Proofs

Arthur Ryman, arthur.ryman@gmail.com

October 20, 2018

Abstract

This article is a Z Notation specification for proofs and proof checking. It has been type checked by *f*UZZ. The definitions that appear here are taken from Lemmon's book, *Beginning Logic*. The purpose of this specification is to guide the development of a proof checker aimed at Z specifications

1 Introduction

For a long time I have thought that it would be extremely useful to be able to write formal proofs concerning the mathematical objects defined in Z specifications. There are some very mature proof assistants available. I know something about Coq, but unfortunately it's style of proof is very different that that one finds in mathematical papers. A Coq proof consists of a list of tactics that represent higher level aggregates of deductions. This makes sense for a proof assistant since its job is to help the user discover proofs.

While a proof assistant might make sense is some contexts, proper development of a mathematical paper consists of a gradual introduction of concepts and lemmas leading to the main results. The proofs should, in some sense, write themselves. The focus of a mathematical paper should be on explanation and clarity. The proofs should be easy to read. I'd therefore really like something that would let me write and check natural looking proofs.

In contrast to Coq, the style of proof presented by Lemmon is very clear and explicit. However, the task of checking such a proof could easily be delegated to a program, much the same way that *f*UZZ type checks Z.

I believe that the kernel of a proof checker could be very small. It is basically an engine driven by a set of deduction rules. The engine simply needs to check that each deduction rule gets applied correctly. Even if it turns out that writing such an engine is too much work, the exercise of developing at least a simple version should give me a greater appreciation of tools like Coq and enable me to use them more productively.

My plan of attack is to formalize the concept of proof as described by Lemmon, starting with the propositional calculus.

2 Propositions

The *propositional calculus* defines a set of formal *statements* or *propositions* without being concerned about the subject matter described by those statements. The only restriction on these statements is that, in any given context, they possess a *truth value* of either *true* or *false*.

2.1 *Prop*

A proposition is also referred to as a *well-formed formula* or *wff* for short. This terminology stems from the traditional development of the propositional calculus in terms of a language of sentences over an alphabet with rules that prescribe when a given sentence is *well-formed*. However, here we will dispense with the language viewpoint, and its associated parsing issues, and move directly to the end result of parsing, namely the creation of *abstract syntax trees* or *ASTs* for short.

Z notation has a convenient mechanism for specifying the structure of ASTs, namely that of *free types*. Let *Prop* denote the free type of all propositions of the propositional calculus. The free type definition of *Prop* is as follows.

$$
\begin{array}{lll}
PropVar & ::= & P \mid Q \mid R \mid S \mid T \\
& \mid & (\_') \langle\!\langle PropVar \rangle\!\rangle \\
Prop & ::= & true \mid false \\
& \mid & \mathrm{var} \langle\!\langle PropVar \rangle\!\rangle \\
& \mid & \neg \langle\!\langle Prop \rangle\!\rangle \\
& \mid & (\_ \land \_) \langle\!\langle Prop \times Prop \rangle\!\rangle \\
& \mid & (\_ \lor \_) \langle\!\langle Prop \times Prop \rangle\!\rangle \\
& \mid & (\_ \Rightarrow \_) \langle\!\langle Prop \times Prop \rangle\!\rangle \\
& \mid & (\_ \Leftrightarrow \_) \langle\!\langle Prop \times Prop \rangle\!\rangle
\end{array}
$$

Note that the definition of *Prop* depends on the definition of *PropVar*, the set of propositional variables. There are two *constant* propositions, namely *true* and *false*. Any propositional variable defines a proposition. Propositions are built up recursively from the constants and variables using *logical connectives*. Here I use logical connective symbols defined by Z rather than those defined by Lemmon.

2.2 *PropVar*

The separation of the form of a statement from any given subject matter is accomplished by the use of *propositional variables* that stand for arbitrary statements. Let *PropVar* denote the free type of all propositional variables.

Remark. *There are a countable infinity of propositional variables.*

$$PropVar \rightarrowtail\!\!\!\!\!\rightarrow \mathbb{N} \neq \varnothing$$

2.3 P \propP, Q \propQ, R \propR, S \propS, and T \propT

Traditionally, arbitrary statements are represented by single letters such as P, Q, R, S, and T.

Remark. *Each of P, Q, R, S, and T is a propositional variable.*

$$\{P, Q, R, S, T\} \subset \mathit{PropVar}$$

2.4 $'$ \propPrime

Typical propositions contain a small number of distinct statements, in which case the letters can be used. If more statements occur then the letters are decorated with one or more primes, e.g. P', Q'', etc.

Example. *P' and Q'' are propositional variables.*

$$P' \in \mathit{PropVar}$$
$$Q'' \in \mathit{PropVar}$$

Remark. *Appending a prime to a propositional variable is an injection from PropVar to PropVar.*

$$(\_') \in \mathit{PropVar} \rightarrowtail \mathit{PropVar}$$

2.5 *true* \trueProp and *false* \falseProp

Let *true* denote the proposition that is true in all contexts and let *false* denote the proposition that is false in all contexts. The propositional *true* and *false* are said to be *constant* because they do not depend on the context.

2.6 var \varProp

Let V be a propositional variable. Let $\mathrm{var}(V)$ denote the proposition defined by V.

Remark. *The sets PropVar and Prop are different types. The expression P is a propositional variable and the expression $\mathrm{var}(P)$ is a proposition.*

$$P \in \mathit{PropVar}$$
$$\mathrm{var}(P) \in \mathit{Prop}$$

2.7 $\neg$ \notProp

Let A be a proposition. Let $\neg A$ denote the *negation* of A.

Example. $\neg\,(\mathrm{var}\,P)$ *is a proposition.*

$$\neg\,(\mathrm{var}\,P) \in \mathit{Prop}$$

2.8 $\neg$ `\notPropV`

We can simplify the notation for negating a proposition defined by a propositional variable by defining a function that directly negates the propositional variable and produces a proposition.

$$
\begin{array}{|l}
\neg\ : \mathit{PropVar} \rightarrowtail \mathit{Prop} \\
\hline
\forall\, V : \mathit{PropVar} \bullet \\
\qquad \neg\ V = \neg\,(\mathrm{var}\ V)
\end{array}
$$

2.9 $\wedge$ `\andProp`

Let A and B be propositions. Let $A \wedge B$ denote the *conjunction* of A and B.

2.10 $\wedge$ `\andPropVP`, $\wedge$ `\andPropPV`, and $\wedge$ `\andPropVV`

We can simplify the notation for conjunctions involving propositions defined by propositional variables as follows.

$$
\begin{array}{|l}
\_ \wedge \_ : \mathit{PropVar} \times \mathit{Prop} \rightarrowtail \mathit{Prop} \\
\_ \wedge \_ : \mathit{Prop} \times \mathit{PropVar} \rightarrowtail \mathit{Prop} \\
\_ \wedge \_ : \mathit{PropVar} \times \mathit{PropVar} \rightarrowtail \mathit{Prop} \\
\hline
\forall\, V : \mathit{PropVar};\ A : \mathit{Prop} \bullet \\
\qquad V \wedge A = (\mathrm{var}\ V) \wedge A \\
\\
\forall\, A : \mathit{Prop};\ V : \mathit{PropVar} \bullet \\
\qquad A \wedge V = A \wedge (\mathrm{var}\ V) \\
\\
\forall\, V, W : \mathit{PropVar} \bullet \\
\qquad V \wedge W = (\mathrm{var}\ V) \wedge (\mathrm{var}\ W)
\end{array}
$$

2.11 $\vee$ `\orProp`

Let A and B be propositions. Let $A \vee B$ denote the *disjunction* of A and B.

2.12 $\vee$ `\orPropVP`, $\vee$ `\orPropPV`, and $\vee$ `\orPropVV`

We can simplify the notation for disjunctions involving propositions defined by propositional variables as follows.

$$
\begin{array}{|l}
\_ \lor \_ : \mathit{PropVar} \times \mathit{Prop} \rightarrowtail \mathit{Prop} \\
\_ \lor \_ : \mathit{Prop} \times \mathit{PropVar} \rightarrowtail \mathit{Prop} \\
\_ \lor \_ : \mathit{PropVar} \times \mathit{PropVar} \rightarrowtail \mathit{Prop} \\
\hline
\forall V : \mathit{PropVar};\ A : \mathit{Prop} \bullet \\
\qquad V \lor A = (\mathrm{var}\ V) \lor A \\
\forall A : \mathit{Prop};\ V : \mathit{PropVar} \bullet \\
\qquad A \lor V = A \lor (\mathrm{var}\ V) \\
\forall V, W : \mathit{PropVar} \bullet \\
\qquad V \lor W = (\mathrm{var}\ V) \lor (\mathrm{var}\ W)
\end{array}
$$

2.13 $\Rightarrow$ `\impliesProp`

Let A and B be propositions. Let $A \Rightarrow B$ denote the *implication* of A and B.

2.14 $\Rightarrow$ `\impliesPropVP`, $\Rightarrow$ `\impliesPropPV`, and $\Rightarrow$ `\impliesPropVV`

We can simplify the notation for implications involving propositions defined by propositional variables as follows.

$$
\begin{array}{|l}
\_ \Rightarrow \_ : \mathit{PropVar} \times \mathit{Prop} \rightarrowtail \mathit{Prop} \\
\_ \Rightarrow \_ : \mathit{Prop} \times \mathit{PropVar} \rightarrowtail \mathit{Prop} \\
\_ \Rightarrow \_ : \mathit{PropVar} \times \mathit{PropVar} \rightarrowtail \mathit{Prop} \\
\hline
\forall V : \mathit{PropVar};\ A : \mathit{Prop} \bullet \\
\qquad V \Rightarrow A = (\mathrm{var}\ V) \Rightarrow A \\
\forall A : \mathit{Prop};\ V : \mathit{PropVar} \bullet \\
\qquad A \Rightarrow V = A \Rightarrow (\mathrm{var}\ V) \\
\forall V, W : \mathit{PropVar} \bullet \\
\qquad V \Rightarrow W = (\mathrm{var}\ V) \Rightarrow (\mathrm{var}\ W)
\end{array}
$$

2.15 $\Leftrightarrow$ `\equivProp`

Let A and B be propositions. Let $A \Leftrightarrow B$ denote the *equivalence* of A and B.

2.16 $\Leftrightarrow$ `\equivPropVP`, $\Leftrightarrow$ `\equivPropPV`, and $\Leftrightarrow$ `\equivPropVV`

We can simplify the notation for equivalences involving propositions defined by propositional variables as follows.

$$
\begin{array}{|l}
\_ \Leftrightarrow \_ : PropVar \times Prop \rightarrowtail Prop \\
\_ \Leftrightarrow \_ : Prop \times PropVar \rightarrowtail Prop \\
\_ \Leftrightarrow \_ : PropVar \times PropVar \rightarrowtail Prop \\
\hline
\forall V : PropVar; A : Prop \bullet \\
\qquad V \Leftrightarrow A = (\mathrm{var}\ V) \Leftrightarrow A \\
\\
\forall A : Prop; V : PropVar \bullet \\
\qquad A \Leftrightarrow V = A \Leftrightarrow (\mathrm{var}\ V) \\
\\
\forall V, W : PropVar \bullet \\
\qquad V \Leftrightarrow W = (\mathrm{var}\ V) \Leftrightarrow (\mathrm{var}\ W)
\end{array}
$$