CS 5860 Programmable Specifications

Thur Oct 6, 2011

PLAN

1. Programmes to specifications
2. Formalizing arithmetics – issues
3. Primitive Recursive Arithmetic (PRA)
4. Expressing arithmetic in First-Order Logic
   Equality, Zero, Successor, Addition, Multiplication

1. Programmes to specifications

Last time we mentioned that specifying a programming task precisely and writing code to accomplish the task is a key formal method. This is precisely what we have been practicing using Fol as a dependently typed programming language.

We also saw last time that some specifications, such as $(\sim \forall x.\, B(x)) \Rightarrow \exists y. \sim B(y)$ is not a solvable task. We say that the specification is not solvable or not programmable. We changed it to a solvable task by adding the hypotheses

$d: \forall x.(B(x) \vee \sim B(x)$ and

$ed: (\exists y. \sim B(y)) \vee \sim (\exists y. \sim B(y))$

Think about these two tasks. Are they programmable? If not, can we add assumptions to change that?

① $\sim \forall x \exists y.\, R(x,y) \Rightarrow \exists x \forall y \sim R(x,y)$

② $\sim \exists x \forall y.\, R(x,y) \Rightarrow \forall x \exists y. \sim R(x,y).$

CS 5860

Thur Oct 6, 2011

1 Programming to "specs" continued.

Some mathematicians such as E. Bishop think that the general form of most interesting mathematics problems is showing $\exists y \forall x. A(x,y)$. Thus problems of form ① below are especially interesting

$$① \quad \exists y \forall x\, A(x,y) \Rightarrow \exists v \forall u.\, B(u,v)$$

Bishop claims that problems of type ① are equivalent to those of type ② below

$$② \quad \forall y \exists v \forall u\, (\forall x.\, A(x,y) \Rightarrow B(u,v)).$$

He claims ① $\Longleftrightarrow$ ②

We can see that ② $\Rightarrow$ ① is programmable. Here are hints about the "universal math problem."

UMP Theorem $\forall y \exists v \forall u\, (\forall x\, A(x,y) \Rightarrow B(u,v)) \Rightarrow$
$(\exists y \forall x.\, A(x,y) \Rightarrow \exists v \forall u. B(u,v))$

$h: \forall y \exists v \forall u.\, (\forall x.\, A(x,y) \Rightarrow B(u,v)),\ e: \exists y \forall x.\, A(x,y)$
$\vdash \exists v. \forall u.\, B(u,v)$

Can you finish the proof and discover the method of using form ② to solve the form ①?

I'll give the program after Fall Break.

CS 5860 — Formalizing Arithmetic in FOL

Thur Oct 6, 2011 — (Peano and Hertas Arithmetic)

2. Formalization issues.

We will want to express ideas such as

"a number is zero" say $Z(x)$

"numbers x and y are equal" say $Eq(x,y)$

"z is the sum of x and y" say $Add(x,y,z)$

"z is the product of x and y" say $Mult(x,y,z)$

"x is less than y" say $Less(x,y)$

Can we express what we need to say using FOL?

We can imagine expressing that x is zero by a predicate $Z(x)$. We could say that "0 is unique" by writing

$$\forall x \forall y. \left(Z(x) \,\&\, Z(y) \Rightarrow Eq(x,y) \right)$$

We know the properties of Equality and can state them as axioms.

1. Reflexive $\forall x.\ Eq(x,x)$
2. Transitive $\forall x,y,z.\ (Eq(x,y) \,\&\, Eq(y,z)) \Rightarrow Eq(x,z)$
3. Symmetric $\forall x,y.\ (Eq(x,y) \Rightarrow Eq(y,x))$

Let's now examine how we define Add and Mult, first informally and then in FOL.

CS 5860
Thur Oct 6, 2011 Primitive Recursive Arithmetic

Examples of primitive recursion

$a_0(x,y) = x+1$ $\quad s(x) = x+1$ $\quad$ so $\quad a_0(x,y) = s(x)$

$a_1(x,y) = add(x,y)$

$$\begin{cases} add(0,y) = y \\ add(s(x),y) = s(add(x,y)) \end{cases}$$

note $\quad s(add(x,y)) = a_0(a_1(x,y),y)$

$a_2(x,y) = mult(x,y)$

$$\begin{cases} mult(0,y) = 0 \\ mult(s(x),y) = add(mult(x,y),y) \end{cases}$$

i.e. $(x+1)\cdot y = x\cdot y + y$

note $\quad add(mult(x,y),y) = a_1(a_2(x,y),y)$

$a_3(x,y) = exp(x,y)$

$$\begin{cases} exp(0,y) = 1 \quad \text{i.e. } y^0 = 1 \\ exp(s(x),y) = mult(exp(x,y),y) \end{cases}$$

$y^{x+1} = y^x \cdot y$

note $\quad mult(exp(x,y),y) = a_2(a_3(x,y),y)$

$a_4(x,y) = hyperexp(x,y)$

$$\begin{cases} hyperexp(0,y) = y \\ hyperexp(s(x),y) = exp(hyperexp(x,y),y) \end{cases}$$

note $\quad exp(hyperexp(x,y),y) = a_3(a_4(x,y),y)$

$a_{n+1}(x,y)$

$$\begin{cases} a_{n+1}(0,y) = y \\ a_{n+1}(s(x),y) = a_n(a_{n+1}(x,y),y) \end{cases}$$

Can think of $a_n(x,y)$ as a function of n,x,y.
This is Ackerman's function in one form.
It is not primitive recursive.

Peano Arithmetic

Most axiomatizations of arithmetic are based on the Peano axioms. These axioms characterize the natural numbers together with the operations + and *. If we include the axioms of equality, then Peano Arithmetic can be defined as

$$\text{Peano Arithmetic} \equiv \mathcal{L}(=,+,*,0,1;\ \text{ref, sym, trans, subst,}\ \text{not-surjective, injective, induction,}\ \text{functionality}_+, \text{add-base, add-step,}\ \text{functionality}_*, \text{mul-base, mul-step})$$

where the axioms are as follows

Equality Axioms

ref:	$(\forall x)\ x=x$	
sym:	$(\forall x,y)\ (x=y \supset y=x)$	
trans:	$(\forall x,y,z)\ ((x=y \wedge y=z) \supset x=z)$	
subst:	$(\forall x,y)\ (x=y \supset P(.,x,.) \supset P(.,y,.))$	*for every* P

Successor Axioms

non-surjective	$(\forall x) \sim(x+1 = 0)$	
injective	$(\forall x,y)\ (x+1=y+1 \supset x=y)$	
induction	$(P(0) \wedge (\forall x)(P(x) \supset P(x+1))) \supset (\forall x)P(x)$	*for every* P

Addition Axioms

add-base	$(\forall x)\ (x+0 = x)$	
add-step	$(\forall x,y)\ (x+(y+1) = (x+y)+1)$	

Multiplication Axioms

mul-base	$(\forall x)\ (x*0 = 0)$	
mul-step	$(\forall x,y)\ (x*(y+1) = (x*y)+x)$	

If we drop multiplication and its axioms, we get a very simple arithmetical theory called *Presburger Arithmetic*, which is quite expressive but still decidable.

Inductively Ordered Integral Domains satisfy the Peano Axioms and vice versa.