1 Introduction

1.1 Discussion

Last time, in Lecture 20, we defined the finitely axiomatizable theory of "simple arithmetic" called $\mathcal{Q}$ for the "classical" version and $i\mathcal{Q}$ for the intuitionistic one. In this lecture we will sketch the proof that any extension T of these theories as well as $\mathcal{Q}$ and $i\mathcal{Q}$ themselves, have the property that we cannot decide whether formulas are true (realizable) with respect to the theory axioms. This result applies to HA and PA as extensions of $\mathcal{Q}$.

Moreover, we can show that there is no algorithm $Decide_T$ which given any formula G of FOL decides whether it is valid (true in every FOL model) even though we assume that every specific formula G is either valid or not, e.g. even if we reason classically. This result is *Church's Theorem*.

1.2 Lecture Plan and Reading

We will provide reading from Kleene that proves *Gödel's 1st incompleteness theorem* based on the representation theorem we sketch. Here are the steps we take to establish these results.

- Define the *primitive recursive functions* and the *general recursive functions*, and show they are all *representable in* $i\mathcal{Q}$.
- Define *Gödel numberings*.
- Prove Gödel's *Diagonal Lemma*.
- Prove undecidability theorems.
- State Gödel's incompleteness theorems.

2 Recursive Functions

2.1 Previous examples

In lecture 14 we proved using Nuprl this theorem:

$$\forall x : \mathbb{N}.\ \exists r : \mathbb{N}.\ r^2 \leq x < (r+1)^2$$

Nuprl produced a recursive function $sqrt(x)$ to compute the value r. The function is an example of *primitive recursion* whose general form over $\mathbb{N}$ is:

$$\begin{aligned} f(0, y) &= g_1(y) \\ f(n+1, y) &= g_2\big(n, f(n, y), y\big) \end{aligned}$$

The $sqrt\, i$ function from Nuprl is:

$$\begin{array}{l} \textit{let rec } sqrt\, i = \\ \quad \textit{if } i = 0 \textit{ then } 0 \\ \quad \textit{else let } r = sqrt(i-1) \\ \qquad \textit{in if } (r+1)^2 \le n \textit{ then } r+1 \\ \qquad\quad \textit{else } r \end{array}$$

The primitive recursive form is:

$$\begin{aligned} root(0) &= 0 \\ root(n+1) &= g_2\big(n, root(n)\big) \\ \text{where } g_2(x, y) &= \textit{if } (y+1)^2 \le (x+1) \textit{ then } (y+1) \\ &\quad\ \textit{else } y \end{aligned}$$

What is useful about this form is that for every primitive recursive function f we can prove by standard induction

$$\forall x : \mathbb{N}.\, \forall y : \mathbb{N}^k.\, \exists v : \mathbb{N}.\big(f(x, y) = v\big).$$

Let's consider a "while rule" proof that there is an integer square root. We use the Hoare while rule for *partial correctness*, that is, we don't prove termination of the loop at this point.

$r : \mathbb{N}, n : \mathbb{N}$	are declarations
$r := 0$	assign r an initial value
$\{r^2 \le n\}$	assertion true by Arith
while $(r+1)^2 \le n$ ***do***	
$\quad \{r^2 \le n\}$	loop invariant
$\quad r := r + 1$	
$\quad \{r^2 \le n\}$	by assignment rule
od	
$\{r^2 \le n\}$	an invariant
$\{n < (r+1)^2\}$	loop exit condition
$\{r^2 \le n < (r+1)^2\}$	combine assertions

We know that the loop terminates because the distance between n and r decreases on each iteration. What is the recursive function equivalent to this?

$$\begin{array}{ll} root(r,n) = & \textit{if } (r+1)^2 \leq n \textit{ then } root(r+1,n) \\ & \textit{else } r \\ root(0,n) & \text{returns the integer square root.} \end{array}$$

Theorem: $\forall n : \mathbb{N}. \forall r : \mathbb{N}. \big(r < n \Rightarrow Root\big(root(r,n), n\big)\big)$

The while rule in general for partial correctness is

$\{$Invariant$\}$

while $b\,exp$ ***do***

$\{$Invariant$\}$

body

$\{$Invariant$\}$

od

$\{$Invariant & $\neg\, b\,exp\}$

We can prove the iterative version using the *Least Number Principle* (LNP)

$$\exists x : \mathbb{N}.\, P(x) \Rightarrow \exists y : \mathbb{N}. \big(P(y)\,\&\,\forall z : \mathbb{N}. \big(z < y \Rightarrow \neg\, P(z)\big)\big)$$

$P(x) = R(x,n) = n < (x+1)^2$
$\exists x : \mathbb{N}.\, P(x)$ take $x = n$
$\exists y : \mathbb{N}.\, n < (y+1)^2 \quad \forall z : \mathbb{N}. \big(z < y \Rightarrow \neg\, R(z,n)\big)$
$\neg\, R(z,n) = \neg\, \big(n < (z+1)^2\big)$
$\forall z : \mathbb{N}. \big(z < y \supset \neg\, \big(n < (z+1)^2\big)\big)$
$\forall z : \mathbb{N}. \big(z < y \supset (z+1)^2 \leq n\big)$

Thus for $z = y - 1$ we have $\big((y-1)+1\big)^2 \leq n$ so $y^2 \leq n$.

We can "prove" the LNP using the Hoare while rule for partial correctness as follows.

Program/Proof

$$
\begin{array}{l}
\textit{if } P(0,n) \textit{ then return } 0 \\
\textit{else} \\
\{\neg\, P(0,n)\} \\
r := 0 \ \big\{\forall z : \mathbb{N}.\big(z < r+1 \Rightarrow \neg\, P(z,n)\big)\big\} \\
\textbf{\textit{while}} \ \neg\, P(r+1,n) \ \textbf{\textit{do}} \\
\quad \big\{\neg\, P(r,n)\big\} \\
\quad \big\{\forall z : \mathbb{N}.\big(z < r+1 \Rightarrow \neg\, P(z,n)\big)\big\} \\
\quad r := r+1 \\
\quad \big\{\neg\, P(r,n)\big\} \\
\quad \big\{\forall z : \mathbb{N}.\big(z < r+1 \Rightarrow \neg\, P(z,n)\big)\big\} \text{ by Lemma} \\
\textbf{\textit{od}} \\
\big\{P(r+1,n)\big\} \ \text{thus} \ \exists y : \mathbb{N}.\, P(y,n) \ \text{ take } \ y = r+1 \\
\big\{\forall z : \mathbb{N}.\big(z < r+1 \Rightarrow \neg\, P(z,n)\big)\big\} \\
\big\{\forall z : \mathbb{N}.\big(z < y \Rightarrow \neg\, P(z,n)\big)\big\}
\end{array}
$$

Note, we assume that $P(r,n)$ is *decidable*, we can compute whether $P(r,n)$ or $\neg\, P(r,n)$.

In some cases we can implement recursion using *iteration* on a finite set of *state variables* like r. This kind of evaluation is called *tail recursive* (it avoids using a stack implementation for those who know about compilers).

Compare the normal recursive evaluation of factorial defined by

$$
\begin{array}{l}
\mathit{fact}(0) = 1 \\
\mathit{fact}(n+1) = \mathit{fact}(n) \cdot (n+1)
\end{array}
$$

and this version

$$
\begin{array}{l}
a := 1 \\
m := 1 \\
\{a = m!\} \\
\textbf{\textit{while}} \ m \leq n \ \textbf{\textit{do}} \\
\quad \{m \leq n\} \\
\quad a := a * m \\
\quad m := m+1 \\
\quad \big\{a = (m-1)!\big\} \\
\textbf{\textit{od}} \\
\{n = m+1\} \\
\big\{a = (m-1)!\big\} \\
\{a = n!\}
\end{array}
$$

The while loop is recursive

$$wh(n,m,a) = \quad \textit{if} \;\; m \leq n \;\; \textit{then} \;\; wh(n,m+1,a*m) \quad \textit{else} \; a$$

$fact(n) = wh(n,1,1)$.

2.2 Primitive Recursion

Primitive recursive functions.

Let $s(x) = x+1$ call this $a_0(x,y)$

$add(0,y) = y$
$add(x+1,y) = s\big(add(x,y)\big)$ call this $a_1(x,y)$

$mult(0,y) = 0$
$mult(x+1,y) = add\big(mult(x,y),y\big)$ call this $a_2(x,y)$

$exp(0,y) = 1$ (this $y^0 = 1$)
$exp(x+1,y) = mult\big(exp(x,y),y\big)$ $(y^{x+1} = y^x * y)$ call this $a_3(x,y)$

$hyp(0,y) = y$
$hyp(x+1,y) = exp\big(hyp(x,y),y\big)$ $y^{y^{y^{\cdot^{\cdot^{\cdot^{y}}}}}} \Big\} x$ call this $a_4(x,y)$

$a_{m+1}(0,y) = y$
$a_{m+1}(x+1,y) = a_m\big(a_{m+1}(x,y),y\big)$
This gives a version of *Ackermann's function* $ack(m,x,y)$,
a non primitive recursive function.

Over lists primitive recursion is:

$$f(nil,y) = g_1(y)$$
$$f(h.t,y) = g_2\big(h,t,f(t,y),y\big)$$

The append function, $l_1 \mathbin{@} l_2$ is:

$$nil \mathbin{@} y = y$$
$$(h.t) \mathbin{@} y = h.(t \mathbin{@} y)$$

2.3 Reduction to minimization

Boolos and Jeffrey prove that we can use the least number operator, μ, to replace primitive recursion. This is equivalent to showing that while-programs over $\mathbb{N}$ can compute all computable functions. We assume this is known from basic computer science courses. If not, the result is covered well in Boolos and Jeffrey posted as supplemental material.

Theorem: A function is *general recursive* iff it is Min-recursive, e.g. defined by $+, *, f_=, id_i^n$, composition, minimization of regular functions.

2.4 Representation of Min-recursive functions

- $x + y$ is rep by $x + y = z$
- $x * y$ is rep by $x * y = z$
- $eq(x, y)$ is rep by $(x = y \;\&\; z = t) \vee (x \neq y \;\&\; z = f)$
- If f is rep by F
  and g_i is rep by G_i
  then $\lambda\, x_1, \ldots, x_n.\, f\big(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)\big)$ is
  rep by $\exists y_1, \ldots, y_m.\, G_1(x_1, \ldots, x_n, y_1) \;\&\; \cdots \;\&\; G_m(x_1, \ldots, x_n, y_m) \;\&\; F(y_1, \ldots, y_m, z)$
- If f is rep by $F(x, i, z)$ then $g(x) = \mu i.\, f(x, i) = 0$ is
  rep by $F(x, y, 0) \;\&\; \forall w.\big(w < y \supset \sim F(x, w, 0)\big)$.

Theorem: Every general recursive function is representable in $i\mathcal{Q}$.

See Boolos & Jeffrey posted reading.

3 Gödel numberings

We want to reason about formulas (and proofs) in a very precise formal theory, the *meta theory*. Gödel chose to use HA or PA for this purpose so that there would be no doubts about the legitimacy of his results. So he defined a *translation* of PA into $\mathbb{N}$. We now know that it is only necessary to translate $\mathcal{Q}$ into $\mathbb{N}$. This is delicate but possible. The details are quite tedious and ad hoc, but once this is done, we can mostly ignore those details. We use the precise account given by Boolos & Jeffrey pages 170-171. See the supplemental reading for this lecture.

The key result we need is a function called $diag$ which given a formula A of $\mathcal{Q}$ and its Gödel number, denoted $\ulcorner A \urcorner$, provides the Gödel number of the formula

$$* \quad \exists x.\big(x = \ulcorner A \urcorner \;\&\; A(x)\big)$$

where A has only the variable x free.

We can use the representation result to show that there is a computable function $diag$ such that if $n = \ulcorner A(x) \urcorner$, then $diag(n)$ is the Gödel number of $*$ above.

Boolos & Jeffrey give $diag$ explicitly as

$diag(n) = 4515788 \mathbin{\widehat{*}} \big(num(n)\big) \mathbin{\widehat{*}} \big(3 \mathbin{\widehat{*}} (n \mathbin{\widehat{*}} 2)\big)$
where this $\widehat{*}$ operator (they use only $*$) is
defined as $m \mathbin{\widehat{*}} n = m * 10^{lh(m)} + n$ where
$lh(n) = \mu x(0 < x \;\&\; n < 10^x)$, the least positive number x where
10^x bounds n (for positive x).

For details see supplemental notes on Gödel numberings.

4 The Diagonal Lemma in $\mathcal{Q}$

Let $Diag(x, y)$ represent the $diag$ function, then

$$Diag(\overline{n}, \overline{k}) \text{ iff } diag(n) = k$$

Let F be this formula: $F(x) = \big(\exists y.\, Diag(x, y) \;\&\; B(y)\big)$ and
let $f = \ulcorner F \urcorner$ and $G = \exists x.\big(x = \overline{f} \;\&\; F(x)\big)$.

Note that by pure logic:
$G \Leftrightarrow \exists y.\big(Diag(\overline{f}, y) \;\&\; B(y)\big)$

Let $g = \ulcorner G \urcorner$ hence $diag(f) = g$ and hence $Diag(\overline{f}, \overline{g})$.

Therefore
$G \Leftrightarrow \exists y.\big(y = \overline{g} \;\&\; B(y)\big)$
so
$G \Leftrightarrow B(\overline{g})$, that is
$G \Leftrightarrow B(\ulcorner G \urcorner)$.

We have just proved:

Diagonalization Lemma — Let T be a theory in which $diag$ is representable by $Diag$ and let $B(y)$ be any formula of T with only y free, then we can find a sentence G of T such that $\models_T G \Leftrightarrow B(\ulcorner G \urcorner)$.