

COMMENTING PROOFS

James R. Geiser
Massachusetts Institute of Technology
Artificial Intelligence Laboratory
Cambridge, Massachusetts

Abstract

The goal of this investigation is the development of a semantics for 1st order theories based on certain new syntactic structures in formal proofs which derive from their pragmatic and semantic aspects. The present report is mainly concerned with these new syntactic structures, the motivation behind them, their technical definition and their basic properties. Their role in a new semantics for Intuitionistic Peano Arithmetic is indicated in the last section.

1. Introduction

This paper constitutes a summary of a seminar entitled "Commenting Proofs" given at the Artificial Intelligence Laboratory at M. I. T. during the spring of 1974. The seminar was concerned with new syntactical structures in formal proofs which derive from their pragmatic and semantic aspects. These structures can be used to provide a semantical interpretation for proofs quite distinct from that of Model Theory and rather more akin to the construction semantics for Intuitionistic Peano Arithmetic developed by K. Godel and the A. I. procedural interpretation of the logical connectives and quantifiers in a la C. Hewitt's Planner Language. The work is actually a synthesis of elements from Yessenin-Volpin's foundational studies (1) and G. Sussman's work on building a "causal" history of a computation from the comments of the corresponding documented program (2).

For the most part we shall restrict ourselves to the context of Peano Arithmetic and the primitive recursive arithmetic of addition, multiplication and exponentiation in particular. At the end of this report a few remarks will be made on how these ideas are extended to a richer deductive environment.

In our work we distinguish between formal proofs (a sequence of sentences satisfying the usual syntactic criteria) and "real" proofs (a formal proof constructed by someone in accordance with a motivational system of goals). In the latter case each line may be commented by intrinsic (formal-syntactic) remarks and extrinsic (goal related) remarks. These remarks not only explain the purpose of a particular line, but at the same time establish connections with other lines (previous lines and lines yet to be constructed). In fact these remarks point to connections between the very signs that make up the lines of

the proof. These connections, which, using Yessenin-Volpin's terminology, are called identificational connections or ids, are the links of a very detailed syntactic structure which resides implicitly in real proofs, a structure of causal chains connecting the occurrences of symbols. This is the new syntactic structure mentioned in the first paragraph.

With this sort of information it becomes possible to answer a question like "what part of the term $11 \cdot 111$ is responsible for the third stroke in the right hand side of the equation $11 \cdot 111 = 111111$?". Such a part will be called an ingredient of a term, consisting of a certain list structure whose atoms are the occurrences of symbols in the term. If we think of a term t being evaluated to its numeral value $|t| = 11 \dots 1$ then the ingredients represent the computational history of each stroke in $|t|$.

Much of the present report is concerned with defining the ingredients of a term and characterizing their dependence on the computational paths used to evaluate the term. The semantics which arises from this comes from considering a proof as encoding a procedure for manipulating list structures so that if, for example, the proof is of $t = s$ then the encoded procedure induces an isomorphism from the ingredients of t onto the ingredients of s . While ingredients and the resulting semantics can be implemented it seems best at this point to treat these ideas theoretically and to be aware of their possible relevance to such concepts as consistency, relevant entailment, computational efficiency and methods of proof. We won't develop these connections in this report other than to briefly point out a connection between computational efficiency and the "normal" form of an ingredient in section 7.

2. Recursive Arithmetic.

Recursive Arithmetic consists syntactically of terms and equations, and deductively of computations (constructed by means of the recursion axioms and the substitution rule).

Terms are expressions formed from $1, +, \cdot, \exp, (,)$ by means of the following rules.

- 1) 1 is a term.
- 2) If t is a term then so is tl .
- 3) If t and s are terms then so are $(t + s)$, $t \cdot s$, and $\exp(t, s)$.

We shall use the symbols $t, r, s, \dots, t_1, r_1, s_1, \dots$ to denote terms. We shall make use of the usual conventions for ignoring parentheses. Also note that our use of the word "term" doesn't allow for the occurrences of free variables, i.e. they are always to be closed.*

Terms of the form $1, 11, \dots, 1^{(n)}$ (n concatenated strokes) are called numerals.

An equation is an expression of the form $t = s$.

The Recursion Axiom Schemata:

- Addition A1 $(t + 1) = t1$.
 A2 $(t + s1) = (t + s)1$.
 Multiplication M1 $t \cdot 1 = t$.
 M2 $t \cdot (s1) = t \cdot s + t$.
 Exponentiation E1 $\exp(t, 1) = t$.
 E2 $\exp(t, s1) = \exp(t, s) \cdot t$.

The Substitution Rule Schema:

Line LA $t = s(r)$
 Line LB $r = r'$
 Line LC $t = s(r')$.

Here $s(r)$ denotes a term in which the term r has one or more indicated occurrences, and $s(r')$ denotes the term resulting from the replacement in $s(r)$ of r by r' at the indicated occurrences of r in $s(r)$. Note that this is more general than uniform substitution. We say that LC follows from LA and LB by substitution.

A proof or computation is formally defined as a sequence of equations each of which is an instance of a recursion axiom or else follows by substitution from two previous equations. We consider two simple yet illuminating examples.

Example 2.1.

- L1 $1 + 11 = (1 + 1)1$
 L2 $1 + 1 = 11$
 L3 $1 + 11 = 111$.

Example 2.2 .

- L1 $1 \cdot 11 = 1 \cdot 1 + 1$
 L2 $1 \cdot 1 = 1$
 L3 $1 \cdot 11 = 1 + 1$
 L4 $1 + 1 = 11$
 L5 $1 \cdot 11 = 11$.

3. Commenting Proofs .

We begin commenting these computations by means of intrinsic and extrinsic remarks--this terminology is after Sussman. Then we use these remarks to generate ids between the occurrences of symbols in the computation. By

* The notational methods that we shall develop for $+$, $\cdot$, and $\exp$, will serve to handle all other primitive recursive functions.

tracing out chains of ids we can establish an accountability for every sign in the computation. This will at the same time make explicit the semantics of the computation, i.e. which occurrences of symbols are synonymous and what are the computational roles of each sign.

The intrinsic or formal comments make note of :

- 1) if the line is an axiom; in which case a pointer is generated to the axiom schema in question;
- 2) if the line follows by substitution; in which case pointers are generated to the lines from which it follows and to the occurrences of the term to be replaced.

The extrinsic or goal-related remarks consist of:

- 1) the top level goal statement (i.e. find the value of term t);
- 2) the assertion that line L ($s = s'$) is generated in order to simplify by substitution a term $t(s)$ in a previous line L' ;
- 3) the assertion that line L'' is the result of a substitution rule from lines L, L' whose purpose it was to achieve simplification by means of this substitution;
- 4) the assertion that the line matches the top level goal.

Example 2.1 Commented.

The top level goal is to evaluate $1 + 11$.

- L1 $1 + 11 = (1 + 1)1$ (Axiom A2) (Purpose is to simplify $1 + 11$ of top level goal.)
 L2 $1 + 1 = 11$ (Axiom A1) (Purpose is to simplify $1 + 1$ of r.h.s. of L1 using substitution.)
 L3 $1 + 11 = 111$ (Sub. L1, L2) (Purpose is to fulfill the goal of L2. L3 matches top level goal.)

Example 2.2 is commented in a similar manner.

4. Identificational Connections.

We now add a third type of comment to the analyzed computation, namely we make note of the identificational connections (ids). First of all each axiom is to be accompanied by certain ids as follows.

- A1 $t + 1 = t1$
 A2 $t + s1 = (t + s)1$
 M1 $t \cdot 1 = t$
 M2 $t \cdot (s1) = t \cdot s + t$
 E1 $\exp(t, 1) = t$
 E2 $\exp(t, s1) = \exp(t, s) \cdot t$

For example, in A2 we would say that s and

t in the r.h.s. are rewritten from the s and t in the l.h.s. and this justifies their synonymy. On the other hand we also make an identificational connection between the two occurrences of 1 and this constitutes our semantical interpretation of + in terms of the successor function x1.

Now consider M2. We say that both t's in the r.h.s. are rewritten from the l.h.s. t. Similarly the s in the r.h.s. is rewritten from the s in the l.h.s. . However, we associate the second t in the r.h.s. with the stroke 1 of the l.h.s. ---this is part of our semantical interpretation of ·. Thus we are looking at m·n as saying add m to itself n times; in this computation n acts as a counter for the n different rewritings of m. The strokes of m are the counter or control elements in this case.

Let A be an axiom and id(A) the set of ids associated with A. More explicitly: if a stroke p on the r.h.s is simply rewritten from a stroke q on the l.h.s. (without a control element) then put id(p,q) in id(A); if on the other hand p in the r.h.s. is rewritten from q in the l.h.s. under the control element q' and using an axiom for the operation f (either · or exp) then put id(p, (q, q', f)) in id(A). *

Ids come from the substitution rule in accordance with the following schema.

$$\begin{array}{l} \text{La } t = r(s) \\ \text{Lb } s = s' \\ \text{Lc } t = r(s') \end{array}$$

These ids actually come from the extrinsic comments associated with these lines: Lb's purpose is to simplify the indicated s in the r.h.s. of La. Hence the s in the l.h.s. of Lb is rewritten from the s in the r.h.s. of La. Lc achieves the goal of Lb; hence the context r in the r.h.s. of Lc is rewritten from the context r in the r.h.s. of La and the s' in the r.h.s. of Lc is rewritten from the s' of the r.h.s. of Lb. Also, the l.h.s. of Lc is rewritten from the l.h.s. of La.

Fully commented by ids Examples 2.1 and 2.2 look like the following.

Example 2.1 Commented by ids.

$$\begin{array}{l} \text{L1 } 1 + 11 = (1 + 1)1 \\ \text{L2 } 1 + 1 = 11 \\ \text{L3 } 1 + 11 = 111 \end{array}$$

* A set of ids like id(A) is usually taken symetrically, i.e., if id(u,v) is in id(A) then so is id(v,u).

Example 2.2 Commented by ids.

$$\begin{array}{l} \text{L1 } 1 \cdot 11 = 1 \cdot 1 + 1 \\ \text{L2 } 1 \cdot 1 = 1 \\ \text{L3 } 1 \cdot 11 = 1 + 1 \\ \text{L4 } 1 + 1 = 11 \\ \text{L5 } 1 \cdot 11 = 11 \end{array}$$

We can now trace out paths of ids, thereby diagramming the computational relations between the different occurrences of strokes in the proof. For example, in Example 2.1, if we denote the occurrences of strokes in L3 by p, p', p'', q, q', q'' (left to right) then we can see that p is connected to q, p' is connected to q' and p'' is connected to q''. Further more these are the only connections between the strokes of L3. This yields a very nice correspondence between the l.h.s. and the r.h.s. strokes of L3.

The case of Example 2.2 is more complicated in that some of the occurrences of strokes act as counter elements and when in this role they do not get rewritten. Label the strokes in L5 p, p', p'' q q' (left to right). Trace out a path from q or q' of L5 to a pattern of strokes p, p', and p'' of L5, using the ids for the axioms and the substitution rule. We see that q traces out to the pattern (p, p', ·) and q' traces out to the pattern (p, p'', ·). Thus we are lead to say that q is rewritten from p under the control element p' while q' is rewritten from p under the control element p''. (p, p', ·) and (p, p'', ·) are the patterns in 1 · 11 of L5 "responsible" for q and q' respectively in this computation. These patterns, which will be precisely defined in section 6, we call the ingredients of 1 · 11. Generally speaking, when a term t is evaluated (i.e. proved equal to some numeral n) we may identify the ingredients of t responsible for each stroke in n. In section 7 we determine how the ingredients depend on the computation path used to evaluate the term t.

5. Computation Paths.

On the previous pages we have presented two examples in some detail in order to give an intuitive picture of what is happening with proofs, comments and ids. We shall now turn to a detailed study of the possible computation paths from a term.

We shall restrict our attention to computations which have the following (standard) form:

$$\begin{array}{l} \text{L1 } t_1 = t_2 \\ \cdot \\ \cdot \end{array}$$

.
. $L2i-1 \quad t_1 = t_{i+1}$
 $L2i \quad s_i = s_i'$
 $L2i+1 \quad t_1 = t_{i+2}$
.
.
 $L2n-1 \quad t_1 = t_n$.

where $L1$ is an axiom and for $i = 1, \dots, n-2$, $L2i+1$ follows from $L2i-1$ and $L2i$ by substitution. We shall assume that these computations have been commented and the appropriate ids have been made.

The sequence of terms $t (=t_1), t_2, \dots, t_n$ is called a computation path from t to t_n . Consider a step of the computation (called a simple reduction).

$La \quad t = r(s)$
 $Lb \quad s = s'$
 $Lc \quad t = r(s')$

We write $r(s) \rightarrow r(s')$ to indicate that $r(s')$ has been obtained from $r(s)$ by a simple reduction. If $u = v$ is an axiom we shall also write $u \rightarrow v$ and call it a simple reduction.

Define $ID(r(s) \rightarrow r(s'))$ to be the set of derived ids between the signs of $r(s)$ and $r(s')$. Specifically:

- 1) If p is an occurrence of a stroke in $r(s')$ which is rewritten from q in $r(s)$ via the id a then $id(p, q)$ is in $ID(r(s) \rightarrow r(s'))$.
- 2) If p is an occurrence of a stroke in $r(s')$ which is rewritten from q in s' of Lb via c and q is rewritten from q' in s of the l.h.s. of Lb (i.e. the id $id(q', q)$ is in $id(s = s')$, see section 4), and q' is rewritten from q'' in s in the r.h.s. of La via b , then $id(p, q'')$ is in $ID(r(s) \rightarrow r(s'))$.
- 3) If p is an occurrence of a stroke in $r(s')$ and is rewritten from q in s' of Lb via c and q is rewritten from q' in s of the l.h.s. of Lb under the control of q'' in the l.h.s. of Lb for the function f (i.e. $id((q', q'', f), q)$ is in $id(s = s')$), and q' and q'' are rewritten from q^* and q^{**} respectively in s in the r.h.s. of La via b then $id(p, (q^*, q^{**}, f))$ is in $ID(r(s) \rightarrow r(s'))$.
- 4) If $u = v$ is an axiom define $ID(u \rightarrow v)$ to be equal to $id(u = v)$.

Every occurrence of a stroke p in $r(s')$ is associated through $ID(r(s) \rightarrow r(s'))$ with a unique stroke q in $r(s)$ of La or a unique pattern (q^*, q^{**}, f) of strokes in $r(s)$.

If $t_1, \dots, t_n$ is a computation path P and t_n is a numeral then P is called an evaluation path and t_n is called the value of t_1 w.r.t. P .

Define $T(t)$ to be the set of all terms occurring in the computation paths from t ; the rela-

tion $s \rightarrow r$ determines a partial ordering of $T(t)$ which we will now investigate.

The notation $s \dashrightarrow r$ denotes a computation path from s to r . We say that $t_1 \dashrightarrow t_n$ is a subterm path if none of the reductions $t_i \rightarrow t_{i-1}$ involve the main function (outer most function symbol in polish notation).

Definition 5.1 A loop consists of two different computation paths going from a term t to a term t' . A diamond is a loop which has either form I or II below. We illustrate this using $\cdot$ as the main function symbol.

Form I (where the indicated paths are subterm computation paths)

Form II (where both broken paths are subterm paths; we may assume that each involves that same associated subterm paths $s \dashrightarrow s'$ and $r \dashrightarrow r'$).

Definition 5.2 Two computation paths are simple variants if they differ by a diamond, i.e. they look like



Two paths are homologous if there is a sequence of computation paths $P_1, P_2, \dots, P_n$ such that P_1 and P_n are the two paths in question and for $i = 1, \dots, n-1$, P_i and P_{i+1} are simple variants. Note that homologous evaluation paths assign the same value to a term.

Lemma 5.3a. A split of the form

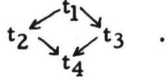
$s'.r' \dashrightarrow s''.r''$ or $s'.r' \dashrightarrow s'.r.l \dashrightarrow s.r$

can be resolved into a diamond. (The dotted paths are subterm paths. Multiplication is serving as a paradigm case.)

Lemma 5.3b. Any split can be resolved into a loop.

Lemma 5.3 is an analogue to the Church-Rosser theorem for the Lambda Calculus. The proof of this and other results is by an induction argument on the rank $rk(t)$ of a term, where rk is an integer valued function (primitive recursive) defined so that a) if s is a subterm of t then $rk(s) < rk(t)$ and b) if $s \rightarrow t$ then $rk(t) < rk(s)$. The existence of such a function shows us that any computation path from t has less than or equal to $rk(t)$ steps. If we were using all primitive recursive functions then such a rank function could be

general recursive but not primitive recursive, e. g. like Ackerman's function. The organization of the proof is to dovetail 5.3a and 5.3b, first proving 5.3a for $\text{rk}(t) \leq n$ and then 5.3b for $\text{rk}(t) \leq n$. The proof is somewhat more complicated than the usual one because we do not require substitution to be uniform, a condition which would considerably simplify the structure of $T(t)$, namely any split of a major connective would always immediately resolve:



6. Ingredients.

Definition 6.1. Let $q_1, \dots, q_n$ denote the occurrences of strokes in the term t . $\text{Ing}(t)$ denotes the set of list expressions obtained from $q_1, \dots, q_n$, $\cdot$, and exp , according to the following rules.

- 1) $q_1, \dots, q_n$ are in $\text{Ing}(t)$.
- 2) If $\underline{i}$ and $\underline{j}$ are in $\text{Ing}(t)$ and f is $\cdot$ or exp then $(\underline{i}, \underline{j}, f)$ is in $\text{Ing}(t)$.

The members of $\text{Ing}(t)$ are called the abstract ingredients of t , and $q_1, \dots, q_n$ are called the simple ingredients of t . $\text{Ing}_0(t)$ denotes the set of simple ingredients of t . In the expression $(\underline{i}, \underline{j}, f)$, $\underline{j}$ is called the control element and $\underline{i}$ is called the raw material. To understand the motivation behind this definition let us once again consider the recursive definition of $u \cdot v : u \cdot 1 = u$ and $u \cdot v1 = u \cdot v + u$. Thus v act as a control element and u as a fixed parameter. Correspondingly, if $\underline{i}$ and $\underline{j}$ were ingredients in r and s resp. then in the term t equal $r \cdot s$, $(\underline{i}, \underline{j}, f)$ would be an ingredient in t with $\underline{j}$ as the control element and $\underline{i}$ as the "raw material" to be recopied.

In the analysis of ingredients and how they change as we trace out the ids in a computation it soon becomes clear that these changes tend to respect the internal structure of ingredients and can be characterized as "homomorphisms" of this structure. So we make a definition.

Definition 6.2. A mapping $H: \text{Ing}(t) \rightarrow \text{Ing}(s)$ is a homomorphism (hom) iff for all $(\underline{i}, \underline{j}, f)$ in $\text{Ing}(t)$
 $H((\underline{i}, \underline{j}, f)) = (H(\underline{i}), H(\underline{j}), f)$.

Facts about Homomorphisms.

- 6.3a If $H, G: \text{Ing}(t) \rightarrow \text{Ing}(s)$ are homomorphisms which agree on $\text{Ing}_0(t)$ then $H = G$.
- 6.3b Any map $H: \text{Ing}_0(t) \rightarrow \text{Ing}(s)$ extends to a unique hom from $\text{Ing}(t)$ into $\text{Ing}(s)$.
- 6.3c Hom H is 1-1 on $\text{Ing}(t)$ iff H is 1-1 on $\text{Ing}_0(t)$.

There are infinitely many abstract ingredients in $\text{Ing}(t)$ only some of which represent real computations. We now use the idea of homomorphism to single these out in a precise manner.

We associate with a simple reduction $t \rightarrow s$ the following hom $H[t \rightarrow s]$. Let $\text{ID}(t \rightarrow s)$ be the set of ids accompanying $t \rightarrow s$ as defined in section 5. $H[t \rightarrow s]$ is the unique hom from $\text{Ing}(s)$ into $\text{Ing}(t)$ determined by the following (here q denotes any simple ingredient in $\text{Ing}_0(s)$).

$$H[t \rightarrow s](q) = \begin{cases} p & \text{if id}(p, q) \text{ is in } \text{ID}(t \rightarrow s), \\ (p, p', f) & \text{if id}((p, p', f), q) \text{ is in } \text{ID}(t \rightarrow s). \end{cases}$$

$H[t \rightarrow s]$ is 1-1 on $\text{Ing}_0(s)$ and hence is 1-1 on all of $\text{Ing}(s)$. We can now define the computationally meaningful ingredients.

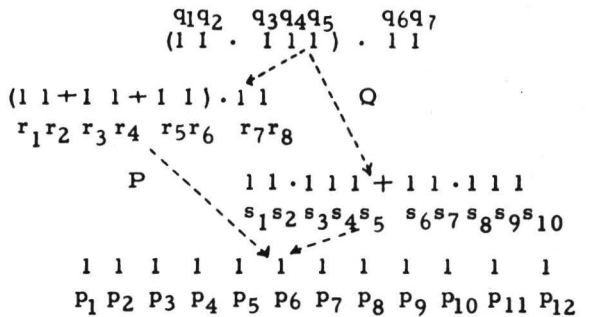
Definition 6.4. Let $P = t_1, \dots, t_n$ be an evaluation path for t ; thus $|t| = t_n$. Define the hom $H[P] = H[t_1 \rightarrow t_2] \circ H[t_2 \rightarrow t_3] \circ \dots \circ H[t_{n-1} \rightarrow t_n]$. The set of (real) ingredients of t w. r. t. P is the set $H[P](\text{Ing}_0(t_n))$ and is denoted by $\text{Ing}(t; P)$.

Note that $H[P]$ is a 1-1 map of $\text{Ing}_0(|t|)$ into $\text{Ing}(t)$. So the cardinality of $\text{Ing}(t; P)$ is equal to the integer denoted by the numeral $|t|$. We think of $H[P](q)$ as the unique "computational pattern" in t which is "responsible" for q via P .

7. Invariance of Ingredients.

How does $H[P]$ depend on P ? This is answered in Theorem 7.6, but first consider the following example.

Example 7.1.



Computing $H[P](p_9)$ and $H[Q](p_9)$ we get

$$H[P](p_9) = \underline{i} = ((q_1, q_4, \cdot), q_7, \cdot) \text{ and}$$

$$H[Q](p_9) = \underline{j} = ((q_1, q_7, \cdot), (q_4, q_7, \cdot), \cdot).$$

The basic difference between $\underline{i}$ and $\underline{j}$ is the temporal order in which the control elements q_4 and q_7 are operating and not the ultimate control relationships.

The above relationship between $\underline{i}$ and $\underline{j}$ is made into a basic equivalence relation.

Definition 7.2. Let $\underline{i}$ and $\underline{j}$ be ingredients in

$\text{Ing}(t)$. We define $\underline{i} \approx \underline{j}$ iff there is a sequence of pairs $(\underline{i}_1, \underline{j}_1), \dots, (\underline{i}_n, \underline{j}_n)$ of ingredients of $\text{Ing}(t)$ such that $\underline{i} = \underline{i}_n, \underline{j} = \underline{j}_n$, and for $k=1, \dots, n$ either

- 1) $\underline{i}_k = \underline{j}_k$,
- 2) there exists ingredients u, v, w and function symbols f, g such that $\{\underline{i}_k, \underline{j}_k\}$ equals $\{(\underline{u}, \underline{v}, f), (\underline{w}, g), (\underline{v}, \underline{w}, g), f)\}$, or
- 3) there exists k', k'' less than k such that $\underline{i}_k = (\underline{i}_{k'}, \underline{i}_{k''}, f)$ and $\underline{j}_k = (\underline{j}_{k'}, \underline{j}_{k''}, f)$.

Our Example 7.1 suggests that as we move from one computation path P to a simple variant P' of P we will find that $H[P](q)$ is changed into an equivalent ingredient $H[P'](q)$; this is the content of Theorem 7.7.

7.3 Facts about $\approx$.

- 7.3a) $\approx$ is an equivalence relation on $\text{Ing}(t)$.
- 7.3b) If $\underline{i} \approx \underline{j}$ and $\underline{i}' \approx \underline{j}'$ then $(\underline{i}, \underline{j}, f)$ is equivalent to $(\underline{i}', \underline{j}', f)$.
- 7.3c) If $\underline{i}_k \approx \underline{j}_k$ for $k = 1, 2, 3$ then $(\underline{i}_1, \underline{i}_2, f), (\underline{i}_3, g) \approx (\underline{j}_1, \underline{j}_3, g), (\underline{j}_2, \underline{j}_3, g), f)$.

Theorem 7.4. Let $H : \text{Ing}(s) \rightarrow \text{Ing}(t)$ be a hom. Then H preserves $\approx$, i.e. if $\underline{i}$ and $\underline{j}$ are in $\text{Ing}(s)$ and $\underline{i} \approx \underline{j}$ then $H(\underline{i}) \approx H(\underline{j})$.

Theorem 7.5. Let $G, H : \text{Ing}(s) \rightarrow \text{Ing}(t)$ be two homs and suppose that for all p in $\text{Ing}_0(s)$, $G(p) \approx H(p)$. Then $G(\underline{i}) \approx H(\underline{i})$ for all $\underline{i}$ in $\text{Ing}(s)$.

Lemma 7.6. Let t be any term with $\text{rk}(t) \leq n$.
 I. If P and Q are computation paths from t to t' forming a diamond and $\underline{i}$ is a real ingredient in $\text{Ing}(t')$ then $H[P](\underline{i}) \approx H[Q](\underline{i})$.
 II. If P and Q are computation paths from t to t' and $\underline{i}$ is a real ingredient of $\text{Ing}(t')$ then $H[P](\underline{i}) \approx H[Q](\underline{i})$.

The Lemma is proved by induction on n and dovetailing I and II. Lemma 5.3 plays a major role. As an immediate consequence of this Lemma we get the main result of this section.

Theorem 7.7. (Invariance of Ingredients.) Let P and Q be two evaluation paths for t . Then for all p in $\text{Ing}_0(t)$, $H[P](p) \approx H[Q](p)$.

There is a natural direction of simplification built into the equivalence relation $\approx$, namely we will say that $((\underline{i}, k, f), (\underline{j}, k, f), g)$ reduces to $((\underline{i}, \underline{j}, g), k, f)$. We then define an ingredient to be in normal form if it can't be reduced nor does it contain any ingredients which can be so reduced. If one tries to make a cost estimate for various computation paths it appears that if $H[P](q)$ is in normal form then P is the least expensive computation path. Such estimates are not so easy to make in a reliable way but the examples which have been studied are quite suggestive.

8. Final Remarks.

The ideas presented in this paper can straightforwardly be applied to other deductive-computational systems such as the Lamda Calculus and Curry's Combinatorial Logic.

The next step in this investigation extends our analysis to the full 1st Order Intuitionistic Theory of Peano Arithmetic. The semantics developed provides a means whereby with each theorem A we can associate a collection $M(A)$ of procedures extracted from the proofs of A . This will be the subject of a second paper being prepared under an NSF grant through Boston University.

Two examples will give some idea of how this semantics works.

Suppose A is the sentence $t = |t|$. Then $M(A)$ will contain the procedures which compute the isomorphisms $H[P] : \text{Ing}_0(|t|) \rightarrow \text{Ing}(t)$ determined by each evaluation path P for t .

Suppose A is the sentence $B \supset C$. Then a procedure in $M(A)$ will be a method for converting a procedure in $M(B)$ into a procedure in $M(C)$. If A is the sentence $B \vee C$ then a procedure in $M(A)$ would construct a sentence D (which is either B or C) and a procedure in $M(D)$.

Negation is defined in terms of implication $\sim A := A \supset 1 = 11$. This at first seems a little odd because it says that you have a procedure which will transform any procedure in $M(A)$ into a procedure in $M(1 = 11)$, and, of course, with the means that we allow there are no procedures in $M(1 = 11)$. If we look at a concrete example like $11 = 111 \supset 1 = 11$, this becomes less mysterious since with a little thought we can see how a set of rules, purportedly establishing an isomorphism between $\text{Ing}_0(11)$ and $\text{Ing}_0(111)$ could be actually modified so as to establish an isomorphism between $\text{Ing}_0(1)$ and $\text{Ing}_0(11)$.

The Induction Schema works out very nicely. For if we have procedures M, M' in $M(A(1))$ and $M(\forall x (A(x) \supset A(x1)))$ respectively, then our procedure in $M(\forall x A(x))$ starts with M and makes a recursive call to M' .

References.

- (1) Yessenin-Volpin, A.S., "Ultraintuitionism and the Antitraditional Program for the Foundations of Mathematics", Proceedings of the Summer conference on Intuitionism and Proof Theory at Buffalo, New York, 1968, North Holland.
- (2) Sussman, G., "A Computational Model of Skill Acquisition", Doctoral Thesis, MIT, 1973.