

Shigeki Goto

Musashino Electrical Communication Laboratory
Nippon Telegraph and Telephone Public Corporation
Musashino-shi Tokyo, 180 JAPAN

There have already been many program synthesizers which are based on classical logic. There is, however, some evidence that intuitionistic logic is more suitable for computer programs. This paper presents a program synthesis method based on intuitionistic logic. The synthesizing method is essentially an application of Gödel's interpretation. An experimental program synthesizer NJL, which performs Gödel's interpretation, is implemented in LISP. NJL takes natural deduction proofs as input and produces LISP programs.

1. Introduction

In this paper, programs on natural numbers will be treated mainly and the number theory based on intuitionistic logic is considered.

Program synthesis by theorem proving approach treats the logical formula which represents the specification of the desired program. Typical specification forms: (Manna and Waldinger [1])

$$\forall \bar{x} [\varphi(\bar{x}) \supset \exists \bar{z} \psi(\bar{x}, \bar{z})] \text{ ----- (1)}$$

Here, $\bar{x}$ is an input vector $\bar{x} = (x_1, x_2, \dots, x_m)$ and $\bar{z}$ is an output vector $\bar{z} = (z_1, z_2, \dots, z_n)$, $\varphi(\bar{x})$ is an input predicate and $\psi(\bar{x}, \bar{z})$ is an output predicate. The aim of program synthesis is to construct a function F such that $\bar{z} = F(\bar{x})$.

The present method is based upon Gödel's interpretation, which transforms (1) into (2) below:

$$\exists \bar{z} \forall \bar{x} [\varphi(\bar{x}) \supset \psi(\bar{x}, \bar{z}(\bar{x}))] \text{ ----- (2)}$$

Gödel's interpretation assures that there exists a term $\bar{t}$ such that $\forall \bar{x} [\varphi(\bar{x}) \supset \psi(\bar{x}, \bar{t})]$ holds. The term $\bar{t}$ is determined through the proof of (1) and the synthesized program can be defined as $F = \lambda \bar{x}. \bar{t}$.

2. Logical framework

To handle logical formulas in an intuitionistic framework, a good choice is to adopt the natural deduction system, which was devised by G.Gentzen. It is mentioned in many logic textbooks (e.g. Troelstra[4]).

Before giving Gödel's theorem, several definitions are necessarily introduced, since Gödel's interpretation utilizes primitive recursive functionals of finite type.

Def.1 (Type)

1. Symbol 0 is a type.
2. If δ and τ are types, then $\delta \langle \tau \rangle$ is also a type.

Although the same symbol "0" is used, type symbol 0 should be distinguished from natural number 0.

Def.2 (Finite type functionals)

1. Every natural number is a type 0 functional.
2. If type δ functional and type τ functional are already defined, then a type $\delta \langle \tau \rangle$ functional is a certain map from the set of type τ functionals into the set of type δ functionals.

The set of finite type functionals is an extension of the natural number. It is necessary to extend intuitionistic number theory to treat finite type functionals.

Def.3 (System FT)

- 1) FT terms and primitive recursive functionals:
 - i) 0 (natural number) is a type 0 term.
 - ii) Each free variable of type δ is a type δ term.
 - iii) If f is a type 0 term and S is the successor function, then $S(f)$ is also a type 0 term.
 - iv) If f is a type δ term and x is a bound variable of type τ , then $\lambda x f$ is a type $\delta \langle \tau \rangle$ term.
 - v) If f is a type $\delta \langle \tau \rangle$ term and x is a type τ term, then $f(x)$ is a type δ term.
 - vi) If g is a type τ term and h is a type $\tau \langle 0 \rangle \langle \tau \rangle$ term, then $\rho[g, h]$ is a type $\tau \langle 0 \rangle$ term. $\rho[]$ means the primitive recursion.
 - vii) A term is called a primitive recursive functional, if and only if it contains no free variables.

2) FT axioms and rules:

- i) All axioms and rules of intuitionistic number theory, except what concern the quantifiers, are taken as FT axioms and rules.

ii) Additional axioms and rules are:

Axioms for λ (lambda)

$(\lambda x f(x))(t) = f(t)$,

$\lambda x f(x) = f$,

If f and g are the same, except bound variables, then $f = g$.

Axioms for ρ []

$\rho[g, h](0) = g$,

$\rho[g, h](S(n)) = h(n, \rho[g, h](n))$.

Axiom for ()

$f = g \wedge x = y \supset f(x) = g(y)$

Substitution rule

Any term can be substituted for a free variable of the same type.

FT has no quantifier symbols. To treat quantifiers, the system QFT is formulated.

Def.4 (QFT)

QFT is a quantificational extension of FT.

1) Each formula in FT is also a formula in QFT.

2) If $A(a)$ is a QFT formula and " a " is a free variable, then $\forall x A(x)$ and $\exists x A(x)$ are also QFT formulas, where x is a bound variable of the same type as " a " and x never occurs in $A(a)$.

QFT axioms and rules are not explicitly defined here, since they are not used in this paper.

To each formula of intuitionistic number theory, Gödel's interpretation associates a formula of the form $\exists x \forall y A(x, y)$, which is called $\exists\forall$ -formula in QFT.

Def.5 ($\exists\forall$ -formula)

To each formula A in intuitionistic number theory, a $\exists\forall$ -formula A^* in QFT is associated. The definition is by induction on the logical complexity of A .

0) If A is a prime formula, then A^* is A .
From (1) to (4), it is assumed that B and C are subformulas of A and that B^* and C^* are already defined. B^* is $\exists x \forall y B'(x, y)$ and C^* is $\exists u \forall v C'(u, v)$.

1) If A is $\neg B$, then A^* is $\exists z \forall x \neg B'(x, z(x))$.

2) If A is $B \wedge C$, then A^* is $\exists x \exists u \forall y \forall v (B'(x, y) \wedge C'(u, v))$.

3) If A is $B \vee C$, then A^* is $\exists d \exists x \exists u \forall y \forall v [(d = 0 \wedge B'(x, y)) \vee (d \neq 0 \wedge C'(u, v))]$.

4) If A is $B \supset C$, then A^* is $\exists w \exists z \forall x \forall v [B'(x, z(x, v)) \supset C'(w(x, v))]$.

For (5) and (6), it is assumed that $F^*(a)$ is $\exists x \forall y F'(a, x, y)$.

5) If A is $\forall u F(u)$, then A^* is $\exists z \forall u \forall y F'(u, z(u), y)$.

6) If A is $\exists u F(u)$, then A^* is $\exists u \exists x \forall y F'(u, x, y)$.

On the interpretation, the important theorem holds.

Theorem (Gödel)

If a formula $A(\bar{z})$, containing at most $\bar{z}$ free, is provable in intuitionistic number theory and $\exists\forall$ -formula $A^*(\bar{z})$ is $\exists \bar{x} \forall \bar{y} A'(\bar{x}, \bar{y}, \bar{z})$, then there exists a vector $\bar{t}$ of primitive

recursive functionals in FT, such that $A'(\bar{t}(\bar{z}, \bar{y}), \bar{y}, \bar{z})$ is provable in FT.

The proof is so lengthy that it cannot be shown here (Troelstra[4]). Roughly speaking, the theorem is proved by interpreting axioms and rules of intuitionistic number theory into FT axioms and rules. For example, the induction rule (IND) in intuitionistic number theory can be interpreted in FT as follows:

	$[A(a)]$	a is an eigen variable.
IND	$\frac{A(0) \quad A(Sa)}{A(t)}$	S is the successor function. t is any term.

It is not expedient to apply Gödel's interpretation directly to a natural deduction, since it is not easy to handle the assumption dependency. The present system converts the original natural deduction proof into the LJ-like sequential form. The conversion technique is described in Gentzen[5]. For simplicity, it is assumed that neither $A(0)$ nor $A(Sa)$ depends on other assumptions.

$\frac{\rightarrow A(0) \quad A(a) \rightarrow A(Sa)}{\rightarrow A(t)}$
--

If $A(a)$ is $\exists z \forall w B(z, w, a)$, then the corresponding induction rule in FT is

$\frac{B(t0, w, 0) \quad B(z1, t1(a, z1, w2), a) \rightarrow B(t2(a, z1), w2, Sa)}{B(\rho[t0, t2](t), w, t)}$

Other axioms and rules are interpreted similarly and Gödel's theorem can be proved by induction on the complexity of $A(\bar{z})$ proof.

3. Synthesizer NJL

This section describes synthesizer NJL, which is a PDP-11 program written in LIPQ[3]. NJL is named after NJ (intuitionistic natural deduction) and Lisp. The rest of this paper illustrates an NJL synthesis example. Gödel's theorem is applied to program synthesis as follows:

(1) Make a logical formula expressing the specification of the desired program. Here, the "quotient remainder theorem" is taken for an example:

$\forall x1 \forall x2 [x2 \neq 0 \supset \exists z1 \exists z2 (x1 = x2.z1 + z2 \wedge z2 < x2)]$, where $x1$ is a dividend, $x2$ is a divider, $z1$ is the quotient and $z2$ is the remainder.

(2) Prove the formula of (1) in the number theory. A brief sketch of the proof is

	$[QR(a, b)]$	
IND	$\frac{QR(0, b) \quad QR(Sa, b)}{QR(t, b)}$	t is any term.
	$\forall x1 \forall x2. QR(x1, x2)$	

where $Q(a,b)$ stands for $\exists z1 \exists z2 (a = b.z1 + z2 \wedge z2(b))$. NJL is equipped with proof checking capability.

(3) Convert the proof of (2) into the sequential form. This step is automatically performed in NJL system.

(4) Interpret the proof of (2) into FT. First, NJL makes the corresponding $\exists V$ -formulas to the sequential forms of (3). For instance, formula of (1) is associated with a $\exists V$ -formula

```
((E (Z1 X1 X2) (Z2 X1 X2)) ((X1 X2)
(IMP NIL (IMP (NE X2 0)
(AND (EQ X1 (+ (* X2 (Z1 X1 X2))
(Z2 X1 X2) ))
(LT (Z2 X1 X2) X2) )))))
```

In NJL, formulas are expressed in prefix form, i.e. $((E X) (A X))$ for $\exists x A(x)$, $((X) (A X))$ for $\forall x A(x)$ and $(IMP A B)$ for $A \supset B$. The outer $(IMP NIL A)$ means a sequent $\rightarrow A$.

Next, the desired terms are determined by applying Gödel's theorem. The determination process is really a proof of Gödel's theorem for the formula of (1). NJL represents primitive recursive functionals by LISP programs.

Def.6 (Representation in LISP)

- 1) The natural number 0 is represented by the integer 0 in LISP.
 - 2) Each free variable of type 6 is represented by a variable in LISP program.
 - 3) If f is a type 0 term and represented by f' in LISP, then $S(f)$ is represented by $(ADD1 f')$ in LISP1.6 or $(+ f')$ in LIPQ.
 - 4) If f is a type 6 term, represented by f' in LISP, and x is a type 1 term, represented by x' in LISP, then $\lambda x f$ is represented by $(LAMBDA (x') f')$ in LISP.
 - 5) If f is a type 6(1) term represented by f' and x is a type 1 term represented by x' , then $f(x)$ is represented by $(f' x')$ in LISP.
 - 6) If g is a type 1 term represented by g' and h is a type 1(0)(1) term represented by h' , then $P[g,h]$ is represented by
- ```
(LABEL R001
(LAMBDA (X)
(COND ((ZEROP X) g')
(T (h' (SUB1 X) (R001 (SUB1 X))))))
or
(DE R001 (X)
(COND ((ZEROP X) g')
(T (h' (SUB1 X) (R001 (SUB1 X))))))
```
- 7) Every term which contains no free variables, i.e. primitive recursive functional, is represented by a LISP program.

In this example, Z1 and Z2 below are the synthesized programs.

```
(DE Z1 (X1 X2)
(COND ((ZEROP X1) 0)
(T (COND ((LT (S (Z2 (- X1) X2)) X2)
(Z1 (- X1) X2))
(T (S (Z1 (- X1) X2)))))))
```

```
(DE Z2 (X1 X2)
(COND ((ZEROP X1) 0)
(T (COND ((LT (S (Z2 (- X1) X2)) X2)
(S (Z2 (- X1) X2)))
(T 0)))))
```

This ends the example.

It is well-known that induction rule corresponds to recursive program (Manna and Waldinger [1]).

#### 4. Concluding remarks

Since Gödel's interpretation utilizes the primitive recursive functional, which is an extension of the primitive recursive function on natural numbers, the synthesized program becomes necessarily total (Sato[6]).

NJL can be applied to LISP linear lists. In that case, an induction schema LS2 (McCarthy[7]) is used instead of IND.

#### ACKNOWLEDGEMENTS

The author wishes to express his sincere thanks to Professor M.Sato at the University of Tokyo for his kind directions and advice. He also gratefully thanks Dr. N.Ikeno and LIPQ group of Musashino Electrical Communication Laboratory for their guidance and encouragement.

#### REFERENCES

- [1] Z.Manna and R.J.Waldinger, Toward automatic program synthesis, Comm. ACM, vol.14, no.3, pp.151-165, 1971.
- [2] S.Goto, Program synthesis through Gödel's interpretation, Proceedings of The International Conference on Mathematical Studies of Information Processing, 1978.
- [3] I.Takeuchi and H.Okuno, A list processor LIPQ, 2nd USA-JAPAN Computer Conference Proceedings, pp.416-421, August, 1975.
- [4] A.S.Troelstra, Metamathematical investigation of intuitionistic arithmetic and analysis, Lecture Notes in Math. 344, Springer, 1973.
- [5] M.E.Szabo (ed), The collected papers of Gerhard Gentzen, North-Holland, 1969.
- [6] M.Sato, Towards a Mathematical Theory of Program Synthesis, IJCAI-79.
- [7] J.McCarthy, Representation of Recursive programs in First order logic, The International Conference on Mathematical Studies of Information Processing, 1978.