Proofs

Arthur Ryman, arthur.ryman@gmail.com

November 10, 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 from that one finds in mathematical papers.

A Coq proof consists of applications of so-called *tactics*. Each tactic represents a higher level aggregate of deductions. This makes sense for a proof assistant since its job is to help the user discover proofs. Tactics are like macros and as such they alleviate the user from much low-level tedium. However, the analog of a macro in normal mathematical writing is a lemma. Perhaps tactics are more useful for proving formal properties of programming languages where the mathematical objects of interest are complex, but finite, recursive structures.

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. In fact Lemmon makes that point that, although there is no mechanical way to discover proofs, they can be mechanically checked.

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, and then move on to predicate calculus.

2 The Propositional Calculus 1

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*.

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. This approach corresponds to Lemmon's Chapter 1. He returns to the issue of formal languages in Chapter 2.

2.1 *Prop*

Z notation has a convenient mechanism for specifying the structure of ASTs, namely that of *free types*. My first impulse was to use that mechanism to define propositions. However, the problem with free types is that they are it closed in the sense that all ways of constructing members of a free type must be specified when the free type is defined. Lemmon's book gradually introduces ways of constructing propositions, so in the interest of following his development of the subject as closely as possible, I won't define propositions that way.

On closer examination of free types in Z, it will be observed that they are merely syntactic sugar for introducing a new given set along with an exhaustive set of constructors for its elements. The constraint expressing the condition that the constructors are exhaustive is equivalent to saying that the domains of the constructors partition the set of propositions.

Therefore, given any subset of constructors, one can define the corresponding set of propositions that can be constructed from them. This allows new constructors to be gradually introduced. I'll take that approach.

Let *Prop* denote the set of all propositions.

$$[\mathit{Prop}]$$

2.2 *PropVar*

The separation of the form of a statement from its content with respect to any given subject matter is accomplished by the use of *propositional variables.* A propositional variable stands for an arbitrary statement that is either true or false. Let *PropVar* denote the set of propositional variables.

$$PropVar : \mathbb{P}\, Prop$$

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 which represent distinct propositions.

$$\mathrm{P}, \mathrm{Q}, \mathrm{R}, \mathrm{S}, \mathrm{T} : PropVar$$

$$\mathsf{disjoint}\ \langle \{\mathrm{P}\}, \{\mathrm{Q}\}, \{\mathrm{R}\}, \{\mathrm{S}\}, \{\mathrm{T}\} \rangle$$

2.4 *PropLetter*

Let *PropLetter* denote the set of all proposition letters.

$$PropLetter == \{\mathrm{P}, \mathrm{Q}, \mathrm{R}, \mathrm{S}, \mathrm{T}\}$$

2.5 $'$ `\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. Appending a prime to a propositional variable is an injection from *PropVar* to *PropVar*.

$$\_' : PropVar \rightarrowtail PropVar$$

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

$$\mathrm{P}' \in PropVar$$

$$\mathrm{Q}'' \in PropVar$$

A propositional variable is either a letter or is primed.

$$\langle PropLetter, \mathrm{ran}(\_') \rangle\ \mathsf{partition}\ PropVar$$

2.6 $-$ `\notProp`

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

$$| \quad - : \mathit{Prop} \rightarrowtail \mathit{Prop}$$

Example. $-\mathrm{P}$ *is a negation.*

$$-\mathrm{P} \in \mathit{Prop}$$

2.7 *Negation*

Let *Negation* denote that set of all negations.

$$\mathit{Negation} == \mathrm{ran} -$$

2.8 & `\andProp`

Let A and B be propositions. Let $A \mathbin{\&} B$ denote the *conjunction* of A and B.

$$| \quad \_ \mathbin{\&} \_ : \mathit{Prop} \times \mathit{Prop} \rightarrowtail \mathit{Prop}$$

2.9 *Conjunction*

Let *Conjunction* denote the set of all conjunctions.

$$\mathit{Conjunction} == \mathrm{ran}(\_ \mathbin{\&} \_)$$

2.10 v `\orProp`

Let A and B be propositions. Let $A \mathbin{\mathrm{v}} B$ denote the *disjunction* of A and B.

$$| \quad \_ \mathbin{\mathrm{v}} \_ : \mathit{Prop} \times \mathit{Prop} \rightarrowtail \mathit{Prop}$$

2.11 *Disjunction*

Let *Disjunction* denote the set of all disjunctions.

$$\mathit{Disjunction} == \mathrm{ran}(\_ \mathbin{\mathrm{v}} \_)$$

2.12 $\longrightarrow$ `\impliesProp`

Let A and B be propositions. Let $A \longrightarrow B$ denote the *conditional* of A and B.

$$| \quad \_ \longrightarrow \_ : \mathit{Prop} \times \mathit{Prop} \rightarrowtail \mathit{Prop}$$

2.13 *Conditional*

Let *Conditional* denote the set of all conditionals.

$$Conditional == \operatorname{ran}(\_ \longrightarrow \_)$$

2.14 $\longleftrightarrow$ `\equivProp`

Let A and B be propositions. Let $A \longleftrightarrow B$ denote the *biconditional* of A and B.

$$\_ \longleftrightarrow \_ : Prop \times Prop \rightarrowtail Prop$$

2.15 *Biconditional*

$$Biconditional == \operatorname{ran}(\_ \longleftrightarrow \_)$$

3 Proofs

Lemmon has a nice way of presenting proofs. Here's an example.

1 $P \Rightarrow Q, P \vdash Q$

1	(1)	$P \Rightarrow Q$	A
2	(2)	P	A
1,2	(3)	Q	1,2 MPP

A proof consists of a *sequent* to be proved and a finite sequence of one or more *lines*. The lines are labelled by consecutive natural numbers, starting at 1. Each line contains a proposition and the application of the *proof rule* used to add it to the proof. A proof rule is either an *assumption* or a *derivation*. The *rule of assumption* allows the addition of any proposition. The *rules of derivation* allow the addition of a *conclusion* derived from *premises* that have been previously added. A premise is therefore either an assumption or the conclusion of a previous deduction. Every line of the proof can therefore be traced back to a finite, possibly empty, set of assumptions upon which it ultimately *depends*.

3.1 *ProofRule*

Let *ProofRule* denote the set of proof rule applications.

$$\begin{aligned} ProofRule ::= {} & \mathrm{A} \\ | {} & \mathrm{MPP}\langle\langle \mathbb{N}_1 \times \mathbb{N}_1 \rangle\rangle \end{aligned}$$

Example. A *and* MPP$(1, 2)$*are proof rules applications.*

$$\mathrm{A} \in ProofRule$$

$$\mathrm{MPP}(1, 2) \in ProofRule$$

3.2 *ProofLine*

A proof line consists of a finite, possibly empty, set of assumptions, a proposition, and the proof rule application used to derive the proposition. Let *ProofLine* denote the set of all proof lines.

ProofLine

$$\begin{array}{l} \mathit{assumptions} : \mathbb{F}\,\mathbb{N}_1 \\ \mathit{prop} : \mathit{Prop} \\ \mathit{rule} : \mathit{ProofRule} \end{array}$$

3.3 *ProofLineTuple*

Let *ProofLineTuple* denote the tuple formed by the components of a proof line.

$$\mathit{ProofLineTuple} == \mathbb{F}\,\mathbb{N}_1 \times \mathit{Prop} \times \mathit{ProofRule}$$

Example. *Line 3 of proof* **1** *corresponds to the following proof line.*

$$(\{1, 2\}, \mathrm{Q}, \mathrm{MPP}(1, 2)) \in \mathit{ProofLineTuple}$$

3.4 tuple `\proofLineTuple`

Let tuple map a proof line to its tuple.

$$\mathrm{tuple} == (\lambda\, \mathit{ProofLine} \bullet (\mathit{assumptions}, \mathit{prop}, \mathit{rule}))$$

Remark. *The mapping from proof lines to tuples is a bijection.*

$$\mathrm{tuple} \in \mathit{ProofLine} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathit{ProofLineTuple}$$

3.5 *Argument*

An *argument* is a finite sequence of proof lines. Let *Argument* denote the set of all arguments.

$$\mathit{Argument} == \mathrm{seq}\,\mathit{ProofLineTuple}$$

3.6 Proof_1

Let Proof_1 denote the argument defined by proof **1**.

$$\begin{array}{l} \mathit{Proof}_1 == \\ \quad \{1 \mapsto (\{1\}, \mathrm{P} \longrightarrow \mathrm{Q}, \mathrm{A}), \\ \quad 2 \mapsto (\{2\}, \mathrm{P}, \mathrm{A}), \\ \quad 3 \mapsto (\{1, 2\}, \mathrm{Q}, \mathrm{MPP}(1, 2))\} \end{array}$$

Example. *$Proof_1$ is an argument.*

$$Proof_1 \in Argument$$

3.7 *ArgumentLine*

A argument is said to be *sound* or *valid* if each proof line in the argument obeys the rules of derivation. When assessing the validity of a proof line, we need to specify its line number within the argument and examine its contents. Let *ArgumentLine* denote the set of all arguments and line numbers within it.

$$\begin{array}{|l}
\hline
\textit{ArgumentLine} \\
\textit{argument} : \textit{Argument} \\
\textit{lineNumber} : \mathbb{N}_1 \\
\hline
\textit{lineNumber} \in \operatorname{dom} \textit{argument} \\
\hline
\end{array}$$

- The line number corresponds to a proof line within the argument.

Example. *Line 3 in contained in proof* **1**.

$$\begin{array}{l}
\textbf{let } \textit{argument} == Proof_1;\ \textit{lineNumber} == 3 \bullet \\
\quad \textit{ArgumentLine}
\end{array}$$

3.8 *RuleOfAssumption*

A proof line that uses the rule of assumption A in an argument is sound if it depends only on itself. Let *RuleOfAssumption* denote the set of all argument lines that use the rule of assumption soundly.

$$\begin{array}{|l}
\hline
\textit{RuleOfAssumption} \\
\textit{ArgumentLine} \\
P : \textit{Prop} \\
\hline
\textit{argument}(\textit{lineNumber}) = (\{\textit{lineNumber}\}, P, \mathrm{A}) \\
\hline
\end{array}$$

- The line depends only on itself and the rule is the rule of assumptions A.

3.9 *AssumptionArgumentLine*

Let *AssumptionArgumentLine* denote the set of all argument lines that are sound applications of the rule of assumption.

$$\textit{AssumptionArgumentLine} \mathrel{\widehat{=}} \textit{RuleOfAssumption} \upharpoonright \textit{ArgumentLine}$$

Example. *Lines 1 and 2 of $Proof_1$ are sound applications of the rule of assumption.*

$$\begin{array}{l} \textbf{let } argument == Proof_1 \bullet \\ \quad \forall\, lineNumber : \{1,2\} \bullet \\ \qquad AssumptionArgumentLine \end{array}$$

3.10 *RuleOfMPP*

A proof line that uses the rule MPP(i,j) in an argument is sound if i and j are lines that precede it, the proposition on line i is an implication, the proposition on line j is the antecedent of the implication, the proposition of the proof line is the consequent of the implication, and the proof line's assumptions is the union of the assumptions of lines i and j. Let *RuleOfMPP* denote the set of all argument lines that use the rule of MPP soundly.

$$\begin{array}{|l} \hline RuleOfMPP \\ ArgumentLine \\ i, j : \mathbb{N}_1 \\ ProofLine_1 \\ ProofLine_2 \\ P, Q : Prop \\ \hline i < lineNumber \land j < lineNumber \\ argument(i) = (assumptions_1, P \longrightarrow Q, rule_1) \\ argument(j) = (assumptions_2, P, rule_2) \\ argument(lineNumber) = (assumptions_1 \cup assumptions_2, Q, \mathrm{MPP}(i,j)) \\ \hline \end{array}$$

3.11 *MPPArgumentLine*

Let *MPPArgumentLine* denote the set of all argument lines that are sound applications of the rule of MPP.

$$MPPArgumentLine \mathrel{\widehat=} RuleOfMPP \upharpoonright ArgumentLine$$

Example. *Line 3 of $Proof_1$ is a sound application of the rule of MPP.*

$$\begin{array}{l} \textbf{let } argument == Proof_1; \\ \quad lineNumber == 3 \bullet \\ \qquad MPPArgumentLine \end{array}$$

3.12 *SoundArgumentLine*

An argument line is sound if it is a sound application of one of the rules of derivation. Let *SoundArgumentLine* denote the set of all sound argument lines.

$$\begin{aligned}&\mathit{SoundArgumentLine} \mathrel{\widehat=}\\ &\quad \mathit{AssumptionArgumentLine} \lor\\ &\quad \mathit{MPPArgumentLine}\end{aligned}$$

3.13 *Proof*

A proof is an argument in which every line is sound. Let *Proof* denote the set of proofs.

$$\begin{aligned}&\mathit{Proof} == \{\, \mathit{argument} : \mathit{Argument} \mid\\ &\quad \forall\, \mathit{lineNumber} : \operatorname{dom} \mathit{argument} \bullet\\ &\qquad \mathit{SoundArgumentLine} \,\}\end{aligned}$$

Example. *Proof_1 is a proof.*

$$\mathit{Proof}_1 \in \mathit{Proof}$$

3.14 *Proof_2*

Here is Lemmon's proof **2**.

2 $\neg\, Q \Rightarrow (\neg\, P \Rightarrow Q), \neg\, Q \vdash \neg\, P \Rightarrow Q$

1	(1)	$\neg\, Q \Rightarrow (\neg\, P \Rightarrow Q)$	A
2	(2)	$\neg\, Q$	A
1,2	(3)	$\neg\, P \Rightarrow Q$	1,2 MPP

Let Proof_2 denote this proof.

$$\begin{aligned}&\mathit{Proof}_2 ==\\ &\quad \{1 \mapsto (\{1\}, -\mathrm{Q} \longrightarrow (-\mathrm{P} \longrightarrow \mathrm{Q}), \mathrm{A}),\\ &\quad 2 \mapsto (\{2\}, -\mathrm{Q}, \mathrm{A}),\\ &\quad 3 \mapsto (\{1, 2\}, -\mathrm{P} \longrightarrow \mathrm{Q}, \mathrm{MPP}(1, 2))\}\end{aligned}$$

Example. *Proof_2 is a proof.*

$$\mathit{Proof}_2 \in \mathit{Proof}$$

4 The Curry-Howard Correspondence

The *Curry-Howard Correspondence* is an interpretation of propositions and proofs in terms of types. To each proposition there correspondences a type. The inhabitants of the type that corresponds to a proposition are proofs of that proposition. The logical connectives that build up propositions correspond to type constructors.

To illustrate the correspondence, consider the implication logical connective $P \Rightarrow Q$. It corresponds to the type constructor $P \longrightarrow Q$. A proof f of $P \Rightarrow Q$ corresponds to a function that maps any proof of P to some proof of Q.

Of course, $P \longrightarrow Q$ is not a type in Z. It is a subset of the type $\mathbb{P}(P \times Q)$. Nevertheless, we'll press on and pretend that it is a Z type for now.

The rules of derivation correspond to the construction of a proof from assumptions. For example the rule of MPP corresponds to function application. Consider Proof_1. It corresponds to the following.

$$\begin{array}{|l}
\hline
\mathit{CHProof}1[P, Q] \\
f : P \longrightarrow Q \\
x : P \\
y : Q \\
\hline
y = f(x) \\
\hline
\end{array}$$