Applied Logic & Computer Assisted Theorem Proving. Introductory lecture

Erik Palmgren

August 31, 2010

1 Introduction

Methods of mathematical logic are becoming widely used in engineering of computer systems and programs to guarantee correctness of operation, and to facilitate construction and modification. Applications go far beyond the traditional use of boolean algebra or propositional logic in design of digital circuits.

In the course Applied Logic we will learn mathematical foundations of formal specification and methods for proving correctness of systems. We learn to formulate and solve problems using

- Modal logics: temporal logic, epistemic logic, computational tree logic
- Predicate logic, classical and intuitionistic
- Type theory (Martin-Löf type theory, Calculus of inductive constructions)

We lear to employ computer tools for such problems:

- Theorem provers
- Model checkers
- Proof support systems
- Program extractors

To build, and also to use, such tools reliably we need to understand the mathematics behind them to a larger or lesser extent. This branch of mathematics is of course mathematical logic. We will go into the necessary theoretical parts throughout the course.

The course Computer Assisted Theorem Proving will essentially be a subset of the Applied Logic course with regards to the lectures. Computer assisted theorem proving, that is to use computers to construct formal proofs of mathematical theorems is also becoming popular in pure mathematics. Particularly so for proofs that involve a large number of cases to check in combination with calculations. Two examples are

- The Four Colour Theorem. Georges Gonthier, *A computer-checked proof of the Four Colour Theorem.*

  `http://research.microsoft.com/en-us/um/people/gonthier/4colproof.pdf`

- Kepler's conjecture on sphere packing

  `http://sites.google.com/site/thalespitt/kepler-conjecture`

Another promising area seems to be verification of numerical methods.

The objective of CATP course will be to master, and understand the mathematical foundations of one proof assistant, namely the Coq system. This will include doing a small project or study in theorem proving in a suitable area.

2 The difficulty of theorem proving

The famous mathematician David Hilbert expressed around 1900 his credo (roughly):

> Every precise, wellformulated mathematical problem P can be decided. That is, we will be able to prove P or to prove that P is false.

Two wellknown examples of such problems in number theory are

(P_1) *Fermat's last theorem*: the equation

$$x^n + y^n = z^n$$

has no solution in integers x, y, z for integers $n > 2$ and $|x|, |y|, |z| > 1$.

(P_2) *Goldbach's conjecture:* every even integer $n > 3$ is the sum of two primes.

Fermat's last theorem was finally proved 1995 (1997) after about 350 years of efforts, using some of the most sophisticated techniques of mathematics. Goldbach's conjecture is still undecided.

Both problems, P_1 and P_2, may straightforwardly be formulated in Peano Arithmetic (PA) using first order logic. (Exercise: do this.) The language of Peano arithmetic uses only the symbols $L_{\mathrm{PA}} = \{+, \cdot, **, 0, S(.)\}$ plus logical symbols and $=$. Here $S(x)$ stands

for the operation of adding one to x, and $x ** y$ stands for the exponentiation operation (written x^y). Axioms for arithmetical operations are

$$\begin{array}{lll} x+0=x & x\cdot 0=0 & x^0=S(0) \\ x+S(y)=S(x+y) & x\cdot S(y)=x\cdot y+x & x^{S(y)}=x^y\cdot x \end{array}$$

Axioms expressing that we are dealing with the natural numbers $\mathbb{N}=\{0,1,2,3,\ldots\}$

$$S(x)\neq 0$$

$$(\forall x\in\mathbb{N})(\forall y\in\mathbb{N})(S(x)=S(y)\Rightarrow x=y).$$

The *induction principle*: For any property $P(x)$ (more precisely, given a first order formula over L_{PA}) where x ranges over $\mathbb{N}$

$$P(0)\wedge(\forall x\in\mathbb{N})[P(x)\Rightarrow P(S(x))]\Longrightarrow(\forall x\in\mathbb{N})P(x).$$

(Recall: $A\wedge B$ is read *A and B*. $(\forall x\in M)P(x)$ is read *for all x in the set M, $P(x)$ holds.*)

Remark 1. Though P_1 is true, it is not known whether it can be proved on the basis of the above axioms for Peano arithmetic using first order logic. However its almost certain that it can be proved on the basis of axiomatic set theory such as ZFC.

Remark 2. The exponentiation operation $x ** y = z$ can in fact be defined as a relation $R(x,y,z)$ from the other operations. This is rather involved to prove.

Gödel's Incompleteness Theorem (1931) *Suppose that T is a consistent first order axiom system containing PA. Then there is a closed first order formula Q (statement) expressed using L_{PA} so that neither Q nor $\neg Q$ can be proven in T.*

Gödel's result showed that Hilbert's naive belief from 1900 is wrong, if we fix a mathematical theory T which is sufficiently rich to contain basic arithmetic axioms.

Gödel's theorem is proven in standard, second courses in logic. The theorem builds on the fact that PA can encode all kinds of finite objects and processes. In particular it can encode the notion of formal proof of a theorem in PA. The system the can thus make statements about itself, and express a formula to the effect "I am not provable". The technical details are complicated and the proof is somewhat subtle. We will not go into it in this course.

Remark 3. To see that finite sets may be encoded in PA we can use the coding of Ackermann. Let $p_1,p_2,p_3,\ldots$ be an enumeration of the prime numbers in strictly increasing order. If m_k codes the finite set A_k, $k=1,\ldots,n$ then

$$p_1^{m_1}p_2^{m_2}\cdots p_k^{m_k}\text{ codes }\{A_1,\ldots,A_n\}.$$

Thus e.g. $\emptyset$ is coded by 1 and $\{\emptyset,\{\emptyset\}\}$ is coded by $2^1 3^{2^1}(=18)$.

Membership and equality have to be defined to ignore multiple copies and order of elements. (Some exercises will concern these problems.)

3 Encoding reasoning in formal logic and Coq

In mathematics we deal with statements or propositions that have definite truth-values, the statements are true or false (but not both). We make assertions or judgments about the propositions. These can be absolute (unconditional) like

$$1 + 1 = 2 \text{ is true}$$

or hypothetical (conditional)

$$\text{assuming } x^2 = 0 \text{ is true, the equality } x = 0 \text{ is true.}$$

The latter is abbreviated

$$x^2 = 0 \text{ true} \vdash x = 0 \text{ true.}$$

In general a hypothetical judgement (about) truth has the form

$$A_1 \text{ true}, A_2 \text{ true}, \ldots, A_n \text{ true} \vdash B \text{ true.}$$

Often "true" is dropped, when this is understood, so that hypothetical judgements read

$$A_1, A_2, \ldots, A_n \vdash B. \tag{1}$$

The list of assumptions $A_1, A_2, \ldots, A_n$ is sometimes called a *context.*

In formal logic or in the Coq system we represent propositions by *formulas*

informal language	formal logic	Coq (ASCII coded)
A and B	$A \wedge B$	A `/\` B
A or B	$A \vee B$	A `\/` B
A implies B	$A \Rightarrow B$	A `->` B
false	$\bot$	`False`
for all $x \in S$, A	$(\forall x \in S)A$	`forall` $x : S$`,` A
there is $x \in S$ s.t. A	$(\exists x \in S)A$	`exists` $x : S$`,` A

Predicate logic can easily be handled in Coq using the rich type structure. This type structure is a vast generalization of types that appear in mundane programming languages. Every wellformed term in Coq has a type.

```
0 : nat
1+1 : nat
```

The type `nat` is itself a term and has a type:

```
nat : Set
```

Logical expression have the type `Prop`, e.g.

```
A /\ B : Prop
```

`Prop` and `Set` are in turn types

```
Prop : Type
Set : Type
```

Predicate logic speaks of predicates and relations defined on sets.

```
D : Set
```

A one-place (unary) predicate on `D` is in Coq a function (a so-called propositional function)

```
P : D -> Prop
```

that to each $x : D$ assigns a proposition `P x`. The intended understanding is that `P x` holds if and only if `P` is true at `x`. In set theory a predicate P on D is usally identified with a subset of D, and thus $P(x)$ holds iff $x \in P$.

Binary relations are defined as two-place propositional functions

```
R: D -> D -> Prop
```

and `R x y` is true iff `x` related to `y` via `R`. We may also have relations and predicates between different sets

```
R : D -> E -> Prop
```

We refer to the Coq tutorial [2] for an introduction to doing predicate logic in Coq. For a review of natural deduction see [1].

References

[1] M.R.A. Huth and M.F. Ryan. *Logic in Computer Science: Modelling and reasoning about systems.* Second edition. Cambridge University Press 2004.

[2] G. Huet, G. Kahn and C. Paulin-Mohring. The Coq Proof Assistant - A Tutorial. URL: http://coq.inria.fr