INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

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

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

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

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

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.
Perry, Doyt Lee

COMPUTABILITY AND COMPLEXITY ISSUES OF TRANSLATOR GENERATION

The Ohio State University

University Microfilms International 300 N. Zeeb Road, Ann Arbor, MI 48106

Copyright 1982 by Perry, Doyt Lee
All Rights Reserved
COMPUTABILITY AND COMPLEXITY ISSUES OF TRANSLATOR GENERATION

DISSERTATION

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

By
Doyt Lee Perry, B.S, M.Sc.

* * * * *

The Ohio State University
1982

Reading Committee: Approved By

Dr. H.W. Buttelmann

Dr. L.J. White

Dr. T.J. Long

Advisor
Department of
Computer and Information Science
To my Grandmothers:

Bertha Zeroll,
from whom I learned patience,

and Grace Perry,

who taught me perseverance
I am indebted to Bill Buttelmann who, as my advisor and friend, provided inspiration, guidance, and support for this work. Lee White gave important suggestions and much encouragement (and kept me sane with hours of friendly competition on the racquetball court). Without the assistance of Tim Long, I doubt that this work would have been completed. Not only did he provide valuable technical guidance, but his personal friendship and support helped me cope with the many moments of discouragement that accompanied my work. Art Pyster, and Ramachandran Krishnaswamy, who preceded me in this research, have been most helpful through the years.

In working so long on my doctorate, I have been sustained by the relationships and help of many friends. My thanks to Wayne and Pam Carlson, Al and Betty Baker, Dave and Julie Rypka, Ed Cohen, Dee Marik, Paul Amer, Pat DeRuyter, Debi Vancil, John and Mary Beth Lohse, Zeke and Lynne Haley, Bruce Flinchbaugh, Charlie Shubra, Mary Lieb, Ahmed El-Magarmid, and Fawzi Gherfal. Tamer Ozsu was generous in guiding me through the intricacies of text editing and formatting. Doreen Close gave cheerful and patient support to me in the difficult final months of writing and typing.

And finally, my thanks to my parents, Doyt and Loretta Perry, who have encouraged me for all these years without asking why I was still in school.
VITA

January 15, 1951
Born - Columbus, Ohio

1969-1973
B.S.,
Bowling Green State University,
Bowling Green, Ohio

1974-1976
Graduate Teaching Associate,
Department of Computer and
Information Science,
The Ohio State University,
Columbus, Ohio

1975
M.Sc.,
The Ohio State University,
Columbus, Ohio

1976-Present
Graduate Associate,
Department of Computer and
Information Science,
The Ohio State University,
Columbus, Ohio

FIELDS OF STUDY

Major Field: Computer and Information Science

Studies in Formal Languages and Automata Theory
Professor H.W. Buttelmann

Studies in Theory of Computability and Complexity
Professor T.J. Long
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEDICATION</td>
<td>ii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>VITA</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1. Translation</td>
<td>1</td>
</tr>
<tr>
<td>1.2. Framework for Studying Translation</td>
<td>2</td>
</tr>
<tr>
<td>1.3. Historical Perspective</td>
<td>5</td>
</tr>
<tr>
<td>1.4. Translator Generation</td>
<td>13</td>
</tr>
<tr>
<td>1.5. Computability and Complexity of Translator Generation</td>
<td>13</td>
</tr>
<tr>
<td>1.6. Key Issues Addressed</td>
<td>17</td>
</tr>
<tr>
<td>2. Basic Terminology and Definitions</td>
<td>22</td>
</tr>
<tr>
<td>2.1. Proof Notation</td>
<td>22</td>
</tr>
<tr>
<td>2.2. Strings</td>
<td>22</td>
</tr>
<tr>
<td>2.3. Sets</td>
<td>23</td>
</tr>
<tr>
<td>2.4. Functions, Predicates, and Computations</td>
<td>24</td>
</tr>
<tr>
<td>2.5. Computability and Complexity</td>
<td>28</td>
</tr>
<tr>
<td>2.6. Trees</td>
<td>33</td>
</tr>
<tr>
<td>2.7. Context Free Grammars</td>
<td>38</td>
</tr>
<tr>
<td>3. Languages and Language Definitions</td>
<td>41</td>
</tr>
<tr>
<td>3.1. Languages</td>
<td>41</td>
</tr>
<tr>
<td>3.2. Language Definitions</td>
<td>43</td>
</tr>
<tr>
<td>3.3. Language Definition System</td>
<td>50</td>
</tr>
<tr>
<td>3.4. Syntax-Semantics Tradeoffs in Language Definition</td>
<td>54</td>
</tr>
<tr>
<td>3.5. The Languages Described by Language Definition</td>
<td>58</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The Translation Problem</td>
<td>2</td>
</tr>
<tr>
<td>2. The translation problem with multiple translations</td>
<td>3</td>
</tr>
<tr>
<td>3. Using formal specification of source language syntax</td>
<td>6</td>
</tr>
<tr>
<td>4. Using formal specification of source syntax and semantics</td>
<td>7</td>
</tr>
<tr>
<td>5. Using formal specification of source syntax and semantics</td>
<td>9</td>
</tr>
<tr>
<td>6. Using formal syntax and semantics of both languages</td>
<td>10</td>
</tr>
<tr>
<td>7. Semantic-directed translation</td>
<td>12</td>
</tr>
<tr>
<td>8. Translation with semantic filtering</td>
<td>14</td>
</tr>
<tr>
<td>9. The structure of a compiler-compiler</td>
<td>15</td>
</tr>
<tr>
<td>10. Translator generation producing plug-in semantics</td>
<td>18</td>
</tr>
<tr>
<td>11. The Arithmetic Hierarchy</td>
<td>32</td>
</tr>
<tr>
<td>12. Branching diagram for trees</td>
<td>34</td>
</tr>
<tr>
<td>13. Composition of trees</td>
<td>35</td>
</tr>
<tr>
<td>14. Box-bracketings of trees</td>
<td>37</td>
</tr>
<tr>
<td>15. A language definition D in LDS[(V, A, f)]</td>
<td>52</td>
</tr>
<tr>
<td>16. Plugging in programs for functions in an LDS</td>
<td>54</td>
</tr>
<tr>
<td>17. The structure of TAUBAR</td>
<td>78</td>
</tr>
<tr>
<td>18. Trees for Example 4.2.3</td>
<td>84</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

1.1. Translation

The notion of translation is one familiar to users of more than one language. Simply stated, translation is the conversion of sentences in one language to sentences of the same meaning in another language. Translation is a directed operation in that one of the languages is designated as a source language from which we translate; a second language serves as a target into which we translate. We do not expect that the appearance of the source sentence and its translation in the target language to necessarily be similar; however, we do expect them to be closely related in meaning. Since a source or target sentence may have several meanings, it is reasonable to ask only that a source sentence and its translation share one meaning and not necessarily be identical in all meanings. This leads to a more precise statement of the translation problem studied in this thesis:

Given a source and target language, specify a translation relation between the sentences of the two languages such that if the relation holds between a source sentence and a target sentence, then these two sentences share a meaning.

There is a wide range of interests in the problem of translation. Language users and linguists are concerned with natural language translation; for example, translating German to English. For three decades computer scientists have also been extensively engaged in translation research. On the one hand, they have investigated the application of computers in mechanizing natural language translation. On the other hand, computer scientists have created many artificial languages to facilitate human interaction with computer systems. Programming languages, command languages, and a variety of so-called "human interfaces" must all be converted into the digital codes that can be processed by computers. Indeed, the study of programming language compilers and interpreters is an important area of computer science research. All this activity has spawned other research that has attempted to abstract and formalize several aspects of translation. Such formalization has permitted careful analysis of certain translation
issues and development of an underlying theory of translation.

This thesis extends previous work done in this formal vein. The emphasis is on determining the limitations of formal methods for discovering translations. It examines why, in some circumstances, general methods for finding translations do not exist. In other instances, where a solution is possible, this research examines the amount of work to be done to find translations. Its intended contribution is to develop and organize a body of results that address the computability and complexity issues of finding translations between languages.

Figure 1: The Translation Problem

1.2. Framework for studying translation

For our work, a solution to the translation problem involves specification of a translation mechanism that, when given a source sentence, will produce a target sentence that shares a meaning with the source sentence (see Figure 1). Because of ambiguities of the source sentence and synonymous sentences in the target language, there may be multiple translations (see Figure 2).
This simple "black box" view of the translation problem is appropriate for discussing the mechanization of natural and programming language translation, especially if there is a generous use of the concept of "sentence". For example, due to contextual relationships, an entire program may be required by a programming language translator to appropriately compile individual program statements; consequently, a "sentence" for such a translator might be the entire program. The framework suggested is also a suitable starting point for sketching the development of translation theory. The maturing of that theory is reflected in a fleshing out of the skeletal framework of Figure 1.

Any theoretical study of translation must address two issues. First, to what extent and in what way are the source and target language described? In some cases, there will be highly formal specification of the languages, while in other instances, the source or target languages are informally defined or implicitly understood. Notably, there may be a combination of formal and informal techniques; for instance, precisely defining the syntax of a language while loosely describing its meanings. A second facet of
the translation process that must be described is the translation mechanism itself. Once again there are a variety of approaches that range from informal description of the translator to highly mathematical formalisms for directing translations.

When care is taken to formally describe the languages and translation mechanism involved in a solution to the translation problem, it is possible to analyze the translator to assess its "correctness". Does it produce target sentences that are semantically equivalent to a source sentence input? We may also judge the power of the translation mechanism to translate between a wide variety of languages. Perhaps certain mechanisms cannot translate between as many languages as another mechanism. Formal analysis may also permit investigation of the efficiency of a specific translation mechanism to ascertain how much work it must do to effect a particular translation. This would permit us to judge if one translation mechanism was better than another at carrying out translations. This thesis addresses these issues and others that are raised by a formalization of the translation process.

It is useful at this point to enumerate several questions that help to characterize different approaches to the translation problem:

1. To what extent are the languages described? Is there a formal definition of the source language? Is there a formal definition of the target language?

2. Are both the syntax and semantics formally defined? Is the syntax described formally but the semantics given informally?

3. To what extent is the translation mechanism described? Is it a formal theoretical model of translation? Is it an informally described procedure? Is it a collection of ad hoc techniques?

4. Where is the syntactic description of a language used? Is it used formally in the translation mechanism? Is it implicitly used by the designer of the translator to make translation work? Is it used in analyzing the power or correctness of the translator?

5. Where is the semantic description of a language used? Is it used formally in the translation mechanism? Is it implicitly used by the designer of the translator to make the translation work? Is it used in analyzing the power or correctness of the translator?
Such questions are helpful in studying the historical context for our research and in identifying the particular translation methodologies that are examined in this thesis.

1.3. Historical perspective

Interest in employing computers to perform translation appeared early in the development of modern digital computers. In 1949, Warren Weaver [Wea49] suggested that the computer could be used to translate between natural languages. The next decade saw the attempted development of several natural language translators. However, their success was quite limited as they involved basically dictionary look up of language synonyms and word-for-word substitution [Edw63]. During the 1950's other translation problems were also arising in the use of computer systems. Programming computers in their primitive digital codes gave way to assembly language programming which required assembler programs to be transformed into digital machine code. FORTRAN was developed as a popular higher level programming language which required even more sophisticated conversion into machine language. Here, as in natural language translation, the techniques applied were principally ad hoc and not particularly systematic.

In 1959, Chomsky [Cho59] provided a needed boost to researchers in translation by providing a formalization of the syntax of language. This mathematical foundation for expressing syntax was especially influential on the development of programming languages and their translations. By 1960, context free grammars and the related Backus-Naur Form were being used to specify the syntax of languages [Bac60]; notably BNF was employed in the ALGOL60 Report. Eventually BNF descriptions were used to derive parsers that analyzed the syntax of programs [Iro61]. Parsing was actively investigated in the Sixties and schemes were introduced that used parsing of the source programming language to guide translation or compiling of that language [Bro62]. This is suggested in Figure 3 where the existence of a formal specification of the source language syntax is indicated. Note that only an informal idea of the source semantics and target language (here machine language) was employed, although notions of these were necessarily involved in developing the procedures that actually did translation. Use of formal syntactic description of natural languages proved much harder [Oet61],[Boo67] and despite considerable impetus given by Chomsky's work, natural language translation research slowed during the 1960's.
Attention to syntactic processing resulted in the notion of "syntax-directed translation" [Che64]. This was formalized by Lewis and Stearns [Lew68] into mathematical models of translation. Figure 4 depicts their approach to translation. Syntax was formally described for both the source and target languages, and the translation mechanism was formalized as a kind of pushdown automaton. Aho and Ullman [Aho69] performed a theoretical analysis of the same combination of translation formalisms. Their work on translation was one of the first important theoretical studies of the limitations of formal translation.

In connection with the scheme suggested by Figure 4, some other research in automata theory should be noted. Several researchers [Don65], [tha67], [Rou70] investigated the notion of tree automata—a generalization of finite state automata where trees rather than strings are the input. This led to the development of tree "transducers" that transform trees of one type into trees of another. Since trees are a popular means for describing the syntax of a language, tree transducers provided another formal tool to be used as a translation mechanism. Their work was the first to
Figure 4: Using formal specification of source and target syntax
suggest that a formal syntactic specification could usefully be provided for both the source and target language (see Figure 4).

Despite the research spurred on by a formal specification of the syntax of languages, little progress was made on the general problem of mechanized translation. As early as 1962, Bar-Hillel [Bar62] suggested that further progress awaited development of a theory of semantics similarly formal to that of the syntax. In fact, demonstrations of the effectiveness of syntax-directed translations required careful crafting of the language descriptions to indirectly account for semantics. For example, guiding translation of a programming language by parsing might require rather unusual syntactic descriptions that would force a meaningful sequence of actions by the translation mechanism.

Outside of translation, there was active research on formally specifying the semantics of language— in particular for programming languages. The focus on programming language semantics was a consequence of their more precise syntax and their more well-defined meanings when compared to natural language. One approach was interpretive semantics. Interpretive schemes displayed the meaning of a program as a sequence of descriptive steps or snapshots that suggest the operation of the program on particular data. An early example of this technique was McCarthy's [Mcc60] description of LISP using the lambda calculus of Church. The Vienna Definition Language developed by Lucas [Luc69] is another example. Interpretive semantics are particularly useful for describing how a program works.

Another approach taken was to develop so-called descriptive semantics. These methods attempted to define a program in another notation which hopefully is clearer than the programming language itself. The purpose is to explain what the program does rather than how it does it. Knuth [Knu68] made a significant suggestion for a descriptive semantics for programming languages. His attribute grammars provided a formal description of the semantics of a language which was closely tied to its formal syntactic definition. Figure 5 suggests Knuth's scheme. However, "translation" in his approach was more a computation of the meaning of a sentence. Noteworthy is the combination of formal semantics with formal syntax and a scheme for computing meaning.

It should not be surprising that descriptive semantics are most appropriate for studying translations. In translation, it is more important that a FORTRAN statement and an ALGOL statement are functionally equivalent rather than comparing how the FORTRAN statement is carried out in its environment versus the processing of the ALGOL statement in its environment. As a point of
clarification, the term "programming language semantics" is often identified with "axiomatic semantics" designed for program verification. In this thesis, there is no attention given to issues of program correctness; the focus remains on programs "performing the same function".

In the early 1970's, several researchers combined the formal specification of the syntax and semantics of languages with formal translation mechanisms to create a variety of models for formal translation (see Figure 6). Lewis, Rosenkrantz, and Stearns [Lew74] used Knuth's attributed grammars in proposing a translation scheme. Benson [Ben74] approached translation from the perspective of category theory. Like Lewis, Rosenkrantz, and Stearns, his translation mechanism was syntax-directed. But the presence of a formal semantics of the source and target language permitted him to make assertions about the correctness of the translation - i.e., guaranteeing translated sentences were semantically equivalent to the original source sentence.
Figure 6: Using formal syntax and semantics of both languages
Buttelmann [But74] suggested a model of language that included a formal grammar and formal specification of semantics based on that grammar. Elements of a language were not only required to be syntactically well-formed but they were also required to be meaningful. Thus a syntactically correct sentence could be excluded from a language if the formal semantics could not assign a meaning to it. Using this language model, Buttelmann investigated a form of syntax-directed translation between languages. Like Benson, he was able to identify conditions which could guarantee the correctness of the translation.

In review, the research of Lewis, Rosenkrantz, and Stearns, as well as Benson and Buttelmann, significantly used formal syntax and semantics for both the source and target language (see Figure 6). All involved a translation mechanism that was guided by the syntax of the source language. The distinction of Benson and Buttelmann's work was the use of formal semantics in the analysis of the correctness of translation.

Pyster [Pys75] extended the work of Buttelmann and Knuth, combining the language definition mechanisms of Buttelmann with the attributed semantics of Knuth. Pyster was the first to actually use the semantics of the source and target language in the translation mechanism. In contrast with the syntax-directed translation that had previously been employed, Pyster formalized the notion of semantic-directed translation (see Figure 7). This involved a mapping of a source sentence directly to some representation of its meaning. This process was then reversed as the meaning representation was converted to a target sentence. In addition, Pyster also developed several translators that were a compromise between syntactic and semantic guidance (see Figure 8). Meaning could explicitly be used to discard parses of source sentences that were meaningless or to eliminate meaningless target parse trees. Finally, meaning could be applied in the normally purely syntactic step of transforming source parse trees to target parse trees. Another contribution of Pyster was a careful comparison of the power of his various translation schemes. For example, he was able to show that certain translations, which were possible using a combined syntactic-semantic mechanism, were impossible using a purely syntactic approach.

Krishnaswamy [Kri76] provided even further analysis of translation methodologies such as those developed by Pyster and Buttelmann. Where Pyster examined relative power of the translation schemes, Krishnaswamy focused on the correctness of the translations induced by various methods. By examining the formal syntax and
Figure 7: Semantic-directed translation
semantics of the source and target languages, Krishnaswamy developed a set of conditions that assured that the translation preserved the meaning of a source sentence.

1.4. Translator Generation

As a consequence of the increased formalization of language definition and of specifying the translation mechanism, researchers were able to observe an emerging structure for translators—especially of programming language compilers. Brooker and Morris [Bro63] suggested the term "compiler-compiler" to describe a system which, when given a formal description of a source programming language syntax, would produce automatically a compiler for programs in that language (see Figure 9). The approach exploited the fact that certain components, like lexical analysis and parsing, were common to almost all compilers. Such systems could eliminate much work for compiler designers. However, handling the "semantics" of such programming languages was not well understood and still was handled in a largely informal manner.

Buttelmann [But74] articulated the theoretical version of this concept in developing his translator generators. Buttelmann suggested a translator generation scheme that would, from the formal description of the source and target languages, automatically produce a translation mechanism that would translate between the languages. Furthermore, the translation would be correct and semantic preserving. Krishnaswamy [Kri76] further investigated automatic generation of translators that are semantic preserving.

1.5. Computability and Complexity of Translator Generation

This thesis is primarily concerned with the problem of translator generation. Experience in developing translators for natural and programming languages suggests that construction of a translator is a difficult task. To better understand the difficulty, this thesis examines the problem from a theoretical perspective. Using formal language descriptions and formal translation methods, several issues related to translation generation are identified and analyzed. We focus on two questions:

1. Is it possible to do automatic translator generation?

2. When it can be done, how difficult is it?
Figure 8: Translation with semantic filtering
Figure 9: The structure of a "compiler-compiler"
Whether or not translator generation can be done is an issue of computability or decidability. It questions the existence of a procedure that, when given descriptions of arbitrary source and target languages, can discover a translator that correctly translates between the two languages (and perhaps can inform us if no translator exists should that be the case). It implies a single general procedure that should work for any pair of language descriptions of source and target. Not only should this one procedure be able to find a translator for descriptions of say, FORTRAN and ALGOL, but the same procedure should work for descriptions of any source-target pair of languages.

This basic computability question will be posed several times in this thesis for different versions of the translator generation problem. That it should frequently be answered negatively should not be a surprise in light of numerous related problems in computer science and mathematics that have been found to be undecidable or noncomputable.

The other basic question concerns the difficulty of actually doing translator generation in those situations where it is possible. It is a question of complexity. Experience in analyzing other problems prepares us to expect a range of possible answers. Translator generation procedure may be found to be intractable—possible to do but any procedure for doing it would require an inordinate amount of work. Such procedures may be found to be possibly intractable—closely relate to an established class of problems (e.g. NP-complete problems) that may well be intractable but cannot yet be proven to be. Perhaps translator generation procedures may be discovered that are provably good—that is they use a reasonable amount of resources to produce translators. This work attempts to organize a body of complexity results relating to the translator generation problem.

It is important to note that translator generation is performed on descriptions of the source and target languages, and thus sensitive to the particular schemes used to formally define the languages involved. The language definition schemes used in this thesis are closely related to the work of Buttelmann, Pyster, and Krishnaswamy; however, to permit analysis of complexity issues, more detail has been added to the definition of semantics. This will allow us to examine the impact that particular language definitions have on the work performed in translator generation.

Among the several models that have been proposed for the translation mechanism, this thesis focuses on a particular table driven process for carrying out translation. Thus, the end product of the translator generation process will be a table that can be
used in conjunction with a simple, standard tree transduction algorithm to convert source parse trees to target parse trees. Figure 10 suggests the relationship between the translator generation process and the particular translation mechanism featured in this work. The framework for the translators studied herein is also displayed in Figure 10. Its choice is consistent with the approach taken by Krishnaswamy [Kr76], Buttelmann [Bu74], and Pyster [Ps75]. Pyster showed the use of the semantic filter on source parse trees greatly increases the power of this translation scheme to translate between a wider variety of languages. However, Pyster also showed that avoiding use of semantics in the translation process will reduce the complexity of translation. In fact, the table driven tree transduction process that converts source parse trees to target parse trees is relatively efficient. By isolating the use of semantics in the source sentence parsing stage, the advantageous power of semantic processing is combined with the relative efficiency of syntactic processing in the remaining stages of the translation process.

Care should be taken to distinguish the two processes that have been mentioned. Other research has examined the process of translation - the actual conversion of source sentence to target sentence (which in this thesis will primarily be a table-driven process). This research studies the process of translation generation - the discovery of a table (or other specification) that, when plugged into a translator, will guide the translation process.

1.6. Key Issues Addressed

It is the goal of this research to identify and examine several issues concerning translator generation. In particular, we are interested in the existence of such translator generators for general classes of language definitions and languages. In the cases where translator generation is possible, we are curious about the complexity of procedures that carry it out. Buttelmann [Bu74], Pyster [Ps75], and Krishnaswamy [Kr76] have all touched on these problems of translator generation and the current work extends and develops their ideas.

Specifically, the current work focuses on three issues of translator generation. First, we examine the computability of translator generation, both in general and for restricted classes of languages and language definitions. Second, we assess the complexity of translator generation where it is possible. Finally, we explore the computability and complexity of oracular translator generation - an approach in which an external decision making source
Figure 10: Translator generation producing "plug in" tables
is utilized in generation of translators. The important questions are summarized below.

**Computability**

Our research endeavors to answer the following:

1. Is translator generation a computable problem?

   Is it decidable if a translation exists between two languages?

   Is it decidable if a table translation exists between two languages?

   Is it decidable if a table translation exists between a pair of languages as defined by specified language definitions?

2. Are there necessary and sufficient conditions that guarantee the existence of a translation or table translation between a pair of languages (or language definitions)?

3. Should the answers to question 1 be negative, what can we say about the decidability of the corresponding questions for suitable restrictions of the class of languages or language definitions?

   a. Are there noteworthy observations we can make about those classes of languages or language definitions for which the translator generation problem is decidable?

   b. What restrictions (on the definition of either the syntax or semantics or both) on language definitions could be enforced to guarantee that translator generation is decidable?

4. Should the answers to question 1 be negative, what can we say about the relationship of the computability of the various translator generation problems to the computability of other known problems?
An answer to this last question would suggest how difficult the computability of the translator generation problem is by relating it to other noncomputable problems. Moreover, answers to this question will suggest what type of information (or decisions) might have to be supplied by an oracle to actually do oracular translator generation.

**Complexity**

Where we are able to find a class of language definitions for which translator generation is computable, we can pose the following:

1. Can we comment on the complexity of procedures that actually perform various types of translator generation?

   a. Is the problem intractable (i.e. all procedures carrying out a solution require an inordinate amount of resources most of the time)?

   b. Is the problem related in complexity to other known problems that may or may not be tractable (e.g. is it NP-complete, etc.)?

   c. Is there a good algorithm for performing translator generation and can it be specified?

2. In the case of provably or possibly intractable translator generation problems for a class of language definitions, is the result a consequence of how we choose the language definitions?

   Put another way, the question asks if a pair of language definitions force a translator generation procedure to work hard, would there be alternative language definitions for the same source and target languages that would not be too bad from a complexity point of view?

**Oracular translator generation**

We explore the possibility of using an "external agent" to assist in generating translators:

1. What type of information could be supplied by an oracle
in order to direct oracular translator generation?

2. Would it be possible to decide if a table translation exists between language or language definitions if such information were provided by an oracle?

3. Where it is possible, what can we say about the complexity of any procedure that performs oracular translator generation?

   a. What would be the complexity of the procedure excluding any work done by the oracle?

   b. Can we say anything about the number of calls it would make on the oracle?
2. BASIC TERMINOLOGY AND DEFINITIONS

In this chapter, we establish the basic terminology and notation that will be used in this thesis. Most of the concepts are consistent with accepted usage in the area of formal languages and the theory of computation and will be noted without comment. Additional explanation is included when ideas have been specially adapted or extended for this research.

2.1. Proof Notation

In preparing the proofs in this thesis we will frequently make use of simplifying notation when it adds clarity to the presentation. In general, use will be made of parentheses and bracketing to isolate components of proof statements and equations. In addition:

\[ \Rightarrow \] will be used to denote implication (if...then construction) when it adds clarity. At other times the words "if" and "then" will be used.

s.t. may be used for the phrase "such that"

o/w represents "otherwise"

BWOC denotes "by way of contradiction"

2.2. Strings

Strings are fundamental to this work and will normally be represented by lower case letters w, x, and y. Multiple strings may be distinguished by subscripting or numbering. Symbols in strings will normally be represented by lower case letters like a and v and also may be subscripted. Definitions and notation related to strings are:

alphabet is a finite, nonempty set of symbols
ab or a//b is the concatenation of a and b

string is a concatenation of an ordered finite sequence of zero or more symbols

NULL or λ is the null string consisting of no symbols

|w| is the length of string w ( |λ| = 0 )

a^k is the string consisting of k consecutive occurrences of symbol a

A* is the set of strings formed from alphabet A that are of length 1 or more

A^e is the set A* together with the null string

2.3. Sets

Normally sets will be collections of strings and will usually be represented by upper case letters A, B, etc. Different sets may be distinguished by subscripting or numbering. In the following, suppose A and B are sets of strings.

w in A or w∈A indicates that w is an element of A

A U B is the union of A and B

A - B is the set difference of A and B

A ∩ B is the intersection of A and B

AB or A//B is the set of all strings x//y such that x is in A and y is in B

A X B is the set of all ordered pairs (x,y) such that x is in A and y is in B

|A| is the number of elements in the set A

POWERSET(A) is the set of all nonempty subsets of set A

SEQ(A) is the set of all finite ordered sequences of elements of A

NAT is the set of natural numbers 0, 1, 2, ...
When it is necessary, NAT can be identified with the set of strings on some single symbol alphabet $A = \{a\}$. The association that is natural is $n = |w|$ with $w$ in $\{a\}^*$.

2.4. Functions, Predicates, and Computations

This thesis deals primarily with computability and complexity. It is presumed that the reader is familiar with fundamental concepts and terminology in these areas. A variety of texts (including Rogers [rog67] and Machtey and Young [Mac78] introduce these topics. To familiarize the reader with the terminology and usage appearing in this work, an informal narrative follows. When unusual notation or special use of a term is employed, a definition is provided.

Suppose $D_1, ..., D_m, R_1, ..., R_n$ are sets of strings on some alphabet. An $m+n$-ary relation $F$ is any subset of $D_1 \times ... \times D_m \times R_1 \times ... \times R_n$. In this case

$$\text{DOM}(F) = \{(d_1, ..., d_m) | (d_1, ..., d_m, r_1, ..., r_n) \text{ in } F \text{ for some } (r_1, ..., r_n) \text{ in } R_1 \times ... \times R_n \}$$

is called the domain of $F$.

$$\text{RAN}(F) = \{(r_1, ..., r_n) | (d_1, ..., d_m, r_1, ..., r_n) \text{ in } F \text{ for some } (d_1, ..., d_m) \text{ in } D_1 \times ... \times D_m \}$$

is called the range of $F$. Should $\text{DOM}(F) = D_1 \times ... \times D_m$ we say $F$ is total and should $\text{RAN}(f) = R_1 \times ... \times R_n$ we say $F$ is onto. When each element $(d_1, ..., d_m)$ is uniquely matched with an element of $\text{RAN}(f)$ and each element $(r_1, ..., r_n)$ is uniquely matched with an element of $\text{DOM}(f)$ then $F$ is said to be one-to-one.

When each element $(d_1, ..., d_m)$ of $\text{DOM}(f)$ is associated with exactly one element $(r_1, ..., r_n)$ of $\text{RAN}(f)$ then $F$ is a function. This is denoted by

$$F: D_1 \times ... \times D_m \rightarrow R_1 \times ... \times R_n$$

and a particular association is specified by

$$F(d_1, ..., d_m) = (r_1, ..., r_n).$$

Functions, rather than relations are used mostly in this thesis, and are often designated by $f$, $g$, $h$, $F$, and $H$. A function is said to be
defined at an argument \((d_1, \ldots, d_m)\) in the specification above. A function is **cofinite** if it fails to be defined for only a finite number of arguments in \(D_1 \times \ldots \times D_m\).

**DEFINITION 2.4.1: Special functions**

Suppose \(A, B, A_1, \ldots, A_j, B_1, \ldots, B_k, C_1, \ldots, C_m, D_1, \ldots, D_n\) are sets and \(f\) and \(g\) are functions where:

- \(f: A_1 \times \ldots \times A_j \to B_1 \times \ldots \times B_k\)
- \(g: C_1 \times \ldots \times C_m \to D_1 \times \ldots \times D_n\)

1) Then \(1STELT: A \times B \to A\) is the function that selects the first of the arguments, i.e. \(1STELT(a, b) = a\) for all \((a, b)\) in \(A \times B\)

2) \(2NDELT: A \times B \to B\) is the function that selects the second of two arguments, i.e. \(2NDELT(a, b) = b\) for all \((a, b)\) in \(A \times B\)

3) \(f \# g: A_1 \times \ldots \times A_j \times C_1 \times \ldots \times C_m \to B_1 \times \ldots \times B_k \times D_1 \times \ldots \times D_n\)

   is defined by

   \[f \# g(a_1, \ldots, a_j, c_1, \ldots, c_m) = (b_1, \ldots, b_k, d_1, \ldots, d_n)\]

   iff

   \[f(a_1, \ldots, a_j) = (b_1, \ldots, b_k)\] and \(g(c_1, \ldots, c_m) = (d_1, \ldots, d_n)\)

4) \(f \% g: C_1 \times \ldots \times C_m \to B_1 \times \ldots \times B_k\) denotes the functional composition of \(f\) and \(g\).

**DEFINITION 2.4.2: Special functions on NAT**

Let \(A\) be any alphabet.

\(PAIR: \text{NAT} \times \text{NAT} \to \text{NAT}\) is a function that builds a unique correspondence between \(\text{NAT} \times \text{NAT}\) and \(\text{NAT}\):

i) \(PAIR\) is one-to-one and onto

ii) \(PAIR(n, m) < PAIR(n+1, m)\) and \(PAIR(n, m) < PAIR(n, m+1)\)

for all \(n, m\) in \(\text{NAT}\)
COD_A : A* -> NAT and DEC_A : NAT -> A* are one-to-one and onto functions defined so that COD_A \circ DEC_A is the identity function on NAT and DEC_A \circ COD_A is the identity function on A*.

PAIR is referred to a "pairing function" and COD and DEC are called "coding and decoding functions" for A (the subscript will be omitted when the alphabet is clear). It is not necessary to specify a particular pairing function or coding functions for A. It is sufficient to note there are many natural ways to define functions with these properties. Also, PAIR can be extended to domains NAT \times NAT \times \ldots \times NAT in a natural way that preserves the one-to-one and onto correspondence. To aid readability \langle \rangle will often be used instead of PAIR: \langle n, m \rangle = PAIR(n, m)  

**DEFINITION 2.4.3: Homomorphism on strings**

Let A and B be alphabets. A string homomorphism is specified by providing a total function h from A to B*.

h is extended to all of A* by:

i) h(\lambda) = \lambda

ii) h(a) = a for all a in A

iii) h(xy) = h(x)\cdot h(y) for all x, y in A*

We refer to the extension of h to the domain A* as a homomorphism.

**Predicates** capture the mathematical notion of "properties". To suggest the entity (a_1, \ldots, a_n) in A_1 \times \ldots \times A_n has property P is written as P(a_1, \ldots, a_n). A characteristic function of predicate P is a two-valued function that maps (a_1, \ldots, a_n) to one value if (a_1, \ldots, a_n) has property P and to the other value if (a_1, \ldots, a_n) doesn't have property P.

A variety of models of computation have been advanced for specifying the algorithmic computation of functions. Examples of such models are Turing machines and Markov algorithms. These are used to specify procedures that, when given an element of the domain
of a function, produce the appropriate element of the range. The 
\textit{partial recursive functions} are identified as those functions 
\textit{computable} by a Turing machine (or some other accepted model of 
computation). A \textit{recursive function} is a partial recursive function 
that is also total. \textit{Primitive recursive functions} are a well-known 
class of functions that also have a precise formal definition. 
Predicates may also be referred to as recursive or primitive 
recursive by referring to the associated characteristic function. 
\textit{Recursively enumerable sets} (r.e. sets) are those that can be 
specified by the set of partial recursive functions (i.e. an 
r.e. set is one that is the domain of a partial recursive function 
or is the range of a partial recursive function).

The functions that have been included in this chapter as 
special functions (1STELT, 2NDELT, PAIR, COD, and DEC) are known to 
be primitive recursive (and hence partial recursive). If the 
component functions are primitive (partial) recursive then their 
combination using either the \% or \# operations is also primitive (partial) recursive.

Finally, we may want to speak about the ability to decide the 
membership of elements in one set based on the membership of related 
elements in another set. A set \(A\) is said to be (many-one) reducible 
to a set \(B\) if there is a recursive function \(f\) such that \(x \in A \iff f(x) \in B\).

\textit{Turing machines} are a well-known model of computation 
consisting of two-way infinite tapes on which symbols are written, 
alphabets of symbols that may appear on the tapes, a set of states 
that the machine may be in, and a program that directs the operation 
of the Turing machine. This program indicates what step the Turing 
machine next takes based on what's on the tapes and what state the 
machine is in. Turing machines can be used either to compute 
functions (by receiving as input the argument of a function and 
leaving as output on the tape the desired associated element of the 
range). They may also be used as recognizers of predicates by 
receiving arguments and halting in either an accepting state or 
rejecting state (for recursive sets) or perhaps not halting at all 
(for r.e sets). A Turing machine is said to be \textit{deterministic} if at 
any point in its computation there is no ambiguity (i.e. no 
alternatives) about what its next step will be. A Turing machine is 
\textit{nondeterministic} if at any step there may be a finite number of 
alternative actions that could be taken. \textit{DTM} will denote a 
deterministic Turing machine while \textit{NDTM} will denote a 
nondeterministic Turing machine.
2.5. Computability and Complexity

This thesis deals with various "problems" of translation and translator generation. In order to formally deal with these issues, each problem's solution is cast as a function or predicate. That is, we seek a function that maps instances of a problem to the specific solution for that instance. Problems for which there is a procedural solution (e.g., a Turing machine that computes the function) are said to be computable. Should no procedural solution exist, the problem is unsolvable or noncomputable. Decision problems are those whose solutions are "yes" or "no" answers. Decision problems are characterized as decidable or undecidable.

In order to avoid attaching results in this thesis to a particular model of computation, an abstract notion of these models is developed. The approach here is that of Machtley and Young [Mac78]. It gives a general way of speaking about any reasonable system for specifying procedures that compute functions.

**DEFINITION 2.5.1: Acceptable programming system**

Let <> be a pairing function. An acceptable programming system is a listing \( f_0, f_1, \ldots \) of all partial recursive functions from \( \text{NAT} \) to \( \text{NAT} \) where:

1) there is an index \( \text{UNIV} \) in \( \text{NAT} \) such that for all \( i, x \) in \( \text{NAT} \) \( f_{\text{UNIV}}(\langle i, x \rangle) = f_i(x) \)

2) there is a total recursive function

\[
\text{COMP}: \text{NAT} \times \text{NAT} \rightarrow \text{NAT}
\]

such that

\[
f_{\text{COMP}}(i, j) = f_i \% f_j
\]

\( f_{\text{UNIV}} \) is a universal partial recursive function and \( \text{COMP} \) is associated with the operation of functional composition. [*]

This definition implies that there is a particular function (indexed by \( \text{UNIV} \)) that can simulate the action of any other listed function on any input. \( \text{COMP} \) provides an effective way of performing the composition of functions. An example of an acceptable programming system is the set of functions computed by the set of Turing machines. In this case the indexing of the functions corresponds to
a listing of the descriptions of the Turing machine programs.

A significant and frequently used property of acceptable programming systems is the ability to alter "programs" so that certain "inputs" are held constant. This property is captured by the existence of an "s-m-n function".

**DEFINITION 2.5.2: s-m-n function**

Let $\langle \rangle$ be a pairing function. Let $f_0, f_1, ...$ be an acceptable programming system. Then the s-m-n function $\text{SMN}$

$\text{SMN}: \text{NAT} \times \text{NAT} \rightarrow \text{NAT}$ is defined by

$$f_{\text{SMN}}(\langle i, m, x_1, ..., x_m \rangle)(y_1, ..., y_n) = f_i(x_1, ..., x_m, y_1, ..., y_n)$$

for all $i, m, n, x_1, ..., x_m, y_1, ..., y_n$ in $\text{NAT}$, $m, n > 0$. [∗]

It should be noted that for any acceptable programming system, the existence of an s-m-n function is assured [Mac78]. Of course, in many actual programming systems there is a natural and obvious method for constructing "programs" with certain of their inputs held constant. Another function found in studying acceptable programming systems is $\text{STEP}$.

**DEFINITION 2.5.3: STEP**

Let $f_0, f_1, ...$ be an acceptable programming system.

Then $\text{STEP}: \text{NAT} \times \text{NAT} \times \text{NAT} \rightarrow \text{NAT}$ is a recursive function defined so that:

1) There is a $k$ in $\text{NAT}$ such that $\text{STEP}(i, j, k) > 0$

iff

$$f_i(j) \text{ defined}$$

2) If $\text{STEP}(i, j, k) > 0$, then $\text{STEP}(i, j, k) = f_i(j) + 1$
Once again, the existence of the function \( \text{STEP} \) is assured for acceptable programming systems. In the specific case of Turing machines, \( \text{STEP} \) might be the function that asks whether a particular Turing machine, on a given input, has halted in exactly \( k \) steps. The \( \text{STEP} \) function is often referred to as Kleene's \( T \)-predicate.

Many decision problems have been characterized as undecidable by showing that their characteristic functions are not computable. But there is a structure that exists even among the undecidable problems, and hence some problems are "more undecidable" than others. Kleene [Kle36] established an arithmetic hierarchy that arranges problems according to their relative computability.

Consider the collection of sets denoted by total characteristic functions (here we will assume over domains of the natural numbers). These may be thought of as functions that characterize decision problems or as representing predicates.

**DEFINITION 2.5.4:** \( \Pi_n \sum_n \)

Let \( \langle \rangle \) be a pairing function. Then

\[
\Pi_0 = \sum_0 = \text{all sets characterized by recursive functions}
\]

\[
\sum_{n+1} = \text{the sets characterized by}
\begin{align*}
\{ f & : f(x) = 1 \text{ if there is a } y \text{ s.t. } g(\langle x, y \rangle) = 1 \\
& = 0 \text{ o/w}
\end{align*}
\text{ for all } x \text{ in NAT, for some } g \text{ in } \Pi_n \}
\]

\[ \Pi_n = \text{the sets characterized by}
\begin{align*}
\{ f & : f(x) = 1 - g(x) \text{ for some } g \text{ in } \sum_n \}
\end{align*}
\]

The collection of sets in \( \Pi_n \) and \( \sum_n \) for some \( n \) in NAT is referred to as the arithmetic hierarchy. Many observations may be made and proven about the arithmetic hierarchy. They are given here without proof (for reference on proofs, see [rog67]).

1) \( \Pi_0 \) and \( \sum_0 \) characterize the recursive sets

2) \( \sum_1 \) are the r.e. sets

3) A function \( f \) characterizes a set in \( \sum_k \) if it can be described by a sequence of \( k \) alternating quantifications
of the form:

\[ f(x) = (\exists x_1)(\forall x_2)...(\exists x_k) [g(x,x_1,...,x_k)] \]

where \( g \) is recursive.

4) A function \( f \) characterizes a set in \( \Pi_k \) if it can be described by a sequence of \( k \) alternating quantifications of the form:

\[ f(x) = (\forall x_1)(\exists x_2)...(\exists x_k) [g(x,x_1,...,x_k)] \]

where \( g \) is recursive.

NOTE: In observations 3 and 4, the notation is meant to suggest that there are \( k \) quantifications alternating between \( \forall \) and \( \exists \).

5) \( \Pi_k \cup \Sigma_k \) is properly contained in \( \Pi_{k+1} \cap \Sigma_{k+1} \) for all \( k \) in \( \text{NAT} \).

6) \( \Pi_k \) is properly contained in \( \Pi_{k+1} \) and \( \Sigma_k \) is properly contained in \( \Sigma_{k+1} \).

7) To show that a set \( A \) is \( \Sigma_k \)-complete (i.e. contained in \( \Sigma_k \cap \Pi_k \) but not in \( \Sigma_k \cap \Pi_k \)) we need only to show that

a) \( A \) is in \( \Sigma_k \)

b) find a known \( \Sigma_k \)-complete set and reduce it to \( A \)

A corresponding approach must be taken for \( \Pi_k \)-complete.

The arithmetic hierarchy is schematically described in Figure 11.

**Complexity**

In assessing the inherent difficulty of a solution to a translation problem, use is made of several accepted notions in
Figure 11: The Arithmetic Hierarchy
complexity. Specific analyses utilize Turing machines as a model of computation. The next definition permits us to speak about the "time" and "space" taken by a Turing machine to process a particular input.

**DEFINITION 2.5.5: Space and time measures for Turing machines**

Let $M$ be a Turing machine. Then

$$D_{TM}[time]_M(x_1, \ldots, x_k) = |x_1/\ldots/x_k| + \text{the number of steps carried out in } M\text{'s computation on } (x_1, \ldots, x_n) \text{ if } M\text{ halts on input } (x_1, \ldots, x_n)$$

$$D_{TM}[space]_M(x_1, \ldots, x_k) = \text{the number of tape squares used by } M \text{ (originally occupied by } (x_1, \ldots, x_k) \text{ or later visited) in its computation on input } (x_1, \ldots, x_k) \text{ if } M\text{ halts on input } (x_1, \ldots, x_n)$$

$D_{TM}[time]$ or $D_{TM}[space]$ will be undefined at $(x_1, \ldots, x_k)$ should $M$ not halt on input $(x_1, \ldots, x_k)$

$N_{DTM}[time]_M$ and $N_{DTM}[space]_M$ are similarly defined but respectively refer to the minimum time or space needed among all accepting computations of $M$ on input $(x_1, \ldots, x_k)$ (if there is an accepting computation) [*]

### 2.6. Trees

Trees are usually denoted by the letter $t$ and are subscripted or denoted $t_1, t_2$, etc.

**DEFINITION 2.6.1: Tree**

Let $V$ be a finite nonempty alphabet. Then a tree over $V$ is defined to be either:

1) $A$ in $V$

2) $A\langle t_1 t_2 \ldots t_k \rangle$ where $k > 0$, $A$ in $V$, and $t_1, \ldots, t_k$ are trees over $V$
TREE(V) will refer to the set of all trees over V.

Although the "angle bracket" notation above often appears in the literature, it also popular to present trees as branching diagrams.

![Branching diagrams of trees](image)

Figure 12: Branching diagrams of trees

**EXAMPLE 2.6.1:** A<B C<D B>> and B<C B> are represented in Figure 12.

Since this concept of trees is familiar, it is assumed the reader is also familiar with the ideas of node, height, root, and frontier of a tree. In this thesis, these are denoted:

- HGT(t) is the height of tree t
- ROOT(t) is the root of a tree t (its "starting point")
- FR(t) is the frontier of tree t (left to right concatenation of symbols appearing at the "leaves" of the tree)
- FR(t,i) is the symbol appearing at the ith position of the frontier
Composition

Translation mechanisms in this thesis depend heavily on building up larger trees out of smaller trees. To this end, composition is defined and a notation presented that permits us to clearly specify the compositional structure of a tree. We begin with an example that illustrates tree composition.

EXAMPLE 2.6.2: The composition of trees in Figure 12 can be accomplished in either of two ways to form respectively $A<B C B>D B>C B>>$ or $A<B C<D B>C B>>$ depicted in Figure 13.

Figure 13: Composition of trees

Note that the composition requires the matching of the root of one tree with a node on the frontier of another tree. To unambiguously define composition three things must be specified - the tree that will be extended, the tree that will be composed on it, and the position on the first tree's frontier where the composition should take place.
DEFINITION 2.6.2: Tree composition COMP

Let \( t \) and \( t' \) be trees over \( V \). Then the partial function

\[
\text{COMP}: \text{TREES}(V) \times \text{TREES}(V) \times \text{NAT} \rightarrow \text{TREES}(V)
\]

is defined by

\[
\text{COMP}(t, t', k) =
\begin{align*}
t' & \quad \text{if } t \in V \text{ and ROOT}(t') = t \text{ and } k = 1 \\
\text{or} & \\
A\langle t_1...t_{j-1} \, \text{COMP}(t_j, t', d) \, t_{j+1}...t_n \rangle & \quad \text{if } t = A\langle t_1...t_n \rangle \text{ and } \\
i) & |FR(t_1)\,...\,|FR(t_{j-1})| < k, \\
& k \leq |FR(t_1)...FR(t_j)| \\
\text{for some } j = 1,...,n \\
\text{and ii)} & d = k - |FR(t_1)...FR(t_j)|
\end{align*}
\]

COMP\((t, t', k)\) is undefined if any of these conditions are not met. [\(*\)]

DEFINITION 2.6.3: \( \text{COMP}^\ast_T \)

Let \( T = \{t_1,\ldots,t_m\} \) be an ordered finite set of trees in \( \text{TREES}(V) \) for alphabet \( V \). Then

\( \text{COMP}^\ast_T: \text{finite subsets of } \text{NAT} \times \text{NAT} \rightarrow \text{TREES}(V) \)

is defined so that if

\[
C = \{(j_0,k_0),...(j_n,k_n)\}
\]

is an ordered finite subset of \( \text{NAT} \times \text{NAT} \) then

\[
\text{COMP}^\ast_T = \text{COMP}(...(\text{COMP}(\text{COMP}(t_{j_0}, t_{j_1}, k_1), t_{j_2}, k_2),\ldots), t_{j_n}, k_n))
\]  [\(*\)]
Box Bracketing

In conjunction with the composition operation, it is useful to be able to refer to particular parts of a tree’s overall structure. The box-bracketing notation provides that ability:

DEFINITION 2.6.4: Box-bracketing

Let $T = \{t_0, \ldots, t_n\}$ be an ordered set of trees over $V$ and suppose $k_0, \ldots, k_n$ are in $\text{NAT}$. Then $t_0 \text{[} t_1 \ldots t_n \text{]}$ is a box-bracketing of tree $t$ over $V$

iff

$$t = \text{COMP}^*_T(\{(t_0, k_0), \ldots, (t_n, k_n)\})$$

where $k_1 = 1$, $n = |\text{FR}(t_0)|$

and $k_i = |\text{FR}(t_1) / \ldots / t_{i-1})| + 1$ for $i = 2, \ldots, n$ [*]

Essentially a box bracketing isolates the tree $t_0$ and the trees $t_1, \ldots, t_n$ that "hang" from the frontier of $t_0$. Hence it must be the case that $\text{FR}(t_0, i) = \text{ROOT}(t_i)$ for $i = 1, \ldots, n$ for the COMP* operation to be defined.

![Figure 14: Box-bracketings of Trees](image-url)
EXAMPLE 2.6.3: Consider the tree \( t = A\langle B \ C \ D \ B \langle C \ B \rangle \rangle \)
of the previous example. Box-bracketings of \( t \) are:

\[ A\langle B \ C \rangle [ B \ C \langle D \ B \langle C \ B \rangle \rangle ] \text{ and } A\langle B \ C \langle D \ B \rangle \rangle [ B \ D \ B \langle C \ B \rangle \rangle ] \]

Figure 14 diagrams these box-bracketings.

Note that a tree may have multiple box bracketing. It is useful to refer to \( t_0 \) as the supertree and \( t_1, \ldots, t_k \) as subtrees in the box bracketing and to refer to any of \( t_0, \ldots, t_k \) as component trees. When \( t_0[ t_1, \ldots, t_k ] \) is a box-bracketing of tree \( t \) we will denote this by \( t = t_0[ t_1, \ldots, t_k ] \). [\(*\)]

Using the composition operation on some "starting set" of trees can permit the generation of a large number of trees. This idea is captured in the following definitions.

DEFINITION 2.6.5: GEN(T)

Let \( T \) be a finite ordered set of trees over alphabet \( V \).

\[ \text{GEN}(T) = \{ \text{COMP}^*_{T}(C) \mid \text{for all finite sequences } C \text{ in } \text{SEQ}(\text{NAT} \times \text{NAT}) \} \] [\(*\)]

2.7. Context Free Grammars

Context free grammars are the basis of syntactic descriptions of languages in this thesis. To facilitate our discussion, trees are used instead of the traditional productions or rewriting rules.

DEFINITION 2.7.1: Context free grammar (cfg)

\( G = (VN, VT, PR, AX) \) is a context free grammar
iff

**VN**
is a finite nonempty set of symbols referred to as **nonterminals**.

**VT**
is a finite nonempty set of symbols such that **VN** ∩ **VT** is empty and is referred to as a set of **terminals**.

**PR**
is a finite set of trees over (**VN** ∪ **VT**) of height 1 such that for all **t** in **PR**, ROOT(**t**) is in **VN**. Each **t** is a **production**.

**AX**
is a distinguished element of **VN** referred to as the **axiom**.

Unlike the rewriting method specified by traditional grammars, derivation of syntactic elements in our system depends on tree composition. We are interested in sets of trees rooted in the same nonterminal as well as trees that have only terminals at their frontier.

**DEFINITION 2.7.2: Generating sets of trees**

Let **G** = (**VN**, **VT**, **PR**, **AX**) be a **cfg**, with **A** in **VN**. Then generating sets for **G** are defined:

**GEN(\(G\))** = **GEN(\(PR\))**

**GEN(\(G,A\))** = \{ **t** in **GEN(\(G\))** | ROOT(**t**) = **A** \}

**GENC(\(G,A\))** = \{ **t** in **GEN(\(G\))** | ROOT(**t**) = **A** and FR(**t**) in **VT**+ \}

**GENC(\(G\))** = \{ **t** in **GEN(\(G\))** | FR(**t**) in **VT**+ \}

Trees in **GENC(\(G\))** are referred to as **complete trees**.

In the literature, the context free grammar is employed to define a language. In our work, the term language is reserved for a more complicated concept. Herein, grammars are used to define a set of **sentences** (understood to be merely syntactically well-formed entities).
DEFINITION 2.7.3: SEN(G)

Let $G = (VN, VT, PR, AX)$ be a cfg. Then

$$SEN(G) = \{ w \mid w = FR(t) \text{ and } t \in GENC(G, AX) \}$$

DEFINITION 2.7.4: Special context free grammars

Consider cfg $G = (VN, VT, PR, AX)$

$G$ is **linear** if for all $t$ in $PR$, there is at most one nonterminal appearing in the frontier of $t$

$G$ is **regular** if for all $t$ in $PR$, $t = B<C$ where $B, C$ in $VN$ and $v$ in $VT$

$G$ is **finite** if $GEN(G)$ is finite
3. LANGUAGES AND LANGUAGE DEFINITIONS

This chapter presents the formal models of language and translation used in this thesis. They are closely related to the approaches of Buttelmann [But74], Pyster [Pys75], and Krishnaswamy [Kri76], but are tailored to be used in computability and complexity analyses. Our language model provides for both syntax and semantics and our language specification includes a way of describing the form and meaning of language elements. We will propose systems of language definitions that permit selection of the symbols and operations that specify individual languages. In particular, we will extensively develop a language definition system with meanings and operations drawn from the natural number system.

In addition to basic definitions, this chapter includes the development of certain theoretical results fundamental to our translation studies.

3.1. Languages

The traditional formal languages view of language is that of a set of strings over some alphabet. Specifically, there are three important aspects of the standard approach:

1. A syntactic alphabet is specified.

2. A language is simply any collection of strings, each string of finite length, over the syntactic alphabet.

3. A language is defined by a formal grammar (e.g. a cfg)

While this approach is useful, it is insufficient for studying translation between languages, where the meaning of language entities is as important as their syntactic form. In the model of language used here there are notable differences:

1. Both a syntactic and semantic alphabet are specified.

2. A language is a collection of sentence-meaning pairs, drawn
from the respective alphabets.

3. A language definition includes not only a formal grammar for describing a syntactically well-formed language element, but also a semantics that attaches meaning to it.

This pairing of sentence with meaning allows several language issues to be formally discussed. For example, the ambiguity of a language can be addressed by permitting a particular sentence to be associated with multiple meanings. In translation, the semantic equivalence of a source and target sentence can be readily checked.

**DEFINITION 3.1.1: Language**

Let $V$ be an alphabet (called the syntactic alphabet).
Let $A$ be an alphabet (called the semantic alphabet).

A language $L$ is any subset of $V^* \times A^*$. We refer to $L$ as a language on $V$ and $A$. \(^*\)

Remember here that a language is not a collection of sentences in the normal use of the term "sentence". For us, a sentence refers to the syntactic component $w$ of a language element $(w,m)$ while $m$ is the meaning of $w$.

**DEFINITION 3.1.2: MEAN(w), MEAN(L), and MSEN(L)**

Let $L$ be a language on $V$ and $A$ and $w$ in $V^*$.

$\text{MEAN}(w) = \{m \in A^* \mid (w,m) \in L\}$

$\text{MEAN}(L) = \{m \in A^* \mid (w,m) \in L \text{ for some } w \in V^*\}$

$\text{MSEN}(L) = \{w \in V^* \mid (w,m) \in L \text{ for some } m \in A^*\}$ \(^*\)

Later on, it will be seen that it is possible that a string on the syntactic alphabet is well-formed but meaningless (i.e. not in $\text{MSEN}(L)$). This captures the notion of a nonsense sentence in English (e.g. "John stubbed his stomach").

**EXAMPLE 3.1.1:** Let $A = \{0,1,\ldots,9\}$ and $V = \{0,1\}$

Consider $L = \{(w,m) \mid w \text{ is a string of } 0\text{'s and } 1\text{'s \begin{equation*} \end{equation*} beginning and ending in } 1 \text{ and }$
m is the value of w treated as a binary number

\[=\{(1,1),(11,3),(101,5),\ldots\}\]

\[\text{MEAN}(1001) = \{7\} \quad \text{MEAN}(L) = \{1,3,5,\ldots\}\]

**DEFINITION 3.1.3: Ambiguous and unambiguous languages**

A language \(L\) on \(V\) and \(A\) is ambiguous if there is some \(w\) in \(\text{MSEN}(L)\) such that \((w,m_1)\) and \((w,m_2)\) are in \(L\) but \(m_1\) and \(m_2\) are distinct.

A language \(L\) on \(V\) and \(A\) is unambiguous if for all \(w\) in \(\text{MSEN}(L)\), \(|\text{MEAN}(w)|\) is 1. \([*]\)

An ambiguous language has at least one sentence with two or more distinct meanings. An unambiguous language associates a single unique meaning with every meaningful sentence.

### 3.2. Language Definitions

Traditionally, formal grammars have been used to define languages of a purely syntactic type. For our fuller notion of language a more powerful language definition scheme is needed. The philosophy is that expressed by Buttelmann [But74]:

The definition 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 meaning of its constituents.

The context free grammar of our definition scheme provides the syntactic structure. It remains for us to associate meaning with the constituents and to provide a mechanism for conveying meaning in the syntactic structure.

Buttelmann introduced three formal notions that associate a semantics with a context free grammar. The first identifies the set of meanings used in the definition scheme. This includes not only meanings of sentences but also includes the "unit meanings" from which other meanings are determined. He also specified the set of meanings that might be associated with a symbol in the cfg.
Consider terminal symbols, the obvious constituents of a sentence. It is clear that their meanings should contribute to the overall meaning of a sentence. In addition, Buttelmann showed precisely how the meaning associated with a syntactic structure is a function of its constituent meanings. These notions are captured formally as the universe of discourse, meaning function, and a collection of semantic functions that are associated with a context free grammar.

**DEFINITION 3.2.1: Universe of discourse**

Let \( A \) be an alphabet. Then a universe of discourse \( U \) is any subset of \( A^* \). 

**DEFINITION 3.2.2: Meaning function \( MU \)**

Let \( G=(VN,VT,PR,AX) \) be a cfg and \( U \) a universe of discourse. Then a function \( MU:(VN \cup VT) \rightarrow \text{POWERSET}(U) \) is a meaning function if:

i) \( MU \) is total

ii) \( MU(v) \) is an r.e. set for all \( v \) in \( (VN \cup VT) \)

Essentially \( MU \) identifies the set of possible meaning associated with each nonterminal or terminal in the grammar \( G \).

**DEFINITION 3.2.3: Semantic function**

Let \( G = (VN,VT,PR,AX) \) be a cfg, \( U \) a universe of discourse and \( MU \) a meaning function based on \( G \) and \( U \). Suppose \( t \) in \( (\text{GEN}(G) \cup VN \cup VT) \) with \( \text{ROOT}(t) = v_0 \) and \( \text{FR}(t) = v_1 \ldots v_k \)

Then \( g:MU(v_1) \times \ldots \times MU(v_k) \rightarrow MU(v_0) \) is a semantic function for \( t \) if:

i) \( g \) is partial recursive

ii) \( g \) is the identity function when \( t \) is in \( (VN \cup VT) \).
The semantic function of a tree \( t \) tells how to associate a meaning with the tree as a function of what are meanings associated with the elements of the frontier of \( t \). Range and domain of semantic functions are restricted by the set of meanings associated with symbols on the frontier and root of the associated tree. Note, by the way, that semantic functions may be partial and thus not associate any overall meaning with a tree for certain meanings of its constituents.

**DEFINITION 3.2.4: SEMANTICS\(_G\)**

Let \( G = (VN,VT,PR,AX) \) be a cfg. A semantics for \( G \) is \( \text{SEMANTICS}_G = (U,MU,FUN) \)

where \( U \) is a universe of discourse,
- \( MU \) is a meaning function based on \( G \) and \( U \)
and \( FUN \) is a collection of semantic functions for all trees \( t \) in \( \text{GEN}(G) \).

Two issues need to be addressed at this point. One concerns the means by which the combination of meaning and semantic functions are used to provide meanings. The second asks how the meaning and semantic functions can be conveniently specified. \( \text{SEMANTICS}_G \) requires an infinite number of semantic functions for grammars defining an infinite language. Consequently, a finite presentation of these semantic functions is required (akin to the ability of formal grammars to describe an infinite collection of sentences). We address these issues in turn.

First, it is our goal to associate meanings with syntactically well-formed sentences. To do this, we focus on the semantic functions of sentence-defining trees (those axiom-rooted complete trees). Considering the set of meanings of the terminals as arguments to the semantic functions of the tree, the set of meanings determined by the tree are all values that result from applying the semantic function to some combination of meanings of the terminals on the frontier of the tree. This idea is formalized:

**DEFINITION 3.2.5: TREE\textsc{mean}(t)**

Let \( G = (VN,VT,PR,AX) \) be a cfg with \( \text{SEMANTICS}_G = (U,MU,FUN) \)
If \( t \) is in \( \text{GENC}(G,v_0) \) with \( \text{ROOT}(t) = v_0 \) and \( \text{FR}(t) = v_1 \ldots v_k \) let

\[
\text{TREE\textsc{Mean}}(t) = \{ m \mid m = f(x_1, \ldots, x_k) \text{ where } f \text{ is } \\
\text{the semantic function of } t, \\
x_i \text{ in } \text{MU}(v_i) \text{ for } i=1, \ldots, k \}
\]

Then it follows that \( \text{MEAN}(w) = \{ m \mid m \text{ in } \text{TREE\textsc{Mean}}(t), t \in \text{GENC}(G, AX), \text{ and } \text{FR}(t) = w \} \)

The preceding definition illustrates a simple scheme for computing meanings for strings which as promised, involves the meaning of constituents and syntactic structure in computing meanings for sentences. It remains to simply define the components \( U, \text{MU}, \) and \( \text{FUN} \) for a grammar. We suggest a scheme for functional composition that is based on the syntactic structure of trees:

**DEFINITION 3.2.6: SEMFUN\(_t\)**

Suppose \( G = (\text{VN},\text{VT},\text{PR},\text{AX}) \) is a cfg with meaning function \( \text{MU} \) based on universe of discourse \( U \). Suppose \( t_0, \ldots, t_k \) are in \( (\text{GEN}(G) \cup \text{VN} \cup \text{VT}) \) with \( f_0, \ldots, f_k \) being their respective semantic functions.

If \( t = t_0[t_1 \ldots t_k] \) is in \( \text{GEN}(G) \) then

\[
\text{SEMFUN}\_t = f_0%(f_1# \ldots #f_k)
\]

As a first result, we observe that \( \text{SEMFUN}\_t \) is indeed a semantic function.

**THEOREM 3.2.1:** Suppose \( G = (\text{VN},\text{VT},\text{PR},\text{AX}) \) is a cfg with meaning function \( \text{MU} \) based on universe of discourse \( U \). Suppose \( t_0, \ldots, t_k \) are in \( \text{GEN}(G) \) with \( f_0, \ldots, f_k \) being their respective semantic functions.

Then if \( t = t_0[t_1 \ldots t_k] \) is in \( \text{GEN}(G) \) then \( \text{SEMFUN}\_t \) is a semantic function.

**PROOF:** Since \( t = t_0[t_1 \ldots t_k] \) is in \( \text{GEN}(G) \) we have \( |\text{FR}(t_0)|=k \) and \( \text{FR}(t_0,i)=\text{ROOT}(t_i) \) for \( i=1, \ldots, k \). Then

\[
f_1\# \ldots \#f_k: \text{MU}(\text{FR}(t_1,1)) \times \ldots \times \text{MU}(\text{FR}(t_k,|\text{FR}(t_k)|)) \rightarrow \\
\text{MU}(\text{ROOT}(t_1)) \times \ldots \times \text{MU}(\text{ROOT}(t_k))
\]
by the definition of $\#$ and semantic functions $f_i$, $i=1,\ldots,k$

Since $FR(t) = FR(t1)//\ldots//FR(tk)$, $ROOT(t0) = ROOT(t)$ and $FR(t0,i) = ROOT(ti)$ for $i=1,\ldots,k$ we know

$f0%(f1\ldots#fk)$ is well-defined and

$SEMFUN_t = f0%(f1\ldots#fk)$ where

$f0%(f1\ldots#fk) : MU(FR(t,1)) \times \ldots \times MU(FR(t,|FR(t)|) \rightarrow MU(ROOT(t))$

and hence $SEMFUN_t$ is a semantic function. [*]

This result suggests a method for defining semantic functions for all trees described by a grammar. It is necessary to provide semantic functions only for the productions and terminals of $G$. Semantic functions for the remainder of GEN($G$) can be generated from them.

DEFINITION 3.2.7: Language Definition (LD)

A language definition $D = (G,S)$ consists of

i) $G = (VN,VT,PR,AX)$, where $G$ is a cfg

ii) $S = (A, SF)$ where

$A$ is an alphabet

$SF = \{h_t \mid t \in PR \cup VT\}$

is a collection of partial recursive functions such that:

$h_t : A^* \rightarrow A^*$ for $t$ in $VT$

$h_t : A^* \times \ldots \times A^* \rightarrow A^*$ is a function of $k$ arguments

for $t$ in $PR$ and $|FR(t)| = k$

It is understood that a semantics $SEMANTICS_G = (U,MU,FUN)$ is implied by $S = (A, SF)$. In particular:
for $t$ in $PR$ \hspace{1cm} $SEMFUN_t$ is $h_t$

for $t$ in $GENC(G) - (VN \cup VT \cup PR)$ \hspace{1cm} $SEMFUN_t$ is found via Definition 3.2.6

for $v$ in $VT$ \hspace{1cm} $MU(v) = RAN(h_v)$

for $v$ in $VN$ \hspace{1cm} $MU(v) = U\{\text{TREE}MIN(t) \mid t \in GENC(G,v)\}$

$UNIV = U\{MU(v) \mid v \in VN \cup VT\}$ \hspace{1cm} [*]

EXAMPLE 3.2.1: Let $D = (G, S)$ with

$G = (\{Z,W\}, \{0,1\}, \{Z<W^1>, Z<1>, W, 1>, W<W^0>, W<W^1>\}, Z)$

$S = (\{a\}, SF)$ with $SF$ as follows:

$h_0(x) = \text{NULL}$ for all $x$ in $\{a\}^*$

$h_1(x) = a$

$h_{Z<W^1>} = h_{W<W^0>} = h_{W<W^1>} = f$

$h_{Z<1>} = h_{W<1>} = \text{ID}$

where $f(x_1,x_2) = x_1//x_1//x_2$ for all $x_1,x_2$ in $\{a\}^*$

and $ID(x) = x$ for all $x$ in $\{a\}^*$

This indirectly defines $U$, $MU$, and $FUN$ as follows:

$MU(0) = \{\text{NULL}\}$

$MU(1) = \{a\}$

$MU(W) = \{a, a, a, a, a, ...\}$

$MU(Z) = \{a, a, a, a, a, a, a, ...\}$

$SEMFUN_{W<1>} = \text{ID}$

$SEMFUN_{W<W^1>0} =$

$SEMFUN_{W<W^0>1} \% (SEMFUN_{W<W^1>} \# SEMFUN_0) = f%(ID\#ID)$

$SEMFUN_{Z<W<W^1>0>1} = f%(f%(ID\#ID))\#ID$
(the semantic function for this tree is thus
\[ f(f(ID\#ID))\#ID)(x_1,x_2,x_3) \]
\[ = (x_1//x_1//x_2)//(x_1//x_1//x_2)//x_3 \]
\[ = x_1^4//x_2^2//x_3 \]
for all \( x_1, x_2, x_3 \) in \( A^* \)

It is now possible to formally describe the language defined by
a particular language definition. As suggested before, it is the
set of sentence-meaning pairs where the sentence is generated by the
grammatical component of an LD, and the meaning is derived by
examining the range of the semantic functions associated with
derivations of the sentence.

**Definition 3.2.8:** \( L(D) \)

Let \( D = (G,S) \) be a language definition with \( AX \) being
the axiom of \( G \). Then

\[ L(D) = \{(w,m) | m \text{ in TREEMEAN}(t) \text{ for some } t \text{ in } GENC(G,AX) \text{ with } FR(t) = w \} \]

**Theorem 3.2.2:** Let \( D = (G,S) \) be an LD with \( AX \) the
axiom for \( G \). Then for all \( w \) in \( SEN(G) \)

\[ MEAN(w) = \{ m | m \text{ in TREEMEAN}(t) \text{ for some } t \text{ in } GENC(G,AX) \text{ with } w = FR(t) \} \]

**Proof:** This follows directly from the definitions
of \( MEAN \) and \( L(D) \) \([*]\)

**Example 3.2.2:** Consider the language definition of
the previous example.

The string 101 is in \( SEN(G) \) since 101 = \( FR(t) \) for
t = Z\\(W\\(W\\(1\\)0\\)1\\)

\[\text{SEMFUN}_t = \{x_1^4//x_2^2//x_3 \mid x_1, x_3 \text{ in } \text{MU}(1), x_2 \text{ in } \text{MU}(0) \}\]

= \{\text{aaaaaa}\}

Thus \((101,\text{aaaaaa})\) is in \(L(D)\). In fact,

\[L(D) = \{(1,a),(11,aaa),(101,aaaaa),\ldots\}\]

### 3.3. Language definition system

In order to be precise in the specification of language definitions, a means is needed for describing the functions in the semantic component. Once again there are many possible methods that could be used (e.g. Turing machines, PASCAL programs, etc.) but we will attempt to be as general as possible. To this end, a language definition system is proposed that provides the tools for defining languages over a particular syntactic alphabet, semantic alphabet, and using a specified set of semantic functions.

**DEFINITION 3.3.1 Language Definition System (LDS)**

Let \(V\) be a syntactic alphabet, \(A\) a semantic alphabet, and \(f_0, f_1, \ldots\) a listing of partial recursive functions on \(A^*\).

Then a language definition system is a listing \(D_0, D_1, \ldots\) of all language definitions of the form \(D = (G, S)\) where:

- \(G = (V_N, V_T, PR, AX)\) is a cfg with \(V_T\) in \(V\)
- \(S = (A, SF)\) with

  \[
  SF = \{f_{R(t)} \mid t \text{ in } V_T \text{ or } PR\}\]  
  \[
  R: (V_N \cup V_T) \rightarrow \text{NAT} \text{ is total}
  
  f_{R(t)}: A^* \rightarrow A^* \quad \text{for } t \text{ in } V_T
  
  f_{R(t)}: A^{*k} \rightarrow A^* \quad \text{for } t \text{ in } PR \quad |F_R(t)| = k
  
  \]

For convenience, this language definition system will be denoted by \(LDS[V,A,f]\) and the member definition denoted by
D=(G,S) with S=(A,R). Since language definition systems will be used throughout this thesis, we adopt some simple conventions for referring to the language definitions in an LDS. The notation LDS[V,A,f] will imply that the listing of functions is by subscripting: $f_0, f_1, \ldots$. Also, the listing of language definitions will imply that:

$$D_i = (G_i, S_i) \quad G_i = (V_{Ni}, V_{Ti}, PR_i, AX_i) \quad \text{and} \quad S_i = (A, R_i)$$

This numbering/subscripting will be used unless otherwise stated.

Intuitively $f_0, f_1, \ldots$ is a list of functions computed by "programs" and R is simply a specification of the program to be identified with each terminal and production. This is suggested in Figure 15. Note that $f_0, f_1, \ldots$ need not be a listing of all partial recursive functions; however, we will often choose a listing which is indeed an acceptable programming system.

Experience in computability studies suggests the importance of simplifying notation and terminology while preserving conceptual content in the development of theory. For this reason, much of computability theory focuses on one argument functions on the natural numbers and shows how results can be transferred to multiple argument functions on any domain. In this thesis, we adopt this approach as well and develop our computability results with a particular language definition system based on the natural numbers. It should be observed that this particular language definition system differs from the definition just given, but the differences are not of conceptual consequence (rather they relate to the problem of handling semantic functions of many arguments).

**DEFINITION 3.3.2: Acceptable language definition system**

Let $\langle \rangle$ be a pairing function, and $f_0, f_1, \ldots$ be an acceptable programming system and let V be a syntactic alphabet. Then an acceptable language definitions system on NAT is a listing $D_0, D_1, \ldots$ of all language definitions of the form $D=(G,S)$ where:

$$G = (V_{N}, V_{T}, PR, AX) \text{ is a cfg with } V_{T} \text{ in } V$$

$$S = (A, SF) \text{ where } A \text{ is an alphabet used to represent NAT}$$
Grammars for Languages on V

Language Definition

\[ D \]

\[ \begin{array}{c|c}
VN & f_0 \\
AX & f_1 \\
VT & f \\
PR & f \\
\end{array} \]

Figure 15: a language definition \( D \) in LDS\([V,A,f]\)
and \( SF = \{ h_t \mid t \text{ in } VT \text{ or } PR \} \)

specified by a total function

\[
R : (VN \cup VT) \rightarrow \text{NAT} \text{ so that}
\]

i) \( h_t = f_R(t) \) for \( t \) in \( VT \)

\[
i) h_t(n_1, \ldots, n_k) = f_R(t)(<n_1, \ldots, n_k>)
\]

for all \( n_1, \ldots, n_k \) in \( \text{NAT} \)

for \( t \) in \( PR, |FR(t)| = k \)

Such an LDS will be denoted by acceptable LDS\([V,\text{NAT},f]\)

and a language definition in such a system will be denoted

by \( D=(G,S) \) with \( S=(A,R) \). \([*]\)

Here the set \( A \) will not normally be explicitly given but we will

simply give meanings in a decimal notation.

Hence the semantics of an LD in such a system consists of a

list of indices ("programs") that are associated with the terminals

and productions. For a production, the indexed function is assumed

to be applied to an encoding (via the pairing function) of the

possibly multiple arguments. This encoding of multiple arguments

requires an adjustment in the way we determine semantic functions in

such language definition systems. This is suggested in the

following definition.

**DEFINITION 3.3.3: \( R^* \)**

Let \( D_0, D_1, \ldots \) be an acceptable LDS\([V,\text{NAT},f]\) where each

LD is \( D=(G,S) \) with \( G=(VN,VT,PR,AX) \) and \( S=(R) \). Then the

function \( R: (VN \cup VT) \rightarrow \text{NAT} \) is extended to \( R^*:\text{GEN}(G) \rightarrow \text{NAT} \)

so that:

i) \( R^*(t) = R(t) \) if \( t \) in \( PR \)

\[
n) f_{R^*}(t)(<n_1, \ldots, n_k>) = 
SEM\text{FUN}_t(n_1, \ldots, n_k) \text{ \ for } t \text{ in } \text{GEN}(G)-PR
\]

\[|FR(t)| = k\] \([*]\)
It will later be proven that $R^*$ can be effectively computed from $R$ in an acceptable LDS[$V, \text{NAT}, f$].

3.4. Syntax–Semantic Tradeoffs in Language Definitions

Having introduced our models for language definition and language, it is appropriate to now pause and make several observations. A language definition in an LDS[$V, A, f$] defines a language which is a subset of $V^* \times A^*$. Such languages can be either finite or infinite. Note they can be infinite due to either syntactic or semantic considerations. For instance, if the syntax generates an infinite number of sentences each of which is meaningful, then the resulting language is infinite. An infinite language might also result from an LD whose syntax generates a finite number of sentences but whose semantics assigns an infinite number of meanings to at least one of those sentences (this would be a rather unusual language!).

This interplay between syntax and semantics arises in several contexts. Given any language there may be an infinite number of ways to define it in some language definition systems. First, there are infinitely many CFGs that generate the same set of sentences. For each of these there may be (depending on the semantic function) a corresponding semantics that, when combined with the syntax, describe the same set of sentence–meaning pairs. Infinitely many LDs may also arise as a consequence of the particular listing of functions that is a part of the language definition system. For example, it is well known that the same function may be listed an
infinite number of times in an acceptable programming system. Should such a programming system be used in the semantics of an LDS, there are infinitely many ways to select an index for the same semantic function in a particular LD. In terms of the picture of Figure 15, there are infinitely many programs that compute a particular program — any of these programs could be "plugged in" an LDS. This is illustrated in Figure 16. This flexibility in defining the semantics of an LD is useful but may also be a cause for concern. On the one hand, in computability studies it is not as important since we focus on the function defined by a "program" and not on how that program works. On the other hand, in complexity research, the particular method used to compute a function is of paramount importance.

It will be useful at times to refer to the set of language definitions that contain the same context-free grammars defining the syntax of a language. The next definition formalizes this concept and the subsequent theorem confirms the multiplicity of "equivalent" language definitions in acceptable language definition systems.

**DEFINITION 3.4.1: LD_{G}**

Let D0, D1, ... be a LDS[V,A,f]. Then

\[ \text{LD}_{G} = \{ i \mid D_i = (G, S_i) \text{ for some semantic } S_i \} \]

**THEOREM 3.4.1:** Let D0, D1, ... be an acceptable LDS[V,NAT,f] where Di=(Gi, Si) for all i in NAT. Then

1) \( \text{LD}_{G_i} \) is infinite for all i in NAT

2) \( \{ j \in \text{LD}_{G_i} \mid L(D_i) = L(D_j) \} \) is infinite for all i in NAT

**PROOF:** Let D0, D1, ... be an acceptable LDS[V,NAT,f] based on acceptable programming system \( f_0, f_1, ... \).

For any specific i, we will show how to construct infinitely many Dj such that \( L(D_i) = L(D_j) \). This will prove claim 2) above and will also imply claim 1) above.
i) We note a well-known result for acceptable programming systems. For any $n$ in NAT,

$$\{ k \text{ in } \text{NAT} \mid f_k = f_n \}$$

is infinite.

ii) Then for any $G_i$, pick a $v$ in $V_T_i$. Suppose $R_i(v) = n$.

Consider a semantics $S = (R)$ defined by:

$$R(t) = R_i(t) \quad t \neq v$$

$$R(v) = k \quad \text{such that } f_k = f_n$$

From i) we see there are infinitely many $k$ that could be used in step ii) each producing a different semantics and, hence, a different language definition. Since $\text{RAN}(f_k) = \text{RAN}(f_k)$ each of these LDs defines exactly the same language as $D_i$. [*]

Choices of equivalent combinations of syntax and semantics are also significant beyond allowing multiple definitions of a language. Many combinations suggest tradeoffs between syntactic and semantic simplicity of definition. Certain languages can be defined either by a straightforward syntax combined with a complicated semantics, or by a more cumbersome syntax used with a simpler semantics. This is particularly true when the semantics is used as a "filter" to discard well-formed sentences. Recall that a sentence generated by the grammatical component of an LD is not a meaningful sentence unless the LD's semantic component associates a meaning with it. This filtering and syntax semantics tradeoff is illustrated in the next example.

EXAMPLE 3.4.1: Let $D_0, D_1, \ldots$ be acceptable LDS$[V, \text{NAT}, f]$ where ZERO, ID, $i$, and $j$ are in NAT such that:

$$f_{ID}(n_1) = n_1 \quad \text{for all } n_1, n_2 \text{ in } \text{NAT}$$

$$f_i(<n_1, n_2>) = 2^n_1 + n_2$$

$$f_{ZERO}(n_1) = 0$$

$$f_{ONE}(n_1) = 1$$

$$f_j(n_1) = n_1 \quad \text{if } n_1 \text{ is odd}$$

$$= \text{undefined} \quad \text{otherwise}$$
Consider LD $D = (G, S)$ with

$G = ([Z, W], \{0, 1\}, \{Z \prec W_1, Z \prec 1, W \prec W_1, W \prec 0, W \prec 1\}, Z)$

$S = (R)$ with

$R(0) = \text{ZERO} \quad R(1) = \text{ONE}$

$R(Z \prec W_1) = R(Z \prec W_0) = R(W \prec W_1) = 1$

$R(Z \prec 1) = R(W \prec 1) = \text{ID}$

and LD $D' = (G', S')$ with

$G' = ([Z, W], \{0, 1\}, \{Z \prec W, W \prec W_0, W \prec W_1, W \prec 0, W \prec 1\}, Z)$

$S' = (R')$ with

$R'(0) = \text{ZERO} \quad R'(1) = \text{ONE}$

$R'(Z \prec W) = j$

$R'(W \prec W_0) = R'(W \prec W_1) = 1$

$R'(W \prec 0) = R'(W \prec 1) = \text{ID}$

Note that $D'$ includes a semantics that is more involved than that of $D$ but that the grammar of $D'$ is somewhat more straightforward. We observe that

$\text{SEN}(G) = \{1, 11, 101, \ldots\} = \text{all "odd" binary strings}$

$\text{SEN}(G') = \{0, 1, 10, 11, \ldots\} = \{0, 1\}^*$

but that $L(D) = L(D') = \{(1, 1), (11, 3), (101, 5), \ldots\}$

How is it that the languages of $D$ and $D'$ are the same? To explain consider the string "10". "10" is not a sentence according to $D$ and hence cannot be a meaningful sentence. However "10" is a sentence according to $D'$, since $10 = FR(t)$ with $t = Z \prec W \prec 0 \prec 1$. Then, in $D'$

$\text{SEMFUN}_t(\langle n_1, n_2 \rangle) = f_j \% (f_4 \% (\text{ID#ID})) (\langle n_1, n_2 \rangle)$

$= 2 \% n_1 + n_2 \quad \text{if } (2 \% n_1 + n_2) \text{ is odd}$

$= \text{undefined} \quad \text{otherwise}$

Since $t$ is the only way "10" is defined in $D'$ and $\text{TREE MEAN}(t)$ is empty, "10" is given no meaning by the semantics of $D'$. All other strings in $\{0, 1\}^*$ which represent "even" numbers will be similarly meaningless; hence, we see the semantics of $D'$ selecting or filtering the sentences that have meaning.
3.5. The Languages Described by Language Definition Systems

The flexibility of language definitions suggest that our model might have extensive expressive power for describing languages. It is of interest to know just what class of languages can be described by a language definition system. In addition, given many choices for an LDS, one might ask whether there is a particular LDS that must be used for defining certain languages. A related issue in our computability studies is the question of whether the results we obtain are dependent on the choice of language definition system we make. These issues are addressed in this section, beginning with a justification of restricting our attention to language definition systems based on the natural numbers.

In discussing LDs, we have suggested that there is considerable freedom in the choice of semantic alphabets and of describing semantic functions. Experience in the study of computability has shown that examination of issues is significantly simplified if we restrict attention to functions on the natural numbers. We adopt here the same approach and will restrict our work primarily to acceptable language definitions. There are three reasons to support this approach:

1. Proofs will be greatly simplified.

2. We can more easily employ known computability results.

3. It will emphasize that presentation of semantics can be done in many equivalent ways.

We begin by noting some results from recursive function theory that can be substantiated in the literature [rog67].

THEOREM 3.5.1: Let A be an alphabet and COD_A and DEC_A be coding functions.

Then if f is a computable function of m arguments, 
\[ f: A^* \times \ldots \times A^* \rightarrow A^* \]

there is a computable function 
\[ \text{fnat}: \text{NAT} \times \ldots \times \text{NAT} \rightarrow \text{NAT} \]

of m arguments s.t.

\[ f(x_1, \ldots, x_m) = \text{DEC}_A \% \text{fnat}(\text{COD}_A(x_1), \ldots, \text{COD}_A(x_m)) \]
for all $x_i$ in $A^*$, $i=1...m$

Furthermore, if $fnat$ is a function of $m$ arguments,

$$fnat: \text{NAT} \times \ldots \times \text{NAT} \to \text{NAT},$$

then there is a function

$$f: A^* \times \ldots \times A^* \to A^*$$

of $m$ arguments s.t.

$$fnat(n_1,\ldots,n_m) = \text{COD}_A \% f(\text{DECA}(n_1),\ldots,\text{DECA}(n_2))$$

for all $ni$ in $\text{NAT}$, $i=1...m$

**PROOF:** A proof appears in [Mac78].

The first part of this proposition suggests that if $f$ is an $m$-argument function on $A^*$, then there is an $m$-argument function $fnat$ on $\text{NAT}$ that does the "analogous" mapping on coded arguments that $f$ does on the same arguments from $A^*$. The second part states the complementary notion for functions $fnat$ on $\text{NAT}$. The next theorem extends this idea to claim the existence of language definitions based on $\text{NAT}$ that are "analogous" to language definitions involving other semantic alphabets.

**THEOREM 3.5.2:** Suppose $D_0,D_1,\ldots$ is an LDS[$V,A,f$] containing a language definition $D=(G,S)$ and that $\text{COD}_A$ is a coding function.

Then there is an acceptable language definition system LDS[$V,\text{NAT},f'\]$ containing a language definition $D'$ s.t.

$$(w,m) \in L(D) \iff (w,\text{COD}_A(m)) \in L(D').$$

**PROOF:** (by induction)

Let $\text{COD}$ and $\text{DEC}$ be the coding functions $\text{COD}_A$ and $\text{DECA}$

The grammars of $D$ and $D'$ will be identical and the result actually proven states that for all $t$ in $\text{GEN}(G)$ such that $|FR(t)| = k$, if $g$ is the semantic function of $t$ according to $D$ and $g'$ is the semantic function of $t$ according to $D'$, then

$$g(x_1,\ldots,x_k) = \text{DEC} \% g'(<\text{COD}(x_1),\ldots,\text{COD}(x_k)>))$$

for $x_1,\ldots,x_k$ in $A^*$. 
Suppose $D=(G,S)$ with $G=(VN,VT,PR,AX)$ and $S=(A,R)$. We will construct $D'=(G,S')$ in LDS[$V,NAT,f'$] with $S'=(R')$ where

\[ f'_R(t)(x_1) = \text{DEC} \% f'_R(t)(\text{COD}(x_1)) \quad \text{t in } VT \]

\[ f'_R(t)(x_1,\ldots,x_k) = \text{DEC} \% f'_R(t)(\langle\text{COD}(x_1),\ldots,\text{COD}(x_k)\rangle) \]

\[ t \text{ in } PR \text{ with } |FR(t)| = k \]

for all $x_1,\ldots,x_k$ in $A^*$

To prove the claim, suppose that $t$ is in $\text{GEN}(t)$ and $g$ is the semantic function of $t$ according to $D'$. We observe:

1) if $t$ is in $PR$, then the semantic function of $t$ according to $D'$ satisfies the claim (by its definition)

2) Now suppose our claim is true for all trees in $\text{GEN}(t)$ with $HGT(t) \leq h$. Let $t=t_0[t_1\ldots t_n]$ be a tree in $\text{GEN}(t)$ with $HGT(t) > h$, $HGT(t_i) \leq h$, and $|FR(t)|=m$, $i=0\ldots n$.

According to definition, $g = g_0 \% (g_1\#\ldots\#g_n)$ where $g_i$ is the semantic function of $t_i$ according to $D$, for $i=1\ldots n$.

By hypothesis, the semantic functions $g_0',\ldots, g_n'$ of $t_0,\ldots,t_n$ respectively are, according to $D'$, such that

\[ g_i(x_1,\ldots,x_k) = \text{DEC} \% g'_i(\langle\text{COD}(x_1),\ldots,\text{COD}(x_k)\rangle) \quad k=|FR(t_i)| \]

Then $g(x_1,\ldots,x_m) = g_0 \% (g_1\#\ldots\#g_n)(x_1,\ldots,x_m)$

\[ = \text{DEC} \% g_0' \% [(\text{COD}g_1\#\ldots\#(\text{COD}g_n)](x_1,\ldots,x_m) \]

\[ = \text{DEC} \% g_0' \% \text{COD}(\text{DEC}g_1')\#\ldots\#\text{COD}(\text{DEC}g_n') \]

\[ = \text{DEC} \% g'_i(\langle\text{COD}(x_1),\ldots,\text{COD}(x_m)\rangle) \]

where $g' = g_0\%(g_1\#\ldots\#g_n')$ is the semantic function of $t$ according to $D'$.

From 1) and 2) our claim is proven by induction.

Now suppose $(w,m)$ in $L(D)$, $w = w_1\ldots w_k$. Then there must be a tree $t$ in $\text{GEN}(G,AX)$ with $FR(t)$, semantic function, and
values \( x_1, \ldots, x_k \) associated respectively with \( w_1, \ldots, w_k \) on the frontier of \( t \) such that \( g(x_1, \ldots, x_k) = m \).

For \( i=1, \ldots, k \), \( x_i \) is associated with \( w_i \) if \( x_i \) in \( \text{RAN}(f_R(w_i)) \).

However, since \( f_R(w_i) = \text{DEC} \% f'(w_i) \% \text{COD} \), it must be the case that \( \text{COD}(x_i) \) is associated with \( w_i \) according to \( D' \). Hence \( \langle \text{COD}(x_1), \ldots, \text{COD}(x_k) \rangle \) is associated with the frontier of \( t \) by \( D' \), and by our claim:

\[
m = \text{DEC} \% g'(\langle \text{COD}(x_1), \ldots, \text{COD}(x_k) \rangle) \quad \text{where} \quad g' \quad \text{is the}
\]

is the semantic function associated with \( t \) by \( D' \).

Thus \( \text{COD}(m) = g'(\langle \text{COD}(x_1), \ldots, \text{COD}(x_k) \rangle) \)

which means \( (w, \text{COD}(m)) \) in \( L(D') \).

An analogous argument would show that if \( (w, \text{COD}(m)) \) is in \( L(D') \) then \( (w, m) \) is in \( L(D) \), and this would complete the proof.

**Theorem 3.5.3**: Suppose \( D_0, D_1, \ldots \) is an acceptable \( \text{LDS}[V, \text{NAT}, f] \) containing a language definition \( D = (G, S) \) and that \( \text{DEC}_A \) is a coding function for alphabet \( A \).

Then for any programming system \( \text{LDS}[V, A, f'] \), there is a language definition \( D' \) such that

\[
(w, m) \text{ in } L(D) \quad \text{iff} \quad (w, \text{DEC}_A(m)) \text{ in } L(D').
\]

**Proof**: It would proceed analogously to the proof of Theorem 3.5.2.

As a consequence of this result we can be confident that results in this thesis related to acceptable \( \text{LDS} \) may be transferred to language definitions using semantics other than the natural numbers. The next result suggests an association between the languages described by language definitions and the class of recursively enumerable sets. Two preliminary lemmas provide tools for use in achieving the result.

**Lemma 3.5.4**: There is a recursive function \( \text{ENUM} : \text{NAT} \rightarrow \text{SEQ}(	ext{NAT}) \)

s.t. \( \text{RAN}(\text{ENUM}) \) is the set of all ordered finite sequences of pairs of natural numbers.
PROOF: Let \( \langle \rangle \) be a pairing function. Define

\[
\text{ENUM}(n) = \{(j_1,k_1),\ldots,(j_m,k_m)\} \quad \text{if} \quad \langle m,p \rangle = n \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{and} \quad \langle i,1 \rangle = 1 \\
\text{for} \quad i=1\ldots m
\]

Since the pairing function uniquely maps onto \( \text{NAT} \), this function is well-defined. \( [*] \)

**Lemma 3.5.5:** Let \( \langle \rangle \) be a pairing function and \( f_0,f_1,\ldots \) be an acceptable programming system. Then there is a recursive function \( \#\text{IND} : \text{NAT} \times \text{NAT} \times \text{NAT} \times \text{NAT} \rightarrow \text{NAT} \) defined so that

\[
f_{\#\text{IND}}(i,j,m,n)(\langle x_1,\ldots,x_m,y_1,\ldots,y_n \rangle) \\
= \langle z_1,z_2 \rangle \quad \text{if} \quad f_i(\langle x_1,\ldots,x_m \rangle) = z_1 \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{and} \quad f_j(\langle y_1,\ldots,y_n \rangle) = z_2 \\
= \text{undefined} \quad \text{if} \quad f_i(\langle x_1,\ldots,x_m \rangle) \text{ undefined} \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{or} \quad f_j(\langle y_1,\ldots,y_n \rangle) \text{ undefined}
\]

PROOF: Define

\[
g(i,j,m,n,\langle x_1,\ldots,x_m,y_1,\ldots,y_n \rangle) \\
= \langle z_1,z_2 \rangle \quad \text{if} \quad f_i(\langle x_1,\ldots,x_m \rangle) \text{ defined and} = z_1 \text{ and} \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{and} \quad f_j(\langle y_1,\ldots,y_n \rangle) \text{ defined and} = z_2 \text{ and} \\
= \text{undefined otherwise}
\]

This function is partial recursive by definition, and using the \( s-m-n \) function, it is easy to define \( \#\text{IND} \) so that

\[
f_{\#\text{IND}}(i,j,m,n)(\langle x_1,\ldots,y_n \rangle) = g(i,j,m,n,\langle x_1,\ldots,y_n \rangle) \quad [*] 
\]

Next, it is shown that there is an effective way to find the semantic function of any tree defined according to an LD in an acceptable LDS.
THEOREM 3.5.6: Let \( D_0, D_1, \ldots \) be an acceptable LDS\([V, \text{NAT}, f]\) and \( \text{ENUM} \) be an enumerating function as defined in Theorem 3.5.4. Then there is a recursive function \( \text{SEM}: \text{NAT} \times \text{NAT} \rightarrow \text{NAT} \) defined so that:

\[
 f_{\text{SEM}}(i, j) = \text{SEMFUN}_t 
\]

if \( \text{COMP}^{*}_{\text{PR}_i}(\text{ENUM}(j)) = t \) is well-defined

= the everywhere undefined function otherwise

where \( D_i = (G_i, S_i) \) and \( G_i = (V_{N_i}, V_{T_i}, \text{PR}_i, A_{X_i}) \)

PROOF: We sketch how to define \( \text{SEM}(i, j) \) for all \( i, j \) in \( \text{NAT} \).

i) Given the trees \( \text{PR}_i \), attempt to construct \( \text{COMP}^{*}_{\text{PR}_i}(\text{ENUM}(j)) = t \) from the sequence \( \text{ENUM}(j) \).

ii) If \( t \) is not well-defined, set \( \text{SEM}(i, j) \) to be the index of the everywhere undefined function. If \( t \) is well-defined an index for \( \text{SEMFUN}(t) \) can be found by repeated application of the function \( \text{COMP} \) of the definition of acceptable programming systems together with the function \( \#\text{IND} \) of Theorem 3.5.5. One would start with the indices supplied in the definition of \( S_i \). [\*]

COROLLARY 3.5.7: Let \( D_0, D_1, \ldots \) be an acceptable LDS\([V, \text{NAT}, f]\) Then for all \( i \) in \( \text{NAT} \), \( R_i^*: \text{GEN}(G_i) \rightarrow \text{NAT} \) is effectively computable.

(This result was promised in discussing \( R^* \) as defined in Definition 3.3.3).

PROOF: This follows directly from Theorem 3.5.6. [\*]

THEOREM 3.5.8: Let \( D_0, D_1, \ldots \) be an acceptable LDS\([V, \text{NAT}, f]\) and \( \text{COD}_V \) be a coding function for alphabet \( V \).

Then there is a total recursive function \( h: \text{NAT} \rightarrow \text{NAT} \) s.t. \( \text{RAN}(f_h(i)) = \{ \langle \text{COD}_V(w), m \rangle \mid (w, m) \in L(D_i) \} \).

PROOF: Let \( \text{SEM} \) be as defined in Theorem 3.5.6. Then \( H: \text{NAT} \times \text{NAT} \times \text{NAT} \rightarrow \text{NAT} \) can be defined so that:
\[ H(i, j, x) = \langle \text{COD}_V(w), m \rangle \]

if i) \( t = \text{COMP}^\text{PR}_i(\text{ENUM}(j)) \)

is well defined

ii) \( t \in \text{GENC}(G_i, AX_i) \)

iii) \( FR(t) = w = v_0 \ldots v_n \)

iv) \( \langle q_i, \ldots, q_n \rangle = x \)

v) \( q_k \in \text{RAN}(f_{R_i}(v_k)) \)

\[ k = 1 \ldots n \]

vi) \( m = f_{\text{SEM}}(i, j)(x) \)

= undefined otherwise

NOTE: This definition is quite complex. It is asking (in order):

i) does \( \text{ENUM}(j) \) define a tree in \( \text{GEN}(G_i) \)

ii) does that tree define a sentence

iii-v) does \( x \) encode a valid set of meanings for the frontier of \( t \)

vi) does the semantic function of \( t \) compute meaning \( m \) when given \( x \)

Such a function is partial recursive as a consequence of each of the steps being computable.

Using the s-m-n function, we define \( h: \text{NAT} \to \text{NAT} \) so that

\[ f_h(i)(\langle j, x \rangle) = H(i, j, x) \]

Then \( \text{RAN}(f_h(i)) = \{ \langle \text{COD}_V(w), m \rangle \mid (w, m) \in L(D_i) \} \)

since with all \( j \) and \( x \) in \( \text{NAT} \), we will eventually inquire about all trees in \( \text{GENC}(G_i, AX_i) \) and about all possible arguments to their semantic functions.

[\ast]

COROLLARY 3.5.9: Let \( D_0, D_1, \ldots \) be an an acceptable LDS across \( \text{NAT, } f \).

Then for all \( i \) in \( \text{NAT} \), \( L(D_i) \) is recursively enumerable.

PROOF: Let \( \text{COD}_V \) and \( \text{DEC}_V \) be coding functions. Observing that the function \( g: \text{NAT} \to \text{NAT} \times \text{NAT} \) defined by

\[ g(\langle \text{COD}_V(x), y \rangle) = (x, y) \]

is partial recursive, we have for all \( i \) in \( \text{NAT} \), \( \text{RAN}(g \circ f_h(i)) = L(D_i) \).

(Here \( h \) is the function of Theorem 3.5.8) Since the range of a partial recursive is recursively enumerable, we conclude
that \( L(D_i) \) is r.e. for all \( i \) in \( \text{NAT} \).

**THEOREM 3.5.10:** Let \( D_0, D_1, \ldots \) be an acceptable LDS\([V, \text{NAT}, f]\). If \( L \) is an r.e. subset of \( V^* \times \text{NAT} \), then \( L = L(D_i) \) for some \( i \) in \( \text{NAT} \).

**PROOF:** Appears in Krishnaswamy [Kri76]

**THEOREM 3.5.11:** A subset \( L \) of \( V^* \times \text{NAT} \) is recursively enumerable iff

\[
L = L(D_i) \text{ with } D_i \text{ in an acceptable LDS}[V, \text{NAT}, f]
\]

**PROOF:** From Theorems 3.5.9 and 3.5.10

Our work has identified the set of recursively enumerable languages on \( V^* \times \text{NAT} \) to be precisely the set of languages defined by an acceptable LDS\([V, \text{NAT}, f]\). The ability to transfer this result into other semantic alphabets, suggests that all and only recursively enumerable languages can be defined by language definitions fitting our model. Thus our scheme is quite powerful in its ability to define languages. It is worth observing that the method of adding semantics to syntax greatly expands the definitional capability of context free grammars. In the Chomsky hierarchy of grammars, only the "unrestricted" type has the power to describe all r.e. languages.
4. TRANSLATION AND TRANSLATOR GENERATION

4.1. Translation

In developing a theory of translation we have begun by setting down a formal model of language that notably includes provision for meaning associated with sentences. Furthermore, we have advanced a particular method of defining a language. It is capable of describing languages according to our model and moreover, of explicitly specifying the syntax and semantics of the language. Parallel to this development of a general model and a definition mechanism, we now present fundamental notions of translation. The first captures the functional nature of translation and carefully formalizes the desirable properties of translation between languages. The second idea is a more precise method of performing translation between languages for which formal language definitions are available.

**DEFINITION 4.1.1: Translation**

Let \( L_1 \), a subset of \( V_1^* \times A_1^* \), and \( L_2 \), a subset of \( V_2^* \times A_2^* \), be languages.

Then a translation \( \text{TRANS} \) is any function

\[
\text{TRANS}: \text{MSEN}(L_1) \times \text{NAT} \rightarrow \text{MSEN}(L_2)
\]

This simple definition captures the basics of translation. First, a translation is between meaningful sentences of the source language and the meaningful sentences of the target language. And second, there may be multiple sentences in the target that are translations of a particular source sentence. Hence, it is possible that \( \text{TRANS}(w,n_1) = w_1' \) and \( \text{TRANS}(w,n_2) = w_2' \) for \( w \) in \( \text{MSEN}(L_1) \) and \( n_1 \) not equal to \( n_2 \). Then \( w_1' \) and \( w_2' \) need not be the same, but both are considered translations of \( w \). In theory this definition permits a source sentence to have an infinite number of translations.
The concept of translation commonly includes other implication beyond these basic notions. For instance we may want all the target sentences that are semantically equivalent to a source sentence. Although we take the notion of translation to imply that every source sentence is translatable to some target sentence, we might permit some small number of source sentences to be without a translation. These desirable properties are made precise in the following definition:

DEFINITION 4.1.2: Translation properties

Let TRANS: MSEN(L1) X NAT -> MSEN(L2) be a translation from L1 (in V1* X A1*) to L2 (in V2* X A2*).

TRANS is semantic preserving
   if for all (w,m) in DOM(TRANS)
       MEAN(w) \cap MEAN(TRANS(w,n)) is nonempty

TRANS is complete if for all w1 in MSEN(L1)
   and all w2 in MSEN(L2) such that
   MEAN(w1) \cap MEAN(w2) is nonempty
   that there is n in NAT where TRANS(w1,n) = w2

TRANS is total if for all w in HSEN(L1) there
   is an n in NAT such that (w,n) in DOM(TRANS).

TRANS is almost everywhere defined (denoted "a.e.")
   if there is a k in NAT so that for all w in MSEN(L1)
   with |w| > k, there is an n in NAT such that
   (w,n) in DOM(TRANS) [*]

A translation is semantic preserving when it maps a source sentence to a target sentence that has at least one (but not necessarily all) meaning in common with the source. Complete translations find all target sentences with meaning in common with the source sentence. A total translation gives a translation (again not necessarily all translations) of any meaningful sentence. Almost everywhere translations provide for all but a finite number of meaningful sentences in the source (according to the definition, the only meaningful sentences not translated would be of length k or less). Examples of translations with these properties are given in Example 4.1.1. Note that the properties of being semantic preserving, complete, and total (or almost everywhere) are independent - one can apply while the others do not. Of course, a total translation is necessarily a.e., although the converse is not true.
EXAMPLE 4.1.1: Suppose $L_1$ and $L_2$ are languages on 
\{a,b\} and \{c\} with 

$L_1 = \{(a^k, c^k) \mid k \in \text{NAT}\}$

$L_2 = \{(b^k, c^j) \mid j \leq k, k \in \text{NAT}\}$

Consider translations $g_i: \{a\}^* \times \text{NAT} \to \{b\}^*$ for $i=1,..,7$

- $g_1(a^{k+1}, 1) = b^k$ if $k$ is odd, undefined otherwise
- $g_2(a^k, 1) = b^k$ if $k$ is odd, undefined otherwise
- $g_3(a^k, n) = b^n$ if $k$ is odd and $n \in \text{NAT}$, undefined otherwise
- $g_4(a^k, 1) = b^{k+1}$ if $k$ is in NAT, undefined otherwise
- $g_5(a^{k+1}, 1) = b^k$ if $k$ is in NAT, undefined otherwise
- $g_6(a^k, 1) = b^n$ if $k$ is in NAT and $n > k$, undefined otherwise
- $g_7(a^k, 1) = b^k$ if $\text{DOM}(f_k)$ is finite, undefined otherwise

where $f_0, f_1, \ldots$ is an acceptable programming system.

Then we can summarize the properties of these translations as follows:

- Semantic Preserving: $g_2$, $g_6$, $g_7$
- Complete: $g_3$, $g_6$
- Total: $g_4$, $g_6$
- Almost Everywhere: $g_4$, $g_5$, $g_6$
- None of the Above: $g_1$

$g_7$ is semantic preserving but not computable.
The definition of semantic preserving may at first seem peculiar. It might be more natural to suggest that the meanings of \( w_1 \) be identical to the meanings of \( w_2 \). For unambiguous languages this is indeed a consequence of being semantic preserving. However, for ambiguous languages (e.g. natural language) we usually do not require a translation to be identical in all its interpretations to the meaning or meanings of the source sentence. Thus here we require that a translation have only one common meaning.

It should be obvious that desirable translations — ones that are semantic preserving as well as at least almost everywhere — should not necessarily exist between any pair of languages. One does not anticipate, for example, that English will translate into FORTRAN. In example 4.1.2 we see a demonstration of languages between which there is no translation. In the theorem that follows these observations are applied to our formal model of language.

**EXAMPLE 4.1.2:** Consider the following languages \( L_i \) on \([a]\) and \([b]\)

\[
L_1 = \{(a^k, b^{2k}) \mid k \in \text{NAT} \}
\]

\[
L_2 = \{(a^k, b^{2k+1}) \mid k \in \text{NAT} \}
\]

\[
L_3 = \{(a^k, b^{2k}) \mid k \in \text{NAT, } k \text{ divisible by 5} \}
\]

Since all the languages are unambiguous, any semantic preserving translation will be complete. We observe that a total, semantic preserving translation is possible only between \( L_3 \) and \( L_1 \). A semantic preserving translation (although not almost everywhere or total) translation is possible between \( L_1 \) and \( L_3 \). No semantic preserving translation involving \( L_2 \) as source or target is possible since \( \text{MEAN}(L_2) \) is disjoint from the meanings of \( L_1 \) and \( L_3 \).

**THEOREM 4.1.1:** There are languages between which there is no semantic preserving translation. There are languages between which there is no almost everywhere (and hence total) translation.

**PROOF:** See example 4.1.2.

[•]
THEOREM 4.1.2: Let \( D_0, D_1, \ldots \) be an acceptable \( \text{LDS}[V, \text{NAT}, \gamma] \). Then there are indices \( i \) and \( j \) such that no semantic preserving translation exists between \( L(D_i) \) and \( L(D_j) \). There are also indices \( k \) and \( q \) such that no almost everywhere (and hence no total) translation exists between \( L(D_k) \) and \( L(D_q) \).

PROOF: Since \( \{b\}^* \) can serve as an alphabet to represent \( \text{NAT} \), the LD's of Example 4.1.2 surely will be found in any acceptable language definition system (consult Theorem 3.5.10) \([\star]\)

4.2. Table Translation

Much of the theoretical work on translation has featured so called "syntax-directed" translation schemes. These approaches transform source sentences to target sentences by transforming a parse tree of the source sentence into a representation of a target sentence. Based on parsing schemes and manipulating the derivation of a source sentence, such methods are purely syntactic. Many methods use a table that matches source language syntactic entities with corresponding target language syntactic entities. Perhaps a production of a source grammar is matched with a production of the target grammar. The actual translation process is driven by looking up corresponding source-target entries in the table. In such a case, the "meanings" of the involved languages are only utilized when the table is created and not used at the time of translation.

The benefits of this approach are essentially the efficiency and speed with which such a process may operate. Compiling of programming languages requires such speed and syntax-directed compilation has been very important in implementing programming languages. On the other hand, when translation is purely syntax directed, there are inherent theoretical limitations in the class of languages that can be translated. Pyster [Pys75] investigated these limitations and as well suggested a variety of translation schemes that actively used semantics at the time of translation. This addition of semantic processing increased the power of translators but also increased the work in translating. Semantic processing is complicated and reduces the efficiency of the translation process.

In his work, Krishnaswamy [Kri76] chose to look at a model of translation that combined semantic with syntactic processing but restricted the use of semantics. By isolating the use of meaning,
the actual source-to-target transformation could remain purely syntactic and thus capitalize on its efficiency. However, by retaining some semantic processing he maintained the power of this translation scheme to translate a wide variety of languages.

In the same spirit, this thesis examines a particular model of translation that is schematically presented in Figure 10. Semantic processing is restricted to examination of the meanings associated with a parse tree of a source sentence. Sentences without meaningful parses will be discarded and not further processed. Meaningful parses will be permitted to pass on to a table driven tree transducer that will produce a parse tree of the target language. The frontier of this parse tree is then "stripped off" as the target sentence translation.

Actually the method will be a bit more complex due to the possibility that each step may permit several valid intermediate results. This will be explained in the precise development of this table translation method (called so since the heart of the transformation is table driven tree transduction). We begin by defining the type of translation tables and tree transduction used in this thesis.

**DEFINITION 4.2.1: Translation table TAU**

Let $G_1 = (VN_1, VT_1, PR_1, AX_1)$ and $G_2 = (VN_2, VT_2, PR_2, AX_2)$ be cfgs. Then a translation table is a triple $(T_1, T_2, TAU)$ where:

1) $T_1$ is a finite subset of $GEN(G_1)$

2) $T_2$ is a finite subset of $(GEN(G_2) \cup VN_2 \cup VT_2)$

3) $TAU: T_1 \times NAT \rightarrow T_2 \times SEQ(NAT)$ is defined so that

   A) if $TAU(t) = (t', (x_1, \ldots, x_k))$ then
   $|FR(t)| = k$ and $0 \leq x_i \leq |FR(t)|$ for $i = 1, \ldots, k$ \[[^*]\]

   B) there is an $n$ in NAT such that for all $j > n$, and for all $t$ in $T_1$, $TAU(t, j)$ is undefined.

We will normally refer to a translation table $(T_1, T_2, TAU)$ as translation table $TAU$ where it is understood that $TAU$, its domain, and range are all finitely specified.
Condition B) insures that there are at most a finite number of target trees associated with each source tree in T1. Such a finite function may be viewed as a table that associates with each source tree t in T1 a sequence of pairs consisting of a target tree in T2 together with an index vector. The index vector will specify how target trees are to be attached to the target tree entry in the tree transduction process. This is made clear in the description of tree transduction that follows. The method described proceeds in "bottom up" fashion. Essentially, a target entry in the table is systematically substituted for the corresponding source entry with a rearrangement of trees hanging from the frontier (the rearrangement is dictated by the index vector). The result is a step-by-step conversion of a source tree into a target tree. Some of the earlier translation models used only the productions of source and target grammars as possible entries in their tables. Our approach permits compound trees (of height more than one) to be in the table. An example of a translation table appears in Example 4.2.1.

**DEFINITION 4.2.2: Tree transduction $\text{T}_{\text{AU}}^*$**

Let $\text{T}_{\text{AU}}: T_1 \rightarrow T_2 \times \text{SEQ(NAT)}$ be a translation table as suggested in Definition 4.2.1. Then $\text{T}_{\text{AU}}$ is extended to a mapping $\text{T}_{\text{AU}}^*$ of trees to trees by:

$\text{T}_{\text{AU}}^*: \text{GEN}(T_1) \rightarrow \text{POWERSET}(\text{GEN}(T_2))$ where

1) $\text{T}_{\text{AU}}^*(t)$ includes $t'$ if $t$ in $T_1$ and for some $j$

   $\text{T}_{\text{AU}}(t,j) = (t',(x_1,\ldots,x_k))$

2) $\text{T}_{\text{AU}}^*(t)$ includes $t'=t_0'[t_1'\ldots t_k']$ if $t=t_0[t_1\ldots t_m]$ in $\text{GEN}(T_1)$

   i) $t_0$ in $T_1$ with a $j$ s.t.

   $\text{T}_{\text{AU}}(t_0,j)=(t_0',(x_1\ldots x_k))$

   ii) $t_0$ in $T_1$ with a $j$ s.t.

   $\text{T}_{\text{AU}}(t_0,j)=(t_0',(x_1\ldots x_k))$

   iii) $t_i' = \text{FR}(t_0',i)$ if $x_i = 0$ or $\text{HGT}(t_{x_i})=0$

   for $i = 1,\ldots,k$

   iv) $t_i'$ is in $\text{T}_{\text{AU}}^*(t_{x_i})$

   if $x_i > 0$ and $\text{HGT}(t_{x_i})>0$

   for $i = 1,\ldots,k$

   v) $t_0'[t_1'\ldots t_k']$ is well-defined
NOTE:

Condition i) selects a particular supertree \( t_0 \) of \( t \), while condition ii) selects a particular TAU entry for \( t_0 \) (including a tree substitute \( t_0' \) from GEN(T2). In iii) and iv) the index vector is used to arrange subtrees on the substitution tree. When \( x_i \) is zero, nothing "hangs" from position \( i \) on the frontier of \( t_0' \). When \( x_i \) is greater than 0, a tree transduction of the tree hanging at position \( x_i \) on the frontier of \( t_0 \) is "grafted" on at position \( i \) of \( t_0' \) (this is not done if there is no tree hanging at position \( x_i \) of \( t_0 \)). Condition v) ensures the resulting tree must be a well-defined tree of GEN(T2).

Observe that a tree transduction may associate multiple target trees with a single tree \( t \) in GEN(T1). First, there may be multiple box-bracketings of \( t \), each guiding a different transduction. Moreover, TAU may provide a choice of substitutions for each subtree of \( t \). In order to account for these choices of bracketing and substitutions, we provide an alternate definition of TAU* (with a different domain and range) that we will use in our work. The context of any reference to TAU* in this thesis should identify which version of transduction is being used.

DEFINITION 4.2.3: TAU* (a second definition)

Let \( G_1 \) and \( G_2 = (VN_2, VT_2, PR_2, AX_2) \) be cfgs, \( T_1 = \{s_1, ..., s_m\} \) is a finite subset of GEN(T1) and \( T_2 \) a finite subset of GEN(G2) \( \cup VN_2 \cup VT_2 \).

Suppose

TAU: \( T_1 \times \text{NAT} \to T_2 \times \text{SEQ}(	ext{NAT}) \) is a translation table.

Then TAU*: \( \text{GEN}(T_1) \times \text{NAT} \to \text{GEN}(T_2) \) is defined by:

1) \( \text{TAU}^*(t, j) = t' \) if \( t = s_z \text{ in } T_1, j = (z, y) \) and \( \text{TAU}(t, y) = (t', (x_1, ..., x_k)) \)

2) \( \text{TAU}^*(t, j) = t_0'[t_1'...t_k'] \) if
   i) \( t = t_0[t_1...t_n] \text{ in } \text{GEN}(T_1) - T_1 \)
   ii) \( t_0 = s_z \text{ in } T_1, |\text{FR}(t_0)| = n \)
iii) \( j = \langle z, m_0, j' \rangle \)

iv) \( \text{TAU}(t_0, m_0) = (t_0', (x_1 \ldots x_k)) \) is defined

v) \( j' = \langle m_1, \ldots, m_k \rangle \)

vi) \( t_i' = \text{FR}(t_0', i) \) and \( m_i = 0 \) if
\( x_i = 0 \) or \( \text{HGT}(t_{x_i}) = 0 \)

for \( i = 1, \ldots, k \)

vii) \( t_i' = \text{TAU}^*(t_{x_i}, m_i) \) if
\( x_i > 0 \) and \( \text{HGT}(t_{x_i}) > 0 \)

for \( i = 1, \ldots, k \)

viii) \( t_0'[t_1' \ldots t_k'] \) is well-defined

If the conditions i) through viii) are not satisfied
\( \text{TAU}^*(t, j) \) is undefined. [*]

As a consequence of this cumbersome definition, we can (for
each combination of a valid box-bracketing of a tree \( t \) in \( \text{GEN}(T_1) \)
with a set of valid TAU selections for substitution) find a tree in
\( \text{GEN}(T_2) \). If this tree is well-defined, it is a proper transduction
of the source tree. The reader should observe that this function
\( \text{TAU}^* \) is always partial (e.g., since there will be an infinite number
of \( y \) for which \( \text{TAU}(t, y) \) is undefined). Also, the two definitions of
\( \text{TAU}^* \) are related in the following sense:

**THEOREM 4.2.1:** Let \( \text{TAU}^*:\text{GEN}(T_1) \rightarrow \text{POWERSET}(\text{GEN}(T_2)) \)
be the \( \text{TAU}^* \) of Definition 4.2.2 and
\( \text{TAU}^2*:\text{GEN}(T_1) \times \text{NAT} \rightarrow \text{GEN}(T_2) \) be the \( \text{TAU}^* \) of
Definition 4.2.3, both based on translation table
\( \text{TAU}: T_1 \times \text{NAT} \rightarrow T_2 \) for cfgs \( G_1 \) and \( G_2 \).

Then \( t' \) in \( \text{TAU}^1(t) =\Rightarrow \)
there is a \( j \) in \( \text{NAT} \) s.t. \( \text{TAU}^2(t, j) = t' \)

And \( \text{TAU}^2(t, j) = t' =\Rightarrow t' \) is in \( \text{TAU}^1(t) \).

**PROOF:** Informally, we simply have to pick the particular
\( j \) that encodes the combination of box-bracketing for \( t \)
and the particular set of substitutions of trees in \( T_2 \)
that are used in the transduction that associates a \( t' \)
in the target with a tree \( t \) in the source. [*]
As a first point, it should be emphasized that it is possible that for some trees in GEN(T1) there will be no tree transduction based on the particular translation table. This is possible because transductions may call for a tree composition which is not well-defined in the target syntax. This possibility as well as the basics of transduction are exhibited in the following example.

**EXAMPLE 4.2.1:** Consider any cfgs G1 and G2 that could include $T_1 = \{Z<Za>, Z<Z<Za>a>, Z<a>\}$ as a subset of GEN(G1) and $T_2 = \{W<bW>, W<bW<bW>>, X<b>\}$ as a subset of GEN(G2).

A possible translation table is

- $\text{TAU}(Z<Za>) = (W<bW>, (0,1))$
- $\text{TAU}(Z<a>) = (X<b>, (0))$
- $\text{TAU}(Z<Z<Za>a>) = (W<bW>, (0,1))$

In that case $\text{TAU}^*(Z<a>) = \{X<b>\}$
- $\text{TAU}^*(Z<Za>) = \{W<bW>\}$
- $\text{TAU}^*(Z<Z<Za>a>) = \{W<bW>, W<bW<bW>>\}$

One tree in the last item comes directly from TAU; the other is explained by the transduction below:

- $t_0 = Z<Za>$
- $t_1 = Z<Za>$
- $t_2 = a$

Then $Z<Z<Za>a> = t_0[t_1 t_2]$ and so

$\text{TAU}^*(t_0[t_1 t_2])$ includes

$W<bW>[b t_1']$ where $t_1'$ in $\text{TAU}^*(t_1)$ since $\text{TAU}(Z<Za>) = (W<bW>, (0,1))$

Which leads to $W<bW>[b W<bW>]$ in $\text{TAU}^*(t_0[t_1 t_2])$

However, $\text{TAU}^*(Z<Z<a>a>)$ is empty since the transduction called for asks $X<b>$ to be composed at position two of the frontier of $W<bW>$ (but this composition is not well-defined since $X$ is not equal to $W$)

Having developed a tree transduction mechanism that transforms trees based on one grammar to trees based on a second grammar, it
remains to be seen how this mechanism can be used to specify translation. First of all, we want to see how tree translation can be applied in the context of a source-target pair of formally specified language definitions. Then we need to suggest a mapping of source sentences to target sentences that is based on the transduction's mapping of source trees to target trees. Associating translation tables with formal language definitions is a simple matter. Each "entry" in a translation table will henceforth consist of a source tree (described by the grammar of the source LD) together with a target tree (of the grammar of a target LD) and an appropriate index vector. Such trees need not be production trees, but may be compound trees.

As a first step to specifying translations between languages described by formal LDs, we propose a mapping of source language sentences to strings related to the grammar of a target LD. This mapping is notable in that:

1. It is based in part on tree transduction driven by a translation table.
2. It also employs semantic processing (made possible by the presence of a formal description of semantics in the LDs).

In fact, the range of the mapping will consist of strings of target terminals and nonterminals. The semantic processing only involves the source language and checks to see if a parse tree "has meaning". Should the particular tree have a meaningful interpretation, the tree is passed on to the transduction process which may produce a target tree. The string at the frontier of this target tree is then used as the result. These ideas are formalized Definition 4.2.4.

**DEFINITION 4.2.4: TAUBAR**

Let $D_1$ and $D_2$ be language definitions in $\text{LDS}[V,A,f]$ with $D_i = (G_i,S_i)$ and $G_i=(V_{N_i},V_{T_i},P_{R_i},A_{X_i})$, $S_i=(A,R_i)$ $i=1,2$

Let $\text{COD}$ be the coding function $\text{COD}$, and $\text{ENUM}: \text{NAT} \rightarrow \text{NAT} \times \text{NAT}$ as in Theorem 3.5.8.

Let $T_1$ be a finite subset of $\text{GEN}(G_1)$ and $T_2$ be a finite subset of $\text{GEN}(G_2) \cup V_{N_2} \cup V_{T_2}$, and $\text{TAU}: T_1 \times \text{NAT} \rightarrow T_2 \times \text{SEQ}(\text{NAT})$ be a translation table.

Then $\text{TAUBAR}: \text{MSEN}(L(D_1)) \times \text{NAT} \rightarrow (V_{N_2} \cup V_{T_2})^*$ is defined by $\text{TAUBAR}(w_1,n) = w_2$ if...
i) \( n = \langle j, k, q \rangle \)

ii) \( t = COMP^{*}_{T1}(ENUM(j)) \) is in GENC(G1,AX1)

iii) \( FR(t) = w_1 \quad |FR(t)| = z \)

iv) \( \langle COD(x_1), \ldots, COD(x_z) \rangle = k \quad x_i \) in \( A^* \quad i=1, \ldots, z \)

v) \( \text{SEMFUN}_t(x_1, \ldots, x_z) \) based on \( ENUM(j) \) defined

vi) \( t' = TAU^*(t,q) \)

vii) \( FR(t') = w_2 \)

viii) \( t' \) in GENC(G2,AX2)

TAUBAR(wl,n) is undefined should any of conditions i) through viii) not hold.

NOTE:

TAUBAR requires a sequence of checks to assure that it will be defined for argument \( (wl,\langle j,k,q \rangle) \). First, \( ENUM(j) \) must specify a well-defined complete tree \( t \), rooted in the axiom (condition ii). Then the frontier of \( t \) must be \( w_1 \) (condition iii). Number \( k \) must encode a set of arguments for which the semantic function of \( t \) is defined (conditions iv, v). Then the tree \( t \) is transduced according to \( q \) and the frontier of the resulting tree is "stripped off" as the value of TAUBAR. Hence for argument \( (wl,\langle j,k,q \rangle) \) to "pass through" the TAUBAR computation we must have (see Figure 17):

1. \( ENUM(j) \) specifying a parsing tree in GENC(T1) for sentence \( wl \).

2. \( k \) must encode a set of arguments for which the semantic function of the parse tree is defined.

3. \( q \) is the appropriate encoding of structure and substitutions based on TAU that will specify an appropriate transduction of the parse tree.

It is clear that TAUBAR is always a partial function (e.g. there are many \( j \) for which \( ENUM(j) \) would not specify a parse tree of \( wl \), the semantic function may be undefined for many arguments, and the transduction of \( t \) is defined only for a unique \( q \)).
Figure 17: The structure of TAUBAR
THEOREM 4.2.2: Let $D_0, D_1, \ldots$ be an acceptable LDS$[V, \text{NAT}, f]$. Then there is a total recursive function $\text{INDTAUBAR}$

1) with domain the set of all possible translation tables between grammars of LDS$[V, \text{NAT}, f]$

2) with range being NAT

3) $f_{\text{INDTAUBAR}}(\text{TAUBAR}(<\text{COD}_V(w), n>) = \text{COD}_V(w')$

iff

$\text{TAUBAR}(w, n) = w'$

where $\text{TAUBAR}$ is the string mapping based on $\text{TAU}$.

PROOF: This is a simple use of the s-m-n function in conjunction with the construction of $\text{TAUBAR}$ given in Definition 4.2.4. [*]

This result suggests that $\text{TAUBAR}$ is a function that can be effectively computed from $\text{TAU}$. It is the theoretical justification for our table translation framework - namely that we can "plug in" a translation table $\text{TAU}$ into a general translation mechanism to drive an actual translation.

We can now formalize the idea of table translation that will be used throughout the remainder of this thesis. We presented the basic framework in Figure 10, and we now make precise the combination of this framework with the table translation mechanism developed in this chapter.

DEFINITION 4.2.5: Table Translation

Let $D_0, D_1, \ldots$ be LDS$[V, A, f]$ and $\text{TAU}$ be a translation table between the grammars of language definitions $D_i$ and $D_j$.

Then $\text{TAUBAR}$ (based on $\text{TAU}$) is a table translation between $D_i$ and $D_j$ if $\text{RAN}(\text{TAUBAR})$ is a subset of $\text{MSEN}(L(D_j))$.

We often refer to this as the translation induced by translation table $\text{TAU}$ and say that $\text{TAU}$ is a table
translation between \( D_1 \) and \( D_2 \).

The next example illustrates the nature of table translation.

**EXAMPLE 4.2.2:** Consider LDS\([0,1,a],\text{NAT},f\) with LDs \( D_1 \) and \( D_2 \):

\[ D_1 = (G_1, S_1) \text{ with } G_1 = ([Z,W],[0,1],[Z<W>,W<W1>,W<W0>,W<1>],Z) \]

and \( S_1 = (R_1) \) with \( \text{RAN}(f_{R_1}(0)) = \{0\} \)

\[ \text{RAN}(f_{R_1}(1)) = \{1\} \]

\[ f_{R_1}(Z<W>)(x_1) = x_1 \text{ if } x_1 = 2^k - 1 \text{ for some } k \text{ in } \text{NAT} \]

\[ f_{R_1}(Z<W>)(x_1) = \text{undefined otherwise} \]

\[ f_{R_1}(W<W0>)(x_1,x_2) = 2^*x_1 + x_2 \]

\[ f_{R_1}(W<W1>)(x_1,x_2) = 2^*x_1 + x_2 \]

\[ f_{R_1}(W<1>)(x_1) = x_1 \text{ for all } x,x_1,x_2 \text{ in } \text{NAT} \]

\[ D_2 = (G_2, S_2) \text{ with } G_2 = ([Y],[a],[Y<Ya>,Y<a>],Y) \]

and \( S_2 = (R_2) \) with \( \text{RAN}(f_{R_2}(a)) = \{0\} \)

\[ f_{R_2}(Y<a>)(x_1) = x_1 \]

\[ f_{R_2}(Y<Ya>)(x_1,x_2) = 2^*x_1 + 1 \]

Then \( L(D_1) = \{(1,1),(11,3),(111,7),\ldots\} \) and

\( L(D_2) = \{(a,0),(aa,1),(aaa,3),\ldots\} \)

It should be clear that a total, semantic preserving translation exists from \( L(D_1) \) to \( L(D_2) \). Now consider two possible translation tables that induce that translation:

i) \( \text{TAU1 with } \)

\[ \text{TAU1}(Z<W>,1) = (Y, (1)) \]

\[ \text{TAU1}(W<W1>,1) = (Y<Ya>, (1,0)) \]

\[ \text{TAU1}(W<W0>,1) = (Y, (0)) \]

\[ \text{TAU1}(W<1>,1) = (Y<Y<a>),(0,0)) \]
TAU1(W<WO>,1) = (Y, (0,0))
TAU1(W<1>,1) = (Y<Y<a>a>, (0,0))

ii) TAU2 with
TAU2(Z<W>,1) = (Y, (1))
TAU2(W<W1>,1) = (Y<Y<Ya>a>, (1,0))
TAU2(W<1>,1) = (Y<Y<a>a>, (0,0))

It is informative to examine the TAUBAR mappings for these strings "10" and "11".

10: Under TAU1, there is a unique parse
t = Z<Y>W<W0>W<W1>W<1>0 for 10. However, the corresponding tree function SEMFUN(t) is undefined for the arguments (0,0) and so t will not be transduced. Under TAU2, 10 has no parse.

11: Under both TAU1 and TAU2 there is a unique parse
t = Z<Y>W<W1>W<1>1 for 11. For argument (1,1) the semantic functions are defined. The steps in the transduction would successively produce trees
Y Y<Y<Ya>a> Y<Y<Ya>a> Y<Y<Ya>a> a a

This suggests how semantic filtering can be used in TAU2 and how it is avoided in TAU2. One may observe that by the construction of TAU1 and TAU2 (and noting that each sentence in L(D1) has a unique parse) that for all k = 1, 2, ...

1^k in SEN(G1) will be uniquely mapped by TAUBAR to a^{k+1}. This mapping is clearly a total translation.

Moreover, since MEAN(1^{k+1}) = (2^k - 1) according to D2, the translation induced by TAU1 (or TAU2) is also semantic preserving.

In the next example we demonstrate that it is possible that a table translation may not exist between language definitions even though a translation may exist between the languages described by those definitions.
EXAMPLE 4.2.3: Consider language definitions D and D' from an acceptable LDS[V,NAT,f].

D = (G,S) with G = ([Z,S],[0],{Z<S>,S<Sb>,S<b>}),Z)

and S = (R) where R is defined:

\[ f_R(b)(x_1) = 0 \]
\[ f_R(Z<S>)(x_1) = f_R(S<b>)(x_1) = x_1 \]
\[ f_R(S<Sb>)(<x_1,x_2>) = x_1 + 1 \]

for all \( x_1, x_2 \) in NAT

D' = (G',S')

with G' = ([Z,Y],[0,1],[Z<Y>,Y<Y0>,Y<Y1>,Y<1>,Z<0>]),Z)

and S' = (R') where R' is defined:

\[ f_{R'}(0)(x_1) = 0 \]
\[ f_{R'}(1)(x_1) = 1 \]
\[ f_{R'}(Z<Y>)(x_1) = f_{R'}(Z<0>) = x_1 \]
\[ f_{R'}(Y<1>)(x_1) = x_1 \]
\[ f_{R'}(Y<Y0>)(x_1,x_2) = 2^*x_1 + x_2 \]
\[ f_{R'}(Y<Y1>)(x_1,x_2) = 2^*x_1 + x_2 \]

for all \( x_1, x_2 \) in NAT

Notice that \( L(D) = \{ (b^j+1,j) \mid j \text{ in NAT} \} \) is unambiguous and

that \( L(D') = \{ (w,m) \mid w \text{ is in } ([0] \cup \{0,1\}^+) \text{ and } m \text{ is the "value" of } w \text{ as a binary number} \} \)

It should also be clear that there is a total, semantic preserving translation between \( L(D) \) and \( L(D') \) since each \( b^j+1 \) can be translated to that unique \( w \) in \( \text{SEN}(G') \) that is the binary representation of \( j \).

Now suppose that there is a table translation \( \text{TzA} \) between D and D' inducing a total, semantic-preserving translation. Since the table is finite, the language infinite, and the grammar G is so simple, it must be the case that there is at least one entry in
the translation table containing a source tree of the form \( tm \) (\( m>0 \)) seen in Figure 18. Since there is a single parse tree for each sentence in the source, and each tree has a unique meaning, it must be the case that \( tm \) is associated in TAU with a tree \( tn' \) (\( n>0 \)) of the form seen in Figure 18. (Note that these trees are of the only form that permit a sequence of compositions). Let \( FR(tm) = Yw \) where \( w \) is a string on \((0,1)^+\) of length \( n \) with the binary value of \( w \) being \( K \).

Then the semantic function of \( tm \) is \( f(x_0,x_1,\ldots,x_m) = x_0 + m \).

and the semantic function of \( tn' \) is \( f'(x_0,x_1,\ldots,x_n) = 2^n x_0 + K \).

Because these functions are so disimilar, it is in fact impossible to form a translation table with such an entry associating \( tm \) and \( tn' \). This is illustrated in Figure 18 by a particular example (this example is representative of the problem but need not be present in any specific case).

Suppose that \( t_A' \) was associated with \( t_A \) by the table TAU. Then TAU must also associate \( t_B' \) with \( t_B \) and \( t_C' \) with \( t_C \).

Examining the meanings associated with these trees and recalling TAU was to induce a semantic-preserving translation, we would have:

\[
m = 2^n + K
\]

\[
2m = (2^n + K) \times 2^n + K
\]

and \[
3m = ((2^n + K) \times 2^n + K) \times 2^n + K
\]

These three equations are not mutually satisfiable and hence the existence of a translation table associating \( tn' \) with \( tm \) is not possible. Thus there can be no finite length table inducing both a total and semantic-preserving translation between \( D \) and \( D' \). Note that the same argument could be used to assert there was no translation table inducing an almost-everywhere, semantic-preserving translation. Also, observe that for any parse tree of a meaningful source sentence, there is a parse tree of a meaningful target sentence that shares a meaning with it. The problem is that an infinite number of such pairings cannot be induced by a translation table of finite length.
Figure 18: Trees for Example 4.2.3
4.3. Translator Generation Problems

The major thrust of this work is to study the generation of translations within a theoretical framework. Based on previous development of translation theory, we have suggested formal models for language, language definition, translation, and table translation. In the remainder of this thesis we investigate the existence and discovery of either translations or table translations between languages (including those described by formal language definitions). Our interest is in finding general procedures that can analyze language definitions or languages and then determine whether a translation (possibly a table translation) exists between them. Perhaps these procedures can actually construct the translation if it does exist.

Informally, we will for now refer to these as the translator generation decision problem (TG decision problem) and the translator generation problem (TG problem). We will formally state these questions in the course of this thesis. The decision problem asks for a "yes" or "no" answer - given a source-target pair of languages (perhaps specified by formal language definitions) determine if there is any translation (perhaps table translation) between languages. The computation problem is more complicated - given a source-target pair of languages (perhaps specified by formal language definitions) construct a translation (perhaps a translation table) if there is one; otherwise, indicate no translation exists. It is intended that any translator will suffice (for there may be multiple translators or translation tables capable of inducing a translation).

The translator generation problem and the translator generation decision problems are clearly related. From a computational point of view the answer to the TG problem indirectly answers the TG decision problem - if a translator is generated then one exists; if it doesn't exist this the TG problem solution must indicate so. From a complexity point of view it is intuitively clear that the work done in a solution to the TG problem should not be less than the work done in solving the TG decision problem (again because answers to the TG problem usually answer the TG decision problem).

This thesis examines the TG and TG decision problems in many versions with the common goal of locating algorithmic solutions to the problems. That is, we seek a general procedure that will solve the problem for all its individual instances. In some cases, we will be unable to find such procedures. In those problems with procedural solutions, we endeavor to study the complexity of such procedures.
5. COMPUTABILITY OF TRANSLATOR GENERATION

5.1. Computable Translations

We begin our computability study by examining the notion of translation that we introduced in Chapter 4. We recall that our definition tells us that a translation is a function that maps a source sentence to possibly multiple target sentences. Several natural qualifications on this general notion were suggested. Surely we would want the translations of a source sentence to share a meaning with that source sentence. To be useful, we would want a translation to translate all meaningful source sentences, or at least all but a finite number of them. And we might want complete translations — ones that give all possible target sentences that could be a translation of a particular source sentence. Perhaps most fundamental of all we would want a translation to be computable. Intuitively, we take that to mean that we could describe procedures that would perform the translation.

The first result we present suggests that between any pair of recursively enumerable languages there is a computable translation that is both semantic preserving and complete. Such a translation may not be a useful one, in that it may not translate an infinite number of source sentences. It in fact may be the empty function if there are absolutely no sentences in the source and target languages that share a meaning.

**THEOREM 5.1.1:** For all recursively enumerable languages \( L_1 \) and \( L_2 \) there is a partial recursive function

\[ g : \text{MSEN}(L_1) \times \text{NAT} \to \text{MSEN}(L_2) \]

such that \( g \)

is a semantic preserving, complete translation between \( L_1 \) and \( L_2 \).

**PROOF:** We will show how to construct the function \( g \)
Let \( L_1 \) be an r.e. language on \( V_1 \) and \( A \) and 
\( L_2 \) be an r.e. language on \( V_2 \) and \( A \).

i) We first define some useful functions:

Let \( f_1 : \mathbb{N} \to V_1^* \times A^* \) 
\( f_2 : \mathbb{N} \to V_2^* \times A^* \)
be recursive enumerating functions for \( L_1 \) and \( L_2 \) 
(\( f_1 \) and \( f_2 \) exist since \( L_1 \) and \( L_2 \) are r.e.,)

Let \( <> \) be a pairing function and define

\[
p_1(<n_1,n_2>) = n_1 \quad \text{and} \quad p_2(<n_1,n_2>) = n_2
\]

for all \( n_1, n_2 \) in \( \mathbb{N} \)

ii) This permits us to define the function

\[
h : \mathbb{N} \to \text{MSEN}(L_1) \times \text{MSEN}(L_2)
\]

by:

\[
h(n) = (w_1,w_2) \quad \text{if} \quad (w_1,m_1) = f_1%p_1(n)
\]
\[
(w_2,m_2) = f_2%p_2(n)
\]

and \( m_1 = m_2 \)

= undefined otherwise

We note that \( h \) is partial recursive since
\( f_1, f_2, p_1, p_2 \) are recursive and that

\[
\text{RAN}(h) = \{(w_1,w_2) \mid w_1 \in \text{MSEN}(L_1), w_2 \in \text{MSEN}(L_2),
\]
\[
\text{MEAN}(w_1) \cap \text{MEAN}(w_2) \text{ is nonempty}\}
\]

Now we define \( g : \text{MSEN}(L_1) \times \mathbb{N} \to \text{MSEN}(L_2) \) by

\[
g(w_1,k) = w_2 \quad \text{if} \quad h(k) = (w_1,w_2)
\]

= undefined o/w

Note that \( g \) is partial recursive since \( h \) is.

iii) Suppose \( w_1 \) in \( \text{MSEN}(L_1) \) and \( k \) in \( \mathbb{N} \) s.t. \( (w_1,k) \) in \( \text{DOM}(g) \)

\[
g(w_1,k) = w_2 \implies h(k) = (w_1,w_2)
\implies (w_1,w_2) \text{ in RAN}(h)
\implies \text{MEAN}(w_1) \cap \text{MEAN}(w_2) \text{ is nonempty}
\implies \text{MEAN}(w_1) \cap \text{MEAN}(g(w_1,k)) \text{ is nonempty}
\]

Hence \( g \) is semantic preserving.
iv) Suppose \( w_1 \) in \( \text{MSEN}(L_1) \) and \( w_2 \) in \( \text{MSEN}(L_2) \) s.t. 
\( \text{MEAN}(w_1) \cap \text{MEAN}(w_2) \) contains \( m \).

Then \( (w_1,m) \) is in \( L_1 \) and there is a \( k_1 \) in \( \text{NAT} \) s.t. \( f_1(k_1) = (w_1,m) \)

And \( (w_2,m) \) is in \( L_2 \) and there is a \( k_2 \) in \( \text{NAT} \) s.t. \( f_2(k_2) = (w_2,m) \)

From the definition of \( \langle \rangle \), there is a unique \( n = \langle k_1,k_2 \rangle \)

Then \( f_1p_1(n) = (w_1,m) \) and \( f_2p_2(n) = (w_2,m) \) and \( h(n) = (w_1,w_2) \). Thus \( g(w_1,n) = w_2 \).

From this we can see that \( g \) is complete.

We have thus constructed a semantic preserving, complete translation from \( L_1 \) to \( L_2 \). [*]

This result guarantees the existence of computable translations that preserve meaning and are complete. However, they may not be total. We observed in the previous chapter that there are languages between which there are no total translations, or almost everywhere translations. The construction above could not produce total (or a.e) translations in those cases. However, suppose there is an almost everywhere (or total) translation between two languages. Is there a computable almost everywhere (or total) translation between them?

**COROLLARY 5.1.2:** If there is any almost everywhere, semantic preserving translation between r.e. languages \( L_1 \) and \( L_2 \), there is a computable, complete, almost everywhere, semantic preserving translation from \( L_1 \) to \( L_2 \).

**PROOF:** Let \( L_1 \) be a r.e. language on \( V_1 \) and \( A \) while \( L_2 \) is an r.e. language on \( V_2 \) and \( A \).

Further suppose that there is an almost everywhere semantic preserving translation

\( \text{TRANS}: \text{MSEN}(L_1) \times \text{NAT} \rightarrow \text{MSEN}(L_2) \)

where there is an \( n \) in \( \text{NAT} \) such that if
w in \text{MSEN}(L_1) \text{ and } |w| > n \text{ then there is a } k \text{ in } \text{NAT}
\text{ and } \text{TRANS}(w,k) \text{ is defined.}

Now let \( g: \text{MSEN}(L_1) \times \text{NAT} \rightarrow \text{MSEN}(L_2) \) be the function
of \text{THEOREM 5.1.1}. \text{ We know } g \text{ is complete, semantic preserving and computable.}

We further claim that \( g \) is an almost everywhere translation.

Suppose, BWOC, that \( g \) was not an a.e. translation.
Then in particular there must be a \( w \) in \text{MSEN}(L_1),
\( |w| > n \) (the same \( n \) as for \text{TRANS})
such that \( g(w,k) \) is undefined for \( k \) all in \text{NAT}.

But since \text{TRANS} is almost everywhere, and \( |w| > n \), we
know there is a \( k_1 \) such that \text{TRANS}(w,k_1) \text{ is defined,}
say \text{TRANS}(w,k_1) = w_2.

Since \text{TRANS} is semantic preserving, we know
\( \text{MEAN}(w) \cap \text{MEAN}(w_2) \) is nonempty. But then since
\( g \) is \text{complete} there must be a \( k_2 \) such that
\( g(w,k_2) = w_2 \). This contradicts our assumption.

Hence, BWOC, we conclude \( g \) is also almost everywhere. \([*)]

\text{COROLLARY 5.1.3:} \text{ If there is a total semantic preserving translation between recursively enumerable languages }
L_1 \text{ and } L_2, \text{ there is a computable, complete, and total semantic preserving translation between } L_1 \text{ and } L_2.

\text{PROOF:} \text{ The proof parallels that of the previous proof.}
\text{ Here we see that } n = 0. \([*)

Our formal definitions had characterized the notions of "almost everywhere" and "total". They left open whether or not such translations were computable. \text{ Theorem 5.1.1 and its corollaries assert that the defining properties are sufficient to guarantee translations of either type are indeed computable. These results are important to our work and will be applied to establish other computability results.}
5.2. Domain Sets

As an application of the results contained in the last section, we will introduce a set of index pairs related to the domains of partial recursive functions and use the results of the last section to explore some properties of this set. The observations made about this set will be applied later in this thesis as a tool for obtaining other results.

THEOREM 5.2.1: Let \( f_0, f_1, \ldots \) be an acceptable programming system.

Then there are indices ID, SUCC, and ZERO, as well as a total recursive function \( g: \text{NAT} \rightarrow \text{NAT} \) such that:

1) \( f_{\text{ID}}(n) = n \) for all \( n, n_1, n_2 \) in \( \text{NAT} \)

2) \( f_{\text{ZERO}}(n) = 0 \)

3) \( f_{\text{SUCC}}(\langle n_1, n_2 \rangle) = n_1 + 1 \)

4) \( f_g(i)(n) = n \) if \( f_{i}(n) \) defined
   \[ = \text{undefined} \] otherwise

PROOF: 1), 2) and 3) are a consequence of the function described being partial recursive, and hence being listed in an acceptable programming system. 4) can be derived using the s-m-n function. [*]

DEFINITION 5.2.1: Domain Set DOMSET

Let \( D_0, D_1, \ldots \) be an acceptable LDS\([V, \text{NAT}, f]\) based on acceptable programming system \( f_0, f_1, \ldots \) and let ID, SUCC, ZERO, and \( g \) be as in the previous definition. Then a domain set is defined:

\[
\text{DOMSET} = \{ h_1(i), h_2(i) \mid i \in \text{NAT} \}
\]

where \( h_1 \) and \( h_2 \) are total recursive functions defined so that
\[ D_{h1}(i) = (G,S) \text{ where } G = \{ [Z], [b], [Z<S>, S<Sb>, S<b>], Z \} \]

and \( S = (R) \) where \( R \) is defined:

\[
\begin{align*}
R(b) &= R(S<b>) = \text{ZERO} \\
R(Z<S>) &= \text{ID} \\
R(S<Sb>) &= \text{SUCC}
\end{align*}
\]

\[ D_{h2}(i) = (G,S_i) \text{ where } G \text{ is as above and } S_i = (R) \]

and \( R \) is as above but \( R(Z<S>) = g(i) \) \hspace{1cm} [*]

A domain set is a special set of index pairs with the indices referring to language definitions in LDS[V,NAT,f]. There would be many ways to define a domain set depending on the choices of indices ID, SUCC, ZERO, and function \( g \). However, no matter how constructed, a domain set has some interesting properties that we will exploit in our work. An obvious characteristic is that the first element of each index pair in a domain set is fixed (i.e. the function \( h1 \) is a constant function). Other properties are revealed in the next theorem.

**THEOREM 5.2.2:** Let \( \text{DOMSET} \) be a domain set described in the previous definition for LDS[V,NAT,f] and let \( h2 \) be the function specified in Definition 5.2.1.

Then if \((k,q)\) is in \( \text{DOMSET} \) where \( h2(i) \leq q \) we have:

1) \( L(Dk) = \{(b_{i+1},j) \mid j \text{ in NAT } \} \) is unambiguous

2) \( L(Dq) = \{(b_{j+1},j) \mid j \text{ in NAT and } f_i(j) \text{ defined} \} \) is unambiguous

3) There is a total, semantic preserving translation between \( L(Dk) \) and \( L(Dq) \)

iff

\( f_i \) is total

iff
There is a translation table (with corresponding source-target trees in a table entry computing identical functions) inducing a total translation between \( L(D_k) \) and \( L(D_q) \)

4) There is an almost everywhere, semantic preserving translation between \( L(D_k) \) and \( L(D_q) \)

\[ \text{iff} \]

\( f_i \) has a cofinite domain

\[ \text{iff} \]

There is a translation table inducing an almost everywhere translation between \( L(D_k) \) and \( L(D_q) \)

**PROOF:** Let \( \text{DOMSET} = \{(h_1(i), h_2(i) \mid i \in \text{NAT}) \} \) be as defined in Definition 5.2.1, with \( G, S, S_i, h_1, \) and \( h_2 \) as defined there.

We observe that grammar \( G \) is shared by \( D_k \) and \( D_q \) and that \( \text{GENC}(G, Z) \) consists of trees \( t_j \) of the form shown in Figure 19.

\[ \begin{array}{c}
  \text{Figure 19: Trees in } \text{GENC}(G, Z) \text{ for Theorem 5.2.1}
\end{array} \]
According to language definition Dk, \( t_j \) is the only phrase structure with frontier \( b_j^{+1} \) and that

\[
\text{RAN}(f_{t_j}) = \{ j \}
\]

According to Dq, \( t_j \) is the only phrase structure with frontier \( b_j^{+1} \) and it may or may not be a meaningful sentence.

In fact, \( b_j^{+1} \) is in \( \text{MSEN}(L(Dq)) \)

iff

\[
\text{RAN}(f_{R^*(t_j)}) = \{ j \}
\]

iff

\( f_i(j) \) defined

These observations are offered without proof and may be confirmed by examining Definition 5.2.1.

1) \( L(Dk) = \{(w,m) \mid \text{there is a } t \text{ in } \text{GENC}(G,Z) \text{ s.t.} \)
\[
w = \text{FR}(t) \text{ and } m \in \text{RAN}(f_{R^*(t)}) \}\)

\[
= \{(b_j^{+1},j) \mid j \in \text{NAT} \}
\]

2) \( L(Dq) = \{(w,m) \mid \text{there is a } t \text{ in } \text{GENC}(G,Z) \text{ s.t.} \)
\[
w = \text{FR}(t) \text{ and } m \in \text{RAN}(f_{R^*(t)}) \}\)

\[
= \{(b_j^{+1},j) \mid j \in \text{NAT} \text{ and } f[i](j) \text{ defined} \}
\]

3) A total semantic preserving translation exists between \( L(Dk) \) and \( L(Dq) \)

iff

for all \( w \) in \( \text{MSEN}(L(Dk)) \), there is \( w' \) in \( \text{MSEN}(L(Dq)) \)

s.t. \( \text{MEAN}(w) \cap \text{MEAN}(w') \) is nonempty

iff

for all \( j \) in \( \text{NAT} \), \( b_j^{+1} \) is in \( \text{MSEN}(L(Dq)) \)

iff
for all \( j \) in \( \mathbb{N} \), \( f^i(j) \) is defined

iff

\( f^i \) is a total function

The translation table \( \tau \) given by

\[
\begin{align*}
\tau(Z^S, 1) &= (Z^S, (1)) \\
\tau(S^Sb, 1) &= (S^Sb, (1, 0)) \\
\tau(S^b, 1) &= (S^b, (0))
\end{align*}
\]

is a translation table inducing a translation between \( D_k \) and \( D_q \)

iff

for all \( j \) in \( \mathbb{N} \), \( f_S(i)(j) = j \)

iff

for all \( j \) in \( \mathbb{N} \), \( f_i(j) \) is defined.

These observations offer proof of point 3) of the theorem.

4) There is an almost everywhere, semantic preserving translation between \( L(D_k) \) and \( L(D_q) \)

iff

there is \( n > 0 \), s.t. for all \( w \) in \( \text{MSEN}(L(D_k)) \)

\(|w| > n \implies \)

there is \( w' \) in \( \text{MSEN}(L(D_q)) \) s.t. \( \text{MEAN}(w) \cap \text{MEAN}(w') \)

is nonempty

iff

there is \( n > 0 \) s.t. \( j > n \implies b^{j+1} \) in \( \text{MSEN}(L(D_q)) \)

iff

there is \( n > 0 \) s.t. \( j > n \implies f_i(j) \) defined

iff

\( f_i \) has a cofinite domain
The translation table $\tau$ suggested by

$$\tau(t_n,1) = (t_n, (1,0,\ldots,0)) \text{ where } t_n \text{ is as in Figure 19}$$
$$\tau(S^{<S_b>},1) = (S^{<S_b>},(1,0))$$
$$\tau(S^{<b>},1) = (S^{<b>},0)$$

is a translation table inducing an almost everywhere translation between $L(D_k)$ and $L(D_q)$

iff

for all $j > n$, $g(i)(j) = j$

iff

for all $j > n$, $f_i(j)$ is defined

These observations provide proof of point 4 of the theorem [*]

The proof illustrates how the results of the last section can be combined with our knowledge of acceptable language definition system. The properties of DOMSETs will also be used in proofs that follow.

5.3. Translator Generation Decision Sets

In section 5.1 we developed simple necessary and sufficient conditions that guarantee the existence of a computable translation between two languages. In this section, we study the problem of deciding whether computable translations exist between a pair of languages. In particular, we are interested in whether there are general procedures that, when given a description of a source and target language, will discover a translation table determining a translation between the two languages. To be precise we should say "finds a translation table if one exists", since there may be no such table for a given source-target pair.

As we noted in the last chapter, there is a relationship between the solvability of the problem of searching for a table translation if one exists and the decision problem of testing for
the existence of a table translation. Since an answer to the search problem provides an answer to the decision problem, it suffices in this chapter to consider only the computability of TG decision problems. We know the corresponding TG search problems can have no "better" solvability.

**DEFINITION 5.3.1: Translator generation decision sets**

Let $D_0$, $D_1$, ... be an acceptable LDS[$V$, $\text{NAT}$, $f$]

$L[\text{TR}, \text{TOT}] = \{(i,j) | \text{there is a total, semantic preserving translation between } L(D_i) \text{ and } L(D_j)\}$

$L[\text{TT}, \text{TOT}] = \{(i,j) | \text{there is translation table inducing a total, semantic preserving translation between } L(D_i) \text{ and } L(D_j)\}$

$D[\text{TT}, \text{TOT}] = \{(i,j) | \text{there is translation table inducing a total, semantic preserving translation between } D_i \text{ and } D_j\}$

**NOTE:** These TG decision sets refer to total translations. We will add analogous decision sets for a.e. translations later.

Note the distinctions between the sets above. $D[\text{TT}, \text{TOT}]$ describes all ordered source-target LD pairs between which a total translation exists (and is specified by a translation table). $L[\text{TT}, \text{TOT}]$ describes all ordered source-target language pairs between which a total table translation exists (even though a table translation may not exist between each language definition pair describing those languages).

**EXAMPLE 5.3.1:** Suppose $L(D_i) = L(D_h)$ and $L(D_j) = L(D_k)$ and a total table translation exists between $D_i$ and $D_j$, but no total table translation exists between $D_h$ and $D_k$.

Then $(i,j)$ is in $D[\text{TT}, \text{TOT}]$ and $(i,j)$ is in $L[\text{TT}, \text{TOT}]$.

Also $(h,k)$ is not in $D[\text{TT}, \text{TOT}]$ but $(h,k)$ is in $L[\text{TT}, \text{TOT}]$ because there is a table translation between the languages described by $D_h$ and $D_k$. 
In addition, \(L[TT,TOT]\) and \(L[TR,TOT]\) are distinguished since there may be a total translation between two languages but not a table translation between the two languages.

From the theory of computability, we are aware of a distinction between the recursive and recursively enumerable sets. Suppose that one of our TG decision sets were found to be r.e. but not recursive. What would be the significance of this? Such a result says that a procedure exists that can answer affirmatively to the existence of a total (table) translation, if there is one, but may not answer if no such translation exists. Such "partial" decision procedures will be examined later.

We are aware too of the existence of non-r.e. sets and it is possible that our TG decision sets may be non-r.e. In this case, we would be interested in the relationship of the computability of the TG decision set to that of other known non-r.e. sets. Before beginning our computability analysis, we summarize some previous observations about TG decision sets.

THEOREM 5.3.1: Let \(D_0, D_1, \ldots\) be an acceptable LDS\([V,NAT,f]\).

1) There are \(i,j\) in NAT s.t.

\[(i,j) \notin L[TR,TOT] \]
\[(i,j) \notin L[TT,TOT] \]
\[(i,j) \notin D[TT,TOT] \]

2) \(L[TT,TOT]\) properly contains \(D[TT,TOT]\)

3) \(L[TR,TOT] = L[TT,TOT]\)

PROOF: 1) This simply follows from Corollary 4.1.2

2) From Krishnaswamy [Kri76]

3) From Krishnaswamy [Kri76] [*]

The first part of Theorem 5.3.1 merely states that there are languages (and hence language definitions) between which there is no total translation (and hence no table translation). Part 2) indicates that although a table translation may exist between two
languages, it is not necessary that it exist between all LDs describing those languages. Finally the last part suggests that there is a total table translation between languages if there is a computable total translation between them. The proof of this result necessarily employs some "unnatural" LDs for the two languages involved.

We first demonstrate that none of the TG decision sets are recursive. Such a result is not unexpected, since we have full freedom choosing the semantic functions within LDs and we have included all recursively enumerable languages. Thus when translation requires semantic equivalence, the TG problem may be related to known unsolvable problems of language equivalence and functional equivalence. Results for translation have appeared in the work of Pyster [Pys75] and Krishnaswamy [Kri76]. We will provide an alternative proof here to suggest how we can employ the results of the previous sections.

**THEOREM 5.3.2:** \[ L[TR,TOT] \] is not recursive for acceptable LDS[V,NAT,f]

**PROOF:** Let \( D_0, D_1, \ldots \) be acceptable LDS[V,NAT,f] with DOMSET as defined in Definition 5.2.1. In particular let \( \text{DOMSET} = \{ h_1(i), h_2(i) \mid i \in \text{NAT} \} \) From Theorem 5.2.2 we know that a total, semantic preserving translation exists between \( L(D_{h_1(i)}) \) and \( L(D_{h_2(i)}) \)

\[ \text{iff} \]

\[ f_i \text{ is total} \]

Hence \( (h_1(i), h_2(i)) \) is in \( L[TR,TOT] \) iff \( f_i \) is total

Now suppose that \( L[TR,TOT] \) was recursive. Then the reduction above would permit us to decide if \( f_i \) was total

But \( \{ i \mid f_i \text{ is total} \} \) is a well-known non-recursive set.

Hence, BWOC, \( L[TR,TOT] \) cannot be recursive

**COROLLARY 5.3.3:** \( L[TT,TOT] \) is not recursive for acceptable LDS[V,NAT,f]

**PROOF:** From Theorem 5.3.1, \( L[TT,TOT] = L[TR,TOT] \)

\[ (*) \]
THEOREM 5.3.4: $D_{[TT,TOT]}$ is not recursive for acceptable LDS $[V,NAT,f]$

PROOF: This result follows from a proof similar to that of Theorem 5.3.2 when we recognize that, from Theorem 5.2.2

that there is a table translation between $D_{h1}(i)$ and $D_{h2}(i)$ inducing a total semantic preserving translation

iff

$f_i$ is total

The previous three results dash any hope of finding general procedures that decide whether total translation or table translations exist between languages (or table translations between language definitions). Of course, as a consequence there can be no general solutions to the corresponding TG search problems. Two natural questions now arise. First, what more can we say about the computability of the TG decision problems? Perhaps we can look for general but "partial" decision procedures. Perhaps other problems can be found that are related in their computability. Secondly, can we invoke any restrictions on language or language definitions so that the restricted classes would have decidable TG decision problems? We examine these issues in the next two sections.

5.4. The Arithmetical Hierarchy and the TG Decision Problems

In this section we relate the computability of the TG decision problem previously discussed to the computability of other known problems from the theory of computability. This will be done by placing the TG decision sets within the arithmetic hierarchy and showing they are complete at their respective levels. As a result, we will be able to compare the computability of the TG decision problems to other natural problems in order to get a sense of how "hard" the TG decision problems are. Moreover, the location of the TG decision problems in the hierarchy will be useful later when we consider what information might be needed to provide oracular translator generation. Finally, from the specific completeness results presented, it will be easy to demonstrate there are no
partial decision procedures for total translator generation.

We recall that the notions of the arithmetical hierarchy were presented in terms of sets of natural numbers. For our work, we will place the TG decision sets within the arithmetic hierarchy and rely on our previous association of TG decision sets with their corresponding TG decision problems. We begin by establishing $L[TR, TOT]$ in the hierarchy.

**Theorem 5.4.1:** $L[TR, TOT]$ is in $\Pi_2$.

**Proof:** Let $D_0,D_1,...$ be an acceptable LDS[$V,NAT,f$], with COD the coding function $COD_Y$ and $STEP: NAT \times NAT \times NAT \to NAT$ be as in Def. 2.5.3.

Then there is a $k$ in $NAT$ such that $STEP(i,j,k) > 0$

iff

$f_i(j)$ is defined

Let $h: NAT \to NAT$ be as defined in Theorem 3.5.8, where $DOM(f_h(i)) = \{<COD(w),m> | (w,m) \in L(D_i) \}$

For total translations we have

$L[TR,TOT] = \{(i,j) | \text{ for all } w \in MSEN(L(D_i)), \text{ there is } w' \in MSEN(L(D_j)) \text{ such that } MEAN(w) \cap MEAN(w') \text{ is nonempty} \}$

$= \{(i,j) | PRED(i,j) \}$

where $PRED(i,j)$ is the predicate described by

$(\forall w)(\forall m)(\forall k)(\exists w') \exists q)

STEP(h(i),<COD(w),m>,k)>0 \implies STEP(h(j),<COD(w'),m>,q)>0$

where $w,w'$ in $V^*$ and $m,k,q$ are in $NAT$.

Observing that the predicate $PRED$ is of the form "$(\forall) (\exists)"$

we conclude that $L[TR,TOT]$ is in $\Pi_2$

[8]

**Theorem 5.4.2:** $L[TR,TOT]$ is $\Pi_2$-complete.
PROOF: We have just shown that $L[TR,TOT]$ is in $\Pi_2$. It remains to show that each $\Pi_2$-complete set is reducible to $L[TR,TOT]$. We use the known $[rog67]$ $\Pi_2$-complete set

$$\{ i \mid f_i \text{ is total} \}.$$ 

Let $DOMSET = \{(h1(i),h2(i)) \mid i \in \text{NAT} \}$ be the set of Definition 5.2.1 with total recursive $h1$ and $h2$ as defined there. We observe that, by Theorem 5.2.1,

$$j \in \{ i \mid f_i \text{ total} \}$$

iff

there is a total semantic preserving translation between $L(D_{h1(j)})$ and $L(D_{h2(j)})$

iff

$(h1(j),h2(j))$ is in $L[TR,TOT]$.

This reduction leads to the conclusion that all $\Pi_2$-complete sets reduce to $L[TR,TOT]$ and hence $L[TR,TOT]$ must be $\Pi_2$-complete. [*]

COROLLARY 5.4.3: $L[TT,TOT]$ is $\Pi_2$-complete.

PROOF: $L[TR,TOT] = L[TT,TOT]$ from Theorem 5.3.1. [*]

COROLLARY 5.4.4: $L[TR,TOT]$ and $L[TT,TOT]$ are not r.e.

PROOF: Since $L[TR,TOT]$ is $\Pi_2$-complete we know that $L[TR,TOT]$ is in $\Pi_2 - (\Pi_2 \cap \Sigma_2)$

But the r.e. sets are in fact the class $\Sigma_1$ which
is contained in $\Pi_2 \cap \Sigma_2$.

Hence $L[TR,TOT]$ cannot be r.e. $[\ast]$.

These results point out that "partial" decision procedures for total translation (or table translation) do not exist (procedures that would at least find translation if they exist). We should add that Theorems 5.3.2 and Corollary 5.3.3 also follow from Corollary 5.4.4 since all recursive sets are r.e. We next look at $D[TT,TOT]$.

**THEOREM 5.4.5:** $D[TT,TOT]$ is in $\Sigma_3$.

**PROOF:** Let $D_0, D_1, \ldots$ be an acceptable LDS[$V,NAT,f$] with COD the coding function $COD_\gamma$, and STEP as in Definition 2.5.3. Let $TAU$ and $TAUBAR$ be mappings as defined in Chapter 4 and $INDTAUBAR$ be the function of Theorem 4.2.2.

We have $D[TT,TOT] = \{(i,j) \mid$ there is a translation table $TAU$ inducing a total, semantic-preserving translation between $Di$ and $Dj\}$

$= \{(i,j) \mid$ there is translation table $TAU$ for $Gi$ and $Gj$ such that for all $w$ in $MSEN(L(Di))$, there is an $n$ in $NAT$ such that $TAUBAR(w,n)$ is defined and $MEAN(w) \cap MEAN(TAUBAR(w,n))$ is nonempty\}

$= \{(i,j) \mid$ PRED$(i,j)$ $\}$ where PRED$(i,j)$ is:

$((\exists T1)(\exists T2)(\exists TAU)(\forall w)(\forall m)(\forall j)(\exists k)(\exists w')(\exists j')(\exists j''))$

$[\{(T1,T2,TAU) is a translation table for Gi and Gj\} and$

$\{STEP(h(i),<COD(w),m>,j) > 0 \Rightarrow$

$STEP(INDTAUBAR(TAU),COD(w),k) = COD(w') + 1$

and $STEP(h(i),<COD(w'),m'>,j') > 0$

and $STEP(h(j),<COD(w'),m''>,j'') > 0 \} \} ]$

This complicated predicate is of the form "$(\exists)(\forall)(\exists)"
and hence we conclude that
\[
D_{TT,TOT} \text{ is in } \sum_3
\]  

[\text{\textit{\textbf{THEOREM 5.4.6}}: \text{\textbf{D[TT,TOT] is } \sum_3\text{-complete.}}}
\]

\text{PROOF: \textbf{Let D}_0,D_1,\ldots \text{ be an acceptable programming system.}}

\text{We have shown that } D_{TT,TOT} \text{ is in } \sum_3.

\text{It remains to show that a known } \sum_3\text{-complete set, namely } \{i \mid \text{DOM}(f_i) \text{ is cofinite} \} \text{ is reducible to } D_{TT,TOT}.

\text{Let } h_1, g, \text{ SUCC, ZERO, and ID be as defined in Definition 5.2.1.}
\text{Let } h_3: \text{NAT } \rightarrow \text{ NAT be the total recursive function defined so:}

\[
D_{h_3}(i) = (G,S_i) \text{ with }
G = \{(Z,S,Y),(b,0,1),(Z<S>,S<Sb>,S<b>,Z<Y>,Y<Y0>,Y<Y1>,Y<1>,Z<0>),Z)\}
\]

\text{and } S_i = (R) \text{ where } R \text{ is defined:}

\[
\begin{align*}
R(0) &= R(b) = R(S<b>) = R(Z<0>) = \text{ZERO} \\
R(1) &= R(Y<1>) = \text{COMP}(\text{SUCC, ZERO}) \\
R(Z<S>) &= g(i) \\
R(Z<Y>) &= \text{ID} \\
R(S<SB>) &= \text{SUCC} \\
f_{R(Y<Y0>)}(x_1,x_2) &= f_{R(Y<Y1>)}(x_1,x_2) = 2x_1 + x_2 \\
\text{for all } x_1,x_2 \text{ in NAT}
\end{align*}
\]

\text{We make the following observations:}

i) The language definition } D_{h3}(i) \text{ is a "union" of the language definitions } D' \text{ of example 4.2.3 and the LD } D_{h2}(i) \text{ of Definition 5.2.1.}

ii) \text{L}(D_{h1}(i)) = \{(b^{j+1},j) \mid j \text{ in NAT} \}

iii) \text{L}(D_{h3}(i)) = \{(b^{j+1},j) \mid f_i(j) \text{ defined} \} \cup \{(w,m) \mid w \text{ in } \{(0) \cup (0,1)^+ \} \text{ and } m \text{ is the binary "value" of } w \}
iv) There is no translation table inducing an almost everywhere, semantic-preserving translation between $D_{h1}(i)$ and $D'$ (based on example 4.2.3).

v) There is a translation table inducing an almost everywhere, semantic-preserving translation between $D_{h1}(i)$ and $D_{h2}(i)$

$\text{iff}$

$\text{DOM}(f_1)$ is cofinite (from Thm. 5.2.2)

vi) Given any single meaningful sentence $w$ of $L(D_{h1}(i))$, we can find a parse tree for it, and a parse tree according to $D'$ that defines a sentence that shares a meaning with $w$. (Again from Example 4.2.3).

Then from these observations, we conclude that a

Translation table exists inducing a total, semantic-preserving translation between $D_{h1}(i)$ and $D_{h3}(i)$

$\text{iff}$

$\text{DOM}(f_1)$ is cofinite

(for each of the sentences not translated into $L(D_{h2}(i))$ we can find a sentence in $L(D')$ into which to translate it. Since there are only a finite number of such sentences, we can simply match the source and target parse trees of these sentences and add them to the translation table inducing an almost everywhere translation to give a table inducing a total translation).

The reduction $j$ in \{1 \mid \text{DOM}(f_1) \text{ cofinite}\}

$\text{iff}$

$(h1(i),h3(i)) \in D[TT,TOT]$

suffices to show that $D[TT,TOT]$ is $\Sigma_3$-complete. [*]

COROLLARY 5.4.7: $D[TT,TOT]$ is not recursively enumerable.

PROOF: Parallels that of Corollary 5.4.4. [*]

Suppose that we instead look for "partial" translations— that is almost everywhere translations that are capable of translating
all but a finite number of sentences in the source language.

DEFINITION 5.4.1: Almost Everywhere TG Decision Set

Let $D_0, D_1, \ldots$ be an acceptable LDS[$V, NAT, f$].

$L_{TR,AE} = \{(i,j) \mid \text{there is an almost everywhere,} \quad \text{semantic preserving translation} \quad \text{between } L(D_i) \text{ and } L(D_j) \}$

We now examine the computability of $L_{TR,AE}$ by locating it within the arithmetic hierarchy.

THEOREM 5.4.8: $L_{TR,AE}$ is in $\sum_3$.

PROOF: Let $D_0, D_1, \ldots$ be an acceptable LDS[$V, NAT, f$] with $STEP, h, COD$ being the same as in Theorem 5.4.1. Then we have

$L_{TR,AE} = \{(i,j) \mid \text{there is } n \in NAT \text{ s.t.} \quad \text{for all } w \in MSEN(L(D_i)), \ |w|>n, \quad \text{there is a } w' \in MSEN(L(D_j)) \text{ with} \quad \text{MEAN}(w) \cap \text{MEAN}(w') \text{ nonempty} \}$

$= \{(i,j) \mid \text{PRED}(i,j) \}$ where PRED is a predicate

$(\exists n)(\forall w)(\forall m)(\forall k)(\exists w')(\exists k')$ \[
((\text{STEP}(h(i),<\text{COD}(w),m>,k)>0 \text{ and } |w|>n) \Rightarrow \text{STEP}(h(j),<\text{COD}(w'),m>,k')>0)
\]

where $n, m, k, k'$ are in $NAT$ and $w, w'$ in $V^*$. Observing that PRED here is of the form "$(\exists)(\forall)(\exists)"

we conclude that $L_{TR,AE}$ is in $\sum_3$. [*]

THEOREM 5.4.9: $L_{TR,AE}$ is $\sum_3$-complete.

PROOF: We have shown that $L_{TR,AE}$ is in $\sum_3$ in the previous theorem. We now reduce a known
\[ \Sigma_3 \text{-complete set, namely} \]

\[ \{ i \mid f_i \text{ has a cofinite domain} \} \]

to the set \( L[TR,AE] \).

Let \( \text{DOMSET} = \{ h_1(i), h_2(i) \mid i \in \text{NAT} \} \) be the set of Definition 5.2.1 with total recursive \( h_1 \) and \( h_2 \) as suggested there. Then by Theorem 5.2.1 we have

An almost everywhere, semantic preserving translation exists between \( L(D_{h_1}(i)) \) and \( L(D_{h_2}(i)) \)

\[ \text{iff} \]

\( f_j \) has a cofinite domain

Hence \((h_1(j), h_2(j)) \in L[TR,AE] \text{ iff} \)

\[ j \in \{ i \mid f_i \text{ has cofinite domain} \} \]

From this reduction we conclude that

\[ L[TR,AE] \text{ is } \Sigma_3 \text{-complete} \quad [*] \]

Thus the problem of finding "near" translations is in a sense "harder" than the problem of finding total translations. Once again we can conclude that \( L[TR,AE] \) is not r.e.

**COROLLARY 5.4.10:** \( L[TR,AE] \) is not recursively enumerable.

**PROOF:** Analogous to Corollary 5.4.4. \quad [*]

As a consequence of these results it is clear that the decision and search problems for translations and table translations are unsolvable. They are neither recursively solvable nor "partially" solvable. These problems have a computability that is related to that of other known problems that are similarly placed in the arithmetic hierarchy. We have noted that the problem of finding almost everywhere translations is harder in this sense of computability than that of finding total translations.

In addition, we will later consider oracular translator generation. This involves the additional ability in a translator
generator to ask questions of some external source to aid in the search for a translator. Known relationships between the sets located at various levels of the arithmetic hierarchy can guide us to oracle-based procedures for translator generation. This will be explored in a later chapter.

Given that the general problem of finding translations or table translators is unsolvable, it is necessary to look next at restrictions of the general problem. In the subsequent sections of this chapter we seek appropriate ways of limiting the class of languages or language definitions in the hopes of finding natural subproblems of the TG decision and search problems that will yield to algorithmic solution.

5.5. Translation for Classes of Restricted Language Definitions

In this section, we embark on the study of restriction on the general class of language definitions. Our aim is to find subclasses of language definitions that would be attractive from the perspective of translation. There are several desirable properties of such classes:

1. They are general, in that a wide variety of languages are described by LDs in the subclass.

2. They are natural in that restrictions imposed on the creating of LDs are reasonable and related to the set of languages they describe.

3. They are defined by a precise set of restrictions on how LDs are created.

4. They are analyzable from the point of view of translation—that is, the translation decision problems for the subclass are solvable.

Obviously, we will focus on this last property but the others are also important. In the remainder of this chapter, we examine the various properties in more detail and explore the relationship between them. In addition, we will propose various specific restrictions and look at the problems of translator generation for the subclasses determined by those restrictions.

In the previous sections, we have examined a variety of translation decision problems through the vehicle of translation decision sets. We have been able to comment on a few computability
questions involving these sets. Early work by Buttelaann [But74] focused on the issue of generating translation tables that could drive translation between languages described by specified LDs. He was interested in tables that were consistent with syntactic descriptions of the LDs. It was not as important whether there were other LDs for those languages (perhaps bearing little relationship to the given LDs) between which a translation might exist. Hence, Buttelaann was more interested in TG procedures for D[TT,TOT] than for L[TT,TOT]. This section continues in this spirit and exclusively focuses on the decision procedures for D[TT,TOT].

We first need to develop more carefully the notions suggested in the list of properties. For instance, there are alternative ways to discuss limited subclasses of LDs. On the one hand, we could suggest a subclass B of LDs and study translation for LD pairs where both the source and target LDs are drawn from B. On the other hand, we could specify a subclass C of LD pairs, and study the TG problem for ordered source-target pairs in the subclass. Studies of traditional compiler theory are of the latter type, since they investigate translation from higher-level languages to machine-level languages. Since it is also the more general viewpoint, we here specify subclasses of ordered source-target pairs and study the TG problems of these subclasses.

Let us next turn our attention to what it means for a subclass of language definitions to be precisely restricted. At the very least this should imply that we can generate a list of the language definitions that make up the subclass. Or it might suggest that we have the means to check if the set of restrictions have been adhered to or not. In the first case it would be desirable that the subclass be recursively enumerable; while in the second case the subclass should be recursive.

**DEFINITION 5.5.1: D[C,TT,TOT]**

Let LDS[V,A,f] be a language definition system and C a subset of NAT X NAT. Then

\[ D[C,TT,TOT] = \{(i,j) \in C \mid \text{there is a translation table inducing a total, semantic preserving translation between } \text{D}_i \text{ and } \text{D}_j \} \]

D[C,TT,TOT] simply extends the idea of D[TT,TOT]

Note that D[TT,TOT] = D[NAT X NAT, TT, TOT] \[\star\]
We have suggested that the TG problem for a subclass of language definitions is analyzable if we can recursively solve the translation decision problems for the subclass. Relying on our previous discussion, we can express this as follows:

**DEFINITION 5.5.2: Decidable Translator Generation**

Let LDS\([V,NAT,f]\) be an acceptable language definition system and \(C\) a subset of \(NAT \times NAT\).

Then the translator generation problem of \(C\) is decidable if \(D[C,TT,TOT]\) is recursive. [*]

Observe that we have introduced two uses of recursive. A subclass of source-target LD pairs is well-specified if the subclass is r.e or perhaps recursive. It has a decidable TG problem if the associated decision set for the subclass is recursive. The next result relates these two notions.

**THEOREM 5.5.1:** Let LDS\([V,NAT,f]\) be an acceptable language definition system. Then there is a subset \(Cl\) of \(NAT \times NAT\) such that:

\(Cl\) is recursive but \(D[Cl,TT,TOT]\) is not recursive.

**PROOF:** Let \(Cl = NAT \times NAT\)

\(Cl\) is clearly recursive but its decision set is not (note that \(D[NAT \times NAT,TT,TOT] = D[TT,TOT]\) which has been shown to not be recursive). [*]

The next result shows that if we insist that a subclass of language definitions contain a wide variety of language definitions that there may be difficulties in deciding translator generation for that subclass.
THEOREM 5.5.2: Let $D_0,D_1,\ldots$ be an acceptable LDS[$V,NAT,f$] with $L(D_k)$ nonempty and $L(D_p)$ being empty.

If $C$, a subset of $NAT \times NAT$, contains $L_{D_k} X \{p\}$ then $D[C,TT,TOT]$ is not recursive.

PROOF: Let $C$ contain $L_{D_k} X \{p\}$ with $D_k=(G_k, S_k)$, $L(D_k)$ nonempty and $L(D_p)$ empty. Suppose, by way of contradiction, that $L[C,TT,TOT]$ is recursive.

We first observe that for all $i$ in $NAT$ a translation table inducing a total translation exists between $D_i$ and $D_p$ if and only if $L(D_i)$ is empty.

We now define two useful functions:

i) Using the $s-m-n$ property, we define $g: NAT \rightarrow NAT$ so that for all $j$ in $NAT$,

$$f_g(j)(n) = n \quad \text{if } f_j(j) \text{ defined for } n \text{ in } NAT$$

$$= \text{undefined otherwise}$$

ii) Using $g$ and COMP of acceptable programming system $f_0,f_1,\ldots$ we define $h: NAT \times NAT \rightarrow NAT$ so that

$$D_h(i,j) = (G', S') \quad \text{with } G' = G_i \text{ and } S'=(R')$$

$$\quad \text{with } R'(t) = R_i(t) \quad \text{if } t \text{ in } VT_i \text{ or }$$

$$\quad \text{in } PR_i \text{ & } \text{ROOT}(t)\neq AX_i$$

$$= \text{COMP}(g(j),R_i(t)) \quad \text{if } t \text{ in } PR_i \text{ and }$$

$$\quad \text{ROOT}(t)=AX_i$$

Since $g$ and COMP are recursive, so too is $h$.

Several observations can be made:

A) For all $j$, $h(i, j)$ is in $L_{D_i}$

B) $f_g(j)$ is either the identity function (if $f_j(j)$ defined) or the empty function (if $f_j(j)$ undefined)

C) Hence $f_{\text{COMP}}(g(j),R_i(t)) = f_{R_i}(t)$ (if $f_j(j)$ defined) or is empty otherwise
D) Finally $L(D_{h(i,j)}) = L(D_i)$ if $f_j(j)$ defined
or is empty if $f_j(j)$ undefined

In particular we see that $L(D_{h(k,j)})$ is empty
iff
$f_j(j)$ undefined

This is true since $L(D_k)$ is nonempty. Hence a translation
table cannot exist between $D_{h(k,j)}$ and $D_p$
if $f_j(j)$ is defined.

Since $D[C,TT,TOT]$ is assumed to be recursive, this
reduction would suggest that \{ j | $f_j(j)$ undefined\}
is recursive. But this is a well-known non-recursive
set [rog67]. This leads us to conclude that
$C$ cannot have a decidable translator generation problem.

NOTE: What we have proven is actually even stronger than
the statement of the proof, for we don't need all
of $LD_{Gk} \times \{p\}$ to be in $C$, but only the set

\{(h(k,j),p) | j in NAT, L(Dk) nonempty, L(Dp) empty\}. [*]

This result shows that a subclass of LDs cannot have a
decidable translator generation problem if it contains all language
definitions involving a particular grammar. This result is also
used in subsequent theorems.

Having touched on the ideas of "well-specified" and
"analyzable" subclasses of LD pairs, we consider next the notion of
a "natural" subclass. General language definitions provide
considerable freedom in describing the syntax and semantics of a
language. In particular, one can choose from an infinite number of
equivalent grammars (equivalent in the sense they generate the same
set of sentences) although a change in productions within a grammar
might necessitate altering their associated semantic functions.
Moreover, there is considerable choice available in describing the
semantic functions themselves. In any programming system, there are
an infinite number of equivalent programs for the same function.
According to our definitions, any of these equivalent programs can
be specified in a language definition system without altering the
language described or forcing any change in the syntactic component
of an LD.

The existence of a translation or table translation between two
language definitions is also independent of how respective semantic
and meaning functions of the two languages are described. Existence of table translations may depend on the syntactic structure and semantic functions within the source and target LDs, but does not depend on how those semantic functions are described. It would thus seem desirable that a subclass of language definition pairs be "function-description" independent. We propose this as a reasonable property of natural subclasses— that no matter what restrictions be placed on creating language definitions that full freedom be permitted in selection of the programs used to describe the semantic and meaning functions.

DEFINITION 5.5.3: Equivalent language definitions

Let LDS[V,A,f] be a language definition system containing two language definitions D1 and D2. Then D1 and D2 are equivalent language definitions iff L(D1) = L(D2). [*]

DEFINITION 5.5.4: Semantically equivalent language definitions

Let LDS[V,A,f] be a language definition system containing two language definition D1 and D2 with:

\[ D_i = (G_i, S_i) \quad G_i = (VNi, VTi, PRi, Axi) \quad S_i = (Ai, Ri) \quad i = 1, 2 \]

Then D1 and D2 are semantically equivalent language definitions iff

1) \[ G_1 = G_2 \]

2) \[ f_{R1}(t) = f_{R2}(t) \] for all \( t \) in \( VT1 \cup PR1 \) [*]

If two language definitions are semantically equivalent they are certainly equivalent. However, if they are equivalent, they need not be semantically equivalent (e.g., they might have different grammars). We now extend these ideas to formalize what we mean by a function-description independent subclass of language definition pairs.
DEFINITION 5.5.5: Language index set

Let $D_0, D_1, \ldots$ be an LDS $[V, A, f]$ and $C$ a subset of $\text{NAT} \times \text{NAT}$. Then $C$ is a language index set iff for all $i, j, m, n$ in $\text{NAT}$,

If 1) $(i, j) \in C$
2) $D_i$ equivalent to $D_m$
3) $D_j$ equivalent to $D_n$ [*]

then $(m, n) \in C$

DEFINITION 5.5.6: Language definition index set

Let $D_0, D_1, \ldots$ be an LDS $[V, A, f]$ and $C$ a subset of $\text{NAT} \times \text{NAT}$. Then $C$ is a language definition index set iff for all $i, j, m, n$ in $\text{NAT}$

If 1) $(i, j) \in C$
2) $D_i$ semantically equivalent to $D_m$
3) $D_j$ semantically equivalent to $D_n$ [*]

then $(m, n) \in C$

Thus semantically equivalent language definitions have identical syntactic components and their semantics differ only in the way semantic and meaning functions are defined. Language definition index sets contain ordered index pairs for language definitions and include all index pairs for LDs that are semantically equivalent to index pairs which are in the LD index set. The terminology employed is consistent with the notions of "function index set" of recursive function theory ([rog67]).

As we now will see, the requirement that a subclass of language definitions be function-description independent conflicts with other desirable properties of a subclass.

THEOREM 5.5.3: Let $C$, a subset of $\text{NAT} \times \text{NAT}$, be a language index set for an acceptable LDS $[V, \text{NAT}, f]$.

Then $C$ is recursive iff 1) $C$ is empty
or 2) $C$ is $\text{NAT}$.
PROOF: Let \( \text{DO}, \text{DI}, \ldots \) be an acceptable LDS[\(V, \text{NAT}, f\)].

If \(C\) is \(\text{NAT} \times \text{NAT}\) or \(C\) is empty, then \(C\) is trivially recursive. So let us suppose that \(C\) is a recursive, nonempty, proper subset of \(\text{NAT} \times \text{NAT}\).

Let \(h: \text{NAT} \times \text{NAT} \to \text{NAT}\) be the same function as was described in the proof of Theorem 5.5.2. In addition to the observations made there, we note \(D_h(i, j)\) has been constructed so that

\[ D_h(i, j) \text{ and } D_i \text{ are equivalent iff } f_j(j) \text{ is defined} \]

Now let us select specific \(i\) and \(j\) so that \(L(D_i)\) is empty and \(L(D_j)\) is empty. Then either \((i, j)\) is in \(C\) or is not. Without loss of generality presume that \((i, j)\) is not in \(C\).

Since \(C\) is nonempty, there must be \(n\) in \(\text{NAT}\) such that \((n, n)\) is in \(C\), and further that \(L(D_n)\) is not empty (this follows since if \(L(D_n)\) was empty, then \((n, n)\) would be equivalent to \((i, j)\) and thus would not be in \(C\).

Now for all \(k\) in \(\text{NAT}\), consider \((h(n, k), h(n, k))\):

- if \(f_k(k)\) defined then \(L(D_{h(n, k)})\) is nonempty and \(D_{h(n, k)}\) would be equivalent to \(D_n\)
- if \(f_k(k)\) undefined then \(L(D_{h(n, k)})\) is empty and \(D_{h(n, k)}\) would be equivalent to \(D_i\) and \(D_j\).

Thus \((h(n, k), h(n, k))\) in \(C\) iff \(f_k(k)\) defined

But since \(C\) is recursive, this reduction would imply that \(\{k \mid f_k(k) \text{ is defined}\}\) was recursive. Since this is well-known to not be recursive, we have, BWOC, that \(C\) is not recursive if it is a nonempty, proper subset of \(\text{NAT} \times \text{NAT}\). [\(\ast\)]

This result is a "Rice's Theorem" (see [Mac78]) for language index sets. It suggests that we have no non-trivial subclasses of language definitions that can be effectively defined as long as we insist that all equivalent language definitions of a language must be included if any definition is. Note also that if \(C = \text{NAT} \times \text{NAT}\), that we have no hope of deciding existence of table translations by
our previous results!

But what of language definition index sets? Suppose that we do not insist on including all equivalent LDs but merely those that use the same semantic functions (albeit via different function definitions). The companion result is similar but not quite as broad as for language index sets. It states that there are no nonempty well-defined subclasses of language definitions that are both language definition index sets and admit decidable table translation.

**THEOREM 5.5.4:** Let $C$, a subset of $\text{NAT} \times \text{NAT}$, be a language definition index set for acceptable $\text{LDS}[V,\text{NAT},f]$ describing at least one pair of nonempty languages. Then

1) $C$ is not recursive

or 2) $D[C,TT,TOT]$ is not recursive.

**PROOF:** Let $D_0,D_1,\ldots$ be an acceptable $\text{LDS}[V,\text{NAT},f]$.

i) $C$ cannot be empty for it must describe at least one language pair.

ii) If $C$ were $\text{NAT} \times \text{NAT}$, $C$ would not have a decidable TG decision problem by Theorem 5.3.4.

iii) So, by way of contradiction, assume $C$ is a recursive, nonempty, proper subset of $\text{NAT} \times \text{NAT}$ and that $D[C,TT,TOT]$ is recursive.

Let $h:\text{NAT} \times \text{NAT} \rightarrow \text{NAT}$ be the function defined in the proof of Theorem 5.5.2. Observe also that when $L(D_i)$ is nonempty, for all $j$, $D_h(i,j)$ is semantically equivalent to $D_i$

$s_{ij}(j)$ defined.

Now suppose $(i,j)$ is in $C$ such that $L(D_i)$ and $L(D_j)$ are nonempty (such an $i$ and $j$ must exist for $C$)

Consider the set $B = \{(h(i,m),h(j,n)) \mid n,m \in \text{NAT}\}$
Since $D[C,TT,TOT]$ is recursive, $C$ cannot contain $B$ since this would violate Theorem 5.5.2. Hence, there must be some $m,n$ in $\text{NAT}$ so that $(h(i,m),(h(j,n))$ is not in $C$.

But then $L(D_h(i,m))$ is empty and $L(D_h(j,n))$ is empty since $L(D_h(i,m))$ nonempty $\Rightarrow f_m(m)$ defined $\Rightarrow D_h(i,m)$ semantically equivalent to $D_1$.

Since the analogous implication holds for $D_h(j,n)$ we would have $(h(i,m),h(j,n))$ in $C$ since $C$ is a language definition index set.

From this analysis, we have for all $k$ in $\text{NAT}$,

$$(h(i,k),(h(j,k)) \text{ in } C \text{ iff } f_k(k) \text{ defined.}$$

Since $C$ is recursive, this reduction would lead to a solution to the halting problem. This contradiction leads us to conclude $D[C,TT,TOT]$ cannot be recursive. [*]

These results suggest that interesting subclasses of language definitions (from the point of view of the translator generation decision problem) will not be found if we insist on independence from the way we define semantic functions. That is, interesting classes will not include all possible language definitions for a particular language. This suggests investigation of what restrictions could be placed on the way language definitions are prepared. The hope is that such restrictions would yield a natural, interesting subclass of language definitions for which we could effectively perform translator generation.

5.6. Syntactic and Semantic Restrictions

In the previous section we considered several desirable general properties of subclasses of language definition pairs. It was our hope that we could find a meaningful class of LDs for which we could perform generation of translators. After formulating several reasonable criteria, we saw that they competed with each other in the sense of not being mutually satisfiable.
We now take a different attack on the problem. We will propose a series of specific restrictions of language definitions and will study their effect on translator generation problems. It is our goal to find a suitable restriction or combination of restrictions on the class of language definitions that will yield a procedure for effectively carrying out translator generation.

The limitations are suggested from experience in defining grammars and partial recursive functions. In particular, we consider the following restrictions on syntactic definition:

1. Linear cfgs
2. Regular grammars
3. Finite context free grammars – those that define only a finite number of trees (and hence sentences)

For the semantic component, it is natural to consider the traditional restrictions of partial recursive functions to recursive functions and primitive recursive functions. Moreover, it is reasonable to limit the set of meanings associated with a terminal symbol through the meaning function of an LD. For instance, it would be natural to permit only a finite number of meanings for each terminal.

We now consider the impact of these restrictions. First we observe that the purely syntactic limitations are insufficient to guarantee the existence of a procedure for deciding existence of translators.

THEOREM 5.6.1: Let $D_0, D_1, \ldots$ be an acceptable LDS$[V,NAT,f]$, and

- $B_1 = \{i \mid D_i = (G_i,S_i) \text{ where } G_i \text{ is linear}\}$
- $B_2 = \{i \mid D_i = (G_i,S_i) \text{ where } G_i \text{ is regular}\}$
- $B_3 = \{i \mid D_i = (G_i,S_i) \text{ where } G_i \text{ is finite}\}$

Then $D[B_i, B_i, TT, TOT]$ is not recursive for $i=1,2,3$.

PROOF: This is a corollary of Theorem 5.5.2 when it is noted that each of $B_1$, $B_2$, and $B_3$ contain grammars generating both empty and nonempty languages [•]
We now propose an appealing restriction on the way language definition semantics are specified. It is reasonable to assume that each of the terminals of a language may have only a finite number of meanings. Certainly, in natural languages, words are presumed to possess only a finite set of interpretations. The notion is captured in the following definition:

**DEFINITION 5.6.1: Simple language definition**

Let \( V \) be a syntactic alphabet, \( A \) a semantic alphabet and \( f_0, f_1, \ldots \) be a listing of partial recursive functions.

Then a **simple language definition** is a listing \( D_0, D_1, \ldots \) of all language definitions of the form \( D = (G, S) \) where

\[
G = (V_N, V_T, PR, AX)
\]

is a cfg and \( S = (A, MF, SF) \) with

\[
SF = \{ f_R(t) \mid t \in PR \}
\]

where \( R: PR \rightarrow \text{NAT} \) is a total function such that

\[
f_R(t): (A^*)^k \rightarrow A^* \quad \text{for} \ t \in PR
g_{t} \text{ with } |R(t)| = k
\]

\[
MF = \{ M_t \mid t \in V_T \}
\]

where

\[
M_t \text{ is a finite subset of } A^* \text{ for all } t \in PR
\]

Such a language definition system will be denoted by **simple LDS**\([V,A,f]\) with \( S = (A, M, R) \). For a simple, acceptable **LDS**\([V,\text{NAT},f]\) we will denote the semantics by \( S = (M, R) \).

The structure of a simple LDS is suggested in Figure 20. Note that semantic functions in a simple language definition system are defined as they were in an ordinary language definition system. The next theorem examines the TG decision problem of such an LD.
Figure 20: A simple language definition system
THEOREM 5.6.2: Let D₀,D₁,... be a simple, acceptable LDS[V,NAT,f]. Then D[TT,TOT] is not recursive.

PROOF: This can be proven by adapting the proof of Theorem 5.3.4 to a simple language definition system. We note that the meaning functions used to describe grammars in the proof of that theorem assign a single meaning to terminal symbols in the grammar.

From this theorem we can see that using simple semantics does not in itself help the decidability of table translation. We next see that its combination with the syntactic restrictions of Theorem 5.6.1 is not even sufficient to achieve an effective total table translation procedure.

THEOREM: 5.6.3: Let D₀,D₁,... be a simple LDS[V,NAT,f] and

B₁ = {i | Di=(Gi,SI) and Gi is linear }
B₂ = {i | Di=(Gi,SI) and Gi is regular}
B₃ = {i | Di=(Gi,SI) and Gi is finite }

Then D[Bᵢ X Bᵢ,TT,TOT] is not recursive, for i=1,2,3.

PROOF: This is proven by adapting the proof of Theorem 5.5.2 to a simple language definition system. We note that the meaning function used in the proof of that theorem assigns a single meaning to each terminal in the grammar. Couple this with the observation that the classes B₁, B₂, and B₃ contain language definitions generating nonempty and empty languages.

We next study possible restrictions on the semantic functions of a language definition. While the previous restrictions on grammars are decidable — that is, one can algorithmically decide if a cfg is linear, regular, or finite — the proposed restricting of semantic functions to the class of recursive functions is problematic. It is undecidable if a partial recursive function is recursive. Thus,
limiting semantic functions in an LD to only recursive functions would not provide a subclass that would be well-defined. Hence, it will not be considered here. We turn our attention to semantic functions drawn from the class of primitive recursive functions.

**DEFINITION 5.6.2:** Primitive recursive language definition system

Let \( V \) be a syntactic alphabet, and \( f_0, f_1, \ldots \) be a listing of all primitive recursive functions. Then a primitive recursive language definition system is a listing \( D_0, D_1, \ldots \) of all language definitions of the form \( D = (G, S) \) where \( G = (V_N, V_T, P_R, A_X) \) is a CFG and \( S = (A, S_F) \)

where \( A \) is an alphabet in which \( \text{NAT} \) can be represented

and \( S_F = \{ h_t \mid t \in (V_T \cup P_R) \} \)

specified by a total function \( R : (V_T \cup P_R) \rightarrow \text{NAT} \)

so that \( h_t = f_R(t) \) if \( t \in V_T \)

\[ h_t(n_1, \ldots, n_k) = f_R(t)(<n_1, \ldots, n_k>) \]

if \( t \in P_R \)

\[ |f_R(t)| = k \]

Each such language definition system will be denoted by primitive recursive LDS\([V, \text{NAT}, f]\) with semantics \( S = (R) \).

**NOTE:** Analogous to acceptable LDS, we may define simple, primitive recursive LDS\([V, \text{NAT}, f]\) with \( S = (M, R) \). Also, we will carry over the notion of translator generation decision sets to primitive recursive LDS. [*]

Note that the semantic functions for a primitive recursive LDS are all total and so we do not have the concept of "meaningless sentence." Thus, our results for acceptable language definition systems need not carry over to primitive recursive LDS. In addition, although the restriction to primitive recursive may seem severe, all of the common arithmetical and string operations are known to be primitive recursive. The next theorem assesses the translator generation decision problem for primitive recursive LDS.
THEOREM 5.6.4: Let $D_0, D_1, \ldots$ be a primitive recursive $\text{LDS}[V, \text{NAT}, f]$. Then $D[\text{NAT} \times \text{NAT}, \text{TT}, \text{TOT}]$ is not recursive.

PROOF: Consider the recursive functions $h_1: \text{NAT} \to \text{NAT}$ and $h_2: \text{NAT} \to \text{NAT}$ where

$D_{h_1}(i) = (G, S)$ with $G = (\{X\}, \{b\}, \{X^{<b>}\}, X)$ and $S$ has

- $f_R(b)(n) = 0$ for all $n$ in $\text{NAT}$
- $f_R(X^{<b>})(n) = n$ for all $n$ in $\text{NAT}$

$D_{h_2}(i) = (G, S)$ with $G = (\{Z, S\}, \{b\}, \{Z^{<Sb>}, S^{<Sb>}, S^{<b>}\}, Z)$ and $S$ has

- $f_R(b)(n) = 0$ for all $n$ in $\text{NAT}$
- $f_R(Z^{<Sb>})(\langle n_1, n_2 \rangle) = f_i(n_1)$
- $f_R(S^{<Sb>})(\langle n_1, n_2 \rangle) = n_1 + 1$
- $f_R(S^{<b>})(n) = n$ for all $n, n_1, n_2$ in $\text{NAT}$

We observe $L(D_{h_1}(i)) = \{(b, 0)\}$ and $L(D_{h_2}(i)) = \{(b^{k+2}, f_i(k)) \mid k \text{ in } \text{NAT}\}$.

Hence a total table translation (specified by the obvious one-entry table) exists between $D_{h_1}(i)$ and $D_{h_2}(i)$ iff $0$ is in the range of $f_i$.

If the total table translator generation problem was decidable then the set $\{i \mid \text{prim. rec. } f_i \text{ has 0 in its range}\}$ would be recursive.

But this set is known to not be recursive. Thus $D[\text{NAT} \times \text{NAT}, \text{TT}, \text{TOT}]$ is not recursive.

[\text{*]}

COROLLARY 5.6.5: Let $D_0, D_1, \ldots$ be a primitive recursive $\text{LDS}[V, \text{NAT}, f]$ and

$B_1 = \{i \mid Di = (Gi, Si) \text{ with } Gi \text{ linear}\}$
B2 = \{i \mid Di = (Gi, Si) \text{ with } Gi \text{ regular}\}.

Then D[Bi X Bi, TT, TOT] is not recursive, for i=1, 2.

**PROOF:** The proof of Theorem 5.6.4 applies when you observe that for all i in NAT, h1(i) is in B1 and B2 and h2(i) is in B1 and B2. [*]

**COROLLARY 5.6.6:** Let D0, D1, \ldots be a simple, primitive recursive LDS[V, NAT, f] and B1 and B2 as in the previous corollary.

Then D[Bi X Bi, TT, TOT] is not recursive for i=1, 2.

**PROOF:** The proof of Theorem 5.6.4, together with the comments of Corollary 5.6.5 can be extended to a simple, primitive recursive LDS when it is observed that the meaning functions employed in Theorem 5.6.4 assign a single meaning to the terminal symbol used in the grammars of construction. [*]

**THEOREM 5.6.7:** Let D0, D1, \ldots be a primitive recursive LDS[V, NAT, f] and B = \{i \mid Di=(Gi, Si) and Gi is finite\}.

Then D[B X B, TT, TOT] is not recursive.

**PROOF:** Consider the functions h1 and h2 from NAT to NAT:

Dh1(i) = (G, S) where G = (\{Z\}, \{b\}, \{Z\langle b\rangle\}, Z)
and S has

\[
\begin{align*}
    f_R(b)(n) &= 0 \\
    f_R(Z\langle b\rangle)(n) &= n \quad \text{for all } n \text{ in NAT}
\end{align*}
\]

Dh2(i) = (G, S) where G is as in h2(i)
and S has

\[
\begin{align*}
    f_R(b)(n) &= n \\
    f_R(Z\langle b\rangle)(n) &= f_4(n)
\end{align*}
\]
for all \( n \) in \( \text{NAT} \)

Observe that

\[
L(D_{h_1}(i)) = \{(b, 0)\}
\]

\[
L(D_{h_2}(i)) = \{(b, k) \mid k \in \text{RAN}(f_i)\}
\]

and \( h_1(i) \) and \( h_2(i) \) are in \( B \) for all \( i \) in \( \text{NAT} \).

Then a total table translation (via the obvious one-entry table) exists between \( D_{h_1}(i) \) and \( D_{h_2}(i) \) iff

\( 0 \) in \( \text{RAN}(f_i) \).

Once again the recursiveness of \( D[\mathbb{B} \times \mathbb{B}, \mathbb{T}, \mathbb{T}, \mathbb{O}] \) would imply the recursiveness of \( \{i \mid 0 \in \text{RAN}(f_i)\} \) for primitive recursive functions.

Thus, \( D[\mathbb{B} \times \mathbb{B}, \mathbb{T}, \mathbb{T}, \mathbb{O}] \) is not recursive \([\ast]\)

Once again, we note that the combination of natural syntactic and semantic restrictions on the way language definitions are formed are not sufficient to characterize a class where we can decide whether table translations exist between members of the classes. In the next section, we investigate the translator generation problem for a subclass of language definitions the combines the severest syntactic and semantic restrictions that we have posed.

### 5.7. Language Definition Classes with Decidable TG Problems

In the last section several semantic and syntactic restrictions on the formation of language definitions were proposed. As we systematically surveyed the subclasses of language definitions formed from applying these restrictions solely or in combination we found the corresponding total table translation problem was undecidable.

It has been our hope to discover suitable restrictions that would yield a decidable table translation problem for the restricted class of LDS. In this section we find a combination of conditions on the preparation of language definitions that are sufficient to yield an effective procedure for deciding if a total table translation exists between a pair of languages described by such language definitions. Unfortunately this "optimistic" result ti
tempered by the observation that the languages described by our restricted LDs are quite simple.

**THEOREM 5.7.1:** Let $D_0, D_1, \ldots$ be a simple LDS $[V, A, f]$ based on an r.e. list $f_0, f_1, \ldots$ of total functions, and let

$$B = \{ i \mid D_i = (G_i, S_i) \text{ and } G_i \text{ is finite} \}.$$

Then there is an effective procedure for deciding if a total translation exists between any pair of language definitions drawn from $B$ (i.e. $D[B \times B, TT, TOT]$ is recursive).

**PROOF:** Let $D_i = (G_i, S_i)$ and $D_j = (G_j, S_j)$ be language definitions in $D[\{V, A, f\}]$. Consider the following algorithm:

1. Generate $T_i = \{ t \mid t \in GENC(G_i, A_X) \}$
2. Generate $T_j = \{ t \mid t \in GENC(G_j, A_X) \}$
3. Find the set $L_i = \{ (w, m) \mid w=FR(t) \text{ and } m \text{ in the range of } SEMFUN(t) \text{ for a } t \in T_i \}$
4. Find the set $L_j = \{ (w, m) \mid w=FR(t) \text{ and } m \text{ in the range of } SEMFUN(t) \text{ for a } t \in T_j \}$
5. Let $S = \{ w \mid (w, m) \text{ is in } L_i \}$ Assume $S = \{ w_1, \ldots, w_m \}$
6. Repeat while $(w, m)$ is in $L_i$
   6.1. Choose $w$ from $S$
   6.2. Check to see if there is an $m$ and $w'$ such that $(w, m)$ is in $L_i$ and $(w', m)$ in $L_j$.
   6.3. If not, stop and claim "No Translation"
   6.4. If so, set $S = S - \{ w \}$
7. Halt and claim "There is a Translation".

**OBSERVATIONS:**

A. Steps 1 and 2 will create finite sets $T_i$ and $T_j$ since the grammars $G_i$ and $G_j$ generate a finite number of trees, and
of these, only a finite number will be complete and axiom-rooted.

B. Steps 3 and 4 will create finite sets \( L_i \) and \( L_j \) since the semantic functions are total and there are at most a finite number of meanings for each element on the frontier of a tree.

C. \( S \) is finite since \( L_i \) is finite.

D. Step 6 will either halt if there is some source sentence that cannot be translated or the algorithm will halt at step 7 if each source sentence can be translated.

Hence, the algorithm given above is guaranteed to halt and will answer whether or not a translation exists between the two language definitions \( D_i \) and \( D_j \).

\[ \text{[\textit{\*}] \]}

COROLLARY 5.7.2: Let \( D_0, D_1, \ldots \) be a simple LDS\([V,A,f]\) with \( f_0, f_1, \ldots \) being an r.e. listing of some set of total function. Let \( C = \{ i \mid D_i = (G_i, S_i) \text{ and } G_i \text{ is finite} \} \).

Then there is an effective procedure for generating a table translation between \( D_i \) and \( D_j \) (\( i,j \) in \( C \)) if one exists, and informing one of the fact should no table translation be possible (i.e., the translator generation problem is solvable).

PROOF: Simply extend the algorithm of Theorem 5.7.1. Should a total translation exist, we know that for all \( w \) in \( S \), there is \( w', m, t_i \) and \( t_j \) such that:

\[
\begin{align*}
&i) t_i \text{ in } T_i \\
&ii) t_j \text{ in } T_j \\
&iii) w = FR(t) \\
&iv) w' = FR(t_j) \\
&v) m \text{ is in the range of } SEMFUN(t) \text{ and } SEMFUN(t')
\end{align*}
\]

The translation table can be formed by adding, for each \( w \) in \( S \), an entry in the table consisting of

\[ TAU( t_i, 1 ) = (t_j, (0,0,\ldots,0)) \]

where the index vector consists of \( m \) zeros if \( |FR(t_j)| = m \)

It should be clear that this table induces a total translation between \( D_i \) and \( D_j \).

\[ \text{[\textit{\*}] \]}


In our systematic survey of translator generation decision problems we have finally discovered a particular language definition system and a subclass of language definitions for which we can decide whether table translations exist between source-target LDs drawn from the class. But what can we say about this class? First, we note that there are an infinite number of language definitions in the class \( B \) of Theorem 5.7.1. (there are an infinite number of finite cfgs for languages over \( V \), and for each grammar, an infinite number of semantics which could be paired with it). Further, these LDS in \( B \) define an infinite number of different languages (again since there are an infinite number of different sentence sets defined by the class of all finite grammars). But each of the languages defined by language definitions in \( B \) is finite! This is a consequence of there only being a finite number of sentences generated by each LD in \( B \), and for each of there is only a finite set of associated meanings attached by the simple semantics. Certainly a class of LDs that avoids describing an infinite language is not a very rich characterization and the positive TG decision result for such a class is not as satisfying as one might like.

It is also worthy of mention that the algorithm for construction of the translation table suggested in Theorem 5.7.2 is far from elegant. Based on the algorithm given in Theorem in 5.7.1, the generation method is a brute force technique that essentially is a sentence by sentence table look up. Although it affirms the existence of a translator generator, it also motivates a search for methods that could more efficiently solve the problem. We conclude this section with one obvious application of Theorem 5.7.1.

**COROLLARY 5.7.3:** Let \( D_0,D_1,\ldots \) be a simple, primitive recursive LDS\([V,NAT,f]\) and \( B = \{ i \mid D_i=(G_i,S_i) \text{ and } G_i \text{ is finite} \} \).

Then 1) \( D[B \times B,TT,TOT] \) is recursive

2) There is an algorithm for generating a translation table inducing a total translation between \( D_i \) and \( D_j \) \( (i,j) \in B \) if one exists, and indicating when doesn't exist.

**PROOF:** From Theorem 5.7.1, when one notes that there is a listing \( f_0,f_1,\ldots \) of all primitive recursive functions. [\( * \)]
5.8. Context Free Language Definition Systems

In the last section, we systematically imposed more severe restrictions on the syntax and semantics of language definition systems in the search for decidable translator generation problems. In this section, we introduce a category of LDS that is based on very simple semantic-defining functions. The reader will note that the functions provide for little more than the concatenation of strings on the semantic alphabet. There are two reasons for studying this simple LDS - first, to illustrate the computability of translator generation is not merely a consequence of undecidable semantic questions. And second, to provide a very simple language definition scheme in which to analyze the complexity of translator generation (in a subsequent chapter). We begin by introducing the very simple semantic functions available in this LDS.

DEFINITION 5.8.1: Concatenation programming system

Let \( A \) be an alphabet and \( \text{COD}_A \) and \( \text{DECA} \) be coding functions.

Then \( F^A = \{ f_i \mid i \in \text{NAT} \} \) is a concatenation programming system on \( A \) if for all \( i \in \text{NAT} \)

\[
f_{2i}(x) = \text{DECA}(i) \quad \text{for all } i \in \text{NAT}
\]

\[
f_{2i+1}(x_1, \ldots, x_{i+1}) = x_1/\ldots/x_{i+1}
\]

Essentially, \( f_{2i} \) maps all arguments to the string in \( A^* \) that is coded by \( i \). \( f_{2i+1} \) concatenates \( (i+1) \) arguments on \( A^* \) into another string on \( A^* \). \( f_1 \) is the identity.\([\ast]\)

Thus, there are an infinite number of functions from \( A^* \) to \( A^* \), but for \( k > 2 \), there is a single function of \( k \) arguments. Concatenation semantics includes functions that are thus very simple, permitting only concatenation of strings.

We now propose a new type of language definition system based on concatenation programming systems. In this section these semantic functions are paired with context-free grammars. Notably, there are some restrictions (not present in the previous LDS in this
thesis) on what functions may be associated with terminals and productions in an LD.

DEFINITION 5.8.2: Context free language definition system

Let \(V\) and \(A\) be alphabets, and \(F//A\) be a concatenation programming system on \(A\). Then a context free language definition system is a listing \(D_0, D_1, \ldots\) of all language definitions of the form \(D = (G, S)\) where

\[
G = (VN, VT, PR, AX) \text{ is a cfg}
\]

and \(S = (A, SF)\) for

\[
SF = \{ f_{R(t)} \mid t \in VT \cup PR \} \text{ with}
\]

1) \(R(t) = 2i\) for some \(i \in \text{NAT}\)

if \(t \in VT\)

2) \(R(t) = 2i+1\) if \(t \in PR\)

with \(|FR(t)| = i+1\)

(in other words, \(R\) associates a single string \((\text{DEC}_A(i))\) with each terminal \(t\) in \(VT\), and the one appropriate concatenation or identity function with each production) We will denote such a language definitions system by context free LDS[\(V, A, F//\)]. \([*)\]

Now this type of language definition system is termed "context free" not because of the grammar component, but rather because the semantic functions are so tightly tied to the syntax. Essentially, the semantic function of a complete tree concatenates the single meanings of terminals arranged at the frontier. Note also that a context free language definition system is inherently simple for each terminal has only a single meaning. Consider Example 5.8.1:

EXAMPLE 5.8.1: Suppose \(V = \{a, b\}\) and \(A = \{c\}\). Then \(D = (G, S)\) appears in a context free LDS[\(V, A, F//\)].

\[
G = (\{Z, W\}, \{a, b\}, \{ZW^>, W<Wa>, W<Wb>, W<a>, W<b>, Z\})
\]

\[
S = (A, R) \quad \text{where}
\]

\[
f_{R(a)}(x) = cc \quad \text{for all } x \in A^*
\]
\[ f_R(b)(x) = \text{NULL} \]
\[ R(Z\langle W\rangle) = R(W\langle a\rangle) = R(W\langle b\rangle) = 1 \]
\[ R(W\langle Wa\rangle) = R(W\langle Wb\rangle) = 3 \]

Then if \( t = Z\langle W\langle W\langle a\rangle b\rangle b\rangle a\rangle \), then
\[ \text{SEMFUN}_t(x_1,x_2,x_3,x_4) = x_1//x_2//x_3//x_4 \]
and \( \text{TREEMEAN}(t) = cc//\text{NULL}//\text{NULL}//cc = cccc \)

This suggests that \( L(D) = \{ (w,c^{2k}) \mid w \in \{ a,b \}^* \text{ and } k \text{ is the number of } a's \text{ in } w \} \).

The strict relationship between syntax and semantics in a context free language definition system is precisely stated in the following result.

**THEOREM 5.8.1:** Let \( D_0, D_1, \ldots \) be a context free LDS[\( V, \text{NAT}, F// \)]. Then for all \( i \in \text{NAT} \), there is a homomorphism \( h_i: V \rightarrow A^* \) such that \( L(D_i) = \{ (w, h_i(w)) \mid w \in \text{SEN}(G_i) \} \).

**PROOF:** We actually prove a somewhat stronger result; namely that for all \( t \in \text{GENC}(G_i) \cup \text{VT}_i \), that the semantics \( S_i \) associates a single meaning \( h_i(w) \) with \( w = f_R(t) \).

First, let \( h_i(t) = \text{RAN}(f_{R_i}(t)) \) for all \( t \in \text{VT}_i \).

Since \( h_i \) is a homomorphism, we have \( h_i(xy) = h_i(x)//h_i(y) \) for all \( x,y \in V^+ \).

1) Let \( t \) be in \( \text{VT} \). Then \( f_R(t) = t \) and the single meaning associated with \( t \) is \( h_i(t) \).

2) Now suppose that our claim has been proven for all \( t \in \text{GENC}(G_i) \cup \text{VT}_i \) of height \( k \) or less. And suppose that \( t = t_0[t_1, \ldots, t_m] \) with \( t_0 \in \text{PR}_i \) and \( t_1, \ldots, t_m \) all of height \( k \) or less.

Then the meaning associated with \( t \) is
\[ f_{R_i}(t_0)(x_1, \ldots, x_m) \] where \( x_j \) is the meaning associated with tree \( t_j \), \( j = 1, \ldots, m \).
Thus, the meaning associated with \( t \) is
\[
\text{fr}_i(t_0)(x_1, \ldots, x_m)
\]
\[
= \text{fr}_i(t_0)(\text{hi}(\text{FR}(t_1)), \ldots, \text{hi}(\text{FR}(t_m)))
\]
\[
= \text{hi}(\text{FR}(t_1))//\ldots//\text{hi}(\text{FR}(t_m))
\]
\[
= \text{hi}(\text{FR}(t_1))//\ldots//\text{FR}(t_m)
\]
\[
= \text{hi}(\text{FR}(t))
\]

From 1) and 2) and induction we conclude that for all \( t \) in GENC(Gi) \( \cup \) VTi the single meaning associated with \( t \) is \( \text{hi}(\text{FR}(t)) \).

Certainly then for all \( t \) in GENC(Gi,AXi) we have \( \text{hi}(\text{FR}(t)) \) as the meaning associated with \( \text{FR}(t) \) and thus
\[
L(D_i) = \{(w,\text{hi}(w)) \mid w=\text{FR}(t), \ t \in \text{GENC}(Gi,AXi) \}
\]
\[
= \{(w,\text{hi}(w)) \mid w \in \text{SEN}(Gi) \}. \ [*]
\]

Now consider the problem of deciding if a translation exists between source-target pairs of LDs drawn from context free language definition systems.

THEOREM 5.8.2: Let \( D_0, D_1, \ldots \) be a context free LDS\([V,A,F//]\) with \( A \) containing at least 2 symbols.

Then \( D[\text{NAT} \times \text{NAT}, TT, TOT] \) is not recursive.

PROOF: We assume, without loss of generality, that \( V \) is contained in \( A \). If it were not, then we could encode \( V = \{v_0, v_1, \ldots, v_m\} \) into \( A = \{a_0, a_1, \ldots, a_n\} \) by associating \( a_{0,i}/a_{1,i} \) with \( v_i \). Then the questions below referring to \( V^+ \) would be rephrased as questions about \( \{a_{0,i}/a_{1,i} \mid i=1,\ldots,n\}^+ \) and the proof below will hold. Our assumption permits a cleaner presentation.

We begin by defining recursive functions \( h_1 \) and \( h_2 \) from \( \text{NAT} \) to \( \text{NAT} \) as follows:
\[
D_{h_1(i)} = (G,S) \quad \text{where} \ G = (\text{VNi} \cup \{Z\}, V, \text{FR}, AXi)
\]
with \( PR = \text{PR}_i \cup \{Z\langle Zv \rangle, Z\langle v \rangle \mid v \in V \} \cup \{Z_i\langle Z \rangle \} \)

and \( S = (A, R) \) with \( \text{RAN}(f_{R(y)}) = \{v\} \) for \( v \in V \)

and \( f_R(t) \) being the unique concatenation or identity function in \( F//A \)

for \( t \in \text{PR} \)

\[ D_{h2}(i) = (G, S) \text{ where } G = G_i \text{ and } S = (A, R) \]

with \( \text{RAN}(f_{R(y)}) = \{v\} \) for all \( v \in V^{Ti} \)

\[ R(t) = R_i(t) \text{ for all } t \in \text{PR}_i \]

Several observations are possible about \( h_1 \) and \( h_2 \). Clearly they are both related to \( G_i \).

i) The meaning associated with a complete tree in either \( D_{h1}(i) \) or \( D_{h2}(i) \) is simply the frontier of that tree!

That is a consequence of the meaning of a terminal being itself and the semantic functions of productions simply concatenate the meanings of terminals at their frontiers.

ii) \( L(D_{h1}(i)) = \{(w, w) \mid w \in V^+ \} \)

This is a consequence of observation i) and the fact that \( G_{h1}(i) \) includes \( G_i \) plus enough other productions to insure that \( \text{SEN}(G_{h1}(i)) = V^+ \)

iii) \( L(D_{h2}(i)) = \{(w, w) \mid w \in \text{SEN}(G_i) \} \)

This is a consequence of observation i) together with the fact that \( G_{h2}(i) = G_i \).

Now since each sentence of \( D_{h2}(i) \) has a unique meaning (namely itself!),

a total, semantic-preserving translation exists between \( D_{h1}(i) \) and \( D_{h2}(i) \) iff

\[ V^+ = \text{MEAN}(L(D_{h2}(i))) \]
iff
\[ \text{SEN}(G_i) = \text{SEN}(G_{h2(i)}) = V^+ \]

Hence \( \text{SEN}(G_i) = V^+ \iff (h_1(i), h_2(i)) \text{ in } L[\text{NAT} \times \text{NAT}, \text{TR}, \text{TOT}] \)

To complete the proof, we observe that if \( \text{SEN}(G_i) = V^+ \), then there is an obvious translation table inducing a total, semantic-preserving translation between \( D_{h1(i)} \) and \( D_{h2(i)} \).

It simply pairs the productions of \( PR_i \) (a subset of the source productions) with themselves in the target grammar. The index vector maps frontier elements to themselves. Thus we have
\[ \text{SEN}(G_i) = V^+ \iff (h_1(i), h_2(i)) \text{ in } D[\text{NAT} \times \text{NAT}, \text{TT}, \text{TOT}] \]

If \( D[\text{NAT} \times \text{NAT}, \text{TT}, \text{TOT}] \) were recursive, this reduction would provide a method of deciding if \( \text{SEN}(G_i) = V^+ \), but the latter is a well-known undecidable problem.
Thus we conclude that \( D[\text{NAT} \times \text{NAT}, \text{TT}, \text{TOT}] \) cannot be recursive. [*]

Here the undecidability of the translator generation question is a result of an unsolvable syntactic question of context free grammars. This is an example of an undecidable translator generation problem whose undecidability hinges solely on a syntactic question. One should note that the equivalence of two semantic functions in a concatenation programming system is simply determined by their definitions. Similarly, whether such semantic functions halt on a particular argument (they're total, by definition) or whether their range contains a particular value are both decidable questions. Thus, reducing translator generation decision problems to undecidable semantic problems (as was done previously) was not possible for a context free LDS. However, there is a related subclass of language definitions for which we can decide translator generation questions.

THEOREM 5.8.3: Let \( D_0, D_1, \ldots \) be a context free LDS[\( V, A, F//\Lambda \)].

Then for \( C = \{ i \mid D_i = (G_i, S_i) \text{ and } G_i \text{ is finite} \} \)

1) \( D[C \times C, \text{TT}, \text{TOT}] \) is recursive.
2) there is a procedure for generating a translation table
inducing a total, semantic preserving translation
between language definitions in C, or indicating if they
don't exist.

PROOF: Since $F^//_A$ is a listing of recursive functions and
a context free language definition system is inherently simple
the arguments of Theorem 5.7.1. can be extended to be prove
1) while Corollary 5.7.2 implies 2).

Thus we have located another class of language definitions where we
can decide translator generation specified by translation tables.
This class contains only finite languages in which each sentence is
paired with a meaning according to some string homomorphism and
hence the class represents a set of quite simple languages.

5.9. Computability - A Body of Negative Results

This chapter has attempted to carefully organize a collection
of results concerning the plausibility of solving translator
generation decision problems. For the most part the results are
negative - seemingly at every step of our systematic and natural
simplification of the translator generation problem we have met with
the impossibility of finding a general algorithm that can decide
translator generation. Certainly, formal theoretical approaches to
the TG problem are doomed to fail. And perhaps these results are
theoretical support for why we find producing actual translators a
difficult human task - there is difficult syntactic and semantic
processing that must go into the creation of a translation.

We should point out that the body of negative results presented
in this thesis are much broader than we first anticipated. From
earlier work it was known that the translator decision sets were not
recursive. But we found them to be associated wit even "more
unsolvable" problems higher in the arithmetical hierarchy. As we
looked at subclasses of language definitions, it was appealing to at
least have the full freedom to select the "programs" that defined
semantic functions. But we found there were no non-trivial classes
of language definitions that would have a decidable translator
generation problem if we insisted on this freedom. And as we
abandoned general language definition schemes and looked at more
restricted syntactic and semantic descriptions, we hoped to find
interesting classes of LDs that would yield decidable translator
generation. But we found decidable translation only for extremely
simple classes of language definitions. In Figure 21 we summarize our findings.

<table>
<thead>
<tr>
<th>SEMANTICS</th>
<th>Partial Recursive</th>
<th>Primitive Recursive</th>
<th>Concatenation Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General Cfg</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>Linear Cfg</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>Finite Cfg</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
</tbody>
</table>

Figure 21: Summary of computability results

Having carefully looked at a variety of computability questions, we now turn to the question of complexity. At the end of this chapter we did develop instances of language definition classes for which we could solve translator generation problems. In the next chapter, we examine the complexity of the solutions to these problems.
6. COMPLEXITY OF TRANSLATOR GENERATION

6.1. Assessing the Complexity of Translator Generation

In the last chapter, we extensively surveyed the translator generation decision problem. In most settings we found that deciding the existence of a table translation or translation was algorithmically impossible. We were able to locate a few classes of LDs where we could decide if translations or table translations exist. In this chapter, we focus on those decidable problems and assess the complexity of deciding translator generation in each case. We shift our attention from whether a solution is possible to how hard is the solution.

Complexity analyses can be of various types and several approaches appear in this chapter. Here we will make some general observations about the complexity of the decidable translator generation problems such as those in Section 5.7. These comments will not require any particular complexity measures. For some of the simpler systems of Section 5.8, we will propose some reasonable measures for a complexity analysis and provide more precise results in terms of those measures. In either case, we intend to make statements about the complexity of any algorithm that solves a decision problem. Thus we are analyzing a particular method of solution, but rather commenting about the complexity inherent in the problem itself.

This chapter examines the complexity of deciding the existence of a translation or translation table and not the complexity of generating a translation table. As we observed in Chapter 4, the complexity of generation can be no better than that of deciding existence.
6.2. Complexity of Deciding Translation for Finite Grammars in Simple Systems

In section 5.7, we discovered a category of language definition system, and classes of LDs within those systems that permitted us to decide the existence of translations for source-target pairs in those classes. In this section, we study the translator generation function that decides the existence of translators. We are interested in how hard it is to compute this function, i.e. how hard it is to decide translation for these classes. Our approach here is to compare the translator generation function for an LDS with the functions used in the language definitions of that LDS. For certain sets of semantic functions, this technique will allow us to strongly comment on the complexity of translator generation.

DEFINITION 6.2.1: A special set of language definitions

Let \( f_0, f_1, \ldots \) be a recursively enumerable list of total functions of one argument on the natural numbers s.t.

\[
\{f_0, f_1, \ldots \} \text{ is} \\
1) \text{ closed under composition} \\
2) \text{ contains the functions} \\
\text{POS}(n) = 0 \text{ if } n=0 \\
\text{POS}(n) = 1 \text{ if } n>0 \\
\text{CONST}[k](n) = k \text{ for all } n
\]

and let \( \text{LDS}[V, \text{NAT}, f] \) be a simple language definition system based on \( f_0, f_1, \ldots \).

Define the function \( g : \text{NAT} \times \text{NAT} \to \text{NAT} \) and the index \( q \) by:

\[
D_g(i, y) = (G, S) \text{ where } G = ((Z, X), \{b\}, \{Z<X>, X<b>\}, Z) \\
\text{ and } S = (R) \text{ where} \\
R(X<b>) = i \\
f_R(b) = \text{CONST}[y] \\
f_R(Z<X>) = \text{POS}
\]

\[
D_q = (G', S') \text{ where } G' = ((Z), \{b\}, \{Z<b>\}, Z) \\
\text{ and } S' = (R') \text{ where} \\
f_{R'}(b) = \text{CONST}[1] \\
f_{R'}(Z<b>) = \text{CONST}[1]
\]

[\text{[\text{\star}]}]
We now make several observations about the language definitions suggested above.

**THEOREM 6.2.1:** Let $D_0, D_1, \ldots$ be a simple LDS[$V, \text{NAT}, f]$ based on a listing of recursive function $f_0, f_1, \ldots$ as in Definition 6.2.1, with function $g$ and index $q$ as defined there. Then

1) $L(D_q) = \{(b,1)\}$

2) $L(D_g(i,y)) = \{(b,0)\}$ if $f_i(y) = 0$ $i, y$ in NAT $= \{(b,1)\}$ if $f_i(y) > 0$

3) A total, semantic preserving (table) translation between $D_g(i,y)$ and $D_q$ exists

iff $f_i(y) > 0$

**PROOF:** 1) and 2) follow directly from the definition of the languages described by $q$ and $g(i,y)$

3) follows from the definition of total, semantic preserving translation since such a translation can exist if and only if 1 is in $\text{MEAN}(D_g(i,y))$. In that case the obvious simple translation table:

$\text{TAU}(Z<X<b>>,1) = (Z<b>,(0))$

suffices to define the total translation if $f_i(y) > 0$ $[\ast]$  

We next demonstrate that when the functions available in a language definition system satisfy certain reasonable conditions (including those of Definition 6.2.1) then the translator generation function associated with finite grammars in that LDS cannot be among the functions used in the LDS. Although this might at first seem neither surprising nor significant, this result allows us to make some interesting judgments about the complexity of translator generation.
THEOREM 6.2.2: Let \{f_0, f_1, \ldots\} be a recursively enumerable set of total functions of one argument on NAT that is closed under composition and contains the functions POS and CONST[k] for all k in NAT (these functions were defined in Definition 6.2.1). Further suppose that the function set contains the functions:

a) \( h(<i,y>) = <g(i,y), q> \) where \( g \) and \( q \) of Def. 6.2.1

b) \( \text{SWITCH}(n) = 1 \) if \( n=0 \)
   \( = 0 \) if \( n>0 \)

and let LDS\([V,NAT,f]\) be a simple LDS based on \{f_0, f_1, \ldots\}

If \( C = \{i \mid D_i=(G,S) \text{ with } G \text{ finite} \} \) then

1) \( D[C \times C,T,T,TOT] \) is recursive and

2) the function that characterizes \( D[C \times C,T,T,TOT] \) for \( C \times C \) is not in \{f_0, f_1, \ldots\}.

PROOF: We begin by defining the predicate \( \text{DECIDE}: \text{NAT} \rightarrow \text{NAT} \)

\[
\text{DECIDE}(<i,j>) = 1 \text{ if } i \text{ and } j \text{ are in } C \text{ and a total, semantic preserving table translation exists between } D_i \text{ and } D_j
\]

\[= 0 \text{ otherwise} \]

1) the function \( \text{DECIDE} \) is a decision function for table translation for the class \( C \times C \). That \( \text{DECIDE} \) can be effectively computed follows from the fact that it is decidable if a cfg generates a finite number of trees (and hence whether \( i \) and \( j \) are in \( C \)) together with an argument paralleling that of Theorem 5.7.1 (noting here that the semantics is simple and the functions total)

2) now suppose, by way of contradiction, that \( \text{DECIDE} \) was a function in \( \{f_0, f_1, \ldots\} \). We observe that:

A) Since \( \text{DECIDE} \), \( h \), and \( \text{SWITCH} \) are in \( \{f_0, f_1, \ldots\} \) and the set is closed under composition, we know there must be some \( j \) such that \( f_j = \text{SWITCH} \cdot \text{DECIDE} \cdot h \)
B) From the definition of $f_j$ we have

$$f_j(i,y) = 0 \quad \text{if there is a translation table}$$

between $D_{g(i,y)}$ and $D_q$ inducing a total, semantic preserving translation

$$= 1 \quad \text{otherwise}$$

This follows from $D_{g(i,y)}$ and $D_q$ having grammars that generate a finite number of trees, and hence we are assured that for all $i$ and $y$, $g(i,y)$ and $q$ are in $C$.

C) $f_j$ is a 0–1 valued function since SWITCH is.

D) Since for all $i$ and $y$, $g(i,y)$ and $q$ are in $C$, we have also that $\text{DECIDE}_{h}(i,y) = 1$ if $f_i(y) > 0$

$$= 0 \quad \text{otherwise}$$

This comes from Theorem 6.2.1.

E) From C) and D) we have $\text{DECIDE}_{h}(i,y) = f_j(y)$ for all $y$ in $\text{NAT}$

But then no table translation exists between $D_{g(j,y)}$ and $D_q$

iff

$$f_j(j,y) = 1 \quad \text{iff } \text{DECIDE}_{h}(j,y) = 1$$

iff

a total, semantic preserving table translation exists between $D_{g(j,y)}$ and $D_q$

This contradiction shows that the decision function DECIDE cannot be in the set $\{f_0, f_1, \ldots\}$

When this result is used in the context of some well-known sets of total functions, it provides information on the computational complexity of language definition systems based on those function sets. Indeed, there are a number of sets of functions satisfying
the conditions set forth in Theorem 6.2.2.

COROLLARY 6.2.3: Let D0, D1, ... be a simple, primitive recursive LDS[V, NAT, f] and C = {i | Di=(Gi, Si), Gi a finite cfg}

Then there is no primitive recursive procedure that decides the existence of a total, semantic preserving translation (or translation table inducing such a translation) for any source-target pair of LDs drawn from C.

PROOF: This result follows from Theorem 6.2.2 when it is noted that there is a listing f0, f1, ...
of all primitive recursive functions and that
i) the primitive recursive functions are closed under composition
ii) POS, SWITCH, and CONST[k] are primitive recursive for all k in NAT
iii) the function h is primitive recursive since it involves a simple combination of syntactic and semantic components as suggested by the constructions of function g and index q.

Thus, by Theorem 6.2.2, there is a decision function for C X C, but that function is not primitive recursive. [*]

COROLLARY 6.2.4: Let D0, D1, ... be a simple, primitive recursive LDS[NAT, V, f] and C = {i | Di=(Gi, Si) Gi finite}

Then there can be no primitive recursive bound on the complexity of a procedure for deciding the existence of total, semantic preserving translations (or translation tables inducing such translations) for the class C X C.

PROOF: Suppose we employ some complexity measure, such as TMtime, to measure the complexity of primitive recursive procedures. It is a well-known result that any procedure whose running time is bound by a primitive recursive function is itself primitive recursive. Hence the existence of a primitive recursive bound on a solution of a translator generation decision problem would imply
COROLLARY 6.2.5: Let $D_0, D_1, \ldots$ be a simple LDS\[V, \text{NAT}, f\]
where $f_0, f_1, \ldots$ is a listing of all polynomial
time computable functions. Let $C = \{ i \mid D_i = (G_i, S_i) \}
G_i \text{ finite} \}

Then any procedure for deciding the existence of a total,
semantic preserving translation (or total table translation)
is not polynomial time computable.

PROOF: It is asserted here without proof that the simple
functions of Theorem 6.2.2 are polynomial time computable.
Further, the polynomial time computable functions are
closed under composition. As a consequence of Theorem
6.2.2, a decision procedure does exist, but cannot be
polynomial time.

These applications of Theorem 6.2.2 provide very strong
comments on the difficulty of deciding or performing translator
generation, hence it is indeed possible. For example, for the case of
primitive recursive functions this means that the complexity of
deciding translation is not even on the order of:

$$\text{SUPERFX}(k, n) = 2^2 \ldots \cdot \text{where exponentiation is done } k \text{ times}$$

This is obviously a very intractable problem for which there is no
practical solution. Similarly, the result for polynomial time
computable functions shows there is no practical decision procedure
for the TG problem for LDS's based on them. A close examination of
the proofs of Theorems 6.2.1 and 6.2.2, indicates that the
complexity difficulty cited above is not a result of placing
language definitions in the class $C \times C$, but rather comes from the
semantic functions used in the semantics of the language
definitions.

Once again, we have been confronted with a negative result that
speaks to the difficulty of translator generation. We have found
that where the translator generation problems are computable, that
any method of solving them will require inordinate amounts of work for an infinite number of instances of the problems. The complexity here is inherent in the problem and not a function of the particular procedure used to solve it.

6.3. Complexity of TG Problems for Simple, Context Free Language Definition Systems

In section 5.8, we identified a very simple set of semantic functions and studied the context free language definition systems based on them. We found that for the class of LDs with finite cfg (defining their syntax) that the translator generation decision problem were solvable. In this section we provide a careful analysis of the complexity of solving that TG decision problem. It should serve as a promising opportunity to get a "positive" complexity result since this LDS is very simple. In addition, the techniques used in this analysis are illustrative of those we use in examining other TG problems.

In the last section we made somewhat general observation about the problem complexity. By comparison, we will be much more precise in this section. In order to do this we will need to adopt a measure of the "size of the problem", to identify an appropriate model of computation, and to select a complexity measure to study the work done by TG decision procedures. Having done that, we will be in a position to assess the complexity of the TG decision problem. This will be done by determining a function that gives a lower bound on the work done by any procedure solving that TG problem.

We adopt some specific, but reasonable measures of the size of context free grammars and the size of the semantics in a context free LDS. These measures are then extended to a size measure for a language definition, and for pairs of language definitions.

DEFINITION 6.3.1: GRsize(G), SEMsize(S)

Let $D=(G,S)$ be a language definition in context free LDS[$V,A,F//f$], with $G = (VN, VT, PR, AX)$ and $VT = (vl,...,vj)$ and $PR = (t1,...,tm)$ and $S = (R)$. Let $h$ be the string homomorphism associated with $D$ by Theorem 5.8.1.

Then there is a $K_F > 0$ such that
GRsize(G) = 
(\mid VN \mid + \mid VT \mid + m + \mid FR(t_1) \mid + \ldots + \mid FR(t_m) \mid ) (\log(\mid VN \mid + \mid VT \mid )

SEMsize(S) = 
K_F (\mid h(v_1) \mid + \ldots + \mid h(v_j) \mid + \mid FR(t_1) \mid + \ldots + \mid FR(t_m) \mid ) (\log |A|)

LDsize(D) = GRsize(G) + SEMsize(S)

These are referred to as size measures for grammar G, semantics S, and language definition D, respectively.

These definitions provide measures of the size of the syntactic and semantic components of an LD in a context free language definitions system. The measure GRsize is that of Hunt, Rosenkrantz, and Szymanski ([Hun76]) for context free grammars. SEMsize acknowledges the existence of "simple" procedures for computing the functions of a concatenation programming system. For terminals one need only write out the corresponding string in A*. For productions, one must simply concatenate n strings together (if the frontier of t is of length n). That we should add the sizes of the syntactic and semantic components to measure the size of a language definition seems reasonable. Similarly, we can compound measures for grammars and language definitions:

**DEFINITION 6.3.2:** GRsize(G1,G2) LDsize(D1,D2)

Let GRsize and LDsize be as defined in Definition 6.3.1., and D1=(G1,S1) and D2=(G2,S2) be language definitions in context free LDS[V,A,F//].

Then GRsize(G1,G2) = GRsize(G1) + GRsize(G2)

LDsize(D1,D2) = LDsize(D1) + LDsize(D2)

Although Definition 6.3.2 uses the same function names as the previous definition, the context of usage of these measures will suggest which of the meanings we intend for them. The analysis we perform here will be based on these assumptions about the size of grammars and semantics in a context free LDS. We will employ Turing machines as a model of computation and use the measures DTMtime and NDTMtime for gauging the work done by the Turing machines. Certainly we could design deterministic Turing machines that compute...
the functions in a concatenation programming system an satisfy the
assumptions on SEMsize given in Definition 6.3.1. Before completing
our complexity analysis, we present some preliminary lemmas about
language definitions in a context free LDS.

**DEFINITION 6.3.3: \( D_g(i) \)**

Let \( D_0, D_1, \ldots \) be a context free LDS\([V,A,F//]\) with \( V \) a subset
of \( A \). Then \( g: \text{NAT} \rightarrow \text{NAT} \) is a total function defined so that

\[
D_g(i) = (G_i, S) \quad \text{where} \quad S = (\mathcal{R}) \quad \text{and}
\]

\[
f_{R(v)}(x) = v \quad \text{for} \quad v \in \text{VTi}
\]

\[
f_{R(t)} \quad \text{is as normal} \quad \text{for} \quad t \in \text{PRi}
\]

for cont. free
LDS

\[
(*)
\]

**LEMMA 6.3.1:** Let \( D_0, D_1, \ldots \) be a context free LDS\([V,A,F//]\) with \( V \) a subset
of \( A \) and \( g \) as defined in Definition 6.3.3.

Then \( L(D_g(i)) = \{(w,w) \mid w \in \text{SEN}(G_i) \} \) for all \( i \in \text{NAT} \)

**PROOF:** According to Theorem 5.8.1, the homomorphism determining
\( L(D_g(i)) \) is \( h(v) = v \) for all \( v \in \text{VTi} \). When extended this
homomorphism is the identity mapping. The result follows from
Theorem 5.8.1. \([*)\]

**LEMMA 6.3.2:** Let \( D_0, D_1, \ldots \) be a context free LDS\([V,A,F//]\) with
\( V \) a subset of \( A \) and \( g \) as defined in Definition 6.3.3. Then
if \( G_i \) and \( G_j \) are finite cfgs,

there is a translation table between \( D_g(i) \) and \( D_g(j) \)
inducing a total, semantic preserving translation

iff

\( L(D_g(i)) \) is a subset of \( L(D_g(j)) \)

iff

\( \text{SEN}(G_i) \) is a subset of \( \text{SEN}(G_j) \)
PROOF: The languages \( L(D_g(i)) \) and \( L(D_g(j)) \) are finite.

Hence if a total translation exists, surely a total table translation exists (simply pair the derivation tree of a source sentence with the derivation tree of its target sentence translation). By Lemma 6.3.1, the meaning of a sentence according to these LDs is the sentence itself. Thus

A total table translation exists between \( D_g(i) \) and \( D_g(j) \)

iff

A total translation exists between \( L(D_g(i)) \) and \( L(D_g(j)) \)

iff

\( (w,w) \text{ in } L(D_g(i)) \implies (w,w) \text{ in } L(D_g(j)) \)

iff

\( w \text{ in SEN}(G_i) \implies w \text{ in SEN}(G_j) \)

The last two implications suggest that

\( L(D_g(i)) \subseteq L(D_g(j)) \) and \( SEN(G_i) \subseteq SEN(G_j) \) [★]

We have suggested a particular construction that adds to any context free grammar \( G_i \) a particularly simple semantics. We showed that the effect of these semantics is to "lift" the frontier of trees defined by the grammar. This assures us that the meaning of a sentence such a language definition is the sentence itself. When the construction is applied to finite context free grammars, Lemma 6.3.2 shows a direct relationship between the table translation problem for the constructed LDs and the containment problem of the grammars on which the LDs are based. This relationship will be exploited in the complexity analysis that follows. We now observe that the construction of Definition 6.3.3 can be done rather efficiently.

THEOREM 6.3.3: Let \( GR_{\text{size}} \) and \( LD_{\text{size}} \) be as defined in Definition 6.3.1, for a context free LDS[\( V, A, F// \)] with \( V \) a subset of \( A \) and \( g \) as defined in Definition 6.3.3.

Then there is a deterministic Turing machine \( T \) and a positive
constant \( K \) such that if \( G_i \) is a context free grammar, we have

1) When \( i \) is input to \( T \), \( T \) will output \( D_g(i) \)

2) \( \text{LDsize}(D_g(i)) \leq K \cdot \text{GRsize}(G_i) \)

3) \( \text{DTMtime}_T(i) \leq K \cdot \text{GRsize}(G_i) \)

**PROOF:** Suppose that \( G_i = (V_{Ni}, V_T, P_R, A_X) \) and \( i \) is input to \( T \).

The Turing machine \( T \) will work in a straightforward way to construct the appropriate semantics to attach to \( G_i \). For each \( t \) in \( (V_T \cup P_R) \) we "plug in" the appropriate index according to Definition 6.3.3.

\[
\text{LDsize}(D_g(i)) = \text{GRsize}(G_i) + \sum_{t} K_F (|V_T| + |P_R(t_1)| + \ldots + |P_R(t_m)|)(|A|) \\
\leq K' \cdot \text{GRsize}(G_i) \text{ for a suitable } K'
\]

It should be clear that the work performed by \( T \) can be done in time proportional to the length of the constructed \( D_g(i) \).

Thus a constant \( K > K' \) can be chosen so that

\[ \text{DTMtime}_T(i) \leq K \cdot \text{GRsize}(G_i) \] [*]

We next state a result of Hunt, Rosenkrantz, and Szymanski ([Hun76]) which comments on the complexity of the containment problem for finite context free grammars.

**THEOREM 6.3.4:** Let \( \text{GRsize} \) be as defined in Definition 6.3.1.

Then there is a constant \( J > 0 \) such that if \( T \) is any nondeterministic, multitape Turing machine solving the containment problem for the class of context free grammars generating finite languages, then

\[ \text{NDTMtime}_T(G, G') > 2^{(J \cdot \text{GRsize}(G, G') / \log \text{GRsize}(G, G'))} \]

for infinitely many pairs \((G, G')\) of finite cfgs.
Based on this result, and the relationship between the containment problem of finite grammars and the table translation problem of context free language definitions systems, we are ready to develop a complexity result for the decision procedures discussed in Section 5.8.

**Theorem 6.3.5** Let $D_0, D_1, \ldots$ be a context free LDS $[V,A,F,F]$ with $V$ a subset of $A$. Let $g$ be the function defined in Definition 6.3.3 and $\text{LDsize}$ the measure developed in Definition 6.3.1. Let $B = \{i \mid D_i = (G_i,S_i) \text{ and } G_i \text{ finite}\}$.

Then there is a constant $K > 0$ such that if $T$ is any nondeterministic, multitape Turing machine computing $D[B \times B, T, T, T]$ then

$$\text{NDTMtime}_T(i,j) > 2(K^{*}\text{LDsize}(D_i,D_j)/\log \text{LDsize}(D_i,D_j))$$

for infinitely many $(i,j)$ in $B \times B$.

**Proof:** The proof will proceed by way of contradiction. However, we must first make some observations about any nondeterministic, multitape Turing machine $T'$ computing $D[B \times B, T, T, T]$. We note that such a $T'$ can be combined with the deterministic Turing machine $T$ of Theorem 6.3.3 to create a composite nondeterministic, multitape Turing machine $T''$ that works as follows:

Given a pair $(i,j)$ in $B \times B$ on its input tape $T''$ would:

1. Use the program of $T$ to create $D_g(i)$ on a working tape
2. Use the program of $T$ to create $D_g(j)$ on a working tape
3. Use the working tapes as an input to be processed by $T'$ and to use as output the result $T'$ generates from these inputs.

From this construction, we observe that the composite machine $T'$ will answer "yes" if there is a total, semantic preserving table translation between $D_g(i)$ and $D_g(j)$ and "no" if no such translation exists.
By Theorem 6.3.2, we know that such a table translation exists between $D_g(i)$ and $D_g(j)$ iff $SEN(Gi) \subseteq SEN(Gj)$.

Three other observations are also useful. The first two follow from the definition of exponentiation.

A. for all $k_1, k_2 > 0$ $k_1 \cdot n \leq 2^{(k_2 \cdot n / \log n)}$ a.e. $n$

B. for all $k_1 > 0$ $k_1 \cdot 2^n \leq 2^{k_1 \cdot n}$ a.e. $n$

C. there is a $k > 0$ such that for all $i, j$ in NAT

$$GRsize(Gi, Gj) \leq LDsize(Dg(i), Dg(j)) \leq k \cdot GRsize(Gi, Gj)$$

Now suppose by way of contradiction, that for all $I > 0$, there is a nondeterministic, multaitape Turing machine $T'$ computing $D[B \times B, TT, TOT]$ such that

$$NDTMtime_T((i, j)) \leq 2(I \cdot LDsize(Di, Dj) / \log LDsize(Di, Dj))$$

a.e. $(i, j)$ in $B \times B$

Then we can select a particular $I = J/2K$ where $K$ is the constant of Theorem 6.3.3 and $J$ is the constant of Theorem 6.3.4.

Let $T'$ be the particular Turing machine satisfying the above assumption.

Then we have

$$DTMtime_T(i) + DTMtime_T(j)$$

$\leq K \cdot GRsize(Gi) + K \cdot GRsize(Gj)$ by Theor. 6.3.3

$\leq K \cdot GRsize(Gi, Gj)$

$\leq K \cdot LDsize(Dg(i), Dg(j))$ from C)

$\leq K \cdot K \cdot GRsize(Gi, Gj)$ from C)

$\leq 2(J \cdot GRsize(Gi, Gj) / 2 \cdot \log GRsize(Gi, Gj))$

a.e. $(i, j)$ from A)

By our assumption,
Let \( T'' \) be the composite Turing machine built from \( T \) and \( T' \) of our assumption. Then \( T'' \) solves the containment problem for \( B \times B \) and by Theorem 6.3.4

\[
\text{NDTMtime}_{T'}(i,j) \leq 2^{\left( J \times \text{LDsize}(D_g(i), D_g(j)) / 2 \times \text{log} \text{LDsize}(D_g(i), D_g(j)) \right)}
\text{a.e.} \ (i,j) \text{ in } B \times B
\]

\[
\leq 2^{\left( J \times \text{GRsize}(G_i, G_j) / 2 \times \text{log} \text{GRsize}(G_i, G_j) \right)}
\text{a.e.} \ (i,j) \text{ in } B \times B \text{ from C}
\]

Let \( T'' \) be the composite Turing machine built from \( T \) and \( T' \) of our assumption. Then \( T'' \) solves the containment problem for \( B \times B \) and by Theorem 6.3.4

\[
\text{NDTMtime}_{T''}(i,j) > 2^{\left( J \times \text{GRsize}(G_i, G_j) / \text{log} \text{GRsize}(G_i, G_j) \right)}
\text{i.o.} \ (i,j) \text{ in } B \times B
\]

Let \( H(i,j) = 2^{\left( J \times \text{GRsize}(G_i, G_j) / \text{log} \text{GRsize}(G_i, G_j) \right)} \)

From the construction of \( T'' \), and Theorem 6.3.4 we have

\[
\text{DTMtime}_T(i) + \text{DTMtime}_T(j) + \text{NDTMtime}_{T'}(g(i), g(j)) > H(i,j)
\text{i.o.} \ (i,j) \text{ in } B \times B
\]

which implies (from our previous steps) that

\[
2 > 2^{\left( J \times \text{GRsize}(G_i, G_j) / 2 \times \text{log} \text{GRsize}(G_i, G_j) \right)}
\text{i.o.} \ (i,j) \text{ in } B \times B
\]

and finally by A)

\[
2^{\left( J \times \text{GRsize}(G_i, G_j) / \text{log} \text{GRsize}(G_i, G_j) \right)} > H(i,j)
\text{i.o.} \ (i,j) \text{ in } B \times B
\]

But this suggests \( H(i,j) > H(i,j) \text{ i.o.} \ (i,j) \text{ in } B \times B \). Hence by way of contradiction the Theorem is proven [\*]

This result suggests that although there is a procedure for deciding table translation for the class of language definitions discussed in
Section 5.8, that any formal approach will require an intractable amount of work for infinitely many instances of the problem. This is a surprising result in that the subclass of languages is strikingly simple. More perspective on this result is provided in the next section where we examine the effect of language definition choice on the intractability proven here.

6.4. Existence of Good Choices of Language Definitions

We have shown in Section 5.8 that there was a subclass of LD pairs in context free language definitions for which we could effectively decide existence of translators. In the last section we discovered that any procedure solving this decision problem would require a provably intractable amount of time for infinitely many instances of the problem. An "instance" in this case is a selection of particular source-target pair of language definitions. Suppose that the choice of (i,j) is one of the instances requiring an inordinate amount of work. Is it true that we would get provably intractable complexity for all language definition pairs describing L(Di) and L(Dj)? Or are there LDs Dm and Dn with L(Dm)=L(Di) and L(Dn)=L(Dj) such that the pair (m,n) can be processed in "reasonable" time? The issue here is whether the bad complexity results are a function of the languages involved or the language definitions used.

In this section, we describe a particular algorithm that decides the existence of a table translation between language definitions of the LD class exhibited in Section 5.8. We intend to show that, using this particular algorithm, that for infinitely many source-target pairs of finite languages, there are choices of language definitions describing those languages for which the algorithm will work in "reasonable" time. This implies the infinitely many instances of provable intractable complexity result from "inappropriate" (from the complexity perspective) choices of language definitions.

THEOREM 6.4.1: Let D0,D1,... be a context free LDS[V,A,F//] with V a subset of A, and LDsize as defined in Definition 6.3.1. Let B = {i | Di=(Gi,Di) and Gi finite}

Then there is a deterministic Turing machine T that computes D[B X B,TT,TOT], such that for any pair of languages L and L' on V* X A*, with
(x,y) in L \implies x=y \quad \text{and} \quad (x,y) \text{ in } L' \implies x=y

there is a pair (i,j) in B \times B such that
1) L(D_i) = L
2) L(D_j) = L'
3) TMtime_T(i,j) requires only polynomial time relative to LDsize(D_i,D_j).

PROOF: Let g be as defined in Definition 6.3.3. We first describe the Turing machine T by providing the procedure that T will carry out. This will be done by describing some procedures that will be used as a part of the overall procedure.

PROC1: To Find GENC(G,AX) for finite G = (VN,VT,PR,AX)

1. Set TR = \{ t \in PR \mid \text{ROOT}(t) = AX \}
2. Set TR' = TR \cup \{ t \mid t' \in T \text{ and } TR' \in PR \text{ s.t.} \ t = COMP(t',t'',k) \text{ for some } k \}
3. If TR \neq TR', set TR = TR' and go to step 2
   Else go to step 4.
4. GENC(G,AX) = \{ t \in TR \mid FR(t) \in VT^+ \}

It should be clear that PROC1 finds GENC(G,AX) using the normal derivation process. Since G is finite, this procedure must eventually halt.

PROC2: To Find L(D) from GENC(G,AX) for finite G and S=(R)
Let h(v) = RAN(f_{R}(v)) for all v in VT

1. Repeat for each t in GENC(G,AX)
   1.1. Find w=v_1...v_n, where w=FR(t), v_k in VT \ k=1..n
   1.2. Set m = NULL
   1.3. Repeat for k=1,2,...,n
      1.3.1. Set m = m//h(v_k)
   1.4. Add (w,m) to L(D)
The algorithm enumerates \( L(D) \) by finding the meaning associated with the frontiers of axiom-rooted, complete trees. That this successfully associates a meaning with a sentence is a consequence of Theorem 5.8.1.

PROC3: To Check if a Translation Exists Between Two Finite Languages \( L \) and \( L' \).

1. Repeat for each \((w,m)\) in \( L \)
   
   1.1. Find the set \( \text{MEAN}(w) = \{ m' \mid (w,m') \text{ in } L \} \)
   
   1.2. Set flag to 0
   
   1.3. Repeat for each \((w',m')\) in \( L' \)
       
       1.3.1. If \( m' \) in \( \text{MEAN}(w) \) set flag to 1
   
   1.4. If flag is 0, then halt with "No Translation"

2. Halt with "Table Translation Exists"

This is an obvious way to check for a total, semantic preserving translation. If a total translation exists between finite languages, it is a simple matter to exhibit a table translation inducing a total translation (by matching the derivation tree of a source sentence with the target derivation tree of its translation).

Now the procedure that is carried out by Turing machine \( T \) is described by:

PROC: To Decide if a Table Translation Exists Between \( D_i \) and \( D_j \) (given \( G_i \) and \( G_j \) are finite grammars and the semantics are those of a context free LDS)

1. Use PROC1 to find \( \text{GENC}(G_i, Ax_i) \)
2. Use PROC1 to find \( \text{GENC}(G_j, Ax_j) \)
3. Use PROC2 to find \( L(D_i) \) from \( \text{GENC}(G_i, Ax_i) \) and \( S_i \)
4. Use PROC2 to find \( L(D_j) \) from \( \text{GENC}(G_j, Ax_j) \) and \( S_j \)
5. Use PROC3 to check whether a total, semantic preserving table translation exists between \( D_i \) and \( D_j \).
It should be clear that PROC will compute \( D[B \times B, TT, TOT] \) since it will check the languages developed by PROC1 and PROC2.

Now suppose \( L = \{(wl, wl), \ldots, (wp, wp)\} \) and \( L' = \{(wl', w2'), \ldots, (wq', wq')\} \) are languages on \( V^* \times A^* \).

We select language definitions \( D_g(i) \) and \( D_g(j) \) where

\[
G_i = ([Z], V, PR, Z) \text{ with } PR = \{Z<..w..> | (w, w) \text{ in } L\}
\]

\[
G_j = ([Z], V, PR', Z) \text{ with } PR' = \{Z<..w'..> | (w', w') \text{ in } L'\}
\]

By the construction of \( D_g(i) \) and \( D_g(j) \) and Theorem 6.3.1,

\[
L(D_g(i)) = L \quad \text{and} \quad L(D_g(j)) = L'
\]

It remains to analyze the time needed by T to process \((g(i), g(j))\). We assert that for \( J = |w1| + \ldots + |wp| \)

\[
J' = |w1'| + \ldots + |wq'|
\]

A. Step 1 of PROC requires on the order of \((|L| + J)^2\) steps

Since all the productions of \( G_i \) are complete and axiom-rooted the steps 1) through 4) of PROC1 are carried out only once. The step requiring the most work would be step 3) or PROC1 that checks for all possible compositions among trees in TR.

B. Step 2 of PROC requires on the order of \((|L'| + J')^2\) steps

By the same analysis as in A.

C. Step 3 of PROC requires on the order of \((|L| + J)\) steps

Basically, PROC2 will require enough concatenations at step 1.3.1 to build up all the associated meanings. Then each \((w, m)\) must be added to \( L(D) \) at step 1.4.

D. Step 4 of PROC requires on the order of \((|L'| + J')\) steps

By the same analysis as in C.

E. Step 5 of PROC requires on the order of \(|L|(|L| + |L'|)\) steps

For each element of the source language, one must look through all other elements of the source language (looking for
ambiguities), and then inspect each element of the target language for possible translations.

Thus the overall procedure carried out by $T$ requires on the order of

$$\left(|L| + |J|\right)^2 + \left(|L'| + |J'|\right)^2 + |L| + J + |L'| + J' + |L|(|L| + |L'|) \text{ steps which is certainly of the order}$$

$$\left(|L| + J + |L'| + J'\right)^3$$

Now this order of complexity is polynomial in $\text{LDsize}(D, D')$ which is greater than $\left(|L| + |L'| + J + J'\right)$ by our definitions.

Thus we see that $T$ requires only a polynomial number of steps to process the chosen LD pair $\langle g(i), g(j) \rangle$.

How does this result relate to that of Theorem 6.3.5? Consider the procedure of Theorem 6.4.1. It must be the case that this procedure will require an intractable amount of time to process an infinite number of problem instances. But for each of these instances, there would be alternative equivalent language definitions defining the same languages for which the procedure would work "fast". Thus, the negative results of the last section are due to "inappropriate" (from the complexity point of view) choices of language definitions. Theorem 6.4.1 asserts there are "good" alternatives.

On the other hand, the construction used in the proof of Theorem 6.4.1 is not a very attractive one. In effect, it suggests that we compute the entire language described by a language definition in order to find an appropriate alternative LD. Since we use language definitions to avoid exhaustive listing of languages, this construction is not attractive. However, it is a construction that proves the existence of "better" LDs (in the context of complexity) - perhaps there are alternative language definitions that preserve the reasonable complexity results of Theorem 6.4.1 while avoiding the cumbersome enumeration of a language used in the construction of 6.4.1.
7. ORACULAR TRANSLATOR GENERATION

The computability and complexity results accumulated in the last two chapters speak to the inherent difficulty of performing translator generation. We have sought purely algorithmic means to find translators given only formal descriptions of the languages involved. In this chapter we investigate the possibility of using an "external knowledge source" to assist in the generation process. This is depicted in Figure 22. This external source, or oracle, can be consulted by a TG procedure from time to time. Hopefully, if the oracle is sufficiently knowledgable, it can provide enough information to permit us to procedurally generate translators. Obviously, such an oracle could not provide "computable" information, for this would provide a computable solution to the TG problems (and we have shown they do not exist). If we are generous, we might view the oracle-assisted approach as a model of a human-machine translator generation system.

In this chapter we concern ourselves with three issues. First, we are interested in what kind of information might be provided by an oracle to assist in translator generation. In so doing, we will formalize a suggestion made by Buttelmann ([But74]) for a particular type of table translator generation. Secondly, we wish to analyze the computability of the oracular translator generation procedure we propose. This will be done primarily by viewing it in the context of the arithmetic hierarchy. Finally, we can make some simple assertions about the utilization of the oracle in a TG procedure. In particular, we examine how many calls are made on the oracle and how much work will be done by the procedure outside of these calls.

7.1. Oracle Turing Machines

To clarify the concept of oracle-based procedures, we identify a particular model of computation based on oracles. It is related to the traditional Turing machine model of computation, but provides for a precise characterization of oracle consultation.
Figure 22: Oracular Translator Generation
DEFINITION 7.1.1: Oracle Turing Machine

An oracle Turing machine consists of an oracle set with knowledge of membership in some set, a finite state control, two read-write heads and a pair of two-way infinite tapes (designated as the "primary tape" and the "oracle tape"). A program for an oracular Turing machine is an extension of a program for a normal Turing machine. It features specification of two special states - an "oracle consultation" state and a "resume computation" state. Computation proceeds as for a normal Turing machine, but symbols may be read or written on both the primary and oracle tapes.

However, when the oracle Turing machine is in the "oracle consultation" state, operation is different from the normal. In this state, the oracle machine will make a transition to the "resume computation" state and the contents y of the oracle tape will be replaced by an indication of whether y was in the oracle set or not. This is accomplished in one step and the contents of the primary tape are left unchanged.

An oracle Turing machine is illustrated in Figure 23.

A configuration can be used to characterize the state of processing of an oracle Turing machine. It consists of the:

1. Contents of each of the tapes

2. The position being scanned by the read-write heads on each of the tapes.

3. The state of the finite-state control of the oracle Turing machine.

This informal definition should provide a more precise idea of what an oracle based procedure is. When we speak of oracle-based procedures in this thesis we are suggesting that they could be carried out on an appropriately designed oracle Turing machine.

DEFINITION 7.1.2: A-recursive, A-recursively enumerable

Let T be an oracle Turing machine using an oracle that provides information about membership in some set A. Then if T is used to recognize a set B (always halting and stating whether an input set is in B or not) then we say B is A-recursive. If T is
Figure 23: Oracle Turing Machine
used to enumerate B, we say B is \textit{A-recursively enumerable}.

7.2. Elementary Table Translation

In this chapter we examine a particular type of table translation that was suggested by Buttelmann ([But74]). Its significance is twofold. First, it is of historical interest, since Buttelmann was one of the first to study formal translator generation. More importantly, this kind of table translation captures the natural idea that the trees paired in a translation table entry should compute the same (or nearly the same) semantic function. Although this was not strictly required by the table translations studied so far in this thesis, most of the examples utilized tables that satisfied this criteria.

In this section, we define Buttelmann's \textit{elementary} table translation and set the stage for proposing an oracle-based procedure for generation of elementary table translations. We begin by defining a useful relationship between functions.

\textbf{Definition 7.2.1:} \textit{f} $\rightarrow$ \textit{g}

Let \textit{f} and \textit{g} be functions of \textit{k} arguments on \textit{A*}.

\[ f \rightarrow g \]

iff

\[ f(x_1,\ldots,x_k) \text{ defined} \Rightarrow \]

i) \textit{g}(\textit{x_1,\ldots,x_k}) \text{ defined}

ii) \textit{f}(\textit{x_1,\ldots,x_k}) = \textit{g}(\textit{x_1,\ldots,x_k})

for all \textit{x_1,\ldots,x_k} in \textit{A*}.

\textit{f} $\rightarrow$ \textit{g} essentially says that the domain of \textit{f} is a subset of the domain of \textit{g} and the functions are equal as restricted to the domain of \textit{f}. We say that \textit{g} is an \textit{extension} of \textit{f}. \hfill [*]

\textbf{Definition 7.2.2:} Elementary table translation

Let \textit{D_1} and \textit{D_2} be language definitions where

\textit{D_i}=(\textit{G_i},\textit{S_i}) \quad \textit{G_i}=(\textit{VNi},\textit{VTi},\textit{PRi},\textit{AXi}) \quad \textit{S_i}=(\textit{A},\textit{Ri}) \quad i=1,2
Let TAUDOT: VN1 -> VN2 be any function s.t. TAUDOT(AX1) = AX2

Then an elementary translation table is triple (T1,T2,TAU) where

A) T1 is a finite subset of GEN(G1)
B) T2 is a finite subset of (GEN(G2) U VN2 U VT2)
C) TAU: T1 -> T2 X SEQ(NAT)

so that if TAU(t) = (t',(x1,...,xk)) then
|FR(t')| = k and 0 <= xi <= |FR(t)|
for i = 1,...,k

Then TAU defines an elementary table translation iff

i) GEN(T1) includes all but a finite number of trees in GEN(G)

ii) For all t in T1, TAU(t) = (t', (x1,...,xm)) m = |FR(t)|

where 1) ROOT(t') = TAUDOT(ROOT(t))

2) 0 <= xi <= n i=1,...,m |FR(t)| = n

3) xi != 0 => FR(t',i) = TAUDOT(FR(t,xi)) i=1,...m

4) xi = 0 => FR(t',i) in VT2

5) There is a function h_t: A* X...X A* -> A* X...X A* of n arguments into m values

with h_t(y1,...,yn) = (z1,...,zm) yi in A* i=1..n

and zi = yxi if xi != 0

zi in RAN(fR(t)) if xi = 0

so that SEMFUN_t —> SEMFUN_t* % h_t

We note that xi = 0 iff FR(t',i) in VT2.

[•]

TAU* and TAUBAR can be defined for elementary translation tables in a manner analogous to that of Chapter 4.

This somewhat complicated definition asserts that the semantic function of source and target entries in TAU are closely related via the index vector (which provides for rearrangement of arguments and fixing of the values of certain arguments). Note that TAU provides
only one target tree corresponding to each source tree. This differs from the translation tables developed in Chapter 4. Table translations and elementary table translations are not the same. In [Kr176], Krishnaswamy provides an example of language definitions between which there a table translation, but no elementary table translation. We can analyze translator generation problems as we did in Chapter 5 by identifying translator generation decision sets.

DEFINITION 7.2.3: Elementary table translation decision sets

Let $D_0, D_1, \ldots$ be an acceptable LDS$[V, \text{NAT}, f]$. Then

$$D[\text{ELEM}, \text{AE}] = \{(i, j) \mid \text{there is an elementary translation table inducing an a.e., semantic preserving translation between } D_i \text{ to } D_j\}$$

$$L[\text{ELEM}, \text{AE}] = \{(i, j) \mid \text{there is an elementary translation table inducing an a.e., semantic preserving translation between } L(D_i) \text{ and } L(D_j)\}$$

We note that this chapter focuses on the problem of discovering almost everywhere translations since this was the type of translation developed by Buttelmann in his early work.

7.3. Elementary Table Translation and the Arithmetic Hierarchy

Having identified a new breed of translation table with stricter semantic requirements, it is useful to examine the translator generation decision problems associated with this type of translation. Since it is a variation of table translation studied in Chapter 5, one would not expect positive computability results. However, by placing these problems in the arithmetic hierarchy, we can compare their computability to that of the other problems studied in Chapter 5. In addition, the method used to study the computability of elementary table translation decision problems provides insight into the information needed in an oracle-based approach to the problem. We define a useful index set first.
DEFINITION 7.3.1: EXT

Let \( f_0, f_1, \ldots \) be an acceptable programming system on the natural numbers. We define the set \( \text{EXT} \), a subset of \( \mathbb{N} \times \mathbb{N} \), as follows:

\[
\text{EXT} = \{(i,j) \mid f_i \rightarrow f_j \}
\]

\( \text{EXT} \) provides a listing of all index pairs in which the second function in a pair is an extension of the first element. As a preliminary to the study of the table translation decision sets, we establish the placement of \( \text{EXT} \) in the arithmetic hierarchy.

THEOREM 7.3.1: \( \text{EXT} \) is in \( \Pi_2 \)

PROOF: Let \( f_0, f_1 \) be an acceptable programming system.

Then \( \text{EXT} = \{(i,j) \mid f_1 \rightarrow f_j \} \)

\[= \{(i,j) \mid \text{PRED}(i,j) \} \text{ where PRED is the predicate} \]

\[
\forall n \exists k \exists k' \left( \text{STEP}(i,n,k) > 0 \text{ and STEP}(j,n,k') > 0 \text{ and STEP}(i,n,k) = \text{STEP}(j,n,k') \right)
\]

where \( n, k, k' \) in \( \mathbb{N} \).

Observing that PRED is of the form "\( \forall \exists \)"

we conclude that \( \text{EXT} \) is in \( \Pi_2 \). \[\ast\]

THEOREM 7.3.2: \( \text{EXT} \) is \( \Pi_2 \)-complete.

PROOF: We have shown in Theorem 7.3.1 that \( \text{EXT} \) is in \( \Pi_2 \)

To show that \( \text{EXT} \) is complete we must show that a known \( \Pi_2 \)-complete set, namely \( \{ i \mid f_i \text{ is total} \} \), is reducible to \( \text{EXT} \).

Let \( \langle \rangle \) be a pairing function. We define the partial recursive functions \( g_1 \) and \( g_2 \) from \( \mathbb{N} \) to \( \mathbb{N} \) by:
\[ g_1(<i,j>) = 1 \quad \text{for all } i, j \text{ in } \mathbb{N} \]
\[ g_2(<i,j>) = 1 \quad \text{if } f_1(j) \text{ defined} \]
\[ = \text{undefined otherwise} \]

Using the s-m-n function, we can define recursive functions \( h_1 \) and \( h_2 \) from \( \mathbb{N} \) to \( \mathbb{N} \) such that
\[ f_{h_1}(i)(j) = g_1(<i,j>) \quad \text{for all } i, j \text{ in } \mathbb{N} \]
\[ f_{h_2}(i)(j) = g_2(<i,j>) \]

Since for all \( i \), \( f_{h_1}(i) \) is total and everywhere equal to 1
we have \( f_{h_1}(i) \rightarrow f_{h_2}(i) \) iff \( f_{h_2}(i) \) is total
and everywhere equal to 1
iff \( f_1 \) is total

Thus \( i \in \{ j \mid f_j \text{ total} \} \) iff \( (h_1(i), h_2(i)) \) in \( \text{EXT} \).

From this reduction, we conclude that \( \text{EXT} \) is
\[ \Pi_2 \text{-complete.} \]

Having placed \( \text{EXT} \) in the arithmetic hierarchy we can next locate \( \text{D[ELEM,AE]} \) by using its relationship to \( \text{EXT} \).

**Theorem 7.3.3**: \( \text{D[ELEM,AE]} \) is in \( \sum_3 \).

**Proof**: In [But74], Buttelmann detailed a procedure that, based on information provided by \( \text{EXT} \), found an elementary table translation between two language definitions (if one existed!). The procedure shows that \( \text{D[ELEM,AE]} \) is r.e. using an oracle for \( \text{EXT} \), and hence that \( \text{D[ELEM,AE]} \) must be in \( \sum_3 \). \[ * \]

**Theorem 7.3.4**: \( \text{D[ELEM,AE]} \) is \( \sum_3 \)-complete.

**Proof**: We have shown that \( \text{D[ELEM,AE]} \) is in \( \sum_3 \).

It remains to show that a known \( \sum_3 \)-complete set,
Suppose that $D_0, D_1, \ldots$ is an acceptable LDS[$V, NAT, f]$. We define recursive function $h_1$ and $h_2$ from NAT to NAT so that:

$D_{h_1}(i) = (G, S)$ with $G = (\{Z, Y\}, \{b\}, \{Z \in \{Y\}, Y \in \{b\}, Y \in \{b\}, Z\})$ and $S = (R)$ where

$$\begin{align*}
RAN(f_R(b)) &= \{0\} \quad \text{for all } n, n_1, n_2 \\
f_R(Z \in \{Y\})(n) &= n \quad \text{in NAT} \\
f_R(Y \in \{b\})(\langle n_1, n_2 \rangle) &= n_1 + 1 \\
f_R(Y \in \{b\})(n) &= 0
\end{align*}$$

$D_{h_2}(i) = (G, S)$ with $G = (\{Z, Y\}, \{b\}, \{Z \in \{Y\}, Y \in \{b\}, Y \in \{b\}, Z\})$ and $S = (R)$ where

$$\begin{align*}
RAN(f_R(b)) &= \{0\} \\
f_R(Z \in \{Y\})(n) &= n \quad \text{if } f_i(n) \quad \text{defined} \\
f_R(Z \in \{Y\})(n) &= \text{undefined} \quad o/w \\
f_R(Y \in \{b\})(\langle n_1, n_2 \rangle) &= n_1 + 1 \\
f_R(Y \in \{b\})(n) &= 0 \quad \text{for all } n, n_1, n_2 \quad \text{in NAT}
\end{align*}$$

We observe that $h_1$ is a constant function and that

$L(D_{h_1}(i)) = \{(b^{j+1}, j) \mid j \text{ in NAT}\}$ is unambiguous.

Further, $L(D_{h_2}(i)) = \{(b^{j+1}, j) \mid f_i(j) \text{ defined}\}$ is unambiguous.

I. We now show that if the domain of $f_i$ is cofinite, then there is elementary table translation between $D_{h_1}(i)$ and $D_{h_2}(i)$ inducing an a.e., semantic preserving translation.

Suppose $f_i(j)$ is defined for all $j \geq n$. Consider the TAU suggested in Figure 24. We observe that

$\text{SEMFUN}_{t_1}(\langle x_0, \ldots, x_n \rangle) = \text{SEMFUN}_{t_1}(\langle x_0, \ldots, x_n \rangle)$
That \( t_1 \) and \( t_2 \) have the same semantic function follows from the fact that in either case, the argument supplied to the semantic function of the supertree \( Z\langle Y \rangle \) is at least \( n \). By our assumption the semantic functions of \( Z\langle Y \rangle \) are the same in both language definitions for arguments of this size. The relationship of the other pairs of semantic function follows from the language definitions. It should be clear that \( \text{GEN}(\text{DOM}(\text{TAU})) \) includes all but a finite number of trees generated by the grammar of \( D_{h1}(i) \) and that a \( \text{TAUDOT} \) function in which \( \text{TAUDOT}(Z) = Z \) and \( \text{TAUDOT}(Y) = Y \) is consistent with Definition 7.2.2. The translation induced by this \( \text{TAU} \) associates

\[(b^{j+1},j) \text{ with } (b^{j+1},j) \text{ for all } j \geq n.\]

Hence \( \text{TAU} \) induces an almost everywhere, semantic preserving translation between the two language definitions \( D_{h1}(i) \) and \( D_{h2}(i) \).

II. Now we must show that if there is an elementary table translation inducing an almost everywhere, semantic preserving translation between \( D_{h1}(i) \) and \( D_{h2}(i) \) that the domain of \( f_i \) is cofinite.

Let \( \text{TAU} \) be any elementary translation table from \( T_1 \) to \( T_2 \) as defined in Definition 7.2.2. Since \( \text{GEN}(T_1) \) must generate all but a finite number of trees in \( \text{GEN}(G_1) \) and since there are an infinite number of trees in \( \text{GEN}(G_1) \) with supertree \( Z\langle Y \rangle \) there must be a tree \( t \) in \( T_1 \) of the form shown in Figure 25.

Since \( \text{TAUDOT} \) must be satisfied, \( \text{TAU}(t) \) must include a tree \( t' \) also rooted in \( Z \). From the target grammar, the tree \( t' \) can be a complete tree (in which case the associated index vector is all zeros) or it can be a tree with a single nonterminal on the frontier (in this case the index vector may be all zeros or may have a single nonzero entry). These observations about index vectors result from the fact that index vector elements corresponding to terminals must be 0. The three cases are sketched in Figure 25. We consider each case in turn.
Figure 24: TAU for Theorem 7.3.4
A. Here \( \text{SEMFUN}_t(x_0, \ldots, x_n) = x_0 + n \) for all \( x_0 \) in \( \text{NAT} \)

\[
\text{SEMFUN}_t'(x_0, \ldots, x_m) = m \quad \text{\( f_i(m) \) defined}
\]

\[
\text{SEMFUN}_t' \rightarrow \text{SEMFUN}_t, \text{ would imply that}
\]

\( f_i(m) \) defined and \( x_0 + n = m \) for all \( x_0 \) in \( \text{NAT} \)

This is clearly impossible and so this case cannot hold.

B. Here \( \text{SEMFUN}_t(x_0, \ldots, x_n) = x_0 + n \) for all \( x_0 \) in \( \text{NAT} \)

\[
\text{SEMFUN}_t'(x_0, \ldots, x_m) = K + m \quad \text{\( f_i(K+m) \) defined}
\]

\[
\text{SEMFUN}_t' \rightarrow \text{SEMFUN}_t, \text{ would imply that}
\]

\( f_i(K+m) \) is defined for all \( K \) in \( \text{NAT} \) and further that \( x_0+n = K+M \) for all \( x_0, N \) in \( \text{NAT} \)

This too is clearly not possible.

C. Here \( \text{SEMFUN}_t(x_0, \ldots, x_n) = x_0 + n \) for all \( x_0 \) in \( \text{NAT} \)

\[
\text{SEMFUN}_t'(x_0, \ldots, x_m) = x_0 + m \quad \text{if \( f_i(x_0+m) \) defined}
\]

\[
\text{undefined otherwise}
\]

But \( \text{SEMFUN}_t' \rightarrow \text{SEMFUN}_t, \text{ would imply that}
\]

\( f_i(x_0+m) \) was defined and \( x_0+n = x_0+m \)

for all \( x_0 \) in \( \text{NAT} \).

This could only be true if \( f_i(j) \) defined for all \( j \geq n \) and \( n = m \).

Since C is the only possible consequence of the existence of an elementary translation table, we conclude that \( f_i(j) \) defined for all \( j \geq n \) and hence \( f_i \) is cofinite.

Now from I) and II) we have \( (h_1(i), h_2(i)) \) in \( \text{D[ELEM,AE]} \)

\[
i \text{in} \{ j \mid \text{DOM}(f_j \text{ is cofinite}) \} \]
and this reduction is sufficient to show that all 
\[ \sum_3 \text{-complete sets are reducible to } D[\text{ELEM, AE}] \]. Thus we conclude that \( D[\text{ELEM, AE}] \) is \( \sum_3 \text{-complete}. \) [*

This lengthy proof tells us that the solution to elementary table translation is as "hard" as the almost everywhere translation problems examined in Chapter 5. The procedure for \( D[\text{ELEM, AE}] \) suggested by Buttelmann was a "partial" one that, using EXT for consultation, was able to find elementary table translations if they existed. Should one not exist there was no guarantee that the Buttelmann procedure would halt. We next claim that we can do no better than this "partial" solution, since we cannot use EXT to recursively solve the \( D[\text{ELEM, AE}] \) problem.

**COROLLARY 7.3.5:** \( D[\text{ELEM, AE}] \) is not EXT-recursive.

**PROOF:** We have shown that EXT is \( \Pi_2 \)-complete and 
\[ D[\text{ELEM, AE}] \) is \( \sum_3 \)-complete. Hence \( D[\text{ELEM, AE}] \) is in \( \sum_3 \) and hence cannot be EXT-recursive. [*

Although EXT was not sufficient information to allow us to decide \( D[\text{ELEM, AE}] \), it can be useful in other problems. In the next theorem, we show that EXT is of sufficient power to permit us to decide if a particular function has a domain corresponding to the language of a particular language definition.

**THEOREM 7.3.6.** Let \( D_0, D_1, \ldots \) be an acceptable LDS\([V,NAT,f]\) and COD\( V \) be a coding function.

Then the set \( \{(i,j) \mid \text{DOM}(f_i) = \{<\text{COD}_V(m)>(w,m) \text{ in } L(D_j)\}) \) is EXT-recursive.

**PROOF:** From Theorem 3.5.8, we know there is a recursive function \( h: \text{NAT} \rightarrow \text{NAT} \) such that
\[ t = \]
\[ t' = Y^m (0, \ldots, 0) \]
\[ t' = Y^m (0, \ldots, 0) \]
\[ Z \]
\[ Y^m \]
\[ Y^m \]
\[ Y^m \]
\[ Y^m \]
\[ Y^m \]

**CASE A:**

\[ t' = Y^m (0, \ldots, 0) \]

**CASE B:**

\[ t' = Y^m (0, \ldots, 0) \]

**CASE C:**

\[ t' = Y^m (1, 0, \ldots, 0) \]

*Figure 25: Possible Trees in TAU for Theorem 7.3.4*
From recursive function theory we know the existence of a recursive functions $g$ and $\text{CHAR}$ from $\text{NAT}$ to $\text{NAT}$ such that:

$$\text{DOM}(f_g(k)) = \text{RAN}(f_k)$$

and

$$f_{\text{CHAR}}(k)(n) = \begin{cases} 1 & \text{if } f_k(n) \text{ defined} \\ \text{undefined otherwise} & \end{cases}$$

Thus we have, for all $i, j$ in $\text{NAT}$, that

$$\text{DOM}(f_i) = \{<\text{COD}_V(w), m> | (w, m) \in L(D_j) \}$$

iff

$$\text{DOM}(f_i) = \text{RAN}(f_h(j))$$

iff

$$f_{\text{CHAR}}(i) \rightarrow f_{\text{CHAR}}g^h(j) \text{ and } f_{\text{CHAR}}g^h(j) \rightarrow f_{\text{CHAR}}(i)$$

iff

$$(\text{CHAR}(i), \text{CHAR}g^h(j)) \in \text{EXT} \text{ and } (\text{CHAR}g^h(j), \text{CHAR}(i) \in \text{EXT}$$

This proves that, with an oracle for $\text{EXT}$ we can decide if

$$\text{DOM}(f_i) = \{<\text{COD}_V(w), m> | (w, m) \in L(D_j) \}$$

This theorem suggests that, with the assistance of an oracle for $\text{EXT}$, we can decide if the domain of a partial recursive function is a coding of a language defined by a particular language definition. We can use this to decide the equivalence of language definitions.

**COROLLARY 7.3.7:** Let $D_0, D_1, \ldots$ be an acceptable $\text{LDS}[V, \text{NAT}, f]$. Then the set $\{(i, j) | D_i \text{ equivalent to } D_j \}$ is $\text{EXT}$ equivalent.

**PROOF:** Let $\text{COD}_V g$ and $h$ be the recursive functions used in the last theorem.

Then $D_i$ is equivalent to $D_j$
iff
\[ L(D_i) = L(D_j) \]

iff
\[ \text{DOM}(f_{g \% h}(i)) = \{ \langle \text{COD}_V(w), m \rangle | (w, m) \in L(D_j) \} \]

and thus, since \( g \) and \( h \) are recursive and the question above can be answered by consultation with \( \text{EXT} \) (as seen in the last theorem), we conclude that
\[
\{ (i, j) \mid D_i \text{ is equivalent to } D_j \} \text{ is also } \text{EXT}-\text{recursive.} \quad [*]
\]

Thus \( \text{EXT} \) can be used to establish equivalence of the languages defined by a pair of language definitions. This power can be used in studying the translator generation decision problem for \( L[ELEM, AE] \).

**THEOREM 7.3.8:** \( L[ELEM, AE] \) is in \( \sum_3 \).

**PROOF:** Let \( D_0, D_1, \ldots \) be an acceptable \( \text{LDS}[V, NAT, f] \) and \( \text{EXT} \) as defined in Definition 7.3.1.

From Corollary 7.3.7, we know that for all \( i, j, k, q \) in \( NAT \), that we can "recursively" decide (using \( \text{EXT} \))
\[ L(D_i) = L(D_k) \text{ and } L(D_j) = L(D_q) \]

From Theorem 7.3.3, we know that \( D[ELEM, AE] \) is "recursively enumerable" using \( \text{EXT} \). By appropriately dovetailing this enumeration of pairs \( (i, j) \) in \( D[ELEM, AE] \) with the generation and checking (using \( \text{EXT} \)) of all pairs \( (k, q) \) in \( NAT \) using the test above, we can recursively enumerate using \( \text{EXT} \) the set

\[
\{ (k, q) \mid \text{there is } (i, j) \text{ in } D[ELEM, AE] \text{ and } L(D_i) = L(D_k) \text{ and } L(D_j) = L(D_q) \}
\]

which is \( \{ (k, q) \mid \text{there is an elementary translation table inducing an a.e. semantic preserving translation from } L(D_k) \text{ to } L(D_q) \} \)

= \( L[ELEM, AE] \). Since this set is enumerable using a
set known to be $\Pi_2^1$-complete, we conclude that 
$L[\text{ELEM}, \text{AE}]$ must be in $\sum_3$.

THEOREM 7.3.9: $L[\text{ELEM}, \text{AE}]$ is $\sum_3$-complete.

PROOF: We have shown in Theorem 7.3.8 that $L[\text{ELEM}, \text{AE}]$ is in $\sum_3$. It remains to show that a known $\sum_3$-complete set, namely \{\(i \mid \text{DOM}(f_i)\) is cofinite\} is many-one reducible to $L[\text{ELEM}, \text{AE}]$.

Let $D_0, D_1, \ldots$ be an acceptable LDS[$V, \text{NAT}, f$] and let $h_1$ and $h_2$ be the total functions used in the proof of Theorem 7.3.4.

i) From that proof, if was asserted that:

\[ i \in \{j \mid \text{DOM}(f_j)\ \text{is cofinite}\} \Rightarrow (h_1(i), h_2(i)) \in D[\text{ELEM}, \text{AE}] \]
\[ (h_1(i), h_2(i)) \in D[\text{ELEM}, \text{AE}] \Rightarrow (h_1(i), h_2(i)) \in L[\text{ELEM}, \text{AE}] \text{ since } D[\text{ELEM}, \text{AE}] \text{ is a subset of } L[\text{ELEM}, \text{AE}]. \]

ii) Now suppose $(h_1(i), h_2(i))$ is in $L[\text{ELEM}, \text{AE}]$

\[(h_1(i), h_2(i)) \in L[\text{ELEM}, \text{AE}] \Rightarrow \text{an elementary table translation exists inducing an a.e., semantic preserving translation between } L(D_{h1}(i)) \text{ and } L(D_{h2}(i)) \]
\[ \Rightarrow \text{there is some } n \in \text{NAT such that for all } j \geq n \]
\[ \text{there is a translation of } b^{j+1} \]
\[ \text{(here } L(D_{h1}(i)) = \{(b^{j+1}, j) \mid j \in \text{NAT}\}) \]
\[ \Rightarrow \text{there is some } n \in \text{NAT such that for all } j \geq n \]
\[ f_i(j) \text{ is defined (from the definition of } D_{h2}(i)) \]
\[ \Rightarrow \text{domain of } f_i \text{ is cofinite.} \]
Hence from i) and ii), we have

\[ i \in \{ j \mid \text{DOM}(f_j) \text{ is cofinite} \} \iff (h_1(i), h_2(i)) \in L[\text{ELEM,AE}] \]

This reduction proves then that \( L[\text{ELEM,AE}] \) is \( \sum_3 \)-complete. [*]

This last result is provided for completeness of our analysis of translation problems with respect to the arithmetic hierarchy. It was not the intention of Buttelmann to examine the existence of elementary table translation between languages, but rather between language definitions. It is interesting to note that both \( D[\text{ELEM,AE}] \) and \( L[\text{ELEM,AE}] \) lie at the same level of the hierarchy. This is primarily a result of the powerful knowledge available from oracle EXT.

The contribution of this chapter is theoretical substantiation of Buttelmann's suggestion for translator generation. We have found a partial procedure for performing translator generation when we permit consultation with oracle EXT. In the next section, we examine the question of how many questions may have to be asked of oracle EXT, and of how much work is done by the oracle based procedure outside of its oracle consultations.

### 7.4. Complexity of Oracular Translator Generation

In section 7.3, we found that an oracle-based procedure for translator generation existed. It featured consultation with an oracle for the set \( \text{EXT} \), and produced elementary table translations inducing almost everywhere, semantic preserving translations (if they exist!). Such an oracle-based procedure could be programmed for an oracle Turing machine as defined in Section 7.1. In this section, we look carefully at such an oracle Turing machine and examine two questions:

1. How many times does the oracle Turing machine call on the oracle?

2. How much work does the oracle Turing machine do outside of these calls?

The results provide some idea of the "complexity" of using an oracle to perform translator generation. Before proceeding, we need to formalize some ideas on the operation of an oracle Turing machine.
DEFINITION 7.4.1: CONSULT and HALT for oracle Turing machines

Let $T_0, T_1, \ldots$ be an enumeration of oracle Turing machines and $CON_1, CON_2, \ldots$ be an enumeration of possible configurations of oracle Turing machines. We define sets CONSULT and HALT from $\mathbb{N}$ to $\mathbb{N}$ as follows:

$(i, j)$ in CONSULT if
i) $CON_j$ is a possible configuration of $T_i$
ii) $T_i$, when started in configuration $CON_j$ will eventually enter the "oracle consultation" state of $T_i$

$(i, j)$ in HALT if
i) $CON_j$ is a possible configuration of $T_i$
ii) $T_i$, when started in configuration $CON_j$ does not enter the "oracle consultation" state but eventually will enter the "halt" state.

For $CON_j$ to be a possible configuration of $T_i$ means that states and symbols appearing in $CON_j$ are consistent with those used in $T_i$. [*]

Two observations are in order about this definition. First, the formalisms commonly used to describe Turing machines and their configurations are readily used in an enumeration of them. For instance, normally we use descriptions based on some defining alphabet, an enumeration of Turing machines can be done by enumerating strings on the defining alphabet. A similar approach may be taken for configurations.

A second observation is that we do not need to know what information is provided by the oracle in order to evaluate CONSULT and HALT. CONSULT essentially looks for the next consultation (if there is one!). HALT looks for the halt state or the next consultation (if either of these events occur). In either case, it is not necessary to actually perform a consultation and hence we do not need to know what oracle is being used.

We next examine the computability of the sets CONSULT and HALT.

THEOREM 7.4.1: The sets CONSULT and HALT of Definition 7.4.1 are recursively enumerable.
PROOF: Let $T_0, T_1, \ldots$ be an enumeration of oracle Turing machines and $C_{O0}, C_{O1}, \ldots$ an enumeration of configurations. Informally, whether $(i, j)$ is in $\text{CONSULT}$ involves checking if the state and strings on the tapes given in $C_{Oj}$ are possible for $T_i$ (this can be done by examining the state set and tape alphabet specified by $T_i$). If so, then $C_{Oj}$ provides the information necessary to simulate the operation of $T_i$ starting processing as specified by $C_{Oj}$. If it enters the oracle consultation state, then $(i, j)$ is in $\text{CONSULT}$. $\text{HALT}$ can be similarly computed. 

**Theorem 7.4.2:** The sets $\text{CONSULT}$ and $\text{HALT}$ of Definition 7.4.1 are not recursive.

**Proof:** The set of all normal Turing machines are a subset of the set of all oracle Turing machines (they are ones in which the oracle consultation state does not occur among the possible transitions in the Turing machine programs). If $\text{HALT}$ were recursive, it could be used with these programs to test whether a given normal Turing machine would halt on a given input (check $(i, j)$ where $j$ is the initial configuration). Similarly, we could alter any Turing machine and replace all transitions into a halt state into a transition into the consultation state. This modification would permit $\text{CONSULT}$ to solve the halting problem when the modification was applied to "normal" Turing machines among the oracle Turing machines (if $\text{CONSULT}$ were recursive). Hence, $\text{CONSULT}$ cannot be recursive. 

**Theorem 7.4.3:** The sets $\text{CONSULT}$ and $\text{HALT}$ of Definition 7.4.1 are $\text{EXT}$-recursive.

**Proof:** Since the sets $\text{CONSULT}$ and $\text{HALT}$ have been shown to be recursively enumerable, they characterize sets in $\Sigma_1$ within the arithmetic hierarchy. Since $\text{EXT}$ is in $\Pi_2$-complete, and $\Sigma_1$ is in $\Pi_2$, it must be the case that $\text{CONSULT}$ and $\text{HALT}$ are sets that are $\text{EXT}$-recursive.

We now use these observations about $\text{CONSULT}$ and $\text{HALT}$ in commenting on the number of consultations and amount of work that is done by
our oracle-based procedure for translator generation.

THEOREM 7.4.4: Let $D_0, D_1, \ldots$ be an acceptable LDS[$V, \text{NAT}, f$] and $T_0, T_1, \ldots$ be an enumeration of oracle Turing machines.

Then if $T_k$ is an oracle Turing machine (using EXT as an oracle) that partially decides elementary table translation (i.e. on input $(i, j)$, $T_k$ halts if $(i, j)$ in $D_{\text{ELEM,AE}}$ and $T_k$ does not halt should $(i, j)$ not be in $D_{\text{ELEM,AE}}$)

Then there is:

1) No recursive bound on the number of consultations $T_k$ makes with oracle EXT

2) No recursive bound on the number of steps taken by $T_k$ in finding that $(i, j)$ is in $D_{\text{ELEM,AE}}$ when it halts on $(i, j)$.

PROOF: By way of contradiction, suppose there is a recursive function $G: \text{NAT} \times \text{NAT} \rightarrow \text{NAT}$ such that $G$ bounds the number of times $T_k$ will enter the oracle consultation state $q_C$:

$T_k$ on input $(i, j)$ enters state $q_C$ less than $G(i, j)$ times.

If this were true, then it would be possible to construct another oracle Turing machine, also using EXT as an oracle, that would operate as follows:

On input $(i, j)$ it would:

1. Set $C$ to be the initial configuration.

2. Set $m = 0$

3. Repeat while $(k, C)$ in CONSULT and $m < G(i, j) )$

   3.1. Simulate $T_k$ until it enters the oracle consultation state
   3.2. Consult EXT for the same answer $T_k$ would have received.
   3.3. Set $m = m + 1$
   3.4. Adjust $C$ to the configuration of $T_k$ at this point
4. Consult EXT one more time to see if \((k,C)\) in HALT

5. If \((k,C)\) in HALT then halt and claim \((i,j)\) in \(D[\text{ELEM},\text{AE}]\)

6. Otherwise halt and claim \((i,j)\) not in \(D[\text{ELEM},\text{AE}]\).

At step 4, we know that no more consultations will be made of the oracle and so we can ask EXT about whether \(T_k\) would eventually halt. If it does we include \((i,j)\) in \(D[\text{ELEM},\text{AE}]\); otherwise we see \((i,j)\) not in \(D[\text{ELEM},\text{AE}]\). This modified oracle Turing machine, using EXT as an oracle, is thus able to decide if \((i,j)\) in \(D[\text{ELEM},\text{AE}]\) or not. But this contradicts Theorem 7.3.5. which found that \(D[\text{ELEM},\text{AE}]\) is not EXT-recursive. Thus we conclude that no recursive bound \(G\) can exist on the number of consultations.

Now suppose, by way of contradiction, that there was a recursive bound on the number of steps taken by the oracle Turing machine \(T_k\). This does not include the time taken by the oracle to give an answer to a consultation, but only includes one step to consult the oracle. This would permit the construction of another Turing machine using EXT as an oracle that would simulate \(T_k\) until it either halted or reached the bound on the number of steps it needed to find \((i,j)\) was in \(D[\text{ELEM},\text{AE}]\). Again, this constructed machine would suggest that \(D[\text{ELEM},\text{AE}]\) was EXT-recursive. This too, would contradict Theorem 7.3.5. and hence we conclude there can be no recursive bound on the number of steps taken by \(T_k\) in finding \((i,j)\) in \(D[\text{ELEM},\text{AE}]\). [*]

Thus, we can perform a "partial" oracular table translation using an oracle for EXT to find elementary table translations when they exist; however, we cannot place a bound on how many questions will be asked or how much work will be done outside of these consultations. This suggests that we cannot meaningfully talk about complexity bounds for oracular translator generation when using EXT as an oracle.
8. CONCLUSION

8.1. Summary

In this work we have extended the theory of translation by carefully identifying and investigating the problem of automatic generation of translators. The immediate context for this work was provided by Buttelmann [But74], Pyster [Pys75], and Krishnaswamy [Kri76], who identified formalisms for languages, translation, and translator generation. Continuing in the same theoretical spirit, we have generalized and extended several of their ideas in the course of our work. The primary conclusion of this thesis is that searching for general methods of automatically generating translators is likely to fail. In pursuing this result we believe several contributions were made to translation theory.

We were able to provide a variation of the language definition schemes of Buttelmann, Pyster, and Krishnaswamy that makes more precise the means by which the semantics of a language definition are specified. We demonstrated that one type of language definition system, namely an acceptable LDS, is representative of all language definition systems in the sense that results developed for an acceptable LDS will carry over to other language definition systems. This permitted us to focus our computability studies on one system and be confident our results were generally applicable.

We were able to formalize translation and suggested two basic problems for study - the translator generation problem and the translator generation decision problem. We noted that solutions to these are related in that generating a translator is one way of confirming a translator exists. This permitted us to look primarily at the problem of deciding the existence of translations with the assurance our results were relevant to generating translators.

In the area of computability, we were able to establish, using the vehicle of translator generation decision sets, a framework for assembling the known results about the complexity of translator generation. We added several results that more precisely characterized the computability of translator generation. Significant among these was the placing of several translator...
generation problems in the arithmetic hierarchy. This permits those problems to be compared to other known unsolvable problems.

Given the impossibility of general solutions, we examined the effects of restricting the classes of language definitions for which we desire to decide translation. When we found subclasses of language definitions having certain desirable properties, we discovered that we were unable to decide translator generation for these subclasses. We then looked at several specific restrictions based on ideas from formal languages and computability theory. When most combinations of these restrictions were enforced, they were found to yield classes of language definitions which still had undecidable translator generation problems. It was demonstrated that noncomputable problems arose for both syntactic and semantic reasons. However, a characterization was made of some language definition classes for which we could solve the translator generation and TG decision problems. Included was one class that used extremely simple operations in defining the semantics of a language.

For the classes of language definitions found to have solvable translator generation problems, we analyzed the complexity of their solutions. We found an inherent "hardness" about those problems that implied that any solution would need to use an inordinate amount of resources (such as time) when applied to infinitely many instances of the problems. Although we normally associate high complexity with semantic processing, it was found that even the language definition systems using very simple semantics were seen to have provably intractable solutions. A postscript on these complexity results noted that "bad complexity" instances may arise from the particular choice of definitions for languages.

Finally, we gave a formal sketch of an oracle-based translation scheme that uses an outside source of knowledge to assist in performing translation. We focused on a translator generation scheme suggested by Buttelmann [But74] that used a particular oracle and a particular method of translation. In that case we discovered an oracle-based method of "partially solving" the translator generation problem. However, we found that there is no bound on the work done by such a procedure, nor on how many consultations it would request of the oracle.
8.2. Assessing the Results

We believe there are two primary contributions made by this work. First, a body of computability and complexity results for translator generation have been developed and organized. Secondly, the work implies that any attempt to find formal, procedural solutions to automatic translator generation will likely run afoul of computability or complexity difficulties. The results presented here are undeniably negative. It was not the intent nor the expectation of this work to acquire such a collection of pessimistic comments on translator generation. At each step we were surprised and fascinated by the levels of noncomputability and intractability of the translator generation problems. If anything, these results may be theoretical evidence for what we know from experience in programming and natural language translation - discovering and implementing translations can be difficult.

8.3. The Future

The results in this thesis suggest a search for algorithmic translator generation is destined to be difficult. In the face of this, we suggest some courses for future research in translator generation.

Heuristic Approaches

One alternative is to abandon the search for formal algorithmic solutions that guarantee semantic-preserving translations or total translations. This might involve the discovery of translation heuristics that permit procedures to cut through the exhaustive searching that often leads to the high complexity of a problem. The price paid for using such heuristics might be that all sentences of the source language might not have a translation. Perhaps not all translations will be semantic preserving. Such approximate translation might be acceptable in many cases, especially if the translator generation process using heuristics possessed reasonable complexity.

Translation semantics

We have been faithful to an abstract view of language definition and semantics (we have not looked at semantics for programming languages or for natural languages). Perhaps one could develop a "translation semantics" specially developed with translation in mind. If a class of languages were defined in terms
of these semantics, perhaps the generation of translators for this class could be done efficiently. Such an idea is consistent with some work by Krishnaswamy [Kr76], where he used "identical semantics" as a means of doing translator generation.

Translation between Similar Languages

Often the translation we want to do is not between radically different languages but rather between "dialects" of the same language. This is especially true of the programming languages area, where we often convert programs written in one version of a programming language to equivalent programs in a different version of the same language. Similarity between languages, their syntax and semantics might reduce the work needed to perform translator generation.

Choosing Appropriate Language Definitions

As a final suggestion, some of our results suggest the sensitivity of translator generation to the particular choice of language definitions selected to describe source and target languages. This raises the possibility that translator generation might be possible or tractable if only we could select the "right" language definitions. One aspect of this is the balance between syntax and semantics in a definition. Although this thesis makes some suggestions, it is important to study further the effect the selection of language definitions has on the computability and complexity of translator generation.

Finally, note that this thesis has focused on formal translations and on general procedures for generation of translators. Pitched at this abstract level, it has no direct application to practical problems of translation methodologies. For example, no attempt has been made to study the particular translator generation problems for programming languages. In the spectrum that ranges from the theoretical to the practical, our experience over many years has led us to conclude that finding translators is a difficult practical matter. We believe that this thesis has echoed, on the theoretical end, that translator generation is difficult.
LIST OF REFERENCES


Kri76 KRISHNASWAMY, RAMACHANDRAN *Methodology and Generation of Language Translators*, Ph.d Dissertation, The Ohio State University, 1976.


Pys75 PYSTER, ARTHUR Formal Translation of Phrase Structure Languages, Ph.d Dissertation, The Ohio State University, 1975.


