GENTZEN-TYPE FORMAL, SYSTEM REPRESENTING PROPERTIES OF FUNCTION AND ITS IMPLEMENTATION

Toshio Nishimura
Dept. of Math
Univ. of Tsukuba
Ibaragi Pref. Japan

Masakazu Nakanishi and Morio Nagata
Faculty of Eng., Keio Univ.
Yokohama, Japan

Yoshiaki Iwamaru
The Mitsui Bank Ltd
Tokyo, Japan

ABSTRACT

We describe a theorem-prover (called TKP 1), which is based on a Gentzen-type formal system [14). TKP 1 can directly deal with functionals and the composition of functionals, it comprises the fixed point operator and a kind of facility for induction. Let us attempt to prove P(F(x, y)) for F(x,y) such that F(x, y) = $\bigvee_{n=0}^{\infty} f^n(x, y)$. Provided that P(F(x,y)) can be obtained from P f f^n(x,y)) n=0, 1,2,..., TKP 1 automatically gives the induction hypothesis P (f^n (x, y)), and then prove P(f^{n-1}, (x,y)). It. can efficiently make proving procedure for properties of recursive programs. We can supply assumptions and definitions at will. TKP 1 displays an easily read proof-figure.

KEY WORDS

LISP, automatic theorem proving, inductions, proving programs correct, fixed point operation, Gent/en-type formal system, composition of functional, infinitely long expression.

1. INTRODUCTION

Many methods for proving theorems about programs have been studied. ([1), [2], [6] etc.) Some of them are works on the fixed point operator (2), [15]. We have tried an implementation of an automatic theorem proving system, called TKP 1 (Tsukuba-Keio Prover No. 1), which is based on a Gentzen-type formal system representing the properties of functions (14). The plausibility and the completeness of this system are certified by the monotonic functional interpretation [13], 1141. Moreover, the cut-elimination theorem (3) holds in this system. The basic expressions of this system are functionals and composition of functionals, wherein the usual logical formula is a kind of functional. Connectives between functionals are decomposed in a way similar to logical connectives. Besides, the system includes the operator for infinitary sum, from which we can apply the fixed point operator in this system. Let us consider to prove $\bigvee_{n=0}^{\infty} f^n(x) \to G$.
In this case the following sequents must be proved.

$$f^n(x) \to G \qquad (n=0,1,2,\ldots)$$

In order to prove these sequents,we may use the mathematical induction.
First, we prove $F^{\circ}(x) \to G$. Next $f^{n-1}(x) - G$ is tried to prove under the induction hypothesis $f^n(x) \to G$.
Because TKP 1 can directly deal with the composition of functionals and provides a kind of facility of fixed point operator and induction, TKP 1 can efficiently process the proofs for properties of recursive programs. As TKP 1 is developed as the cut-less system, it can easily decompose a theorem to be proved into some subtheorems. And it accomplished the proofs of most problems described in (8), and moreover, acceding to user's demands, assumptions and definitions are acceptable by TKP 1, and it displays the proof-figure in a plain form after problems are proved. In section 2, we shall explain the outline of the formal system and in 3 describe the formal system, The way to prove the problem shall be explained in 4. The outline of implementation is given in 5. In 6 we discuss some improvements and its future applications of TKP 1.

2. OUTLINE OF FORMAL SYSTEM

Now we shall briefly explain our system, which is mainly based on 2 valued logic. The underlying formal system is given in [14], and it can be interpreted by monotonic functionals. The plausibility and the completeness can be certified in a way similar to [13].
Let F and G be functions of the type α->β and β-> *Y* respectively. We denote the composition of F and G by

F- G (F- G(x)=G(F(x)))

If F and G are compatible and of the same type, the join of F and G is denoted by

FVG

A formula P may have the value true (i) or false (ϕ). Let the function ϕ always have the value ϕ, and the function I the value i. Then a formula p can be considered as functions and I according that they have values ϕ and I respectively. And the expression PG has the value G if P is true and ϕ if P is false.
Let us consider the following expression

if P (x) then y else f(x)

The value of this expression is y if P(x) is true.

and is f(x) if P(x) is false. We can represent this by the following composition.

$$P(x) \cdot y \vee \neg P(x) \cdot f(x)$$

where P(x) denotes the negation of P(x).
Next, we consider the program

begin F; G;
loop : **if** P **then begin** H;K;**goto** *loop* **end end**

This can be represented by

$$F \cdot G \left(\overset{\infty}{\underset{n=0}{\vee}} (P \cdot H \cdot K)^n \right) \cdot \neg P$$

where $A^0 = I$, $A^1 = A$, $A^2 = A \cdot A^1, \ldots, A^{n+1} = A \cdot A^n$
and $\overset{\infty}{\underset{n=0}{\vee}} A^n = A^0 \vee A^1 \vee A^2 \vee \ldots$
When A and B are formulas, we can consider that the composition A B means 'A and B' , because $I \cdot I = I$, $I \cdot \Phi = \Phi$, $\Phi \cdot I = \Phi$ and $\Phi \cdot \Phi = \Phi$. The similar method concerning I and Φ is shown in [2).
Similarly the join AVB beans *A or B', because $I \vee I = I$, $I \vee \Phi = I$, $\Phi \vee I = I$ and $\Phi \vee \Phi = \Phi$.
The program

begin i: = 1; s: = 0;
loop : if](i > N) **then begin** s: = s + a_i;
i: = i + 1; **goto** *loop* **end end**

will show the same result as $s := a_1 + \ldots + a_N$.
This represented by the expression of the form

$$(i:=1)(s:=0) \overset{\infty}{\underset{n=0}{\vee}} (\quad (i > N)(s := s + a_i)(i := i+1))^n \ (i>N) \rightarrow s := a + \ldots + a$$

which is called a sequent. ' F - G' means that G is an extension of F. In the 2-valued case, for formulas A and B, *A > B' means that A implies B, and so is identical with A-* B in Gentzen's original form [3J.
Now, we shall consider the following recursive definition of the function F (of type a).

F(x,y) =_if P(x) then y else h(F(k(x),y))

It is well known that F can be defined as the least fixed point J/o f"('o) (denoted by *dl),* if we introduce the following function f of the type a-a 111).

$$f^0(\Phi) = I(\Phi) = \Phi$$
$$f^{n+1}(\Phi) = \text{if } p(x) \text{ then } y \text{ else } h(f^n(\Phi)(k(x),y))$$
(represented as $p(x) \cdot y \vee \neg p(x) \cdot h(f^n(\Phi)(k(x),y))$)

On the other hand the function G of the same type as F is defined by

$$G(x,y) = \text{if } p(x) \text{ then } y \text{ else } (G(k(x),h(y))$$

Then we can rewrite G as follows.

$$G(x,y) = \partial g(x,y) = \Phi \vee \overset{\infty}{\underset{n=0}{\vee}} (p(x) \cdot y \vee \neg p(x) \cdot g^n(\Phi)(k(x), h(y)))$$

In order to prove that F is the same function as G, it is sufficient to show the following two sequents:

$$\partial f(x,y) \rightarrow g(x,y) \text{ and } g(x,y) \rightarrow \partial f(x,y)$$

We shall use the first of these to illustrate the proving procedure of our theorem prover discussed in 4. As is shown in the following section, rules of inference will be given symmetrically for the left hand side and the right hand side of -* .

3. THE FORMAL SYSTEM

Here, we describe simply the axioms and rules of inference. The meanings of most symbols employed here have been briefly explained in the previous section . Herein we use Greek capital letters, r , A etc. , to represent a finite set of functionals such as F». . . . ., F™ . In Miner's LCF system UOJ, ",M in both sides denotes the conjunction. In our system," in the left hand side of ■+ represents the conjunction and ", " in the right side denotes the disjunction. A proof-figure is a tree constructed by sequents, in which every uppermost sequent *is* an axiom and upper sequents and a lower sequent are connected by a rule of inference.

3. 1 Axioms

In general, various assumptions are acceptable by TKP 1 as axioms, however, we establish our system using only logical axioms as the most basic ones.
Logical axioms are sequents of the following form.

1. $\Phi \rightarrow \Delta$, where Φ is the particular constant.
2. $\Gamma_1, F, \Gamma_2 \rightarrow \Delta, F, \Delta_2$
3. $\Gamma_1, AP_1 \ldots P_k B, \Gamma_2 \rightarrow \Delta_1, AP_1 \ldots P_k B, \Delta_2$, where P_i is a formula.

3. 2 Rules of Inference

3. 2. 1 Rules of Replacement

(1) IF, FL $\Phi \vee F$ and $F \vee \Phi$ can be replaced by F, the converse also hold.
(2) ΦF and $F \Phi$ can be replaced by Φ, the converse also holds.
(3)].I a n]Φ: a n be replaced by $ and I respectively, the converse also holds.
(4) $P \equiv Q$, $P \supset Q$, $\neg(P \cdot Q)$, $\neg(P \vee Q)$ and $\neg\neg P$ can be replaced by $(P \supset Q) \wedge (Q \supset P)$, $\neg P \vee Q$, $\neg P \vee \neg Q$ $\neg P \cdot \neg Q$ and P respectively, the converse also holds.
(5) If 4> occurs in the left hand side of a sequent, then the left side can be replaced by * and $ in the right can be omitted.

3. 2. 2 Rules of Inference with respect to Logical Connectives

(1) $\vee$ - left

$$\frac{\Gamma_1, AFB, \Gamma_2 \rightarrow \Delta \qquad \Gamma_1, AGB, \Gamma_2 \ \Delta}{\Gamma_1, A \cdot (F \vee G) \cdot B, \Gamma_2 \rightarrow \Delta}$$

$\vee$ - right

$$\frac{\Gamma \rightarrow \Delta_1, AFB, AGB, \Delta_2}{\Gamma \rightarrow \Delta_1, A \cdot (F \vee G) \cdot B, \Delta_2}$$

where F and G are compatible functionals.

(2) ∂ - left

$$\frac{\Gamma_1, A\cdot F^n(\phi)\ B, \Gamma_2 \to \Delta \qquad n=0,1,2,\ldots}{\Gamma_1,\ A\cdot \partial F\cdot B, \Gamma_2 \to \Delta}$$

∂ - right

$$\frac{\Gamma \to \Lambda_1, A\cdot F^m(\phi)\cdot B, \Delta_2, \quad A\cdot \partial F\cdot B}{\Gamma_1,\ A\cdot \partial F\cdot B,\ \Delta_2}$$

From the compatibility of $F^n(\phi)$ and ∂F, these rules of inference are reasonable.

(3) $\forall$ - left

$$\frac{\Gamma_1,\ A\cdot P(g/h)\cdot B, \Gamma_2, A\cdot \forall h\cdot P\cdot B \to \Delta}{\Gamma_1, A\cdot \forall h P\cdot B, \Gamma_2 \to}$$

where g is an arbitrary functional of the same type as h, and P [g/h) denotes the result obtained by replacing h for g. Moreover, we shall use the symbols $\forall$ -right, $\forall$-left, aright and J-left as follows.

$\forall$ - right

$$\frac{\Gamma \to \Lambda_1,\ A\cdot P(f/h)\cdot B, \Delta_2}{\Gamma \to \Delta_1,\ A\cdot \forall h P\cdot B, \Delta_2}$$

where f is an arbitrary free variable of the same type as h not contained in the lower sequent.

(4) $\exists$ - left

$$\frac{\Gamma_1,\ A\cdot P(f/h)\cdot B, \Gamma_2 \to \Lambda}{\Gamma_1,\ A\cdot \exists hP\cdot B, \Gamma_2 \to \Lambda}$$

where f is a variable satisfying the condition in the description of $\forall$- right

$\exists$ - right

$$\frac{\Gamma \to \Delta_1,\ A\cdot P(g/h)\cdot B, \Delta_2,\ A\cdot \exists hP\cdot B}{\Gamma \to \Delta_1,\ A\cdot \exists hP\cdot B,\ \Delta_2}$$

where g is an arbitrary functional satisfying the condition in the description of $\forall$ - left.

3. 2. 3 Other Rules of Inference

As the occasion demands, we can add the following practical rules.

$$\frac{\Gamma_1, \Gamma_2 \to \Delta_1, C, \Delta_2 \qquad \Gamma_1, C, \Gamma_2 \to \Delta_1, \Delta_2}{\Gamma_1,\ \Gamma_2 \to \Delta_1,\ \Delta_2}$$

and

$$\frac{A_1, \ldots, A_m \to B_1, \ldots, B_n \qquad F \to G}{A_1 F, \ldots, A_m F \to B_1 G, \ldots, B_n G}$$

where A_M . . . , Am , B, , . . . , B« are of type $\alpha \to \beta$ and F and G of type $\beta \to \gamma$. These rules* are easily verified.

4. PROVING PROCEDURE OF TKP 1 AND ITS EXAMPLE

In this section, we illustrate the theorem prover based on the formal system which is described in the previous section.

4.1 Illustrative Example

We shall illustrate the proving procedure of TKP 1 by giving the proof of the following example :

$$F(x,y) \Leftarrow \underline{\text{if}}\ p(x)\ \underline{\text{then}}\ y\ \underline{\text{else}}\ h(F(k(x),y))$$
$$G(x,y) \Leftarrow \underline{\text{if}}\ p(x)\ \underline{\text{then}}\ y\ \underline{\text{else}}\ G(k(x),\ h(y))$$
$$\text{Then} \qquad F(x,y) = G(x,y)$$

As described in the previous section, F and G can be defined by the series of formulas :

$$f^0(x,y) = g^0(x,y) = \phi \tag{4.1}$$
$$f^{n+1}(x,y) = p(x)\cdot y \vee \neg p(x)\cdot h(f^n(k(x),y)) \tag{4.2}$$
$$g^{n+1}(x,y) = p(x)\cdot y \vee \neg p(x)\cdot g^n(k(x),h(y)) \tag{4.3}$$
$$F(x,y) = \partial f(x,y) = \bigvee_{n=0}^{\infty} f^n(x,y) \tag{4.4}$$
$$G(x,y) = \partial g(x,y) = \bigvee_{n=0}^{\infty} g^n(x,y) \tag{4.5}$$

where $f^n(\phi)$ and $g^n(\phi)$ are abbreviated simply by f^n and g^n, respectively.
Let us illustrate how to prove

$$F(x,y) \to G(x,y)$$

in our theorem prover. We adopt (4.2), (4.3), (4.4) and (4.5) as definitions, and the following as an assumption :

$$h(\phi) = \phi \tag{4.6}$$

(4.1) is provided in TKP 1.
As

$$F(x,y) \to G(x,y) \tag{4.7}$$

is not an axiom, (4.7) is automatically transformed to (4.8) by (4.4) and (4.5).

$$\bigvee_{n=0}^{\infty} f^n(x,y) \to \bigvee_{n=0}^{\infty} g^n(x,y) \tag{4.8}$$

Then TKP 1 attempts to prove

$$f^0(x,y) \vee f^{n+1}(x,y) \to \bigvee_{n=0}^{\infty} g^n(x,y) \tag{4.9}$$

At the same time, TKP 1 automatically generates the induction hypothesis as an assumption

$$f^n(x,y) = g^n(x,y) \tag{4.10}$$

In (4.9) it is clear that f^0-case is provable, so we trace the f^{n+1}- case.

(4.9-1) $f^{n+1}(x,y) \to g^{n+1}(x,y)$

Then we have

(4.10-2) $p(x)\cdot y \vee \neg p(x)\cdot h(f^n(k(x),y)) \to g^{n+1}(x,y)$

The p(x)·y - case is also easily shown, because, by (4.3),

(4.10-3) $p(x)\cdot y \vee \neg p(x)\cdot h(f^n(k(x),y)$
$\rightarrow p(x)\cdot y \vee \neg p(x)\cdot g^n(k(x),h(x))$

The prover tries to prove

(4.10-4) $\neg p(x)\cdot h(f^n(k(x),y))$
$\rightarrow p(x)\cdot y,\ \neg p(x)\cdot g^n(k(x),h(y))$
under the hypothesis of the induction (4.10)

(4.10-4) and the result obtained by the replacement of f^n with g^n in (4.10-4), where this replacement is possible according to the hypothesis (4.10), have no logical connectives and do not match to definitions and assumptions (4.1) ~ (4.6) and (4.10). Therefore TKP 1 fails to prove (4.10-4), and it must backtrack to (4.8). Now it attempts to prove

$$f^0(x,y) \vee f^1(x,y) \vee f^{n+2}(x,y) \rightarrow \bigvee_{n=0}^{\infty}(x,y) \quad (4.11)$$

At the same time, TKP 1 automatically generates the induction hypothesis $f^n(x,y) = g^n(x,y)$ and $f^{n+1}(x,y) = g^{n+1}(x,y)$

The case $f^0(x,y) \rightarrow \bigvee_{n=0}^{\infty}(x,y)$ is trivial. On the other hand, the case $F^1(x,y) \rightarrow \bigvee_{n=0}^{\infty}(x,y)$ is provable by using the assumption (4.6). Now we show below the proof of

$$f^{n+2}(x,y) \rightarrow g^{n+2}(x,y) \quad (4.12)$$

which is accomplished by TKP 1

$f^{n+2}(x,y) \rightarrow g^{n+2}(x,y)$
by definitions (4.2) and (4.3)
$p(x)\cdot y \vee \neg p(x)\cdot h(f^{n+1}(k(x),y)) \rightarrow p(x)\cdot y \vee \neg p(x)\cdot g^{n+1}(k(x),h(y))$
by inference rule of $\vee$ - left
$p(x)\cdot y \rightarrow p(x)\cdot y,\ \neg p(x)\cdot g^{n+1}(k(x),h(y))$ (Axiom 2.)
$\neg p(x)\cdot h(f^{n+1}(k(x),y)) \rightarrow p(x)\cdot y,\ \neg p(x)\cdot g^{n+1}(k(x),h(y))$
by $f^{n+1}(x,y) = g^{n+1}(x,y)$
$\neg p(x)\cdot h(g^{n+1}(k(x),y)) \rightarrow p(x)\cdot y,\ \neg p(x)\cdot g^{n+1}(k(x),h(y))$
by definition (4.3) and $g^{n+1}(x,y)$
$\neg p(x)\cdot h\{p(k(x))\cdot y \vee \neg p(k(x))\cdot g^n(k(x),h(y))\} \rightarrow \neg p(x)\cdot f^{n+1}(k(x),h(y))$
by definition (4.2) and decomposition of function
$\neg p(x)\{p(k(x))\cdot h(y) \vee \neg p(k(x))\cdot h(g^n(k^2(x),h(y)))$
$\rightarrow \neg p(x)\cdot\{p(k(x))\cdot h(y) \vee \neg p(k(x))\cdot h(f^n(k^2(x),h(y)))\}$
by $\vee$ - left, $\vee$ - right and $f^n(x,y) = g^n(x,y)$
$\neg p(x)\cdot p(k(x))\cdot h(y) \rightarrow \neg p(x)\cdot p(k(x))\cdot h(y),\ \neg p(x)\cdot h(g^n(k^2(x),h(y)))$ (Axiom 2.)
$\neg p(x)\cdot\neg p(k(x))\cdot h(g^n(k^2(x),h(y))) \rightarrow \neg p(x)\cdot p(k(x))\cdot h(y),\ \neg p(x)\cdot h(g^n(k^2(x),h(y)))$ (Axiom 2.)

This completes the proof of

$$F(x,y) \rightarrow G(x,y).$$

4.2 Application of Rules of Inference

First, TKP 1 examines whether the same expression exists in both sides of a sequent or not and whether an undefined element exists in the left hand side of a sequent or not. If there exists a same expression in both sides or in the left hand side in a sequent, the sequent is provable. If a sequent is not provable and includes some logical connectives, TKP 1 transforms it according to the top level logical connectives in the left and right hand side of the sequent.

We shall show the transformation rules. In these rules the left hand side of $\Rightarrow$ is a given sequent and the right hand side is transformed sequents.

(1) $\neg$ (not)

$\Gamma, \neg A, \Delta \rightarrow \Pi \Rightarrow \Gamma, \Delta \rightarrow A, \Pi$
$\Gamma \rightarrow \Delta, \neg A, \Pi \Rightarrow A, \Gamma \rightarrow \Delta, \Pi$

(2) $\wedge$ (and)

$\Gamma, A \wedge B, \Delta \rightarrow \Pi \Rightarrow \Gamma, A, B, \Delta \rightarrow \Pi$
$\Gamma \rightarrow \Delta, A \wedge B, \Pi \Rightarrow \Gamma \rightarrow \Delta, A, \Pi$ and $\Gamma \rightarrow \Delta, B, \Pi$

(3) $\vee$ (or)

$\Gamma, A \vee B, \Delta \rightarrow \Pi \Rightarrow \Gamma, A, \Delta \rightarrow \Pi$ and $\Gamma, B, \Delta \rightarrow \Pi$

(4) Implication and Equivalence

Implication and equivalence are replaced by $\neg A \vee B$ and $(A \supset B) \wedge (B \supset A)$ respectively.

(5) Decomposition of Functions

McCarthy's conditional expression is based on the form of if - then - else, e.g.
if p then c1 else e2.
Such an expression p is called a p - type expression. In the proving process, the following rules are applied to the expression which contains p - type expressions,

(i) f (p) is decomposed into p
(ii) f(p *op* k) and f(k *op* p) are decomposed respectively into p *op* f(k) and f(k)op p, where *op* is a logical connective and k is not a p - type expression,
(iii) $f(\neg p)$ is decomposed into $\neg p$, where p is a p - type expression.

(6) Undefined Element

$f^0 \Rightarrow \phi \qquad \phi \Rightarrow \omega$

These rules are applied in the following order.

$\neg$ in both side, $\wedge$ in left side, $\vee$ in right side, $\vee$ in left side, $\wedge$ in right side, implication in both sides, equivalence in both sides, decomposition of function and undefined element.

4.3 Application of Assumptions and Definitions

When a sequent can not be transformed by rules

of inference and is not provable, TKP 1 attempts to apply assumptions and definitions which are given by the user to the sequent. In applications of assumptions and definitions, a pattern-matching facility is required. In matching processes for assumptions and definitions, TKP 1 first tries to make assumptions match and then definitions afterwards. When an assumption matches two or more subexpressions of a sequent, it first applies to the innermost expression contained in the leftmost term. If an assumption (or a definition) matches a subexpression, then the right part of the assumption (or the definition) is substituted for the subexpression. The prover tries to prove each new sequent. When TKP 1 can not prove any new sequent, *it* will *try to* apply the next assumption (or definition) to the old sequent. If a sequent transformed by an assumption (or a definition) is the same form as a sequent which already appeared in the proving process, it is considered that the matching fails.

4.4 Induction

There are situations in which our formal system must prove the sequents including infinitary sums. Therefore TKP 1 has the procedure for a fixed point operator and mathematical induction. Oir system employs the induction with respect to the number of applications of functions. *The* procedure of transformation of a fixed point operator is as follows. In this section ex(x) denotes an expression with subexpression x.

(1) $ex(\partial f)$ is transformed to $ex(f^0) \vee ex(f^{n+1})$, $(n = 0, 1, 2, \ldots)$

(2) If the prover can not prove the expression, $ex(\partial f)$ is transformed to $ex(f^0) \vee ex(f^1) \vee ex(f^{n+2})$, $(n = 0, 1, 2, \ldots)$

(3) The prover generates automatically the induction hypothesis for i^{n+1} or $^{n+2}$ to prove the given proposition for all n.

The prover attempts to prove the transformed expression. If it is proved, a proof tree which denotes the proving procedure is constructed. If it fails to prove them, it automatically prints out the trial process of the proof. This output shows us what kind of assumption is required.

5. IMPLEMENTATION

5.1 TKP 1 and KLISP

Our program, called TKP 1, is written in KLISP (Keio LISt Processor) language which is a subset of LISP 1.5 [9] [1]. KLISP interpreter on TOSBAC 3400/30 (24 bits/word, 16K words) has about 4K cells of free list. As it has GLISP system which translates programs in M - expression into S - expression, so we have written TKP 1 in M - expression.
TKP 1 consists of three phases ; *translator,* *prover* and *viaualizer.* These are successively executed. Problems in input form are read by the translator of TKP 1 and are translated into internal forms which can be easily manipulated by the prover. When the prover succeeds in proving the given problems, the prover puts proof trees in S - expression form into an auxiliary memory. Finally the visualizer receives these proof trees and displays the proof figures in printed form.

5.2 Input Form of TKP 1

Using the example of 4. 1, let us explain the input form of TKP 1. Inputs for TKP 1 are assumptions, definitions and a sequent to be proved. Each of assumption, definition and a sequent is called a statement, and any set of statements which contains a sequent is called a problem. In the case of the example of 4. 1, inputs for TKP 1 are as follows.

$$? F(\$X, \$Y) \rightarrow ? G(\$X, \$Y) \qquad (5.1)$$

$$H(@) \rightarrow @ \qquad (5.2)$$

$$: F\langle N+1\rangle(X, Y) = p(X).Y \vee \backslash P(x).\ H(F\langle N\rangle (K(X), Y)) \qquad (5.3)$$

$$: G\langle N+1\rangle(X, Y) = P(X).Y \vee \backslash P(X).\ G\langle N\rangle (K(X), H(Y)) \qquad (5.4)$$

(5. 1) is a sequent to be proved. In our system "?" and "@" means ∂ and ω respectively.

$$\$x$$

where x is an identifier. For example, $a \cdot (b + c) \rightarrow a \cdot b + a \cdot c$ is written as follows.

$\$A * (\$B + \$C) \rightarrow \$A * \$B + \$A * \$C$

(5. 3) and (5. 4) are definitions. A general form of a definition is

$$:r = e$$

where r is usually $f\langle n\rangle (x_1, x_2, \ldots, x_m)$ $(m \geqq 0)$, n must be "N + i" (i is a positive integer) and e is an expression.
A definition is considered as a kind of an assumption by the processor, but a formal parameter x_i needs not to be written as an arbitrary element $\$x_i$.

5.3 Translation and Retranslation

Translator translates (5. 1) and (5. 4) into internal forms. For example (5. 1) is translated to

(*ARROW((* F)(= X)(= Y)))(((* G) (= X) (= Y)))

If the prover cannot prove a problem, no proof trees are generated. When it fails to prove the problem, the trial process to prove this is always displayed.
When a proof tree is generated, the prover constructs a proof tree and puts it into an auxiliary memory.
Visualizer translates this tree to a proof figure using the structure of stairs corresponding to the level of sequent in the proof tree. An example of the proof figure will be shown in the APPENDIX.
The detailed grammer of the input form and trans-

lation rules are described in [5],

6. CONCLUSIONS

Many computer programs to prove theorems about programs have been implemented{(I), (6) etc.). One of the main features of our theorem prover is based on a Gentzen - type formal system. Most significant features of TKP 1 are the following ; (1) Since cut operation is not required in our formal system, TKP 1 can be implemented as a fully automatic theorem prover. (2) The fixed point operator can be decomposed as it is a logical operator. (3) An induction hypothesis is automatically generated. (4) TKP 1 performs a part of course-of-values induction (7). A Gentzen - type theorem prover which has the facility to display proof figures is very useful to show readable and understundable proof procedures. If we make an interactive theorem prover, this feature is very powerful. The other feature involves using an inductive method and fixed point theorem. Some programs using induction methods have been already studied (11 (.10), in our prover, propositions to be proved are not restricted by a particular programming language. Providing assumptions and definitions to this prover, results in a large generality and flexibility with our program. In spite of the ability to prove all examples described in (8), our formal system and our theorem prover are quite concise, namely, 4K cells are available for the prover and data.

Considering the facts described above, we summarize here some future aspects of the study of our program and formal system.

(1) The program can be extended to accept a kind of dynamic assumptions, e.g. , procedural assumptions (4).

(2) It may be anticipated that this prover should contain useful algebraic theorems, and theorems which are unuseful should be automatically deleted from the system. Theorems prepared should be selected according to user[1] s requirements.

(3) We concider that our theorem prover will have more extensive applications when provided with an interactive facilities.

REFERENCES

[1] Boyer, R.S. and Moore J.S. "Proving Theorems About LISP Functions, "Proceedings of 3rd 1JCA1, 1973, pp. 486 - 493

(2) deBakker, J.W. and deRoever, W.P. "A Calculus for Recursive Program Schemes," in Automata, Languages and Programming, pp. 167-196(ed. Nivat, M.),North-Holland, Amsterdam, 1973.

[3] Gentzen, G. "Untersuchungen uber das logische Schliesenl, II," Math, Zeitschr 39, 1934, pp. 176 - 210, pp. 405 - 431.

(4) Hewitt, C. Description and Theoretical Analysis (Using Schemata) of PLANNER : A Language for Proving Theorems and Manipulating Models in a Robot," ph. D. Thesis, MIT, Cambridge, Massachusetts, 1971.

[5] Iwamaru, Y., Nagata, M. , Nakanishi, M. and Nishimura, T. "implementation of Gentzen - type Formal System Representing Properties of Functions, " Comment. Math. Univ. St. Pauli, 23 - 1, 1974, pp. 45 - *66,*

[6] King, J. C. "A Program Verifier," Ph.D. Thesis, Carnegie - Mellon University, Pittsburgh , Pennsylvania, 1969.

[7) Kleene, S.C. "Introduction to Metamathematics," D. Van Nostrand Company, Inc., Princeton, New Jersey, 1952.

[8] Manna, Z. , Ness, S. and Vuilliemin, J. "Inductive Methods for Proving Properties of Programs," Proceedings of An ACM Conference on Proving Assertions About Programs, New Mexico State University, New Mexico, January 6 - 7 , 1972, pp. 27 - 50.

[9) McCarthy, J., et al. , "LISP 1. 5 Programmer's Manual," MIT Press, Cambridge, Massachusetts, 1962.

[10] Milner, R. "Implementation and Applications of Scott's Logic for Computable Functions," Proceedings of an ACM Conferences on' Proving Assertions About Programs, New Mexico State Univ. , New Mexico, January 6 - 7 , 1972, pp. 1 - 6.

(11) Morris, J.H. "Another Recursion Induction Principle, " CACM, Vol. 14, No. 5, 1971, pp. 351 - 354.

(12) Nakanishi, M. "KLISP Reference Manual (Revised Version)," Keio Institute of Information Science, Keio Univ. , Yokohama, 1970 (in Japanese).

(13) Nishimura, T. "Gentzen - Style Formulation of Systems of Set-calculus," Comment. Math. Univ. St. Pauli, 23 - 1, 1974, pp. 29 - 36.

(14) Nishimura, T. "Gentzen - type Formal System Representing Properties of functions, "Comment. Math. Univ. St.Pauli, 23 - 1, 1974, pp. 37 - 44.

[15] Park, D. "Fixpoint Induction and Proofs of Program Properties/' in Machine Intelligence 5, pp. 59 - 78 (eds. Meltzer, B. and Michie, D.), Edinburgh Univ. Press, Edinburgh, 1969.

[161 Strachey, C. "Towards a Formal Semantics/' in Formal Language Description Languages for Computer Programming, pp. 1 98 - 220 (ed. Steel, T. B.), North-Holland, Amsterdam, 1966.

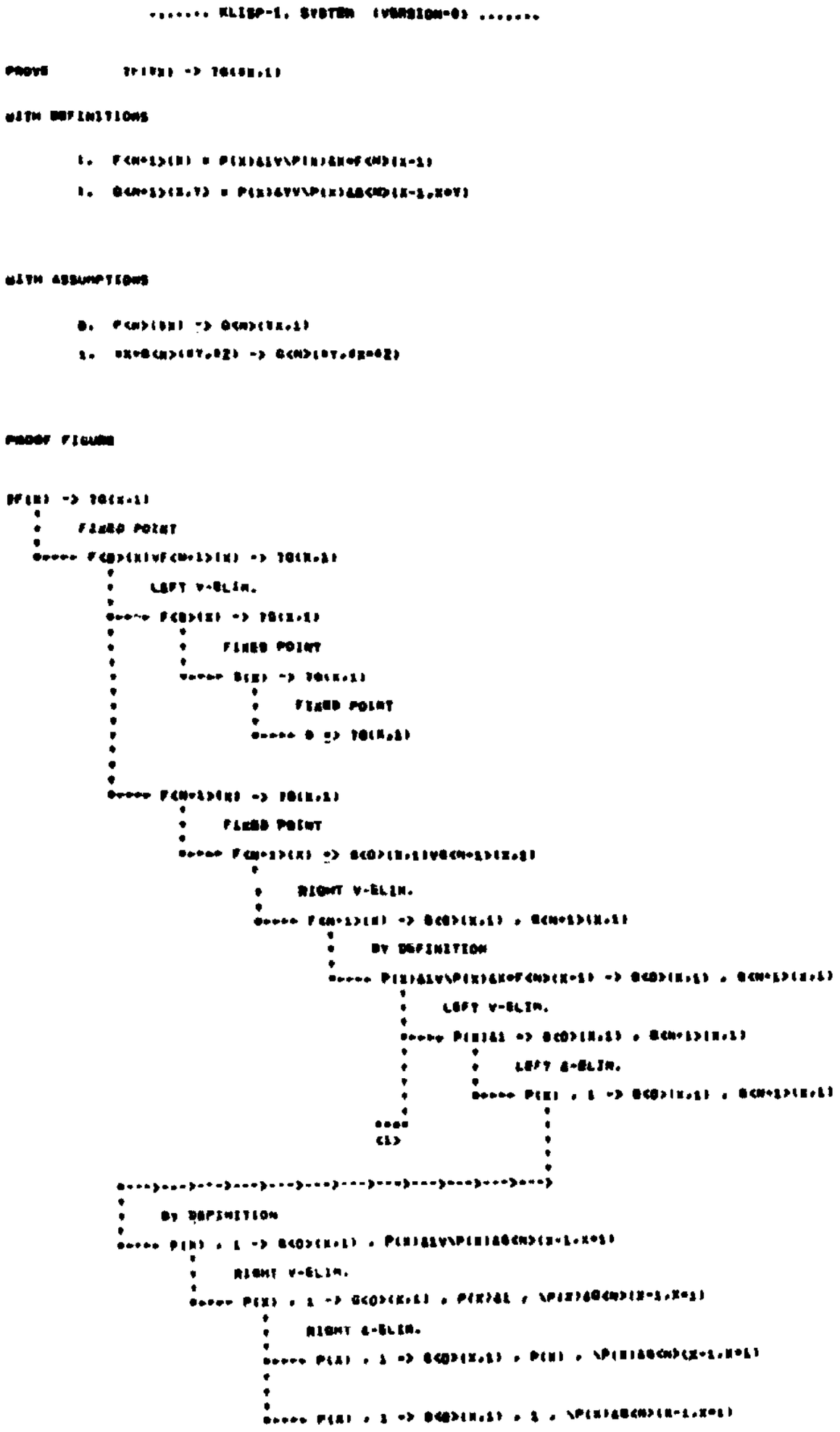
APPENDIX

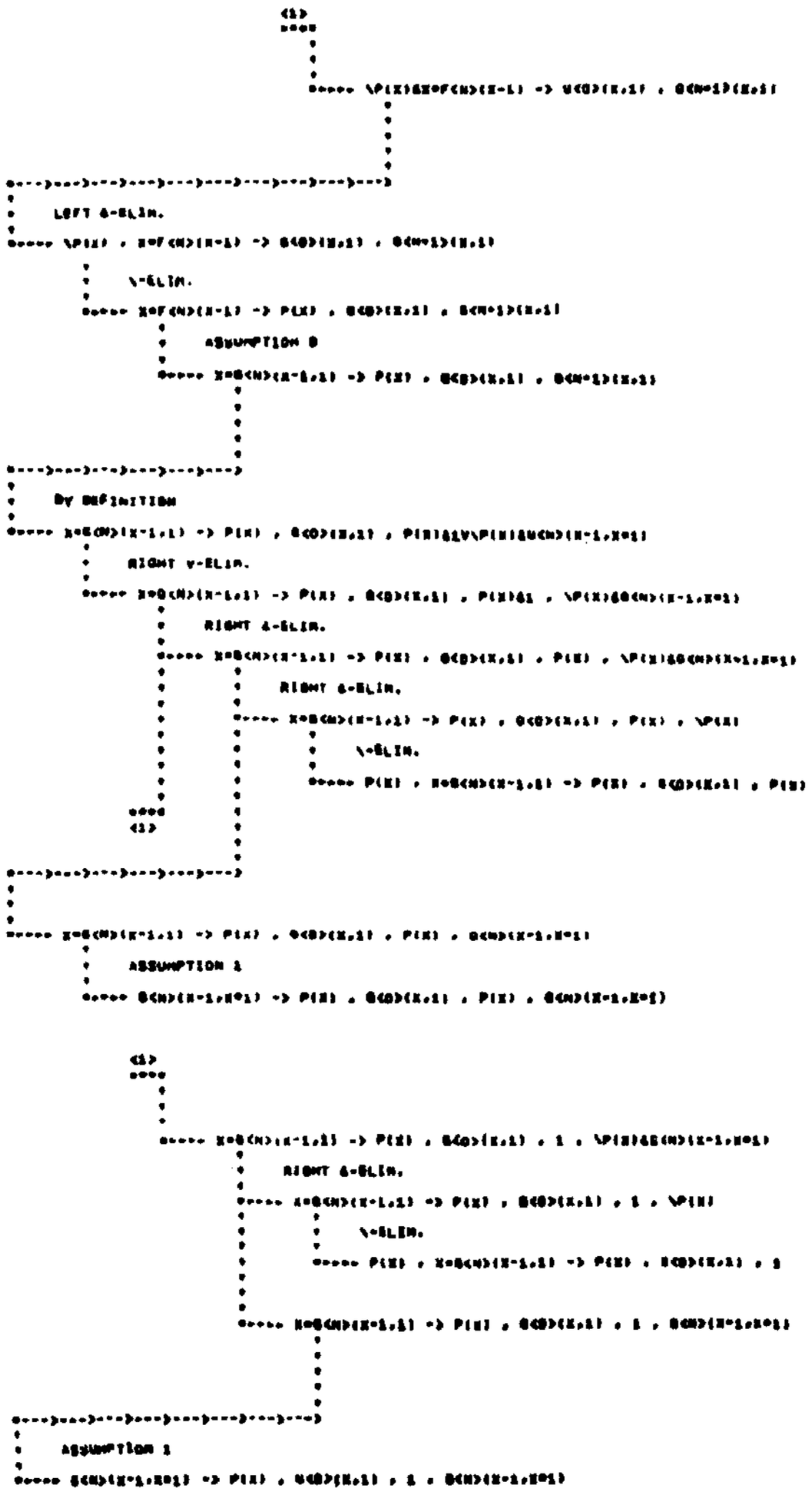
Equivalence of two factorial functions

Prove F(x) -> G(x,1)

Definitions F(x) = if x-0 then 1 else x * F(x~l)
G(x, y) = *if* x = O] then y else G(x-1, x * y)

Assumption x G(y, Z) = G(y, x - Z)





64