American Journal of Computational Linguistics *Microfiche 7*

SEMANTIC DIRECTED TRANSLATION

OF

CONTEXT FREE LANGUAGES

H. William Buttelmann
Ohio State University

ABSTRACT

A formal definition for the semantics of a context free language, called a phrase-structure semantics, is given. The definition is a model of the notion that it is phrases which have meaning and that the meaning of a phrase is a function of its syntactic structure and of the meanings of its constituents. Next we give a definition for translation on context free languages. We then study a certain kind of translation on cfl's, which proceeds by translating on the phrase trees of the languages, and is specified by a finite set of tree-replacement rules. We present a procedure which, given a cfg and phrase-structure semantics for a source language and a cfg and phrase-structure semantics for a target language, will (usually) produce the finite set of tree-replacement rules for the translation, if the translation exists. The procedure may be viewed as a computer program which is a translator generator, and which produces another program that is a translator.

TABLE OF CONTENTS

SEMANTIC DIRECTED TRANSLATION[†] OF CONTEXT FREE LANGUAGES

by

H. William Buttelmann
Department of Computer and Information Science
Ohio State University
Columbus, Ohio 43210

0. Introduction.

This paper presents a formal model of the translation of context free languages. The model is admittedly inadequate to provide for all the intricacies and complexities of the problem of language translation. Nevertheless, I hope that practicing applied and computational linguists will find it intuitively satisfying in its simplicity. At the same time, the model should give us a basis for proving some theoretical results about the nature of language translation.

Translation is necessarily concerned with both syntax and semantics, so we begin with a formal definition of semantics for context free grammars. In Section 2, a simple algorithm for translating from one context free language to another is given. The algorithm is "controlled" by a finite set of rules which specify how to replace phrases in the source language with semantically equivalent phrases in the target language. The translation algorithm, it turns out, is straightforward. The key problem is in "finding" the finite set of rules which correctly specify the translation. The main part of this paper, Section 3, is concerned with that problem. Throughout the paper, we assume that grammars and semantics are given. There is nothing in this paper that tells you how to go about writing the "right" grammar and semantics for a given cfl.

[†] An earlier version of this paper was presented to the Eleventh Annual Meeting of the Association for Computational Linguistics at Ann Arbor, Michigan, August, 1973.

This research was supported in part by NSF grant GN-534.1.

Much of the presentation is formal. Some readers may find it helpful to read only through Example 1, and then to peruse Section 4 (Sample Translations) to pick up some intuition, before proceeding with the rest of the paper.

1. Phrase Structure Syntax and Semantics.

I assume the reader is familiar with the notions of "derivation" and "syntax tree" (alias "derivation tree", alias "phrase marker") for cfg's. Several good texts on these subjects are listed in the bibliography.

The definition of semantics which I am about to give is based on the following two simple notions: 1) it is phrases which have meaning (paragraphs, sentences, clauses, and morphs are special cases of phrases), and 2) the meaning of a phrase is a function of its syntactic structure and of the meanings of its constituents. Keeping in mind that a function is nothing but an assignment of elements in its codomain to elements in its domain, this definition will provide for idiomatic and emotive meaning, as well as denotative or referential meaning, provided such meanings are specified in the universe of discourse. I wish to add before giving the definition that, although I have never seen it in this form before, I do not believe this definition of semantics is original with me. I believe it incorporates the notions of semantics in Benson (1970), Knuth (1968, 1971), some statements attributed to Thompson (cf Benson, 1970), and in Tarski (1936). Now the definition:

Definition 1.

Let $G = (V, \Sigma, P, S)$ be a context free grammar where:

V is the finite nonempty vocabulary,

$\Sigma \subseteq V$ is the terminal alphabet,

$S \in (V - \Sigma)$ is the axiom, and

P is the finite nonempty set of grammar rules, having the form

$$A \to \beta, \quad \text{for } A \in (V - \Sigma) \text{ and } \beta \in V^{+}.$$

A phrase-structure semantics for G is a 7-tuple

$\mathcal{S} = (U, M, \mu, X, \Delta, F, R)$, where:

U is a set, the universe of discourse,

$M \subseteq 2^{U}$ is a finite set of atomic morphemes,

$\mu: V \to 2^{M}$ is the vocabulary meaning function,

$X = \{(,), ,, x_1, x_2, \ldots, x_n\}$ for some integer n,

Δ is a finite set of names of partial recursive functions,

F is a finite set of definitions for the partial recursive functions named in Δ,

R is a finite set of semantic rules, with the property that to each grammar rule $A \to B_1 \ldots B_n$ there is assigned one semantic rule, having the form $r_{A \to B_1 \ldots B_n}(x_1, \ldots, x_n) = \rho$, where $\rho \in (M \cup X \cup \Delta)^{+}$, and $r_{A \to B_1 \ldots B_n}(x_1, \ldots, x_n) = \rho$ specifies a partial recursive function:

$$r_{A \to B_1 \ldots B_n}: \mu(B_1) \times \cdots \times \mu(B_n) \to \mu(A).$$

We also require that $X \cap (M \cup \Delta) = \emptyset$.

There is an example on the next page.

Example 1.

Consider a cfg and phrase-structure semantics for well-formed addition expressions over the alphabet $\Sigma = \{I, +\}$. $L(G) = \{I, I+I, I+I+I, \ldots\}$. $G = (V, \Sigma, P, S)$, and $\mathcal{S} = (U, M, \mu, X, \Delta, F, R)$, where:

grammar	semantics
P:	R:
$S \rightarrow S + S$	$r_{S\rightarrow S+S}(x_1,x_2,x_3) = x_2(x_1,x_3)$
$S \rightarrow I$	$r_{S\rightarrow I}(x_1) = \iota(x_1)$
$V = \{S, I, +\}$	$U = N \cup \{f^+\} \cup \{\iota\}$, where:
$\Sigma = \{I, +\}$	N is the set of non-negative integers, and f^+ and ι are recursive functions defined in F below.
	$M = \{N, 1, f^+\}$ †
	$X = \{(,), ,, x_1, x_2, x_3\}$
	$\Delta = \{\iota, f^+\}$
	$\mu(I) = \{1\}$, $\mu(+) = \{f^+\}$, $\mu(S) = N$

F contains just the following definitions:

ι (identity function on $N \rightarrow N$):

$\iota(x) = x$

f^+ (integer addition on $N \times N \rightarrow N$):

0) $f^+(0,y) = y$

1) $f^+(x',y) = (f^+(x,y))'$ (' is the successor fn.)

Note that $r_{S\rightarrow S+S}(x_1,x_2,x_3) = x_2(x_1,x_3)$ does indeed specify a recursive function on $\mu(S)\times\mu(+)\times\mu(S) \rightarrow \mu(S)$, since if x_1 and x_3 are in $\mu(S) = N$ and if x_2 is in $\mu(+) = \{f^+\}$, then $x_2(x_1,x_3) = f^+(x_1,x_3)$ is in N and f^+ is defined primitive recursive.

Before explaining the example, let's first consider what the semantics is used for. We will need the following notation for trees:

0) a is a tree, for all $a \in \Sigma$.

1) $a\langle t_1 \ldots t_n\rangle$ is a tree, for all $a \in \Sigma$ and trees $t_1, \ldots, t_n$.

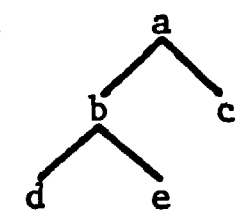
---

† For readability, we write the members of M without unnecessary braces -- i.e., "1" instead of "{1}".

The above inductive definition gives a "standard" parenthesized notation for trees. Let us denote the root of a tree t, rt(t) and the frontier, fr(t). We shall also need the following non-standard notation:

$t_0[t_1 \ldots t_n]$ is a tree if $t_0, t_1, \ldots, t_n$ are trees and if $fr(t_0) = rt(t_1)\ rt(t_2) \ldots rt(t_n)$.

Informally, $t_0[t_1 \ldots t_n]$ is the tree formed by "grafting" each t_i at the i^{th} node of the frontier of t_0, which can be done since this node has the same label as the root of t_i. For example, the tree



is denoted a<b<de>c>, and it has all the following non-standard representations: a<bc>[b<de>c], a<b<de>c>[dec], and a[a<bc>][b<de>c]. As the reader can see, the "box notation" is useful for isolating any rooted subtree. In particular, note that S<S+S>[S<I>+S<I>] is the syntax tree S<S<I>+S<I>> of the grammar of Example 1, with its dominating subtree S<S+S> isolated.

Now back to the semantics. The semantic rules R are used to define a function ϕ on the trees of the grammar which assigns to each syntax tree t a semantic function $\phi(t)$. Then ϕ and the meaning function μ are used to define a meaning function μ on the sentences of G. First, we define ϕ, then μ.

To define ϕ, we must first define the codomain of ϕ, Φ. Informally, Φ is the set of all n-ary functions on $2^U \times \cdots \times 2^U \to 2^U$, for arbitrary n. Formally, let $\Phi^n = \{f: 2^U \times \cdots \times 2^U \to 2^U \quad f \text{ is a function of n arguments}\}$. Then $\Phi = \bigcup_{n=1,2,\ldots} \Phi^n$.

The function $\phi: T(G) \to \Phi$ assigns to each t in $T(G)^+$ a semantic function $\phi(t)$ on $\mu(B_1) \times \cdots \times \mu(B_n) \to \mu(rt(t))$, where $B_1 \ldots B_n = fr(t)$. To specify a semantic function we will use the notation $f(x_1, \ldots, x_n): D \to C$, where f is the name of the function, $(x_1, \ldots, x_n)$ is the vector of arguments, D is the domain, and C is the codomain. ϕ is defined by the following inductive definition:

$^+$ T(G) is the set of syntax trees (partial and complete) of G.

0) $\phi(a)(x):\mu(a) \to \mu(a) = \iota(x):\mu(a) \to \mu(a)$, where $a \in V$,

$\phi(A\langle B_1 \ldots B_n\rangle)(x_1,\ldots x_n):\mu(B_1)\times\cdots\times\mu(B_n) \to \mu(A)$

$= r_{A\to B_1\ldots B_n}(x_1,\ldots,x_n):\mu(B_1)\times\cdots\times\mu(B_n) \to \mu(A)$, where

$A \to B_1\ldots B_n$ is a grammar rule.

1) Let $t = t_0[t_1\ldots t_n] \in T(G)$,

and let $B_{i_{j-1}+1}\ldots B_{i_j} = fr(t_j)$ for $j = 1,\ldots,n$, where $i_0 = 0$.

Then $\phi(t_0[t_1\ldots t_n])(x_1,\ldots,x_{i_n}):\mu(B_1)\times\cdots\times\mu(B_{i_n}) \to \mu(rt(t_0))$

$= \phi(t_0)(\phi(t_1)(x_1,\ldots,x_{i_1}):\mu(B_1)\times\cdots\times\mu(B_{i_1}) \to \mu(rt(t_1)),$

$\phi(t_2)(x_{i_1+1},\ldots,x_{i_2}):\mu(B_{i_1+1})\times\cdots\times\mu(B_{i_2}) \to \mu(rt(t_2)),$

$\ldots,$

$\phi(t_n)(x_{i_n+1},\ldots,x_{i_n}):\mu(B_{i_{n-1}+1})\times\cdots\times\mu(B_{i_n}) \to \mu(rt(t_n)))$

$: \mu(rt(t_1))\times\cdots\times\mu(rt(t_n)) \to \mu(rt(t_0))$

Intuitively, the semantic function assigned to each tree t is the composition of the semantic functions assigned to the subtrees of which t is composed. We leave it to the reader to verify that ϕ is well-defined.

The meaning function μ on sentences is a special case of the meaning function μ on a larger domain -- the set of phrase forms of the grammar. A phrase form is similar to a sentential form, except that it need not be derived from the axiom. Formally, the set of <u>phrase forms</u> of G is the set

$$P(G) = \{ w \mid w \in V^* \text{ and } A \overset{*}{\Rightarrow} w \text{ for some } A \in V\}.$$

The function ϕ is used to define the meaning function as follows. The function $\mu:P(G) \to 2^U$ is defined by the following rule: Let $w = w_1\ldots w_n$ be a phrase form in P(G) and let w have syntax trees $t_1, \ldots, t_m$. Then the set of meanings of w is the set

$$\mu(w) = \bigcup_{i=1,\ldots,m} \phi(t_i)(\mu(w_1),\ldots,\mu(w_n)).$$

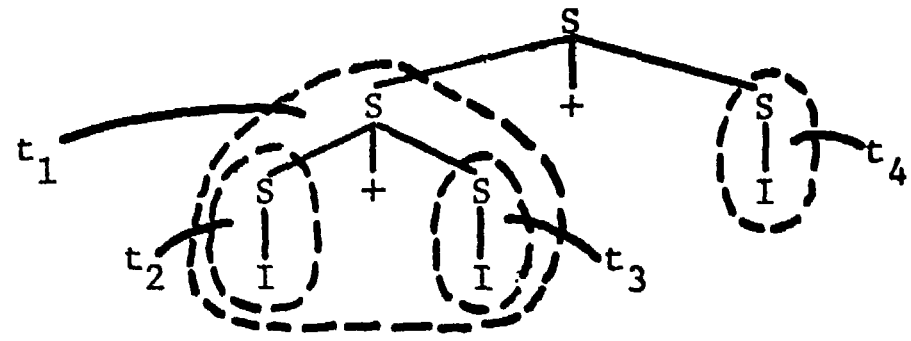
L(G), the language of G, is a subset of P(G), so the meaning function on sentences, $\mu:L(G) \to 2^U$, is just the restriction of μ to L(G).

Since the three functions μ, $\hat{\mu}$, and μ have disjoint domains, they can never be confused, so we shall write μ for all three. μ is the meaning function, which assigns to each sentence, phrase form, and symbol, one or more meanings according to the semantics $\mathcal{S}$. Thus, we are assigning meaning to a sentence by assigning to it the meanings which are computed by the semantic functions specified by its phrase structures, taking as arguments the meanings of the constituents of the sentence. The most elementary constituents of a sentence are the members of Σ which constitute it. One may think of these as the lexical items of the language. Their meanings, which are the arguments of the semantic function, are among the morphemes of the language -- those morphemes which cannot be further separated into morphemes (this is the set of "atomic morphemes", M). Thus, the meaning of a sentence is a function of its morphemes. Which function to use is determined by its syntactic structure. A sentence can be semantically ambiguous if it has more than one syntax tree or if at least one of its constituents is semantically ambiguous.

We return to Example 1 on the next page.

Now consider Example 1. Let w be the sentence "I + I + I". It has the syntax tree

t =



One meaning of w is $\phi(t)(\mu(I), \mu(+), \mu(I), \mu(+), \mu(I))$. For notational purposes, let t_1, t_2, t_3, and t_4 be the subtrees of t circled in the picture. Now compute this member of $\mu(w)$:

$\phi(t)(\mu(I), \mu(+), \mu(I), \mu(+), \mu(I))$

$= \phi(S\langle S+S\rangle[t_1+t_4])(\mu(I), \mu(+), \mu(I), \mu(+), \mu(I))$

$= \phi(S\langle S+S\rangle)(\phi(t_1)(\mu(I), \mu(+), \mu(I)), \phi(+)(\mu(+)), \phi(t_4)(\mu(I)))$

$= \phi(S\langle S+S\rangle)(\phi(S\langle S+S\rangle[t_2+t_3])(\mu(I), \mu(+), \mu(I)), \phi(+)(\mu(+)), \phi(S\langle I\rangle)(\mu(I)))$

$= \phi(S\langle S+S\rangle)(\phi(S\langle S+S\rangle)(\phi(t_2)(\mu(I)), \phi(+)(\mu(+)), \phi(t_3)(\mu(I))), \phi(+)(\mu(+)),$
$\phi(S\langle I\rangle)(\mu(I)))$

$= \phi(S\langle S+S\rangle)(\phi(S\langle S+S\rangle)(\phi(S\langle I\rangle)(\mu(I)), \phi(+)(\mu(+)), \phi(S\langle I\rangle)(\mu(I))),$
$\phi(+)(\mu(+)), \phi(S\langle I\rangle)(\mu(I)))$

$= r_{S\to S+S}(r_{S\to S+S}(r_{S\to I}(\mu(I)), \iota(\mu(+)), r_{S\to I}(\mu(I))), \iota(\mu(+)), r_{S\to I}(\mu(I)))$

$= r_{S\to S+S}(r_{S\to S+S}(\iota(\mu(I)), \iota(\mu(+)), \iota(\mu(I))), \iota(\mu(+)), \iota(\mu(I)))$

$= r_{S\to S+S}(r_{S\to S+S}(\iota(1), \iota(f^+), \iota(1)), \iota(f^+), \iota(1))$

$= r_{S\to S+S}(r_{S\to S+S}(1, f^+, 1), f^+, 1)$

$= r_{S\to S+S}(f^+(1, 1), f^+, 1)$

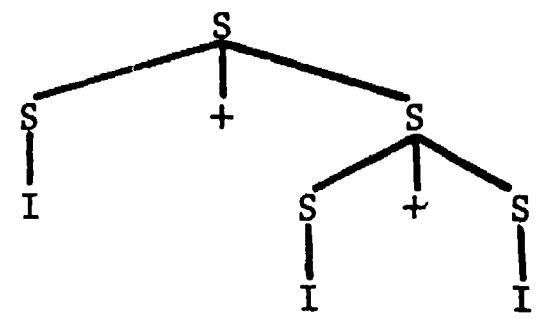
$= f^+(f^+(1, 1), 1)$

$= f^+(2, 1)$

$= 3$

Note that "I + I + I" also has the syntax tree

t' =



but the sentence is not semantically ambiguous since

$\phi(t)(\mu(I), \mu(+), \mu(I), \mu(+), \mu(I)) = \phi(t')(\mu(I), \mu(+), \mu(I), \mu(+), \mu(I))$

2. Translations As Tree Mappings.

Consider now any two cfg's and their associated semantics, G_1, δ_1 and G_2, δ_2. A translation of $L(G_1)$ to $L(G_2)$ is a function $\tau: L(G_1) \to 2^{L(G_2)}$ defined as follows:

$$\tau(w) = \{ w' \mid \mu_2(w') \cap \mu_1(w) \neq \emptyset \}$$

The codomain of a translation must be the power set of the target language, since every sentence in $L(G_1)$ may have many semantically equivalent sentences in $L(G_2)$. In this paper we focus on translations which are specified by a finite set of rules. For these translations, there is a simple algorithm for computing the translation of any sentence. This section presents the method for giving the finite specification of τ and the algorithm for computing the translation.

In fact, instead of specifying a translation on the languages, we specify a translation on the trees of the syntaxes.

To make precise what is meant by "translations which are specified by a finite set of rules" we introduce the concept of a generating set for trees. Let T_1 and T_2 be two sets of trees with labels from some alphabet Σ. Define the set T_Σ^0 to be the set of all trees with single nodes and labels from Σ, i.e., $T_\Sigma^0 = \{ a \mid a \in \Sigma \}$. Informally, T_1 is a generating set for T_2 just in case every tree in T_2 is either in T_Σ^0 or is constructed of a finite number of trees of T_1, and just in case every tree so constructed is in T_2. Formally, let T be a set of trees with labels from Σ. The set

gen(T) of trees generated by T is defined inductively as follows:

0) $T_\Sigma^0 \subseteq gen(T)$ and $T \subseteq gen(T)$,

1) $t_0[t_1 \ldots t_n] \in gen(T)$ if it is defined,
for all positive integers n,
and for all trees $t_0, t_1, \ldots, t_n \in gen(T)$.

T is a generating set for gen(T). We leave it to the reader to verify that every tree in gen(T) can be written in the form $t_0[t_1 \ldots t_n]$, where $t_0 \in (T \cup T_\Sigma^0)$ and each $t_i \in gen(T)$, for $i = 1, \ldots, n$.

Example 2.

The set of production trees of a cfg is a generating set for the set of all the syntax trees of the grammar. Let $G = (V, \Sigma, P, S)$, let P contain k rules, and let $P = \{ A^{(i)} \rightarrow B_1^{(i)} B_2^{(i)} \ldots B_{n_i}^{(i)} \mid i = 1, 2, \ldots, k \}$.

Then the set of production trees of G is the set

$$T_P = \{ A^{(i)} \langle B_1^{(i)} B_2^{(i)} \ldots B_{n_i}^{(i)} \rangle \mid i = 1, 2, \ldots, k \}$$

The set T(G) of all syntax trees of G is the set $gen(T_P)$.

As a more concrete example, consider the cfg G given by the following rules:

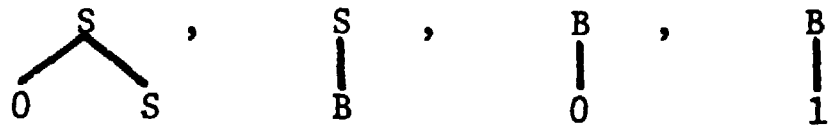
$S \rightarrow 0S$

$S \rightarrow B$

$B \rightarrow 0$

$B \rightarrow 1$

T_P is the set { S<0S>, S<B>, B<0>, B<1> }, or written pictorially:



$T(G) = gen(T_P)$ contains all trees of the following forms:

$S(<0S)^n(>)^n$, $n \geq 0$,

$S(<0S)^n<B>(>)^n$, $n \geq 0$,

$S(<0S)^n<B<0>>(>)^n$, $n \geq 0$,

$S(<0S)^n<B<1>>(>)^n$, $n \geq 0$,

B, 0, 1, B<0>, and B<1>.

The tree t = S<OS<OS<B>>> is in T(G) = gen(T_p) since t = S<OS>[OS<OS<B>>] and S<OS> $\in T_p$; 0 $\in$ gen(T_p), and S<OS<B>> $\in$ gen(T_p). Note that t can also be written as t = S<OS<OS>>[OOS<B>], and again, S<OS<OS>> $\in$ gen(T_p), 0 $\in$ gen(T_p), and S<B> $\in$ gen(T_p).

To specify a translation from $T(G_1)$ to $T(G_2)$ we proceed as follows: Let τ be any partial function on $V_{N_1} \rightarrow V_{N_2}$, and let T be a generating set for $T(G_1)$. Let $\hat{\tau}$ be a function on $T \rightarrow T(G_2) \times N_0^{+\dagger}$ which satisfies the following properties:

if $\hat{\tau}(t) = (t', x_1 \ldots x_n)$ then

i) $rt(t') = \dot{\tau}(rt(t))$, and

ii) $n = |fr(t')|$, and

iii) $0 \leq x_i \leq |fr(t)|$, for i = 1, ..., n, and

iv) $x_i \neq 0 \Rightarrow fr(t')_i = \tau(fr(t)_{x_i})$, for i = 1, ..., n.

Then we define the function τ: $T(G_1) \rightarrow 2^{T(G_2)}$ by the following inductive definition:

0) $t \in T \Rightarrow \tau(t) = t$, where $\hat{\tau}(t) = (t', x)$.

1) $\tau(t_0[t_1 \ldots t_m]) = \tau(t_0)[t'_1 \ldots t'_n]$,

where i) $\hat{\tau}(t_0) = (\tau(t_0), x_1 \ldots x_n)$, and

ii) $$t'_i = \begin{cases} \text{any member of } \tau(t_{x_i}) & \text{if } x_i \neq 0, \\ fr(\tau(t_0))_i & \text{if } x_i = 0. \end{cases}$$

Note that the codomain of τ is the power set of $T(G_2)$ because there may be trees in $T(G_1)$ whose non-trivial factorings into $t_0[t_1 \ldots t_m]$ are not unique. For these trees, $\tau(t) = \{ \dot{\tau}(t_0[t_1 \ldots t_m]) \mid t_0[t_1 \ldots t_m]$ is a representation of t $\}$. As with languages, we will call τ a translation only if it preserves semantics, that is, τ: $T(G_1) \rightarrow 2^{T(G_2)}$ is a translation iff for every tree $t \in T(G_1)$ and for every tree $t' \in T(G_2)$, if $fr(t) = w_1 \ldots w_m$ and $fr(t') = w'_1 \ldots w'_n$, then

$$t' \in \tau(t) \Rightarrow \phi(t)(\mu_1(w_1), \ldots, \mu_1(w_m)) \cap \phi(t')(\mu_2(w'_1), \ldots, \mu_2(w'_n)) \neq \emptyset.$$

† N_0 is the set of non-negative integers.

We will call τ finitely specified (specified by a finite set of rules) iff the generating set T is finite.

Finally, τ is used to define a translation $\tau: L(G_1) \to 2^{L(G_2)}$ as follows: Let $w \in L(G_1)$ have syntax trees $t_1, \ldots t_k$. Then

$$\bar{\tau}(w) = \{ w' \mid \exists\ t' \text{ in } T(G_2) \text{ and } \exists\ t \text{ in } T(G_1) \text{ such that } t' \text{ is a syntax tree of } w' \text{ and } t \text{ is a syntax tree of } w \text{ and } t' \in \tau(t) \}.$$

It follows from the definitions that $\bar{\tau}$ is a translation if τ is. To see this, let $w = w_1 \ldots w_m$ and $w' = w'_1 \ldots w'_n$ and let $w' \in \bar{\tau}(w)$. Then there exist syntax trees t of w and t' of w' such that $t' \in \tau(t)$. Let $\xi_1 = \phi(t)(\mu_1(w_1),\ldots,\mu_1(w_m))$ and $\xi_2 = \phi(t')(\mu_2(w'_1),\ldots,\mu_2(w'_n))$. Then from the definition of $\bar{\mu}$, $\xi_1 \subseteq \mu_1(w)$ and $\xi_2 \subseteq \bar{\mu}_2(w') = \mu_2(\bar{\tau}(w))$. If τ is a translation, then $\xi_1 \cap \xi_2 \neq \emptyset$, so $\bar{\tau}_1(w) \cap \bar{\mu}_2(\bar{\tau}(w)) \neq \emptyset$, and so $\bar{\tau}$ is a translation.

The functions $\dot{\tau}$ and $\hat{\tau}$ are the method for specifving the function τ. The specification is finite just in case the generating set T is finite. The inductive definition for τ gives the algorithm for computing the translation of any tree in $T(G_1)$, and the definition of $\bar{\tau}$, together with this algorithm and a general context free parser such as Floyd's or Early's algorithm, gives the algorithm for computing the translation of any sentence in $L(G_1)$. The next example illustrates.

(In the following and in all subsequent examples, we shall give explicitly only the grammar rules, the right-hand side of the semantic rules, the universes of discourse, the meaning function, and those definitions of partial recursive functions that are necessary. The reader can easily determine the rest of the specifications for the grammars and semantics, if he wishes. For cfg's we shall follow the usual convention that all symbols which do not appear on the left-hand side of some grammar rule are terminal symbols, and that the axiom is the first symbol appearing in the first rule.)

Example 3.

We present two cfg's and their semantics, and a finitely specified translation τ on $T(G_1) \to 2^{T(G_2)}$. To help the intuition, consider that G_1, Δ_1 describes well-parenthesized subtraction expressions, and G_2, Δ_2 describes subtraction expressions in Polish postfix notation.

G_1:	R_1:	G_2:	R_2:
$E \to E-E$	$x_2(x_1,x_3)$	$S \to SS-$	$x_3(x_1,x_2)$
$E \to (E)$	$\iota(x_2)$	$S \to 1$	$\iota(x_1)$
$E \to 1$	$\iota(x_1)$	$S \to 2$	$\iota(x_1)$
$E \to 2$	$\iota(x_1)$	$S \to 3$	$\iota(x_1)$
$E \to 3$	$\iota(x_1)$		

$U_1 = U_2 = N \cup \{f^-\}$, where

N is the set of integers and

f^-: $N \times N \to N$ is ordinary subtraction

$\mu_1(1) = 1$	$\mu_2(1) = 1$
$\mu_1(2) = 2$	$\mu_2(2) = 2$
$\mu_1(3) = 3$	$\mu_2(3) = 3$
$\mu_1(() = \emptyset$	$\mu_2(-) = f^-$
$\mu_1()) = \emptyset$	$\mu_2(S) = N$
$\mu_1(-) = f$	
$\mu_1(E) = N$	

The translation is specified by:

$\dot{\tau}(E) = S$

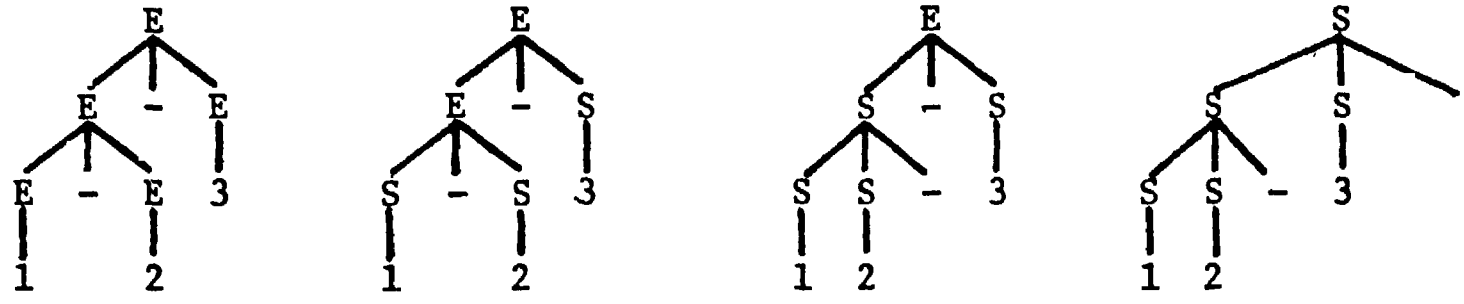
$\hat{\tau}\begin{pmatrix} E \\ | \\ 1 \end{pmatrix} = \begin{pmatrix} S \\ | \\ 1 \end{pmatrix}, 0\Big)$ $\qquad$ $\hat{\tau}\begin{pmatrix} E \\ / \quad \backslash \\ (\quad E \quad) \end{pmatrix} = (S, 2)$

$\hat{\tau}\begin{pmatrix} E \\ | \\ 2 \end{pmatrix} = \begin{pmatrix} S \\ | \\ 2 \end{pmatrix}, 0\Big)$ $\qquad$ $\hat{\tau}\begin{pmatrix} E \\ / \quad | \quad \backslash \\ E \quad - \quad E \end{pmatrix} = \begin{pmatrix} S \\ / \quad | \quad \backslash \\ S \quad S \quad - \end{pmatrix}, 130\Big)$

$\hat{\tau}\begin{pmatrix} E \\ | \\ 3 \end{pmatrix} = \begin{pmatrix} S \\ | \\ 3 \end{pmatrix}, 0\Big)$

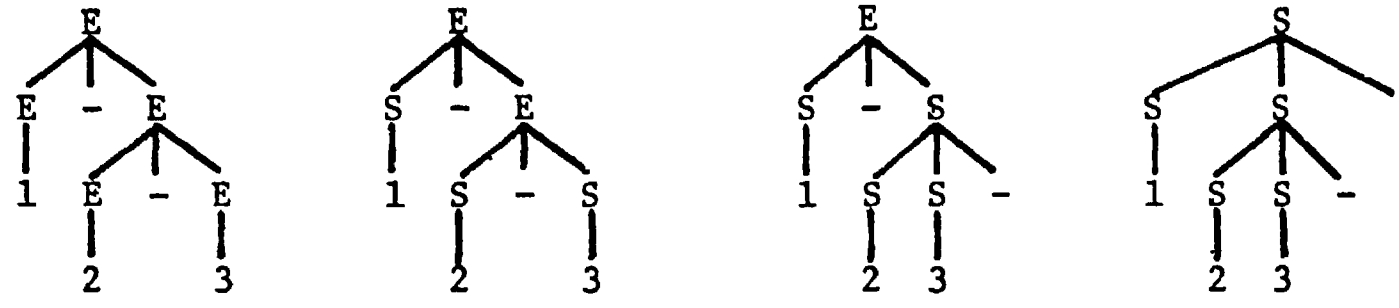
The sentence 1-2-3 is semantically ambiguous (its meanings are 0 and -4), and its two translations are given by:

1)



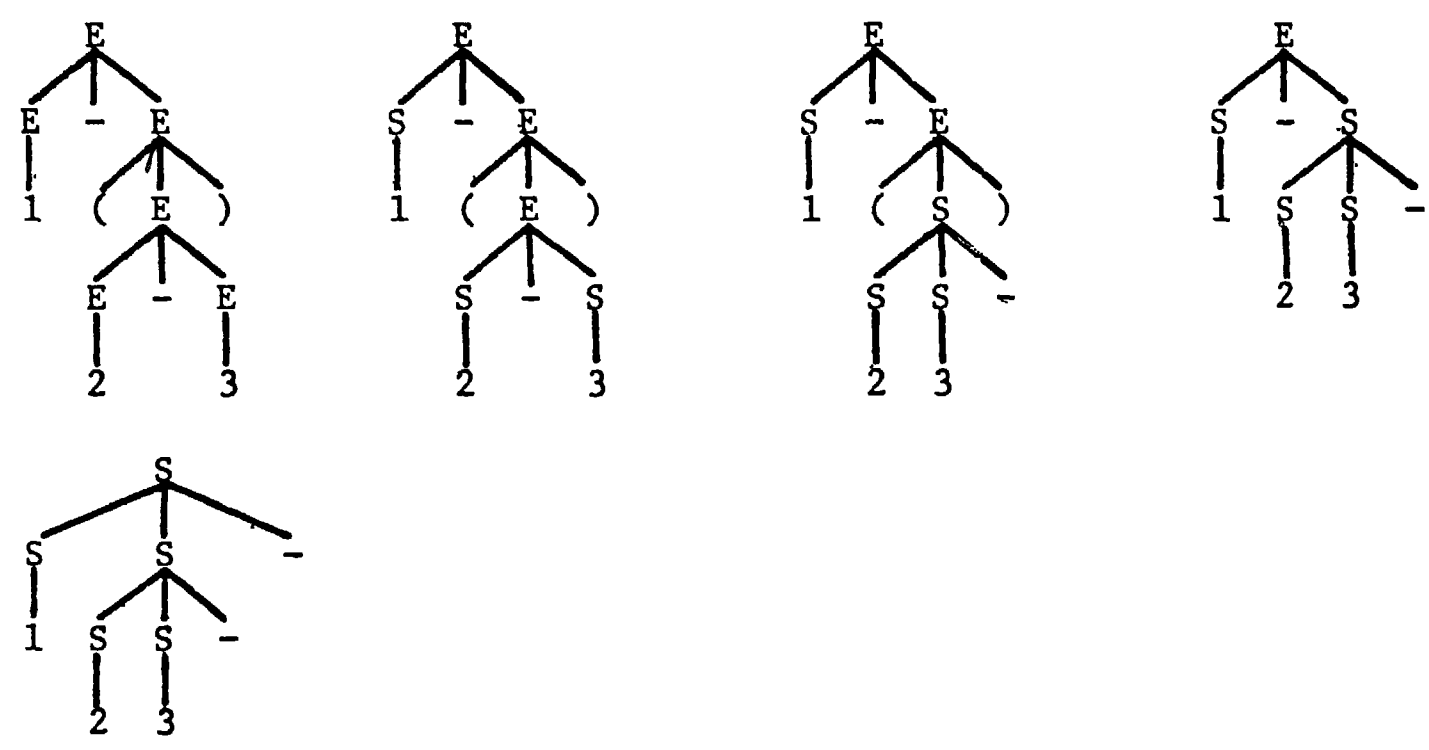
i.e., $\overline{\tau}$(1-2-3) contains 12-3-

2)



i.e., $\overline{\tau}$(1-2-3) contains 123--

On the other hand the sentence 1-(2-3) is unambiguous and is translated:



i.e., $\overline{\tau}$(1-(2-3)) contains 123--

3. A Procedure for Finding Translations (Usually).

Suppose an oracle presents us with two cfg's G_1 and G_2 and their respective semantics $\mathcal{S}_1$ and $\mathcal{S}_2$. Suppose also that a finitely specified translation from $T(G_1)$ to $T(G_2)$ exists. Can we find it? That is, can we

produce the finite set of rules defining the functions $\dot{\tau}$ and $\hat{\tau}$?

In this section we consider a procedure which accepts two arbitrary cfg's and their phrase structure semantics and tries to find a description of such a translation. The procedure may not always work, in that it may not halt or the function τ it describes may be only partial. But τ is guaranteed to be correct; that is, the definitions of $\dot{\tau}$ and $\hat{\tau}$ produced specify a partial function τ: $T(G_1) \to 2^{T(G_2)}$ which is a translation in the sense that, for any $t \in T(G_1)$, if $\tau(t)$ is defined then $\tau(t)$ is a translation of t. First, the procedure is presented; then we give the arguments that τ is a (partial) translation.

Intuitively, the procedure works as follows: We try to find a finite generating set T for $T(G_1)$ and a pair of functions $\dot{\tau}$: $V_{N_1} \to V_{N_2}$ and $\hat{\tau}$: $T \to T(G_2) \times N_0^+$ which have the property that for every tree $t_0 \in T$, if $\hat{\tau}(t_0) = (t_0', x)$, then t_0 and t_0' represent the same semantic function. What is meant by "t_0 and t_0' represent the same semantic function" is just this: If $\hat{\tau}(t_0) = (t_0', x_1 \ldots x_n)$ then $\phi(t_0)(y_1, \ldots, y_m) = \phi(t_0')(y_1', \ldots y_n')$,

$$\text{provided } y_i' = \begin{cases} y_{x_i} & \text{if } x_i \neq 0, \\ \mu_2(fr(t_0')_i) & \text{if } x_i = 0. \end{cases}$$

In general, to get semantic equivalence, one has to be careful how the syntactic variables on the frontier of t_0 are associated by the string x with the syntactic variables on the frontier of t_0', since these represent possible trees with meaning, and hence the domains of the semantic functions for t_0 and t_0'. If such a generating set T and functions $\dot{\tau}$ and $\hat{\tau}$ can be found, the job is finished, since it can then be shown that the function τ defined by $\dot{\tau}$ and $\hat{\tau}$ is a translation.

The procedure begins with the set T_p of production trees of G_1, which is indeed a finite generating set for $T(G_1)$. If the procedure can find a "translation" for each t in T_p, it will be successful, and will halt and output T, $\dot{\tau}$, and $\hat{\tau}$. The procedure systematically picks successive trees t_0 in T_p and searches $T(G_2)$ for a semantically equivalent tree t_0' whose frontier it can match up by some rule x. If it finds one, it outputs the definition $\hat{\tau}(t_0) = (t_0', x)$, deletes t_0 from T_p and tries one of the remaining trees. If it succeeds in exhausting T_p, it is successful.

Suppose, however, that for some t_0 in T_P, the procedure can't find a "translation" in $T(G_2)$. Then if we assume that $\bar{\tau}$ does exist, it must be the case that t_0 is part of a larger tree (or of each of a set of larger trees) which can be "translated". Furthermore, if we also assume that τ is finitely specified, this set is finite. Thus, the procedure tries to construct a new set of trees, T_1, not containing t_0, which is a finite generating set for $T(G_1) - \{t_0\}$. It cannot be the case that the frontier of t_0 is a sentence if we also assume that the existing $\bar{\tau}$ is total on $T(G_1)$. So, losing t_0 from $T(G_1)$ cannot delete any sentences from the language represented by $T(G_1)$. The procedure takes the set T_1 as a new generating set to work with and begins again.

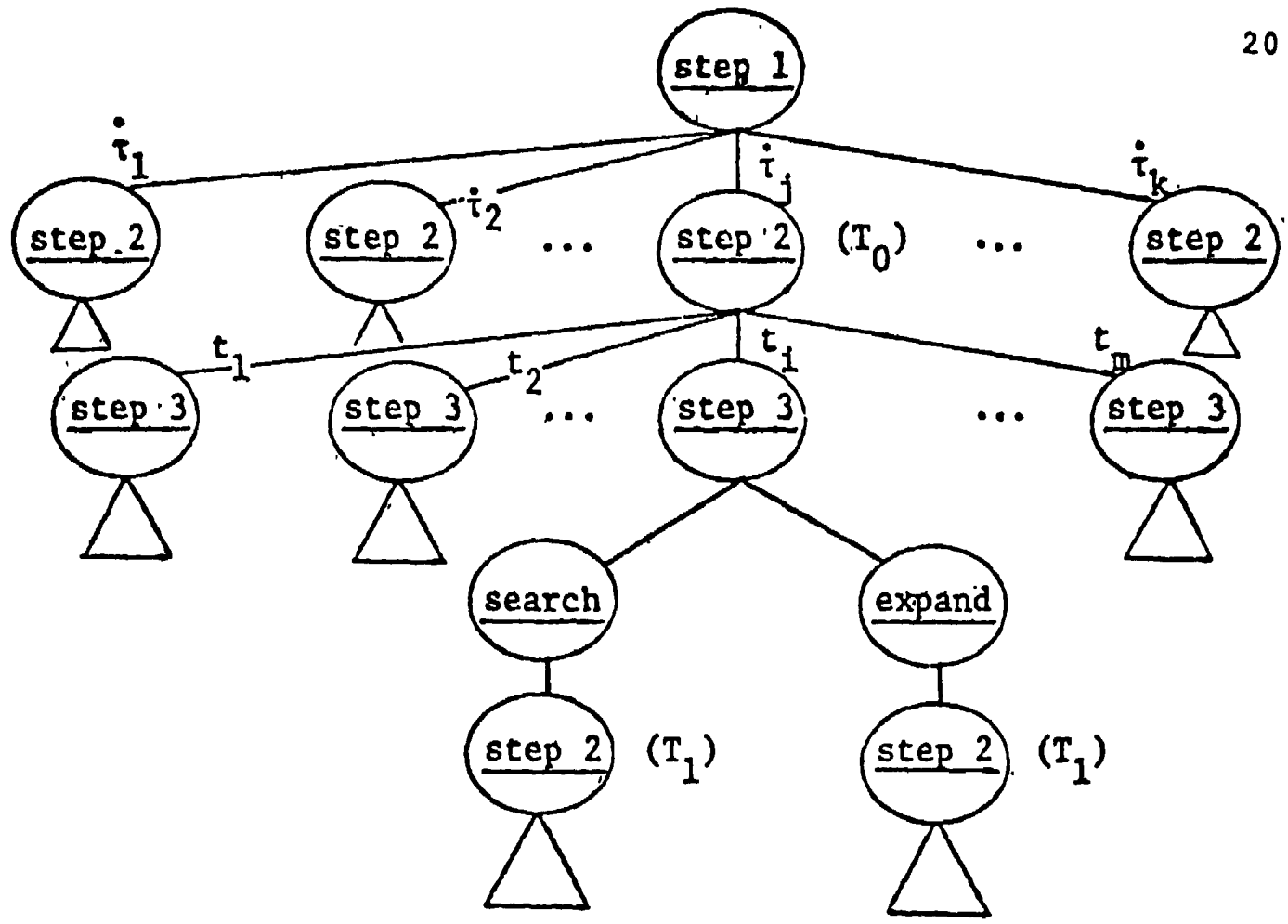
It turns out that finding T depends heavily on the sequence in which successive trees are chosen for translation attempts. Therefore, to guarantee that T will be found if it exists, the procedure tries all possible sequences of trees. The procedure has the general structure of a "tree search", and is represented by the search tree pictured below. Each node in the tree represents a subprocedure which is described below the tree.

PROCEDURE

Given two reduced† cfg's G_1 and G_2 and their respective phrase-structure semantics $\mathcal{S}_1$ and $\mathcal{S}_2$, execute the search tree below for all integer pairs (maxi, maxs) = (1, 1), (1, 2), (2, 1), (2, 2), ... If for any pair step 1 halts and outputs "success", then halt.

---

† Reduced in the sense that each nonterminal symbol is derivable from the axiom and derives terminal strings. It is well-known that every cfg can be put into this form.



step 1:
- Set $i \leftarrow 0$.
- Define the (finite) set of trees

$$T_0 = \{A\langle B_1B_2\ldots B_n\rangle \mid A \rightarrow B_1B_2\ldots B_n \in P\}.$$

- Define the (finite) set of all possible partial functions $\{\dot{\tau}_1, \dot{\tau}_2, \ldots, \dot{\tau}_k\}$ such that for each $j = 1, 2, \ldots, k$,

$\dot{\tau}_j : V_{N_1} \rightarrow V_{N_2}$ and $\dot{\tau}_j(S_1) = S_2$ and for all $A \in V_{N_1}$, $\mu_1(A) = \mu_2(\dot{\tau}_j(A))$.

- Execute step 2 for each function $\dot{\tau}_j$ (i.e., for each $j = 1, 2, \ldots, k$). If for some j the execution of step 2 returns "success", then halt and output "success". If step 2 returns "fail", increase j and continue. If step 2 returns "fail" for all $\dot{\tau}_j$ (i.e., for all j), then halt and output "fail".

step 2: (N.b., T_i is a finite set)
- If $T_i = \emptyset$ then return "success".
- If $i >$ maxi then return "fail".
- Otherwise execute step 3 for each $t \in T_i$. If the execution of step 3 returns "success", then return "success". If step 3 returns "fail", then pick the next t in T_i and execute step 3

again. If step 3 returns "fail" for all t in T_i, then return "fail".

step 3:
- Execute search.
- If search returns "fail" then execute expand.
- If expand returns "fail" then return "fail" to step 2.
- If either search or expand returns "success" then return "success" to step 2.

search:
- Generate the first (maxs) trees of $T(G_2)$:

$$T' = \{t'_1, t'_2, \ldots, t'_{maxs}\}.$$

- Test each tree t' in T' to see if it satisfies each of the following properties:

i) $\dot{\tau}(rt(t)) = rt(t')$

ii) There is a string of non-negative integers $x = x_1x_2\ldots x_n$ such that each of the following is true:

a) $n = |fr(t')|$

b) $x_i \neq 0 \Rightarrow fr(t')_i = \dot{\tau}(fr(t)_{x_i})$ for $i = 1, 2, \ldots, n$

c) $x_i = 0 \Rightarrow fr(t')_i \in \Sigma_2$ for $i = 1, 2, \ldots, n$

d) $\phi(t)(y_1,\ldots,y_m) = \phi(t')(y'_1,\ldots,y'_n)$

$$\text{provided } y'_i = \begin{cases} y_{x_i} & \text{if } x_i \neq 0 \\ \mu_2(fr(t')_i) & \text{if } x_i = 0. \end{cases}$$

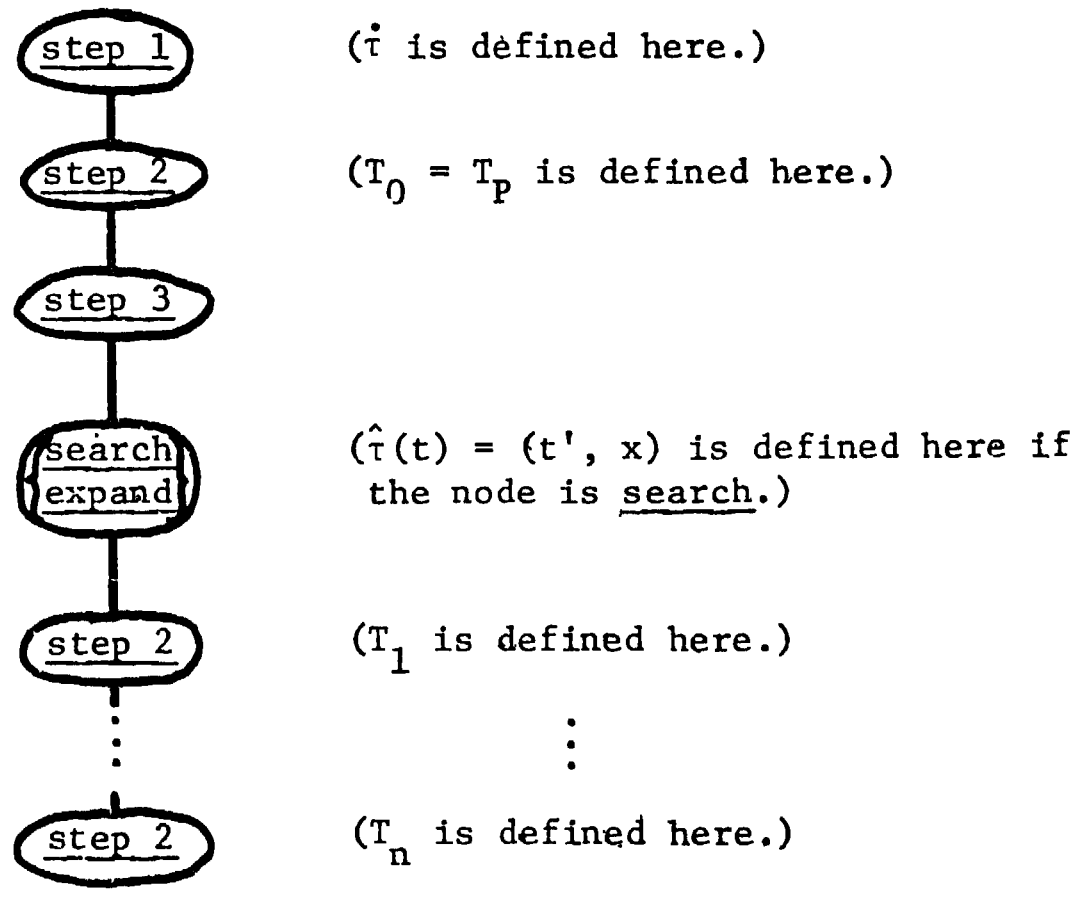
- If no such tree t' exists in T', then return "fail" to step 3.
- If such a tree t' does exist in T' then
  - define $\hat{\tau}(t) = (t', x)$
  - set $i \leftarrow i+1$
  - define the set $T_i = (T_{i-1} - \{t\})$
  - execute a new version of step 2.
- If step 2 returns "success" return "success" to step 3.
- If step 2 returns "fail" return "fail" to step 3.

expand:
- Let $\mathrm{Dom}(\hat{\tau})$ denote the domain of the current version of the function $\hat{\tau}$, i.e., $\mathrm{Dom}(\hat{\tau}) = \{t \mid \hat{\tau}(t)$ has been defined by some execution of search in the current path of the search tree$\}$. Let T_i' denote the (finite) set $(T_i \cup \mathrm{Dom}(\hat{\tau}))$. Define the set T_t to be the smallest set of trees which is a generating set for $(\mathrm{gen}(T_i') - \{t\})$, and which contains the set $(T_i' - \{t\})$. (Note that T_t does not contain t.)
- If T_t is not finite, return "fail" to step 3.
- Set $i \leftarrow i+1$.
- Set $T_i \leftarrow T_t$.
- Execute a new version of step 2.
- If step 2 returns "success" return "success" to step 3.
- If step 2 returns "fail" return "fail" to step 3.

END OF PROCEDURE

Now we want to explain how the PROCEDURE defines functions (possibly partial functions) on $T(G_1) \rightarrow 2^{T(G_2)}$ and on $L(G_1) \rightarrow 2^{L(G_2)}$, and prove that the functions are translations. We shall also show that if the PROCEDURE halts, the translation it total, except under certain easily identifiable conditions.

Consider any path in the search tree. It looks like this:



We need to identify two particular sets of trees associated with this path. Both are finite. The first is the domain of the function $\hat{\tau}$, and the second is the set of trees "excluded" by the successive executions of expand. Note that each execution of the subroutine search adds one item to the definition of the function $\hat{\tau}$, and the entire definition of $\hat{\tau}$ is given by the set of all these items defined by executions of search in the path. The domain of $\hat{\tau}$, then, is the set

$$\mathrm{Dom}(\hat{\tau}) = \{t \mid \hat{\tau}(t) \text{ is defined by some execution of the subroutine search in the path}\}.$$

Similarly, each execution of the subroutine expand, in its first step, defines a new set, T_t, which does not contain the tree t. This step has the effect of excluding the tree t from any further consideration in the translation process. The set of all such trees is the set

$$\mathrm{Excl} = \{t \mid T_t \text{ is defined by some execution of the subroutine exclude in the path}\}.$$

Now, the set $(\mathrm{Dom}(\hat{\tau}) \cup T_n)$ is a finite generating set for the set $(T(G_1) - \mathrm{Excl})$, so the functions $\dot{\tau}$ and $\hat{\tau}$ define a partial function $\tau: T(G_1) \to 2^{T(G_2)}$ according to the definition in Section 2. Furthermore, if $T_n = \emptyset$ then $\hat{\tau}$ is total on the generating set, and so τ is total on $(T(G_1) - \mathrm{Excl})$, and this is just the case when the PROCEDURE halts. Since Excl is a finite set, we have the result that τ is defined on all but a finite number of elements in $T(G_1)$, when the PROCEDURE halts.

Since τ is a partial function on $T(G_1) \to 2^{T(G_2)}$, it follows from the definition of $\bar{\tau}$ in Section 2 that $\bar{\tau}$ is a partial function on $P(G_1) \to 2^{P(G_2)}$ and therefore on $L(G_1) \to 2^{L(G_2)}$. Let P(Excl) denote the set of frontiers of the trees of Excl. Note that each member of P(Excl) is a phrase form. Then when the PROCEDURE halts, since τ is total on $(T(G_1) - \mathrm{Excl})$, it follows that $\bar{\tau}$ is total on $(P(G_1) - P(\mathrm{Excl}))$ and on $(L(G_1) - P(\mathrm{Excl}))$. Thus, $\bar{\tau}$ is total on $L(G_1)$ if PROCEDURE halts and if none of the trees excluded by exclude are complete syntax trees. If complete syntax trees are excluded, then their sentences are the only ones for which $\bar{\tau}$ is not defined.

We have left only to show that $\bar{\tau}$ is a translation. The reader may recall that there may be several nontrivial factorings of trees into a form for which τ is defined, and that this may lead to non-unique translations. Furthermore, the languages may be semantically ambiguous. These conditions make the proof that τ is a translation less lucid, so we shall give here the proof for the case where τ is defined for only one factoring of each tree and there is no ambiguity. It will be helpful in the proof to have the following notation: Let $t \in T(G_1)$ have $fr(t) = w_1 \ldots w_m$ and let $t' \in T(G_2)$ have $fr(t') = w'_1 \ldots w'_n$. Then by $\underline{\phi(t) \equiv \phi(t')}$ we mean $\phi(t)(\mu_1(w_1),\ldots,\mu_1(w_m)) = \phi(t')(\mu_2(w'_1),\ldots,\mu_2(w'_n))$.

Now to the proof. Let $\tau(t) = t'$. We wish to show that $\phi(t) \equiv \phi(t')$. Since $t \in gen(Dom(\hat{\tau}))$, t can be written as $t_0[t_1 \ldots t_m]$ where $t_0 \in Dom(\hat{\tau})$ and each of the trees $t_1, \ldots, t_m \in gen(Dom(\hat{\tau}))$. Let $\hat{\tau}(t_0) = (t'_0, x_1 \ldots x_n)$. Then from the definition of τ, $\tau(t) = t'_0[t'_1 \ldots t'_n]$, where for each $i = 1, \ldots, n$,

$$t'_i = \begin{cases} \tau(t_{x_i}) & \text{if } x_i \neq 0, \\ fr(t'_0)_i & \text{if } x_i = 0. \end{cases}$$

For inductive hypothesis, assume that for each $j = 1,\ldots,m$, $\phi(t_j) \equiv \phi(\tau(t_j))$. Then we have:

$$\phi(t)(\mu_1(w_1),\ldots,\mu_1(w_m)) =$$

$$\phi(t_0)(\phi(t_1)(\mu_1(w_1),\ldots,\mu_1(w_{j_1}),\ldots,\phi(t_m)(\mu_1(w_{j_{m-1}+1}),\ldots,\mu_1(w_m))) \text{ and}$$

$$\phi(t')(\mu_2(w'_2),\ldots,\mu_2(w'_n)) =$$

$$\phi(t'_0)(\phi(t'_1)(\mu_2(w'_1),\ldots,\mu_2(w'_{k_1}),\ldots,\phi(t'_n)(\mu_2(w'_{k_{n-1}+1}),\ldots,\mu_2(w'_n))) \text{ and}$$

$$\phi(t_i) \equiv \begin{cases} \phi(t_{x_i}) & \text{if } x_i \neq 0, \\ \phi(fr(t'_0)_i = \mu_2(fr(t'_0)_i) & \text{if } x_i = 0. \end{cases}$$

For $r = 1,\ldots,m$ let $y_r = \phi(t_r)(\mu_1(w_{j_{r-1}+1}),\ldots,\mu_1(w_{j_r}))$, and for $i = 1,\ldots,n$ let $y'_i = \phi(t'_i)(\mu_2(w'_{k_{i-1}+1}),\ldots,\mu_2(w'_{k_i}))$. Then, if we define $j_0 = k_0 = 0$, the result above demonstrates that

$$y_i' = \begin{cases} y_{x_i} & \text{if } x_i \neq 0, \\ \mu_2(fr(t_0')_i) & \text{if } x_i = 0. \end{cases}$$

Thus, by the definition of $\hat{\tau}$ in search,
$\phi(t)(\mu_1(w_1),\ldots,\mu_1(w_m)) = \phi(t_0)(y_1,\ldots,y_m) = \phi(t')(y_1',\ldots,y_n')$
$= \phi(t')(\mu_2(w_1'),\ldots,\mu_2(w_n'))$, or $\phi(t) \equiv \phi(t')$. Thus τ is a translation.

Finally, as we showed in Section 2, since τ is a translation (on the syntax trees), $\bar{\tau}$ is a translation (on the languages).

In programming terminology a generator is a program whose input is a set of parameters and whose output is a specialized program (cf Brooks and Iverson (1969), p. 365). Then PROCEDURE constitutes a "translator generator": its input is two cfg's and their associated phrase structure semantics, and its output is a table of tree transformations which "drives" a standard tree-mapping program. The tree mapping program is designed to be part of a translator system composed of a parser, the tree mapper, and a frontier stripper (see Figure 1). Translation proceeds as follows: Let G_1 and $\mathcal{S}_1$ be the cfg and semantics for the source language L_1, and G_2 and $\mathcal{S}_2$ the cfg and semantics for the target language L_2. The translator is given a sentence w in L_1. The parser produces a parse tree t(w) for w. (If w is syntactically ambiguous, the parser may produce all the parse trees of w.) If t(w) is in the domain of the function τ defined by the tree mapper, the tree mapper will produce $\tau(t(w))$ whose frontier is a sentence u in L_2.

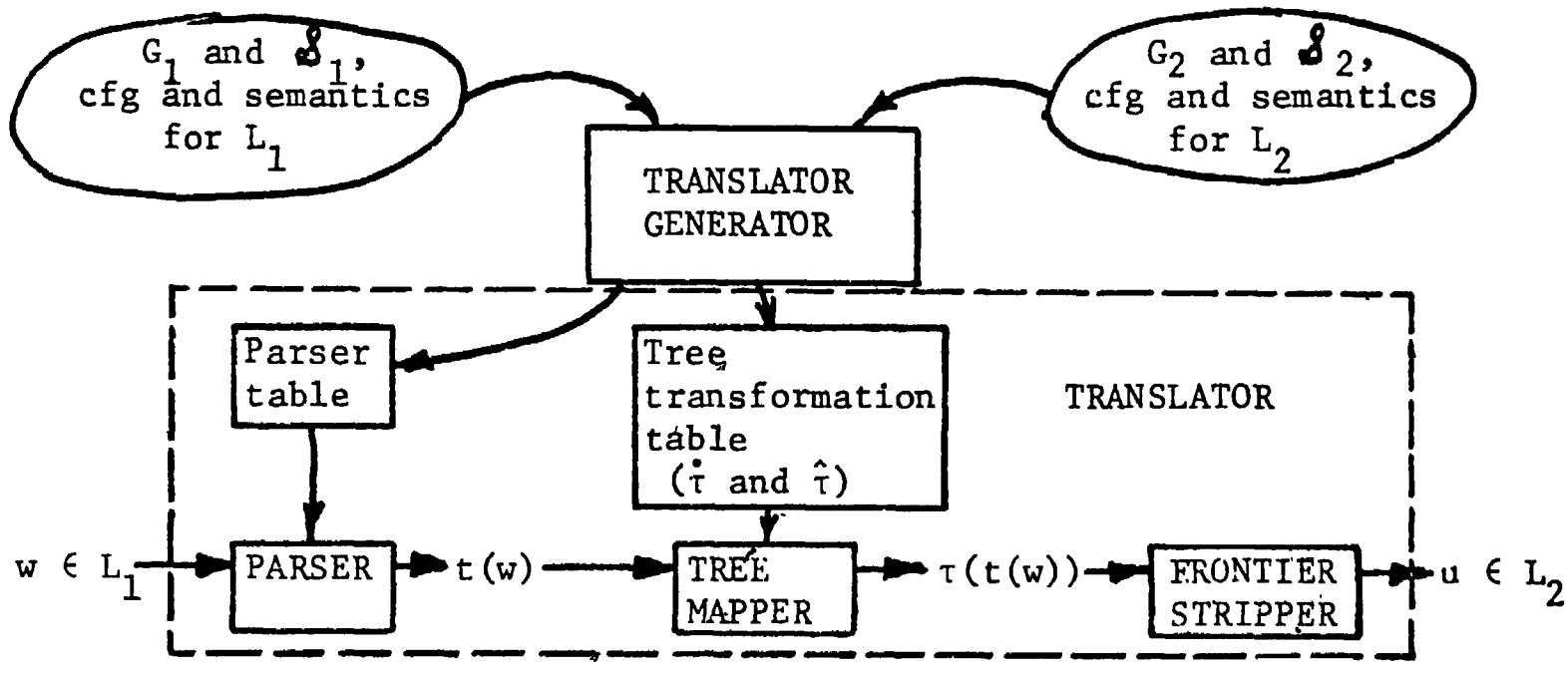


Figure 1. Translator generator and translator.

The importance of the argument that the function defined by the PROCEDURE is a translation, is just that w and u are guaranteed to have the same meaning if they are unambiguous, and if they are ambiguous, w and u are guaranteed to have meanings in common -- i.e., that u is a bona fide translation of w, in the ordinary sense of the word

The usefulness of such a method of translating is that the generator, which has to consider all issues of syntax and semantics, and therefore runs very slowly, need only run once. The translator which it produces should run very fast, since, other than parsing, it only has to transform trees according to the finite set of rules in the tree transformation table (the function $\hat{\tau}$). No semantic computing is required at translate time.

4. Sample Translations.

This section presents some examples of translations on context free languages. The tree search procedure outlined in Section 3 is programmed in CPS and runs on the IBM S/370/165 at Ohio State. All of these translations were "found" by the program.

TRANSLATION I (Postfix to Precedence Infix)

Postfix:		Infix:	
G_1:	$\mathcal{S}_1$:	G_2:	$\mathcal{S}_2$:
Grammar Rules	Semantic Rules	Grammar Rules	Semantic Rules
S → SSO	$x_3(x_1,x_2)$	E → EOT	$x_2(x_1,x_3)$
S → A	$1(x_1)$	E → T	$1(x_1)$
S → B	$1(x_1)$	T → TXF	$x_2(x_1,x_3)$
S → C	$1(x_1)$	T → F	$1(x_1)$
O → +	$1(x_1)$	F → (E)	$1(x_2)$
O → -	$1(x_1)$	F → A	$1(x_1)$
O → *	$1(x_1)$	F → B	$1(x_1)$
O → /	$1(x_1)$	F → C	$1(x_1)$
		O → +	$1(x_1)$
		O → -	$1(x_1)$
		X → *	$1(x_1)$
		X → /	$1(x_1)$

Universe of discourse

$U_1 = U_2 = R \cup F$

R = real numbers = $\{R_1, R_2, R_3, \ldots\}$

$F = \{f^+, f^-, f^*, f^/\}$

Meaning function assigning atomic morphemes to lexical items and syntactic variables:

$\mu_1(A) = R_1$
$\mu_1(B) = R_2$
$\mu_1(C) = R_3$
$\mu_1(+) = f^+$
$\mu_1(-) = f^-$
$\mu_1(*) = f^*$
$\mu_1(/) = f^/$
$\mu_1(0) = F$
$\mu_1(S) = R$

$\mu_2(A) = R_1$
$\mu_2(B) = R_2$
$\mu_2(C) = R_3$
$\mu_2(+) = f^+$
$\mu_2(-) = f^-$
$\mu_2(*) = f^*$
$\mu_2(/) = f^/$
$\mu_2(() = \emptyset$
$\mu_2()) = \emptyset$
$\mu_2(X) = \{f^*, f^/\}$
$\mu_2(0) = \{f^+, f^-\}$
$\mu_2(F) = R$
$\mu_2(T) = R$
$\mu_2(E) = R$

$\Delta_1 = \Delta_2 = \{\iota\}$, and $F_1 = F_2$ contains just the definition: $\iota: N \to N: \iota(x) = x$.

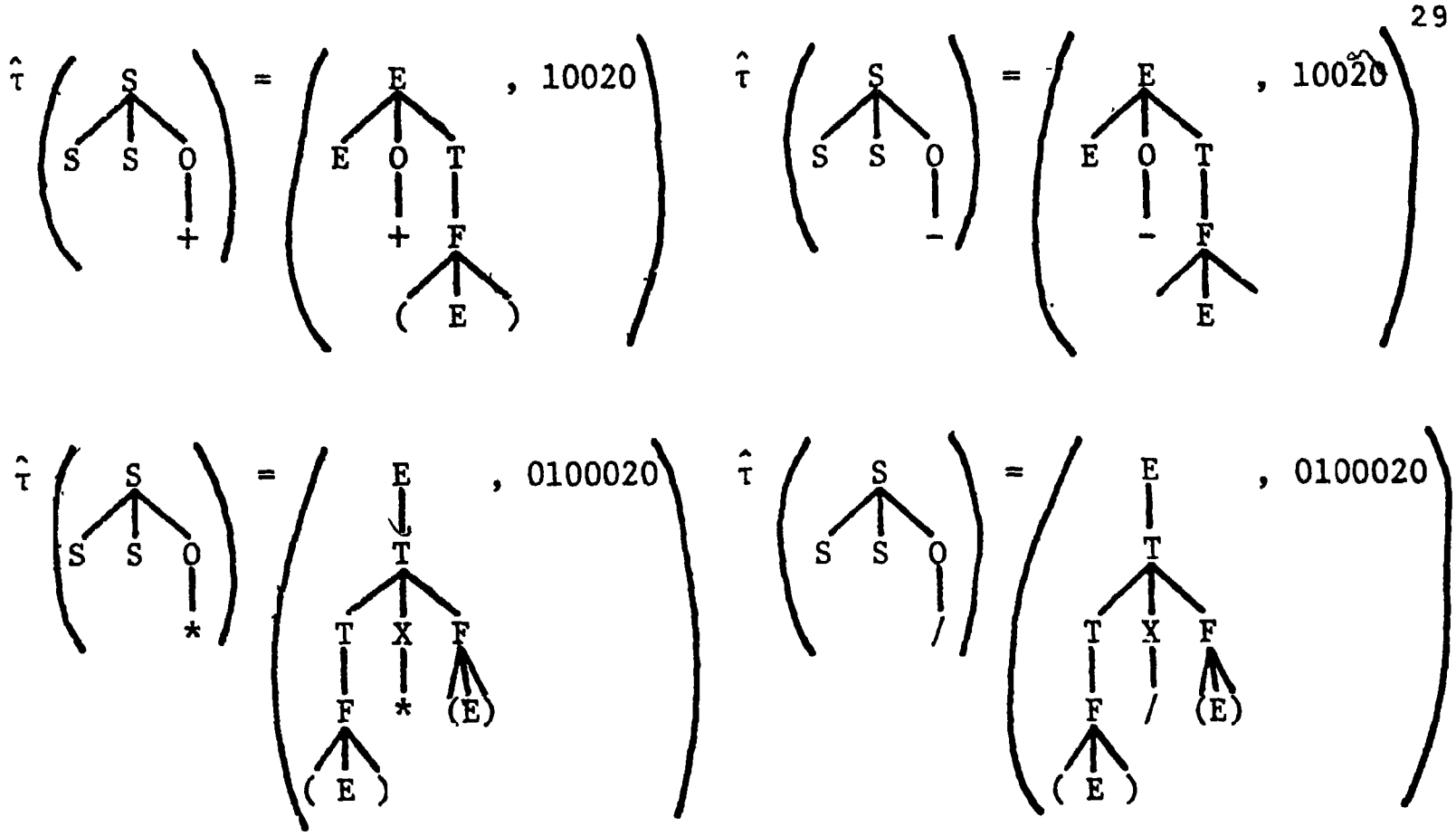
The reader should be able to figure out, after reading the definition in Section 1, that $M_1 = M_2 = \{R, R_1, R_2, R_3, F, f^+, f^-, f^*, f^/\}$ and

$$X_1 = X_2 = \{(,), ,, x_1, x_2, x_3\}.$$

A number of finite specifications for translations are possible. One is:

$\dot{\tau}(S) = E$

$$\hat{\tau}\begin{pmatrix} S \\ | \\ A \end{pmatrix} = \begin{pmatrix} E \\ | \\ T \\ | \\ F \\ | \\ A \end{pmatrix}, \quad 0 \qquad \hat{\tau}\begin{pmatrix} S \\ | \\ B \end{pmatrix} = \begin{pmatrix} E \\ | \\ T \\ | \\ F \\ | \\ B \end{pmatrix}, \quad 0 \qquad \hat{\tau}\begin{pmatrix} S \\ | \\ C \end{pmatrix} = \begin{pmatrix} E \\ | \\ T \\ | \\ F \\ | \\ C \end{pmatrix}, \quad 0$$



It is interesting to note that the PROCEDURE does not have to know how to compute the functions f^{+}, f^{-}, f^{*}, and $f^{/}$ in order to discover this translation. All that is needed is to assume that if a symbol appears in both semantics, it represents the same semantic entity in each, whatever that entity is. For example, consider the two trees in the translation involving "+". Let t = S<SSO<+>>, and t' = E<EO<+>T<F<(E)>>>. All we need to know is that $\phi(t) \equiv \phi(t')$, and it turns out we can find that out without computing f^{+}:

$$\phi(t)(\mu_1(S),\mu_1(S),\mu_1(+))$$

$$= \phi(S<SSO<+>>)(\mu_1(S),\mu_1(S),\mu_1(+))$$

$$= \phi(S<SSO>[SSO<+>])(\mu_1(S),\mu_1(S),\mu_1(+))$$

$$= \phi(S<SSO>)(\phi(S)(\mu_1(S)),\phi(S)(\mu_1(S)),\phi(O<+>)(\mu_1(+)))$$

$$= r_{S\to SSO}(l(\mu_1(S)),l(\mu_1(S)),r_{O\to +}(\mu_1(+)))$$

$$= r_{S\to SSO}(l(\mu_1(S)),l(\mu_1(S)),l(\mu_1(+)))$$

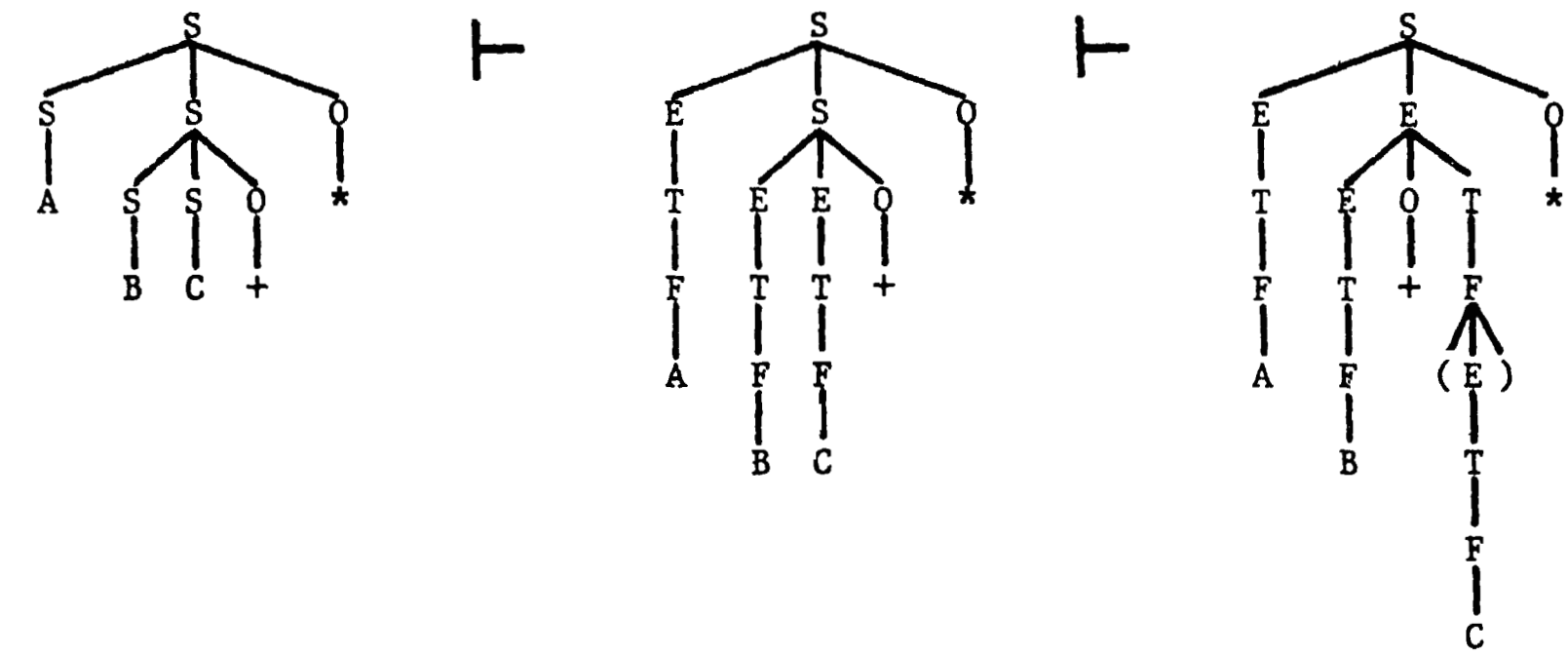
$$= r_{S\to SSO}(\mu_1(S),\mu_1(S),\mu_1(+))$$

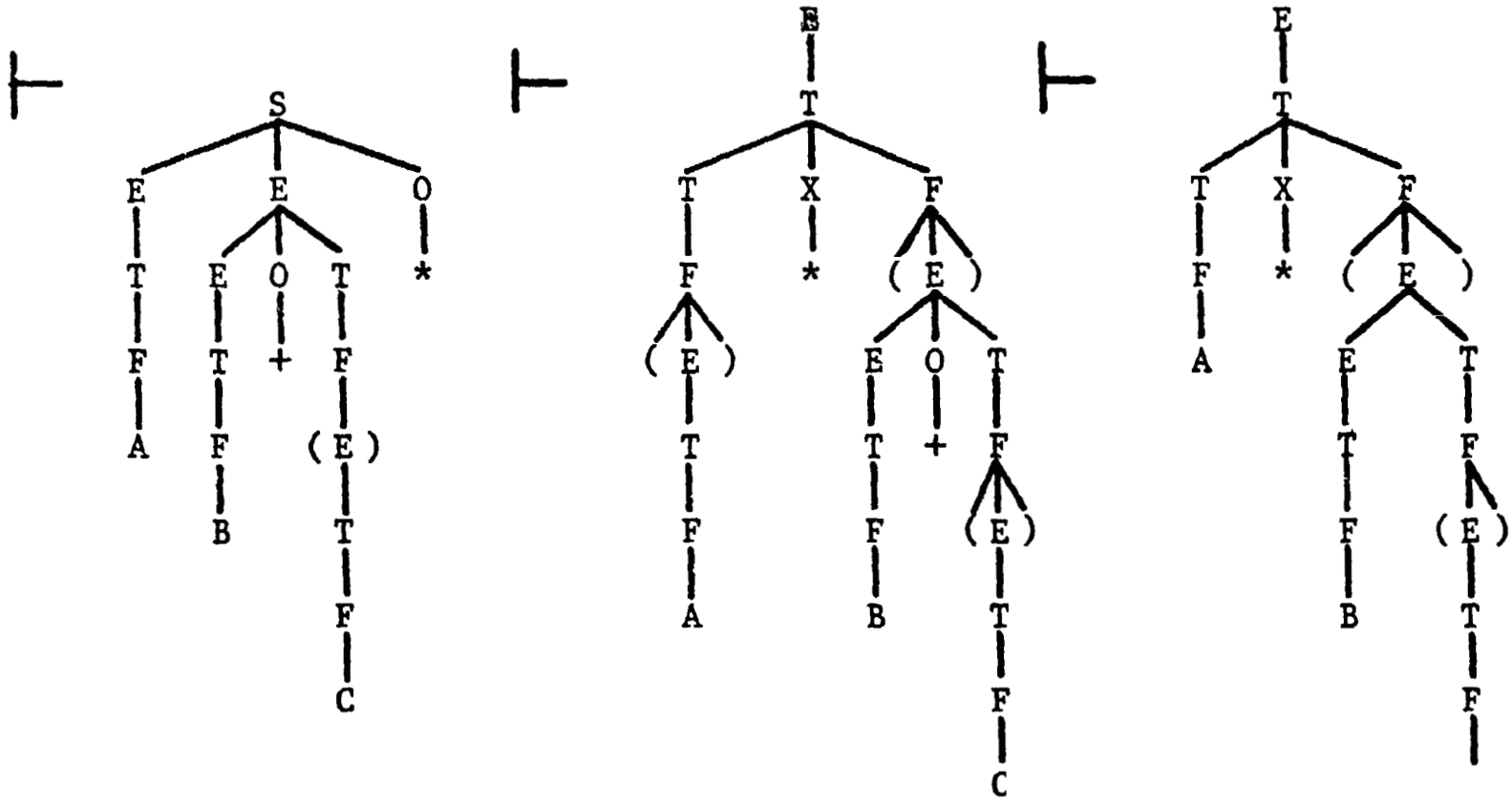
$$= r_{S\to SSO}(R, R, f^{+})$$

$$= f^{+}(R, R)$$

Similarly, $\phi(t')(\mu_2(E),\mu_2(+),\mu_2(()),\mu_2(E),\mu_2())) = f^{+}(R, R)$, so we know that $\phi(t) \equiv \phi(t')$.

Consider now the translation of ABC+*. The following shows that $\overline{\tau}(ABC+*) = A*(B+(C))$:





TRANSLATION II (Explicit * to implicit *)

This translation is interesting because it shows the procedure has the ability to "discover" that a word (*) in $L(G_1)$ has no translation. But it can find a phrase form involving that word which can be translated to a phrase form in $L(G_2)$.

Explicit *:		Implicit *:	
G_1:	$\mathcal{S}_1$:	G_2:	$\mathcal{S}_2$:
$E \to EOE$	$x_2(x_1,x_3)$	$S \to SS$	$f^*(x_1,x_2)$
$E \to A$	$1(x_1)$	$S \to A$	$1(x_1)$
$E \to B$	$1(x_1)$	$S \to B$	$1(x_1)$
$O \to *$	$1(x_1)$		

$$U_1 = U_2 = R \cup \{f^*\}$$

$\mu_1(A) = R$	$\mu_2(A) = R$
$\mu_1(B) = R$	$\mu_2(B) = R$
$\mu_1(*) = f^*$	$\mu_2(S) = R$
$\mu_1(E) = R$	
$\mu_1(O) = f^*$	

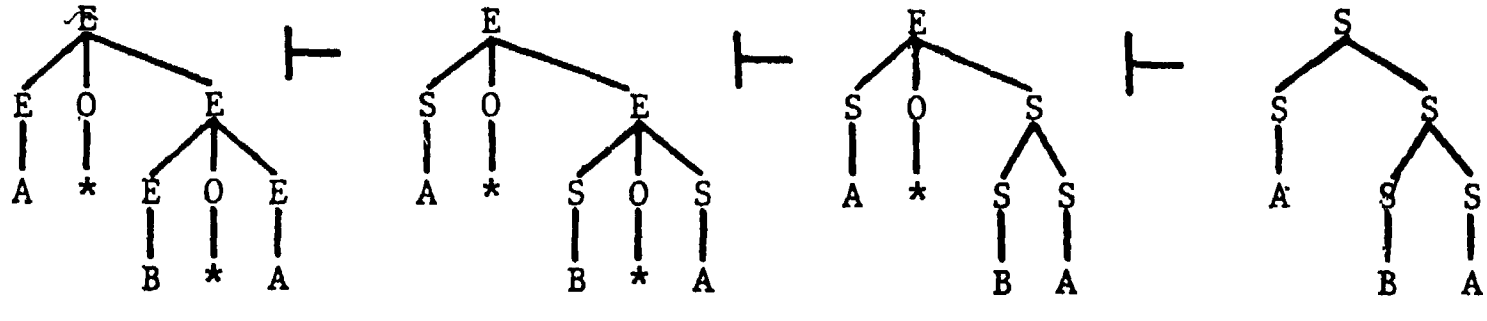
The translation is given by:

$\dot{\tau}(E) = S$

$\hat{\tau}\begin{pmatrix} E \\ | \\ A \end{pmatrix} = \begin{pmatrix} S \\ | \\ A \end{pmatrix}, O$ $\qquad$ $\hat{\tau}\begin{pmatrix} E \\ | \\ B \end{pmatrix} = \begin{pmatrix} S \\ | \\ B \end{pmatrix}, O$

$\hat{\tau}\left(\begin{array}{ccccc} & & E & & \\ & \swarrow & | & \searrow & \\ E & & O & & E \\ & & | & & \\ & & * & & \end{array}\right) = \left(\begin{array}{ccc} & S & \\ \swarrow & & \searrow \\ S & & S \end{array}\right), 13$

$\bar{\tau}(A*B*A) = ABA$:



TRANSLATION III (2, + to 1, +)

$L(G_1)$ is the language of all addition expressions with 2, i.e., the set of all strings of the form 2 + 2 + ... + 2. $L(G_2)$ is the set of all strings of the form 1 + 1 + ... + 1. Under a standard semantics, $L(G_1)$ expresses the even integers and $L(G_2)$ the integers. The procedure "discovers" that the word "2" in $L(G_1)$ must be translated as the phrase "1 + 1" in $L(G_2)$

2,+:		1,+:	
G_1:	$\mathcal{S}_1$:	G_2:	$\mathcal{S}_2$:
$S \to 2+S$	$x_2(x_1,x_3)$	$S \to 1+S$	$x_2(x_1,x_3)$
$S \to 2$	$1(x_1)$	$S \to 1$	$1(x_1)$

$$U_1 = U_2 = N \cup \{f^+\}$$

N = positive integers = {1, 2, ...}

$\mu_1(2) = 2$ $\qquad$ $\mu_2(1) = 1$

$\mu_1(+) = f^+$ $\qquad$ $\mu_2(+) = f^+$

$\mu_1(S) = N$ $\qquad$ $\mu_2(S) = N$

$$\Delta_1 = \Delta_2 = \{1, f^+\}$$

$F_1 = F_2$, which contains the following definition

$f^+:N \times N \to N$ (integer addition):

0) $f^+(0,y) = y$

1) $f^+(x',y) = (f^+(x, y))'$ (' is the successor fn.)

$1:N \to N$ (identity): $1(x) = x$

The translation is specified by:

$\dot{\tau}(S) = S$

$\hat{\tau}\left(\begin{array}{c} S \\ | \\ 2 \end{array}\right) = \left(\begin{array}{c} S \\ 1 \ + \ S \\ \quad\quad | \\ \quad\quad 1 \end{array} \ , \ 000 \right)$ $\qquad$ $\hat{\tau}\left(\begin{array}{c} S \\ 2 \ + \ S \end{array}\right) = \left(\begin{array}{c} S \\ 1 \ + \ S \\ \quad\quad\quad | \\ \quad\quad\quad 1 \ + \ S \end{array} \ , \ 00003 \right)$

Note that to "discover" this translation, the procedure must be able to compute the function f^+, since it needs to know that 2 = 1+1. Consider, for example, $\phi(S\langle 2\rangle)(\mu_1(2)) = r_{S\to 2}(\mu_1(2)) = \iota(2) = 2$, but

$\phi(S\langle 1+S\langle 1\rangle\rangle)(\mu_2(1),\mu_2(+),\mu_2(1)) =$

$\phi(S\langle 1+S\rangle)(\phi(1)(\mu_2(1)),\phi(+)(\mu_2(+)),\phi(S\langle 1\rangle)(\mu_2(1))) =$

$r_{S\to 1+S}(\iota(1),\iota(f^+),\iota(1)) = f^+(1,1) = 2$. To get the last step in the evaluation of the second semantic function, the procedure must be able to compute $f^+(1,1)$.

TRANSLATION IV (1,+ to 2,+)

Suppose the procedure were asked to translate from $L(G_2)$ to $L(G_1)$ in the previous example — i.e., from the integers to the even integers. It would never halt, but it would "discover" that the phrase "1 + 1" is to be translated as the word "2", "1 + 1 + 1 + 1" as "2 + 2", etc. It would define a translation which is total on the strings in $L(G_2)$ whose values are even, and it would continue to look forever for possible translations for the odd-valued strings. We leave it as an exercise for the reader to give the functions $\dot{\tau}$ and $\hat{\tau}$ which define this partial translation.

5. Conclusion and Further Research.

At the present time what is needed more than anything else in the area of language translation is an understanding of the formal nature of semantics, its relation to syntax in language description, and its role in translation. I believe this paper provides some of the basis for that understanding. Incidentally, the reader might have observed that the definition of phrase-structure semantics in Section 1 provides for solutions to the semantic projection problem (cf Katz and Fodor (1964), and Langendoen (1969)).

The reader is certainly aware by now, if not before, that there are many grammars and semantics for a given language. After having played with writing grammars and semantics for simple languages for quite a while now, I believe that, for most languages at least, there are "better" grammars and semantics and "worse" ones. Some just seem to be more elegant or simple, or "natural" than others, for a given language. But I

can't say much of a specific nature about what it means for a grammar and semantics to be "elegant", "simple", or "natural". It seems that some study in this area might give us insight into certain skills for making it easier to write linguistic descriptions suitable for translation.

One phenomenon this model explains is why it is so difficult to compute an inverse translation and get anything like the original. That is, if one starts with sentence w in L_1 and translates to w' in L_2, then translates w' to w" back in L_1, one would like for w and w" to have the same meaning. But the scuttlebutt says it isn't so, and this model shows why. Note that all that is required for $\tau: L_1 \to 2^{L_2}$ to be a translation is that if w' is a translation of w, then $\mu_1(w) \cap \mu_2(w') \neq \emptyset$, i.e., that the source sentence and its translation have <u>some</u> common meaning. Now suppose $\tau': L_2 \to 2^{L_1}$ is also a translation and that $w'' \in \tau'(w')$. Then we have $\mu_1(w) \cap \mu_2(w') \neq \emptyset$ and $\mu_2(w') \cap \mu_1(w'') \neq \emptyset$, <u>but it does not follow</u> that $\mu_1(w) \cap \mu_1(w'') \neq \emptyset$. In order to get back to the original meaning, each translator must produce the entire set $\tau(w)$, rather than just some sentence in $\tau(w)$, and then all of these must be retranslated in entirety. Translation programs don't usually do that. Neither do human translators, for that matter! Alternatively, the translator should be able to give with the translation, its parse and the atomic morphemes associated with the sentence. The procedure in this paper provides for doing that.

The same definition of translation, if it is accurate, also explains another phenomenon of language translation -- how it is that two very different translations can come from the same source. If w' and w" are translations of w, then we have $\mu_1(w) \cap \mu_2(w') \neq \emptyset$ and $\mu_1(w) \cap \mu_2(w'') \neq \emptyset$, but it does not follow that $\mu_2(w') \cap \mu_2(w'') \neq \emptyset$.

For natural language, one would like to extend the theory in this paper to arbitrary phrase structure grammars and to transformational grammars. The extension to transformational grammars requires only

---

† The "lore" has it that someone fed the following sentence to a translator from L_1 to L_2:

"The spirit indeed is willing, but the flesh is weak."

Then he took the translation and fed it into a translator from L_2 to L_1, and got:

"The liquor is all right, but the meat is spoiled."

formalizing the notion of the transform of a semantic function to be associated with each syntax transformation. (For transformational semantic theories which do not allow semantic change in the transformations, the extension to arbitrary phrase structure grammars is sufficient, of course.) The extension to arbitrary phrase structure grammars requires first a formal statement of the "phrase structures" of unrestricted grammars, since these structures are not trees. The author's forthcoming paper, listed in the bibliography, covers the subject of the syntactic structures for unrestricted languages in detail.

There are, of course, schemes for translation other than the one described in this paper. One might think of computing the meaning of a source sentence, and then having some effective way of generating the target sentence directly from the meaning. The scheme in this paper, however, is more attractive at present than such a "direct" scheme, for three reasons: 1) It is intuitively satisfying. I believe I translate by first translating simple phrases and then putting their separate translations together according to some restructuring rules that are guaranteed to preserve semantics. Thus, one "builds up" the translation of a sentence recursively. I am more likely to call the result which I get by first computing the whole meaning and then producing a sentence (often it is a sequence of sentences) with the same meaning, a "paraphrase" or an "interpretation", rather than a "translation". 2) If used much, this scheme is likely to be more efficient than the "direct" scheme, since no semantic computation is required at translate time. All the semantic problems are examined once and for all in the translator generator; at translation time, only a sequence of tree mappings is performed — simply a structure matching and replacing technique. 3) The "direct" scheme requires knowing how to specify linguistic descriptions in such a way that, given a meaning in semantic notation, one can produce a sentence having that meaning. This problem is a difficult one not yet well understood. Presumably, the research currently under way in the field of generative semantics will explicate the issues involved.

Selected Bibliography.

Benson, D. B. (1970). Syntax and semantics: a categorial view. In Information and Control, 16, pp. 738-773.

Benson, D. B. (forthcoming) Semantic preserving translations.

Brooks, F. P., Jr., and Iverson, K. E. (1969). Automatic Data Processing. New York: Wiley.

Buttelmann, H. W. (forthcoming). On the syntactic structures of unrestricted grammars. In Information and Control.

Katz, J., and Fodor, J. A. (1964). The structure of a semantic theory. In Language, 39, pp. 170-210. Reprinted in Fodor and Katz (eds.) The Structure of Language. Englewood Cliffs, New Jersey: Prentice-Hall, pp. 479-518.

Katz, J., and Postal, P. (1964). An Integrated Theory of Linguistic Descriptions. Cambridge: MIT Press.

Hopcroft, J. and Ullmann, J. (1967). Formal Languages and their Relation to Automata. Reading, Mass: Addison-Wesley.

Ginsburg, S. (1963). The Mathematical Theory of Context Free Languages. New York: McGraw-Hill.

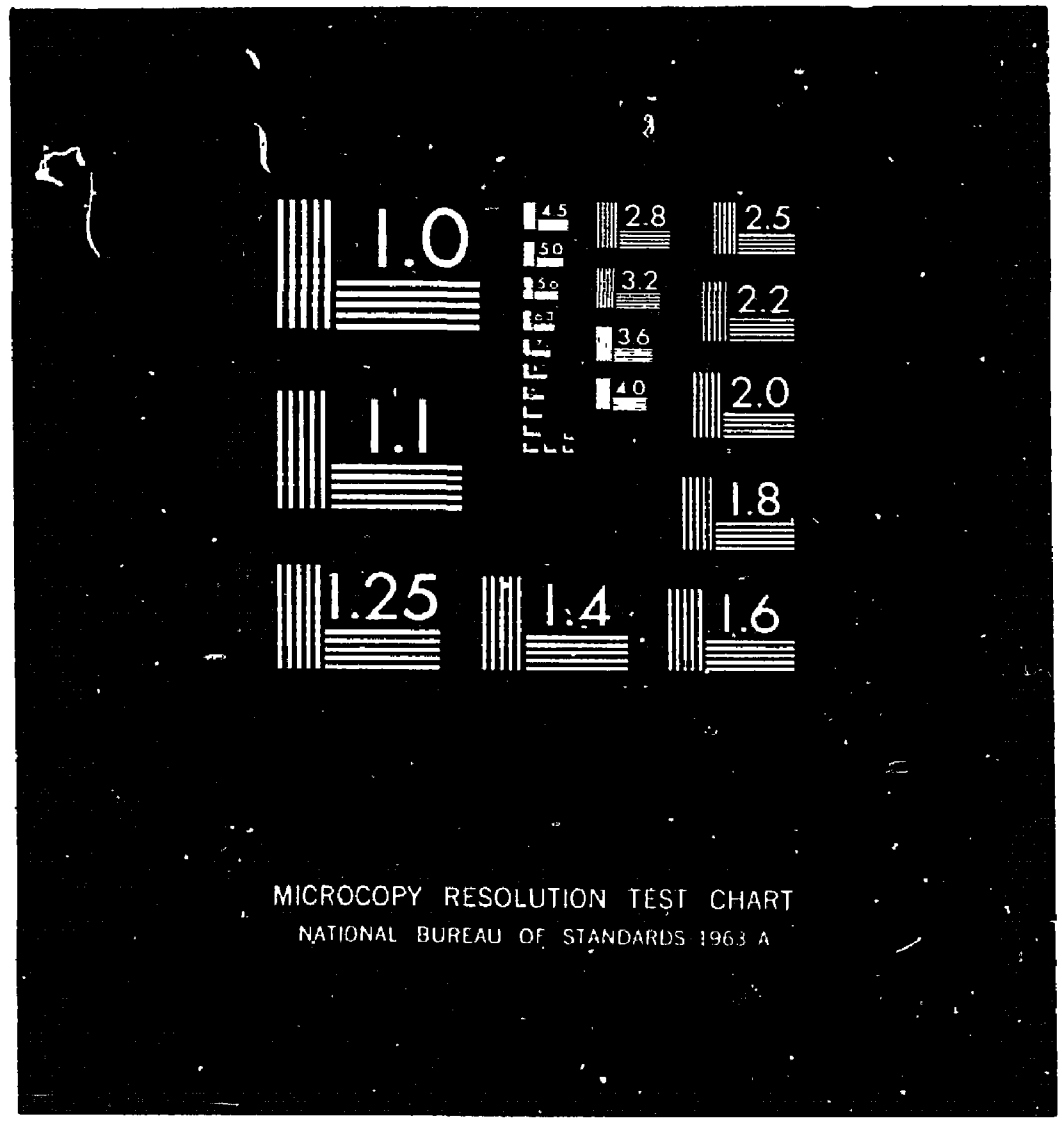
Knuth, D. E. (1968). Semantics of context-free languages. In Mathematical Systems Theory, 2, pp. 127-146.

_____. (1971). Examples of formal semantics. Symposium on Semantics of Algorithmic Languages. Engelor, ed. Lecture Notes in Math #188. New York: Springer-Varlag, pp. 212-235

Langendoen, D. T. (1969). The Study of Syntax. New York: Holt, Rinehart and Winston.

Tarski, A. (1936). Der Wahrheitsbergriff in den formalasierten Sprachen. In Studia Philosophica, I, pp. 261-304. Originally published in 1933.

END


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A