(x_0) = r(x_1) \]

... ... ...

5.1.1.) \( A_1 < b_1 F' \) 
   \[ s(x_0) = s(x_2); \]
   \[ r(x_0) = b_1 \parallel r(x_2) \]

... ... ...

5.1.4.) \( A_1 < b_2 F' \) 
   \[ s(x_0) = s(x_2); \]
   \[ r(x_0) = b_2 \parallel r(x_2) \]

... ... ...

5.5.1.) \( A_1 < b_3 F' \) 
   \[ s(x_0) = s(x_2); \]
   \[ r(x_0) = b_3 \parallel r(x_2) \]

... ... ...

5.7.1.) \( A_1 < b_4 F' \) 
   \[ s(x_0) = s(x_2); \]
   \[ r(x_0) = b_4 \parallel r(x_2) \]

... ... ...

6.2.1.) \( A_1 < b_5 \) 
   \[ s(x_0) = \xi; \]
   \[ r(x_0) = b_5 \]

... ... ...

6.2.4.) \( A_1 < b_5 \) 
   \[ s(x_0) = \xi; \]
   \[ r(x_0) = b_5 \]

where \( h(s(x_1), r(x_1)) = g(r(x_1)) \) if \( f(s(x_1)) \cap g(r(x_1)) \neq \emptyset \), and \( h(x) = "err" \), otherwise.

The relation \( f \) is the same "super" relation as used in the construction of \( Q \), and \( g \) is the corresponding "super" relation for \( Q' \).

ASSOC is defined by:

\[ S \rightarrow S' \quad E \rightarrow E' \quad F \rightarrow F' \]

TABL is given in Figure 5.3.
### Figure 5.3. Phrase-Correspondence Table of $M$

**End Construction.**

From the construction of $Pr$ and $Pr'$, it is clear that $T_1^+ = \{f_r(t) \mid t \in \text{phrc}(Pr)\}$

and that $Vt_2^+ = \{f_r(t) \mid t \in \text{phrc}(Pr')\}$

From T.3.3 and the definition of the "super" relation $\f$, we conclude that for $t \in \text{phrc}(Pr)$, $\text{mean}(t) = \text{mean}(f_r(t))$. Therefore, $L(Q) = L(Q')$. From the definition of relation $h$, it is clear that $h$ is a restriction of relation $g$. From this fact and T.3.3,
we conclude that \( \text{mean}^1(w) \) is a subset of \( \text{mean}^2(w) \). Therefore, \( L(Q') \) is a subset of \( L(Q_2) \).

Suppose \( w = w_1 \ldots w_m \) is a sentence of \( Q \). Because each parse of \( w \) includes a node labeled by \( E \), and \( E <E> \) is a production of \( Q \), there is an infinite number of parses of \( w \) in \( Q \). For \( x \geq 0 \), the lmd, \( Z \), of some parse of \( w \) may be written as

\[
\begin{align*}
(S <E>, 1, 1), \\
(E <E>, 1, 2), \ldots, (E <E>, 1, 2), \\
(E <F>, 1, 3), \\
(F <a(i,1) F>, 1, 4:i1), \ldots \\
(F <a(i,m-1) F>, m-1, 4.i(m-1)), \\
(F <a(i,m)>, m, 5.i m)
\end{align*}
\]

where the production \( E <E> \) appears in \( x \) triples of \( Z \), and where \( a(i,j) = w(j) \) for \( j \in m \).

Consequently, the lmd \( Z' \) of \( t' = \text{TFP}(t,Z) \) induced by \( (t,Z) \) is:

\[
\begin{align*}
(S' <E'>, 1, 1'), \\
(E' <b(j,1) E'>, 1, 2.j1'), \ldots \\
(E' <b(j,x) E'>, x, 2.jx'), \\
(E' <F'>, x+1, 3'), \\
(F' <A(i,1)>, x+1, 4.i1'), \\
(A(i,1)<b(j,1) F'>, x+1, 5.i1.j1'), \ldots \\
(F' <A(i,m-1)>, x+m-1, 4.i(m-1)'), \\
(A(i,m-1)<b(j,x+m-1) F'>, x+m-1, 5.i(m-1).j(x+m-1)'), \\
(<F' <A(i,m)>, x+m, 4.im'), \\
(A(i,m)<b(j,x+m)>, x+m, 5.im.j(x+m)'), \\
\end{align*}
\]

where \( v(k) = b(j,k) \) for \( k \in (x+m) \).
t' may be written as \( S' < E' > [t_0'] \) where \( t_0 \in \text{phrt}(Pr') \). From the definition of \( Ru \), we conclude that \( r(rt(t')) = v \) and \( s(rt(t')) = w \). Therefore, \( \text{val}(rt(t')) \) is equal to \( g(v) \) if \( f(w) \neq \emptyset \) and \( \text{val}(rt(t')) \) is equal to "err" otherwise. Therefore, \( \text{mean}(t') \neq \emptyset \implies v \) is a translation of \( w \), and \( \text{mean}(t') = \emptyset \implies v \) is not a translation of \( w \). For a particular \( x \),

\[
\{ \text{fr}(t') \mid t' \in \text{TFP}(t) \}
\]

is equal to

\[
\{ v \mid v = vl \ldots vx v(x+1) \ldots v(x+m) \}
\]

For \( w \in \text{sen}(Q) \), for \( x > 0 \), there is a parse, \( t \), of \( w \) with \( x \) occurrences of \( E < E > \) as phrase-structures of \( t \). Therefore, for \( x > 0 \), there is a parse, \( t \), of \( w \) such that

\[
\{ \text{fr}(t') \mid t' \in \text{TFP}(t) \}
\]

is equal to

\[
\{ v \mid v = vl \ldots vx v(x+1) \ldots v(x+m) \}
\]

Therefore, for each \( v \in \text{sen}(Q') \) such that \( v \) is a translation of \( w \) and \( |vl| \geq |wl| \), then \( v \in \text{sen}(Q') \) and \( v \in \text{TFS}(w) \).

Therefore, \( M \) induces the partial translation from \( Q_1 \) to \( Q_2 \) such that

\[
v \in \text{TFS}(w) \quad \iff \quad |vl| \geq |wl| \text{ and } v \text{ is a translation of } w
\]

\([\ast]\)

**Lemma 1.5.9.** Let \( L_1 \) and \( L_2 \) be two phrase-structure languages. There is an ipst which induces the partial translation, \( \text{TRANS} \), from \( L_1 \) to \( L_2 \), such that \( \text{TRANS}(w) = \\

\[
\{ v \mid |vl| \geq |wl|, \text{mean}(w) \cap \text{mean}(v) \neq \emptyset, v \text{ is a sentence of } L_2 \}
\]

**Proof:**

**Constructions:**
Suppose \( L_1 = L(Q_1), L_2 = L(Q_2), \) \( Vt_1 = \{a_1, \ldots, a_y\}, \) and \( Vt_2 = \{b_1, \ldots, b_z\}. \)

Construct \( Q \) as follows:

\[
Q = (G, SEM) \quad \text{where} \\
G = \{(S,E,F,B_1, \ldots, B_z), Vt_1, S, Pr\}
\]

and \( Ru \) is the following production-rule pairs:

<table>
<thead>
<tr>
<th>Productions</th>
<th>Semantic Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) ( S\langle E \rangle )</td>
<td>( \text{val}(x_0) = h(s(x_1), r(x_1)) )</td>
</tr>
<tr>
<td>2.1) ( E\langle a_1 \ E \rangle )</td>
<td>( s(x_0) = a_1 \</td>
</tr>
<tr>
<td>2.2) ( E\langle ay \ E \rangle )</td>
<td>( s(x_0) = ay \</td>
</tr>
<tr>
<td>3) ( E\langle F \rangle )</td>
<td>( s(x_0) = s(x_1); ) ( r(x_0) = r(x_1) )</td>
</tr>
<tr>
<td>4.1) ( F\langle B_1 \rangle )</td>
<td>( s(x_0) = s(x_1); ) ( r(x_0) = b_1 \</td>
</tr>
<tr>
<td>4.2) ( F\langle B_2 \rangle )</td>
<td>( s(x_0) = s(x_1); ) ( r(x_0) = b_2 \</td>
</tr>
<tr>
<td>5.1.1) ( B_1\langle a_1 \ F \rangle )</td>
<td>( s(x_0) = a_1 \</td>
</tr>
<tr>
<td>5.2.1) ( B_1\langle ay \ F \rangle )</td>
<td>( s(x_0) = ay \</td>
</tr>
<tr>
<td>6.1.1) ( B_1\langle a_1 \rangle )</td>
<td>( s(x_0) = a_1; ) ( r(x_0) = \epsilon )</td>
</tr>
<tr>
<td>6.2.1) ( B_1\langle ay \rangle )</td>
<td>( s(x_0) = ay; ) ( r(x_0) = \epsilon )</td>
</tr>
</tbody>
</table>
where \( h(s(x_l), r(x_l)) = f(s(x_l)) \) if \( f(s(x_l)) \cap g(r(x_l)) \neq Q \), and \( h(s(x_l), r(x_l)) = "err" \), otherwise. \( f \) is the "super" relation of T.3.3 for \( Q_1 \).

Construct \( Q' \) as follows:

\[
Q = (G', \text{SEM}') \quad \text{where} \\
G' = ((S', E', F'), Vt_2, S', Pr', Ru')
\]

and \( Ru' \) contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Productions</th>
<th>Semantic Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1') ( S' &lt; E' )</td>
<td>( \text{val}(x_0) = g(\text{val}(x_l)) )</td>
</tr>
<tr>
<td>2') ( E' &lt; E' )</td>
<td>( \text{val}(x_0) = \text{val}(x_l) )</td>
</tr>
<tr>
<td>3') ( E' &lt; F' )</td>
<td>( \text{val}(x_0) = \text{val}(x_l) )</td>
</tr>
<tr>
<td>4.1') ( F' &lt; \text{bl} F' )</td>
<td>( \text{val}(x_0) = \text{bl} \parallel \text{val}(x_2) )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>4.z') ( F' &lt; \text{bz} F' )</td>
<td>( \text{val}(x_0) = \text{bz} \parallel \text{val}(x_2) )</td>
</tr>
<tr>
<td>5.1') ( F' &lt; \text{bl} )</td>
<td>( \text{val}(x_0) = \text{bl} )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>5.z') ( F' &lt; \text{bz} )</td>
<td>( \text{val}(x_0) = \text{bz} )</td>
</tr>
</tbody>
</table>

where \( g \) is the "super" relation of \( Q_2 \).

ASSOC is defined by:

\[
S \rightarrow S' \quad E \rightarrow E' \quad T \rightarrow T'
\]

TABL is given in Figure 5.4.
End Construction.

From the construction of $P$ and $Pr'$, it is clear that $Vt1^+ = \{fr(t) | t \in \text{phrc}(Pr)\}$

and that $Vt2^+ = \{fr(t) | t \in \text{phrc}(Pr')\}$

From T.3.3 and the definition of $h$, we conclude that $\text{mean}(w)$ is a subset of $\text{mean}(w)$. Therefore, $L(Q)$ is a subset of $L(Q')$. From T.3.3 and the definition of $g$, we conclude that for $t \in \text{phrc}(Pr)$, $\text{mean}'(t) = \text{mean}(fr(t))$. Therefore, $L(Q) = L(Q')$.

Suppose $w = w_1 ... w_m$ is a sentence of $Q$. Because no parse of $w$ includes a linear recursive sub-structure, there are only a finite number of parses of $w$ in $Q$. For $0 \leq x \leq m$, the Lind $Z_x$ of some parse of $w$
may be written as

\[(S < E >, 1, 1),\]
\[(E < a(i,1) E >, 1, 2,i1), \ldots \]
\[(E < a(i,x) E >, x, 2,i x),\]
\[(E < F >, x+1, 3),\]
\[(F < B(j,1) >, x+1, 4,j1), \ldots \]
\[(B(j,1) < A(i,x+1) F >, x+1, 5,j1.i(x+1)), \ldots \]
\[(F < B(j,m-x-1) >, m-1, 4.j(m-x-1)), \ldots \]
\[(B(j,m-x-1) < a(i,m-1) F >, m-1, 5.j(m-x-1).i(m-1)), \ldots \]
\[(F < B(j,m-x) >, m, 4.j(m-x)), \ldots \]
\[(B(j,m-x) < a(i,m) >, m, 5.j(m-x).m) \]

where \(a(i,k) = w(k)\) for \(k \leq m\).

\(t\) may be written as \(S < E >[t0]\) where \(t0 \in \text{phrc}(Pr')\). From the definition of \(Ru\), we conclude that \(r(rt(t0)) = b1 \ldots b(m-x)\), and \(s(rt(t0)) = w1 \ldots wm\). From the definition of \(h\), \(\text{mean}(t) \neq \emptyset \) iff \(\text{mean}(w) \cap \text{mean}(b1 \ldots b(m-x)) \neq \emptyset \).

Suppose \(t\) is a non-deviant parse of \(w\). The \(\text{ldm}\) \(Z'\), of \(t' = \text{TFP}(t,Z)\) induced by \((t,Z)\) is:

\[(S' < E' >, 1, 1'),\]
\[(E' < E' >, 1, 2'), \ldots \]
\[(E' < E' >, 1, 2'),\]
\[(E' < F' >, 1, 3'),\]
\[(F' < b(j,1)F' >, 1, 4,j1'), \ldots \]
\[(F' < b(j,m-x-1) F' >, m-x-1, 4.j(m-x-1)), \]
\[(F' < b(j,m) >, m-x, 5.jm) \]
Then \( fr(t') = v = b_l \ldots b_{(m-x)} \), and \( mean'(t) = mean2(v) \).

Since we have previously shown that \( t \) is not deviant, then \( mean(w) \cap mean(v) \neq \emptyset \). So \( v \) is a translation of \( w \).

Since \( x \) ranges over all integers from \( 0 \) to \( m \), for each \( v = b_l \ldots b_{(m-x)} \), there is a parse \( t = S<E>[t_0] \), of \( w \), such that \( r(rt(t_0)) \) is \( b_l \ldots b_{(m-x)} \). Therefore, \( v \in \text{sent}(Q_2) \), \( |vl| \leq |wl| \), and \( mean_1(w) \cap mean_2(v) \neq \emptyset \) iff \( v \in \text{TFS}(w) \).

Theorem 5.4. Let \( L_1 \) and \( L_2 \) be two phrase-structure languages. There is an Ipst which induces the partial translation, \( \text{TNS} \), from \( L_1 \) to \( L_2 \), such that \( \text{TNS}(w) = \{ v : mean(w) \cap mean(v) \neq \emptyset, v \text{ is a sentence of } L_2 \} \)

Proof. From L.5.8 and L.5.9 we know there are Ipst \( M \) and \( M' \) such that \( M \) induces the partial translation from \( L_1 \) to \( L_2 \) for translations which are longer than the source string, and \( M' \) induces the partial translation from \( L_1 \) to \( L_2 \) for translations which are shorter than the source string. Form the union machine, \( M'' \), of \( M \) and \( M' \). Assume that \( V_{n_1} \) and \( V_{n_1}' \) are disjoint, except for a common axiom \( Ax_{1''} \). Similarly, assume \( V_{n_2} \) and \( V_{n_2}' \) are disjoint, except for a common axiom \( Ax_{2''} \). For \( i = 1,2 \)

\( Q_{1''} = (Gi'', \text{SEMI}'') \) where

\( Gi'' = (V_{n_1} \cup V_{n_1}', V_t, Ax_{1''}, Pri \cup Pri') \)

\( \text{SYN-ATTi''} = \text{SYN-ATTi} \cup \text{SYN-ATTi}' \),
\( \text{INH-ATTi''} = \text{INH-ATTi} \cup \text{INH-ATTi}' \),
\( att'' = att \cup att' \)
\( U'' = U \cup U' \)
\( \text{att-vali''} = att-vali \cup att-vali' \)
and \( Rui'' = Rui \cup Rui' \).

\( M'' = (Q1'', Q2'', \text{Tabl} \cup \text{Tabl}', \text{ASSOC} \cup \text{ASSOC}') \) where \( \text{Tabl} \cup \text{Tabl}' \) is the concatenation of \( \text{Tabl}' \) to the bottom of \( \text{Tabl} \). Thus if \( \text{Tabl} \) had \( k \) rows and \( \text{Tabl}' \) had \( k' \) rows, then their union would have \( k+k' \) rows.

\( M'' \) clearly induces the union of the two partial translations, TFS and TFS'. Therefore, \( M'' \) induces
Summary

In this chapter we investigated the incomplete phrase-structure transducer which is a generalization of the transduction model introduced in the last chapter, the complete phrase-structure transducer. The source and target systems of the incomplete transducer may be general phrase-structure systems. Two sufficient conditions for an incomplete transducer to be semantic-preserving were stated and proven sufficient. The relative powers of the complete and incomplete transducers to translate between languages was compared. We proved that the incomplete transducer can translate between more language pairs than the complete transducer. We then showed that for a subclass of incomplete transducers, a transducer which relates language A to language B may be inverted into a transducer which relates language B to language A. The complexity of a subclass of incomplete translators was examined next for the special case where the underlying systems are semantically unambiguous. Upper bounds on the execution time of such translators were derived. Finally, we defined another model of translation, the Turing translator, and proved that the incomplete transducer is as powerful as this other model.
Chapter VI - Predicate Phrase-Structure Translators

And the Lord said, Behold the people is one, and they have all one language; and this they begin to do; and now nothing will be restrained from them, which they have imagined to do.

Go to, let us go down, and there confound their language, that they may not understand one another's speech.

So the Lord scattered them abroad from thence upon the face of the earth; and they left off to build the city.

Therefore is the name of it called Babel; because the Lord did there confound the language of all the earth; and from thence did the Lord scatter them abroad upon the face of all the earth.

Gen. xi

Introduction

Complete transducers do not use any semantic information at transduction time. Incomplete transducers use semantic information only to filter deviant parses. Certainly, the meanings of the entries in the table can be considered when it is constructed. The semantic relationships between source and target phrase-structures in each row of the table were the basis for the proof of T.5.1 which established a condition which is sufficient to ensure the correctness of a translator. However, once the table is built, the transformation process itself is entirely syntax-driven. We extend the incomplete transducer by using some semantic information at transformation time. We attach a fourth column to TABL - a condition column. In each row of TABL the single source entry may have several target entries and associated conditions. The conditions "con(i)" are predicates whose arguments are the attribute values of the single source entry.
Basic Definitions

Defn D.6.1. A predicate phrase-structure transducer (p.p.t) is a 4-tuple \( M = (Q_1, Q_2, \text{TABL}, \text{ASSOC}) \) where:

1) \( Q_1 \), the source system, is a pss.
2) \( Q_2 \), the target system, is a pss.
3) \( \text{ASSOC}: V_{n1} \rightarrow V_{n2} \) is a partial function such that \( \text{ASSOC}(Ax_1) = Ax_2 \), and \( \text{att}(v) = \text{att}(\text{ASSOC}(v)) \) for \( v \in V_{n1} \).
4) \( \text{TABL} \) is a finite non-empty table. The \( i \)-th row has four entries:

\[
\begin{align*}
&t \quad t' \quad x_1 \quad \text{con}(1) \\
&t' \quad x_n \quad \text{con}(n)
\end{align*}
\]

where

a) \( t \in \text{phr}(\text{UG}(Q_1)) \).
b) \( t' \in \text{phr}(\text{UG}(Q_2)) \), denoted \( t \in \text{TABL}(i,j) \).
c) \( \text{ASSOC}(\text{label}(\text{rt}(t))) = \text{label}(\text{rt}(t')) \).
d) \( x_j = y_1 \ldots y_n \) is a string of integers, called the \( x \)-vector, which is denoted \( x \in \text{TABL}(i,j) \). If \( f(t) = h_1 \ldots h_m \) and \( f(t') = g_1 \ldots g_n \) then

i) \( \text{label}(g_k) \in V_{n2} \Rightarrow y_k = 0 \);

ii) \( \text{label}(g_k) \in V_{n2} \Rightarrow y_k = z \)

where \( \text{label}(h_z) \in V_{n1} \) and \( \text{ASSOC}(\text{label}(h_z)) = \text{label}(g_k) \).

The \( x \)-vector is also called the association or index vector.

e) \( \text{con}(i) \) is a predicate, \( c \in \text{TABL}(i,j) \), whose arguments are a subset of the attributes of \( t \).

The union of all source \( t \) in \( \text{TABL} \) is called \( s \)-\( \text{TABL} \) and the union of all target \( t' \) in \( \text{TABL} \) is called \( t \)-\( \text{TABL} \).
An example of a predicate transducer is given as example pst-3 in Appendix C. The next definition defines the phrase-transform of a ppst.

**Defn D.6.2.** Let $M$ be a ppst, and let $p \in \text{phrc}(G_1)$. Suppose $Z$ is a leftmost partial derivation of $p$ with respect to $s\text{-TABL}$:

$$(p_1, n_1, m_1), \ldots, (p_z, n_z, m_z)$$

and suppose that val-$p$ is a non-deviant evaluation of $p$. If VAL is the set of evaluations of all the complete trees of $G_1$, then the phrase-transform

$$\text{TFP}: \text{phrc}(G_1) \times \text{LMD}(s\text{-TABL}) \times \text{VAL} \rightarrow \text{phr}(Q_2)$$

is a relation computed by A.6.1.

**Algorithm A.6.1.**

**Purpose:** Compute the phrase-transform of ppst $M$.

**Input:** $p \in \text{phrc}(G_1), Z$, a leftmost partial derivation of $p$, and an evaluation val-$p$ of $p$.

**Output:** A finite set of trees in phr($Q_2$).

**Steps:**

1) Initially, $\text{TFP}(p, Z, \text{val}-p) = p'$, a node whose label is equal to $\text{ASSOC}(\text{label}(\text{rt}(p)))$. $p'$ is said to be "associated" with $\text{rt}(p)$. We write this as $\text{assoc}(p') = \text{rt}(p)$.

2) If $p' = \text{TFP}(p, Z, \text{val}-p)$,

$$q \quad q'! \quad x! \quad \text{con}(1)$$

$$\cdots$$

$$q'!n \quad x!n \quad \text{con}(n)$$

is a row of TABL, for some $i \in Z$, $q = p_i$, $(c_1, \ldots, c_m)$ are nodes at fr($p'_j, y_1$), $\ldots$, fr($p'_j, y_m$), respectively, each associated with $\text{rt}(p_i)$, where $j > k \implies y_j > y_k$, and con($u$) is true for the values of the attributes of $p_i$, then $\text{TFP}(p, Z, \text{val}-p)$ contains
compose(*,qu',y1) .. compose(*,qu',ym)(p')

If x(u,j) = k > 0, then for each occurrence of qu' composed to p', assoc(fr(qu',j)) is fr(p'i,k).

TFP(p) = {TFP(p,Z, val-p) | Z a Impd of p, val-p is an evaluation of p}.

Example E.6.1. Consider example pst-3 in Appendix C. Suppose the source sentence is:

2 2.1 * +

Its parse is pictured in Figure 6.1.

![Figure 6.1. Parse of "2 2.1 * +"]

A detailed explanation of this sentence's translation may be found in Appendix C.

As in the case of the complete and incomplete transducers, the way in which the ASSOC map is defined guarantees that the tree composition specified in transforming a source tree into a target tree is well-defined. If q' is to be composed to fr(t',i), then label(rt(q')) is equal to label(fr(t',i)).
Defn D.6.3. Let $M$ be a ppst. If $w \in \text{sen}(Q1)$, then
\[
\text{TFS}(w) =
\{
t' \mid t \in \text{parse}(w), \ t \in \text{phrc}(Q1),
\quad t' \in \text{TFP}(t), \ t' \in \text{phrc}(Q2)\}
\]

Intuitively, the semantics of the source tree is being used as a selector switch. In each row of TABL there are $n$ possible target trees $(t'1, \ldots, t'n)$, which can correspond to phrase-structure $t$, but $t'i$ is a "valid" transformation of $t$ just when the semantics of $t$ is such that condition $\text{con}(i)$ is satisfied. Only in that case is $t'i$ said to "correspond" to $t$. The set of possible values for attributes of $t$ is divided into a finite number of classes, where membership in a class is determined by which particular conditions are satisfied for that particular evaluation of $t$. If the conditions are mutually exclusive, i.e., if one and only one can be satisfied by any particular evaluation of $t$, then the semantic domain of the attributes of an occurrence of $t$ is in effect partitioned into a finite number of equivalence classes. Because any number of conditions may be satisfied rather than just one, a particular evaluation of an occurrence of $t$ may have several corresponding phrase-structures.

Translation

The next theorem states a condition which is sufficient to ensure that a predicate transducer is semantic-preserving. The condition is similar to the one stated in T.5.1.

Thm T.6.1. Let $M$ be a predicate phrase-structure transducer. For $1 \leq i \leq |\text{TABL}|$, suppose $\text{TABL}(i)$ is
\[
t \ t'1 \ xl \ \text{con}(1) \\
\vdots \\
\ t'n \ xn \ \text{con}(n)
\]
and if $c-\text{TABL}(i,j)$ is satisfied, then the restriction of rule-\(*(t)$ to the domain in which $\text{con}(j)$ is true is a restriction of rule-\(*(t-\text{TABL}(i,j))$. Then $M$ induces a translation.

Proof: Follows directly from the proof of T.5.1. Whenever an occurrence of phrase-structure $t'j$ from $t-\text{TABL}$ corresponds to an evaluated occurrence of source
phrase-structure \( t \), predicate \( \text{con}(j) \) is satisfied and hence so are the conditions of T.5.1. The fact that other \( t \)'s may not satisfy the constraints of T.5.1 is irrelevant since they do not correspond to \( t \) and hence are not used in the transformation.

\[ \text{Corollary C.6.1.} \] Let \( M \) be a predicate phrase-structure transducer, where \( \text{TABL} \) has \( k \) rows. For \( i \in \{1, \ldots, n\} \), suppose \( \text{TABL}(i) \) is

\[
\begin{array}{cccc}
\text{t} & \text{t}' & \text{xl} & \text{con}(1) \\
\vdots & \vdots & \vdots & \\
\text{t}' & \text{xn} & \text{con}(n)
\end{array}
\]

and whenever \( \text{c-TABL}(i,j) \) is satisfied, the restriction of rule \( t \) to the domain in which \( \text{con}(i) \) is true is a restriction of rule \( t-\text{TABL}(i,j) \). Then \( M \) induces a translation.

\[ \text{[*]} \]

**Effective equivalence of PPST and IPST**

Because of T.5.4 and Church's Thesis, we know that ipst is weakly equivalent to post. However, it is not obvious whether a ppst may be effectively converted into an ipst, and even assuming that an effective conversion is possible, it is not clear how closely the syntax and semantics of the equivalent ipst can mirror the syntax and semantics of the original ppst. In this section we show that from any ppst, a weakly equivalent ipst may be effectively constructed which "reflects" in a natural way the syntax and semantics of the source and target systems of the original ppst.

Before we show how to convert a ppst into an ipst, we first note that the normal form for ipst carries over directly to the predicate transducer. Any ppst may be algorithmically transformed into a weakly equivalent ppst in which all \( s-\text{TABL} \) and \( t-\text{TABL} \) entries have unit height. The proof that a transducer may be converted into this normal form is essentially the same as for the cpst given in T.4.3, and so the proof is omitted here.

**Lemma L.6.1.** Let \( M = (Q, \Omega_2, \text{TABL}, \text{ASSOC}) \) be a ppst. There is a weakly equivalent ppst, \( M' \), such that each row in \( \text{TABL}' \) has only a single target entry whose condition is "always".

**Proof:** Algorithm A.6.2 takes a ppst which has \( k \) rows
with multiple target entries and transforms it into a weakly equivalent ppst with \( k-1 \) rows with multiple target entries. \( k \) iterations of the algorithm reduces the original ppst to one with no multiple entry rows. Assume without loss of generality that \( M \) is in normal form.

**Algorithm A.6.2.**

**Purpose:** For \( k > 0 \), transforms a ppst which has \( k \) rows with multiple target entries into a weakly equivalent ppst which has only \( k-1 \) such rows.

**Input:** A ppst \( Q = (G, SEM) \).

**Output:** A ppst \( Q' = (G', SEM') \).

**Steps:**

1) Originally, let \( Q' \) be \( Q \).

2) Select any row \( j \) of TABLE with \( n > 1 \) target entries:

   \[
   d\emptyset<d_1..d_m> \quad t'\mid x! \text{ con}(l) \\
   \quad \quad \quad t'n \quad x_n \text{ con}(n)
   \]

3) Replace row \( j \) with \( n \) rows:

   \[
   d\emptyset<d\emptyset_1<d_1..d_m> \quad t'\mid x! \text{ "always"} \\
   d\emptyset<d\emptyset_2<d_1..d_m> \quad t'n \quad x_n \text{ "always"}
   \]

4) In the underlying grammar of \( Q' \), replace the production \( d\emptyset<d_1..d_m> \) by the \( 2n \) productions, \( d\emptyset<d\emptyset_i>, \ d\emptyset_i<d_1..d_m> \), for \( i \in \mathcal{D} \), used to form the source trees in step 3. Add \( d\emptyset_1, \ldots, d\emptyset_n \) to \( V_{n1} \).

5) Set \( \text{att}(d\emptyset_i) \) equal to \( \text{att}(d\emptyset) \) for \( i \in \mathcal{D} \).

6) Add a new attribute, "\text{proper}" , to \( \text{att}(d\emptyset_i) \) for \( i \in \mathcal{D} \).

7) Whenever production \( d\emptyset<d\emptyset_1> \) appears in a complete derivation, it must be the parent sub-structure of \( d\emptyset_1<d_1..d_m> \). Set up rules so that the values of the synthesized
attributes which would have been assigned to d₀ are instead assigned to d₀i. These values are then passed up to d₀ from d₀i. The values of the inherited attributes of d₀ are passed unchanged to d₀i. These values are then used to determine the values of any attributes of d₁,...,dm that were dependent upon d₀ in the original system. Proper(d₀i) has as its arguments, the attributes of d₀,...,dm. Its value is "err" if the predicate con(i), evaluated on these same arguments would not be satisfied, and is set to "ok" if con(i), evaluated on these same arguments would be satisfied.

M' = (Q₁',Q₂,TABL',ASSOC) is a ppst. We prove that M' is weakly equivalent to M.

Suppose TABL has z rows of which the one replaced in TABL is row j. Relabel the rows:

row₁, ..., row(j-1), row(j,1), ..., row(j,n), ...
rowz

We define a mapping from the trees in phr(Q₁') to those in phr(Q₁):

\text{struct:phr}(Q₁') \rightarrow \text{phr}(Q₁)

where

a) if t = d₀<d₁..dk> then struct(t) = t

b) if t = d₀<d₀i<d₁..dk> then struct(t) = d₀<d₁..dk>

c) if t = t₀[t₁..ta] where t₀ ∈ P₁, then struct(t) = struct(t₀)[ struct(t₁) .. struct(ta) ]

No other attribute is dependent upon "proper". From the way in which the semantic rules of Q₁' are specified, for any evaluation val- t' of t' ∈ phr(Q₁') there is an evaluation val- t of t = struct(t') which assigns the same value to the corresponding nodes of t and t'. Therefore, L(Q₁') is a subset of L(Q₁).

Let t ∈ phr(Q₁) with k occurrences of sub-structure d₀<d₁..dm>. There are k**n trees in struct-inv(t).
We next show that for every evaluation \( \text{val-}t \) of \( t \in \text{phrc}(QI) \) which induces transformation \( \text{TFP}(t) \), there is a \( t' \in \text{phrc}(QI') \) which has an evaluation which induces the same transformation, and conversely. From this we can conclude the \( M' \) and \( M \) are weakly equivalent.

**Case I:** \( \text{TFP}(t) \) is a subset of \( \text{TFP}'(t') \)

Consider the transformation of any \( t \in \text{phrc}(QI) \) for evaluation \( \text{val}-t \). For the transformation of each occurrence of \( \text{dm}<\text{dm}<\text{dm} \), a target tree \( t'i \) is selected for which \( \text{con}(i) \) is satisfied. There is some \( t' \in \text{struct-inv}(t) \) which parallels the selection of \( ti \) in the transformation by containing \( \text{dm}<\text{dm}<\text{dm} \) in place of \( \text{dm}<\text{dm}<\text{dm} \). This is true for each occurrence of \( \text{dm}<\text{dm}<\text{dm} \). \( \text{val}-t \) assigns each occurrence of "proper" the value "ok" in this case since \( \text{con}(i) \) is satisfied. Therefore, \( t' \) is not deviant, and \( \text{TFP}'(t) \) is a subset of \( \text{TFP}(t) \).

**Case II:** \( \text{TFP}'(t') \) is a subset of \( \text{TFP}(t) \)

The proof of this case is analogous to the proof of case i. Consider the transformation of any \( t' \in \text{phrc}(QI') \). Corresponding to the transformation of each occurrence of \( \text{dm}<\text{dm}<\text{dm} \), \( t'i \) is selected. Since \( t' \) is not deviant, \( \text{con}(i) \) must be satisfied for the attribute values of this occurrence of \( \text{dm}<\text{dm}<\text{dm} \) or else \( \text{val}-t'(\text{proper}(\text{dm}i)) \) would equal "err". Therefore, there is a parallel transformation of \( t = \text{struct}(t') \) for the corresponding evaluation \( \text{val}-t \) of \( t \) in which for the sub-structure \( \text{dm}<\text{dm}<\text{dm} \) corresponding to \( \text{dm}<\text{dm}<\text{dm} \), target tree \( t'i \) is selected. Hence \( \text{TFP}'(t') \) is a subset of \( \text{TFP}(t) \).

**Lemma L.6.2.** For each predicate transducer \( M \), there is a weakly equivalent incomplete transducer \( M' \).

**Proof:** Using L.6.1, \( M \) can be reduced to a post which has no multiple-entry rows. Then all conditions on the rows must be "always". The added power of semantic selection of corresponding target trees at transduction time has been rendered useless. This is weakly equivalent to the ipst formed by "chopping" the condition column off of the table; i.e., if
$M' = (Q1', Q2, TABL', ASSOC)$ is the eventual output of A.6.2, then $ipst M'' = (Q1, Q2, TABL'', ASSOC)$ is weakly equivalent to $M'$ where if

\[ t \cdot t' \cdot x \ "always" \]

is a row of $TABL'$, then

\[ t \cdot t' \cdot x \]

is a row of $TABL''$.

[*]

As the price paid for this conversion, the syntactic ambiguity of the source system is greatly increased. Most of the parses of the ipst constructed by A.6.1 will probably be deviant. When the ipst is actually used to transduce, these parses must be examined in order to find one or more that is not deviant. So a large percentage of the time spent on transducing a sentence could be spent simply trying to find a non-deviant parse. Therefore, it may be possible to build faster transducers using the ppst as the transduction model rather than the ipst. The equivalence between ppst and ipst is a weak one; i.e.

\[ ppst =_{w} ipst \]

The next lemma shows that in general this is the strongest claim that can, in general, be made; i.e.

\[ ipst <_{s} ppst \]

**Lemma L.6.3.** There is a predicate translator $M$ which has no strongly equivalent incomplete translator $M'$.  

**Proof:** The ppst is example pst-4 in Appendix D.

1) For any source phrase-structure in which $E$ is a frontier symbol, $E$ is an operand of either a multiplication or an addition operation.

2) Which target phrase-structure corresponds to the source phrase-structure in any row of $TABL$ depends upon the data-types of the two operands of the source phrase-structure.

3) Unless $E$ is terminated within phrase-structure $t$, the data-type of an occurrence of $E$ cannot be computed from syntactic considerations alone.
The source system generates all postfix expressions over +, 1, and 1.. Therefore, there must be at least one s-TABL entry in which one operand is not terminated; i.e., s-TABL must include at least one entry which looks like: $t_0[\ldots E<Ex?\ldots]$ or $t_0[\ldots E<xE?\ldots]$ where ? is either 0, 0<< or 0<< and x is either E<1>, E<1.> or E.

5) If $E<Ex?>$ is a frontier phrase-structure of some s-TABL entry, then the corresponding t-TABL entry must look like

1) $t_0'[\ldots ER<ER y ?\ldots]$.
2) $t_0'[\ldots ER<EI z ?\ldots]$.
3) $t_0'[\ldots EI<EI w ?\ldots]$

where y is either ER, EI, ER<1.>, $z$ is either ER or ER<1.>, and w is either EI or EI<1>, with the obvious association between nonterminals of the source and target entry. If (1) is selected, then translations in which the source data-type of the left operand is integer will be wrong. If (2) is selected, then translations in which the source data-type of the second operand is real will be wrong. Similarly, for (3). The case for $t_0[\ldots E<xE>\ldots]$ is analogous to that for $t_0[\ldots E<Ex>\ldots]$. Therefore, $M'$ does not exist.

Thm 6.2 As models of translation,

$$\text{ipst} \leq \text{ppst}$$

$$\text{ipst} \equiv \text{ppst}$$

Proof: By definition $\text{ipst} \leq \text{ppst}$. By L.6.2, $\text{ppst} \equiv \text{ppst}$. Therefore, $\text{ipst} \equiv \text{ppst}$. By L.6.3, $\text{ipst} \neq \text{ppst}$. Therefore, $\text{ipst} \leq \text{ppst}$.

The ppst provides a stronger motivation for introducing inherited attributes than previously presented. The predicate transducer derives its additional power over the incomplete transducer by using semantic information which is available "locally" - within the sub-structure being transformed. Having inherited attributes lets us send information generated "higher" in the tree, contextual information, to a point "lower" in the tree. This information would not be available if only synthesized
attributes were used. For example, to perform arithmetic data-type conversions in an assignment statement, the datatypes of the variables in the expression must be known. Suppose the declaration statements were located "higher" in the tree than where the arithmetic was being done. Without inherited attributes it would be necessary to "postpone" the conversions by passing the arithmetic information up the tree until it was as "high" as the declarations. Only at that point in the tree could the conversions be done, since only then would the necessary information for the conversions be known. This would, of course, be very unsatisfying both intuitively and linguistically.

An important advantage of using complete transducers over the other two models of phrase-structure transduction presented previously is that at no time in the transduction process do we use semantic information. Hence, when using a complete transducer to compute the transductions of sentence $w$, it is never necessary to compute mean($w$) or to test the parses of generated target strings for semantic deviancy. If a ppst is defined using complete source and target systems (call it a pcphst), the members of parse($w$) must be evaluated even though none of the parses are deviant. The evaluation is necessary in order to tell which conditions in the correspondence table are satisfied. By Th.6.2, we know that a pcphst may be algorithmically converted into a weakly equivalent ipst. However, if it could instead be converted into a weakly equivalent complete phrase-structure transducer, then the semantics could be totally ignored in the transduction process. Depending upon the complexity of the the semantic rules and conditions, this could result in a considerable savings in computation time. Unfortunately, as Th.6.3 proves, this conversion cannot always be done.

**Thm T.6.3.** $cpst \prec w pcptst$.

**Proof:** The pcptst which has no weakly equivalent cpst is example pst-5 in Appendix E. Leggett [74] (see Appendix H for proof) has shown that there is no finitely specified transducer as defined by Buttelmann, whose source and target systems are complete, which will translate from $L_1 =$

$$
\{(n,1) \mid n \text{ is a perfect square}\} \cup \\
\{(n,0) \mid n \text{ is not a perfect square}\}
$$

to $L_2 =$

$$
\{(0,0),(1,1)\}
$$
The complete translation is a special case of Buttelmann's finitely specified translation where the arguments of the latter translator are just complete source and target systems. Hence, there is no cpst which will induce a translation from L₁ to L₂. pst-5 translates from L₁ to L₂.

Thm T.6.4. The set of pairs of context-free languages which can be translated using an incomplete phrase-structure translator without discarding deviant parses is strictly smaller than the set of pairs of context-free languages which can be translated using an incomplete phrase-structure translator with a deviance check.

Proof: Let L₁ = \{(1*₂ₙ,n!) | n ∈ INT⁺\} and let L₂ = \{(1*ₙ,n) | n ∈ INT⁺\}. The translation of 1*₂ₙ is 1*ₙ(ₙ!). We show that no incomplete translator which does not check for deviance can induce this translation. the transformation step of the incomplete transducer is the same as that of the complete transducer. T.4.4 established that the pseudo-weight of the target tree is bounded by an integer constant raised to the power of the pseudo-weight of the source tree. Therefore, the same holds true for target trees induced by an incomplete transducer. Let w be any integer in L₁. It has a meaningful parse whose weight is bounded by a constant times the length of w. By T.4.4, the transform of this parse cannot have a frontier whose length is the factorial of |w|.

Complexity

The definition of honesty given in chapter iv may be extended without modification to the predicate phrase-structure translator. The execution time of an honest ppst differs from that of an honest ipst only by the extra time required to evaluate the conditions of the rows of TABL in the transformation step. Therefore, the analysis of the translation time of w in chapter v for the ipst is directly applicable here. The conditions in one row of TABL must be evaluated for each sub-structure of t ∈ parse(w). Let z be the maximum number of conditions in any row of TABL. The maximum number of conditions which must be evaluated is z*wt(t). If t has no linear recursive sub-structures, then wt(t) < kn where n = |w|, and k is an integer constant. If the maximum complexity of any condition is F, then the additional time required is less
than or equal to

\[ O(zknF(n)) \quad (6.1) \]

time steps. This reduces to

\[ O(nF(n)) \quad (6.2) \]

time steps. Time bound 6.2 should be added to times 5.4, 5.5, 5.6, and 5.7, respectively, to obtain the times for best case multi-linear, best case exponential, worst case multi-linear, and worst case exponential analysis of the predicate phrase-structural translation. This gives bounds 6.3, 6.4, 6.5, and 6.6, respectively.

\[ O(n^{**3} + nF(n)) \quad (6.3) \]

\[ O(k^{**n} + nF(n)) \quad (6.4) \]

\[ O(n^{**3} + pn + pnF(n)) \quad (6.5) \]

\[ O(k^{**n} + pn + pnF(n)) \quad (6.6) \]

**Summary**

The **predicate phrase-structure transducer** was defined and analyzed. It differs from the incomplete transducer only in that semantic information in the form of predicates are used to direct the transformation of the parse of the source sentence. We stated two sufficient conditions for a predicate transducer to be semantic preserving and proved their sufficiency. Next we showed that any predicate transducer can be algorithmically converted into an incomplete transducer in such a manner as to preserve the "overall structure" of both the syntactic and semantic components of the original predicate transducer. The time required for a subclass of predicate translators to execute was then analyzed. The main result we showed is that there is an apparent trade-off between the degree of syntactic ambiguity in the source system of a transducer and the power of the transducer itself; i.e., by introducing a higher degree of syntactic ambiguity into the source system, the incomplete transducer can be substituted for the predicate transducer.
There is every reason to believe that within the next two decades machines will be available outside the laboratory that will do a credible job of original thinking... There is no basis for knowing where this process will stop, nor, as Wiener has pointed out, is there any comfort in the assertion that, since man built the machine, he will always be smarter or more capable than it is.

Donald Michael - 1962

Introduction

In one row of TABL in an incomplete transducer, the single s-TABL entry always has a single corresponding t-TABL entry. In this chapter we investigate a phrase-structure transducer whose TABL rows have multiple source and multiple target entries. The set of s-TABL and t-TABL entries correspond as a block. Intuitively, a set of physically isolated source sub-structures are treated as a unit for transformation purposes. They are transformed into a block of physically isolated target sub-structures. The block phrase-structure transducer, unlike the predicate transducer, uses no semantic information in the transformation process. This extension is purely syntactic in nature.

Basic Definitions

**Defn D.7.1.** A block phrase-structure transducer (bpst) is a 4-tuple \( M = (Q_1, Q_2, \text{TABL, ASSOC}) \) where:

1) \( Q_1 \), the **source system**, is a pss.
2) \( Q_2 \), the **target system**, is a pss.
3) \( \text{ASSOC} : \text{powerset}(V_{n1}) \rightarrow \text{powerset}(V_{n2}) \rightarrow \emptyset \) is a partial function such that
\( \text{Ax1} \in X \in \text{domain}(\text{ASSOC}) \implies X = \{\text{Ax1}\} \) and
\( \text{ASSOC}(X) = \{\text{Ax2}\} \).
\( \text{Ax2} \in Y \in \text{range}(\text{ASSOC}) \implies Y = \{\text{Ax2}\} \) and
\( \text{ASSOC}^{-1}(Y) = \{\text{Ax1}\} \).

4) \( \text{TABL} \) is a finite non-empty table. The \( i \)-th row has four entries:
\[
(t_1, \ldots, t_n) \quad (t'_1, \ldots, t'_m) \quad x \quad x'
\]
where
- a) \( t_i \in \text{phr}(\text{UG}(C_1)) \cup V_{n1} \), where at least one \( t_i \) has height one, and no two trees have a root labeled by the same nonterminal.
- b) \( t'_i \in \text{phr}(\text{UG}(C_2)) \).
- c) \( \text{ASSOC}((\text{label}(\text{rt}(t_1)), \ldots, \text{label}(\text{rt}(t_n)))) = (\text{label}(\text{rt}(t'_1)), \ldots, \text{label}(\text{rt}(t'_m))) \)
- d) Suppose \( f_1 \ldots f_c = \text{fr}(t_1) \ldots \text{fr}(t_n) \) and \( g_1 \ldots g_d = \text{fr}(t'_1) \ldots \text{fr}(t'_m) \). For some \( k \geq 1 \), \( x \) is a \( k \) by \( z \) binary matrix, and \( x' \) is a \( k \) by \( z' \) binary matrix. For \( i \in \mathbb{Z} \), if \( \text{label}(f_j) \in V_{t1} \), then \( x(i, j) = 0 \).
  For \( i \in \mathbb{Z}' \), if \( \text{label}(g_j) \in V_{t2} \), then \( x'(i, j) = 0 \). If \( \text{label}(g_j) \in V_{n2} \), then the \( j \)-th column of \( x' \) has a single non-zero entry. For \( i \in \mathbb{Z} \), if the \( i \)-th row of \( x' \) has a non-zero entry, then the \( i \)-th row of \( x \) has a non-zero entry.
- e) If \( F(i) = \{\text{label}(f_j) | x(i, j) = 1\} \), and \( \text{G}(i) = \{\text{label}(g_j) | x'(i, j) = 1\} \)
  then \( \text{ASSOC}(F(i)) = \text{G}(i) \).

Suppose \( \text{TABL} \) has \( j \) rows and row \( i \) is
\[
(t_1, \ldots, t_m) \quad (t'_1, \ldots, t'_n) \quad x \quad x'
\]
Four sets are defined:
- \( s-\text{TABL}(i) = (t_1, \ldots, t_m) \)
- \( t-\text{TABL}(i) = (t'_1, \ldots, t'_n) \)
s-TABL is the ordered union of
\[ \{ t \mid t \in s-\text{TABL}(i) \text{ for } i \in \mathbb{I} \} \]
t-TABL is the ordered union of
\[ \{ t \mid t \in t-\text{TABL}(i) \text{ for } i \in \mathbb{I} \} \]

In all previous phrase-structure transducers introduced in chapters iv, v and vi, a lone target nonterminal B always was associated with a single source nonterminal A. This meant that in a vague sense, A and B served the same "purpose" in their respective systems. For example, in pst-1 in Appendix A, the source axiom, S, and the target axiom, S, both may root arbitrary arithmetic expressions. The former roots postfix expressions and the latter roots infix expressions. This was reflected in the fact that single phrase-structures rooted by A were set in correspondence with single phrase-structures rooted by B in the rows of TABL. In the bpst we will examine transductions in which it is no longer possible to isolate a single source and single target nonterminal as serving equivalent purposes in their respective systems. Rather, it will be necessary to associate a set of source nonterminals and a set of target nonterminals. Individually, no associations between source and target nonterminals will adequately indicate the correspondence. Because of this, a set of phrase-structures rooted by a set of nonterminals \( \{A_1, \ldots, A_n\} \) in domain(ASSOC) is set in correspondence with a set of phrase-structures rooted by ASSOC(\( \{A_1, \ldots, A_n\} \)). In the bpst, the format of the \( \mathbf{x} \)-vector has been changed. The motivation for introducing binary matrices \( \mathbf{x} \) and \( \mathbf{x}' \) must await the definition of the phrase-transform of a bpst.

**Defn D.7.2.** Let \( \mathcal{M} \) be a bpst. Suppose \( p \in \text{phra}(Q_1) \) and \( Z \) is a lmpd of \( p \) with respect to \( s-\text{TABL} \):

\[ (p_1,n_1,m_1), \ldots, (p_z,n_z,m_z) \]

The phrase-transform, \( \text{TFP}_{\text{phra}}(Q_1) \times \text{LMD}(s-\text{TABL}) \rightarrow \text{phra}(Q_2) \), is a relation computed by algorithm A.7.1.

**Algorithm A.7.1.**

**Purpose:** Compute the phrase-transform of cpst \( \mathcal{M} \).
**Input:** p ∈ phra(G1) and Z, a leftmost partial derivation of p with respect to s-TABL.

**Output:** A finite set of trees in phra(G2).

**Steps:**

1) Initially, TFP(p, Z) = p', a node labeled by Ax2. p' is said to be associated with rt(p). We write this as assoc(p') = rt(p).

2) Suppose

   a) p' = TFP(p, Z).
   b) (q_1, ..., q_s) = (q_1', ..., q'_s') x x' is a row of TABL.
   c) For each i ∈ s, there is a j ∈ z such that q_i = p_j. Call them (p(i, 1), ..., p(i, s)).
   d) There is a set of nodes (c_1, ..., c_s') at fr(p', y_1), ..., fr(p', y_s'), respectively, such that

   \[ \text{assoc}((c_1, ..., c_s')) = (rt(p(i, 1)), ..., rt(p(i, s))) \]

   where j > k => y_j > y_k.

Then TFP(p, Z) =

\[ \text{compose}(*, q'_1, y_1) \# \ldots \# \text{compose}(*, q'_s, y_s')(p') \]

Let fr-p =

\[ fr(p(i, 1)) \# \ldots \# fr(p(i, s)) \]

the phrase-structures of p chosen in (2c), and let fr-r =

\[ fr(r_1) \# \ldots \# fr(r_s') \]

where r_i is the occurrence of q'_i just composed to p'. Suppose x is a k by z matrix and x' is a k by z' matrix. For i ∈ k,
assoc(∪(fr-r(j) | x'(i,j) = 1, j ∈ z')) =
∪(fr-p(j) | x(i,j) = 1, j ∈ z)

TFP(p) = (TFP(p,Z) | Z is a lmpd of p).

[*] Because a more elaborate procedure is necessary to
define TFP here than is required of earlier models of
phrase-structure transducer, it may not be clear that TFP,
as stated in D.7.2, is well-defined. We prove that this is
the case in the next lemma.

**Lemma L.7.1.** Let M = (Q1,Q2,TABL,ASSOC) be a bpst. TFP(M)
is well-defined.

**Proof:** If at some point in the transformation process
more than one row of TABL is applicable at step 2c,
then the choice of which row to apply is arbitrary.
TFP is well-defined if the order in which the
applicable rows are selected does not effect which tree
is eventually output as the transformation of the
source tree.

Let row i and row j of TABL both be applicable at
the same time, where row i is

(wl<..> .. wc<..>) (ul<..> .. ud<..>) x1 x1'

and row j is

(yl<..> .. yd<..>) (vl<..> .. vb<..>) x2 x2'

Let the occurrences of wl<..> .. wc<..> in source tree
t be Wl<..> .. Wc<..> and let the occurrences of
yl<..> .. yd<..> be Yl<..> .. Yd<..>. Let the
occurrences of rt(ul<..>) .. rt(ud<..>) in
t' = TFP(t,Z) at this point of the transformation be
U1 .. Ud and let the occurrences of
rt(vl<..<>) .. rt(vb<..<>) be V1 .. Vb.

Because each column in x1', x2' and all other x'
vectors in TABL are allowed to have only a single 1 in
each column, a node in t' can belong to one 1 set of
nodes in t' which is associated with some set of nodes
in t. Therefore,

(U1,..Ud) ∩ (V1,..,Vb) = Φ

Let the terminated trees rooted by each Wl be denoted
WWl and the terminated tree rooted by each Yi be
denoted by \( Y_1 \). Because \((U_1, \ldots, U_d) \cap (V_1, \ldots, V_b)\) is empty, it is true that nodes(TFP(\#W_1, \ldots, \#W_c)) int nodes(TFP(Y_1, \ldots, Y_d)) is also empty. Therefore, the order in which the transformation steps are applied does not affect which target tree is output as the transformation of \( T \).

[\*]

The string-transform of a bpst is defined in exactly the same manner as for an ipst.

**Defn D.7.3.** Let \( M \) be a bpst. Define the string-transform of \( M \), \( \text{TFS} : \text{sen}(Q_1) \rightarrow \text{sen}(Q_2) \) by \( \text{TFS}(w) = \)

\[
\{ t' | t \in \text{parse}(w), t \in \text{phrc}(Q_1), t' \in \text{TFP}(t), t' \in \text{phrc}(Q_2) \}
\]

[\*]

**Example E.7.1.** Consider example pst-8 in Appendix G. Suppose the source sentence is:

\[
\begin{align*}
C &= 2, \\
\text{DCL} \ A \ &\text{INTEGER} \\
A &= 4 + C \\
\text{DCL} \ C \ &\text{INTEGER} \\
\text{STOP}
\end{align*}
\]

Appendix G contains a detailed step-by-step look at how the transduction is performed to yield:

\[
\begin{align*}
&\text{INTEGER} \ A \\
&\text{INTEGER} \ C \\
&\text{C} = 2. \\
&A = 4 + C \\
&\text{END}
\end{align*}
\]

[\*]

**Translation**

T.7.1 states a condition which is sufficient to ensure that a transducer is semantic preserving. But before we can present this theorem, we must revise the definitions of **normal form** and **restriction** which have been used in the last three chapters. The original definitions are applicable
only for phrase-structure transducers which set single
source and single target phrase-structures in correspondence
with each other.

**Defn D.7.4.** A bpst is in normal form if for each row
\[(p_1, \ldots, p_m) \times (p'_1, \ldots, p'_n)\]
each pi either has height 0 or 1 and each p'_j has height 1.

The proof that every bpst can be put into normal form
is very similar to that given for T.4.3, and so will not be
given here. Source trees which have zero height in the
original transducer are left unchanged, while trees with
height greater than one are shrunk to unit height using the
method explicated in T.4.3.

In a row of TABL, there is no meaningful correspondence
between the individual trees in each s-TABL and t-TABL
entry. However, in certain cases, it may be possible to
isolate corresponding attributes of source and target trees
in the row entries.

**Defn D.7.5.** Let M be a bpst. Suppose that for all
\{(v_1, \ldots, v_m)\} in domain(ASSOC),
\[ASSOC((v_1, \ldots, v_m)) = (v'_1, \ldots, v'_n)\]
implies that for distinct i and j,
\[\text{att}(v_i) \cap \text{att}(v_j) = \emptyset \text{ att}(v'_i) \cap \text{att}(v'_j) = \emptyset\]
Suppose that
\[(p_1, \ldots, p_m) \times (p'_1, \ldots, p'_n) \times x'\]
is a row of TABL. Let c' be a root or frontier node of some
p'i' (call it p'a') with attribute attu.

Define the partial function "corr":
\[\text{corr: att-p' } \rightarrow \text{ att-p}\]
where
\[\text{att-p} = (\text{att-p}(p_i) : i \in m)\]
and where
\[\text{att-p'} = (\text{att-p'}(p'_i) : i \in n)\]
If \( \text{label}(c') \in V_{n2} \) and \( \text{attu} \in \text{att}(\text{label}(c')) \), then \( \text{corr}(\text{attu}(c')) \) is defined, and its value is determined as follows:

**Case I:** \( c' = \text{rt}(p'a') \)

There is a unique \( c \), the root of some \( p_j \) with attribute \( \text{attu} \). Let \( \text{corr}(\text{attu}(c')) = \text{attu}(c) \). We say that \( \text{attu}(c) \) "corresponds to" \( \text{attu}(c') \).

**Case II:** \( c' \in \text{fr}(p'a') \)

Suppose \( x \) and \( x' \) have \( k \) rows. If \( \text{label}(c') \in V_{n2} \), then for exactly one \( i \in k \), \( c' \in G(i) \). There is a unique \( c \in F(i) \) on the frontier of some \( p_j \) with attribute \( \text{attu} \). Define \( \text{corr}(\text{attu}(c')) = \text{attu}(c) \). We say that \( \text{attu}(c) \) "corresponds to" \( \text{attu}(c') \). If \( \text{label}(c') \in V_{t2} \), then \( \text{corr}(\text{attu}(c')) \) is not defined.

[*]

The definition of "restriction" which we used in the last three chapters is not adequate for block transducers and must be revised. This revision is stated as D.7.6.

**Defn D.7.6.** Let \( M \) be a bpst. Suppose that for all \( \langle v_1, \ldots, v_m \rangle \in \text{domain}(ASSOC) \)

\[
\text{ASSOC}((v_1, \ldots, v_m)) = (v'1, \ldots, v'n)
\]

implies that for distinct \( i \) and \( j \),

\[
\text{att}(v_i) \cap \text{att}(v_j) = \text{att}(v_i') \cap \text{att}(v_j') = \emptyset
\]

Suppose that

\[
\langle p_1, \ldots, p_m \rangle \ (p'1, \ldots, p'n) \ x \ x'
\]

is a row of TABL. Let \( c' \) be a root or frontier node of some \( p'j \) (call it \( p'a' \)) with attribute \( \text{attu} \) such that \( \text{corr}(\text{attu}(c')) = \text{attu}(c) \). Suppose \( c \) is in tree \( pa \). Let \( \text{rule1} \) and \( \text{rule2} \) either be \( \text{rule-\#}(\text{attu}(c), pa) \) and \( \text{rule-\#}(\text{attu}(c'), p'a') \) or \( \text{rule}(\text{attu}(c), pa) \) and \( \text{rule}(\text{attu}(c'), p'a') \), respectively, where

- **rule1:** \( \text{attu}(c) = f(\text{attu}(c1), \ldots, \text{attu}(cy)); \)  
  \( Dl \ldots xDy \rightarrow D \)

- **rule2:** \( \text{attu}(c') = g(\text{attu}(c'1), \ldots, \text{attu}(z'(c'z))); \)  
  \( Cl \ldots xCz \rightarrow C \)
rule1 is a restriction of rule2 (rule1 ≤r rule2) if

1) f is a restriction of g; and

2) corr(attui(c'i)) = attui(ci); or
corr(attui(c'i)) is not defined.

If S and U are sets of semantic rules or their closure such that there is a bijection from the members of S to the members of U where biject(s) = u implies s ≤ u, then S is a restriction of U (S ≤r U).

Lemma L.7.2. Let M be a bpst. Suppose that for each row of TABL

\[(t_1, \ldots, t_m) \leq \leq (t'_1, \ldots, t'_n) \times x \times x' \]

\[\text{rule-*(}(t_1, \ldots, t_m)\leq \leq \text{rule-*(}(t'_1, \ldots, t'_n))\] Suppose p ∈ phraQL), Z is a lmd of p with respect to s-TABL, p' = TFP(pZ), Z' is the lmd of p' induced by (pZ), and Z is a lmd of supertree p of p. Suppose c', with exit attribute attu, belongs to nodes(p'). If

corr(attu(c')) = attu(c), then

\[\text{rule-*(}(attu(c),p_0) \leq \leq \text{rule-*(}(attu(c'),p') \]

Proof: By induction on the number of triples in Z. Suppose Z =

\[(p_1,n_1,m_1), \ldots, (p_z,n_z,m_z)\]

Assume M is in normal form.

Base Step: z = 1
1) Assume Z has one triple.
2) If any row of TABL is applicable, it looks like:

\[q \quad q' \quad x \quad x' \]

Then q = p_0 and q' = p'. By hypothesis,

\[\text{rule-*(}(attu(c),q) \leq \text{rule-*(}(attu(c'),q') \]
Therefore,
\[ \text{rule-}\star(a(\text{ttu}(c)),p_0) \leq \text{rule-}\star(a(\text{ttu}(c')),p') \]

**Induction Step:** \( z = k \)

3) Assume that the lemma is true for all \( Z \) with less than \( k \) triples.

4) Suppose \( p \in \text{phra}(Q_1) \) such that \( Z \) has \( k \) triples. We will examine the relationship between \( p \) and \( p' \) just before the last step of the transformation which defines \( p' \). \( p_0 \) can be rewritten as \( r_0[\text{rl},..,\text{ry}] \) where

a) \( r_0 \in \text{phra}(s-\text{TABL}) \) has a lmd \( Y \) which is a lmpd of \( p_0 \) according to \( Z \). \( Y \) has \( < k \) triples.

b) Let \( E = (r_i \mid \text{wt}(r_i) = 1 \text{ for } i \in \chi) \).

c) \( p' \) has a box bracketing, \( r_0[r'_1...r'_y] \).

d) Let \( E' = (r'_i \mid \text{wt}(r'_i) = 1 \text{ for } i \in \chi') \).

e) There is a subset \( A \) of \( \{r(t(r_i)) \mid i \in \chi\} \) which contains \( \{r(t(r_i)) \mid r_i \in E\} \), such that \( A \) is associated with a subset \( A' \) of \( \{r(t(r_i)) \mid i \in \chi'\} \) which contains \( \{r(t(r_i)) \mid r_i \in E'\} \).

f) There is a row of \( \text{TABL} \)

\[ (q_1,...,q_m) \ (q'_1,...,q'_n) \ x \ x' \]

such that \( \text{label}(A) = (r(t(q_i)) \mid i \in \chi \) and \( \text{label}(A') = (r(t(q'_i)) \mid i \in \chi'). \)

g) \( E = (q_i \mid \text{wt}(q_i) = 1) \).
\( E' = (q'_i \mid \text{wt}(q'_i) = 1) \).

5) \( a(\text{ttu}(c')) \) is an exit attribute of \( p' \) or \( c' \) is either \( r(t(p')) \) or a member of \( \text{fr}(p') \).

6) For a unique \( c \in \text{nodes}(p_0) \), \( \text{corr}(a(\text{ttu}(c')) \) is \( a(\text{ttu}(c)) \). By hypothesis, \( \text{rule-}\star(a(\text{ttu}(c)),r_2) \leq \text{rule-}\star(a(\text{ttu}(c')),r') \). Let \( a(\text{ttu}(c_1),...,a(\text{ttu}(c_v)) \) be the arguments of \( \text{rule-}\star(a(\text{ttu}(c)),r_0) \) and \( a(\text{ttu}(c_1'),...,a(\text{ttu}(c_v')) \) be the arguments of \( \text{rule-}\star(a(\text{ttu}(c'),r') \). Either \( \text{corr}(a(\text{ttu}(c')) \) is \( a(\text{ttu}(c_i), \text{corr}(a(\text{ttu}(c'_i))) \) is not defined. The arguments of \( \text{rule-}\star(a(\text{ttu}(c)),r_0) \) are all
entrance attributes of $r_0$, and all of the arguments of \( \text{rule-} \ast \text{attu}(c'), r_0' \) are entrance attributes of $r_0'$. If all of the entrance attributes in the arguments of \( \text{rule-} \ast \text{attu}(c), p_0 \) and \( \text{rule-} \ast \text{attu}(c'), p' \) are also entrance attributes of $r_0$ and $r_0'$, then the theorem is true.

7) Suppose \( \text{attui}(c_i) \) is not an entrance attribute of $r_0$. Then it must be a root exit attribute of some $r_i \in E$. Call it $r_a$. Similarly, \( \text{attui}'(c'_i) \) must be a root exit attribute of some $r'_i \in E'$. Call it $r_a'$. \( \text{corr}(\text{attui}'(c'_i)) = \text{attui}'(c'_i) \). $r_a$ is equal to some member of \( \{t_1, \ldots, t_m \} \) and $r_a'$ is equal to some member of \( \{t'_1, \ldots, t'_n \} \). By hypothesis,

\[
\begin{align*}
\text{rule-} \ast (\text{attui}(rt(ra)), ra) &< r \\
\text{rule-} \ast (\text{attui}'(rt(r'a')), r'a')
\end{align*}
\]

8) Substitute \( \text{rule-} \ast (\text{attui}(rt(ra)), ra) \) for \( \text{attui}(c_i) \) in \( \text{rule-} \ast (\text{attui}(c), r_0) \) and substitute \( \text{rule-} \ast (\text{attui}'(rt(p'a'), p'a')) \) for \( \text{attui}'(c'_i) \). Call the rules before the substitution "first generation" and those after the substitution "second generation". The second generation of \( \text{rule-} \ast (\text{attui}(c), r_0) \) is a restriction of the second generation of \( \text{rule-} \ast (\text{attui}(c'), r_0') \).

9) The arguments of \( \text{rule-} \ast (\text{attui}'(c'_i), r'a') \) are either entrance attributes of $r_a'$, or attributes such that \( \text{corr}(\text{attui}'(c'_i)) \) is not defined. If none of the arguments have a corresponding attribute in $p_0$, or are all entrance attributes with corresponding attributes in $p_0$, then

\[
\begin{align*}
\text{rule-} \ast (\text{attu}(c), r_0) &= \text{rule-} \ast (\text{attu}(c), p_0) \\
\text{rule-} \ast (\text{attu}(c'), r_0') &= \text{rule-} \ast (\text{attu}(c'), p')
\end{align*}
\]

and hence, the theorem follows. So assume not. Then the arguments must be exit attributes of $r_0'$. We can thus continue the substitution stated above with third, fourth, ..., generation versions of \( \text{rule-} \ast (\text{attu}(c), r_0) \) and \( \text{rule-} \ast (\text{attu}(c'), r_0') \). Since $Q_1$ and $Q_2$ are proper, the set of attributes on which any attribute is dependent is finite and non-circular. Therefore, there is a maximum generation for \( \text{rule-} \ast (\text{attu}(c), r_0) \) and \( \text{rule-} \ast (\text{attu}(c'), r_0') \). These are
rule-*(attu(c),p0) and rule-*(attu(c'),p'). The arguments for rule-*(attu(c'),p') must either be attributes with corresponding attributes in rule-*(attu(c),p0), or the arguments of rule-*(attu(c'),r'0) must have no corresponding argument. Hence,

\[ \text{rule-*(attu(c),p0) \leq rule-*(attu(c'),p')} \]

[\star]

Thm T.7.1. Suppose M is a bpst. If for each row of TABL

\[(t_1, \ldots, t_m) \leq (t'_1, \ldots, t'_n) \]

\[\text{rule-*(} (t_1, \ldots, t_m) \text{)} \leq \text{rule-*(} (t'_1, \ldots, t'_n) \text{)}\],

then M induces a translation.

Proof: Suppose \( w \in \text{sen}() \), \( p \in \text{parse}(w) \), \( p \) has non-deviant evaluation \( \text{val-}p \), \( Z \) is a m1mpd of \( p \) with respect to \( s\text{-TABL} \), \( p' = \text{TFP}(p,Z) \), and \( p' \in \text{phrc(Pr2)} \). Let \( c = \text{rt}(p) \) and \( c' = \text{rt}(p') \).

By L.7.2, for each exit attribute, \( \text{attu} \), of \( c \),

\[\text{rule-*(} (\text{attu}(c),p0) \text{)} \leq \text{rule-*(} (\text{attu}(c'),p') \text{)}\].

Since \( p' \) is complete, all of its entrance attributes are attributes of terminal symbols. Therefore, the values of the entrance attributes of \( p' \) are all finite sets. Since \( \text{rule-*(} (\text{attu}(c),p0) \text{)} \leq \text{rule-*(} (\text{attu}(c'),p') \text{)}\), the set of possible values of the \( i \)-th argument of \( \text{rule-*(} (\text{attu}(c'),p') \text{)} \) is a subset of the set of possible values of the \( i \)-th argument of \( \text{rule-*(} (\text{attu}(c),p) \text{)} \). Therefore, there is an evaluation, \( \text{val-}p' \), of \( p' \) such that \( \text{val-}p(\text{attu}(c)) \) is equal to \( \text{val-}p'(\text{attu}(c')) \).

[\star]

Corollary C.7.1. Suppose M is a bpst. If for each row of TABL

\[(t_1, \ldots, t_m) \leq (t'_1, \ldots, t'_n) \]

\[\text{rule}((t_1, \ldots, t_m)) \leq \text{rule}((t'_1, \ldots, t'_n))\],

then M induces a translation.

Proof:

\[\text{rule}((t_1, \ldots, t_m)) \leq \text{rule}((t'_1, \ldots, t'_n)) \]

\[\Rightarrow \]

\[\text{rule-*(} (t_1, \ldots, t_m) \text{)} \leq \text{rule-*(} (t'_1, \ldots, t'_n) \text{)}\]

[\star]
Another condition which is sufficient to ensure the correctness of a predicate transducer is given as C.7.2. It is a natural extension of T.7.1. TFP can easily be extended to be defined over sets of trees, rather than just axiomatic trees of phr(GI). We leave the details to the reader.

Defn D.7.7. Suppose M is a bpst.

1) Suppose 
\((t_1, \ldots, t_n) \wedge (t'_1, \ldots, t'_m) \wedge x'\)

is row y of TABL. Let 

\(f_1 \ldots f_r \wedge g_1 \ldots g_s\)

be the frontiers of \(t_1 \ldots t_n\) and \(t'_1 \ldots t'_m\), respectively.

2) Suppose x is a k by z matrix and \(x'\) is a k by \(z'\) matrix. Select a row of \(x\), call it row \(k'\). Define two sets \(F^{**}\) and \(G^{**}\) as follows:

a) \(F^i = F(k') \wedge G^i = G(k')\).

b) For \(v, u \in k\), if \(x(v, j) = 1, x(u, j) = 1\) and \(f v \in F^i\) then \(F^{i+1} = F^i \cup F^u\) and \(G^{i+1} = G^i \cup G^u\).

c) Repeat step (b) until \(G^{i+1} = G^i \wedge F^{i+1} = F^i\) 
Call these sets \(G^{**}\) and \(F^{**}\), respectively.

3) Define the penultimate extension, \(PE((t_1, \ldots, t_n))\) of \((t_1, \ldots, t_n)\) as 

\(\{(t^{"1}, \ldots, t^{"n}) \mid \text{for } i \in n,\)
\(t^{"i} = t_{i}[f(i)][u(i)] \ldots f(i)[u(i)]\)

where if \(f(ij) \in F^{**}\), then \(u(ij)\) is terminated else \(u(ij) = f(ij)\).

4) Define a penultimate extension, \(PE(M)\), of \(M\) as 

\((Q_1, Q_2, PE(TABL), ASSOC)\)

where \(PE(TABL)\) is TABL except that for row \(y\) make the following substitution:

\(\{(t^{"1}, \ldots, t^{"n}) \wedge TFP((t^{"1}, \ldots, t^{"n}) \wedge x''\)
\(x''\mid (t^{"1}, \ldots, t^{"n}) \in PE((t_1, \ldots, t_n))\)
where \( x'' = x \) except that for \( f(i) \in F^* \), \( x(i,j) = 0 \) for \( j \in Z \) and \( x'' = x' \) except that for \( g(i) \in C^* \), \( x'(i,j) = 0 \) for \( j \in Z' \).

5) If \( M'' \) is a penultimate extension of \( M \), then \( (M'')'' \), the penultimate extension of \( M'' \), is also a penultimate extension of \( M \). It is formed by the same process as was \( M'' \) from \( M \). This process may be repeated arbitrarily often until all phrase-structures in the correspondence table are terminated. The output of each iteration is called a penultimate extension of \( M \). Note that every time a set of rows from the penultimate extension of \( (t_1, \ldots, t_n) \) is substituted for \( (t_1, \ldots, t_n) \), the number of nonterminals on the frontiers of each member of \( PE((t_1, \ldots, t_n)) \) decreases over the number of nonterminals on the frontier of \( (t_1, \ldots, t_n) \). Hence, the process must terminate. Indeed, when terminated, \( s\-TABL = domain(TFP) \) and \( t\-TABL = range(TFP) \).

[\*]

The reader should observe that if \( M \) is a bpst, then the \( TABL'' \) of some \( PE(M) \) could have an infinite number of rows. Hence, \( PE(M) \) is not strictly a bpst. However, we can extend in a natural way all of the definitions given for bpst to \( PE(M) \). The method is straightforward and so will not be given here. It is also immediately obvious that \( TFP(M) = TFP(M'') \) for any penultimate extension, \( M'' \), of \( M \).

**Corollary C.7.2.** Let \( M \) be a bpst. If for each row of \( PE(TABL) \) in some \( PE(M) = (Q1, Q2, PE(TABL), ASSOC) \),

\[ (t_1, \ldots, t_r) \quad (t'_1, \ldots, t'_s) \quad x \quad x' \]

\[ \text{rule-}^*((t_1, \ldots, t_r)) \leq_r \text{rule-}^*((t'_1, \ldots, t'_s)) \]

then \( M \) induces a translation.

**Proof:** If \( PE(M) \) induces a translation, then so must \( M \), because \( TFP(M) = TFP(PE(M)) \). The proof of L.7.1 did not rely in any way upon \( TABL \) having a finite number of rows. Therefore, the same reasoning which was used to prove that lemma for a table with a finite number of rows extends immediately to a table with an infinite number of rows.

[\*]
This immediately leads to one more corollary analogous to C.7.1.

Corollary C.7.3. Let \( PE(M) \) be a penultimate extension of a bpst \( M \). If for each row of \( PE(TABL) \)
\[
\{t_1, \ldots, t_r\} \times x \xrightarrow{\text{rule}} \{t_1', \ldots, t_s'\}
\]
rule\((\{t_1, \ldots, t_r\}) \preceq \text{rule}((t_1', \ldots, t_s'))\), then \( M \) induces a translation.

Proof:

\[
\text{rule}((t_1, \ldots, t_n)) \preceq \text{rule}((t_1', \ldots, t_m)) \implies \text{rule-}^*((t_1, \ldots, t_n)) \preceq \text{rule-}^*((t_1', \ldots, t_m))
\]

[*]

Relationship to CPST

In this section we illustrate the significance of the block transducer by showing that a complete transducer is not as powerful as a block transducer whose source and target systems are complete (call such a block transducer a "bcpst"). Since neither a cpst nor a bcpst makes use of deviance to filter parses, any extension in power of the bcpst over the cpst must be due entirely to the reformatting and modified use of the correspondence table.

Thm T.7.2. \( \text{cpst} \ll \text{bcpst}; \text{cpst} \ll \text{s bcpst} \).

Proof: Define a homomorphism

\[
h : (a, b)^+ \rightarrow (a', b')^+
\]
where \( h(a) = a' \) and \( h(b) = b' \). If \( w_1 \ldots w_Z \in (a, b)^+ \), for \( Z \geq 2 \), then \( h(w_1 \ldots w_Z) = h(w_1) \ldots h(w_Z) \). Consider the translation from \( L_1 = \{(w, w) \mid w \in (a, b)^+\} \) to \( L_2 = \)

\[
\{(ww', w) \mid w \in (a, b)^+, w' = h(w)\} \cup \{(ww', *) \mid w \in (a, b)^+, w' \neq h(w)\}
\]

This translation, call it \( \text{TRANS} \), is induced by block translator \( \text{pst-10} \) in Appendix I. Suppose there were a cpst \( M \) which could induce this translation. Note that \( \text{TRANS} \) is 1-1 and that \( M \) cannot be a strictly exponential cpst since \( |\text{TRANS}(w)| = 2|w| \). Therefore, \( M \) is a multi-linear cpst. It is straightforward to show that \( M \) does not satisfy T.4.7, the "pumping" theorem.
for multi-linear cpst. Hence, it is not a multi-linear cpst, and there is no cpst which induces TRANS. Therefore, cpst <w bcpst which implies that cpst <s bcpst.

[\star]

**Complexity**

In this section we analyze the execution time of an honest block translator. It is more difficult to perform the analysis on a bpst than any of the previously presented transducers. The reason for this added difficulty is the greater complexity of each transformation step of a bpst.

Defn D.7.8. A.7.1 transforms a set of source sub-structures at each step. Suppose that in step S1 and step S2, sets of trees T1 and T2 are transformed. We say that T1 and T2 are connected if the root of each tree in T2 is the frontier node of some tree in T1. We say that a series of steps S1, \ldots, S_k, which transform sets of trees T1, \ldots, T_k, form a path if for each pair of steps S_i, S_{(i+1)}, such that 1 \leq i < k, Ti and T_{(i+1)} are connected.

[\star]

Thm T.7.3. Let M be a bpst. There are integer constants y and z such that if t \in \text{phra}(Q1), Z is a lmpd of t, and t' = TFP(t, Z), then

\[ \text{pwt}(t) = j \implies \text{pwt}(t') \leq (yz)^*j \]

**Proof:** Assume without loss of generality that M is in normal form. Consider any path in the transformation of t. In each s-TABL entry, there is one or more trees with unit weight. Therefore, no two distinct steps of the transformation along the same path can transform sub-structures rooted by exactly the same set of nodes of t. Therefore, the largest number of steps possible in a single path is equal to the weight of t. Suppose nodes C = \{c_1, \ldots, c_k\} are the roots of sub-structures \{p_1, \ldots, p_k\} which are transformed during step S_i. There may be several sets of nodes in t' which are associated with C, but each set of such nodes can contain no more than y' trees, where y' is some constant for M. In step S_i, each set of nodes associated with C will, therefore, result in the composition of y' or fewer trees to t'. For each set of nodes associated with C, the composition of y' or fewer trees to t' will lead to y'z or fewer nodes being associated with y or fewer
nodes in \( t \), where \( y \) is the maximum number of source trees in a single row of TABL. Therefore, along a path, as steps 1, 2, ..., \( i \) are executed, the number of sets of nodes of \( t' \) which can be associated with the same set of nodes of \( t \) increases as \( y^*z^i, (y^*z)^2, \ldots, (y^*z)^{i-1} \). The weight of the target tree will be largest if there is just one path in \( t \) and \( (y^*z)^{i-1} \) trees are composed to \( t' \) at the \((i-1)\)-th step of the transform. For suppose there were two paths instead. Then at most
\[
y^*z^i + y^*z^2 + \ldots + y^*z^{i-1}
\]
trees are composed to \( t' \) during the \( i \)-th step of the transformation, where \( a(i) \) is the number of trees in \( t\)-TABL in the first path and \( b(i) \) is the number of trees in \( t\)-TABL in the second path. But
\[
y^*z^i + y^*z^2 + \ldots + y^*z^{i-1}
\]
is always less than or equal to \( y^i z^j \) if \( a(i) + b(i) = j \).

With Theorem T.7.3, we will now analyze the time necessary for an honest bpst to translate a sentence. The definition of honesty used in the last three chapters can be applied here without change. Let \( M \) be an honest bpst in normal form, and suppose \( w \in \text{sen}(Q) \) with length \( n \).

Except for the actual transformation process itself, the bpst translates in exactly the same manner as does the ipst. Therefore, the time required by the bpst to check for deviant parses is the same as that required by the ipst.

There is some integer \( k \) which is constant for \( M \) such that in a single transformation step, there are at most \( k \) target sub-structures composed to the transformation of any tree. Therefore, the amount of work necessary to perform a single transformation step in a bpst is on the order of the same amount of work necessary in the ipst. The number of steps is less than or equal to the pseudo-weight of the target tree generated since each step results in the composition of at least one phrase-structure to the target tree. By T.7.3, there are constants \( y \) and \( z \) such that for source tree \( t \), if \( \text{pwt}(t) = j \), then the pseudo-weight of the transform of the source tree is less than or equal to \( (yz)^j \). There are no definitions of linear, multi-linear, or exponential bpst which would naturally correspond to the
definitions given for cpst in D.4.8. Consequently, we cannot make such fine distinctions between the execution times of various classes of translators. The best case and worst case analyzes, as presented in chapter v, apply here. In particular, time bounds 5.5 and 5.7, with \((yz)\) substituted for \(k\), are, in general, the best and worst times necessary to translate \(w\). These revised time bounds are stated as bounds 7.1 and 7.2, respectively.

\[
0((yz)^{n} + nF(n)) \quad (7.1)
\]

\[
0((yz)^{n} + pn + pnF(n)) \quad (7.2)
\]

where \(p\) is the number of parse of \(w\), and \(F\) is the maximum time complexity of any relation in a production-rule pair of the source system.

**Summary**

The **block phrase-structure transducer** was studied. It differs from the incomplete transducer in that the tree transformer component of the transducer can induce more complex syntactic relationships between source and target systems than can the incomplete transducer. We stated four sufficient conditions for a block transducer to be semantic-preserving and proved their sufficiency. We then isolated the power which the block tree transformer adds to the transduction process by proving that block transducers whose source and target languages are defined by complete systems can translate between more language pairs than the complete transducer of chapter iv. Finally, we examined the execution time for a subclass of block translators and showed that its execution time cannot in general, be bounded by a polynomial function of the length of the source sentence under constraints which are sufficient to guarantee a polynomially bounded execution time for the complete and incomplete translators.
Chapter VIII - Semantic-Directed Generation

...The name of the song is called 'Haddock's Eyes'!

"Oh, that the name of the song, is it?" Alice said, trying to feel interested.

"No, you don't understand," the Knight said, looking a little vexed. "That's what the name is called. The name really is, 'The Aged Aged Man'.

"Then I ought to have said 'That's what the song is called'?" Alice corrected herself.

"No, you oughtn't, that's quite another thing! The song is called 'Ways and Means', but that is only what it is called you know!"

"Well, what is the song then?" said Alice, who was by this time completely bewildered.

"I was coming to that", the Knight said. "The song really is 'A-Sitting On A Gate', and the tune's my own invention".

Lewis Carroll

In chapter iii we specified a mechanism for defining a language, the phrase-structure system. Given a sentence of a pss, there is a well-defined algorithm for computing its set of meanings. The Generation Problem is concerned with the converse problem.
The Generation Problem

The "Generation Problem" may be stated informally as:

State an effective procedure which when given a language $A$ and a meaning "$u$" from the semantic universe of $A$, will construct one or more sentences of $A$ whose set of meanings include "$u$".

In this chapter we define and analyze the augmented grammar, a system which solves the Generation Problem. A variant of the augmented grammar, the inverted phrase-structure system, will be used to construct a semantic-directed translator which will map each meaning of a source sentence into a syntactic structure in the target language. The production rules of an augmented grammar contain in a combined format what was divided into separate syntactic and semantic components in a phrase-structure system. This leads to a more complex grammar and language definition than used in defining a pss but one which provides a clear direct method for using the meaning of a source sentence to construct a parse tree of the target sentence without relying upon syntactic relationships between the source and target languages.

Augmented Grammar

Defn D.8.1. An augmented grammar (ag) is a 5-tuple $(U, V_n, V_t, Ax, Pr)$ where:

1) $U$ is the semantic universe, any r.e. set.
2) $V_n$ is a finite set of nonterminal symbols.
3) $V_t$ is a finite set of terminal symbols.
4) $Ax \in V_n$ is the start symbol or axiom.
5) $Pr$, the production set, is an r.e. set of ordered labeled trees of unit height such that the root is labeled from $(V_n)^xU$ and the frontier from $(V_n \cup V_t)^xU$. For any $u \in U$, if $Ax$ is the label of the first component of any node, then that node is the root of the production in which it appears.

The union of $V_n$ and $V_t$ is denoted $V$ for vocabulary. The set $(V_n)^xU$ is called the nonterminal node set (nns), $(V_t)^xU$ is
the terminal node set (tns), and their union is the vocabulary node set (vns). For \((v,u) \in vns\), \(v\) is the syn-mem and \(u\) is the sem-mem.

Example E.8.1. An augmented grammar for binary numbers:

\[\text{AG} = (U, Vn, Vt, Ax, Pr)\]

1) \(U = \text{INT0}\)
2) \(Vn = (Ax, T)\)
3) \(Vt = (0, 1)\)
4) \(Pr\) is the union of the following sets:
   - a) \(\{(Ax, i) < (T, i) > | i \in \text{INT0}\}\)
   - b) \(\{(T, i) < (T, j) (0, 0) > | i \text{ is even and } j = i/2\}\)
   - c) \(\{(T, i) < (T, j) (1, 1) > | i \text{ is odd and } j = i/2\}\)
   - d) \(\{(T, 0) < (0, 0) >\}\)
   - e) \(\{(T, 1) < (1, 1) >\}\)

D.8.1 is a nonstandard definition of a generative grammar in that the number of grammar productions may be infinite. (Mazurkiewicz [69] also investigated context-free grammars which had an infinite set of production rules. His studies were concerned strictly with the syntactic properties of such grammars. He did not address the Generation Problem at all.) Intuitively, each node of a production represents not only the syntactic information normally associated with a vocabulary symbol, but also contains what one might think of as an attribute value associated with that symbol as well. Since an attribute may in general assume an infinite number of values, the number of productions of an augmented grammar is also allowed to be infinite. The manner in which phrase-structures are generated from an augmented grammar is analogous to the way in which phrase-structures are generated in a context-free grammar.
Defn D.8.2. Let $P_r$ be the set of productions of some augmented grammar. The closure of $P_r$ under tree composition is denoted $\text{phr}(P_r)$, where two nodes $c$ and $d$ are said to have the same labels $(v, u)$ and $(v', u')$ if $v = v'$ and $u = u'$. The following sets are also defined:

1) ("terminated phrases-structures") $\text{phrt}(P_r) = \{ t \in \text{phr}(P_r) \mid c \in \text{fr}(t), \text{label}(c) \in V_t \}$. 
2) ("complete phrases-structures") $\text{phrc}(P_r) = \{ t \in \text{phrt}(P_r) \mid \text{label}(\text{rt}(t)) = A_x \}$. 
3) ("axiomatic phrases-structures") $\text{phra}(P_r) = \{ t \in \text{phr}(P_r) \mid \text{label}(\text{rt}(t)) = A_x \}$. 

[*]

Defn D.8.3. The phrase-structures of augmented grammar $G$ are the phrase-structures of its production set $P_r$. Similarly, for its terminated and axiomatic phrase-structures. The complete phrase-structures of $G$, $\text{phrc}(G) = \{ t \mid t \in \text{phrc}(P_r) \text{ and no sem-mem of any node of } t = \text{"err"} \}$. 

[*]

Example E.8.2. A complete phrase-structure of $AG$ is given in Figure 8.1.

```
(s,3)  
(1,1)  
(1,1)  
```

Figure 8.1. Complete Phrase-Structure of $AG$.

[*]

Defn D.8.4. The syntactic frontier, $\text{syn-fr}(t)$, of $t \in \text{phr}(G)$ is $w_l..w_k$ where $(w_l,m_l)..<(w_k,m_k)$ is the frontier of $t$. 

[*]
**Defn D.8.5.** The sentences of an augmented grammar $G$, $\text{sen}(G) =$

$\{ w | w \text{ is the syntactic frontier of } t \in \text{phrc}(G) \}$

The language of an augmented grammar $G$, $L(G) =$

$\{ (w, u) | \text{there exists } t \in \text{phrc}(G) \text{ such that } x = \text{syn-fr}(t) \text{ and } u = \text{sem-mem}(rt(t)) \}$

Then "11" is a sentence of AG and (11,3) belongs to its language. The first two theorems follow immediately from the above definitions.

**Thm T.8.1.** For any augmented grammar $G$, $\text{phr}(G)$, $\text{phrt}(G)$ and $\text{phrc}(G)$ are all r.e.

**Proof:** Let $p_1, p_2, \ldots, p_n, \ldots$ be an enumeration of $Pr$ and let $Pr(i)$ denote $(p_1, \ldots, p_i)$. Because an ag with a finite set of productions is just a context-free grammar, the transitive closure of any finite set of productions in an ag is effectively enumerable. We take advantage of this fact in defining a procedure which enumerates $\text{phr}(Pr)$. Let $k$ be any positive integer. Enumerate $\text{phr}(Pr)$ using P.8.1.

**Procedure P.8.1.**

**Purpose:** Enumerate $\text{phr}(Pr)$.

**Input:** Augmented grammar $G = (U, V_n, V_t, A_x, Pr)$.

**Output:** $\text{phr}(Pr)$.

**Steps:**

1) $i = 1$

2) Enumerate $p_i$;

3) DO $j = 1$ TO $i$

   Enumerate next $k$ phrase-structures in $\text{phra}(Pr(j))$;

   END;

4) $i = i + 1$
5) GOTO step 2;

This procedure clearly enumerates \( \text{phr}(G) \). To form \( \text{phrt}(G) \) simply test each \( p \in \text{phr}(G) \) as it is enumerated. If \( \text{label}(c) \in V_t \) for each \( c \in \text{fr}(p) \), enumerate \( p \) as a member of \( \text{phrt}(G) \). Similarly, test each \( p \in \text{phr}(G) \) for whether \( \text{label}(\text{rt}(p)) = A_0 \) and that no sem-mem of any node of \( p \) equals "err". If \( p \) passes this test enumerate it as a member of \( \text{phrc}(G) \).

\[*\]

**Thm T.8.2.** For any augmented grammar \( G \), \( \text{sen}(G) \) and \( L(G) \) are recursively enumerable sets.

**Proof:** The theorem follows immediately from the fact that \( \text{phrc}(G) \) is r.e.

\[*\]

Having introduced the augmented grammar, we now show how it can be used to generate the sentences of a synthesized phrase-structure system. The reader should recall that for any general phrase-structure system there is an equivalent syn-pss and that for any syn-pss with several attributes, there is an equivalent one with only one attribute associated with every nonterminal (C.3.1). In the rest of chapter viii and all of chapter ix, we will always assume, without loss of generality, that a syn-pss has only one attribute - "mn". In that case a production-rule pair looks like

\[
\begin{align*}
\text{c}_0 < \text{c}_1 \ldots \text{c}_k > &\quad \text{mn}(\text{c}_0) = f(\text{mn}(\text{c}_1), \ldots, \text{mn}(\text{c}_k)) : D_1 \ldots D_k \rightarrow C
\end{align*}
\]

Because \( f \) is dependent upon each of the \( \text{mn}(\text{c}_i) \), and because there is only one attribute, we will abbreviate the semantic rule above by

\[
\begin{align*}
\text{c}_0 < \text{c}_1 \ldots \text{c}_k > &\quad f : D_1 \ldots D_k \rightarrow B
\end{align*}
\]

**Lemma L.8.1.** Suppose \( Q \) is a syn-pss, and that \( \text{c}_0 < \text{c}_1 \ldots \text{c}_k > \) is a member of \( \text{Pr} \). The set of \( k+1 \) tuples

\[
\{(u_0, \ldots, u_k) \mid \text{there is defined a } \text{val}-t \text{ for } t \in \text{phr}(\text{Pr}) \text{ such that an occurrence of } \text{c}_0 < \text{c}_1 \ldots \text{c}_k > \text{ is a phrase-structure of } t \text{ and for } 0 \leq i \leq k, \text{val}-t(\text{c}_i) = u_i \}
\]

is an recursively enumerable set.

**Proof:** Lemma follows immediately from the fact that
phrc(Pr) is r.e. and that for each \( t \in \text{phrc}(Pr) \), \( \text{val}-t \) is r.e. 

[\*]

**Defn D.8.6.** For any syn-pss \( Q = (U, V_n, V_t, A_x, Pr, Ru) \), the associated augmented grammar of \( Q \), \( aag(Q) \) is 

\( (U, V_n, V_t, A_x, Pr') \) where

1) \( U = \{ \text{att-val}(v, \text{attu}) ; \text{attu} \in \text{att}(v) \text{ for } v \in V \} \)

2) If \( c_0<cl..ck> \in Pr \) and 
   a) there is a \( t \in \text{phrc}(Pr) \) such that there is an occurrence of \( c_0<cl..ck> \) as a sub-structure of \( t \); and 
   b) there is a \( \text{val}-t \) function defined such that for an occurrence of \( c_0<cl..ck> \) in \( t \), \( \text{val}-t(ci) = u_i \) for \( 0 \leq i \leq k \); 

then \( (c_0,u_0)<(c_1,u_1)..<(c_k,u_k)> \) is a production in \( Pr' \). 

[\*]

From D.3.1 we know that for each attribute, \( \text{attu} \), of vocabulary symbol \( v \), the set \( \text{att-val}(v, \text{att}) \) is recursively enumerable. Furthermore, from L.8.1 we know that \( Pr' \) of \( aag(Q) \) is r.e. Therefore, the associated augmented grammar is well-defined. The following theorem is an immediate consequence of D.8.6.

**Thm T.8.3.** \( \text{Sen}(aag(Q)) = \text{Sen}(Q) \). \( L(aag(Q)) = L(Q) \).

**Proof:** Obvious from D.8.6. 

[\*]

Call the set of sentences of \( Q \) which have meaning "\( u \)" - \( \text{phrc}(Q,u) \). The aag can be used to generate parse trees of sentences in \( \text{sen}(Q) \) with a given meaning "\( u \)". The most direct way of solving the Generation Problem using the aag is to simply enumerate \( \text{phrc}(aag(Q)) \) and cull from that set all trees such that the sem-mem of the root is not "\( u \)". Although this procedure does solve the Generation Problem using the aag, it does so by enumerating every complete tree of \( Q \). We wish to find another characterization of the aag which does not necessarily rely upon enumerative techniques in order to generate sentences with specified meanings. The
model we propose to do this with is called an inverted phrase-structure system for reasons which will soon become apparent.

Inverted Phrase-Structure Systems

Defn D.8.7. An inverted phrase-structure system (ipss) is an ordered pair \( Q = (G, SEM) \) where

1) \( G = (Vn, Vt, Ax, Pr) \) is a reduced cfg. No node on the frontier of a production in \( Pr \) may be labeled by the axiom.

2) \( SEM \) is a 3-tuple \( (U, att-val, Ru) \) where
   a) \( U \) is any recursively enumerable set, the universe of discourse.
   b) \( att-val \) is a total function from \( Vx("mn") \) to powerset(\( U \)), the vocabulary meaning function, such that for each \( v \in V \), \( att-val(v, "mn") \) is a recursively enumerable set.
   c) \( Ru \) is a finite non-empty set of ordered pairs, where the first member of the \( i \)-th pair is the \( i \)-th production of \( Pr \), and the second member of the pair is a partial recursive relation

\[
f:C \rightarrow D \times x \times D y
\]

where \( C \) is \( att-val(label(rt(p))) \), \( y = fr(p) \), and each \( Di \) is a subset of \( att-val(label(fr(p,i))) \). We sometimes write the pair as

\[
p \ f:C \rightarrow D \times x \times D y
\]

The relation is called an inverse semantic rule. The ordered pair of production and inverse semantic rule is called a production-rule pair.

[*]

An example of an ipss may be found in sdt-1 in Appendix J.

Defn D.8.8. Let \( Q \) be an ipss. For each \( t \in \text{phra}(Pr) \), if function \( \text{ival} - t \) may be defined, then \( t \in \text{phra}(Q) \).
ival-t: nodes(t)->U

where c0<cl..ck> is a sub-structure of t and

c0<cl..ck> g(x0):C-->D1x..xDk

is a production-rule pair of R u implies that

(ival-t(cl),...,ival-t(ck)) ∈ g(ival-t(c0))

We say that ival-t specifies an evaluation of t.

[*]

For syn-pss Q, for each t in phrc(G), the set of all
val-t is called VAL-T, and the set of all non-deviant val-t
is called NDVAL-T. In a similar fashion, for ipst we define
the set of all ival-t to be IVAL-T, and the set of all
non-deviant ival-t to be NDIVAL-T. If any occurrence of
"mn" has the value "err" in an evaluation of t, then that
evaluation is considered to be deviant as were such
evaluations of trees generated by syn-pss. The complete
phrase-structures of ipss Q', phrc(Q'), are simply

(t : t ∈ phra(Pr'), NDIVAL-T ≠ Ø)

In general, it is impossible to completely evaluate the
attributes of any t ∈ phr(Pr) unless t is terminated. This
is because for a node c of a tree generated by a syn-pss,
mn(c) depends upon the attributes of nodes which are
descendents of c. However, if Q' is an ipss, then for every
t' ∈ phra(Pr'), it is possible to assign a value to every
attribute occurrence. This is because in evaluating t', for
c ∈ nodes(t'), mn(c) may be dependent only upon attributes
of nodes ancestral to c. As stated in D.3.5, val-t was
defined only if t was complete. Because we are dealing with
syn-pss, it is meaningful to talk of evaluating arbitrary
t ∈ phrt(Pr) as well as t ∈ phra(Pr). We find it useful to
extend val-t so that it is defined over all terminated trees
of Pr whether complete or not. We also extend mean(t, val-t)
to t ∈ phr(Pr) as well.

**Lemma L.8.2.** Let Q be an ipss. If

\[ t = t₀[t₁..tn] ∈ phra(Q) \] then \[ t₀ ∈ phra(Q). \]

**Proof:** \[ t ∈ phra(Q) ==> t ∈ phra(Pr) ==> t₀ ∈ phra(Pr) \]

Let ival-t₀ be the restriction of ival-t to nodes(t₀).

[*]
Thm 1.8.4. Let Q be an ipss. phra(Q), phrc(Q), sen(Q), and L(Q) are effectively enumerable sets.

Proof: Clearly, phra(Q) is a subset of phra(Pr). G = (V_n, V_t, A_x, Pr) is a cfg. Therefore, phra(Pr) is r.e. in weight-increasing order (see Appendix K for enumerating procedure). The proof that phra(Q) is r.e. is by induction on the weight of t \in phra(Pr).

Base Step: wt(t) = 1

Suppose t = c_0 < c_1 .. c_k > is a member of phra(Pr). There is some production-rule pair in Ru
c_0 < c_1 .. c_k > g(x_0) : C \rightarrow D_1 x .. x D_k
att-val(c_0) is r.e. and g(u) is r.e. for any u \in U. Dovetail the computation of the following procedure. Enumerate att-val(c_0). For each u \in att-val(c_0) enumerate g(u). Let (u_1 .. u_k) be any k-tuple of the enumeration of g(u_0) for some u_0 \in att-val(c_0). Define
ival-t: nodes(t) \rightarrow U by
ival-t(c_i) = u_i
The set of all possible ival-t is r.e., since one may be defined for each (u_1 .. u_k) \in g(u_0) and g(u_0) is r.e.

Induction Step: wt(t) = k + 1

Assume phra(Q) is r.e. in weight increasing order for trees with weight \leq k. Suppose t \in phra(Pr) has weight k + 1. Then t has some box bracketing t_0 t_1 .. t_z .. t_k where
1) wt(t_0) = k.
2) f_i \in V.
3) wt(t_z) = 1.

By L.8.2, we know that t_0 is not in phra(Q) \implies t is not in phra(Q). Since phra(Q) is effectively enumerable in weight increasing order for trees with weight \leq k, enumerate all trees in phra(Q) with weight \leq k as the first step in testing whether or not t \in phra(Q). Assume t_0 \in phra(Q). By hypothesis, IVAL-T_0 is r.e. Let ival-t_0(1), ..., ival-t_0(i), ... be an enumeration of IVAL-T_0. There is a production-rule pair in Ru
t_z t_r = g(x_0)
Dovetail the computation of the following procedure. For each \( u_i = \text{ival-t0}(i)(\text{rt}(tz)) \) enumerate \( g(u_i) \). For the \( j \)-th member, \((d_1, \ldots, d_m)\), of \( g(u_i) \) define the extention, \( \text{ival-t}(i,j) \), of \( \text{ival-t0}(i) \):

\[
\text{ival-t}(i,j): \text{nodes}(t) \rightarrow U
\]

such that if \( c \in \text{nodes}(t0) \) then

\[
\text{ival-t}(i,j)(c) = \text{ival-t0}(i)(c)
\]

else \( c \in \text{nodes}(\text{fr}(tz)) \) and

\[
\text{ival-t}(i,j)(c) = d_m \text{ for } 1 \leq m \leq |\text{fr}(tz)|
\]

End of induction.

Since \( \text{phra}(Q) \) is r.e., it follows immediately that \( \text{phrc}(Q) \) is r.e. and hence that \( \text{sen}(Q) \) and \( L(Q) \) are r.e. as well.

[\*]

The ipss becomes particularly interesting when it is specially constructed from a syn-pss. In particular, if \( Q = (G, SEM) \) is a syn-pss then let \( Q' = (G, SEM) \) be the associated ipss of \( Q \), \( \text{aipss}(Q) \), where

1) \( U \) is an r.e. set which contains
\( \{ \text{att-val}(v,\text{attu}) \mid \text{attu} \in \text{att}(v) \text{ for } v \in V \} \)

2) For each \( v \in V \), \( \text{att-val}'(v) = \text{att-val}(v,mn) \)

3) If \( R_u \) contains:

\[
c_0<c_1..c_k> f:D1x..xDk--->C
\]

then \( R_u' \) contains:

\[
c_0<c_1..c_k> f^{-}\text{inv}:C--->D1x..xDk
\]

Intuitively, \( \text{att-val}(v) \) is meant to be the set of values which a node labeled \( v \) in any \( t \) in \( \text{phrc}(Pr) \) can be assigned by a \( \text{val-t} \) function. As stated, however, \( \text{att-val}(v) \) can be any superset of these values. We, therefore, wish to define a subset of \( \text{att-val}(v) \) which we call \( \text{MIL-ATT-VAL}(v) \), which is
\(\{u \mid u \in \text{att-val}(v) \text{ and there is some } t \in \text{phrt}(Pr) \text{ such that for } c \in \text{nodes}(t)\) \\
with label v there is an evaluation, \\
\text{val-t for which val-t}(\text{mn}(c)) = u)\)

The next lemma is obvious from the fact that 
\(\text{phrc}(Pr) = \text{phrc}(Pr').\)

**Lemma L.8.3.** Let \(Q\) be a syn-pss and let \(Q' = \text{aipss}(Q)\). For each \(t \in \text{phrc}(Pr)\), there is a bijection from VAL-T to IVAL-T such that

\[\text{biject}(\text{val-t}) = \text{ival-t} \implies \text{val-t} = \text{ival-t}\]

**Proof:** Since \(Pr = Pr'\), then \(\text{phrc}(Pr) = \text{phrc}(Pr')\). Suppose \(\text{val-t}\) is a particular evaluation of \(t\). Then for each sub-structure \(c_0 < c_1 \ldots c_k\) in \(t\) with unit weight, there is a production-rule pair in \(Q\),

\[c_0 < c_1 \ldots c_k \vdash f : D_1 \ldots D_k \rightarrow C\]

If \(u = \text{val-t}(\text{mn}(\text{rt}(t)))\), then \(u \in \text{domain}(f^{-1})\). Hence, we can define \(\text{ival-t}(\text{mn}(\text{rt}(t))) = \text{val-t}(\text{mn}(\text{fr}(t,i)))\) to be equal to \(\text{ival-t}(\text{mn}(\text{fr}(t,i)))\) for \(i \in \mathbb{I}\).

[*]

**Thm T.8.5.** Let \(Q\) be a syn-pss and let \(Q' = \text{its associated ipss}\). Then \(\text{phrc}(Q) = \text{phrc}(Q')\), \(\text{sen}(Q) = \text{sen}(Q')\), and \(L(Q) = L(Q')\).

**Proof:** Since \(Pr = Pr'\), \(\text{phrc}(Pr) = \text{phrc}(Pr')\). By L.8.3 for each \(t \in \text{phrc}(Pr)\) there is a bijection from VAL-T to IVAL-T such that

\[\text{biject}(\text{val-t}) = \text{ival-t} \implies \text{val-t} = \text{ival-t}\]

\(t \in \text{phrc}(Q) \implies \text{NDVAL-T} \not\subseteq Q \implies \text{NDIVAL-T} \not\subseteq Q \implies \text{NDVAL-T} \not\subseteq \text{Qval-t} \implies t \in \text{phrc}(Q).\) Therefore, \(\text{phrc}(Q) = \text{phrc}(Q')\), \(\text{phrc}(Q) = \text{phrc}(Q') \implies \text{sen}(Q) = \text{sen}(Q')\).

Let \(\text{val-t}\) be a non-deviant evaluation of \(t \in \text{phrc}(Q)\). There is a \(\text{ival-t}\) such that \(\text{val-t} = \text{ival-t}\). Therefore, \((\text{fr}(t), \text{mean}(t, \text{val-t})) \in L(Q) \implies (\text{fr}(t), \text{mean}(t, \text{ival-t})) \in L(Q')\). Let \(\text{ival-t} \in \text{NDIVAL-T}\). There is a \(\text{val-t}\) such that \(\text{ival-t} = \text{val-t}\). Therefore, 
\((\text{fr}(t), \text{mean}(t, \text{ival-t})) \in L(Q') \implies (\text{fr}(t), \text{mean}(t, \text{val-t})) \in L(Q)\).
In any syn-pss, the relation $f$ of an semantic rule is, by definition, total recursive. Therefore, its inverse relation is partial recursive. This leads to a translation procedure - TRANS. In the rest of this chapter, we present this translation procedure. In chapter ix, the properties of TRANS will be investigated.

Translation Procedure

The translator has three arguments, (i) the source syn-pss $Q$, (ii) an ipss $Q' = aipss(Q'')$, where $L(Q'')$ is the target language; and (iii) the string $w$ which is to be translated. The translation procedure is divided into two phases: (i) compute the set of meanings $\{u_1, \ldots, u_z\}$ of the source string $w$; (ii) for each $i \in z$, generate a target sentence $v_i$ such that $u_i \in \text{mean}(v_i)$, provided such a sentence exists.

In the first phase, COMPUTE-MN parses the string in terms of the productions of the source system. It then computes the meanings of $w$ using the relations associated with the productions of the system as stated in chapter iii on phrase-structure systems. The set of meanings of $w$ =

$$\text{mean}(t) \mid t \in \text{parses}(w) \text{ and } t \text{ has no linear recursive sub-structures}$$

In chapter iii, it was shown that mean($w$) is effectively computable. Once the set of meanings of the source sentence has been computed, the source sentence and sentence parses are no longer needed. The translation process continues without further reference to it, in marked contrast to the intimate syntactic tie between source and target grammars used in the phrase-structure approach.

In the second phase, for each meaning of the source sentence, a parse of the target tree is generated. The subroutine GEN builds a complete phrase-structure $t$ of the target system such that $u \in \text{mean}(t)$. TRANS outputs one translation for each meaning of the source sentence from among the set of all possible such target sentences (provided such a translation exists). It will not, in general, halt unless a translation is found for each meaning of $w$. 
The procedure is divided into a main routine and several subroutines. The two interesting subroutines are GEN and COMPUTE-MN whose functions were discussed above. They are detailed here in a pseudo-PL/I format. The other subroutines are:

1) MEMBER(X, Y) - test whether X is in r.e. set Y.

2) PARSE(W) - parses string w using source grammar G. It returns the ordered set of all parses which do not have linear recursive sub-structures.

3) MEANING(T) - returns an ordered set of the meanings of tree t, which are computed using source system Q.

In addition, a somewhat unusual language construct is introduced in TRANS - the DO DOVETAIL loop. The normal PL/I DO loop [IBM70] is used to execute a block of code repeatedly varying the value of a counter with each iteration. The counter, itself, may be referenced within the body of the loop. Suppose the loop is constructed so that it will execute k times before completing. It will execute the i-th iteration completely before beginning execution of the (i+1)-st iteration. However, if the i-th iteration does not terminate, then no computation for the (i+1)-st or any later iterations will ever be performed. If the computation of each iteration is independent of the computation of any other iteration, it is possible to partially execute the other iterations even if one or more of them never terminates. The standard technique for doing this is to dovetail the computation [Brai73]. This is the strategy which we will follow here. We indicate this by writing DO DOVETAIL instead of simply DO as the header of the loop. If DO DOVETAIL is used, the computation of the loop is dovetailed over the indicated index values.

**Procedure P,8.2.**

**Main Procedure - TRANS(Q, Q', W)**

**Purpose:** To translate in a semantic-directed manner from L(Q) to L(Q').

**Input:** Source syn-pss Q; target ipss Q' which is associated with syn-pss Q"; source sentence w ∈ sen(Q).

**Output:** For 1 ≤ i ≤ |mean(w)|, one string vi ∈ (sen(Q') ∪ ε), the empty string, for the i-th meaning ui
of $w$. If $v_i \in \text{sen}(Q')$ then $u_i \in \text{mean}(v_i)$.

Steps:
1) $M = \text{COMPUTE-MN}(w, Q)$;
2) IF $M \neq Q$ THEN
3) $T_1$:
   DO DOVETAIL $I = 1$ TO $|M|$
   IF MEMBER($M(I), \text{MIN-ATT-VAL}(S')$) THEN
5) $T' = \text{GEN}(M(I), S', Q');$
6) $V' = \text{FR}(T')$;
7) OUTPUT = $V'$;
8) ELSE
9) OUTPUT = $\varepsilon$;
10) END;
11) END;

Procedure P.8.3 COMPUTE-MN(w, Q)

Purpose: Compute the set of meanings of string $w$ as an ordered set.

Input: The source string $w$ and the source syn-pss $Q$.

Output: An ordered set of all meanings $(u_1, \ldots, u_z)$ of $w$.

Steps:
1) $\text{MEAN} = Q$;
2) $\text{PARSES} = \text{PARSE}(w)$;
3) $K = 1$;
4) $CMI$:
   DO $I = 1$ TO $|\text{PARSES}|$
   $M = \text{MEANING}(\text{PARSES}(I))$;
5) $\text{CM}2$:
   DO $J = 1$ TO $|M|$
   $\text{MEAN}(K) = M(J)$;
7) $K = K + 1$;
8) END;
9) END;
10) RETURN($\text{MEAN}$);

Procedure P.8.4 GEN(SEM, V, Q').

Purpose: Generate a terminated phrase-structure, $t$, of $Q'$ which has an evaluation $\text{val-t}$ such that $\text{mean}(t, \text{val-t}) = \text{SEM}$. 
Input: SEM ∈ U', the universe of Q'; v ∈ V', the vocabulary symbols of Q'; and Q', the target ipss, which is associated with syn-ipss Q''.

Output: t ∈ phrt(Q'') such that mean(t) contains SEM.

Data structures: PRP-N is that set of production-rule pairs whose production is rooted by nonterminal N.

Steps:

1) G1:
   DO DOVETAIl I = 1 TO |PRP-N|;
   R = PRP-N(I);
   T = PRODUCTION(R);
   IF RELATION(R)(SEM) ≠ Q THEN
   5) G2:
      DO DOVETAIl J = 1 TO |RELATION(R)(SEM)|;
   6) G(I,J) = R;
   7) G3:
      DO K = |FR(R)| TO 1 BY -1;
   8) M = RELATION(R)(SEM)(K);
   9) F = FR(G(I,J),K);
   10) IF MEMBER(F,VN') THEN
   11) G(I,J) = COMPOSE(G(I,J),
       GEN(M,F,Q'),K);
   12) ELSE
   13) G(I,J) = COMPOSE(G(I,J),F,K);
   14) RETURN(G(I,J));
   15) END;
   16) END;
   17) ELSE;
   18) END;

[∗]

We can impose an arbitrary ordering on mean(w). Then the outputs of the translator can be ordered, since there is one output for each meaning of w. We will denote the output of the translator for the i-th meaning of w, output(w,i). The set of all such outputs will be denoted output(w). We will denote an occurrence of TRANS for which Q and Q' have been specified (but not w), by TRANS(Q,Q',*). In this case, the source and target languages have been fixed, but not the sentence which is to be translated. TRANS(Q,Q',*) is a translator to convert from L(Q) to L(Q').

Let us say a translation procedure is sound if it never produces an incorrect translation and that it is complete if it always produces a translation whenever one exists. We will say a particular instance of TRANS, TRANS(Q,Q',*), is
sound if it never produces an incorrect translation and that TRANS(Q,Q',*) is complete if it always produces a translation whenever one exists.

Thm 1.8.6. TRANS is both sound and complete.
Proof: Let TRANS(Q,Q',*) be an occurrence of TRANS and suppose w is a source sentence with meanings \( \{u_1, \ldots, u_n\} \). The set of meanings of w is a finite, effectively computable set. Mean(w) is empty if w is not a sentence of 0. The main procedure of TRANS outputs "E", the empty string, as the translation of w iff w is not a sentence. Suppose n, the number of meanings of w, is some positive integer. For each \( u_i \in \text{Mean}(w) \), the main procedure tries to find a sentence \( v_i \in \text{sen}(Q') \) which has among its meanings, the meaning \( u_i \). Since the n searches for the translations are dovetailed, even if the search for a particular \( v_i \) never halts, the other searches may succeed. For a particular \( u_i \), the main procedure first tests whether or not \( u_i \) belongs to MIN-ATT-VAL(S'), where S' is the axiom of the target system. MIN-ATT-VAL(S') is an r.e. set. Therefore, if \( u_i \) does belong to MIN-ATT-VAL(S'), the test will successfully terminate. If \( u_i \) does not belong to MIN-ATT-VAL(S'), then the test either will not terminate or it will terminate unsuccessfully. If the test does not terminate, then there is no sentence of Q' which has meaning \( u_i \). The translator will, correctly, not emit any sentence as the translation of w for meaning \( u_i \). If the test fails, then the translator will output "E", which is again correct since there is no sentence of Q' which has meaning \( u_i \). If the test succeeds, then there is at least one sentence, \( v_i \), of Q' which has meaning \( u_i \). The subroutine GEN is called to find \( v_i \). If it is successful, it returns a parse of \( v_i \). The translator then correctly outputs \( v_i \). GEN is nothing more than a procedure which constructs axiomatic phrase-structures of Q' whose meaning will be \( u_i \). If GEN is called, then there must be a sentence of Q' which has among its meanings \( u_i \). Since phra(Q') is an effectively enumerable set, GEN will succeed in finding such a \( v_i \). Therefore, TRANS is both sound and complete.

[*]
Summary

We defined a mathematical model of semantic-directed translation. The theoretical basis for this model is the augmented grammar and its variant the inverted phrase-structure system. The former is based on the context-free grammar, the latter on the phrase-structure system. We examined several properties of augmented grammars and inverted phrase-structure systems to show their suitability in a model of semantic-directed translation. TRANS, a translation procedure based on the phrase-structure system and the inverted phrase-structure system was defined. In theorem T.8.6 we showed that this procedure is both sound and complete; i.e., it never produces an incorrect translation and always produces a translation if one exists.
Chapter IX - Analysis of The Procedure "TRANS"

"What do you read my Lord?"
"Words, words, words."
Shakespeare

Introduction

The last chapter introduced a procedure - TRANS - which translates in a semantic-directed manner. This chapter examines the properties of that procedure.

Algorithmic Nature of TRANS

In this section, we first prove that there are two syn-pss, Q and Q", for which TRANS(Q,aipss(Q"),*) is not an algorithm. We then determine two sufficient conditions for TRANS(Q,aipss(Q"),*) to be an algorithm.

Thm T.9.1. There are two syn-pss Q and Q" such that TRANS(Q,aipss(Q"),*) is not an algorithm.

Proof: Let Q" be any syn-pss such that L(Q") = {(n,f(n)) | n G INTI) where f is any total recursive function whose range is strictly r.e. By T.3.2 we know such Q" exist. Let Q be any syn-pss such that L(Q) = {(n,n) | n G INTI). There is a translation of n e sen(Q) iff n e range(f). Let Q' = aipss(Q"). By T.8.6, TRANS(Q,Q',*) is both sound and complete. If TRANS(Q,Q',*) were an algorithm then it would be decidable whether there is a translation of n simply by running TRANS(Q,Q',n) and observing whether the output is $\varepsilon$, the empty string, or an integer. Hence it would be decidable whether n e range(f). But by hypothesis, range(f) is not recursive. A contradiction. Therefore, TRANS(Q,Q',*) is not algorithmic.

[\*]

Denote the set of meanings of sentences of syn-pss Q by mean(Q), and the set of meanings of sentences of ipss Q by mean(Q').
ANALYSIS OF THE PROCEDURE "TRANS"

ALGORITHMIC NATURE OF TRANS

Thm T.9.2. If (i) MIN-ATT-VAL(S') is a recursive set in
U' = mean(Q) \cup \{att-val(v,mn) | v \in \{N' \cup T'\}\}; or (ii)
mean(Q) is a subset of mean(Q'); then TRANS(Q,Q',*) is an
algorithm.

Proof:
(i) Let TRANS(Q,Q',w) be a call of TRANS and suppose
MIN-ATT-VAL(S') is recursive in U'.

Case I:
ui is the i-th member of mean(w) which is in
MIN-ATT-VAL(S') \Rightarrow w has a translation vi such that
mean(vi) contains ui. TRANS(Q,Q',*) is complete by
T.8.6. Therefore, output(w,i) will be a sentence of Q'
which is a translation of w.

Case II:
ui is the i-th member of mean(w) which is not in
MIN-ATT-VAL(S') \Rightarrow output(w,i) = \epsilon.

Therefore, TRANS(Q,Q',*) is an algorithm.

(ii) Suppose mean(Q) is a subset of mean(Q').
mean(Q') = MIN-ATT-VAL(S'). Therefore, the test in
step 4 of TRANS
MEMBER(M(I),MIN-ATT-VAL(S'))
terminates successfully. TRANS is complete by T.8.6
\Rightarrow TRANS(Q,Q',w) generates a translation for each
meaning of source sentence w \Rightarrow TRANS(Q,Q',*) is an
algorithm.

[***]

Making TRANS Efficient

An algorithm which does not operate in a time bounded
by a polynomial function of the length of its input is
generally considered "impractical" for implementation on
current or forseeable hardware [AhoHU74]. As defined
earlier, the execution of TRANS(Q,Q',*) requires the
dovetailed execution of three nested sets of statements.
This is necessary because, in general, a particular
iteration of any of loops T1, G1 or G2 may not terminate.
An iteration of T1 may not terminate because step 4 of TRANS
may not terminate. An iteration of G1 may not halt because
loop G2 which is nested inside loop G1 may not terminate
execution. Finally, there may be an infinite number of
iterations of G2 since for a particular I, there may be no J
for which G(I,J) is defined and IRELATION(R)(SEM)I may be
infinite. In general then, the execution time of
TRANS(Q,Q',*) is not bounded by any polynomial function of
MAKING "TRANS" EFFICIENT

Since $G_2$ may execute arbitrarily many times, we investigate restrictions of the original semantic-directed translation procedure $TRANS$ in order to see under what constraints $TRANS$ is in $PTIME$, the set of algorithms whose execute time is bounded above by a polynomial function of the length of the input.

**Weak Inverse Functions**

In this section we investigate a restriction of $TRANS$ in which the cardinality of $RELATION(R)(SEM)$ is one. In this special case, the loop $G_2$ executes only once per call of $GEN$. Instead of using the entire inverse relation in the semantic rule, we will instead use what we call a weak inverse function. Whereas for each $x \in \text{range}(f)$, $f^{-1}(x)$ could contain an infinite number of $k$-tuples, in the case of the weak inverse function, $f^{-inv}(x)$ has exactly one $k$-tuple selected from $f^{-1}(x)$ in some well-defined manner.

**Defn D.9.1.** Let $Q$ be a syn-pss and $Q''$ be its aipss. Suppose

$$c_0 < c_1 \ldots c_k > f: \text{att-val}(c_1)x \ldots x\text{att-val}(c_k) \rightarrow \text{att-val}(c_0)$$

is a production-rule pair of $R_{u}$ in $Q$. Then $R_{u}''$ contains the corresponding production-rule pair

$$c_0 < c_1 \ldots c_k > f^{-inv}: \text{att-val}(c_0) \rightarrow \text{att-val}(c_1)x \ldots x\text{att-val}(c_k)$$

A weak inverse function ($w^{-inv}$) of $f$ is

$$f^{-winv}: U \rightarrow \text{MIN-ATT-VAL}(c_1)x \ldots x\text{MIN-ATT-VAL}(c_k) \cup \{"err"\}$$

where $f^{-winv}$ is a total recursive function such that

$$f^{-winv}(u) = (d_1, \ldots, d_k) \Rightarrow (d_1, \ldots, d_k) \in f^{-inv}(u)$$

and

$$f^{-inv}(u) = \emptyset \Rightarrow f^{-winv}(u) = \{"err"\}$$

[∗]

**Defn D.9.2.** Let $Q = (G, SEM)$ be an ipss associated with syn-pss $Q' = (G', SEM')$, where $G = G'$. $Q'' = (G'', SEM'')$ is a weak inverted phrase-structure system (wipss) associated with $Q'$, awipss($Q'$), where $U'' = U$, att-val'' = att-val and if $R_u$ contains
ANALYSIS OF THE PROCEDURE "TRANS"

WEAK INVERSE FUNCTIONS

\[
c_0 < c_1 \ldots c_k > \quad f^{-1v:C -->D}x_1 \ldots x_D \]
then \( R_u^v \) contains

\[
c_0 < c_1 \ldots c_k > \quad f^{-invv}\text{U -->}D_1x_1 \ldots x_D \quad \text{"err"}
\]

Phra(\( Q'' \)), phrc(\( Q'' \)), sen(\( Q'' \)), and \( L(\( Q'' \)) \) are defined in the same way these sets are defined for an ipss.

\(*\]

Lemma 9.1. Let \( Q \) be an ipss and \( Q' \) be a wipss both associated with syn-pss \( Q'' \). Then phra(\( Q \)), phrc(\( Q \)), sen(\( Q \)), and \( L(\( Q \)) \) are subsets of phra(\( Q' \)), phrc(\( Q' \)), sen(\( Q' \)), and \( L(\( Q' \)) \), respectively.

Proof: Obvious from the definition of wipss.

\(*\]

If a wipss \( Q' \) is used in place of ipss \( Q \) as the second argument in an occurrence of TRANS, loop G2 is executed once. Let \( W\)-TRANS denote an occurrence to TRANS using a wipss as the second argument. Clearly, \( W\)-TRANS is sound because from L.9.1 we know that \( L(\( Q' \)) \) is a subset of \( L(\( Q \)) \). However, \( W\)-TRANS is not complete. E.9.1 presents an instance of a translator using \( W\)-TRANS in which each source sentence has an infinite number of translations, yet \( W\)-TRANS(\( Q, Q', \ast \)) will be able to successfully translate only a single sentence - the null program.

Example E.9.1. Consider example sdt-2 in appendix L. \( L_1 \) and \( L_2 \), the source and target languages, both generate assignment programs which have only assignment and null statements. To illustrate the point of this example, the languages may be simple and so the arithmetic expressions on the right-hand side of the assignment operator are trivial. Instead of using the inverse relation in rule 2, the weak inverse function:

\[
right(mn(x_0)) = (\varepsilon, mn(x_0)) \quad \text{(9.1)}
\]

is chosen. Had the inverse relation been used in an instance of TRANS(\( Q, Q', \ast \)), two possible mappings would have been tried by TRANS in building the target tree:

\[
right(mn(x_0)) = (\varepsilon, mn(x_0)) \quad \text{(9.2)}
\]

\[
split(mn(x_0)) = (\text{front}(mn(x_0)), \text{tail}(mn(x_0))) \quad \text{(9.3)}
\]

where eq 9.2, the one actually used, sends to the left branch of the production the null string and to the right branch, the entire meaning to be generated. Consequently
W-TRANS is unable to find the translation of any source program other than the program consisting solely of the null statement. Yet, there are an infinite number of translations for each source program! The search for a translation of the program: "A = 1;" is pictured in Figure 9.1.

\[ (*) \]

\[
\text{AX} \quad \text{GEN}((A,3),AX,Q')
\]
\[
\mid
\]
\[
P \quad \text{GEN}((A,3),AX,Q')
\]
\[
\text{GEN}(Q,ASN,Q') \quad \text{ASN} \quad P \quad \text{GEN}((A,3),P,Q')
\]
\[
\text{GEN}(Q,ASN,Q') \quad \text{ASN} \quad P \quad \text{GEN}((A,3),P,Q')
\]

Figure 9.1 Runaway Recursion in W-TRANS

It should be obvious from E.9.1 that we can no longer guarantee that W-TRANS will halt either. There would appear to be a trade-off between making TRANS efficient and guaranteeing that the translation procedure is complete and halts on all inputs. We discuss this tradeoff more fully in the rest of this section.

Lemma L.9.2. Let \( f: \text{INT0} \rightarrow \text{INT0} \) be any total recursive function such that \( \text{range}(f) \) is not recursive. Then there does not exist a total recursive function \( g: \text{INT0} \rightarrow (\text{INT0} \cup \{\text{"err"}\}) \) such that

\[
g(n) = f^{-1}(n) \text{ if } f^{-1}(n) \text{ is defined;}
\]
\[
= \text{"err"} \text{ otherwise}
\]

Proof: Suppose by way of contradiction that such \( g \) exists. \( n \in \text{range}(f) \iff n \in \text{domain}(f^{-1}) \iff g(n) \neq \text{"err"}. \) Since \( g \) is total recursive by hypothesis, so
ANALYSIS OF THE PROCEDURE "TRANS"

WEAK INVERSE FUNCTIONS

is range(f). Contradiction. Therefore, g does not exist.

[*]

Thm T.9.3. There are two synthesized phrase-structure systems Q and Q'' such that there is no weak inverted phrase-structure system, Q', of Q'' for which

W-TRANS(Q,Q',*) is complete

Proof: Let f:INT0—>INT0 be a 1-1 total recursive function such that the range of f is strictly r.e. Let Q'' be (Q'',SEM'') where

Pr'' = (S<A>, A<0A>, A<1A>, A<0>, A<1>)

Ru'' contains:

S<A> f:INT0—>INT0

A<0A> plus:(0)xINT0—>INT0

A<1A> plus:(1)xINT0—>INT0

A<0> id:{0}—>INT0

A<1> id:{1}—>INT0

Let Q be any syn-pss such that L(Q) = <(n,n) | n ∈ INT0>. By T.9.1, TRANS(Q,aipss(Q''),*) induces a strictly partial recursive translation mapping from sen(Q) —> sen(Q'').

f-inv is a partial recursive function. Furthermore, f is 1-1 => f-inv is 1-1 as well. Let Q'' = aipss(Q'') be ((S,A),(0,1),S,Pr'',Ru''). Let g be the total recursive function which is the relation in the production-rule pair of Ru'' whose production is S<A>.

Let

C = (n : g(n) ≠ "err")

and let

B = (n : f-inv(n) is defined)

C cannot properly contain B or else TRANS(Q,Q'',*) would induce a translation on some n ∈ C-B contradicting T.8.6. C ≠ B because f-inv is strictly a partial recursive function by L.9.2. Therefore, C is a proper subset of B. INT0 = mean(Q) => there is some n₀ ∈ C-B. g(n₀) = "err" => W-TRANS(Q,Q',n₀) outputs e as the translation of n₀. f-inv(n₀) is defined =>
ANALYSIS OF THE PROCEDURE "TRANS"

WEAK INVERSE FUNCTIONS

\[ \text{TRANS}(Q, \text{aipss}(Q^n), n_0) = \nu \in \text{sen}(Q^n). \]

Therefore, \( \text{W-TRANS}(Q, Q', *) \) is not complete.

\[ [\ast] \]

**Thm T.9.4.** Let \( Q \) and \( Q^n \) be two synthesized phrase-structure systems such that for each \( \nu \in V^n \) of \( Q^n \), \( \text{MIN-ATT-VAL}(\nu) \) is a recursive set whose characteristic function is known. Then there is a \( \text{wipss}(Q^n) = Q' \), effectively constructable from \( Q^n \), such that \( \text{W-TRANS}(Q, Q', *) \) is complete and algorithmic.

**Proof:** Let \( Q = (G, \text{SEM}) \) and \( Q^n = (G^n, \text{SEM}^n) \). Define \( \text{wipss}(Q^n) \) to be \( Q' = (G^n, \text{SEM}^n') \) where if \( R_u^n \) contains

\[ \begin{align*}
    c_0<c_1..c_k> & \; \; \; f(x_1,..,x_k): \\
    \text{att-val}(c_1)x..x\text{att-val}(c_k) & \rightarrow \text{att-val}(c_0) \\
\end{align*} \]

then \( R_u' \) contains

\[ \begin{align*}
    c_0<c_1..c_k> & \; \; \; g(x_0): \\
    U & \rightarrow \text{MIN-ATT-VAL}(c_1)x..x\text{MIN-ATT-VAL}(c_k) \\
\end{align*} \]

where \( g(u_0) \) is computed as follows:

1) \( \text{MIN-ATT-VAL}(c_0) \) is recursive by hypothesis. Test whether \( u_0 \in \text{MIN-ATT-VAL}(C_0) \). If not, then \( f^{-1}(u_0) = Q \) so set \( g(u_0) = \text{"err"} \).

2) There is a procedure (see Appendix K) which enumerates the trees in \( \text{phrt}(P^n) \) in weight increasing order and in a uniform manner for trees with the same weight such that if \( t_0[t_1..t_i[y_1..y_m]..t_k] \) is enumerated before \( t_0[t_1..t_i[y_1'..y_{m'}]..t_k] \) then \( t_i[y_1..y_m] \) is enumerated before \( t_i[y_1'..y_{m'}] \).

3) Order the rules of \( Q^n \). Since \( Q^n \) is a syn-pss, each \( t \in \text{phrt}(Q^n) \) may be evaluated. Given a tree \( t \in \text{phrt}(Q^n) \) there is a uniform way of evaluating \( t \) (if \( Q^n \) is semantically ambiguous) so that \( \text{VAL-T} \) is a well-ordered set. In particular, let

\[(t_0,n_0,m_0),..,(t_z,n_z,m_z)\]

be the leftmost derivation of \( t \). Associated with each evaluation of \( t \) is a number base \( |R| \).
ANALYSIS OF THE PROCEDURE "TRANS"

WEAK INVERSE FUNCTIONS

\[ q_0, q_1, \ldots, q_z \]

where \( q_i \) is the production-rule pair of \( R \) used to evaluate \( t_i \). Call this number "num-val-t" for \( val-t \). Order NUM-VAL-T in ascending numerical order. We will say that \( val'-t < val''-t \) iff \( num-val't < num-val''-t \).

4) Let \( t \) be the first tree enumerated with \( \text{rt}(t) = c_0 \) for which a non-deviant evaluation, \( val-t \) assigns \( mn(\text{rt}(t)) \) the value \( u_0 \). Let \( val'-t \) be the least numbered such \( val-t \). If \( t = c_0 < c_1 \ldots c_k > t_1 \ldots t_k \) then \( g(u_0) = (u_1, \ldots, u_k) \) where for \( i \in k \), \( u_i = val'-t(c_i) \); otherwise, \( g(u_0) = "err" \). Since \( u_0 \in \text{MIN-ATT-VAL}(c_0) \), at least one such \( t \) exists. Therefore, \( g \) is a total recursive function.

Claim: \( W-\text{TRANS}(Q, Q', *) \) is a well-defined, complete, and algorithmic.

Part I: Well-defined.

From the construction, it is clear that \( g \) is total recursive. \( g(u_0) = (u_1, \ldots, u_k) \) \( \Rightarrow \) there is a \( t \) and \( val-t \) such that \( t = c_0 < c_1 \ldots c_k > t_1 \ldots t_k \) for which \( val-t(c_i) = u_i \) \( \Rightarrow \) \( g(u_0) \) is in \( \text{f-inv}(u_0) \) \( \Rightarrow \) \( W-\text{TRANS}(Q, Q', *) \) is well-defined.

Part II: Complete.

Suppose \( \text{mean}(w) = (u_1, \ldots, u_z) \) where \( \text{mean}(w) \) is ordered so that \( u_i < u_j \) for \( i < j \) and suppose that \( \text{TRANS}(Q, \text{aipss}(Q''), w) \) outputs ordered set \( (v_1, \ldots, v_z) \) where \( v_i \) is either the empty string \( \epsilon \) or a sentence of \( Q'' \). We show that \( W-\text{TRANS}(Q, Q', w) \) outputs ordered set \( (v_1', \ldots, v_z') \) where \( v_i' \) is either \( \epsilon \) or undefined iff \( v_i = \epsilon \) and \( v_i' \) is a sentence of \( Q'' \) otherwise. \( W-\text{TRANS}(Q, Q', *) \) is sound. \( \text{TRANS}(Q, \text{aipss}(Q''), *) \) is complete by T.8.6. Therefore, it will follow that \( W-\text{TRANS}(Q, Q', *) \) is also complete.

The \( i \)-th output of \( \text{TRANS}(Q, \text{aipss}(Q''), w) \) is denoted "output\((w, i)\)". Denote the \( i \)-th output of \( W-\text{TRANS}(Q, Q', w) \) by "w-output\((w, i)\)". Since \( W-\text{TRANS}(Q, Q', *) \) is sound, if output\((w, i) = \epsilon \) then w-output\((w, i) = \epsilon \) or is not defined. If output\((w, i) = v_i \in \text{sen}(Q''), dki \) \( \Rightarrow \) w-output\((w, i) = v_i' \in \text{sen}(Q'') \), then \( W-\text{TRANS}(Q, Q', *) \) is complete, then the \( i \)-th output of \( W-\text{TRANS}(Q, Q', w) \) is also a sentence of \( Q'' \).
If \( v_i \) is not \( \varepsilon \), let \( \{ t_i, \ldots \} \) be the set of all sentences of \( O^n \) whose meanings include \( u_i \). Let \( t_a \) be that \( t_{ij} \) which comes first in the enumeration of \( \text{phrt}(P^n) \). Let \( \text{val}^n-t_a \) be that member of \( \text{VAL}-T_A \) with the smallest number which assigns \( \text{mn}(rt(t_a)) \) the value \( u_i \). Suppose \( t_a = t_0[t_1, \ldots, t_d] \) where \( t_0 \in P^n \), and suppose that the \( b \)-th production-rule pair of \( R_u^n \) is used in \( \text{val}^n-t_a \) to evaluate \( t_0 = c_0 < c_1, \ldots, c_d > \)

\[
c_0 < c_1, \ldots, c_d > f(x_1, \ldots, x_k) : \]
\[
\text{att}-\text{val}(c_1)x_1 \ldots \text{att}-\text{val}(c_k) \rightarrow \text{att}-\text{val}(c_0)
\]

Then the \( b \)-th production-rule pair of \( R_u^n \) is

\[
c_0 < c_1, \ldots, c_d > g(x_0):U \rightarrow \text{MIN}-\text{ATT}-\text{VAL}(c_1)x_1 \ldots \text{MIN}-\text{ATT}-\text{VAL}(c_k) \quad \text{("err")}
\]

Because of the way in which \( G \) is defined

\[
g(u_0) = (\text{val}^n-t_a(c_1), \ldots, \text{val}^n-t_a(c_d)) = (u'_1, \ldots, u'_d)
\]

When \( W-\text{TRANS}(Q, Q', w) \) is executing the \( i \)-th iteration of \( T_l \), \( \text{GEN}(u_i, S', Q') \) is called. This in turn leads to the execution of loop \( G_3 \). Suppose \( \text{GEN}(u'_k, c'_k, Q') \) is called for any \( k \in d \), that \( t_a \) is \( t_0[t_1, \ldots, t_k][q_1, \ldots, q_h] \ldots t_d \) where \( t_k \in P^n \), and that the \( r \)-th production-rule pair of \( R_u^n \) is used in \( \text{val}^n-t_a \) to evaluate \( t_0 = c'_0 < c'_1, \ldots, c'_h > \)

\[
c'_0 < c'_1, \ldots, c'_h > f'(x_1, \ldots, x_h) : \]
\[
\text{att}-\text{val}(d'_1)x_1 \ldots \text{att}-\text{val}(d'_h) \rightarrow \text{att}-\text{val}(c_0)
\]

Then the \( r \)-th production-rule pair of \( R_u^n \) is

\[
c'_0 < c'_1, \ldots, c'_h > g'(x_0):U \rightarrow \text{MIN}-\text{ATT}-\text{VAL}(c'_1)x_1 \ldots \text{MIN}-\text{ATT}-\text{VAL}(c'_h) \quad \text{un ("err")}
\]

Because of the way in which \( g' \) is defined

\[
g'(u'_1) = (\text{val}^n-t_a(c'_1), \ldots, \text{val}^n-t_a(c'_h))
\]

This recursive process continues until \( \text{GEN} \) is called for all of the terminated productions of \( t_a \). At that point, \( t_a \) is returned in step 5 of \( \text{TRANS} \) and is assigned to \( T' \).
ANALYSIS OF THE PROCEDURE "TRANS"
WEAK INVERSE FUNCTIONS

Part III: Algorithmic.

Suppose MIN-ATT-VAL(S') is recursive. If ui not in MIN-ATT-VAL(S') then w-output(w,i) = \emptyset. If ui \in MIN-ATT-VAL(S') then ta is defined and w-output(w,i) = ta. Therefore, W-TRANS(Q,Q',*) halts on all inputs.

[*]

In the proof of T.9.4, the method for effectively constructing a semantic-directed translator W-TRANS(Q,Q',*), from Q and Q' relies on an effective enumeration of the phrase-structures of the target language. Because of the high time complexity of this enumerative scheme, we wish to investigate under what conditions translators which possibly have lower time complexities can also be guaranteed to be complete and algorithmic. The next theorem states a condition on the target semantics which is sufficient to insure that W-TRANS(Q,Q',*) is complete and algorithmic.

Thm T.9.5. Let Q and Q" be two syn-pss. Let Q' be an awipss of Q". If

\begin{align*}
g(u_0) = \text{"err"} & \Rightarrow f-\text{inv}(u_0) = Q \\
\text{for } g, \text{ the relation of the } i-th \text{ production-rule pair of } Ru', \text{ and } f-\text{inv}, \text{ the corresponding relation of the } i-th \text{ production-rule pair of } \text{aipss}(Q") \text{ and if for each } t \in \text{phrc}(Q"), n, m \in \text{nodes}(t), \\
n \text{ is the parent of } m & \Rightarrow \\
|v(tr(n))| > |v(tr(m))|
\end{align*}

then W-TRANS(Q,Q',*) is complete and algorithmic.

Proof: If Q' satisfies the hypothesized conditions, then no matter which w-inv functions are used in the semantic rules of Q', in any chain of recursive calls of GEN, the length of the first argument must decrease as the number of calls increases. The recursion must terminate because in some call to GEN

1) The test "MEMBER(F,NON)" in step 10 of GEN fails.
2) RELATION(R)(SEM) in step 4 of GEN is empty for all members of PRP-N.
3) The first argument has length zero.
Therefore, \( W-\text{TRANS}(Q, Q', \star) \) is an algorithm. If \( GEN \) halts because step 10 is executed, then \( W-\text{TRANS}(Q, Q', \star) \) has successfully constructed one branch of the target tree.

Suppose \( GEN(SEM, V, Q') \) halts because \( \text{RELATION}(R)(SEM) \) is empty. By hypothesis, there does not exist \((d_1, \ldots, d_k)\) such that \( f-\text{inv}(\text{sem}) = (d_1, \ldots, d_k) \). But \( SEM \in \text{MIN-ATT-VAL}(V) \), so there is some other production \( V<..> \) with \( w-\text{inv} \) function \( g \) such that

\[
g(\text{sem}) = (e_1, \ldots, e_n)
\]

\( W-\text{TRANS}(Q, Q', \star) \) will dovetail through all of the productions rooted by \( V \) in loop \( G_1 \). Hence, \( W-\text{TRANS}(Q, Q', \star) \) will in some iteration of \( G_1 \) use the production \( V<..> \) mentioned above.

The only remaining case is when the length of the first argument diminishes to zero. But this can never happen since the range of all \( w-\text{inv} \) functions is non-empty.

\[\ast\]

### Criterion Selection

In this section we show that under certain constraints, \( GEN \) may be reorganized so that the recursive call of \( GEN \) within loop \( G_1 \) may be moved to outside the loop. We investigate this phenomenon for \( W-\text{TRANS} \). Whereas in the original version, one call of \( GEN \) is made for each member of \( PRP-N \), in this modified version, a call of \( GEN \) is made for only one member of \( PRP-N \).

#### Procedure P.2.1
**Procedure C-GEN(SEM, V, Q')**

**Purpose:** Generate a terminated phrase-structure, \( t \), of \( Q'' \) which has an evaluation \( \text{val}-t \) such that \( \text{mean}(t, \text{val}-t) = SEM \). Call to \( GEN \) is moved to outside \( G_1 \) in this version of \( GEN \). In original version, a call to \( GEN \) is inside the boundaries of \( G_1 \).

**Input:** \( SEM \in U' \), the universe of \( Q' \); \( V \in V' \), the vocabulary symbols of \( Q' \); and \( Q' \), the target \( \text{wipss} \), which is associated with \( \text{syn-pss} Q'' \).
ANALYSIS OF THE PROCEDURE "TRANS"
CRITERION SELECTION

Output: A tree \( t \in \text{phrt}(Q) \) such that there is an evaluation \( \text{val-}t \) for which \( \text{mean}(t, \text{val-}t) = \text{SEM} \).

Data Structures and Functions: \( \text{PRP-}N \) is that set of rules whose production is rooted by nonterminal \( N \). \( \text{FIRST-MEMBER}(A) \) is the first member of ordered set \( A \). \( \text{SECOND-MEMBER}(A) \) is the second member of ordered set \( A \).

\[
\text{CRITERION}(x) : \text{powerset}(P') \rightarrow P' \times P'
\]
is a total recursive function where for ordered set \( \{p_1, \ldots, p_n\} \in \text{powerset}(P') 
\text{CRITERION}(\{p_1, \ldots, p_n\}) = (p_i, i) \) for some \( i \in n \).

Steps:

1) \( G1 \):
   
   \[
   \text{DO } I = 1 \text{ TO } |\text{PRP-}N|;
   \]
   2) \( R(I) = \text{PRP-}N(I); \)
   3) \( T(I) = \text{PRODUCTION}(R); \)
   4) \( \text{IF } \text{RELATION}(R(I))(\text{SEM}) \neq \text{"ERR" THEN} \)
   5) \( \text{TUPLES}(I) = T(I); \)
   6) \( \text{ELSE} \)
   7) \( \text{TUPLES}(I) = Q; \)
   8) \( \text{END}; \)
   9) \( \text{IF } \text{TUPLES} = Q \text{ THEN RETURN}(); \)
10) \( \text{TUP} = \text{CRITERION}(\text{TUPLES}); \)
11) \( \text{G}(I) = \text{FIRST-MEMBER}(\text{TUP}); \)
12) \( G3 \):
   
   \[
   \text{DO } K = |\text{FR}(\text{G}(I))| \text{ TO } 1 \text{ BY } -1; \]
   13) \( \text{IK} = \text{RELATION}(\text{R}(\text{SECOND-MEMBER}(\text{TUP}))(\text{SEM}); \)
   14) \( \text{F} = \text{FR}(\text{G}(I), K); \)
   15) \( \text{IF } \text{MEMBER}(F, \text{NON}) \text{ THEN} \)
   16) \( \text{G}(I) = \text{COMPOSE}(\text{G}(I), \text{C-GEN}(\text{IK}, F, \text{Q}'), K); \)
   17) \( \text{ELSE} \)
   18) \( \text{G}(I) = \text{COMPOSE}(\text{G}(I), F, K); \)
19) \( \text{RETURN}(\text{G}(I)); \)
20) \( \text{END}; \)

[*]

We will call an instance of \( \#-\text{TRANS} \) which uses \( \text{C-GEN} \) (for "criterion generator") in place of \( \text{GEN} \) a \( \text{WC-TRANS} \). The ordered output of \( \text{WC-TRANS}(Q, Q, w) \) is denoted "wc-output \((w)\)."
Thm T.9.6. Let Q and Q" be two syn-pss. Let Q' be a wipss of Q". Suppose WC-TRANS(Q,Q',*) satisfies the conditions of T.9.5. Then WC-TRANS(Q,Q',*) is both complete and algorithmic.

Proof: The proof that WC-TRANS(Q,Q',*) is algorithmic is identical to that of T.9.5. Likewise, the recursion of C-GEN must terminate because in some call

1) The test "MEMBER(F,HON)" in step 15 of C-GEN fails.
2) TUPLES = Q at step 9 of C-GEN.
3) The first argument of C-GEN has length zero.

The reasoning for cases (1) and (3) is identical to that used in the proof of T.9.5. The difference is in case (2). We show that TUPLES can never be null after all members of PRP-N have been examined; i.e., that case (2) is vacuous, and hence that WC-TRANS(Q,Q',*) is complete. Suppose by way of contradiction that TUPLES is empty after loop G1 is finished executing. This means that the weak inverse function of each member of PRP-N evaluates to "err" for the particular argument SEM. By hypothesis, then, the corresponding inverse relation of each member of PRP'-N where PRP'-N is the set of rules in aipss(Q") whose production is rooted by N must not be defined for SEM. If this is true then there is no terminated tree t, rooted by N in phrt(Q") such that mean(t) includes SEM. C-GEN(SEM,N,Q') is a recursive call to C-GEN. If C-GEN(SEM',N',Q') called C-GEN(SEM,N,Q'), then there is a rule

\[ g(x0):U\rightarrow{\text{MIN-ATT-VAL}(c1)x..} \]
\[ \text{xMIN-ATT-VAL}(ck) \]

where \( g(\text{sem'}) = (..,\text{sem},..) \). By definition of MIN-ATT-VAL, there is some terminated tree t, rooted by N, such that mean(t) contains SEM. A contradiction. Therefore, case (2) can never occur.

[∗]

**Complexity**

We will now investigate the time complexity of WC-TRANS, the semantic-directed translator defined in the last section. In particular, we state sufficient conditions for WC-TRANS(Q,Q',*) to be in PTIME.
Lemma 7.2.3. If $t$ is the output of $C\text{-GEN}(\text{SEM}, N, Q')$, then $t$ has no linear recursive sub-structures.

Proof: Suppose by way of contradiction that the output, $r$, of $C\text{-GEN}(\text{SEM}, N, Q')$ contains a linear recursive sub-structure. By definition of a phrase-structure system, the corresponding entrance and exit attributes of a lrss are equal. Suppose $C\text{-GEN}(U, V, Q')$ is that call which leads to the construction of $t$. A single call of $C\text{-GEN}$ leads to the construction of just one tree. When $t$ has been constructed, $C\text{-GEN}(U, V, Q')$ will be called again. CRITERION depends only on the members of TUPLES. $C\text{-GEN}$ is a function of just $U$, $V$, and $Q'$. The execution of $C\text{-GEN}$ is deterministic. Hence, if $C\text{-GEN}(U, V, Q')$ constructs $t$ and in turn calls $C\text{-GEN}(U, V, Q')$ again, then this second call of $C\text{-GEN}$ must also lead to the construction of $t$ and itself call $C\text{-GEN}(U, V, Q')$ again. This cycle of calls to $C\text{-GEN}$ will continue indefinitely. But then $r$ is not a finite tree, contrary to hypothesis. Therefore, $r$ cannot contain linear recursive sub-structures.

[*]

Thm 7.2.7. Let $Q$ and $Q''$ be any two syn-pss. Let $Q'$ be a mips of $Q''$. If for each $w \in \text{sen}(Q)$, where $n = |w|$,

1) $\text{WC-TRANS}(Q, Q', w)$ halts; and

2) $\text{wc-output}(w) = (v_1, \ldots, v_z)$ => there is a function $f \in \text{POLY}$ such that for $i \in \mathbb{Z}$, $f(n) > |v_i|$, and

3) If $R_u'$ contains

$$ c\{c_1 \ldots c_k \} \quad g: U \rightarrow \text{MIN-ATT-VAL}(c_1)x_1 \times \text{MIN-ATT-VAL}(c_k) $$

un ("err")

then

a) $g \in \text{PTIME}$; and

b) $g(u) = (u_1, \ldots, u_k) => |u_i| \leq |u_1|$; and

4) $\text{MEMBER} \in \text{PTIME}$; and

5) $|\text{parse}(w)| \leq g(n)$ for some $g \in \text{POLY}$; and

6) $|u| \leq r(n)$ for some $r \in \text{POLY}$, $u \in \text{mean}(w)$; and

7) $\text{MEANING} \in \text{PTIME}$ for $t \in \text{phrc}(Q)$. 
ANALYSIS OF THE PROCEDURE "TRANS" COMPLEXITY

then WC-TRANS(Q, Q', *) ∈ PTIME.

Proof: If \text{mean}(w) = Q then the execution time of WC-TRANS(Q, Q', w) is just the execution time of COMPUTE-MN(w). Otherwise, it is the execution times of both COMPUTE-MN(w) and loop T1. The execution time of COMPUTE-MN(w) is dominated by step 2

\begin{equation}
\text{PARSES} = \text{PARSE}(W);
\end{equation}

\begin{equation}
\text{step } 5
\end{equation}

\begin{equation}
M = \text{MEANING(PARSES(I))};
\end{equation}

and the number of times loops CM1 and CM2 are executed. Statement 9.1 executes in time bounded by

\begin{equation}
n^{*3} + ng(n)
\end{equation}

since \|parsec(w)\| ≤ g(n) by hypothesis (5). MEANING ∈ PTIME ⇒ \|\text{MEANING}(T)\| ≤ h(wt(t)) for some h ∈ POLY. For a tree t without any linear recursive sub-structures, as all trees in PARSE(W) are, wt(t) ≤ k1*n for some constant k1. Therefore, h(wt(t)) < h(k1*n). The total time taken to complete the execution of each iteration of statement 9.2 is bounded by

\begin{equation}
g(n)h(k1*n)
\end{equation}

Loop CM2 is executed no more than \|PARSES\| \|MEAN(W)\| times which is bound by

\begin{equation}
g(n)g(n)h(k1*n)
\end{equation}

Therefore, the total execution time of COMPUTE-MN(W) is bounded by

\begin{equation}
n^{*3} + ng(n) + g(n)h(k1*n) + g(n)g(n)h(k1*n)
\end{equation}

which is in POLY.

We next examine the execution time of loop T1. Hypothesis (1) ⇒ each iteration of T1 terminates ⇒ it is not necessary to dovetail the execution of T1. We shall assume that T1 is an ordinary DO loop in the remainder of this proof. The number of iterations of T1 is bounded by

\begin{equation}
g(n)h(k1*n) ∈ POLY
\end{equation}

determined earlier in the analysis. If the execution of each iteration of T1 is bounded by some polynomial q
as a function of \( n \), then the execution time of \( T_1 \) is bounded by

\[
g(n)h(k_1n)q(n) \in \text{POLY}
\]

The execution time of the \( b \)-th iteration of \( T_1 \) for \( b \in |M| \) is dominated by the test in step 4

\[
\text{MEMBER}(M(I), \text{MIN-ATT-VAL}(S'))
\]

the call to \( C\text{-GEN} \) in step 5

\[
T' = \text{GEN}(M(I), S', Q');
\]

and computing \( \text{FR}(T') \) in step 6

\[
V' = \text{FR}(T');
\]

Predicate 9.8 is in \( \text{PTIME} \) by hypothesis (4) and \( |M(I)| \leq r(n) \) for \( r \in \text{POLY} \) by hypothesis (6). Therefore, test 9.8 concludes in time bounded by a polynomial function of \( n \).

The frontier of a tree may be computed in time proportional to its weight. By L.9.3, \( T' \) has no linear recursive sub-structures. Therefore, \( \text{wt}(T') < k_2*|V'| \) for some constant \( k_2 \). By hypothesis (2), \( |V'| \leq f(n) \) for polynomial \( f \). Therefore, statement 9.10 executes in time bounded by

\[
k_2*f(n)
\]

It remains to be shown that the execution time of statement 9.9 is bounded by a polynomial function of \( n \). \( \text{WC-TRANS}(Q, Q', w) \) is an algorithm \( \Rightarrow C\text{-GEN}(M(I), S', Q') \) returns a tree \( T' \in \text{phrc}(Q') \) such that \( ui \in \text{mean}(T') \). \( T' \) has no linear recursive sub-structures (by L.9.3) \( \Rightarrow \text{wt}(T') < k_2*|V'| \). Each call to \( C\text{-GEN} \) adds another phrase-structure with unit weight to \( T' \). Therefore, the number of calls of \( C\text{-GEN} \) is bounded by \( k_2*|V'| \) which by hypothesis (2) is less than or equal to \( k_2*f(n) \). If the execution time of each call of \( C\text{-GEN} \) is bounded by a polynomial function \( Z(n) \) of \( n \), then the total execution time of \( C\text{-GEN}(M(I), S', Q') \) is bounded by

\[
k_2*f(n)Z(n) \in \text{POLY}
\]

The execution time of \( C\text{-GEN}(SEM, N, Q') \) is dominated by the execution time of statement 4 which is dominated by the execution time of
ANALYSIS OF THE PROCEDURE "TRANS"

COMPLEXITY

RELATION(R(I))(SEM)  \hspace{1cm} (9.13)

the execution time of step 10

TUP = CRITERION(TUPLES);  \hspace{1cm} (9.14)

and the number of iterations of loops G1 and G3. The
number of iterations of both loops G1 and G3 is bounded
by a constant k3. Therefore, if the execution time of
each iteration of each loop is bounded by a polynomial
function of n then so is the total execution time of
the loops. RELATION(R(I)) \in PTIME by hypothesis (3i).
Furthermore, from hypotheses (6) and (3ii), it follows
that |SEM| \leq r(n). Therefore, the execution time of
statement 9.13 is bounded by a polynomial function of
n.

The domain of CRITERION is finite so its execution
time is bounded by a constant value k4.

Therefore, a call of C-GEN executes in a time
bounded by a polynomial function of n and therefore so
does WC-TRANS(Q,Q',*).

[\*]

Summary

The time required for TRANS to execute was analyzed.
We stated sufficient conditions for TRANS to be algorithmic.
Furthermore, we made two modifications to TRANS to decrease
the time required to translate. Finally, we proved that
subject to certain constraints, WC-TRANS, a version of TRANS
twice modified to increase efficiency, is in PTIME.
Chapter X - Summary And Conclusions

...which is to be the master...
Lewis Carroll

Summary

The formal properties of language translators have been studied within two distinct frameworks, phrase-structure translation and semantic-directed translation. The first is a method of inducing translations by establishing a correspondence between the parses of sentences in the source and target languages. The second is a method of inducing translations by establishing a correspondence between the meanings of a source sentence and syntactic structures in the target language.

The model of language upon which our translators are based is the phrase-structure system. Having both a formal syntax and a formal semantics, the phrase-structure system combines Buttelmann's formal approach to semantics with the notation of Knuth's attributed grammar. The phrase-structure system can define any language whose sentences form a recursive set and whose meanings form a recursively enumerable set, subject only to the constraints that each sentence has just a finite number of meanings, and the set of meanings of each sentence is computable as a total recursive function of the sentence. Thus, the phrase-structure system seems powerful enough to define all common programming languages and (presumably) all natural languages.

Earlier work in syntax-directed translation relied exclusively upon syntactic relationships between source and target languages to induce the translation. The phrase-structure translators are a family of "hybrid" syntax-directed translators which use the formally defined meaning of a sentence, as well as syntactic relationships, to induce the translation.

Semantic deviance, discussed informally by linguists for many years ([ChomN57]) and formalized by Buttelmann [74], has been applied in this thesis for the first time to
syntax-directed methods. The major result we have shown is that by incorporating deviance in a seemingly innocent fashion into the translation mechanism, the phrase-structure translator becomes equivalent in power to that of a Turing translator; i.e., a Turing machine which accepts a sentence in the source language as input and enumerates, as output, all sentences in the target language which share a meaning with the source sentence. Thus, general translators can be developed within the framework of the context-free grammar, the grammar model upon which the phrase-structure system and all of the phrase-structure translators are based.

For each variant of the phrase-structure translator, sufficient conditions are given for the translator to be semantic-preserving, and to ensure that a phrase-structure translator executes in a time bounded by a polynomial function of the length of the source sentence.

For a subclass of translators it has been shown that a translator which translates language A to language B may be effectively converted into a translator which translates language B to language A.

A semantic-directed translator, TRANS, may be constructed to translate between any two languages definable using phrase-structure systems. Two sufficient conditions for TRANS to halt on every input have been stated and proven sufficient. We have also proven that, subject to certain constraints, the execution time of the semantic-directed translator is bounded by a polynomial function of the length of the source sentence.

There is a tradeoff between the use of syntactic ambiguity in the definition of the source and target languages, and an increase in the "power" of the translator. In particular, weaker variants of the phrase-structure translator may be substituted for stronger variants by making appropriate changes in the definition of the source phrase-structure system to increase its syntactic ambiguity.

Models of Translation Studied

The translation models studied in this thesis may be divided into five major types:

1) **Complete Phrase-Structure Translation**, studied in chapter iv. It is the most basic model of phrase-structure translation, and has the weakest translation powers. It is called "complete" because
the complete set of parses generated by the source and target language systems have meanings associated with them. Thus, the sets of sentences of both languages are the complete sets of sentences of their underlying grammars. The semantics of the source and target languages are not used by the translator in any way to compute the translation of a sentence. The properties of this translator are similar to those of the syntax-directed translation schema defined by Aho and Ullman [69]. The complete translators are capable of inducing translations only over a relatively small class of language pairs. For example, no language whose sentences have context-sensitive features (such as gender agreement between noun and pronoun in English) can be translated.

2) **Incomplete Phrase-Structure Translation** studied in chapter v. The source and target systems of this translator may generate parses which have no meaning. The phrase-structure system is defined so that a string is a sentence only if it has a meaningful parse. Consequently, languages which have context-sensitive features can be given as arguments to the translator. The core of the translator is a tree transducer which accepts meaningful parses of the source sentence and outputs meaningful parses of what becomes the target sentence. This translator differs from the complete translator only in that meaningless parses of source sentences are not transduced and the frontiers of meaningless parses of target sentences are not given as the output of the translator.

3) **Predicate Phrase-Structure Translation**, studied in chapter vi. This translator is a generalization of the incomplete translator, using semantic information to establish the syntactic relationship between the parses of source and target trees at translation time. For example, if the translation of a language construct depended upon the data-type of a certain variable, then the translator could "check" what the variable's data-type was in determining how the translation should proceed.

4) **Block Phrase-Structure Translation**, studied in chapter vii. This translator is a generalization of the incomplete translator. The syntactic relationships which the tree transducer of the incomplete translator can establish are fairly restricted because of the simple nature of the transducer mechanism. In the block translator, trees in isolated parts of the parse
tree can be simultaneously transduced into trees which are positioned into isolated places of the target tree. For example, suppose two statements of a source program which were "far apart" (perhaps one at the beginning of the program, the other at the end) had to be translated as a group into one target language statement. The block translator would allow the syntactically isolated subtrees of the parse which "define" these two source statements to be transduced as a unit.

5) **Semantic-Directed Translation**, studied in chapters viii and ix. TRANS, the semantic-directed translator defined in chapter viii, induces a translation by relying totally on the relationship between the meanings of a source sentence and the way in which those meanings are realized syntactically in the target language. The translation process never directly couples the source and target grammars.

**How Much Semantics?**

Writing the formal specification of the semantics of a language can be quite time consuming compared to writing just the grammar of the language. One of the goals of this thesis has been to demonstrate that writing the semantics is worth the effort. But there are many "levels" of semantic definition. From a pragmatic point of view, the semantics should be written in the least complex way which enables the translator to work correctly. In many cases, the level of "meaning" which a sentence must have in order to do successful translation is surprisingly low. The level of "understanding" necessary for translation may not have to be as deep as for systems, such as robots or information retrieval systems, which must execute instructions. For example, it is not necessary for a translator to "know" whether or not it is Tuesday in order to translate sentence 10.1 from English into French.

If today is Tuesday, pick up the box. 

(10.1)

It is necessary for a robot to "know" if today is Tuesday, however, since its actions depend upon that information. Another example which is not so "world dependent" is resolving pronouns to their objects. Suppose languages A, B, and C are to be translated to one another. A and B have the universal pronoun "it" (in some form), but C has no pronouns at all. In order to translate between A and B, it is not necessary for the semantics of the languages to be written so that pronoun references are all resolved. The
word for "it" in B can be uniformly substituted for the word for "it" in language A. However, to translate from A to C, all references must be resolved, since the referent must be substituted in C for "it" in A.

We have shown that if the semantics of the source and target languages are not too "complex", then it is possible to use a phrase-structure translator to take advantage of the information which the semantics provides and still translate in a time comparable to the time taken by a purely syntax-directed translator. The constructs which we have used to define the examples in this thesis all have a relatively low time complexity. This is encouraging, but we must await efforts to devise more complete programming languages before we can conclude how general a phenomenon this is.

The feasibility of the semantic-directed translator seems to be highly dependent upon how well the translator-writer "knows" the target language. If he can design the target semantics so as to take advantage of the "short-cuts" given in chapter IX, then the semantic-directed translator will execute fairly rapidly. If not, then the execution time will probably not be acceptable for a practical implementation.

Why A Family of Translators?

Theorem T.5.4 states that the incomplete translator can induce whatever partial translation is defined between any two phrase-structure systems. Since the incomplete translator is in terms of language pairs translatable as powerful a model of translation as we can hope for, it may not be clear why the other members of the family have been defined at all. The motivation for the predicate and block translators is two-fold. First, it may be easier for the translator-writer (human or machine) to construct one type of translator than another. Second, a predicate or block translator may execute faster than an incomplete translator.

To explain the first point, consider an analogy to PL/I programs which do list-processing. The PL/I programming language includes features to facilitate the manipulation of list structures. They make it easier to perform many tasks which would otherwise be clumsy to do. Formally, however, every list processing task can be done without ever taking advantage of the available list-processing constructs. The clarity and conciseness of the program would probably suffer, though, without their use. In the same way, the
SUMMARY AND CONCLUSIONS

WHY A FAMILY OF TRANSLATORS?

Predicate and block translators provide the translator-writer with more powerful tools than does the incomplete translator. These tools make it possible for the translator-writer to capture his intuitions more clearly within the structure of the translator itself.

The execution times of the various translators are different. A predicate or block translator which translates between two given languages may execute in vastly different time than an incomplete translator which translates between the same two languages. Yet, even though the incomplete translator has the simplest translation mechanism, it need not be the fastest translator. It could be faster, as fast, or even slower than either a predicate or block translator. Each execution "step" of a block or predicate translator is more complex than that of the incomplete translator and hence requires more time to finish. However, the added power of the block and predicate translators leads them in some cases to do far more "work" per execution step. Consequently, a block or predicate translator could need fewer steps to complete the translation, overshadowing the extra time taken per step.

We know of no fixed formula for deciding which translator is best for any given situation. The best translator to use will depend upon the particular features of the languages and their defining systems.

One point which has clearly emerged from this research is that the efficiency of a translator is highly dependent upon the way in which the grammar and semantics for the source and target languages are defined. A language definition which is "good" for one translation may be "poor" for another. Furthermore, the way in which the source and target systems are defined strongly influences the ease with which the translator can be constructed at all. One of the things which we hope will come out of this research, as well as related research by Buttelmann [74] and Krishnaswamy [76], is a better understanding of what to look for in defining a system for a language in order to facilitate the construction of a translator.

Relationships Between Translation Systems

Besides the phrase-structure translators, there are several other models of translation based on syntax-directed methods. The Hasse diagram in Figure 10.1 summarizes the relative translation strengths of the family of
phrase-structure translators and these other systems. The ordering relation of the lattice is \( \leq \) (see Definition D.5.3). One translation model is \( \leq \) another translation model if the second can induce a translation which subsumes any translation which the first can induce. Two models are equal, \( = \), if the first is \( \leq \) the second and the second is \( \leq \) the first. In Figure 10.1, a single arrow indicates \( < \), and a double arrow indicates \( = \). The block complete and the predicate complete translators are block and predicate translators whose source and target systems do not generate meaningless parses.

![Diagram](image)

Figure 10.1. Relationships Between Translation Systems
SUMMARY AND CONCLUSIONS

RELATIONSHIPS BETWEEN TRANSLATION SYSTEMS

Other Results

One of the advantages that accrues from having the target system defined in the same notation as the source, is that a translation from language A to language B may provide insight into how to translate from B to A. In Theorem T.5.4 we showed that a large class of incomplete translators may be effectively inverted; i.e., a translator outputs target sentence v when given source sentence w if and only if the inverse translator outputs w when given input v.

To help characterize the class of language pairs which can be translated by a subset of the complete phrase-structure translators, we have proven a "pumping theorem" of sorts (Lemma L.4.1, Theorem T.4.7) similar to that of the "uvwxy" theorem for context-free languages (BarPS61). Using it we showed, for example, that there is no complete phrase-structure translator of this certain class which can translate from base-2 to base-10. This pumping theorem breaks down when an attempt is made to generalize it to incomplete translators. This is because of the major difference deviance makes in characterizing the class of languages generable by phrase-structure systems.

The Future

We have shown how an unrestrained use of deviance leads to a very powerful class of translators. Constraints upon the use of deviance may diminish the power of both language definition systems and translators. This could possibly lead to a hierarchy of languages and translators which would be analogous to the Chomsky hierarchy for generative languages. For example, Krishnaswamy [76] has already begun to examine translations in which deviance is used to filter parses that enter the tree transformer, but not those that leave it.

A standard tree transducer [ThatW65] is similar to the tree transducer of a complete translator. The tree transducer of the block translator can be viewed in isolation as a generalization of the standard tree transducer. It is well-known that many interesting transductions over trees cannot be performed using standard transducers. This is because of the "context-free" nature of the transduction process. The block tree transducer is interesting in its own right because it allows us to transduce trees in a "context-sensitive" manner. It is to some degree an alternative to the transformation grammar
approach to language generation [ChomN57]. Using the transformational grammar, a tree is first generated using a context-free base and then "context-sensitive" rules are applied to change the structural properties of the tree. The block tree transducer permits us to take into account context-sensitive considerations as an integral part of the original generation of the tree instead of as a follow-up to the basic generation. In this thesis we have not investigated the power of the block tree transducer per se, but only examined it as a component of a translation mechanism. Theorem T.7.2 has already established the fact that the block tree transducer is more powerful than the standard tree transducer. The investigation of block tree transducers is left as an open research area.

The term "phrase-structure translator" has never been formally defined in this thesis. This is done intentionally to emphasize the fact that the characterizing property of phrase-structure translation is a transduction process which centers around a phrase correspondence table, i.e., a table which relates phrase-structures of the source and target languages. We do not wish to specify precisely the particular nature of that transduction process or the particular format of the table. If we did, we would undoubtedly exclude translators which still capture the spirit of the phrase-structure translator. For example, another type of phrase-structure translator which could be analyzed is one which does not transform the source parse tree from root to frontier. The phrase-structures of a source parse could be transformed in some non-uniform manner, perhaps in a manner determined by the meanings of the tree. This could handle the situation in which for two programming languages, statements from scattered locations in the source program are to be translated into consecutive statements in the target program. The source statements could be translated in the order in which their individual translations are to appear in the target program.

Another interesting class of translators which could be investigated are those in which the history of the translation can be used to partially determine the behavior of the translator. In the same way that the Turing string transducer can capture a history of its own behavior on a particular input string by storing information on a tape, it may be possible to develop Turing transducers which operate on trees. Of course, the finite state tree transducer, which is ten years old, is nothing but the generalization of the finite state string transducer to the two-dimensional data structure of trees. Some preliminary work along these lines has already been reported by Krishnaswamy [75], but
the relative complexity of the Turing tree transducer is
discouraging. Perhaps more elaborate data structures than
just working tapes would lead to translators which are less
complex and more intuitive to construct.

Buttelmann devised a procedure for constructing a
table-driven translator given only the formal descriptions
of the source and target languages. As might be expected,
his procedure is slow because it must consider all of the
complex semantic and syntactic issues involved in building
the translator. It appears that Buttelmann's generation
procedure may be applicable to the incomplete translator.
The predicate and block translators, however, are
sufficiently different in nature that Buttelmann's procedure
does not seem to carry over. We have shown that in some
cases, using a predicate or a block translator can lead to
simpler, more concise translators than are constructable
using incomplete translators. It would be interesting then,
to investigate whether or not a predicate or block
translator "generator" can be found. Buttelmann showed that
all translators constructed by his procedure were truly
semantic-preserving. Could such a broad claim be made about
the output of a predicate or block translator generator, or
would this only be true under certain constraints? And how
would their speed compare to that of Buttelmann's translator
generator?

Recently there has been increased interest in using
syntactically ambiguous grammars to define programming
languages [Aho75]. The motivation offered by Aho, Johnson
and Ullman for their interest in ambiguous grammars is that
it is sometimes easier to write a grammar which happens to
be ambiguous than to try and "work around" the ambiguity. A
famous example of this is the "dangling ELSE" of ALGOL. Our
efforts here provide another incentive to reconsider the
unwritten rule which has developed over the years, that
programming language grammars should be syntactically
unambiguous. We have shown that there is a tradeoff between
the degree of syntactic ambiguity in the source grammar, and
the power of the translator required to perform the
translation. In particular, a predicate translator which
translates between two languages can be effectively
converted into an incomplete translator which translates
between the same two languages. The price for this
conversion is an increase in the syntactic ambiguity of the
source system. Furthermore, one of the basic results of
this thesis is that the incomplete translator is equivalent
in power to that of the Turing translator. The proof of
this depends directly upon the grammars of the source and
target systems being syntactically ambiguous. So a property
of grammars which has usually been considered to be a pariah in parsing, may turn out to reveal much about the formal properties of translation, and should be investigated further.

This thesis has been concerned with the formal study of translation. Consequently, we have not attempted to develop a "cookbook" of semantic constructs useful in defining programming languages. If the translators, either semantic-directed or phrase-structural, are to be implemented, then the development of such a "cookbook" is desirable. We have not implemented the translators which we have defined. Of course, the final test of whether or not the mechanisms we propose are useful in a practical sense must await attempts to build actual translators and to define actively used programming languages as input to them.
BIBLIOGRAPHY


BensD70  Benson, David B., "Syntax and Semantics: A Categorial View", Information and Control, 17, 2 (Sep 70).


BochG74  Bochmann, Gregor V., "Semantic Evaluation From Left to Right", Publication No. 175, Departement d'Informatique, Universite de Montreal (May 74).


ChomN57 Chomsky, Noam, Syntactic Structures, Mouton (1957).


DickF75 Dickey, Fred, personal communication (1975).


Feldman, Jerome and Gries, David, "Translator Writing Systems", CACM, 11, 2 (Feb 68).


Hopcroft, J.E. and Ullman, J.D., Formal Languages and Their Relation to Automata, Addison Wesley (1969).


JazaM74  Jazayeri, Mehdi, "On Attribute Grammars and The Semantic Specification of Programming Languages", Case Western Reserve University, Rpt. No. 1152, Cleveland, Oh (Oct 74).


KrisR75  Krishnaswamy, R., personal communication (1975).

KrisR76  Krishnaswamy, R., "Models of Translation for Formal Languages", Doctoral Dissertation (forthcoming), The Ohio State University, Columbus (1976).


Leggett, E., personal communication (1975).

Lewis, P.M., Rosenkrantz, D.J. and Stearns, R.E., "Attributed Translations", JCSS, 9, 4, pp. 279-307 (Dec 74).


APPENDIX A

Complete Phrase-Structure Translator: pst-l

Source language: Postfix Arithmetic Expressions
Target language: Infix Arithmetic Expressions

Both source and target languages are over the integers 1 and 2, and the operators + and *.

Source Language Specifications

Source system is \((G,SEM)\) where

\[
G = ((S,E),(1,2,+,\times),S,Pr) \text{ and }
\]

\[
syn-att(S) = syn-att(E) = \text{"val"}
\]

"plus" and "mult" are functions defined over \(\text{INT} \times \text{INT} \rightarrow \text{INT}\) which define common addition and multiplication.

Ru contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>S&lt;E&gt;</td>
<td>(\text{val}(x_0) = \text{val}(x_1))</td>
</tr>
<tr>
<td>E&lt;EE+&gt;</td>
<td>(\text{val}(x_0) = \text{plus}(\text{val}(x_1),\text{val}(x_2)))</td>
</tr>
<tr>
<td>E&lt;EE*&gt;</td>
<td>(\text{val}(x_0) = \text{mult}(\text{val}(x_1),\text{val}(x_2)))</td>
</tr>
<tr>
<td>E&lt;1&gt;</td>
<td>(\text{val}(x_0) = 1)</td>
</tr>
<tr>
<td>E&lt;2&gt;</td>
<td>(\text{val}(x_0) = 2)</td>
</tr>
</tbody>
</table>
**Target Language Specifications**

Target system is \((G',SEM')\) where

\[ G' = ((E,S,T,F),(1,2,+,*),(,)),S,Pr') \]

\[ \text{syn-att}(S) = \text{syn-att}(E) = \text{syn-att}(T) = \text{syn-att}(F) = "val" \]

"plus" and "mult" are functions defined over \(\text{INT} \times \text{INT} \rightarrow \text{INT}\) which define common addition and multiplication.

Ru' contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S &lt; E&gt;)</td>
<td>(\text{val}(x0) = \text{val}(x1))</td>
</tr>
<tr>
<td>(E &lt; T&gt;)</td>
<td>(\text{val}(x0) = \text{val}(x1))</td>
</tr>
<tr>
<td>(E &lt; E + T&gt;)</td>
<td>(\text{val}(x0) = \text{plus}(\text{val}(x1),\text{val}(x3)))</td>
</tr>
<tr>
<td>(T &lt; F&gt;)</td>
<td>(\text{val}(x0) = \text{val}(x1))</td>
</tr>
<tr>
<td>(T &lt; T \ast F&gt;)</td>
<td>(\text{val}(x0) = \text{mult}(\text{val}(x1),\text{val}(x3)))</td>
</tr>
<tr>
<td>(F &lt; (E)&gt;)</td>
<td>(\text{val}(x0) = \text{val}(x2))</td>
</tr>
<tr>
<td>(F &lt; 1&gt;)</td>
<td>(\text{val}(x0) = 1)</td>
</tr>
<tr>
<td>(F &lt; 2&gt;)</td>
<td>(\text{val}(x0) = 2)</td>
</tr>
</tbody>
</table>

ASSOC mapping is defined by:

\(E \rightarrow E \quad S \rightarrow S\)
### CORRESPONDENCE TABLE

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>X-Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>row 1:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>row 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
</tr>
<tr>
<td>E +</td>
</tr>
<tr>
<td>E E +</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>E + T</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td>(E)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1, 0, 0, 2, 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>row 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
</tr>
<tr>
<td>E E ⋆</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>I</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td>(E)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>0, 1, 0, 0, 2, 0</td>
</tr>
<tr>
<td>Source</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>row 4:</td>
</tr>
<tr>
<td>I</td>
</tr>
<tr>
<td>row 5:</td>
</tr>
<tr>
<td>I</td>
</tr>
</tbody>
</table>
STEP-BY-STEP TRANSLATION

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Row</th>
</tr>
</thead>
<tbody>
<tr>
<td>step 0:</td>
<td>S _ E _* _</td>
<td>S</td>
</tr>
<tr>
<td>step 1:</td>
<td>S _ E</td>
<td>S _ E</td>
</tr>
<tr>
<td>step 2:</td>
<td>S _ E _* _</td>
<td>S _ E _* _</td>
</tr>
</tbody>
</table>

row 1

row 3
Note that by T.4.2, this transducer is semantic preserving.
APPENDIX B

Complete Phrase-Structure Translator: pst-2

Source language: \( (n,2^{2n}) \mid n \geq 1 \)
Target language: \( (n,n) \mid n \geq 1 \)

Both languages are defined in unary notation over the terminal alphabet \{1\}.

Source Language Specification

Source system is \((G,SEM)\) where

\[
G = ((S,E),(1),S,Pr) \quad \text{and} \quad \text{syn-att}(S) = \text{syn-att}(E) = "val"
\]

"plus" and "expo" are functions defined over \(\text{INT} \times \text{INT} \rightarrow \text{INT}\) which define the common addition and exponentiation functions.

Ru contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>S\langle E \rangle</td>
<td>val(x0) = expo(2,val(x1))</td>
</tr>
<tr>
<td>E\langle 1E \rangle</td>
<td>val(x0) = plus(1,val(x1))</td>
</tr>
<tr>
<td>E\langle 1 \rangle</td>
<td>val(x0) = 1</td>
</tr>
</tbody>
</table>
Target Language Specification

Target system is \((G', \text{SEM}')\) where
\[
G' = ((S, E), (1), S, \text{Pr'}) \quad \text{and}
\]
\[
\text{syn-att}(S) = \text{syn-att}(E) = "val"
\]

"plus" is a function from \(\text{INT} \times \text{INT} \rightarrow \text{INT}\) which defines common addition.

\(Ru'\) contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S&lt;EE&gt;)</td>
<td>(\text{val}(x_0) = \text{plus}(\text{val}(x_1), \text{val}(x_2)))</td>
</tr>
<tr>
<td>(E&lt;1E1E&gt;)</td>
<td>(\text{val}(x_0) = \text{plus}(1, \text{plus}(\text{val}(x_2), \text{plus}(1, \text{val}(x_4)))))</td>
</tr>
<tr>
<td>(E&lt;1&gt;)</td>
<td>(\text{val}(x_0) = 1)</td>
</tr>
</tbody>
</table>

ASSOC mapping is defined by:

\(S \rightarrow S \quad E \rightarrow E\)
### CORRESPONDENCE TABLE

<table>
<thead>
<tr>
<th>Row</th>
<th>Source</th>
<th>Target</th>
<th>X-Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>E</td>
<td>E</td>
<td>0,2,0,2</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>E</td>
<td>E</td>
<td>0,0</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>I</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX C

Predicate Phrase-Structure Translator: pst-3

Source language: Postfix Arithmetic Expressions
Target language: Postfix Arithmetic Expressions

Both source and target languages are defined over the integers 1 and 2, the rationals 1. and 2., and the operators + and * (We will use the set of rational numbers instead of the real numbers to reflect the fact that the "real" numbers of FORTRAN, PL/I, etc., are actually limited precision rational numbers.).

Mixed mode expressions are allowed in the source language. Necessary conversions are handled automatically within the semantics.

No mixed mode expressions are allowed in the target language. To add or multiply an integer and a rational number, the integer expression must be surrounded by a C( ) to indicate that conversion of the integer value to a rational value must take place before the operator is applied to the two operands.

Source Language Specifications

Source system is (G,SEM) where

\[ G = ((S, E, O), (1, 1., 2, 2., +, *), S, Pr) \]

\[ \text{syn-att}(S) = \text{syn-att}(E) = "type", \quad \text{"val"} \]
\[ \text{syn-att}(O) = "op" \]

"plus" and "mult" are functions defined over INTxINT-->INT and over RATNXRATN-->RATN where RATN stands for the rational numbers. The two functions define the common addition and multiplication operations.
"cvir" converts an integer number to a rational number.
"apply(d, b, c)" = d(b, c) where \( d : B \times C \rightarrow W \) is a function, \( b \in B \) and \( c \in C \).

\(Ru\) contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
</table>
| \(S<E>\)  | \( \text{val}(x0) = \text{val}(x1) \)  
\( \text{type}(x0) = \text{type}(x1) \) |
| \(E<\text{EE0}>\) | if \( \text{type}(x1) = \text{type}(x2) \) then  
\( \text{val}(x0) = \) apply(op(x3), val(x1), val(x2))  
else  
if \( \text{type}(x1) = "\text{ratn}" \) then  
\( \text{val}(x0) = \) apply(op(x3), val(x1), cvir(val(x2)))  
else  
\( \text{val}(x0) = \) apply(op(x3), cvir(val(x1)), val(x2))  
if \( \text{type}(x1) \) or \( \text{type}(x2) = "\text{ratn}" \) then  
\( \text{type}(x0) = "\text{ratn}" \)  
else  
\( \text{type}(x0) = "\text{int}" \) |
| \(0<+>\)  | \( \text{op}(x0) = "\text{plus}" \) |
| \(0<*>\)  | \( \text{op}(x0) = "\text{mult}" \) |
| \(E<1>\)  | \( \text{val}(x0) = 1 \)  
\( \text{type}(x0) = "\text{int}" \) |
| \(E<1.>\) | \( \text{val}(x0) = 1. \)  
\( \text{type}(x0) = "\text{ratn}" \) |
| \(E<2>\)  | \( \text{val}(x0) = 2 \)  
\( \text{type}(x0) = "\text{int}" \) |
| \(E<2.>\) | \( \text{val}(x0) = 2. \)  
\( \text{type}(x0) = "\text{ratn}" \) |
Target Language Specifications

Target system is \((G', SEM')\) where

\[
G' = \{(S,E),(1,1,2,+,*),S,Pr'\}
\]

\(\text{syn-att}(S) = \text{syn-att}(E) = "\text{type}" , "\text{val}"\)

"plus", "mult", and "apply" are the same functions as defined in the source language specifications.

Ru' contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
</table>
| \(S<E>\)   | val(x0) = val(x1)  
            | type(x0) = type(x1) |
| \(E<EE+>\) | if type(x1) = type(x2) then  
            | val(x3) = plus(val(x1),val(x3))  
            | type(x0) = type(x1)  
            | else  
            | val(x0) = type(x0) = "err" |
| \(E<E C(E)+>\) | if type(x1) = "ratn" and  
                | type(x2) = "int" then  
                | val(x3) = plus(cvir(val(x1)),val(x2)))  
                | type(x0) = "ratn"  
                | else  
                | val(x0) = type(x0) = "err" |
| \(E<C(E) E+>\) | if type(x1) = "int" and  
                | type(x2) = "ratn" then  
                | val(x3) = plus(cvir(val(x1)),val(x2)))  
                | type(x0) = "ratn"  
                | else  
                | val(x0) = type(x0) = "err" |
| \(E<EE*>\)   | if type(x1) = type(x2) then  
            | val(x0) = mult(val(x1),val(x2))  
            | type(x0) = type(x1)  
            | else  
            | val(x0) = type(x0) = "err" |
E<E C(E)*> if type(x1) = "ratn" and
    type(x2) = "int" then
    val(x0) = mult(val(x1), cvir(val(x2)))
    type(x0) = "ratn"
else
    val(x0) = type(x0) = "err"

E<C(E) E*> if type(x1) = "int" and
    type(x2) = "ratn" then
    val(x0) = mult(cvir(val(x1), val(x2)))
    type(x0) = "ratn"
else
    val(x0) = type(x0) = "err"

E<1> val(x0) = 1
    type(x0) = "int"

E<1.> val(x0) = 1.
    type(x0) = "ratn"

E<2> val(x0) = 2
    type(x0) = "int"

E<2.> val(x0) = 2.
    type(x0) = "ratn"

ASSOC mapping is defined by:

S → S       E → E
## CORRESPONDENCE TABLE

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Condition</th>
<th>X-Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>S E</td>
<td>S E</td>
<td>always</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>if type(x1) = type(x2)</td>
<td>1, 2, 0</td>
</tr>
<tr>
<td>E E O</td>
<td>E E +</td>
<td>if type(x1) = &quot;int&quot;</td>
<td>0, 0, 1, 0, 2, 0</td>
</tr>
<tr>
<td></td>
<td>C(E) E +</td>
<td>and type(x2) = &quot;ratn&quot;</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>E C(E) +</td>
<td>if type(x1) = &quot;ratn&quot;</td>
<td>1, 0, 0, 2, 0, 0</td>
</tr>
<tr>
<td></td>
<td>C(E) E *</td>
<td>and type(x2) = &quot;int&quot;</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>E E *</td>
<td>if type(x1) = type(x2)</td>
<td>1, 2, 0</td>
</tr>
<tr>
<td>E E O</td>
<td>E E *</td>
<td>if type(x1) = &quot;int&quot;</td>
<td>0, 0, 1, 0, 2, 0</td>
</tr>
<tr>
<td></td>
<td>C(E) E *</td>
<td>and type(x2) = &quot;ratn&quot;</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>E C(E) *</td>
<td>if type(x1) = &quot;ratn&quot;</td>
<td>1, 0, 3, 2, 0, 0</td>
</tr>
<tr>
<td></td>
<td>C(E) E *</td>
<td>and type(x2) = &quot;int&quot;</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>always</td>
<td>0</td>
</tr>
<tr>
<td>Source</td>
<td>Target</td>
<td>Condition</td>
<td>X-Vector</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>E 1</td>
<td>E 1</td>
<td>always</td>
<td>Ø</td>
</tr>
<tr>
<td>E 2</td>
<td>E 2</td>
<td>always</td>
<td>Ø</td>
</tr>
<tr>
<td>E 2.</td>
<td>E 2.</td>
<td>always</td>
<td>Ø</td>
</tr>
</tbody>
</table>
STEP-BY-STEP TRANSLATION

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Row</th>
</tr>
</thead>
<tbody>
<tr>
<td>step 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>step 1:</td>
<td></td>
<td>row 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>step 2:</td>
<td></td>
<td>row 2.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>step 3:</td>
<td></td>
<td>row 6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Source Sentence: 22.1**
Target Sentence: C(2)2.C(1)**
Note that by T.6.1, this transducer is semantic preserving.
APPENDIX D

Predicate Phrase-Structure Translator: pst-4

**Source language:** Postfix Arithmetic Expressions  
**Target language:** Postfix Arithmetic Expressions

Both source and target languages are defined over the integer 1, the rational 1., and the operators + and *.

**Source Language Specifications**

The source system of this example is the source system of pst-3 in Appendix C except that the last two rules involving the productions E<2> and E<2.> have been omitted.

**Target Language Specifications**

Target system is \((G, SEM)\) where

\[ G = ((S, ER, EI, O), (1, 1., +, *) , S, Pr) \]

and

\[
\begin{align*}
\text{att}(S) & = \text{"val", "type"} \\
\text{att}(EI) & = \text{att}(ER) = \text{"val"} \\
\text{att}(O) & = \text{"op"}
\end{align*}
\]

Ru contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
</table>
| S<ER>      | val(x0) = val(x1)  
|            | type(x0) = "ratn"  |
| S<EI>      | val(x0) = val(x1)  
|            | type(x0) = "int"   |
| ER<ER ER O>| val(x0) = apply(op(x3), val(x1),  
|            | val(x2))           |
\[ \text{EI<EI EI O> val(x0) = apply(op(x3), val(x1), val(x2))} \]

\[ \text{ER<ER EI O> val(x0) = apply(op(x3), val(x1), cvir(val(x2)))} \]

\[ \text{ER<EI ER O> val(x0) = apply(op(x3), cvir(val(x1)), val(x2))} \]

\[ \text{O<+> op(x0) = "plus"} \]

\[ \text{O <*> op(x0) = "mult"} \]

\[ \text{EI<l> val(x0) = 1} \]

\[ \text{ER<l> val(x0) = 1.} \]

ASSOC mapping is defined by:

\[ S \rightarrow S \quad 0 \rightarrow 0 \]

Note that associating \( E \) with \( ER \) or \( E \) with \( EI \) would be wrong. \( E \) represents both the rational expressions and the integer expressions. \( ER \) represents only the rational expressions, \( EI \) represents only the integer expressions. This is why no lipst can translate from source to target language using these two systems.
## CORRESPONDENCE TABLE

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Condition</th>
<th>X-Vec</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>S</td>
<td>always</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ER</td>
<td>if type(x1)= type(x2)=&quot;ratn&quot;</td>
<td>1,2,0</td>
</tr>
<tr>
<td>E</td>
<td>ER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>ER</td>
<td>if type(x1)=&quot;ratn&quot; and type(x2)=&quot;int&quot;</td>
<td>1,2,0</td>
</tr>
<tr>
<td></td>
<td>EI</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ER</td>
<td>if type(x1)=&quot;int&quot; and type(x2)=&quot;ratn&quot;</td>
<td>1,2,0</td>
</tr>
<tr>
<td></td>
<td>EI</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EI</td>
<td>if type(x1)=&quot;int&quot; and type(x2)=&quot;int&quot;</td>
<td>1,2,0</td>
</tr>
<tr>
<td>Source</td>
<td>Target</td>
<td>Condition</td>
<td>X-Vec</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>-----------</td>
<td>-------</td>
</tr>
</tbody>
</table>
| E E *  | ER     | if type(x1)=
|        |        | type(x2)="ratn" | 1,2,0 |
|        | ER     | if type(x1)="ratn" |
|        |        | and type(x2)="int" | 1,2,0 |
|        | EI     | if type(x1)="int" |
|        |        | and type(x2)="ratn" | 1,2,0 |
|        | EI     | if type(x1)="int" |
|        |        | and type(x2)="int" | 1,2,0 |
| E I    | EI     | always | 0 |
| I I    |       |       |     |
| E I    | ER     | always | 0 |
| I I    |       |       |     |
APPENDIX E

Predicate Phrase-Structure Translator: pst-5

**Source language:** integers in unary notation
**Target language:** \((\emptyset, 1)\)

For the source language, \(\text{mean}(n) = \emptyset\) if \(n\) is not a perfect square and \(\text{mean}(n) = 1\) if \(n\) is a perfect square. For the target language, \(\text{mean}(\emptyset) = \emptyset\) and \(\text{mean}(1) = 1\).

**Source Language Specification**

Source system is \((G, SEM)\) where

\[ G = ((S, E), \{1\}, S, Pr) \]

\(\text{syn-att}(S) = \text{syn-att}(E) = "\text{val}"\)

Ru contains the follows production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
</table>
| \(S<E>\)  | If \(\text{val}(x1)\) is a perfect square then \(\text{val}(x0) = 1\)
|            | else \(\text{val}(x0) = \emptyset\) |
| \(E<1E>\) | \(\text{val}(x0) = \text{plus}(1, \text{val}(x1))\) |
| \(E<1>\)  | \(\text{val}(x0) = 1\) |
Target Language Specifications

Target system is $(G', \text{SEM}')$ where

$$G' = (S, (\emptyset, 1), S, Pr')$$ and

$$\text{syn-att}(S) = "val"$$

Ru' contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S&lt;\emptyset&gt;$</td>
<td>$\text{val}(x\emptyset) = \emptyset$</td>
</tr>
<tr>
<td>$S&lt;1&gt;$</td>
<td>$\text{val}(x1) = 1$</td>
</tr>
</tbody>
</table>

ASSOC mapping is defined by:

$$S \rightarrow S$$

CORRESPONDENCE TABLE

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Condition</th>
<th>X-Vec</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>$S$</td>
<td>if val($x1$) is a perfect square</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;1&gt;$</td>
<td>$&lt;1&gt;$</td>
<td>if val($x1$) is not a perfect square</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;\emptyset&gt;$</td>
<td>$&lt;\emptyset&gt;$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX F

Predicate Phrase-Structure Translator: pst-6

Source language: binary numbers
Target language: ternary numbers

Within the specification of the semantics of the source language is a "translation" function "cvrtbt" which converts the base 2 representation of the number into its base 3 representation. Thus, all of the real work of the translation is "hidden" within the semantics. This makes it possible to set up a ppst which can induce a binary to ternary translation.

Source Language Specification

Source system is (G,SEM) where

\[ G = ((S,E),(0,1),S,Pr) \]

and

\[
\begin{align*}
\text{syn-att}(S) &= "mn"; \quad \text{syn-att}(E) = "mn", "scale" \\
\text{inh-att}(E) &= "dig"
\end{align*}
\]

Ru contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
</table>
| S<E>       | \[
\begin{align*}
\text{mn}(x0) &= \text{mn}(x1) \\
\text{dig}(x1) &= \text{cvrtbt}(\text{val}(x0))
\end{align*}
\] |
| E<1E>      | \[
\begin{align*}
\text{mn}(x0) &= 2**\text{scale}(x0) + \text{mn}(x2) \\
\text{scale}(x0) &= \text{scale}(x2) + 1
\end{align*}
\] |
| E<0E>      | \[
\begin{align*}
\text{mn}(x0) &= \text{mn}(x2) \\
\text{scale}(x0) &= \text{scale}(x0) + 1 \\
\text{dig}(x2) &= \text{substr}(\text{dig}(x0),2)
\end{align*}
\] |
E<0>  \quad mn(x0) = 0 \\
    \quad scale(x0) = 0 \\

E<1>  \quad mn(x0) = 1 \\
    \quad scale(x0) = 0 \\

**Target Language Specification**

Target system is \((G', SEM')\) where

\[ G' = ((S,E),(0,1,2),S,Pr') \]

\[ \text{syn-att}(S) = "mn"; \quad \text{syn-att}(E) = "mn", "scale" \]

\(Ru'\) contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>S&lt;E&gt;</td>
<td>(mn(x0) = mn(x1))</td>
</tr>
</tbody>
</table>
| E<0E>      | \(mn(x0) = mn(x2)\)  \\
|            | \(scale(x0) = scale(x0) + 1\) |
| E<1E>      | \(mn(x0) = 3^{scale(x0)} + mn(x2)\)  \\
|            | \(scale(x0) = scale(x0) + 1\) |
| E<2E>      | \(mn(x0) = 2(3^{scale(x0)}) + mn(x2)\)  \\
|            | \(scale(x0) = scale(x0) + 1\) |
| E<0>       | \(mn(x0) = 0\)  \\
|            | \(scale(x0) = 0\) |
| E<1>       | \(mn(x0) = 1\)  \\
|            | \(scale(x0) = 0\) |
| E<2>       | \(mn(x0) = 2\)  \\
|            | \(scale(x0) = 0\) |
ASSOC mapping is defined by:

S \rightarrow S \quad E \rightarrow E
<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Condition</th>
<th>X-Vec</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>S</td>
<td>always</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>if ( \text{digl}(\text{dig}(x0)) = 0 ) and (</td>
<td>\text{dig}</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>if ( \text{digl}(\text{dig}(x0)) = 0 ) and (</td>
<td>\text{dig}</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>if ( \text{digl}(\text{dig}(x0)) = 1 ) and (</td>
<td>\text{dig}</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>if ( \text{digl}(\text{dig}(x0)) = 1 ) and (</td>
<td>\text{dig}</td>
</tr>
<tr>
<td>OR</td>
<td>E</td>
<td>if ( \text{digl}(\text{dig}(x0)) = 2 ) and (</td>
<td>\text{dig}</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>if ( \text{digl}(\text{dig}(x0)) = 2 ) and (</td>
<td>\text{dig}</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>if ( \text{digl}(\text{dig}(x0)) = 2 ) and (</td>
<td>\text{dig}</td>
</tr>
</tbody>
</table>

where \( \text{digl}(\text{dig}(x0)) \) is the first character of \( v(\text{dig}(x0)) \).
Incomplete Phrase-Structure Translator: pst-7

Source language: binary numbers
Target language: ternary numbers

Pst-7 is pst-6 converted into an ipst using A.6.1.

Source Language Specification

Source system is (G,SEM) where

\[ G = ((S,E,E0,E1,E2), (0,1), S, Pr) \]

and

\[
\begin{align*}
\text{syn-att}(S) &= "mn", \text{syn-att}(E) = "mn", \text{"scale"} \\
\text{syn-att}(E_i) &= "mn", \text{"scale"} \\
&\quad \text{for } i = 0,1,2 \\
\text{inh-att}(E) &= "dig" \\
\text{inh-att}(E_i) &= "dig", \text{"proper"} \\
&\quad \text{for } i = 0,1,2
\end{align*}
\]

Ru contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
</table>
| S<E>       | \[
\begin{align*}
\text{mn}(x0) &= \text{mn}(x1) \\
\text{dig}(x1) &= \text{cvrctb}(\text{val}(x0))
\end{align*}
\] |
| E<Ej>      | for \(i = 0,1\) \[
\begin{align*}
\text{mn}(x0) &= \text{mn}(x1) \\
\text{dig}(x1) &= \text{dig}(x0) \\
\text{scale}(x0) &= \text{scale}(x1)
\end{align*}
\] |
| E<j E>     | for \(i = 0,1\) and \(j = 0,1,2\) \[
\begin{align*}
\text{mn}(x1) &= i \times (2**\text{scale}(x1)) + \text{mn}(x2) \\
\text{scale}(x0) &= \text{scale}(x2) + 1 \\
\text{dig}(x2) &= \text{substr}(\text{dig}(x0), 2) \\
\text{if } \text{substr}(\text{dig}(x0), 1, 1) &= j \text{ and} \\
|\text{dig}(x0)| &> 1 \text{ then} \\
\text{proper}(x0) &= "ok" \\
\text{else} \\
\text{proper}(x0) &= "err"
\end{align*}
\] |
Ej<1> for i = 0,1 and j = 0,1,2
val(x0) = i
dig(x1) = dig(x0)
scale(x0) = ∅
if substr(dig(x1),1,1) = j
   and |dig(x1)| = 1 then
   proper(x1) = "ok"
else
   proper(x1) = "err"
<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>X-Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

for $i=0,1$

for $j=0,1,2$

for $i=0,1$

for $j=0,1,2$
APPENDIX G

Block Phrase-Structure Translator: pst-8

**Source language**: Programs with assignment and declaration statements
**Target language**: Programs with assignment and declaration statements

Because we are interested in emphasizing the fact that the use of a bpst conveniently permits us to translate between languages which place statements at different points in the program, and because we do not care in this example about the details of how the declaration and assignment statements are written, we will specify only part of the syntactic definition of the language. After having seen the many examples presented in the text and the earlier appendices, the reader should have an understanding of how the detailed language definition would look.

**Source Language Specification**

The source language follows the PL/I philosophy towards declaration placement; i.e., a declaration may appear anywhere in the program. It follows the ALGOL philosophy of requiring a declaration for every variable used in the program. The two data-types are "integer" and "rational". Source system is \((G, \text{SEM})\) where

\[
G = ((AX, P, DEC, ASN), (..), AX, Pr) \quad \text{and}
\]

Pr contains the following productions:

<table>
<thead>
<tr>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX&lt;P STOP&gt;</td>
</tr>
<tr>
<td>P&lt;DEC P&gt;</td>
</tr>
<tr>
<td>P&lt;ASN P&gt;</td>
</tr>
<tr>
<td>P&lt;DEC&gt;</td>
</tr>
<tr>
<td>P&lt;ASN&gt;</td>
</tr>
<tr>
<td>DEC&lt;..&gt;</td>
</tr>
<tr>
<td>ASN&lt;..&gt;</td>
</tr>
</tbody>
</table>
Target Language Specification

The target language follows the philosophy of FORTRAN in the placement of declarations; i.e., every declaration must occur at the beginning of the program. As is the case of the source language, every variable used in the program must be declared.

Target system is \( (G', \text{SEM}') \) where

\[
G' = \langle \langle AX, DB, AB, DEC, ASN \rangle, \langle \ldots \rangle, AX, Pr' \rangle
\]

where \( Pr' \) contains the following productions:

<table>
<thead>
<tr>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX\langle DB AB END \rangle</td>
</tr>
<tr>
<td>DB\langle DEC DB \rangle</td>
</tr>
<tr>
<td>AB\langle ASN AB \rangle</td>
</tr>
<tr>
<td>DB\langle DEC \rangle</td>
</tr>
<tr>
<td>AB\langle ASN \rangle</td>
</tr>
<tr>
<td>AB\langle i \rangle</td>
</tr>
<tr>
<td>DB\langle i \rangle</td>
</tr>
<tr>
<td>DEC\langle \ldots \rangle</td>
</tr>
<tr>
<td>ASN\langle \ldots \rangle</td>
</tr>
</tbody>
</table>

ASSOC mapping is defined by:

\[
\begin{align*}
\{AX\} & \rightarrow \{AX\} \\
\{P\} & \rightarrow \{DB, AB\} \\
\{DEC\} & \rightarrow \{DEC\} \\
\{ASN\} & \rightarrow \{ASN\}
\end{align*}
\]
<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>$x$ and $x'$</th>
<th>Row</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{AX}$</td>
<td>$\text{DB} \quad \text{AB \quad END}$</td>
<td>$\begin{bmatrix} 1 \ 1 \end{bmatrix}$</td>
<td>1</td>
</tr>
<tr>
<td>$\text{P}$</td>
<td>$\text{DB} \quad \text{AB}$</td>
<td>$\begin{bmatrix} 1 &amp; 0 \ 0 &amp; 1 \end{bmatrix}$</td>
<td>2.1</td>
</tr>
<tr>
<td>$\text{DEC \quad P}$</td>
<td>$\text{DB} \quad \text{AB}$</td>
<td>$\begin{bmatrix} 1 &amp; 0 \ 0 &amp; 1 \end{bmatrix}$</td>
<td>2.2</td>
</tr>
<tr>
<td>$\text{P}$</td>
<td>$\text{DB} \quad \text{AB}$</td>
<td>$\begin{bmatrix} 1 &amp; 0 \ 0 &amp; 1 \end{bmatrix}$</td>
<td>3.1</td>
</tr>
<tr>
<td>$\text{ASN \quad P}$</td>
<td>$\text{AB}$</td>
<td>$\begin{bmatrix} 1 &amp; 0 \ 0 &amp; 1 \end{bmatrix}$</td>
<td>3.2</td>
</tr>
<tr>
<td>$\text{P}$</td>
<td>$\text{DB} \quad \text{AB}$</td>
<td>$\begin{bmatrix} 1 &amp; 0 \end{bmatrix}$</td>
<td>4</td>
</tr>
<tr>
<td>$\text{DEC}$</td>
<td>$\text{AB}$</td>
<td>$\begin{bmatrix} 1 &amp; 0 \end{bmatrix}$</td>
<td>5</td>
</tr>
<tr>
<td>$\text{P}$</td>
<td>$\text{DB} \quad \text{AB}$</td>
<td>$\begin{bmatrix} 1 &amp; 0 \end{bmatrix}$</td>
<td>6</td>
</tr>
<tr>
<td>$\text{ASN}$</td>
<td>$\text{AB}$</td>
<td>$\begin{bmatrix} 1 &amp; 0 \end{bmatrix}$</td>
<td>7</td>
</tr>
<tr>
<td>$\text{DEC}$</td>
<td>$\text{DEC}$</td>
<td>$\begin{bmatrix} 1 &amp; 0 \end{bmatrix}$</td>
<td>8</td>
</tr>
<tr>
<td>$\text{ASN}$</td>
<td>$\text{ASN}$</td>
<td>$\begin{bmatrix} 1 &amp; 0 \end{bmatrix}$</td>
<td>9</td>
</tr>
</tbody>
</table>
STEP-BY-STEP TRANSLATION

step 0

\[
\begin{align*}
&\text{Ax} \\
&\quad P \quad \text{stop} \\
&\quad \quad \text{ASN} \\
&\quad \quad \quad \text{P} \\
&\quad \quad \quad \quad \text{dcl a int} \\
&\quad \quad \quad \quad \quad \text{ASN} \\
&\quad \quad \quad \quad \quad \quad \text{P} \\
&\quad \quad \quad \quad \quad \quad \quad \text{DEC} \\
&\quad \quad \quad \quad \quad \quad \quad \quad a=4+c \\
&\quad \quad \quad \quad \quad \quad \quad \quad \text{DEC} \\
&\quad \quad \quad \quad \quad \quad \quad \quad \quad \text{dcl c real}
\end{align*}
\]

step 1

\[
\begin{align*}
&\text{Ax} \\
&\quad P \quad \text{stop} \\
&\quad \quad \text{DB} \quad \text{AB} \quad \text{end}
\end{align*}
\]

step 2

\[
\begin{align*}
&\text{Ax} \\
&\quad P \quad \text{stop} \\
&\quad \quad \text{ASN} \\
&\quad \quad \quad \text{P} \\
&\quad \quad \quad \quad (3.1) \quad (3.2) \\
&\quad \quad \quad \quad \quad \text{DB} \quad \text{AB} \quad \text{end} \\
&\quad \quad \quad \quad \quad \quad \text{ASN} \quad \text{AB} \\
&\quad \quad \quad \quad \quad \quad \quad (3.1) \quad (3.2)
\end{align*}
\]
step 6

Ax → P stop
→
ASN

P
c=2.: DEC
→
dcl a int
ASN

P

step 7

Ax → P stop
→
ASN
c=2.

P

DEC
→
dcl a int
ASN

P

a=4+c
step 8

row 4
Source sentence: c = 2.
dcl a int
a = 4 + c
dcl c real

Target sentence: integer a
real c
c = 2.
a = 4 + c
APPENDIX H

Leggett's Theorem

Theorem (Leggett): No finitely specified translator which does not have deviant parses can translate between \( L_1 = \{(n,1) \mid n \text{ is a perfect square}\} \cup \{(n,0) \mid n \text{ is not a perfect square}\} \)

and \( L_2 = \{(0,0),(1,1)\} \)

Pumping Lemma (Bar-Hillel, Perles, and Shamir [61]): Let \( L \) be any context-free language. There are constants \( m \) and \( n \) depending only on \( L \), such that if there is a word \( z \) in \( L \), with \( |z| > m \), then \( z \) may be written as \( z = wxyz \), where \( |xy| \leq n \) and \( w \) and \( z \) are not both \( \varepsilon \), the empty string, such that for each integer \( i \geq 0 \),

\[ w^i x y^i z \]

is in \( L \).

Proof of theorem: Buttelmann's notations for finitely specified translators are somewhat different than ours. For clarity, we have converted Buttelmann's notation to be consistent with that used for phrase-structure systems and translators.

Suppose by way of contradiction that there is a finitely specified translator which induces the above translation. Any phrase-structure system for \( L_2 \) must have as an underlying grammar, a grammar of the form, 

\[ G_2 = (Vn, Vt, Ax, Pr) \]

where

\[ Vn_2 = \{S_1, \ldots, S_k\} \text{ for some integer } k; \text{ and} \]

\[ Vt_2 = (\emptyset, 1); \text{ and} \]

\[ Ax_2 = S_1; \text{ and} \]
\[ \text{Pr2} = (S(i,1)<S(i,2)>, \ldots, S(i,z-1)<S(i,z)>) \]
\[ (S(j,1)<\emptyset>, \ldots, S(j,z)<\emptyset>) \]
\[ (S(k,1)<i>, \ldots, S(k,z'<i>) \]

\( \text{sen}(L1) \) is an infinite set. Furthermore, \( L1 \) can be partitioned into two disjoint sets:

\[
\begin{align*}
\text{La} &= \{(n,\emptyset) \mid n \text{ is not a perfect square}) \\
\text{Lb} &= \{(n,1) \mid n \text{ is a perfect square})
\end{align*}
\]

Both \( \text{sen}(La) \) and \( \text{sen}(Lb) \) are strictly context-sensitive languages (where "language" is used in the classical sense, without reference to meanings). Each string in \( La \) is mapped to \((\emptyset,\emptyset)\) in \( L2 \) and each string in \( Lb \) is mapped to \((1,1)\) in \( L2 \).

Suppose \( w \) is a string in \( L1 \) whose length is greater than the integer \( m \) of the pumping lemma. We claim that if \( M \) exists, then there is an \( M' \) which is both weakly equivalent to \( M \) and also strong. For suppose not. Consider a parse \( t \) of \( w \) and \( t' \), the phrase-transform of \( t \). The frontier of \( t' \) is either \( \emptyset \) or \( 1 \) depending upon whether or not \( \text{fr}(t) \) is in \( La \) or \( Lb \). If \( M \) is not strong, then the leftmost partial derivation of \( t \) which induces \( t' \) could be such that it is not also a leftmost derivation of \( t \). Suppose so. Then \( t \) has a box bracketing \( r(fl..q..fm) \) where \( \text{label}(fi) \in Vt \), subtree \( r \) by itself induces \( t' \) and \( q \) is a terminated sub-structure of \( t \). \( q \) has a sub-structure of the form

\[ s[..s..] \]

where \( s \) is a \( s\)-TABL entry. Otherwise, \( q \) could have been incorporated as a sub-structure into some other \( s\)-TABL entry. From the pumping lemma we know that the frontier of \( q \), call it \( \mathcal{A} \), can be written as \( \mathcal{A} \mathcal{A} \mathcal{A} \mathcal{A} \) such that \( v(i) = \mathcal{A} \mathcal{A} \mathcal{A} \mathcal{A} \) is in the language for \( i > 0 \), and for \( i = 0 \), the translation of \( w \) is the translation of \( v(i) \). But in general, as \( i \) varies, \( v(i) \) will sometimes be in \( La \) and sometimes in \( Lb \). Therefore, the translator is not semantic-preserving. A contradiction. Therefore, a strong translator exists if any does. Let \( M' \) be such a translator. Assume it is in normal form. \( t' \) is induced by no proper sub-tree of \( t \). We can again apply the pumping lemma to \( t \). As \( i \) increases, the frontier of the transform remains unchanged, but \( v(i) \) grows by \( \emptyset \mathcal{A} \mathcal{A} \mathcal{A} \mathcal{A} \). In general, \( v(i) \) will sometimes be in \( La \)
and sometimes in Lb. Therefore, $M'$ is not semantic-preserving. A contradiction. Therefore, $M'$ does not exist and the theorem follows.

[*]
APPENDIX I

Block Phrase-Structure Translator: pst-10

**Source language:** \(((w,w) \mid w \in (a,b)^+\)

**Target language:** Define a homomorphism

\[ h: (a,b) \rightarrow (a',b') \]

such that \( h(a) = a', \ h(b) = b' \). Extend \( h \) to \((a,b)^+\) in the natural way; i.e., if \( x \in (a,b)^+ \) then \( h(ax) = a' \| h(x) \) and \( h(bx) = b' \| h(x) \). The target language is

\[
\{(ww',w) \mid w \in (a,b) \wedge h(w) = w'\}
\]

\[
\{(ww',\ast) \mid w \in (a,b)^+, h(w) \neq h(w')\}.
\]

**Source Language Specification**

Source system is \((G,SEM)\) where

\[
G = ((S,E),(a,b),S,Pr) \text{ and } syn-att(S) = syn-att(E) = "val"
\]

Ru contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S&lt;E&gt;)</td>
<td>(\text{val}(x0) = \text{val}(x1))</td>
</tr>
<tr>
<td>(E&lt;aE&gt;)</td>
<td>(\text{val}(x0) = a | \text{val}(x2))</td>
</tr>
<tr>
<td>(E&lt;bE&gt;)</td>
<td>(\text{val}(x0) = b | \text{val}(x2))</td>
</tr>
<tr>
<td>(E&lt;a&gt;)</td>
<td>(\text{val}(x0) = a)</td>
</tr>
</tbody>
</table>
\( E_b \quad \text{val}(x_0) = b \)
**Target Language Specification**

Target system is $(G', SEM')$ where

$G' = ((S, E, F), (a, b, a', b'), S, Pr')$ and

$\text{syn-att}(S) = \text{syn-att}(F) = \text{syn-att}(E) = "val"

Ru' contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S&lt;EF&gt;$</td>
<td>if $h(\text{val}(x1)) = \text{val}(x2)$ then $\text{val}(x3) = \text{val}(x1)$ else $\text{val}(x3) = '*'$</td>
</tr>
<tr>
<td>$E&lt;a E&gt;$</td>
<td>$\text{val}(x0) = \text{all}\text{val}(x2)$</td>
</tr>
<tr>
<td>$E&lt;b E&gt;$</td>
<td>$\text{val}(x0) = \text{blt}\text{val}(x2)$</td>
</tr>
<tr>
<td>$E&lt;a&gt;$</td>
<td>$\text{val}(x0) = a$</td>
</tr>
<tr>
<td>$E&lt;b&gt;$</td>
<td>$\text{val}(x0) = b$</td>
</tr>
<tr>
<td>$F&lt;a' F&gt;$</td>
<td>$\text{val}(x0) = a'\text{ll}\text{val}(x2)$</td>
</tr>
<tr>
<td>$F&lt;b' F&gt;$</td>
<td>$\text{val}(x0) = b'\text{ll}\text{val}(x2)$</td>
</tr>
<tr>
<td>$F&lt;a'&gt;$</td>
<td>$\text{val}(x0) = a'$</td>
</tr>
<tr>
<td>$F&lt;b'&gt;$</td>
<td>$\text{val}(x0) = b'$</td>
</tr>
</tbody>
</table>

ASSOC mapping is defined by:

$S \rightarrow S \quad E \rightarrow (E, F)$
APPENDIX J

Semantic-Directed Translator: sdt-1

**Source language**: Binary numbers.
**Target language**: Ternary numbers.

For both source and target languages, the meaning of a number is the number itself, but written in decimal notation.

**Source Language Specification**

Source system is \((G, \text{SEM})\) where
\[
G = ((S,T), (\emptyset, 1), S, Pr) \quad \text{and}
\]
Ru contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>S&lt;T&gt;</td>
<td>( mn = mn(xl) )</td>
</tr>
<tr>
<td>T&lt;T 0&gt;</td>
<td>( mn = 2*mn(xl) )</td>
</tr>
<tr>
<td>T&lt;T 1&gt;</td>
<td>( mn = 2*mn(xl) + 1 )</td>
</tr>
<tr>
<td>T&lt;Ø&gt;</td>
<td>( mn = 0 )</td>
</tr>
<tr>
<td>T&lt;1&gt;</td>
<td>( mn = 1 )</td>
</tr>
</tbody>
</table>
Target Language Specification

Target ipss is \((G', \text{SEM}')\) where

\[
G' = (\{S', T', D'\}, \{0, 1, 2\}, S, P_{r'})
\]

att-val is defined by:

\[
\begin{align*}
\text{att-val}(S') &= U \\
\text{att-val}(T') &= U \\
\text{att-val}(D') &= \{0, 1, 2\} \\
\text{att-val}(0) &= 0 \\
\text{att-val}(1) &= 1 \\
\text{att-val}(2) &= 2
\end{align*}
\]

Ru' contains the following production-rule pairs:

\[
\begin{align*}
S' &< T' \quad \text{mn} = \text{mn}(x0) \\
T' &< T' D' \quad \text{mn} = (\text{mn}(x0)/2, \text{mn}(x0)-\text{mn}(x0)/2) \\
T' &< D' \quad \text{if} \ \text{mn}(x0) < 3 \ \text{then} \ \text{mn} = \text{mn}(x0) \\
&\quad \text{else} \ \text{mn is undefined} \\
D' &< \emptyset \quad \text{if} \ \text{mn}(x0) = 0 \ \text{then} \ \text{mn} = 0 \\
&\quad \text{else} \ \text{mn is undefined} \\
D' &< 1 \quad \text{if} \ \text{mn}(x0) = 1 \ \text{then} \ \text{mn} = 1 \\
&\quad \text{else} \ \text{mn is undefined} \\
D' &< 2 \quad \text{if} \ \text{mn}(x0) = 2 \ \text{then} \ \text{mn} = 2 \\
&\quad \text{else} \ \text{mn is undefined}
\end{align*}
\]

Translate From Binary to Ternary

Suppose the source sentence is "101". Call TRANS("101", BN, TN), where BN is the source system, and TN is the target system defined above. Using G of BN, parse tree t is obtained by calling COMPUTE-MN("101").
When $\text{GEN}(5, S', TN)$ is called, tree $t'$ could be generated:

```
S'  GEN(5, S', TN)
  |
T'  GEN(5, T', TN)
  |
T'  GEN(1, T', GEN)  2
  |
  |
```
APPENDIX K

Enumeration of \( \text{phr}(G) \)

The following procedure will enumerate in weight-increasing order, the phrase-structures of a context-free grammar. It has the additional property that if

\[
t_0[t_1..t_i[y_l..y_m]..t_k]
\]

is enumerated before

\[
t_0[t_1..t_i[y'_l..y'_m]..t_k]
\]

is enumerated, then

\[
t_i[y_l..y_m]
\]

is enumerated before

\[
t_i[y'_l..y'_m]
\]

is enumerated.

Procedure K.

**Purpose:** Enumerate \( \text{phr}(G) \).

**Input:** Context-free grammar \( G = (V_n,V_t,A,x,Pr) \).

**Output:** The phrase-structures of \( G \).

**Steps:**

1) \( K = |Pr|; \)
2) \( \text{DO} \ I = 1 \ \text{TO} \ K; \)
3) \( \text{TR}(I) = Pr(I); \)
4) \( \text{END}; \)
5) \( Z = K; \)
6) \( A: \)
   \[
   \text{DO } I = 1 \text{ TO } K; \]
7) \( \text{DO } J = 1 \text{ IFR } (\text{TR}(I)) \text{ TO } 1; \)
8) \( \text{DO } L = 1 \text{ TO } \text{PR(L)}; \)
9) \( \text{IF LABEL} (\text{RT(PR(L))}) = \text{FR} (\text{TR}(I), J) \)
10) \( \text{THEN DO}; \)
11) \( Z = Z + 1; \)
12) \( \text{TR}(Z) = \text{COMPOSE} (\text{TR(I)}, \text{PR(L)}, J); \)
13) \( \text{END}; \)
14) \( \text{END}; \)
15) \( \text{END}; \)
16) \( \text{END}; \)
17) \( K = Z; \)
18) \( \text{GO TO A}; \)
19) \( \text{END}; \)
APPENDIX L

Semantic-Directed Translator: sdt-2

Source language: Assignment programs.
Target language: Assignment programs.

The source and target languages are identical. To illustrate the point of this example, we translate a language to itself. The language contains just two types of statements, an assignment statement and the null statement.

Source Language Specification

Source system is (G, SEM) where

\[ G = ((\text{AX}, \text{P}, \text{ASN}, \text{ID}), (\cdot, \text{A}, \text{B}, 1, 2, =), \text{AX}, \text{Pr}) \]

and Ru contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{AX}\langle \text{P} \rangle</td>
<td>\text{mn}(x_0) = \text{mn}(x_1)</td>
</tr>
<tr>
<td>\text{P}\langle \text{ASN} \rangle \langle \text{P} \rangle</td>
<td>\text{mn}(x_0) = \text{mn}(x_1) \text{ } \text{II} \text{ } \text{mn}(x_2)</td>
</tr>
<tr>
<td>\text{P}\langle \text{ASN} \rangle</td>
<td>\text{mn}(x_0) = \text{mn}(x_1)</td>
</tr>
<tr>
<td>\text{ASN}\langle \cdot \rangle</td>
<td>\text{mn}(x_0) = \varepsilon</td>
</tr>
<tr>
<td>\text{ASN}\langle \text{ID} = \text{E} \rangle</td>
<td>\text{mn}(x_0) = (\text{mn}(x_1), \text{mn}(x_3))</td>
</tr>
<tr>
<td>\text{ID}\langle \text{A} \rangle</td>
<td>\text{mn}(x_0) = &quot;a&quot;</td>
</tr>
</tbody>
</table>
Target Language Specification

Target wipss is \((G', SEM')\) where

\[
G' = (\{Ax, P, ASN, ID, E\}, (A, B, 1, 2, ;, \backslash, =), Ax, Pr')
\]

and \text{att-val} \ is \ defined \ by:

\[
\text{att-val}(ax) = ((a, b), (1, 2)) + \\
\text{att-val}(p) = (a, b)x(1, 2) + \\
\text{att-val}(asn) = (a, b)x(1, 2) \\
\text{att-val}(id) = (a, b) \\
\text{att-val}(e) = (1, 2) \\
\text{att-val}(a) = (a) \\
\text{att-val}(b) = (b) \\
\text{att-val}(1) = 1 \\
\text{att-val}(2) = 2 \\
\text{att-val}(;') = \varepsilon
\]

\(Ru'\) contains the following production-rule pairs:

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX&lt;P&gt;</td>
<td>(mn = mn(x0))</td>
</tr>
<tr>
<td>P&lt;ASN P&gt;</td>
<td>(mn = (\varepsilon, mn(x0)))</td>
</tr>
<tr>
<td>P&lt;ASN&gt;</td>
<td>if (|mn(x0)| \leq 1) then (mn = mn(x0)) else (mn = &quot;err&quot;)</td>
</tr>
<tr>
<td>Production</td>
<td>Semantic Rule</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
</tr>
<tr>
<td>ASN&lt;ID = E&gt;</td>
<td>if ( \text{mn}(x0) \neq \varepsilon ) then  \  \  ( \text{mn}(x1) = \text{first-member}(\text{mn}(x0)) )  \  ( \text{mn}(x3) = \text{second-member}(\text{mn}(x0)) )  \  else  \  ( \text{mn}(x1) = \text{mn}(x3) = &quot;err&quot; )</td>
</tr>
<tr>
<td>ASN&lt;;&gt;</td>
<td>if ( \text{mn}(x0) = \varepsilon ) then  \  ( \text{mn}(x1) = \varepsilon )  \else  \  ( \text{mn}(x1) = &quot;err&quot; )</td>
</tr>
<tr>
<td>ID&lt;A&gt;</td>
<td>if ( \text{mn}(x0) = &quot;a&quot; ) then  \  ( \text{mn}(x1) = &quot;a&quot; )  \else  \  ( \text{mn}(x1) = &quot;err&quot; )</td>
</tr>
<tr>
<td>ID&lt;B&gt;</td>
<td>if ( \text{mn}(x0) = &quot;b&quot; ) then  \  ( \text{mn}(x1) = &quot;b&quot; )  \else  \  ( \text{mn}(x1) = &quot;err&quot; )</td>
</tr>
<tr>
<td>E&lt;1&gt;</td>
<td>if ( \text{mn}(x0) = 1 ) then  \  ( \text{mn}(x1) = 1 )  \else  \  ( \text{mn}(x1) = &quot;err&quot; )</td>
</tr>
<tr>
<td>E&lt;2&gt;</td>
<td>if ( \text{mn}(x1) = 2 ) then  \  ( \text{mn}(x1) = 2 )  \else  \  ( \text{mn}(x1) = &quot;err&quot; )</td>
</tr>
</tbody>
</table>
APPENDIX M

List of Theorems

<table>
<thead>
<tr>
<th>No.</th>
<th>Page</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>T.3.1</td>
<td>43</td>
<td>A set is recursive iff it is the set of sentences of a phrase-structure system.</td>
</tr>
<tr>
<td>T.3.2</td>
<td>44</td>
<td>A set is recursively enumerable iff it is the set of meanings of the sentences of a phrase-structure system.</td>
</tr>
<tr>
<td>T.3.3</td>
<td>45</td>
<td>If ( L ) is the language of phrase-structure system ( Q ), where the cardinality of ( V_n &gt; 2 ), then ( L ) is also the language of some ( Q' ) where the cardinality of ( V_n' = 2 ).</td>
</tr>
<tr>
<td>C.3.1</td>
<td>46</td>
<td>For each general phrase-structure system, there is an equivalent synthesized phrase-structure system.</td>
</tr>
<tr>
<td>T.3.4</td>
<td>50</td>
<td>For every table phrase-structure system, there is an equivalent general phrase-structure system without table attributes.</td>
</tr>
<tr>
<td>T.3.5</td>
<td>52</td>
<td>Let ( L ) be any language such that its sentences, ( \text{sen}(L) ), are a recursive set; each sentence has a finite number of meanings; and the set of meanings of its sentences, ( \text{mean}(L) ), is an r.e. set. If there is a total recursive function ( f ) such that ( f(w) = \text{mean}(w) ) for ( w \in \text{sen}(L) ) and ( f(w) = \text{err} ) for ( w ) not in ( \text{sen}(L) ), and which executes in time ( F(</td>
</tr>
<tr>
<td>T.3.6</td>
<td>53</td>
<td>It is undecidable whether the language of a phrase-structure system is semantically ambiguous.</td>
</tr>
</tbody>
</table>
It is undecidable whether the language of an arbitrary phrase-structure system is empty, finite, or infinite.

There are two semantic-preserving complete phrase-structure translators, \(M\) and \(M'\), such that the target language of \(M\) is the source language of \(M'\), but the composition of their induced translations is not semantic-preserving.

If \(M\) and \(M'\) are two semantic-preserving complete phrase-structure translators such that the target language of \(M\) is the source language of \(M'\), and all of the languages are semantically unambiguous, then the composition of their induced translations is semantic-preserving.

Let \(M\) be a complete phrase-structure transducer. If for each row \(t \ t' \ x\) of \(\text{TABL}\), rule\(\rightarrow\)(\(t\)) is a restriction of rule\(\rightarrow\)(\(t'\)), then \(M\) induces a translation.

Let \(M\) be a complete phrase-structure transducer. If for each row \(t \ t' \ x\) of \(\text{TABL}\), \(\text{rule}(t)\) is a restriction of \(\text{rule}(t')\), then \(M\) induces a translation.

Let \(M\) be a complete phrase-structure transducer. There is a weakly equivalent complete phrase-structure transducer \(M'\) such that all trees in \(\text{TABL}'\) have unit height.

For each complete phrase-structure transducer there is a constant \(k\) such that if \(t \in \text{domain}(\text{TFP})\), and \(t' \in \text{TFP}(t)\), then \(k^{*}\text{pwt}(t) \geq \text{pwt}(t')\).

If complete phrase-structure transducer \(M\) is multi-linear, then there are integer constants \(k\) and \(z\) such that if \(t \in \text{domain}(\text{TFP})\) and \(t' \in \text{TFP}(t)\), then \((k^{*}z)^{*}\text{pwt}(t) \geq \text{pwt}(t')\). Furthermore, if \(M\) is linear, then \(\text{pwt}(t) \geq \text{pwt}(t')\).

For each strong complete phrase-structure transducer \(M\), if \(t \in \text{domain}(\text{TFP})\), \(t' \in \text{TFP}(t)\), and \(t' \in \text{phrc}(\text{Pr}2)\), then \(\text{pwt}(t) \leq \text{pwt}(t')\).
Let $M$ be a multilinear complete phrase-structure transducer such that $TFS$ induced by $M$ is a 1-1 function. Then there are integer constants $x, m,$ and $n$ depending only on $TFS$ such that if $v \in TFS(w)$, $|w| > m$ and $|v| > x*m$, then (1) $w$ may be written as $\alpha \gamma \delta \varepsilon \zeta \iota$, where $|\alpha \gamma \delta \varepsilon \zeta \iota| < n$ and $\emptyset$ and $\varepsilon$ are not both the empty string; (2) $v$ may be written as $\tau \sigma \rho \iota$ for some integer $y \leq x$ where $|\tau \sigma \rho \iota| < n$ and $\emptyset$ and $\varepsilon$ are not both the empty string; and (3) for all $i > 0$,

$$(\alpha \gamma \delta \varepsilon \zeta \iota, \tau \sigma \rho \iota)$$

belongs to $TFS$.

The set of language pairs translatable using a syntax-directed transduction schema as defined by Aho and Ullman [69] is strictly less than the set of language pairs translatable using a complete phrase-structure transducer.

If $M$ is an honest complete phrase-structure translator, then $M$ is algorithmic.

Let $M$ be an incomplete phrase-structure transducer. If for each row of TABL, rule $t \rightarrow t'$ is a restriction of rule $\alpha \gamma \delta \varepsilon \zeta \iota \rightarrow \tau \sigma \rho \iota$, then $M$ induces a translation.

Let $M$ be an incomplete phrase-structure transducer. If for each row of TABL, rule $t \rightarrow t'$ is a restriction of rule $t \rightarrow t'$, then $M$ induces a translation.

As models of translation,

- $cpst < s$ extended
- $cpst < s$ ipst
- $cpst < w$ extended
- $cpst = w$ ipst

For every strong incomplete phrase-structure transducer, there is an inverse phrase-structure transducer such that $TFS(w)$ contains $v$ iff $TFS'(v)$ contains $w$.

Let $L1$ and $L2$ be two phrase-structure languages. There is an incomplete phrase-structure translator which induces the partial translation, $TRANS$, from $L1$ to $L2$, such that $TRANS(w) =$
\{v \mid \text{mean}(w) \neq 0, v \text{ is a sentence of L2}\}

T.6.1 110 Let \( M \) be a predicate phrase-structure transducer, where TABL has \( k \) rows. For \( i \in k \), suppose TABL(\( i \)) is
\[
\begin{align*}
t & \rightarrow t' 1 \quad \text{con}(1) \\
& \ldots \\
& \rightarrow \text{tn con}(n)
\end{align*}
\]
and if \( \text{con}(j) \) is satisfied, then the restriction of rule-\( \ast (t) \) to the domain in which \( \text{con}(j) \) is true is a restriction of rule-\( \ast (t'j) \). Then \( M \) induces a translation.

C.6.1 111 Let \( M \) be a predicate phrase-structure transducer, where TABL has \( k \) rows. For \( i \in k \), suppose TABL(\( i \)) is
\[
\begin{align*}
t & \rightarrow t' 1 \quad \text{con}(1) \\
& \ldots \\
& \rightarrow \text{tn con}(n)
\end{align*}
\]
and if \( \text{con}(j) \) is satisfied, then the restriction of rule(\( t \)) to the domain in which \( \text{con}(j) \) is true is a restriction of rule(\( t'j \)). Then \( M \) induces a translation.

T.6.2 116 As models of translation,
\[
\begin{align*}
\text{ipst} & \leq \text{ppst} \\
\text{ipst} & = \text{w ppst}
\end{align*}
\]

T.6.3 117 As models of translation,
\[
\begin{align*}
\text{cpst} & \leq \text{w pcpst}
\end{align*}
\]

T.6.4 118 The set of pairs of context-free languages which can be translated using an incomplete phrase-structure translator without discarding deviant parses is strictly smaller than the set of pairs of context-free languages which can be translated using an incomplete phrase-structure translator with a deviance check.

T.7.1 131 Let \( M \) be a block phrase-structure transducer. If for each row of TABL
\[
(t_1, \ldots, t_m) \quad (t'_1, \ldots, t'_n) \quad \text{rule-\( \ast ((t_1, \ldots, t_n)) \) is a restriction of rule-\( \ast ((t'_1, \ldots, t'_m)) \)}, \quad \text{then} \quad M \quad \text{induces a translation.}
\]
C.7.1 131 Suppose \( M \) is a block phrase-structure transducer. If for each row of \( \text{TABL} \)
\((t_1,\ldots,t_m)(t'_1,\ldots,t'_n)x \times x'\)
rule\((t_1,\ldots,t_m)\) is a restriction of
rule\((t'_1,\ldots,t'_n))\), then \( M \) induces a translation.

C.7.2 133 Let \( M \) be a block phrase-structure transducer. If for each row of \( \text{PE(TABL)} \) in some
penultimate extension of \( M \),
\((t_1,\ldots,t_m)(t'_1,\ldots,t'_n)\times x'\)
rule\(*\{(t_1,\ldots,t_m)\} \) is a restriction of
rule\(*\{(t'_1,\ldots,t'_n)\})\), then \( M \) induces a translation.

C.7.3 134 Let \( \text{PE}(M) \) be a penultimate extension of block
phrase-structure transducer \( M \). If for each
row of \( \text{PE(TABL)} \):
\((t_1,\ldots,t_m)(t'_1,\ldots,t'_n)\times x'\)
rule\((t_1,\ldots,t_m)\) is a restriction of
rule\((t'_1,\ldots,t'_n))\), then \( M \) induces a translation.

T.7.2 134 As models of translation,
\( \text{cpst} < w \) \( \text{bcpst} \)
\( \text{cpst} < s \) \( \text{bcpst} \)

T.7.3 135 Let \( M \) be a block phrase-structure transducer. There are integer constants \( y \) and \( z \) such that
if \( t \in \text{phra}(Q) \) and \( t' \in \text{TFP}(t) \), then
\( wt(t) = j \implies wt(t') \leq (yz)^j \)

T.8.1 142 For any augmented grammar \( G \), \( \text{phr}(G) \), \( \text{phrt}(G) \), and \( \text{phrc}(G) \) are all recursively enumerable.

T.8.2 143 For any augmented grammar \( G \), \( \text{sen}(G) \) and \( \text{L}(G) \)
are recursively enumerable sets.

T.8.3 144 The set of sentences of the associated
augmented grammar of a phrase-structure
system \( Q \) equal the set of sentences of \( Q \). The
language of the associated augmented
grammar of phrase-structure system \( Q \) equals
the language of \( Q \).

T.8.4 147 Let \( Q \) be an inverted phrase-structure system. Then \( \text{phra}(Q) \), \( \text{phrc}(Q) \), \( \text{sen}(Q) \) and \( \text{L}(Q) \) are
effectively enumerable sets.
T.8.5  149 Let $Q$ be a synthesized phrase-structure system, and let $Q'$ be its associated inverted phrase-structure system. Then $\text{phrc}(Q) = \text{phrc}(Q')$, $\text{sen}(Q) = \text{sen}(Q')$, and $L(Q) = L(Q')$.

T.8.6  154 TRANS is both sound and complete.

T.9.1  156 There are two synthesized phrase-structure systems $Q$ and $Q'$ such that $\text{TRANS}(Q, \text{aipss}(Q'), \ast)$ is not an algorithm.

T.9.2  157 If (i) $\text{MIN-MU}(S')$ is a recursive set in $V' = \text{mean}(Q)$, $\text{att-val}(v, mn : v \in V')$; or (ii) $\text{mean}(Q)$ is a subset of $\text{mean}(Q')$; then $\text{TRANS}(Q, Q', \ast)$ is an algorithm.

T.9.3  161 There are two synthesized phrase-structure systems $Q$ and $Q'$ such that there is no weak inverted phrase-structure system $Q''$ for which $W\text{-TRANS}(Q, Q'', \ast)$ is complete.

T.9.4  162 Let $Q$ and $Q'$ be two synthesized phrase-structure systems such that for each $v$ in $V'$, $\text{MIN-MU}(v)$ is a recursive set whose characteristic function is known. Then there is a weak inverted phrase-structure system $Q''$ of $Q'$, effectively constructable from $Q'$, such that $W\text{-TRANS}(Q, Q'', \ast)$ is both complete and algorithmic.

T.9.5  165 Let $Q$ and $Q''$ be two synthesized phrase-structure systems. Let $Q'$ be a weak inverted phrase-structure system of $Q''$. If $g(u\emptyset) = \text{"err"} \implies f\text{-inv}(u\emptyset) = 0$ for $g$, the tr-map function in the $i$-th production-rule pair of $R_u'$, and $f\text{-inv}$, the corresponding mn-map relation of the $i$-th production-rule pair of $Q'$, and if for each $t \in \text{phrc}(Q'')$, $n, m$ nodes(t), it is true that $n$ is the parent of $m \implies \forall v(\text{tr}(n)) \triangleright \forall v(\text{tr}(m))$, then $W\text{-TRANS}(Q, Q', \ast)$ is complete and algorithmic.

T.9.6  168 Let $Q$ and $Q''$ be two synthesized phrase-structure systems. Let $Q'$ be a weak inverted phrase-structure system of $Q''$. Suppose $W\text{-TRANS}(Q, Q', \ast)$ satisfies the conditions of T.9.5. Then $W\text{-TRANS}(Q, Q', \ast)$ is both complete and algorithmic.
Let $Q$ and $Q''$ be any two synthesized phrase-structure systems. Let $Q'$ be a weak inverted phrase-structure system of $Q''$. If for each $w$ in $\text{sen}(Q)$, where $n = |w|$, (i) $\text{wc-TRANS}(Q,Q',w)$ halts; (ii) $\text{wc-output}(w) = (v_1, \ldots, v_z) \implies$ there is a polynomial function $f$ such that for $i \in \mathbb{Z}$, $f(n) \geq |v_i|$: (iii) if $R_U$ contains $c_0 < c_1 \ldots c_k$ then (a) $g \in \text{PTIME}$, (b) $g(u) = (u_1, \ldots, u_k) \implies |u_i| \leq |u_j|$; (iv) $\text{MEMBER}$ is polynomial bound; (v) $|\text{parse}(w)| \leq g(n)$ for some polynomial function $g$; (vi) $|u| \leq r(n)$ for some polynomial function $r$, $u$ in $\text{mean}(w)$; and (vii) $\text{MEANING}$ is polynomial bound for $t$ in $\text{phrc}(Q)$; then $\text{wc-TRANS}(Q,Q',\ast)$ is polynomial bound.
<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leq d$</td>
<td>33</td>
</tr>
<tr>
<td>$\leq r$</td>
<td>63</td>
</tr>
<tr>
<td>$\leq s$</td>
<td>88</td>
</tr>
<tr>
<td>$\leq w$</td>
<td>88</td>
</tr>
<tr>
<td>$D$</td>
<td>14</td>
</tr>
<tr>
<td>&quot;err&quot;</td>
<td>38</td>
</tr>
<tr>
<td>$\leq s$</td>
<td>88</td>
</tr>
<tr>
<td>$\leq w$</td>
<td>88</td>
</tr>
<tr>
<td>$\Rightarrow$</td>
<td>14</td>
</tr>
<tr>
<td>$= s$</td>
<td>88</td>
</tr>
<tr>
<td>$= w$</td>
<td>88</td>
</tr>
<tr>
<td>$\aleph$</td>
<td>14</td>
</tr>
<tr>
<td>A.4.1</td>
<td>57, 58</td>
</tr>
<tr>
<td>$\text{aag}(Q)$</td>
<td>144</td>
</tr>
<tr>
<td>abbreviations</td>
<td>14</td>
</tr>
<tr>
<td>AG</td>
<td>140, 141, 142</td>
</tr>
<tr>
<td>ag</td>
<td>139</td>
</tr>
<tr>
<td>Aho, A.</td>
<td>6, 27, 56, 59, 61</td>
</tr>
<tr>
<td>ALGOL60</td>
<td>5</td>
</tr>
<tr>
<td>algorithm</td>
<td>17</td>
</tr>
<tr>
<td>algorithmic</td>
<td>17</td>
</tr>
<tr>
<td>alphabet</td>
<td>18</td>
</tr>
<tr>
<td>ambiguity</td>
<td>40, 53</td>
</tr>
<tr>
<td>ancestor</td>
<td>20</td>
</tr>
<tr>
<td>APL</td>
<td>4</td>
</tr>
<tr>
<td>Appendix A</td>
<td>60</td>
</tr>
<tr>
<td>Appendix B</td>
<td>77</td>
</tr>
<tr>
<td>Appendix C</td>
<td>108, 109</td>
</tr>
<tr>
<td>Appendix D</td>
<td>115</td>
</tr>
<tr>
<td>Appendix E</td>
<td>117</td>
</tr>
<tr>
<td>Appendix G</td>
<td>125</td>
</tr>
<tr>
<td>Appendix H</td>
<td>117</td>
</tr>
<tr>
<td>Appendix I</td>
<td>134</td>
</tr>
<tr>
<td>Appendix J</td>
<td>145</td>
</tr>
<tr>
<td>Appendix K</td>
<td>147, 162</td>
</tr>
<tr>
<td>Appendix L</td>
<td>159</td>
</tr>
<tr>
<td>ASSOC</td>
<td>56</td>
</tr>
<tr>
<td>associate augmented grammar</td>
<td>144</td>
</tr>
<tr>
<td>association vector</td>
<td>56</td>
</tr>
<tr>
<td>attribute value</td>
<td>36</td>
</tr>
<tr>
<td>augmented grammar</td>
<td>139</td>
</tr>
<tr>
<td>Ax</td>
<td>23</td>
</tr>
<tr>
<td>axiom</td>
<td>23</td>
</tr>
<tr>
<td>Term</td>
<td>Page Numbers</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>axiomatic phrase-structures</td>
<td>141</td>
</tr>
<tr>
<td>Backus, J.</td>
<td>5</td>
</tr>
<tr>
<td>Backus-Naur Form</td>
<td>5</td>
</tr>
<tr>
<td>Bar-Hillel, Y.</td>
<td>2, 27</td>
</tr>
<tr>
<td>Benson, D.</td>
<td>8, 61</td>
</tr>
<tr>
<td>binary numbers</td>
<td>74, 140</td>
</tr>
<tr>
<td>Blum, M.</td>
<td>27</td>
</tr>
<tr>
<td>BNF</td>
<td>5</td>
</tr>
<tr>
<td>Booth, A.</td>
<td>1</td>
</tr>
<tr>
<td>Boss</td>
<td>14</td>
</tr>
<tr>
<td>box bracketing</td>
<td>22</td>
</tr>
<tr>
<td>bpost</td>
<td>120</td>
</tr>
<tr>
<td>Brooker, R.</td>
<td>5</td>
</tr>
<tr>
<td>Buttelmann, H.</td>
<td>8, 10, 60, 61, 83, 94</td>
</tr>
<tr>
<td>cardinality</td>
<td>15</td>
</tr>
<tr>
<td>category theory</td>
<td>9</td>
</tr>
<tr>
<td>catenation</td>
<td>14</td>
</tr>
<tr>
<td>catenation closure</td>
<td>14</td>
</tr>
<tr>
<td>cfg</td>
<td>23</td>
</tr>
<tr>
<td>Cheatham, T.</td>
<td>5</td>
</tr>
<tr>
<td>child</td>
<td>20</td>
</tr>
<tr>
<td>Chomsky, N.</td>
<td>1, 38</td>
</tr>
<tr>
<td>Church, A.</td>
<td>8</td>
</tr>
<tr>
<td>circular pss</td>
<td>34</td>
</tr>
<tr>
<td>COBOL</td>
<td>4</td>
</tr>
<tr>
<td>compiler-compiler</td>
<td>5</td>
</tr>
<tr>
<td>complete phrase-structures</td>
<td>141</td>
</tr>
<tr>
<td>complete pss</td>
<td>38</td>
</tr>
<tr>
<td>complete pst</td>
<td>55</td>
</tr>
<tr>
<td>complexity</td>
<td>27, 50</td>
</tr>
<tr>
<td>compose</td>
<td>21</td>
</tr>
<tr>
<td>composition</td>
<td>21</td>
</tr>
<tr>
<td>concatenation</td>
<td>15</td>
</tr>
<tr>
<td>connected</td>
<td>135</td>
</tr>
<tr>
<td>context-free grammar</td>
<td>23</td>
</tr>
<tr>
<td>context-free language</td>
<td>60</td>
</tr>
<tr>
<td>context-sensitive</td>
<td>38</td>
</tr>
<tr>
<td>conventions</td>
<td>20</td>
</tr>
<tr>
<td>Cool Hand Luke</td>
<td>1</td>
</tr>
<tr>
<td>correspondence</td>
<td>56</td>
</tr>
<tr>
<td>cpst</td>
<td>55</td>
</tr>
<tr>
<td>Daley, Richard J.</td>
<td>14</td>
</tr>
<tr>
<td>decimal numbers</td>
<td>74</td>
</tr>
<tr>
<td>dependencies</td>
<td>33</td>
</tr>
<tr>
<td>DER-T</td>
<td>25</td>
</tr>
<tr>
<td>derivation</td>
<td>24</td>
</tr>
<tr>
<td>derivation tree</td>
<td>24</td>
</tr>
<tr>
<td>descendent</td>
<td>20</td>
</tr>
<tr>
<td>deviance</td>
<td>38, 83</td>
</tr>
<tr>
<td>deviant evaluation</td>
<td>38</td>
</tr>
<tr>
<td>Term</td>
<td>Page(s)</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>DO DOVETAIL</td>
<td>151</td>
</tr>
<tr>
<td>Doner, J.</td>
<td>7</td>
</tr>
<tr>
<td>dovetail</td>
<td>147, 151</td>
</tr>
<tr>
<td>e</td>
<td>23</td>
</tr>
<tr>
<td>Earley’s algorithm</td>
<td>43</td>
</tr>
<tr>
<td>Earley, J.</td>
<td>43</td>
</tr>
<tr>
<td>Edmundson, H.</td>
<td>1</td>
</tr>
<tr>
<td>empty string</td>
<td>14, 23</td>
</tr>
<tr>
<td>entrance attribute</td>
<td>34</td>
</tr>
<tr>
<td>equivalent phrase-structure system</td>
<td>44</td>
</tr>
<tr>
<td>equivalent phrase-structure transducer</td>
<td>64</td>
</tr>
<tr>
<td>evaluation</td>
<td>37</td>
</tr>
<tr>
<td>execute</td>
<td>27</td>
</tr>
<tr>
<td>execution time</td>
<td>27</td>
</tr>
<tr>
<td>exit attribute</td>
<td>34</td>
</tr>
<tr>
<td>exponential pst</td>
<td>67</td>
</tr>
<tr>
<td>exponential transducer</td>
<td>66</td>
</tr>
<tr>
<td>exponentiation</td>
<td>14</td>
</tr>
<tr>
<td>f-inv</td>
<td>17</td>
</tr>
<tr>
<td>Feldman, J.</td>
<td>6</td>
</tr>
<tr>
<td>finitely specified transducer</td>
<td>94</td>
</tr>
<tr>
<td>Fodor, J.</td>
<td>29</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>4</td>
</tr>
<tr>
<td>fr(t)</td>
<td>19</td>
</tr>
<tr>
<td>frontier</td>
<td>19</td>
</tr>
<tr>
<td>frontier(t,i)</td>
<td>20</td>
</tr>
<tr>
<td>gen(Tr)</td>
<td>23</td>
</tr>
<tr>
<td>gen-pss</td>
<td>45</td>
</tr>
<tr>
<td>general pss</td>
<td>45</td>
</tr>
<tr>
<td>generating set</td>
<td>23</td>
</tr>
<tr>
<td>generation of Tr</td>
<td>23</td>
</tr>
<tr>
<td>Georgetown University</td>
<td>55</td>
</tr>
<tr>
<td>Gries, D.</td>
<td>6</td>
</tr>
<tr>
<td>Hartmanis, J.</td>
<td>27</td>
</tr>
<tr>
<td>height</td>
<td>19</td>
</tr>
<tr>
<td>Hopcroft, J.</td>
<td>27</td>
</tr>
<tr>
<td>ht(t)</td>
<td>19</td>
</tr>
<tr>
<td>implication</td>
<td>14</td>
</tr>
<tr>
<td>improper pss</td>
<td>34</td>
</tr>
<tr>
<td>incomplete pss</td>
<td>38</td>
</tr>
<tr>
<td>index vector</td>
<td>56</td>
</tr>
<tr>
<td>INT0</td>
<td>14</td>
</tr>
<tr>
<td>INT1</td>
<td>14</td>
</tr>
<tr>
<td>inval-t</td>
<td>145</td>
</tr>
<tr>
<td>inverse pst</td>
<td>89</td>
</tr>
<tr>
<td>inverse relation</td>
<td>17</td>
</tr>
<tr>
<td>inverse semantic rule</td>
<td>145</td>
</tr>
<tr>
<td>inverted phrase-structure system</td>
<td>145</td>
</tr>
<tr>
<td>Term</td>
<td>Page(s)</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>inverted pss</td>
<td>145</td>
</tr>
<tr>
<td>ipss</td>
<td>145</td>
</tr>
<tr>
<td>ipst</td>
<td>82</td>
</tr>
<tr>
<td>Irons, E.</td>
<td>5</td>
</tr>
<tr>
<td>isomorphic derivations</td>
<td>25</td>
</tr>
<tr>
<td>IVAL-T</td>
<td>146</td>
</tr>
<tr>
<td>Katz, J.</td>
<td>29, 29, 45, 46, 51</td>
</tr>
<tr>
<td>Knuth, D.</td>
<td>8, 29, 45, 46, 51</td>
</tr>
<tr>
<td>L(G)</td>
<td>142</td>
</tr>
<tr>
<td>L(Q)</td>
<td>39</td>
</tr>
<tr>
<td>labeled ordered tree</td>
<td>18</td>
</tr>
<tr>
<td>language</td>
<td>39, 142</td>
</tr>
<tr>
<td>Lapin, R.</td>
<td>2</td>
</tr>
<tr>
<td>Lauer, P.</td>
<td>8</td>
</tr>
<tr>
<td>Ledgard, H.</td>
<td>2</td>
</tr>
<tr>
<td>leftmost derivation</td>
<td>24</td>
</tr>
<tr>
<td>leftmost partial derivation</td>
<td>26</td>
</tr>
<tr>
<td>Leggett, E.</td>
<td>117</td>
</tr>
<tr>
<td>length</td>
<td>15</td>
</tr>
<tr>
<td>Lewis, P.</td>
<td>6, 50</td>
</tr>
<tr>
<td>linear recursive</td>
<td>42</td>
</tr>
<tr>
<td>linear transducer</td>
<td>66, 69</td>
</tr>
<tr>
<td>LISP</td>
<td>4</td>
</tr>
<tr>
<td>LMD</td>
<td>57</td>
</tr>
<tr>
<td>lmd</td>
<td>24</td>
</tr>
<tr>
<td>lmpd</td>
<td>26</td>
</tr>
<tr>
<td>lrss</td>
<td>42</td>
</tr>
<tr>
<td>Lucas, P.</td>
<td>8</td>
</tr>
<tr>
<td>maximal leftmost partial derivation</td>
<td>26</td>
</tr>
<tr>
<td>McCarthy, J.</td>
<td>8</td>
</tr>
<tr>
<td>mean(t)</td>
<td>39</td>
</tr>
<tr>
<td>mean(t, val-t)</td>
<td>39</td>
</tr>
<tr>
<td>mean(w)</td>
<td>39</td>
</tr>
<tr>
<td>meaning</td>
<td>39</td>
</tr>
<tr>
<td>mechanical translation</td>
<td>1</td>
</tr>
<tr>
<td>mlmpd</td>
<td>26</td>
</tr>
<tr>
<td>multi-linear transducer</td>
<td>66, 69</td>
</tr>
<tr>
<td>NAL</td>
<td>18</td>
</tr>
<tr>
<td>natural language translation</td>
<td>18</td>
</tr>
<tr>
<td>Naur, P.</td>
<td>5</td>
</tr>
<tr>
<td>NDIVAL-T</td>
<td>146</td>
</tr>
<tr>
<td>NDVAL-T</td>
<td>38</td>
</tr>
<tr>
<td>Newman, P.</td>
<td>1</td>
</tr>
<tr>
<td>nrss-parse</td>
<td>43</td>
</tr>
<tr>
<td>nnos</td>
<td>139</td>
</tr>
<tr>
<td>nodes</td>
<td>18</td>
</tr>
<tr>
<td>NODES</td>
<td>18</td>
</tr>
<tr>
<td>non-deviant evaluation</td>
<td>38</td>
</tr>
<tr>
<td>non-deviant VAL-T</td>
<td>38</td>
</tr>
<tr>
<td>Term</td>
<td>Page(s)</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>nonterminal</td>
<td>23</td>
</tr>
<tr>
<td>nonterminal node set</td>
<td>139</td>
</tr>
<tr>
<td>normal form</td>
<td>65, 83, 111, 125, 126, 128</td>
</tr>
<tr>
<td>Oettinger, A.</td>
<td>1</td>
</tr>
<tr>
<td>operate</td>
<td>27</td>
</tr>
<tr>
<td>ordered set</td>
<td>21</td>
</tr>
<tr>
<td>ordered tree</td>
<td>18</td>
</tr>
<tr>
<td>ordering relation</td>
<td>21</td>
</tr>
<tr>
<td>PAE</td>
<td>32, 38, 39, 40</td>
</tr>
<tr>
<td>parent</td>
<td>20</td>
</tr>
<tr>
<td>parse</td>
<td>24</td>
</tr>
<tr>
<td>parse(w)</td>
<td>24</td>
</tr>
<tr>
<td>partition</td>
<td>110</td>
</tr>
<tr>
<td>path</td>
<td>135</td>
</tr>
<tr>
<td>pcpsst</td>
<td>117</td>
</tr>
<tr>
<td>Perles, M.</td>
<td>27</td>
</tr>
<tr>
<td>phr(G)</td>
<td>24, 141</td>
</tr>
<tr>
<td>phr(Pr)</td>
<td>23, 141</td>
</tr>
<tr>
<td>phr(Q)</td>
<td>38</td>
</tr>
<tr>
<td>phra(G)</td>
<td>24, 141</td>
</tr>
<tr>
<td>phra(Pr)</td>
<td>23, 141</td>
</tr>
<tr>
<td>phra(Q)</td>
<td>38</td>
</tr>
<tr>
<td>phrase-structure system</td>
<td>30</td>
</tr>
<tr>
<td>phrase-structures</td>
<td>141</td>
</tr>
<tr>
<td>phrase-transform</td>
<td>57, 82, 108, 122</td>
</tr>
<tr>
<td>phrc(G)</td>
<td>24, 141</td>
</tr>
<tr>
<td>phrc(Pr)</td>
<td>23, 141</td>
</tr>
<tr>
<td>phrc(Q)</td>
<td>38</td>
</tr>
<tr>
<td>phrt(G)</td>
<td>24, 141</td>
</tr>
<tr>
<td>phrt(Pr)</td>
<td>23, 141</td>
</tr>
<tr>
<td>phrt(Q)</td>
<td>38</td>
</tr>
<tr>
<td>possible values</td>
<td>31</td>
</tr>
<tr>
<td>postfix arithmetic</td>
<td>32</td>
</tr>
<tr>
<td>powerset(A)</td>
<td>14</td>
</tr>
<tr>
<td>ppst</td>
<td>107</td>
</tr>
<tr>
<td>ppst equivalent to ipst</td>
<td>111</td>
</tr>
<tr>
<td>Pr</td>
<td>23</td>
</tr>
<tr>
<td>procedural</td>
<td>17</td>
</tr>
<tr>
<td>procedure</td>
<td>17</td>
</tr>
<tr>
<td>production</td>
<td>23</td>
</tr>
<tr>
<td>production-rule pair</td>
<td>145</td>
</tr>
<tr>
<td>programming language translation</td>
<td>1</td>
</tr>
<tr>
<td>proper pss</td>
<td>34</td>
</tr>
<tr>
<td>pseudo-PL/I</td>
<td>27</td>
</tr>
<tr>
<td>pss</td>
<td>30</td>
</tr>
<tr>
<td>pst-1</td>
<td>59, 60</td>
</tr>
<tr>
<td>pst-10</td>
<td>134</td>
</tr>
<tr>
<td>pst-2</td>
<td>77</td>
</tr>
<tr>
<td>pst-3</td>
<td>108, 109</td>
</tr>
<tr>
<td>pst-4</td>
<td>115</td>
</tr>
<tr>
<td>pst-5</td>
<td>117</td>
</tr>
<tr>
<td>Term</td>
<td>Page(s)</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>pst-8</td>
<td>125</td>
</tr>
<tr>
<td>PTIME</td>
<td>28</td>
</tr>
<tr>
<td>pumping theorem</td>
<td>71</td>
</tr>
<tr>
<td>reduced grammar</td>
<td>27</td>
</tr>
<tr>
<td>relation</td>
<td>15</td>
</tr>
<tr>
<td>restriction</td>
<td>63, 125, 127</td>
</tr>
<tr>
<td>root</td>
<td>19</td>
</tr>
<tr>
<td>Rosenkrantz, R.</td>
<td>50</td>
</tr>
<tr>
<td>rt(t)</td>
<td>19</td>
</tr>
<tr>
<td>rule-*</td>
<td>35</td>
</tr>
<tr>
<td>Sattley, K.</td>
<td>5</td>
</tr>
<tr>
<td>sdt-1</td>
<td>145</td>
</tr>
<tr>
<td>sdt-2</td>
<td>159</td>
</tr>
<tr>
<td>SEM</td>
<td>30</td>
</tr>
<tr>
<td>sem-mem</td>
<td>140</td>
</tr>
<tr>
<td>semantic ambiguity</td>
<td>41</td>
</tr>
<tr>
<td>semantic deviance</td>
<td>38</td>
</tr>
<tr>
<td>semantic-directed</td>
<td>3</td>
</tr>
<tr>
<td>semantic-preservation</td>
<td>61</td>
</tr>
<tr>
<td>semantic-preserving</td>
<td>61</td>
</tr>
<tr>
<td>sen(G)</td>
<td>24, 142</td>
</tr>
<tr>
<td>sen(Q)</td>
<td>39</td>
</tr>
<tr>
<td>sentence</td>
<td>39</td>
</tr>
<tr>
<td>sentences</td>
<td>24, 142</td>
</tr>
<tr>
<td>Shamir, E.</td>
<td>27</td>
</tr>
<tr>
<td>simple ipst</td>
<td>89</td>
</tr>
<tr>
<td>SNOBOL</td>
<td>46</td>
</tr>
<tr>
<td>start symbol</td>
<td>23</td>
</tr>
<tr>
<td>Stearns, R.</td>
<td>6, 50</td>
</tr>
<tr>
<td>string transform</td>
<td>110</td>
</tr>
<tr>
<td>string-transform</td>
<td>60, 82</td>
</tr>
<tr>
<td>strong transducer</td>
<td>66, 70</td>
</tr>
<tr>
<td>strongly equivalent</td>
<td>64</td>
</tr>
<tr>
<td>sub-structure of t</td>
<td>25</td>
</tr>
<tr>
<td>supertree</td>
<td>22</td>
</tr>
<tr>
<td>syn-fr(t)</td>
<td>141</td>
</tr>
<tr>
<td>syn-mem</td>
<td>140</td>
</tr>
<tr>
<td>syn-pss</td>
<td>45</td>
</tr>
<tr>
<td>syntactic ambiguity</td>
<td>40</td>
</tr>
<tr>
<td>syntactic-frontier</td>
<td>141</td>
</tr>
<tr>
<td>syntax-directed</td>
<td>3</td>
</tr>
<tr>
<td>syntax-directed transducer</td>
<td>61</td>
</tr>
<tr>
<td>synthesized phrase-structure system</td>
<td>45</td>
</tr>
<tr>
<td>T+</td>
<td>14</td>
</tr>
<tr>
<td>tab-att</td>
<td>46</td>
</tr>
<tr>
<td>TAB-ATT</td>
<td>46</td>
</tr>
<tr>
<td>tab-pss</td>
<td>46</td>
</tr>
<tr>
<td>TABL</td>
<td>56</td>
</tr>
<tr>
<td>table attribute</td>
<td>46</td>
</tr>
<tr>
<td>table pss</td>
<td>46</td>
</tr>
<tr>
<td>Term</td>
<td>Page(s)</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>terminal</td>
<td>23</td>
</tr>
<tr>
<td>terminal node set</td>
<td>140</td>
</tr>
<tr>
<td>terminated phrase-structures</td>
<td>141</td>
</tr>
<tr>
<td>TFP</td>
<td>57, 82, 108, 122</td>
</tr>
<tr>
<td>TFS</td>
<td>60, 82</td>
</tr>
<tr>
<td>Thatcher, J.</td>
<td>7</td>
</tr>
<tr>
<td>time bound</td>
<td>27</td>
</tr>
<tr>
<td>time complexity</td>
<td>27</td>
</tr>
<tr>
<td>tns</td>
<td>140</td>
</tr>
<tr>
<td>topological sort</td>
<td>51</td>
</tr>
<tr>
<td>transduce</td>
<td>60</td>
</tr>
<tr>
<td>transduction</td>
<td>61</td>
</tr>
<tr>
<td>translation</td>
<td>61</td>
</tr>
<tr>
<td>Translation Problem</td>
<td>1</td>
</tr>
<tr>
<td>translator writing system</td>
<td>5</td>
</tr>
<tr>
<td>tree automata</td>
<td>7</td>
</tr>
<tr>
<td>tree transducers</td>
<td>7</td>
</tr>
<tr>
<td>trees</td>
<td>18</td>
</tr>
<tr>
<td>Turing translator</td>
<td>94</td>
</tr>
<tr>
<td>TWS</td>
<td>5</td>
</tr>
<tr>
<td>UG(Q)</td>
<td>30</td>
</tr>
<tr>
<td>Ullman, J.</td>
<td>6, 27, 56, 59, 61</td>
</tr>
<tr>
<td>underlying grammar</td>
<td>30</td>
</tr>
<tr>
<td>upper case letters</td>
<td>20</td>
</tr>
<tr>
<td>useless production</td>
<td>26</td>
</tr>
<tr>
<td>uvwxxy theorem</td>
<td>71</td>
</tr>
<tr>
<td>V</td>
<td>23</td>
</tr>
<tr>
<td>v(attu(c))</td>
<td>36</td>
</tr>
<tr>
<td>V**k</td>
<td>14</td>
</tr>
<tr>
<td>val-t</td>
<td>146</td>
</tr>
<tr>
<td>VAL-T</td>
<td>37</td>
</tr>
<tr>
<td>val-t</td>
<td>37</td>
</tr>
<tr>
<td>valid transform</td>
<td>110</td>
</tr>
<tr>
<td>VDL</td>
<td>8</td>
</tr>
<tr>
<td>Vienna Definition Language</td>
<td>8</td>
</tr>
<tr>
<td>Vn</td>
<td>23</td>
</tr>
<tr>
<td>vns</td>
<td>140</td>
</tr>
<tr>
<td>vocabulary</td>
<td>23, 139</td>
</tr>
<tr>
<td>vocabulary node set</td>
<td>140</td>
</tr>
<tr>
<td>Vt</td>
<td>23</td>
</tr>
<tr>
<td>Walk, K.</td>
<td>8</td>
</tr>
<tr>
<td>weakly equivalent</td>
<td>64</td>
</tr>
<tr>
<td>Weaver, W.</td>
<td>1</td>
</tr>
<tr>
<td>weight</td>
<td>19</td>
</tr>
<tr>
<td>Wilks, Y.</td>
<td>2</td>
</tr>
<tr>
<td>Wright, J.</td>
<td>7</td>
</tr>
<tr>
<td>wt(t)</td>
<td>19</td>
</tr>
<tr>
<td>x</td>
<td></td>
</tr>
<tr>
<td>x***y</td>
<td>14</td>
</tr>
<tr>
<td>Index</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>x-vector</td>
<td>56</td>
</tr>
<tr>
<td>XPL</td>
<td>6</td>
</tr>
<tr>
<td>lwl</td>
<td>15</td>
</tr>
</tbody>
</table>