Gödel Translations

Gödel showed that it is possible to translate classical number theory, say Peano Arithmetic (PA), into intuitionistic number theory, say Heyting Arithmetic (HA), thereby establishing that if HA is consistent, so is PA. The two theories differ only in that PA obeys the law of excluded middle, i.e.

$$PA = HA + (P \vee \neg P)$$

Gödel Translations continued

For instance, the translation of (P v ¬P) in HA is just ¬(P & ¬P), which is the same as

¬¬ (P v ¬P).

In general, Gödel translates formula F of PA into ¬¬F and shows

(PA |- F) implies (HA |- ¬¬F)

Gödel's Theorem for Computing

Gödel's famous Incompleteness Theorem can be proved very easily for an extension of PA to include partial recursive functions, say Scott Arithmetic (SA) based on LCF over the type N. Let a constructive version be CSA.

We take the basic type to be Ñ, those computable terms which have a natural number value iff they converge.

Computability in CSA

Here is how to define general recursive functions. Consider the 3x+1 function with natural number inputs.

f(x) = if x=0 then 1

else if even(x) then f(x/2)

else f(3x+1)

fi

fi

Using Lambda Notation

f = λ(x. if x=0 then 1
else if even(x) then f(x/2)
else f(3x+1))

Here is a related term with function input f

λ(f. λ(x. if x=0 then 1
else if even(x) then f(x/2)
else f(3x+1)))

The recursive function is computed using this term.

Defining General Recursive Functions

```
fix(λ(f. λ(x. if x=0 then 1
              else if even(x) then f(x/2)
                   else f(3x+1)
                   fi
              fi)))
```

Recursion in General

f(x) = F(f,x) is a recursive definition, also

f = λ(x.F(f,x)) is another expression of it, and the CTT definition is:

$$\text{fix}(\lambda(f.\ \lambda(x.\ F(f,x))))$$

which reduces in one step to:

$$\lambda(x.F(\text{fix}(\lambda(f.\ \lambda(x.\ F(f,x)))),x))$$

by substituting the fix term for f in λ(x.F(f,x)) .

Non-terminating Computations

CTT defines all general recursive functions, hence non-terminating ones such as this

$$\mathrm{fix}(\lambda(x.x))$$

which in one reduction step reduces to itself!

This system of computation is a simple functional programming language.

Unsolvable Problems

Suppose there is a function h that decides halting. Define the following element of $\tilde{N}$:

$$d = fix(\lambda(x.\ \text{if } h(x) \text{ then } \uparrow \text{ else } 0 \text{ fi}))$$

where $\uparrow$ is a diverging term, say $fix(\lambda(x.x))$.

Now we ask for the value of h(d) and find a contradiction as follows:

Generalized Halting Problem

Suppose that h(d) = *true,* then according to h, d converges, but according to its definition, the result is the diverging term ↑ because by computing the fix term for one step, we reduce

d = fix(λ(x. if h(x) then ↑ else 0 fi))

to d = if h(d) then ↑ else 0 fi .

If h(d) = *false,* then d converges to 0.

Incompleteness

We can add the predicate Conv(n) for any n in Ñ, asserting that the element n converges.

Suppose we could prove in CSA the following Convergence Theorem (CT).

CT: All n:Ñ. [Conv(n) v ¬Conv(n)].

Then we could extract a computable function

h: Ñ --> Bool. All n:Ñ. (h(n)=*true* iff Conv(n)).

Incompleteness

Thus, we cannot prove CT in CSA (because it is not true), indeed, we just proved ¬CT.

But then we cannot prove ¬¬CT in CSA if CSA is consistent. And if CSA is consistent, so is SA.

Hence we cannot prove CT in SA even though it is true because by Gödel's result we could then prove ¬¬CT. So there is a true sentence of SA which is not provable if CSA is consistent.