COMPUTATIONAL COMPLEXITY OF FORMAL TRANSLATIONS

J. Hartmanis*

TR 73-192

December 1973

Department of Computer Science
Cornell University
Ithaca, New York 14850

*This research has been supported in part by the National Science Foundation Grant GJ-33171X.

COMPUTATIONAL COMPLEXITY OF

FORMAL TRANSLATIONS*

J. Hartmanis

Abstract:

The purpose of this paper is to define a mathematical model for the study of quantitative problems about translations between universal languages and to investigate such problems. The results derived in this paper deal with the efficiency of the translated algorithms, the optimality of translations and the complexity of the translation process between different languages.

Keywords: Universal languages
Goedel numberings
translations
Complexity of translations
optimality
length of translated programs

*This research has been supported in part by the National Science Foundation Grant GJ-33171X.

I. Introduction

The translation of programs from one universal language into equivalent programs of an other universal language plays a important role in practical computing and it has been an active area of research in computer science. Impressive progress has been made in the understanding of how to design compilers (translators) for higher level programming languages and much of the knowledge has by now been codified in a systematic way so that we can successfully teach compiler writing [1,4]. As a matter of fact, the practical understanding of how to write compilers is a considerable success in the development of computer science. For theoretical computer scientists this development is particularly interesting since quite a few theoretical results have found practical application and some theoretical methods have been helpful in the formalization of concepts and in communication about translations and translators.

On the other hand, there is very little known about the quantitative aspects of translations between arbitrary universal languages and we have hardly any global results about the complexity of translations, the quality of translated programs and optimality of translations.

The purpose of this paper is to set up a mathematical model to facilitate the study of quantitative problems about translations between universal languages and to derive some such quantitative results about translations.

To be able to formulate quantitative results about translations we consider two universal languages, a source and a target language, and we define a computational complexity measure for each language. In terms of these complexity measures we investigate the efficiency of translated programs, the optimality of translators, the complexity of the translation process, measured in the complexity measure of the target language, the length of the translated programs, and the dependence of the complexity of translations on the "syntactic" complexity of the source language. We also discuss the adaquacy of this formulation of the translation problem and suggest some possible research directions.

II. Preliminaries

In this section we describe a mathematical formulation of the translation problem and derive some elementary quantitative results about translations between universal languages.

First, we specify the two universal languages $\mathscr{L}_S$ and $\mathscr{L}_T$, referred to as the source and target language, respectively. Let Σ and Γ be two non-empty finite alphabets and let $\mathscr{L}_S$ and $\mathscr{L}_T$ be infinite recursive subsets of Σ^+ and Γ^+, respectively,

$$\mathscr{L}_S = \{w_1, w_2, \ldots\} \subseteq \Sigma^+$$

and

$$\mathscr{L}_T = \{v_1, v_2, \ldots\} \subseteq \Gamma^+.$$

Furthermore, we assume that the above indicated ordering of the sequences of $\mathscr{L}_S$ and $\mathscr{L}_T$, respectively, is recursively computable.

To specify the <u>semantics</u> and the <u>complexity</u> of the sequences or algorithms of $\mathscr{L}_S$ and $\mathscr{L}_T$ let

$$\{\phi_1, \phi_2, \ldots\} \text{ and } \{f_1, f_2, \ldots\}$$

be two admissible enumerations of all partial recursive functions [9] (i.e. Gödel numberings) and let

$$\{\Phi_1, \Phi_2, \ldots\} \text{ and } \{F_1, F_2, \ldots\}$$

be the step counting functions assigned to $\{\phi_1, \phi_2, \ldots\}$ and $\{f_1, f_2, \ldots\}$, respectively. The step counting functions satisfy the two Computational Complexity Axioms [2,6]:

1. For all i and n $\phi_i(n)$ is defined iff $\Phi_i(n)$ is defined, i.e.

$$(\forall i, n)[\phi_i(n)\downarrow \iff \Phi_i(n)\downarrow].$$

2. For all i,n,m it is recursively decidable whether $\Phi_i(n) = m$.

We assign to every w_i in $\mathscr{L}_S$ ϕ_i which specifies the meaning of w_i and, correspondingly, Φ_i which defines the complexity of the algorithm specified by w_i. Similarly for every v_j in $\mathscr{L}_T$ we assign the meaning f_j and the complexity F_j. Thus the source and target languages are specified by the triples

$$\langle \mathscr{L}_S = \{w_1, w_2, \ldots\} \subseteq \Sigma^+, \{\phi_1, \phi_2, \ldots\}, \{\Phi_1, \Phi_2, \ldots\}\rangle$$

and

$$\langle \mathscr{L}_T = \{v_1, v_2, \ldots\} \subseteq \Gamma^+, \{f_1, f_2, \ldots\}, \{F_1, F_2, \ldots\}\rangle ,$$

respectively. All through this paper, for the sake of brevity, we will refer to these triples as $\mathcal{L}_S$ and $\mathcal{L}_T$, respectively, and know that the semantics and complexity of them are defined as above. We will refer to the sequences in $\mathcal{L}_S$ and $\mathcal{L}_T$ as algorithms or programs.

Furthermore, we assume that # is a symbol not contained in Γ and it will be used by a translation to indicate that the input string is not a program of $\mathcal{L}_S$.

A <u>translation</u> <u>from</u> $\mathcal{L}_S$ <u>into</u> $\mathcal{L}_T$ is a recursive mapping σ,

$$\sigma: \Sigma^+ \to \Gamma^+ \cup \{\#\} ,$$

such that:

1. for all w_i in $\mathcal{L}_S$ $\sigma(w_i) \in \mathcal{L}_T$ and $\sigma(w_i) = v_j$ implies that $\phi_i = f_j$,

2. for all w not in $\mathcal{L}_S$ $\sigma(w) = \#$.

Thus a translation maps programs of $\mathcal{L}_S$ onto equivalent programs of $\mathcal{L}_T$ and strings not in $\mathcal{L}_S$ onto the symbol #. In the conclusion of this paper we discuss the adaquacy of this formulation of translations.

Note that in this definition we depart from a previous convention where $\mathcal{L}_S$ and $\mathcal{L}_T$ themselves were the indices of admissible enumerations [3,5]. This older convention, unfortunately trivializes the syntax of the language and is not sufficient for our purposes. At the same time, since we deal with the complexity of the translation σ and how it depends on the syntax and semantics of $\mathcal{L}_S$ and $\mathcal{L}_T$ only in Section IV, we assume in the first three sections, in order to simplify our expressions, that

$$\{w_1, w_2, \ldots\} = \{v_1, v_2, \ldots\} = \{1,2,3,\ldots\}.$$

This permits us, for example, to replace the first condition in the definition of a translation by the assertion that for all i

$$\phi_i = f_{\sigma(i)} \cdot$$

Next we derive several elementary results. First we show that under any translation σ of $\mathscr{L}_S$ into $\mathscr{L}_T$ the running times of the source language algorithms bound recursively the running times of the translated programs and vica versa. For the sake of completeness we give a detailed proof of the first result though the technique of these proofs are well known in complexity theory [2,6].

<u>Theorem</u>: If σ translates $\mathscr{L}_S$ into $\mathscr{L}_T$ then there exists a recursive function $H(\, , \,)$ such that for all i

$$F_{\sigma(i)}(x) \leq H[x, \Phi_i(x)] \text{ a.e.}$$

i.e. for almost all x.

<u>Proof</u>: Define

$$p(i,x,m) = \underline{\text{if}}\ \Phi_i(x) = m\ \underline{\text{then}}\ F_{\sigma(i)}(x)\ \underline{\text{else}}\ 0.$$

It is seen that p is a recursive function, since for all i,x,m we can, because of second complexity axiom, decide whether $\Phi_i(x)=m$. If $\Phi_i(x) \neq m$ then $p(i,x,m) = 0$, otherwise $\Phi_i(x) = m$ and, using the first complexity axiom, we know that $\phi_i(x)$ converges. But then we know, from the definition of a translation, that $f_{\sigma(i)}(x)$ converges and therefore, using again the first complexity axiom, $F_{\sigma(i)}(x)$ converges and yields $p(i,x,m) = F_{\sigma(i)}(x)$. To obtain the desired recursive function H we set

$$H(x,m) = \max\{p(i,x,m) \mid i \leq m\}.$$

This completes the proof.

Similarly we can show that,

<u>Corollary</u>: For any translation σ of $\mathscr{L}_S$ into $\mathscr{L}_T$ there exists a recursive function D such that for all i

$$D[x, \Phi_i(x)] \leq F_{\sigma(i)}(x) \quad \text{a.e.}$$

and

$$D[x, F_{\sigma(i)}(x)] \leq \Phi_i(x) \quad \text{a.e.}$$

Using the same technique we can show that the running times of the translated programs for any two translations σ_1 and σ_2 of $\mathscr{L}_S$ into $\mathscr{L}_T$ are recursively bounded.

<u>Corollary</u>: If σ_1 and σ_2 translates $\mathscr{L}_S$ into $\mathscr{L}_T$ then there exists a recursive function B such that for all i

$$B[x, F_{\sigma_1(i)}(x)] \leq F_{\sigma_2(i)}(x) \quad \text{a.e.}$$

and

$$B[x, F_{\sigma_2(i)}(x)] \leq F_{\sigma_1(i)}(x) \quad \text{a.e.}$$

In many practical cases the translations σ of $\mathcal{L}_S$ into $\mathcal{L}_T$ is a proper into mapping and thus many algorithms in $\mathcal{L}_T$ cannot be obtained by translating algorithms in $\mathcal{L}_S$ into algorithms in $\mathcal{L}_T$. Thus it may turn out that σ produces rather inefficient algorithms and that the best algorithms are not images under σ. This is quite possible and happens for many σ, at the same time, as our next result shows, we can always effectively obtain a recursive function which bounds the difference in running times between algorithms which are and are not images under σ. As in the previous results these bounds do not have to be "good" bounds but they can be effectively computed if we know the algorithm for σ and the languages $\mathcal{L}_S$ and $\mathcal{L}_T$.

Theorem: For any translation σ of $\mathcal{L}_S$ into $\mathcal{L}_T$ we can effectively construct a recursive function K such that for every f_j there exists an i with

$$f_{\sigma(i)} = f_j$$

and

$$F_{\sigma(i)}(x) \leq K[x, F_j(x)] \quad \text{a.e.}$$

Proof: Since $\phi_1, \phi_2, \ldots$ and $f_1, f_2, \ldots$ are admissible enumerations we know by Rogers' Theorem [9] that they are recursively isomorphic, and we can effectively obtain a recursive function ρ such that for all j there exists an i such that

$$f_j = \phi_{\rho(i)} \ .$$

Let

$$t(j,x,m) = \underline{\text{if}}\ \rho(j) = i\ \underline{\text{and}}\ F_j(x) = m\ \underline{\text{then}}\ F_{\sigma(i)}(x)\ \underline{\text{else}}\ 0$$

We see that t is a recursive function and the desired K is obtained by setting

$$K[x,m] = \max\{t(j,x,m) \mid j \leq x\}.$$

This completes the proof.

III. Quality of Translations

In practical applications one is primarily concerned with two quantitative aspects of translations:

1. The quality or efficiency of the algorithms translated by σ.

2. The size and computational complexity of the translator σ itself.

In this section we will discuss the quality of the translated algorithms and deal with the size and speed of the translation process σ itself in the following section.

To properly understand what can and cannot be achieved by translatiors we have to recall a few facts about the set of all algorithms and their computational complexity. It is clear that the properties of the "space" of all algorithms will play an important role of how it can be translated and the properties which hold for all such "spaces" of algorithms will set universal bounds on the quality of translated algorithms.

First of all we know from elementary recursive function theory [9]:

Fact 1: Every Goedel numbering has for every partial recursive function ϕ an infinite number of indices (algorithms) $i_1, i_2, \ldots$, that is $\phi_{i_j} = \phi$. Furthermore this set of indices for ϕ is not recursively enumerable.

From computational complexity theory [2,6] we know an easily provable result:

Fact 2: In any computational complexity measure $\{(\phi_i, \Phi_i)\}$ for every recursive function ϕ there exist arbitrarily inefficient algorithms. More precisely, for every recursive function G and for every i we can effectively construct a j such that

$$\phi_j = \phi_i \quad \text{and} \quad G(x) \leq \Phi_j(x) \quad \text{a.e.}$$

Next we define an algorithm ϕ_i to be g-optimal if no other algorithm can compute this function g-faster than ϕ_i. For a given computational complexity measure $\{(\phi_i, \Phi_i)\}$ and a recursive g, we say that ϕ_i is g-optimal iff for all j such that

$$\phi_i = \phi_j$$

we have

$$g \circ \Phi_j(x) \geq \Phi_i(x) \quad \text{i.o.}$$

Fact 3: For every computational complexity measure there exists a monotonically increasing recursive function g such that for every total $\Phi_j(x) \geq x$ there exists a g-optimal ϕ_i with

$$\Phi_i(x) \leq g \circ \Phi_j(x) \quad \text{a.e.}$$

Thus near every sufficienctly large total Φ_j there are g-optimal algorithms.

On the other hand, a considerably deeper result from computational complexity theory [2,6] shows that in any measure for every recursive G there exist functions which do not have G-optimal algorithms.

Fact 4: In any computational complexity measure for every recursive function G there exists a recursive function ϕ such that for every i with $\phi_i = \phi$ there exists an index j for ϕ such that

$$G[\Phi_j(x)] < \Phi_i(x) \quad \text{a.e.}$$

From these observations we see that for any $\mathcal{L}_S$ and $\mathcal{L}_T$ the set of algorithms is such that every total function has infinitely many algorithms and that among these algorithms are arbitrarily bad ones, furthermore, for sufficiently large g infinitely many functions have g-optimal algorithms but for every recursive G there are recursive functions which do not have G-optimal algorithms. From the last mentioned fact we conclude that no translation can map all algorithms onto g-optimal algorithms since for some functions they simply do not exist. This follows from the intrinsic nature of the set of all algorithms and holds also for non-recursive translations.

Theorem: There is no translation, including non-recursive translations, which can translate any $\mathscr{L}_S$ into any $\mathscr{L}_T$ so that all algorithms for recursive functions in $\mathscr{L}_S$ are mapped onto g-optimal algorithms in $\mathscr{L}_T$, for any recursive function g.

Proof: Follows from Fact 4.

For every recursive translation σ of $\mathscr{L}_S$ into $\mathscr{L}_T$, as shown by one of our theorems, there exists a recursive bound between $F_{\sigma(i)}$ and Φ_i. This implies that very bad algorithms for a recursive function ϕ must again be mapped onto bad algorithms. Thus even though good algorithms may exist for the function ϕ, a recursive translation cannot improve very much on inefficient algorithms for ϕ. The amount of the possible improvement is bounded by the recursive bound D derived previously.

What is more interesting, as our next result shows, is that every translation σ of $\mathscr{L}_S$ into $\mathscr{L}_T$ can be recursively improved. As a matter of fact, using an index of σ and $\mathscr{L}_S$ and $\mathscr{L}_T$ we can constructively obtain a translation σ' of $\mathscr{L}_S$ into $\mathscr{L}_T$ such that every total function has an infinite number of algorithms whose translations under σ' run much faster then their translations under σ.

Theorem: For every recursive function g and every translation σ of $\mathscr{L}_S$ into $\mathscr{L}_T$ we can effectively construct a translation σ' such that for every recursive function ϕ_i there exist infinitely many algorithms ϕ_{i_j}, $j = 1,2,\ldots,$ for which

$$F_{\sigma(i_j)}(x) \geq g \circ F_{\sigma'(i_j)}(x) \quad \text{a.e.}$$

and such that for all other i, $i \neq i_j$,

$$F_{\sigma(i)}(x) \geq F_{\sigma'(i)}(x).$$

Proof: We recall that there exists a recursive function B such that for all i

$$\Phi_i(x) \leq B[x, F_{\sigma(i)}(x)] \quad \text{a.e.}$$

Without loss of generality we can assume that B is montonically increasing in both variables. We also know that there exists a recursive function ρ such that for all i

$$\rho(i) > i,\ \phi_i = \phi_{\rho(i)} \text{ and } \Phi_{\rho(i)}(x) \geq B[x, g[B(x, \Phi_i(x))]].$$

Define

$$\sigma'(j) = \begin{cases} \sigma(i) & \text{if } \rho(i) = j \\ \\ \sigma(j) & \text{otherwise.} \end{cases}$$

Clearly, $\sigma'(j)$ is a recursive function which translates $\mathscr{L}_S$ into $\mathscr{L}_T$. Furthermore, for every recursive function there exist infinitely many algorithms for which σ' yields g-faster algorithms. More precisely, if

$$\rho(i) = j \text{ then}$$

$$\Phi_j(x) \geq B[x, g[B(x, \Phi_i(x))]] \ .$$

But then

$$F_{\sigma(j)}(x) \geq g[B(x, \Phi_i(x)] \geq g[F_{\sigma(i)}(x)] \quad \text{a.e.}$$

Since B bounds Φ_i to $F_{\sigma(i)}$ and Φ_j to $F_{\sigma(j)}$. Thus

$$F_{\sigma(j)}(x) \geq g[F_{\sigma'(j)}(x)] \quad \text{a.e.}$$

For all other algorithms

$$F_{\sigma(i)}(x) = F_{\sigma'(i)}(x) \ ,$$

and we conclude that σ' is the desired improved translation of σ.

For the sake of completeness we mention that for every $\mathscr{L}_S$ and $\mathscr{L}_T$ there is a recursive bound h and a translation σ such that no translation σ' of $\mathscr{L}_S$ into $\mathscr{L}_T$ can yield h faster algorithms for all algorithms in $\mathscr{L}_S$.

Corollary: For any $\mathscr{L}_S$ and $\mathscr{L}_T$ there exists a recursive function h and a σ such that for no σ' mapping $\mathscr{L}_S$ into $\mathscr{L}_T$ can we have for all i

$$h[F_{\sigma'(i)}(x)] \leq F_{\sigma(i)}(x) \quad \text{a.e.}$$

Proof: This result follows directly from the fact that for any computational complexity measure there is a recursive function g such that infinitely many recursive functions have g-best algorithms.

The previous theorem showed that every translation σ of $\mathscr{L}_S$ into $\mathscr{L}_T$ can be recursively improved. Thus no σ can be viewed as optimal. On the other hand, the improvement of σ was achieved by "eliminating" some terribly bad algorithms. In practical translations we are not concerned primarily with improving on terribly bad algorithms; we are much more concerned with translating reasonably good algorithms (written by programmers) into reasonably good machine language code.

To capture this idea precisely we define speed-up classes, P_h, which consist of those algorithms for which there do not exist h-faster ones.

For $\mathcal{L}_S$ and any recursive function h the h-<u>speed-up class</u> is defined as

$$P_h^S = \{i \mid (\forall_j)[\phi_i = \phi_j \Rightarrow h \circ \Phi_j(x) \geq \Phi_i(x) \text{ i.o.}]\}.$$

The P_h^T classes for $\mathcal{L}_T$ are defined correspondingly. We refer to the algorithms in P_h^S as h-<u>best</u> algorithms.

In terms of these speed-up classes we now define optimality of translations. We believe that the speed-up classes form an interesting class of sets just as the complexity classes do and that their structure should be investigated further.

Let σ be a translation of $\mathcal{L}_S$ into $\mathcal{L}_T$ and let ρ be recursive function. Then σ is said to be ρ-<u>optimal</u> iff for every monotonically increasing recursive function h

$$\sigma(P_h^S) \subseteq P_{\rho o h o \rho}^T$$

Thus σ is ρ-optimal if σ maps h-best algorithms in $\mathcal{L}_T$ onto $\rho o h o \rho$-best algorithms in $\mathcal{L}_T$.

We can also define a translation σ of $\mathcal{L}_S$ into $\mathcal{L}_T$ to be r-h <u>optimal</u> iff

$$\sigma(P_r^S) \subseteq P_h^T .$$

The last concept could be of significance when we are interested in what happens to a particular class of r-good algorithms under translation σ, whereas the first concept can be used to describe what happens globally to h-good algorithms under σ. We will now show that every σ translating $\mathscr{L}_S$ into $\mathscr{L}_T$ is ρ-optimal for some recursive ρ which we can effectively compute from an algorithm for σ. To obtain this result we first state a simple result, which can be easily proven.

<u>Lemma:</u> For every translation σ of $\mathscr{L}_S$ into $\mathscr{L}_T$ we can construct a translation σ' of $\mathscr{L}_S$ onto $\mathscr{L}_T$ such that for all i and x

$$F_{\sigma(i)}(x) \geq F_{\sigma'(i)}(x) .$$

To simplify the statement of our next result we make a technical assumption about $\mathscr{L}_S$ and $\mathscr{L}_T$. Without these assumptions we can define ρ as a function of two variables and get the same result in a somewhat more complicated form.

<u>Theorem:</u> Let g be a monotonically increasing, unbounded recursive function and let $\mathscr{L}_S$ and $\mathscr{L}_T$ be such that for all i and x

$$\Phi_i(x) \geq g(x) \quad \text{and} \quad F_i(x) \geq g(x).$$

Then for every translation σ of $\mathscr{L}_S$ into $\mathscr{L}_T$ we can effectively find a recursive function ρ such that σ is ρ-optimal.

<u>Proof:</u> By our previous lemma we can assume without loss of generality that σ is an onto mapping. Since all Φ_i and F_i satisfy the minimal growth criterion ($\Phi_i(x)$, $F_i(x) \geq g(x)$), we can obtain a monotonically increasing, recursive function H such that for all i

$$H \circ \Phi_i(x) \geq F_{\sigma(i)}(x) \quad \text{a.e.}$$

and

$$H \circ F_{\sigma(i)}(x) \geq \Phi_i(x) \quad \text{a.e.}$$

We will show that σ is an $H = \rho$-optimal translation. That is, for every monotonically increasing h

$$\sigma(P_h^S) \subseteq P_{\rho o h o \rho}^T .$$

Let i be in P_h^S and let j be such that

$$f_j = f_{\sigma(i)} .$$

Since σ is an onto mapping, let ℓ be such that $\sigma(\ell) = j$. Then

$$h \circ \Phi_\ell(x) \geq \Phi_i(x) \text{ i.o. and } H \circ \Phi_i(x) \geq F_{\sigma(i)}(x) \quad \text{a.e.}$$

But then

$$H \circ h \circ \Phi_\ell(x) \geq F_{\sigma(i)}(x) \quad \text{i.o.}$$

Since

$$H \circ F_{\sigma(\ell)}(x) \geq \Phi_\ell(x) \quad \text{a.e.}$$

we conclude that

$$H \circ h \circ H \circ F_{\sigma(\ell)} \geq F_{\sigma(i)}(x) \quad \text{i.o.}$$

and therefore

$$H \circ h \circ H \circ F_j \geq F_{\sigma(i)}(x) \quad \text{i.o.}$$

for all j such that $f_j = f_{\sigma(i)}$. Thus we have that

$$\sigma(P_h^S) \subseteq P^T_{HohoH} \text{ ,}$$

as was to be shown.

Our next result shows for every recursive $g(x) \geq x$ there exist pairs of languages $\mathscr{L}_S$ and $\mathscr{L}_T$ such that no translation σ can map $\mathscr{L}_S$ into $\mathscr{L}_T$ g-optimally. The surprising thing in this result is not that $\mathscr{L}_S$ and $\mathscr{L}_T$ exist, but that they can be so chosen that the complexity classes are equal for all recursive t,

$$C_t^S = \{\phi \mid (\exists i)[\phi_i = \phi \text{ and } \Phi_i(x) \leq t(x) \text{ a.e.}\} =$$

$$C_t^T = \{f \mid (\exists j)[f_j = f \text{ and } F_j(x) \leq t(x) \text{ a.e.}\}$$

as well as those classes of functions which have h-best algorithms are equal for all recursive h,

$$Q_h^S = \{\phi \mid (\exists i)[\phi_i = \phi \text{ and } i \text{ in } P_h^S]\} =$$

$$Q_h^T = \{f \mid (\exists j)[f_j = f \text{ and } j \text{ in } P_h^T]\}.$$

That is, $\mathscr{L}_S$ and $\mathscr{L}_T$ are from a computational complexity point of view very similar. On the other hand, no recursive translation can map good algorithms of $\mathscr{L}_S$ onto good algorithms in $\mathscr{L}_T$, since they cannot be found recursively.

In the following proof for a given $\mathscr{L}_S$ and g we will construct the desired $\mathscr{L}_T$. The construction is quite arbitrary but our following result shows that we can get $\mathscr{L}_T$ as an image of $\mathcal{L}_S$ under a recursive translation π so that $\mathcal{L}_S$ cannot be translated by any (recursive) σ g-optimally into $\pi(\mathcal{L}_S)$. In both cases, there exist non-recursive translations mapping $\mathcal{L}_S$ into $\mathcal{L}_T$ e-optimally, e = identity.

<u>Theorem:</u> Let $g(n) \geq n$ be a monotonically increasing recursive function. Then for every $\mathcal{L}_S$ there exists an $\mathcal{L}_T$ such that for all recursive h

$$\{\phi_i | i \in P_h^S\} = \{f_j | j \in P_h^T\}$$

but no translator of $\mathcal{L}_S$ into $\mathcal{L}_T$ can be g-optimal.

<u>Proof</u>: For the sake of clarity we will use a double index to enumerate the algorithms of $\mathcal{L}_T$ as follows:

$$f_{i,j} = \phi_i \quad \text{for all } j .$$

To define the complexity $F_{i,j}$ we will use an auxilliary set A. Let A be a hypersimple set and let

$$\bar{A} = \{x_1, x_2, x_3, \ldots\}$$

with $x_1 < x_2 < x_3 < \ldots$. Since A is hypersimple we know that A is recursively enumerable and that for every recursive function t the relation

$$t(i) > x_i$$

can hold for only finitely many i [9]. We fix a recursive enumeration of A and consider a recursive function h such that P_h^S contains an infinite set of algorithms, $\{i_1, i_2, i_3, \ldots\}$

for infinitely many different recursive functions (we know such an h always exists). Furthermore let G be a recursive function such that for all k $g^k(x) < G(x)$ a.e. Define,

$$F_{i,k}(x) = \underline{\text{if in the enumeration of}}\ A\ \underline{\text{after}}\ x\ \underline{\text{steps at least}}\ i\ \underline{\text{integers less than}}\ k\ \underline{\text{are not yet found to be in}}\ A\ \underline{\text{then}}\ \Phi_i(x)\ \underline{\text{else}}\ G \circ h \circ G \circ \Phi_i(x).$$

Clearly, $f_{i,k}$ with $F_{i,k}$ define a computational complexity measure. Furthermore, for any i

$$F_{i,k}(x) = G \circ h \circ G \circ \Phi_i(x) \quad \text{a.e.}$$

if $k \leq x_i$, where x_i is the i-th element of $\bar{A}$. If $k > x_i$, then

$$F_{i,k}(x) = \Phi_i(x).$$

[Thus, informally speaking, we have moved the good algorithms for every ϕ_i noncomputably far down in the second index of $f_{i,k}$ and therefore no recursive σ can map all ϕ_i's on to the good algorithms in $\mathscr{L}_T$.]

We will now show that if a recursive σ translating $\mathscr{L}_S$ into $\mathscr{L}_T$ is g-optimal, then we can construct a recursive function t such that

$$t(i) > x_i$$

for infinitely many i, which contradicts the fact that A is hypersimple. Thus no translation of $\mathscr{L}_S$ into $\mathscr{L}_T$ can be g-optimal.

To see this we observe that either

$$\sigma(i) = (s,k) \quad \text{with} \quad s \geq i$$

or we can compute

$$\sigma(s) = (s',k') \quad , \; s' \leq s$$
$$\sigma(s') = (s'',k'') \quad , \; s'' \leq s'$$
$$\vdots$$

until we reach $s^{(n)}$ such that

$$\sigma(s^{(n)}) = (s^{(n+1)},k^{(n+1)}) \quad , \; s^{(n+1)} \geq s^{(n)}.$$

If $i \in P_h^S$ then $s^{(n)}$ must be contained in

$$P^T_{g^n \circ h \circ g^n} \subseteq P^T_{G \circ h \circ G}$$

or σ is not g-optimal. Thus we have infinitely many indices of total functions $i_1, i_2, \ldots$ in $P^S_{g^{i_j} \circ h \circ g^{i_j}}$ such that

$$\sigma(i_j) = (s,k) \quad \text{with} \quad i_j \leq s.$$

But then we must have $k \geq x_{i_j}$, since otherwise

$$F_{\sigma(i_j)}(x) = G \circ h \circ G \quad \Phi_{i_j}(x) \quad \text{a.e.}$$

and for $x_{i_j} = k$

$$f_{i_j,k} = f_{\sigma(i_j)} \quad \text{with} \quad F_{i_j,k} \leq g \circ h \circ g \quad \Phi_{i_j}$$

from which we conclude that σ is not g-optimal.

But then we can construct from σ a recursive function t which for infinitely many i satisfies

$$t(i) \geq x_i \ .$$

This is a contradiction since x_i is the i-th largest element of $\bar{A}$, where A is a hypersimple set. Thus no σ can be g-optimal, as was to be shown.

In the previous result the construction of $\mathscr{L}_T$ may appear artificial. On the other hand, it turns out that $\mathscr{L}_T$ can be chosen to be a translation of $\mathscr{L}_S$ into itself for any $\mathscr{L}_S$.

<u>Corollary</u>: For every $\mathscr{L}_S$ and every recursive monotonically increasing $g(x) \geq x$ we can construct a recursive translation π of $\mathscr{L}_S$ into $\mathscr{L}_S$ such that no recursive translation σ of $\mathscr{L}_S$ into $\pi(\mathscr{L}_S)$ can be g-optimal. Furthermore, for all recursive h the complexity classes and the classes of functions with h-best programs are identical for $\mathscr{L}_S$ and $\pi(\mathscr{L}_S)$.

<u>Proof</u>: The proof is obtained by replacing $\mathscr{L}_T$ of the previous proof by $\pi(\mathscr{L}_S)$ for a properly constructed translation π. The proof is technically messy and long but the main ideas are already contained in the previous proof.

It should be observed that $\mathscr{L}_S$ can be mapped by non-recursive translations g-optimally into $\mathscr{L}_T$ or $\pi(\mathscr{L}_S)$, of the two previous results, respectively. Emphasizing the fact that π, for example, does not destroy the good algorithms, it just hides them so that recursive translations cannot find them.

IV. Complexity of Translations

In this section we investigate the complexity of the translation σ of $\mathcal{L}_S$ into $\mathcal{L}_T$ by considering the step counting function of σ in $\mathcal{L}_T$.

To be able to separate the complexity of the translation due to the complexity the syntax of $\mathcal{L}_S$ and $\mathcal{L}_T$ from the complexity of the actual translation we now permit $\mathcal{L}_S$ and $\mathcal{L}_T$ to be arbitrary recursive subsets of Σ^+ and Γ^+ , respectively.

First, we will show that in this case there exist pairs of languages $\mathcal{L}_S$ and $\mathcal{L}_T$ for which the complexity of any translation mapping $\mathcal{L}_S$ into $\mathcal{L}_T$ must exceed an arbitrary recursive. bound. The reason for this is that σ must be able to decide whether $w \in \mathcal{L}_S$ and thus the complexity of the recognition problem for $\mathcal{L}_S$ can be exploited to force the complexity of σ to be arbitrarily high. On the other hand, if $\mathcal{L}_T$ is fixed and the complexity of the recognition problem "w in $\mathcal{L}_S$" is bounded, then the complexity of the translations σ of $\mathcal{L}_S$ into $\mathcal{L}_T$ can be bounded, independently of the semantics of $\mathcal{L}_S$. We will refer to the complexity of the recognition problem "$w \in \mathcal{L}_S$" as <u>syntactic</u> <u>complexity</u>.

<u>Theorem</u>: For every recursive function $g(x) \geq x$ and every $\mathcal{L}_T$, there exists an $\mathcal{L}_S$ such that for every translation σ of $\mathcal{L}_S$ into $\mathcal{L}_T$ $f_\sigma = \sigma$ implies that

$$F_\sigma(x) \geq g(x) \quad \text{a.e.}$$

<u>Proof</u>: Without loss of generality we can assume that $\mathcal{L}_T$ is given by one-tape Turing machines and the amount of tape used is the complexity measure. Then any σ translating $\mathcal{L}_S$ into $\mathcal{L}_T$

can be converted into a recognizing device for sequences in $\mathscr{L}_S$ without increasing the complexity F_σ. But then, if we choose $\mathscr{L}_S$ to be harder to recognize than $g(x)$, that is $\mathscr{L}_S \notin C_g$, then we know that σ must have complexity larger than g, i.e. $F_\sigma(x) \geq g(x)$ a.e., as was to be shown.

Theorem: For every recursive function $g(x)$ and $\mathscr{L}_S$ we can choose an $\mathscr{L}_T$ such that any translation σ of $\mathscr{L}_S$ into $\mathscr{L}_T$ $f_\sigma = \sigma$ implies that

$$F_\sigma(x) \geq g(x) \quad \text{a.e.}$$

Proof: By choosing the set $\mathscr{L}_T$ to be such that no infinite subset of $\mathscr{L}_T$ can be enumerated easily, we can force the complexity of σ to be arbitrarily large.

Next we show that for a fixed $\mathscr{L}_T$ we can recursively bound the complexity of translations from $\mathscr{L}_S$ into $\mathscr{L}_T$ in terms of the syntactic complexity of $\mathscr{L}_S$.

To do this we utilize an auxilliary language $\mathscr{L}_A$ such that for every $\mathscr{L}_S$

there exists a prefix z such that for $w \in \mathscr{L}_S$

$$P_z(w) = zw$$

is a translation of $\mathscr{L}_S$ into $\mathscr{L}_A$. It can be shown that such languages exist and they have been studied before [7,10].

Informally, the prefix expresses: "the data string following me must be interpreted as a program of $\mathscr{L}_S$ whose definition is given here."

Theorem: Consider a fixed language $\mathcal{L}_{T_0}$. Then there exists a recursive function R[,] such that for any $\mathcal{L}_S$ in C_t, t recursive, there exists a translation σ of $\mathcal{L}_S$ into $\mathcal{L}_{T_0}$ with $\sigma = f_\sigma$ and $F_C(w) \leq R[w, t(w)]$ a.e.

Proof: Let ρ be a translation of $\mathcal{L}_A$ into $\mathcal{L}_{T_0}$. To obtain R[,] which bounds the complexity of translations, from languages with syntactic complexity t into $\mathcal{L}_{T_0}$, consider the algorithms

$$\sigma_z(w) = \underline{\text{If}}\ w \in \mathcal{L}_S\ \underline{\text{then}}\ \rho(zw)\ \text{else}\ \#.$$

Clearly, for an appropriately chosen z σ_z translates $\mathcal{L}_S$ into $\mathcal{L}_{T_0}$, since for $w \in \mathcal{L}_S$ zw is an equivalent program in $\mathcal{L}_A$ and therefore $\rho(zw)$ is an equivalent program in $\mathcal{L}_{T_0}$. But then there exists a recursive function r[, ,] which bounds the complexity of $\sigma_z(w)$ in terms of z, w, and t(w), that is

$$F_{\sigma_z}(w) \leq r[z,w,t(w)] \quad \text{a.e.}$$

Letting

$$R[w,t(w)] = \max_{z \leq w}[z,w,t(w)]$$

we see that R satisfies the theorem, as was to be shown.

Thus we see that the complexity of translations into $\mathcal{L}_{T_0}$ can be bounded in terms of the syntactic complexity of $\mathcal{L}_S$. If we choose a reasonable $\mathcal{L}_{T_0}$ (say ALGOL) and a specific natural complexity measure, then R can easily be computed. For example, if we use the step counting on multi-tape Turing machines as one measure then we can choose

$$R[w,t(w)] = \max[w,t(w)].$$

This shows very clearly how the complexity of the translations is bounded by the syntactic complexity of $\mathscr{L}_S$.

Similarly, we can show that for every fixed $\mathscr{L}_{T_0}$ we can recursively bound the length of the translated programs in terms of the length of the source program for all $\mathscr{L}_S$[7].

<u>Theorem</u>: For every fixed $\mathscr{L}_{T_0}$ we can effectively obtain a recursive function p such that for all $\mathscr{L}_S$ there exists a translation σ_p of $\mathscr{L}_S$ into $\mathscr{L}_{T_0}$ and

$$p(|w|) \geq |\sigma_p(w)| \quad \text{a.e.}$$

<u>Proof</u>: Let ρ be a translation of $\mathscr{L}_A$ into $\mathscr{L}_{T_0}$. Then for every $\mathscr{L}_S$ there exista a z such that

$$P_z(w) = zw$$

is a translation of $\mathscr{L}_S$ into $\mathscr{L}_A$. Clearly $\rho(zw)$ is a translation of $\mathscr{L}_S$ into $\mathscr{L}_{T_0}$. Then

$$\rho(|w|) = \max_{z \leq w}\{|\rho(zw)|\}$$

is a recursive function satisfying the theorem.

V. <u>Conclusion</u>

The mathematical formulation of the translation problem in this paper is very general and it does not seem to exclude any intuitively reasonable concepts of translation. Thus many other quantitative problems about translations can be investigated in this framework, and we believe should be. On the other hand, it is also clear that in many regards this formulation is too general. For example, the syntax, semantics and complexity of $\mathscr{L}_S$ and $\mathscr{L}_T$ are defined completely independently. Also all translations are admitted. This seems to be too general a formulation and that an important task is to couple, in a very general way, the syntax semantics and complexity measures of $\mathscr{L}_S$ and $\mathscr{L}_T$ respectively and then restrict somewhat the translations of $\mathscr{L}_S$ into $\mathscr{L}_T$. How this should be done is not at all clear and seems to be related to the difficult problem of defining "natural" computational complexity measures [8]. At the same time, we believe that these are important problems and that the study of formal translations may indicate how an abstract theory linking syntax, semantics and complexity of languages can and should be developed.

VI References

[1] Aho, A.V. and J.D. Ullman, The Theory of Parsing, Translation, and Compiling, Prentice-Hall, Englewood Cliffs, N.J., 1972.

[2] Blum, M., "A Machine-independent Theory of the Complexity of Recursive Functions", JACM Vol. 14, No. 2, (1967), pp. 332-336.

[3] Constable, R.L. and J. Hartmanis, "Complexity of Formal Translations and Speed Up Results", Conf. Record of 3rd Annual ACM Symposium on The Theory of Computing, 1971, pp. 244-250.

[4] Gries, D., Compiler Construction for Digital Computers, Wiley & Sons, New York, 1971.

[5] Hamlet, R.G., "Universal Abstract Programming Languages", Computer Science Center, University of Maryland, Technical Report.

[6] Hartmanis, J. and J.E. Hopcroft, "An Overview of the Theory of Computational Complexity", JACM Vol. 18, 3 (July 1971), pp. 444-475.

[7] Hartmanis, J. and T.P. Baker, "On Simple Göedel Numberings and Translations", Dept. of Computer Science, Cornell University, TR 73-179, July, 1973.

[8] Hartmanis, J., "On the Problem of Finding Natural Computational Complexity Measures", Dept. of Computer Science, Cornell University, TR73-175, June, 1973.

[9] Rogers, H. Jr., Theory of Recursive Functions and Effective Computability, McGraw-Hill Book Co., New York, 1967.

[10] Schnorr, C.P., "Optimal Enumerations and Optimal Göedel Numberings", To appear in Math. Syst. Theory.