Corneliu Hoffman

Discrete Mathematics with Proof assistants

January 5, 2018

Springer

Contents

Preface

> Proofs are to mathematics what spelling (or even calligraphy) is to poetry. Mathematical works do consist of proofs, just as poems do consist of characters.
>
> Vladimir Arnold

There are numerous studies about the image of mathematics among professional mathematicians and among the general public. The general public holds the idea that Mathematics is a series of formulas and calculations that are useful but are so complicated that they satisfy the old Arthur C Clarke quote "Any sufficiently advanced technology is indistinguishable from magic.".

However similar studies among mathematicians provides a completely different result. Indeed most mathematicians think that the main transferable skill that Mathematics Education brings is not the ability to do the long calculations but the ability to reason correctly and to use abstraction in solving problems.

Therefore, while one should not discard the intrinsic value of effective computational tools, Mathematics is "about proofs". Mathematics Education should reflect this. This approach is as old as teaching of Mathematics. Euclid's Elements, probably the oldest textbook in the western world is a collection of proofs and constructions. It has lead the teaching of mathematics for much of last two thousand years.

However, the teaching of proofs is loosing ground in the modern Mathematics curriculum. Indeed, for example, in the UK Advance level exam from 1957, 7 of the 10 questions involved a small proof. By comparison the 2016 equivalent (C4 AQA test), only 16 out of 75 points were proofs. This in not the place to discuss the many reasons for this development. Nevertheless, the result is that many students start university expecting that mathematics is a series of cookbook methods and computations. One of the most serious stumbling blocks in University Mathematics is the lack of exposure to proofs.

This text is meant to be an attempt to address this. There are, of course, countless textbooks of the kind so writing yet another standard one would be rather pointless. We will therefore attempt to be a little non-standard.

In the last few decades computer aided education has come to prominence. Computer Algebra systems such as Maple, Mathematica, Mathlab, Maxima, Sage and so on have permeated the curriculum providing examples, modelling, automated assessment and so on. By comparison, the teaching of proofs and abstractions have seen almost none of these.

Of course computer assisted proofs are almost as old as computers. Already in 1954 Martin Davis encoded Presburger's arithmetic and managed to prove that the sum of even numbers is even. More importantly, a few years later Newell, Simon and Shaw wrote the "Logic Theorist", a first order logic solver that managed to prove 38 of the theorems in Russell and Whitehead's "Principia Mathematica". The development of PROLOG in the 70's offered a reasonably simple context to verify first order logic.

Until quite recently thought, proof assistants belonged to the world of Computer Science, more precisely program verification. Very little of the advances in the domain crossed over into mathematics or mathematics education.

In the 80's a plethora of Proof assistants came to prominence. While initially they mimicked the standard language of mathematics (see for example Mizar), they soon simplified their notations for the sake of efficiency. Despite attempts by some developers (such as the decorative mode Isar for Isabelle), most proof assistants are beyond the reach of a beginning mathematics student.

We have made several attempts to teach Mathematics with the help of Isabelle and Coq and they all had only modest success. The syntax proved to be too much for the students. The solution we found was to develop a separate interface for Coq that will separate the student from both the terseness and the automation power of the theorem prover and will provide an accessible and interactive syntax. The resulting product is called Spatchcoq, after the method of "butterflying " a chicken prior to cooking. We hope that the wonderful authors of Coq would forgive our little inside joke.

The idea of the book is to teach some topics in discrete mathematics (the standard way of introducing proofs) with the help of Spatchcoq. The reader is encouraged to download the software using the instructions in Appendix A.

We will slowly introduce the software together with basic methods of proofs in Chapter 1 van numerous examples. The inpatient reader can skip quickly to Appendix B to get short descriptions of the tactics, respectively to Appendix C to see two detailed examples. You can also find some other examples of proofs at

https://github.com/corneliuhoffman/spatchcoqocaml/tree/master/examples

Chapter 1
Basic proof techniques.

> Nothing goes over my head. My reflexes are too fast, I would catch it.
>
> Drax.

1.1 Motivational Speeches

We start the book with a few sections on Mathematical logic. This is usually a hard thing to grasp at the first sight so we hope to give enough examples to help the reader. One of the issues is the fact the English language is more nuanced than mathematical logic. To exemplify this we will choose a common internet meme[1].

The phrase "I never said she stole my money." has 7 different meanings depending on the emphasis. For example "**I** never said she stole my money" means that perhaps somebody else said it, "I never said **she** stole my money" means that I said that somebody else stole it while "I never said she stole my **money**." means that perhaps she stole something else.

Mathematical logic is a lot more precise than that. Every statement has to be either true or false. Nuances have no place in this. You have to formulate statements in such a way that there is only one interpretation. However, as we will see later, there are some subtleties about the meaning of negation.

[1] My son Luca showed it to me.

1.2 Propositional Calculus

A lot of concepts in Mathematics (and CS) rely on the notion of a proposition. We will consider that an elementary notion.

Definition 1 *A proposition is a sentence which is either true or false but not both.*

This seems like an rather pompous definition but it is very important for what follows. here are some examples of propositions:

- Earth is a planet.
- $2 + 2 = 5$
- $\forall x \in \mathbb{R}, x^2 \geq 0$
- Men are mortal.

However, the day to day language is much richer than mere logic. We can give many examples of sentences that are not propositions.

- What time is it?
- We are better off today than 3 years ago.
- $x + 3 > 5$
- It will rain tomorrow.

1.2.1 Connectors

Just like other mathematical concepts, there is a "calculus" for dealing with propositions. Some versions of this date back to Aristotle but they have been slightly modified thorough the ages. There are five connectors (operators) on the set of propositions. There are: and ($\wedge$), or ($\vee$), implication ($\rightarrow$), iff ($\leftrightarrow$) and negation($\neg$). The first four are so called "binary operators", that is they combine two propositions into one and the last one is a "unary operation" much like minus is for numbers. Most of these notions will seem familiar to you, our approach will however be a little bit more formal. Note also that the symbols are not indempendent, one can express some of the symbols using the others.

Conjunction

If P and Q are two propositions then their conjunction, denoted by $P \wedge Q$ is a proposition that is only true if both P and Q are true. We can write the Truth table bellow:

P	Q	$P \wedge Q$
T	T	T
T	F	F
F	T	F
F	F	F

here are some examples.

- "She is both intelligent and hard working" can be written as ("She is intelligent")$\wedge$("She is hard working").
- $0 < 4 < 5$ can be written as $(0 < 4) \wedge (4 < 5)$.

Disjunction

if P and Q are two propositions then their disjunction, denoted by $P \vee Q$ is a proposition that is only false if both P and Q are false. That is it is true if either P or Q or both are true. Note that in SpatchCoq you can enter this by either clicking on the symbol or by typing \/.

The corresponding truth table is:

P	Q	$P \vee Q$
T	T	T
T	F	T
F	T	T
F	F	F

Here are some examples:

- "He is either at work or on his way home" can be written as ("He is at work")$\vee$("He is on his way home").
- $0 \leq 4$ can be written as $(0 < 4) \vee (0 = 4)$

Note that, unlike in nature language, the connector $\vee$ is not an "exclusive or". The proposition $P \vee Q$ is true in the case both P and Q are true.

Implication

If P and Q are two propositions then $P \to Q$ (reads P implies Q) is a proposition that is only false if P is true and Q is false. In other words, we can produce the following "Truth Table":

P	Q	$P \to Q$
T	T	T
T	F	F
F	T	T
F	F	T

- "If it rains then you need your umbrella" can be written as ("It rains")→("you need your umbrella").

This is the most counterintuitive of all connectors. Note that if the proposition P is false then $P \to Q$ is automatically true regardless of the truth value of Q.

Negation

if P is a proposition then its negation, denoted by $\neg P$ is a proposition that is false if P is true and true if P is false. Note that in SpatchCoq this can be typed by clicking on the symbol or by writing not. The truth table is very simple:

P	$\neg P$
T	F
F	T

- "It is not raining" can be written as ¬(" It rains").
- $0 \leq 4$ can be written as $\neg(0 > 4)$

If and only if

If P and Q are propositions then $P \leftrightarrow Q$ is the same as $(P \to Q) \wedge (Q \to P)$. It means that the two propositions have the exact same truth Value. The truth table is:

P	Q	$P \leftrightarrow Q$
T	T	T
T	F	F
F	T	F
F	F	T

As you saw in the definition of $\leftrightarrow$, the various connectors are not independent. For example note the following truth tables:

P	Q	$P \to Q$	$\neg P$	$(\neg P) \vee Q$
T	T	T	F	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

P	$\neg P$	$\neg(\neg P)$
T	F	T
F	T	F

P	Q	$\neg P$	$\neg Q$	$P \vee Q$	$\neg(P \vee Q)$	$(\neg P) \wedge \neg Q$
T	T	F	F	T	F	F
T	F	F	T	T	F	F
F	T	T	F	T	F	F
F	F	T	T	F	T	T

Two composed propositions that have the exact truth values regardless of the values of their components are called equivalent. In other words the statement $P \to Q$ is equivalent to $\neg P \vee Q$, the propositiont P is equivalent to $\neg\neg P$ and $\neg(P \vee Q)$ is equivalent to proposition $(\neg P) \wedge \neg Q$.

A proposition that is equivalent to the proposition True (i.e one that is True regardless of the value of its components) is called a tautology. For example, if P and Q are equivalent then $(\neg P) \vee Q$ is a tautology. In particular

$$(P \vee Q) \vee (\neg P \wedge \neg Q)$$

is a tautology.

1.2.2 Translation between english and propositional calculus

1.2.3 Inference rules

Of course any argument in propositional logic can be solved with a truth table. However this is quite tedious and hard to extend to more general notions. We prefer to use methods of proof called "inference rules". Each connector has two rules, an introduction and an elimination rule. We will also describe them using the standard logic notation. More precisely, the notation

$$\frac{P}{Q}\ name$$

means that the inference rule "name" allows you to infer Q from P.

remark

In brief, the introduction rule of a connector tells you what to do in order to prove a propositions involving the connector.

remark

The elimination rule of a connector tells you how to use a hypothesis involving the connector to prove other things.

We will list these in the next section. Note that those connectors will be used later in Predicate calculus and there we will be able to give many more examples. We will also take this opportunity to introduce some of SpatchCoq's tactics.

Forward proofs, backward proofs and implication rules

Let us describe the **implication introduction rule**. In order to prove the statement $P \to Q$ we assume P and try to prove Q. In usual logic notation we have:

$$\frac{\begin{array}{c} P \\ \vdots \\ Q \end{array}}{P \to Q}$$

The equivalent SpatchCoq tactic is "Assume P then prove Q."

The **implication elimination rule** is sometimes called "modus ponens". If you have the hypothesis $H : P \to Q$ and the hypothesis $H1 : P$ then you can show Q. In logical notation

$$\frac{P \to Q \quad P}{Q}$$

In SpatchCoq we use the tactic "Apply result (H H1)" or "Apply result H1." followed by "Apply result H."

Most of the rpoofs you have seen written in textbooks are written in a style called "direct proof". Suppose you have a set of hypotheses and you want to prove a conclusion. You then start from the hypotheses and prove a series of intermediate results that then get added to you hypotheses until you can prove the conclusion. Most of the time in practice however the way you arrive to a proof is using a method called "backward proof".

To fix the details we will prove one example, the famous Aristotelian syllogism:

Socrates is a Man.

All men are mortal.

Therefore Socrates is mortal.

We will be somewhat abusive using 3 propositions **Socrates, Man, Mortal**. We will redo this more carefully in Section 1.3.

We have two Axioms,

$$\text{A1} : \textbf{Socrates} \to \textbf{Man}.$$

$$\text{A2: } \textbf{Man} \to \textbf{Mortal}.$$

And we need to show that

$$\textbf{Socrates} \to \textbf{Mortal}.$$

To do that we need to use Implication introduction, that is we need to assume **Socrates** and try to prove **Mortal**.

We start by giving a "foward proof" of this. Since we know A1 and **Socrates**, implication elimination tells us that we have **Man**. Similarly since we know A2 and **Man**, implication elimination gives you **Mortal**, which is what we needed to prove.

We could have argued backwardly as follows: Since we know A2, by implication elimination, in order to prove **Mortal** it is enough to prove **Man**. Similarly from A1 in order to prove **Man** it is enough to prove **Socrates** which we already have as a hypothesis.

In more complicated proofs one often combines the two methods.

The corresponding argument in SpatchCoq goes as follows. We first set up the three variables:

```
Variables Socrates Man Mortal :Prop.
```

And then list the two axioms

```
Axiom A1 : Socrates -> Man.
```

```
Axiom A2 : Man -> Mortal.
```

And finally type

```
Lemma soc: Socrates->Mortal.
```

To get

```

Socrates → Mortal
```

Next we use

```
Assume Socrates then prove Mortal.
```

to get

$$Hyp : Socrates$$

$$Mortal$$

We can now go two ways.

The first one is a "forward proof", very much like the text above, use:

```
Obtain Man applying A1 to Hyp.
```

to get

$$Hyp : Socrates$$
$$H : Man$$

$$Mortal$$

and then

```
Obtain Mortal applying A2 to H.
```

and

```
This follows from assumptions.
```

to finish the proof.

The second method is a "backward proof", this is a method preferred by Coq and therefore by SpatchCoq.

```
Apply result A2.
```

to get

$$Hyp : Socrates$$

$$Man$$

This is equivalent to the above. What we mean is that using A2, we now only need to show Man.

Now we do

```
Apply result A1.
```

to get

Hyp : *Socrates*

Socrates

which follows by assumption, that is

```
This follows from assumptions.
```

Of course this is such a simple example that one can do directly

```
Apply result (A2 (A1 Hyp)).
```

This might be the place to notice that implication elimination behaves much like a function application in standard mathematics. If you know $H : P \to Q$ and you know $H1 : P$ then $(HH1)$ is a proof for Q.
Moreover, the labels of the hypotheses are not mere labels. They are objects of the same type as the respective hypothesis. They can be viewed as witnesses for the truth of the respective propositions. Moreover, if we finish our proof with Qed then the name of Lemma itself becomes a witness for its proof.

Indeed try

```
Lemma soc: Socrates->Mortal.
Assume Socrates then prove Mortal.
Apply result (A2 (A1 Hyp)).
Qed.
Print soc.
```

to get

`soc =λ Hyp : Socrates, A2 (A1 Hyp)`
`: Socrates → Mortal` Which signifies that soc is a function that takes the witness Hyp of the truth of Socrates and produces a witness A2 (A1 Hyp) of the truth of Mortal. We will return to types later.

bf Inference rules for conjunction

The conjunction introduction says that in order to prove $P \wedge Q$, you need to prove both P and Q. In logic notation we have

$$\frac{P \quad Q}{P \wedge Q}$$

In SpatchCoq the tactic we use is "Prove the conjunction in the goal by first proving P then Q."

The Conjunction elimination consists of two separate rules,

$$\frac{P \wedge Q}{P} \text{ and } \frac{P \wedge Q}{Q}$$

To be more precise, if you know $H : P \wedge Q$ then you can deduce $H1 : P$ and $H2 : Q$. The corresponding SpatchCoq tactic is "Eliminate the conjuction in hypothesis H."

To exemplify this, we shall prove the commutativity of conjunction. If P, Q are propositions, then $P \wedge Q \to Q \wedge P$. To do so, we use, as above the implication introduction, so we assume that $P \wedge Q$ holds and show that $Q \wedge P$.

Now we will employ to imply the conjunction elimination. Since we know that $P \wedge Q$ holds, we also know that P holds and that Q holds. by Conjunction introduction we have that $Q \wedge P$ holds.

The formal proof in SpatchCoq is a bit more elaborate. We start with the Lemma:

```
Lemma ancomm(P Q:Prop) : P ∧ Q− > Q ∧ P.
```

to get

$PQ : Prop$

$P \wedge Q \to Q \wedge P$

We then use

```
Assume (P ∧ Q) then prove (Q ∧ P).
```

to get

$PQ : Prop$
$Hyp : P \wedge Q$

$Q \wedge P$

We know use

```
Eliminate the conjuction in hypothesis Hyp.
```

To get

$PQ : Prop$
$Hyp0 : P$
$Hyp1 : Q$

$Q \wedge P$

Now we use

```
Prove the conjunction in the goal by first proving Q then P.
```

To get two goals

$PQ : Prop$
$Hyp0 : P$
$Hyp1 : Q$

Q

and

$PQ : Prop$
$Hyp0 : P$
$Hyp1 : Q$

P

which can each be solved by

```
This follows from assumptions.
```

Inference rules for disjunction

The disjunction introduction consists of two different rules. In order to prove PQ you can either prove the left hand side or the right hand side. tHe logical expressions are

$$\frac{P}{P \vee Q}\ left \text{ and } \frac{Q}{P \vee Q}\ right.$$

In SpatchCoq we have thee tactics: “Prove left hand side.”, “Prove right hand side.” and “Prove * in the disjunction.”

Disjunction elimination is a bit harder to describe but it is a very natural method of “case by case” analysis. If you know $H : P \vee Q$ and you want to prove R then you need to prove R in case P holds as well as in case Q holds.

$$\frac{P \vee Q \quad \begin{array}{cc}\overline{P} & \overline{Q} \\ R & R\end{array}}{R}$$

In SpatchCoq the tactic is: “Consider cases based on disjunction in hypothesis H.”

We now give a detailed proof of the commutativity of disjunction:

$$P \vee Q \to Q \vee P.$$

Of course we first assume $P \vee Q$ happens and show $Q \vee P$. To do so we need to argue by cases using Disjunction elimination.

Case 1: P holds. In this case we will prove the right hand side of the disjunction in the goal.This is an assumption and by disjunction intro we are done.

Case 2: Q holds. In this case we will prove the left hand side of the disjunction in the goal. This is an assumption and by disjunction intro we are done.

Here is the spatchcoq version

```
Lemma ancomm(P Q : Prop) : P ∨ Q -> Q ∨ P.
Assume (P ∨ Q) then prove (Q ∨ P).
Consider cases based on disjunction in hypothesis Hyp.
```

at this point, there are two goals generated.

$P\ Q : Prop$
$Hyp0 : P$

$P \vee Q$

$P\ Q : Prop$
$Hyp1 : Q$

$P \vee Q$

These are easily eliminated by

```
 Prove right hand side.
This follows from assumptions.
```

respectively

```
 Prove left hand side.
This follows from assumptions.
```

Inference rules for negation

The negation introduction's logic statement is

$$\frac{\frac{P}{False}}{\neg P}.$$

In other words, the negation of P is the same thing as $P \to False$. This is an important statement to make and, indeed in SpatchCoq in order to deal with negation you will need to use "Rewrite goal using the definition of not." respectively "Rewrite hypothesis H using the definition of not.". To give an example we shall prove

$$P \to \neg\neg P$$

We of course first assume P and then prove $\neg\neg P$. To do this we first note that this is the same thing as $(P -> False) -> False$ and so we assume $P -> False$ and try to show $False$. Since now we know $P -> False$ and P, we can use implication elimination to get False.

The proof in Spatchcoq is identical:

```
Lemma notnot(P : Prop) : P → ¬¬P.
Assume P then prove (not (not P)).
Rewrite goal using the definition of not.
Assume (P → False) then prove False.
Apply result (Hyp0 Hyp).
```

Inference rules for equivalence

We will not inisist here because $P \leftrightarrow Q$ is the same as $P \to Q \wedge Q \to P$ and so the inference rules are derivative. In particular in spatchCoq we use tactic :

“Prove both directions of P iff Q.”

as introduction rule in order to prove $P \leftrightarrow Q$ and the tactic

“Eliminate the conjuction in hypothesis Hyp.”

to eliminate the hypothesis $Hyp : P \leftrightarrow Q$.

1.2.4 Constructive vs Classical

Classical logic includes a certain axiom that the romans called “tertium non datur” or “the excluded middle”. This Axiom states that of P is a proposition then $P \vee \neg P$ always hold. At the beginning of the 20th century a number of mathematicians started debating the need for such an axiom. They came to be collectively called intuitionists. The trouble with that position is that it takes away from the power of this axiom without necessarily offering something in return. The things you are able to prove are much more restrictive. As a consequence classical logic carried the day.

However at the end of the century, as Theoretical Computer Science started to gain strength and depth, excluding the excluded middled carried another promise: computability. Via the Curry-Howard correspondence, a “constructive proof” (i.e. one without the rule of excluded middle) is equivalent to the construction of a function. In particular, the familiar “proof by contradiction” relies on a variant of the excluded middle, namely the fact that the statements P and $\neg\neg P$ are equivalent. We have seen that $P \leftarrow \neg\neg P$ above but the other implication relies on classical logic.

Some “constructivists” argue that a proof of P should be a witness to its truth and not merely to the falsity of its negation (as it is the case with $\neg\neg P$. This carries quite a bit of weight in the CS world even if not (yet) so much in the Mathematical world.

One of the main methods of ”classical logic” is the so called ”proof by contradiction”. In brief, if you want to prove P then you assume that P is false and then prove a contradiction. The SpatchCoq tactic ”Prove by contradiction.” will transform the statement

into

$$\begin{array}{l}\ldots\\ H : \neg P\end{array}$$

$$False$$

As an example of proof by contradiction, consider P a proposition and $\neg\neg P$ its double negation. Are these two statements equivalent? We have proved above one of the implications. The converse, however is a bit stranger and requires a proof by contradiction.

```
 Lemma oneway (P:Prop): ¬¬P → P.
Assume (not (not P)) then prove P.
```

at this point we have

$$\begin{array}{l}P : Prop\\ Hyp : not(notP)\end{array}$$

$$P$$

so we will employ

```
 Prove by contradiction.
```

to get

$$\begin{array}{l}P : Prop\\ Hyp : not(notP)\\ H : notP\end{array}$$

$$False$$

The rest is quite standard.

```
 Apply result Hyp .
This follows from assumptions.
```

We are not ready to abandon the path of classicism and will assume excluded middle for now. We would however, try to eliminate needlessly using proofs “by contradiction”. This is a good point to look at the axiom ”classic”. If we apply

```
Check classic.
```

we get the resulst

$$classic : \forall P : Prop, P \vee \neg P$$

Therefore, while not an actual independent tactic, applying

"Apply result classic." will solve any goal that looks like

$$P \vee \neg P.$$

For example let us prove that

```
Lemma a (P Q:Prop): (P → Q) → (¬P ∨ Q).
```

We of course first use implication intros by applyiong

```
Assume (P → Q) then prove ((¬P) ∨ Q).
```

and get:

$PQ : Prop$
$Hyp : P \to Q$

$\neg P \vee Q$

At which point we are stuck without any obvious new possibility to advance. We note however that if P was true then we could use Hyp to obtain Q and if $\neg P$ is true then we would have the disjunction automatically. Thefore we do

```
Claim (P ∨ ¬P).
Apply result classic.
```

To have

$PQ : Prop$
$Hyp : P \to Q$
$H : P \vee \neg P.$

$\neg P \vee Q$

We now do a proof by cases that offers no surprises.

```
Consider cases based on disjunction in hypothesis H . Prove right hand
side.
Apply result Hyp .
This follows from assumptions.
Prove left hand side.
This follows from assumptions.
```

Note that the converse is quite easier only requiring a proof by contradiction.

```
Lemma b (PQ : Prop) : (notP ∨ Q) → (P → Q).
Assume ((notP) ∨ Q) then prove(P → Q).
Assume P then prove Q.
Consider cases based on disjunction in hypothesis Hyp .
Prove by contradiction.
Apply result Hyp1.
This follows from assumptions.
This follows from assumptions.
```

See 1.4 for a rather surprising example of classical logic.

1.2.5 Know thine fallacies

Consider the usual modus ponens rule. $\dfrac{\begin{array}{l}Q\\ P \to Q\end{array}}{P}$

Small variations can make this false. Consider for example the following argument from Mounty Python and the Holly Grail.

$$\dfrac{\begin{array}{l}\text{If it is Tuesday then I play poker}\\ \text{I play poker}\end{array}}{\text{It is Tusday}}$$

This is one of the most common "formal" fallacy. It is called "Affirming the consequence". Recall that if P is false then $P \to Q$ is automatically true if P is false and so the deduction above does not hold. It is, however, remarkably prevalent in public discourse especially in adverx

1.2.6 A puzzle

We will now use a puzzle to give a more serious example of propositional calculus, its inference rules and their implementation in SpatchCoq.

The puzzle, "the lady or the Tiger" comes from the book "The Lady Or the Tiger?: And Other Logic Puzzles" by Raymond M. Smullyan. It is slightly adapted for the 21st century.

A prisoner is offered the choice between two doors. Behind each door he could find either the key to his freedom or a very hungry tiger.

- The clue on the first door reads "the key to your freedom is in this room and the tiger in the other".
- The clue on the second door reads "one of the rooms contains the key to your freedom and the other room the tiger."
- He knows that one of the two clues is correct and the other is incorrect.

What would you do in his place?

We will formalise the questions as follows: We will denote by P the proposition "the first room contains the key to freedom" and by Q the proposition "the second room contains the key to freedom". Of course $\neg P$ means "the first room contains the tiger" and $\neg Q$ means "the second room contains the tiger".

The clue on the first door is "the key to your freedom is in this room and the tiger in the other" which can be written as

$$D1 : P \wedge \neg Q$$

.

The second door clue is "one of the rooms contains the key to your freedom and the other room the tiger." which can be rewritten as "**either** the first room has the key and the second the tiger **or** the first room has the tiger and the second the key" so we can write it as:

$$D2 : (P \wedge \neg Q) \vee (\neg P \wedge Q).$$

The fact that exactly one clue is correct and the other is incorrect can be written as "**either** the first door is correct and the second incorrect **or** the first door is incorrect and the second is correct". This can be written as $(D1 \wedge \neg D2) \vee (\neg D1 \wedge D2)$ which expands to

$$((P \wedge \neg Q) \wedge \neg((P \wedge \neg Q) \vee (\neg P \wedge Q))) \vee (\neg(P \wedge \neg Q) \wedge ((P \wedge \neg Q) \vee (\neg P \wedge Q))).$$

This looks horrible. We will however show that the second room has the key, that is Q.

For example, the statement we want to prove is

$$(D1 \wedge \neg D2) \vee (\neg D1 \wedge D2) \rightarrow Q.$$

We can set-up SpatchCoq with

```
Variables P Q:Prop.
```

to define the two propositions P and Q. Then define

```
Definition D1:= P ∧ ¬Q.
```

```
Definition D2:= (¬P ∧ Q) ∨ (P ∧ ¬Q).
```

```
Definition onlyone:= (D1 ∧ ¬D2) ∨ (¬D1 ∧ D2).
```

```
Lemma a: onlyone → Q.
```

After applying the tactic

```
Assume onlyone then prove Q.
```

we get

Hyp : *onlyone*

Q

We now use the tactic that we used for not:

```
Rewrite hypothesis Hyp using the definition of onlyone.
```

to get

Hyp : $(D1 \wedge (not\ D2)) \vee ((not\ D1) \wedge D2)$

Q

We now use

```
Consider cases based on disjunction in hypothesis Hyp .
```

to get two new goals

$Hyp0 : D1 \wedge (not\ D2)$

Q

and

$Hyp1 : not\ D1 \wedge D2$

Q

```
Eliminate the conjuction in hypothesis Hyp0.
Rewrite hypothesis Hyp1 using the definition of D2.
Rewrite hypothesis Hyp using the definition of D1.
```

brings us to

$Hyp : (P \wedge (not\ Q))$
$Hyp1 : not\ (((not\ P) \wedge Q) \vee (P \wedge (not\ Q)))$

Q

we will now use the proof by contradiction (see **??**)

```
Prove by contradiction.
```

to get

$Hyp : (P \wedge (not\ Q))$
$Hyp1 : not\ (((not\ P) \wedge Q) \vee (P \wedge (not\ Q)))$
$H : not\ Q$

$False$

We note that Hyp1 is of type (not X) that is (X→False) and so we can apply it (as in the backward proof mentioned above)

```
Apply result Hyp1
```

gives

$Hyp : (P \wedge (not\ Q))$
$Hyp1 : not\ (((not\ P) \wedge Q) \vee (P \wedge (not\ Q)))$
$H : not\ Q$

$((not\ P) \wedge Q) \vee (P \wedge (not\ Q))$

Now we note that Hyp is exactly the right hand side of the disjunction so we can use.

```
Prove right hand side.
This follows from assumptions.
```

to finish up this part of the proof.

we are now left with

$Hyp1 : not\ D1 \wedge D2$

Q

and, as above we do

```
Eliminate the conjuction in hypothesis Hyp1.
Rewrite hypothesis Hyp using the definition of D1.
Rewrite hypothesis Hyp0 using the definition of D2.
```

to get:

$Hyp : not\ (P \wedge (not\ Q))$
$Hyp0 : (((not\ P) \wedge Q) \vee (P \wedge (not\ Q)))$

Q

Since Hyp0 is a disjunction we do

```
Consider cases based on disjunction in hypothesis Hyp0 .
```

To get again a case by case analysis.

$Hyp : not\ (P \wedge (not\ Q))$
$Hyp1 : (not\ P) \wedge Q$

Q

and

$Hyp : not\ (P \wedge (not\ Q))$
$Hyp2 : P \wedge (not\ Q)$

Q

In the first case we use

```
Eliminate the conjuction in hypothesis Hyp1 .
This follows from assumptions.
```

and in the second we prove by contradiction

```
Prove by contradiction.
Apply result (Hyp Hyp2).
Qed.
```

Exercises

assume $P \to P$.

left $P \to P \vee Q$.

distr $P \wedge (Q \vee R) \to P \wedge Q) \vee (P \wedge R)$.

contrap $(P \to Q) \to (\neg Q \to \neg P)$

implies $(P \to Q) \to (\neg\, P \vee Q)$.

deMorgan $\neg\, (P \vee Q) \to (\neg\, P \wedge \neg\, Q)$.

impand $((P \to Q) \wedge (P \to R)) \leftrightarrow (P \to (Q \wedge R))$

impor $((P \to Q) \vee (P \to R)) \leftrightarrow (P \to (Q \vee R))$

andimp $(P \to (Q \to R)) \leftrightarrow ((P \wedge Q) \to R)$.

andorimp $((P \to R) \wedge (Q \to R)) \leftrightarrow ((P \vee Q) \to R)$

orandimp $((P \to R) \vee (Q \to R)) \leftrightarrow ((P \wedge Q) \to R)$

twoone $(P \vee Q) \wedge \neg P \to Q$

twotwo $\neg Q \wedge (P \to Q) \to \neg P$

twothree $C \wedge (A \to B) \wedge (C \to (A \to \neg B)) \to \neg A$

twofour $(P \vee Q) \wedge (\neg P \vee R) \to Q \vee R$.

1. What can you deduce from the following statements?
   a. All babies are illogical.
   b. Nobody is despised who can manage a crocodile.
   c. Illogical persons are despised.
2. What can you deduce from the following statements?
   a. No ducks waltz.
   b. No officers ever decline to waltz.
   c. All my poultry are ducks.
3. What can you deduce from the following statements?
   a. Things sold in the street are of no great value.
   b. Nothing but rubbish can be had for a song.
   c. Eggs of the Great Auk are very valuable.
   d. It is only what is sold in the street that is really rubbish.
4. What can you deduce from the following statements?
   a. All writers, who understand human nature, are clever.
   b. No one is a true poet unless he can stir the hearts of men.
   c. Shakespeare wrote “Hamlet”.
   d. No writer, who does not understand human nature, can stir the hearts of men.
   e. None but a true poet could have written “Hamlet”.

1.3 Predicate calculus

Nice as it might be, propositional calculus is not complete enough to express what we want. Here are some example of statements that we would like to deal with

- The equation $x^2 + x + 1 = 0$ does not have any solution.
- Some people like bread and some do not.
- If a, b, c are natural numbers,$a|b \wedge a|c \rightarrow a|(b + c)$.
- Any differentiable function is continuous.

All these require more general notion than that of a proposition, that of a predicate. For example $x > 0$ might or might not be true depending on x. We can view this as a function from $\mathbb{R}$ to the sett of propositions or as a set of propositions, parametrised by $\mathbb{R}$.

This exactly the meaning of a predicate, it is a collection of propositions parametrised by a context (type). More precisely a predicate is a function $P : U \rightarrow Prop$.

Here are some predicates.

- P(x): $x^2 + x + 1 = 0$ (here x is a real number).
- P(p): p is a prime. (here p is a natural number)
- P(x) : x is a man. (here x is an animal)
- P(x, y) : x ¿ y. (here both x an y are real numbers and so P : $mathbbR^2 \rightarrow Prop$.

Of course you cannot really prove predicates, just statements. Predicates have “free” variables and those need to be “quantified”. There are two quantifiers that bind variables. as with connectors for propositions they have introduction and elimination rules.

1.3.1 Quantifiers, free and bound variables.

As mentioned above, a predicate is a function which takes values in Prop. As such it has at least free variable (we might consider several variable. predicates). There are two ways to bind predicates, the existential and the universal quantifier. You have used both of them in a somewhat informal way. Very often you see the following colloquial statements.

” Show that $x^2 > 0$. ”

This is formally incorrect and its correct statement is : ” Show that for any real number $x, x^2 > 0$. The second statement is false since $x = 0$ is a counterexample. The first one is not a statement unless x has been defined earlier and, if it has, it might be true or false.

The **existential quantifier** is denoted by $\exists$. Its meaning is quite self explanatory. If $P : U \rightarrow Prop$ is a predicate then $\exists x : U, P(x)$ is a proposition which is true if you can find an x so that $P(x)$

is true. Note that in SpatchCoq you can enter this either by clicking on the symbol or by typing exists.

For example $\exists x : \mathbb{R}, x^2+x+1 = 0$. means that the equation $x^2+x+1 = 0$ has a solution. Therefore our first example of the section "The equation $x^2 + x + 1 = 0$ does not have any solution." can be written as $\neg(\exists x : \mathbb{R}, x^2 + x + 1 = 0)$.

If we consider the predicate "P(x) : x likes bread" on the set People of all people then "Some people like bread and some do not." can be written as $(\exists x : People, P(x)) \wedge (\exists x : People, \neg P(x))$.

The **universal quantifier** is denoted by $\forall$. As with the existential quantifier, the meaning of this is natural, the proposition $\forall a, P(a)$ will hold if the propositions $P(x)$ will hold no matter what x is. Note that in SpatchCoq you can enter this either by clicking on the symbol or by typing forall.

Note that you can encounter this in many forms. Here are some examples:

"All square integers are non-negative" is the same thing as $\forall x \in \mathbb{Z}, x^2 \geq 0$.

"The sum of any two odd numbers is even" is the same thing as $\forall x \in \mathbb{Z} \forall y \in \mathbb{Z}, odd(x) \wedge odd(y) \rightarrow even(x + y)$.

"Anybody has a friend" is the same thing as $\forall x, \exists y, friend(x, y)$.

Note that bound variables can be renamed. For example $\forall x, \exists y, friend(x, y)$ is the smae as $\forall y, \exists x, friend(y, x)$. They are also local variables so they can be reused. for example $\forall x, P(x) \rightarrow \exists y, P(y)$ can be also writtnen as $\forall x, P(x) \rightarrow \exists x, P(x)$. However one needs to be careful doing this.

Inference rules

The **existential introduction** rule: if you have a way to prove $P(a)$ for some $a : U$ then you have proved $\exists x : U, P(x)$. In logic notation this is

$$\frac{P(a)}{\exists x : U, P(x).}$$

In Spatchcoq the tactic that you need in this case is "Prove the existential claim is true for a.". In order to apply this tactic you need the goal to be of the form $\exists x : U, P(x)$ and if you apply it you now need to prove $P(a)$.

Here is a very simple example. Suppose you want to prove that $\exists x, x^2 = 4$. To do so we note that $2^2 = 4$ and so by existential introsductio the result is true. The proof in spatchcoq is

```
Lemma triv:∃n : nat, n^2 = 4.
Prove the existential claim is true for 2.
This follows from reflexivity.
```

The **existential elimination** rule: that if you have a hypothesis of the form $\exists x, P(x)$ then you can deduce $P(a)$ for some a. The logic form is

$$\frac{\exists x : U, P(x).}{P(a) \text{ for some a}}$$

The corresponding SpatchCoq tactic is "Fix VAR the existentially quantified variable in VAR.". More precisely if you have a goal that looks like

$H : \exists x : U, P(x)$

...

then the tactic "Fix a the existentially quantified variable in H." will produce a new goal of the form

$a : U$
$H : P(a)$

...

The **universal elimination** rule: you know $\forall x, P(x)$ you can deduce $P(a)$ regardless of a. The logical notation you is

$$\frac{\forall x : U, P(x).}{P(a) \text{ for any a}}$$

The SpatchCoq tactic is a bit harder to explain. If you have

$H : \forall x : U, P(x)$

...

you can use

"Obtain P(a) using variable a in the universally quantified hypothesis H."

To exemplify this we first consider the following statement

$$\forall x : U, Px \to \exists x : U, Px.$$

Nothing simpler than that right? If a statement is true for all possible values then is of course true for some value. Except for the case where there are no elements of type U at all. In that case the statement $\forall x : U, Px$ will be true but the statement $\exists x : U, Px$ will be false[2].

[2] Sounds confusing? Does it remind you of another confusing constructor? If you said "implies" then you were right. In fact implies is syntactic sugar for a special case of forall. More precisely $P \to Q$ is the same thing as $\forall a : P, Q$, that is for the statement that if you know a proof for P you get one for Q. We will not insist here, the interested reader can have a look at [**?**]

To remedy that, we shall assume the the type U is nonempty. Here is a proof of the statement

```
Variable U:Type.
Lemma a( a:U)(P : U → Prop): (∀x : U, Px) → ∃x, Px.
Assume (∀x : U, Px) then prove (∃x : U, Px).
Obtain (P a) using variable a in the universally quantified hypothesis Hyp.
Prove the existential claim is true for a.
This follows from assumptions.
```

Coq is good at universal elimination and can often match the value of the variable and so if the statement is something like this

$a : U$
$H : \forall x : U, Px$

Pa

then just using

Apply result H.

finishes the proof. For example the proof above can be done as follows:

```
Variable U:Type.
Lemma a( a:U)(P: U->Prop): (∀x : U, Px) → ∃x, Px.
Assume (∀x : U, Px) then prove (∃x : U, Px).
Prove the existential claim is true for a.
Apply result Hyp.
```

The **universal introduction** rule: in order to prove $\forall x : U, P(x)$, you fix a random $a : U$ and prove that $P(a)$ holds. The logical notation is:

$$\frac{P(a) \text{for all a}}{\forall x : U, P(x)}$$

The corresponding SpathCoq tactic works as follows: Suppose the goal is

...

$\forall x : U, P(x)$

Then the tactic Fix an arbitrary element a.

produces the goal

$a : U$

$P(a)$

.

1.4 Proof by contradiction and the Drinker's Paradox

This is a very interesting side effect of classical logic. It was popularised by R Smullyan. The statement is as follows:

In any pub there is a customer so that if he drinks then everybody drinks.

This sounds very counterintuitive but the proof is very nice and it will test your understanding of predicate calculus. In particular there will be a few applications of "proof by contradiction" and one of "Apply result classic." The idea is that you consider two cases. If everybody Drinks then there is no problem, you can pick anybody as you witness. The more difficult case is when not everybody drinks. You then pick one person that does not drink and the statement will still be true. While the idea is quite clear, writing a complete formal proof is rather difficult.

To fix the notations let say that U is the people in the bars and that $Drinks : U \to Prop$ is the predicate that verifies if somebody drinks, With this notation, our paradox becomes:

$$\exists x, (Drinks\ x \to \forall y, Drinks\ y). \tag{1.1}$$

Note that the brackets are essential. Indeed, the statement

$$(\exists x, Drinks\ x) \to (\forall y, Drinks\ y)$$

is quite obviously false.

We will go through the proof in SpatchCoq explaining each step.

We start by introducing some variables and state the Lemma. We first define a type called Customers which should be viewed as the "set" of customers[3]. Then we ask for an element a in this type and a predicate Drinks that tells you whether customers drink. The statement of the lemma is now identical to 1.1.

```
Variable (Customers:Type)(a:Customers)(Drinks: Customers->Prop).
Lemma drinker: ∃ x:Customers, (Drinks x -> ∀ y:Customers, Drinks y).
```

[3] This is a type rather than a set. The interested reader should read D

Alternatively we could have done away with Variables and write in one line at the cost of readability.

```
 Lemma drinke (Customers:Type)(a:Customers)(Drinks: Customers->Prop).r: ∃
x:Customers, (Drinks x -> ∀ y:Customers, Drinks y).
```

The next step is a nonconstructive one. We will claim that either all customers drink or not all customers drink. This is a seemingly silly statement but recall Subsection 1.2.4. We immediatly prove it by using the result classical.

```
Claim ((∀ y:Customers, Drinks y) ∨ not (forall y:Customers, (Drinks y))).
Apply result classic.
```

to get the following:

$H : (\forall y : Customers, Drinks\ y) \vee (\neg(\forall y : Customers, Drinks\ y))$

$\exists x : Customers, (Drinks\ x \rightarrow (\forall y : Customers, Drinks\ y)$

We now execute an or elimination (proof by cases) in H.

```
Consider cases based on disjunction in hypothesis H.
```

to obtain two new goals:

$H : (\forall y : Customers, Drinks\ y)$

$\exists x : Customers, (Drinks\ x \rightarrow (\forall y : Customers, Drinks\ y)$

and respectively

$\neg(\forall y : Customers, Drinks\ y)$

$\exists x : Customers, (Drinks\ x \rightarrow (\forall y : Customers, Drinks\ y)$

This forst goal is quite easy to prove. Since we already know that $\forall y : Customers, Drinks\ y$ holds (that is that everybody drinks) then it does not matter which x we pick so we will pick a and prove it. More precisely we do:

```
Prove the existential claim is true for a.
Assume (Drinks a) then prove (forall y : Customers, Drinks y).
This follows from assumptions.
```

We are now left with case where not everybody drinks. Of course we will pick the one person that does not drink. In SpatchCoq this is a bit more elaborate. We first have to prove that there is somebody that does not drink. We claim this and prove it by contradiction.

```
Claim (exists x:Customers, not (Drinks x)).
Prove by contradiction.
```

to get

$Hyp0 : \neg(\forall y : Customers, Drinks\ y)$
$H : \neg(\exists x : Customers, neg(Drinks\ x))$

$False$

Note that Hyp0 is a negation and (that is of type $P \rightarrow False$) so we acan use implication elimination

```
Apply the result Hyp0.
```

an so we now only need to prove $(\forall y : Customers, Drinks\ y)$, taht is th goal is

$Hyp0 : \neg(\forall y : Customers, Drinks\ y)$
$H : \neg(\exists x : Customers, neg(Drinks\ x))$

$(\forall y : Customers, Drinks\ y)$

We now fix an arbitrary element x (universal introduction) and agaian try to prove by contradiction:

```
Fix an arbitrary element x.
Prove by contradiction.
```

to get

$Hyp0 : \neg(\forall y : Customers, Drinks\ y)$
$H : \neg(\exists x : Customers, neg(Drinks\ x))$
$x : Customers$
$H0 : \neg(Drinks\ x)$

$False$

We now use implication elimination again, this time on H.

```
Apply result H .
```

and so we only need to prove $\exists x0 : Customers, neg(Drinks\ x0)$. We already know from $H0 : \neg(drinksx)$ that the customer x does not drink and so

```
Prove the existential claim is true for x.
This follows from assumptions.
```

will finish the proof of "Claim (exists x:Customers, not (Drinks x))."

Note that, if we were willing to use library theorems, we could obtained the same claim have searched and used the right theorem as follows: First execute search

```
SearchPattern (not (forall _ , _)->_).
```

to get an theorem

$$\text{not\_all\_ex\_not} : \forall(U : Type)(P : U \to Prop), \neg(\forall n : U, \neg P\ n) \implies \exists n : U, P\ n$$

We now use this to opbtain our claim.

```
 Obtain (exists n : Customers, not Drinks n) applying (not_all_ex_not
Customers Drinks) to Hyp0.
```

Either way the claim looks like

$Hyp0 : \neg(\forall y : Customers, Drinks\ y)$
$H : \exists x : Customers, neg(Drinks\ x))$

$\exists x : Customers, (Drinks\ x \to (\forall y : Customers, Drinks\ y)$

A standard existential elimination followed by an existential introduction and an implication introduction. that is

```
Fix b the existentially quantified variable in H .
Prove the existential claim is true for b.
Assume (Drinks b) then prove (forall y : Customers, Drinks y).
```

and we are left with

$Hyp0 : \neg(\forall y : Customers, Drinks\ y)$
$b : Customers$
$H : not(Drinks\ b)$
$HypDrinksb$

$(\forall y : Customers, Drinks\ y)$

Now H and Hyp finish the proof by contradiction.

```
Prove by contradiction.
Apply result H .
This follows from assumptions.
Qed.
```

For conformity here is the full proof bellow:

```
Variable (Customers:Type)(a:Customers)(Drinks: Customers->Prop).
Lemma drinker: ∃ x:Customers, (Drinks x -> ∀ y:Customers, Drinks y).
Claim ((∀ y:Customers, Drinks y) ∨ not (forall y:Customers, (Drinks y))).
Apply result classic.
Consider cases based on disjunction in hypothesis H .
Prove the existential claim is true for a.
Assume (Drinks a) then prove (forall y : Customers, Drinks y).
This follows from assumptions.
Claim (exists x:Customers, not (Drinks x)).
Prove by contradiction.
Apply result Hyp0 .
Fix an arbitrary element x.
Prove by contradiction.
Apply result H .
Prove the existential claim is true for x.
This follows from assumptions.
Fix b the existentially quantified variable in H .
Prove the existential claim is true for b.
Assume (Drinks b) then prove (forall y : Customers, Drinks y).
Prove by contradiction.
Apply result H .
This follows from assumptions.
Qed.
```

Appendix A
The software

A.1 Installation

A.1.1 Mac OSX

1. Download and install the latest version of Coq (it needs to be at least 8.6) from :

   https://coq.inria.fr/download

   Move it to your apps folder.
2. Download and unpack spatchocq.app from move the spatchcoq.app to Applications and start it.
3. when prompted find the Coq installation you have just move above. Navigate to

   `/Applications/CoqIDE_8.6.app/Contents/Resources/bin/`

   and choose coqtop. See Figure A.1

   You only do this once.
4. You only have to do this once. You might also need to install gtk, the simplest way to do this is via homebrew.

   If do not have homebrew installed, install it here.

   or type in the terminal:

   `/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`

   Next type in terminal:

   brew install gtk+
5. enjoy

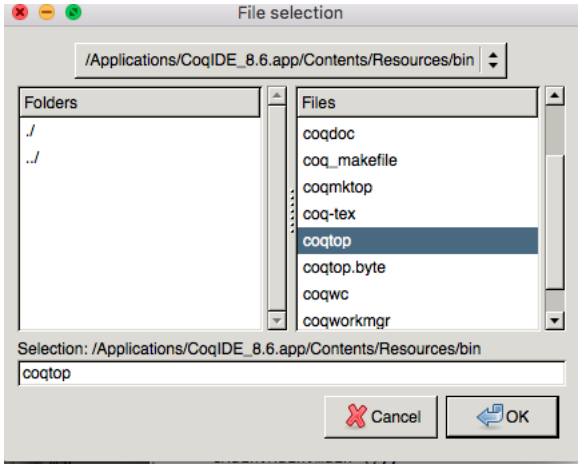


Fig. A.1 Choose the Coq app in a Mac env

A.1.2 Windows

1. get the zipfile spatchcoq.zip, unzip it in a folder on a usb stick and doubleclick the application file spatchcoq. Note this version includes an instalation of Coq (not very extensively tested yet)
2. enjoy

A.1.3 Linux

1. Download and install the latest version of Coq it needs to be at least 8.6 so do not use apt-get install coq
2. go to https://github.com/corneliuhoffman/spatchcoqocaml/tree/master to build from scratch.
3. when prompted go to the Coq folder you just installed with opam and find the application called **coqtop**
4. enjoy

A.2 Introducing the GUI

A.2.1 Main windows

Figure A.2 is a view of the GUI. As you can see there are 4 different windows and three buttons.

1. The Green window : This is the window that keeps the text that has already been processed.
2. The Yellow window ': This is the only window you can type your commands into.
3. The Gray window : this is the Coq feedback window.
4. The White window : this is a window for messages.
5. The run button: this sends the first line from the input window to Coq.
6. The undo button: this undoes the last command.
7. The draw tree button: this draws the proof trees for all the completed theorems.
8. The symbol buttons: These allows one to type mathematical symbols.
9. The Search box/button: These allow searching for theorems by pattern.

Fig. A.2 the GUI

A.2.2 The menus

The File menu (Figure A.3) is quite standard:

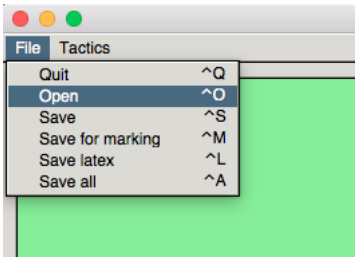


Fig. A.3 the File Menu

The Tactics menu (Figure C.0.1) allows one to pick one of the predefined tactics. Note the place keeper VAR. These can be modified. More on these later.

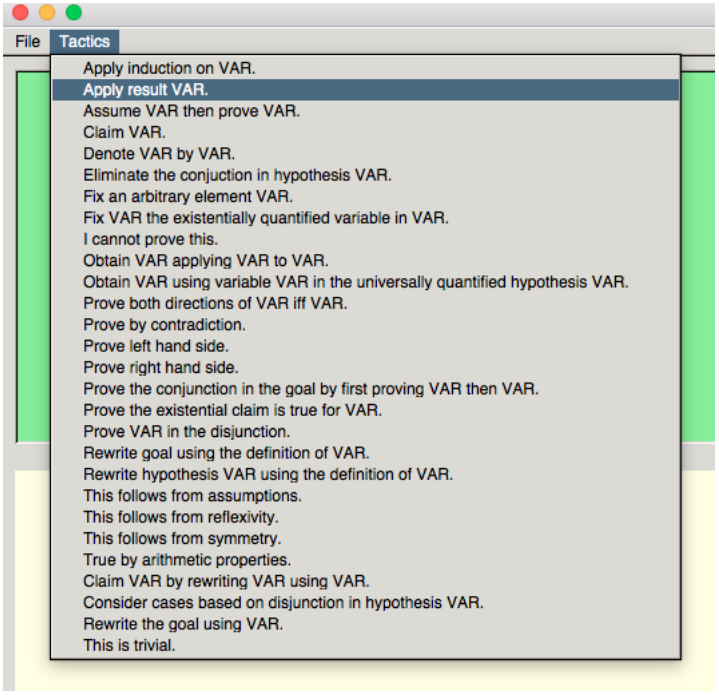


Fig. A.4 the Tactics/Environment Menus

A.2.3 Keyboard shortcuts

Pressing ESC autocompletes the commands and pressing CTRL-r circles around the various possibilities for VAR.

Appendix B
Tactics

This is trivial.

This will only work on very easy statements. If it works it will solve the current goal. Try to avoid overuse. Do better than your lecturers.

I cannot prove this.

If you are stuck this tactic will "prove" the current goal. If you use this in a proof at the end if the proof when you try to use Qed you will get the following error

> "Error:Attempt to save a proof with given up goals. If this is really what you want to do, use Admitted in place of Qed.

To avoid the error just type Admitted instead of Qed.

Prove left hand side.

Suppose you want to prove the following goal:

> ...
>
> $P \vee Q$

The above mentioned tactic will the produce a the following goal

and so you will have to now prove a simple goal.

Prove right hand side.

Symmetric with the above, suppose you want to prove the following goal:

The above mentioned tactic will the produce a the following goal

Prove VAR in the disjunction.

This tactic combines the above two. More precisely, suppose you want to prove the following goal:

Then applying

```
Prove P in the disjunction.
```

will produce the goal

while applying

```
Prove Q in the disjunction.
```

will produce the goal

$$\cdots$$

$$Q$$

Eliminate the conjuction in hypothesis VAR.

Suppose your goal looks like

$$\cdots Hyp : P \wedge Q$$
$$\cdots$$

$$\cdots$$

Then applying

```
Eliminate the conjunction in hypothesis Hyp.
```

will produce a goal similar to the one below:

$$\cdots$$
$$Hyp : P$$
$$Hyp0 : Q$$
$$\cdots$$

$$\cdots$$

allowing you to use the parts of Hyp independently.

Consider cases based on disjunction in hypothesis VAR.

Suppose your goal looks like

$\ldots$
$Hyp : P \vee Q$
$\ldots$

$\ldots$

Then applying

```
Consider cases based on disjunction in hypothesis Hyp.
```

will produce two separate goals similar to the one below:

$\ldots$
$Hyp : P$
$\ldots$

$\ldots$

$\ldots$
$Hyp : Q$
$\ldots$

$\ldots$

obtaining a proof by cases.

Prove the conjunction in the goal by first proving VAR then VAR.

Suppose your goal looks like

$\ldots$

$P \wedge Q$

Then

```
Prove the conjunction in the goal by first proving P then Q.
```

will separate the proof in two different goals

```
...
```
P

```
...
```
Q

Assume VAR then prove VAR.

Suppose your goal looks like

```
...
```
$P \to Q$

then

```
Assume P then prove Q.
```

will modify the goal to

```
...
P
```
Q

Prove both directions of VAR iff VAR.

Suppose your goal looks like

```
...
```

$P \leftrightarrow Q$

then

```
Prove both directions of P iff Q.
```

will split the goal into two different goals

```
...
```

$P \rightarrow Q$

```
...
```

$Q \rightarrow P$

Fix an arbitrary element VAR.

Suppose your goal looks like

```
...
```

$\forall x : S, P(x)$

then

```
Fix an arbitrary element a.
```

will modify the goal to

Fix VAR the existentially quantified variable in VAR.

Suppose your goal looks like

then

```
Fix a the existentially quantified variable in Hyp.
```

will modify the goal to

Obtain VAR using variable VAR in the universally quantified hypothesis VAR.

Suppose your goal looks like

then

```
Obtain Q using variable a in the universally quantified hypothesis Hyp.
```

will attempt to apply the result $P(a)$ to prove the result Q.

Prove the existential claim is true for VAR.

Suppose your goal looks like

```
...
```

$\exists x : S, P(x)$

then

```
Prove the existential claim is true for a.
```

will modify the goal to

```
...
```

$P(a)$

```
Rewrite the goal using VAR.
```

Suppose your goal looks like

```
...
```

$Hyp : x = f$

$P(x)$

then

```
Rewrite the goal using Hyp.
```

will replace every occurrence of x in P by f. Similarly if the goal is


```
Rewrite the goal using Hyp.
```

will replace every occurrence of f in P by x.

Finally if Thm is a theorem whose conclusion includes and equality $x = f$ and if the goal of your theorem looks like

Then

```
Rewrite the goal using Thm.
```

will replace every occurrence of x in P by f.

True by arithmetic properties.

This tactic will attempt to prove the statement by using the ring properties (commutativity, associativity and distributivity) of the natural, integers or reals.

Claim VAR by rewriting VAR using VAR.

This is very similar to

```
Rewrite the goal using VAR.
```

The idea is that

```
Claim Q by rewriting Hyp using Thm.
```

Will attempt to prove the statement Q by applying the rewritten version of Hyp. The rules for Thm are as above.

Claim VAR.

This is forward proof tactic.

```
Claim P.
```

will introduce a new claim, splitting the goal

into

and

Rewrite hypothesis VAR using the definition of VAR.

If the hypothesis Hyp will involve a previous definition d, then

```
Rewrite hypothesis Hyp using the definition of d.
```

will unfold a definition of d inside Hyp.

Apply induction on VAR.

This is a rather general tactic. It will generally act as an induction omnibus. More precisely

```
Apply induction on n.
```

will depend on the (inductive) type of n. For example if n is a natural number and the goal is

```
...
P(n)
```

then

```
Apply induction on n.
```

will split the proof into two goals

```
...
P(0)
```

and

```
...
IHn : P(n)
P(Sn)
```

On the other hand if n is an integer, the goal

```
...
P(n)
```

will be split into 3 cases

```
...
```

$P(0)$

```
...
n : positive
```

$P(Z.pos\ n)$

```
...
n : negative
```

$P(Z.neg\ n)$

Rewrite goal using the definition of VAR.

If the goal will involve a previous definition d, then

```
Rewrite goal using the definition of d.
```

will unfold a definition of d inside the conclusion of the goal.

obtain VAR applying VAR to VAR

.

Prove by contradiction.

Assume the goal is:

```
...
```

P

then

```
Prove by contradiction.
```

will transform the goal to

```
...
¬P
```

$False$

This follows from reflexivity.

Assume the goal is

```
...
```

$a = a$

then

```
This follows from reflexivity.
```

will finish the proof.

This follows from symmetry.

Apply result VAR.

Assume that the goal is

$$\begin{array}{l}\ldots\\ Hyp : P -> Q\\ \ldots\\ \hline Q\end{array}$$

Then

```
Apply result Hyp
```

will transform the goal to

$$\begin{array}{l}\ldots\\ Hyp : P -> Q\\ \ldots\\ \hline P\end{array}$$

Similarly if there is a theorem whose name is thm and whose conclusion is $P \rightarrow Q$then Then

```
Apply result thm
```

will transform the goal to

$$\begin{array}{l}\ldots\\ Hyp : P -> Q\\ \ldots\\ \hline P\end{array}$$

This follows from assumptions.

if the goal is

$$\begin{array}{l}\ldots\\ Hyp : P\\ \ldots\\ \hline P\end{array}$$

Then the tactic

This follows from assumptions.

finishes the proof.

Denote VAR by VAR

This is a techincal tactic.

Denote $expr_0$ **by** $expr_1$**.**

will modify the goal by adding a hypothesis

$$H : expr_1 = expr_0.$$

You usually use this to rewrite terms to simply some notations[1]. .

[1] We owe the idea of this tactic to Curt Bennett

Appendix C
Two simple examples

We give two detailed examples that will exemplify the mechanics of the GUI. For clarity we will use colour boxes that will exemplify the window that we refer to. So green boxes refer to the processed window, yellow ones to the input window and gray ones to the feedback window.

C.0.1 Propositional Calculus

We will prove that if P and Q are propositions then

$$P \vee Q \Rightarrow Q \vee P$$

the way to enter this is:

```
Lemma commor(P Q :Prop): P \/ Q -> Q \/ P .
```

Note that the feedback from Coq says

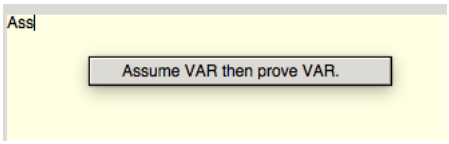
$P, Q : Prop$

$P \vee Q \rightarrow Q \vee P$

This means that the hypotheses are that P and Q are propositions and the conclusion is $P \vee Q \rightarrow Q \vee P$. To prove an implication statement we assume the left hand and try to prove the right hand. Here is how you do it in Spatchcoq. There are two different ways to do this in spatchCoq:

Type “Assume” and press ESC to get a list of tactic choices:

choose the tactic




```
Assume VAR then prove VAR.
```

Press CTRL-r to select the first VAR.

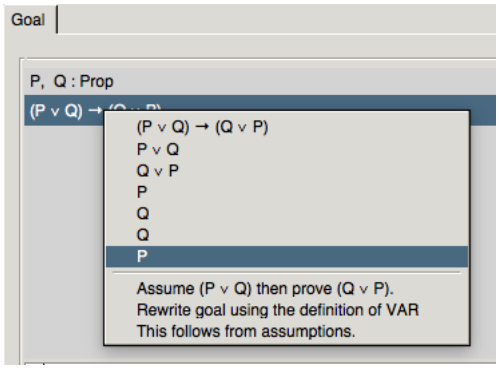
write $(P \vee Q)$ to replace the first VAR. Repeat CTRL-r and and replace the second VAR by $(Q \vee P)$.

The text in the yellow window should now be

```
Assume (P ∨ Q) then prove (Q ∨ P).
```

Click run.

The other variant is to click on the orange goal in the feedback column to get a number of to get a list of possible choices: Note that the choices bellow the horizontal line are tactics while those on



the top are pieces of the goal. You can use a combination of the two methods of course. As before choose

```
Assume (P ∨ Q) then prove (Q ∨ P).
```

and click run.

The response from Coq is

P Q : Prop
Hyp : $P \vee Q$

$Q \vee P$

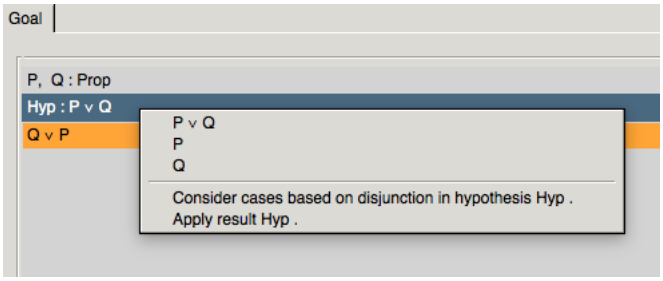
This reflects the fact that we have a new hypothesis tabled Hyp and a new conclusion.

Of course since we have a hypothesis with a disjunction we will use an argument by cases. To do so, type "cases" and press ESC. Choose the following:

```
Consider cases based on disjunction in hypothesis VAR.
```

Press CTRL-r and replace VAR by Hyp. Click run.

Similarly click on the hypothesis Hyp on the right hand side to get:



choose

```
Consider cases based on disjunction in hypothesis Hyp .
```

and click Run.

Notice that there are now two goals:

P Q : Prop
Hyp0 : P
==========

$Q \vee P$

and

P Q : Prop
Hyp1 : Q
==========

$Q \vee P$

corresponding to the two cases to consider. In first goal we will prove the right hand side of the disjunction in the conclusion. To do so, type "right" and press ESC. You get to pick

```
Prove right hand side.
```

and after clicking run you will get the following feedback (note that the second goal stays unchanged)

P Q : Prop
Hyp0 : P
==========

P

Finally you can finish this goal by using the hypothesis Hyp0. To do this you use

```
This follows from assumptions.
```

Note that you have now finished this goal. Repeat the argument for the second goal by using:

```
Prove left hand side.
```

```
This follows from assumptions.
```

to get

no goals

Now type

```
Qed.
```

to save the theorem. It now appears among the proved theorems:

and you can see its proof tree by clicking on draw tree:

Fig. C.1 the tree window

C.0.2 An elementary number theory example

We shall prove the transitivity of divisibility. That is we will prove that

$$\forall a, b, c \in \mathbb{N}, a|b \wedge b|c \Rightarrow a|c.$$

In the process we will introduce definitions and notations.

To start we note that we will be talking about objects of type `nat`. We weill introduce the following definition

```
Definition divides a b := exists x:nat, b = a*x.
```

We hope that the format is quite clear, it resembles the one we used before but uses a few new notions, the operator `:=` which the defines the function `divides` and the quantifier `exists`. Note that we have not explicitly stated that a and b should be natural numbers, Coq will deduce that from the context. We could have been very precise as follows:

```
Definition divides (a b:nat) := exists x:nat, b = a*x.
```

Note that the definition will not get any feedback from Coq. If we want to check that we have correctly defined our notion we can use

```
Check divides.
```

to get

```
Query commands should not be inserted in scripts

divides
: nat -> nat -> Prop
```

or

```
Print divides.
```

to get a more detailed

```
Query commands should not be inserted in scripts

divides = λab : nat, ∃x : nat, b = a * x
: nat -> nat -> Prop
Argument scopes are [nat_scope nat_scope]
```

We will nod describe all this output here but we note the change from `exists` to $\exists$ and the occurrence of λ, a notation for functions.

Next we define a notation for divides

```
Notation " a | b " := (divides a b) (at level 10).
```

Again no feedback from Coq. The definition should be self-evident except for the "(at level 10)" part. We will discuss this elsewhere.

We are now ready to state out theorem

We can state the theorem as (see the corresponding feedback)

```
Theorem refldiv (a b c:nat):
(a | b) ∧ (b | c) -> (a |c).
```

$a, b, c : nat$

$a|b \wedge b|c \to a|c$

but we prefer the version

```
Theorem refldiv: forall a b c, (a | b) ∧ (b | c) -> (a |c).
```

because it is almost identical to the above mathematical statement and it will allow us to show some more tactics. The corresponding feedback is

$\forall a\ b\ c : nat, a|b\ \wedge\ b|c\ \to\ a|c$

Note that Coq has correctly deduced that a, b, c are natural numbers and replaced the quantifier `forall` with $\forall$. Note also that in this form there are no hypotheses.

We fill fix the three variables with the tactics:

```
Fix an arbitrary element a.
Fix an arbitrary element b.
Fix an arbitrary element c.
```

to get

$a, b, c : nat$

$a|b \wedge b|c \to a|c$

As before, in order to prove an implication $A \to B$ we use the tactic

```
Assume A then prove B.
```

More precisely, in this case we have

```
Assume (a | b ∧ b | c ) then prove (a | c).
```

to get

$a, b, c : nat$
$Hyp : a|b \wedge b|c$

$a|c$

Note that hypothesis Hyp is of type $A \wedge B$. We will split this in two hypotheses with:

```
Eliminate the conjuction in hypothesis Hyp.
```

to get

$a, b, c : nat$
$Hyp0 : a|b$
$Hyp1 : b|c$

$a|c$

We seem to have used all the tricks up our selves and so it is time to "unfold" the definitions:

```
Rewrite hypothesis Hyp0 using the definition of divides.
```

$a, b, c : nat$
$Hyp0 : \exists x : nat, b = a * x$
$Hyp1 : b|c$

$a|c$

then

```
Rewrite hypothesis Hyp1 using the definition of divides.
```

$a, b, c : nat$
$Hyp0 : \exists x : nat, b = a * x$
$Hyp1 : \exists x : nat, c = b * x$

$a|c$

and

```
Rewrite goal using the definition of divides.
```

$a, b, c : nat$
$Hyp0 : \exists x : nat, b = a * x$
$Hyp1 : \exists x : nat, c = b * x$

$\exists x : nat, c = a * x$

We now pick x as in the hypothesis Hyp1, that is:

```
Fix x the existentially quantified variable in Hyp1.
```

to get

$a, b, c : nat$
$Hyp0 : \exists x : nat, b = a * x$
$x : nat$
$Hyp1 : c = b * x$

$\exists x0 : nat, c = a * x0$

Note the variable name was changed in the goal but not in Hyp0.

We now use the newly formed Hyp1 as follows:

```
Rewrite the goal using Hyp1.
```

to get

$a, b, c : nat$
$Hyp0 : \exists x : nat, b = a * x$
$x : nat$
$Hyp1 : c = b * x$

$\exists x0 : nat, b * x = a * x0$

Similarly we pick y as in Hyp0 and replace it in the goal

```
 Fix y the existentially quantified variable in Hyp0.
Rewrite the goal using Hyp0.
```

to get

$a, b, cy : nat$
$Hyp0 : b = a * y$
$x : nat$
$Hyp1 : c = b * x$

$\exists x0 : nat, a * y * x = a * x0$

It is now easy to guess that $x0 = y * x$ so we write

```
Prove the existential claim is true for (y*x).
```

to obtain

$a, b, cy : nat$
$Hyp0 : b = a * y$
$x : nat$
$Hyp1 : c = b * x$

$a * y * x = a * (y * x)$

which can be proved by

```
True by arithmetic properties.
```

the total proof is

```
 Definition divides (a b:nat) := exists x:nat, b = a*x.
Notation " a | b " := (divides a b) (at level 10).
Theorem refldiv:forall a b c, (a | b) ∧ (b | c) -> (a |c).
Fix an arbitrary element a.
Fix an arbitrary element b.
Fix an arbitrary element c. Assume (a | b ∧ b | c ) then prove (a | c).
Eliminate the conjuction in hypothesis Hyp.
Rewrite hypothesis Hyp0 using the definition of divides.
Rewrite hypothesis Hyp1 using the definition of divides.
Rewrite goal using the definition of divides.
Fix x the existentially quantified variable in Hyp1.
Rewrite the goal using Hyp1.
Fix y the existentially quantified variable in Hyp0.
Rewrite the goal using Hyp0.
Prove the existential claim is true for (y*x).
True by arithmetic properties.
```

Note that one could use a slightly shorter version of this theorem:

```
Theorem refldiv (a b c:nat): (a | b) ∧ (b | c) -> (a |c).
Rewrite goal using the definition of divides.
Assume ((∃x : nat, b = a ∗ x) ∧ (∃x : nat, c = b ∗ x)) then prove (∃x : nat, c = a ∗ x).
Eliminate the conjuction in hypothesis Hyp.
Fix x the existentially quantified variable in Hyp1.
Rewrite the goal using Hyp1.
Fix y the existentially quantified variable in Hyp0.
Rewrite the goal using Hyp0.
Prove the existential claim is true for (y*x).
True by arithmetic properties.
```

Note also that if you save the latex form of the proof you will obtain the following:

Definition 2 (divides) $divides\,(a\,b : nat) := x : nat,\ b = a * x.$
Theorem 1 (refldiv) $\forall\, a\, b\, c,\ (a \mid b) \wedge (b \mid c) \Rightarrow (a \mid c).$
Proof: In order to show

$$\forall abc : nat, a|b \wedge b|c \Rightarrow a|c$$

we pick an arbitrary

$$a$$

and show

$$\forall bc : nat, a|b \wedge b|c \Rightarrow a|c.$$

In order to show

$$\forall bc : nat, a|b \wedge b|c \Rightarrow a|c$$

we pick an arbitrary

$$b$$

and show

$$\forall c : nat, a|b \wedge b|c \Rightarrow a|c.$$

In order to show

$$\forall c : nat, a|b \wedge b|c \Rightarrow a|c$$

we pick an arbitrary

$$c$$

and show

$$a|b \wedge b|c \Rightarrow a|c.$$

We will assume

$$Hyp : a|b \wedge b|c$$

and show

$$a|c.$$

Since we know

$$Hyp : a|b \wedge b|c$$

we also know

$$Hyp0 : a|bHyp1 : b|c.$$

We use the definition of

$$divides$$

in

$$Hyp0$$

to obtain

$$Hyp0 : \exists x : nat, b = a * x$$

We use the definition of

$$divides$$

in

$$Hyp1$$

to obtain

$$Hyp1 : \exists x : nat, c = b * x$$

Rewriting the definition of

$$divides$$

in our conclusion

$$a|c$$

, we now need to show

$$\exists x : nat, c = a * x.$$

We choose a variable

$$x$$

in

$$Hyp1$$

to obtain

$$x : natHyp1 : c = b * x.$$

We rewrite the goal using

$$Hyp1$$

to obtain

$$\exists x0 : nat, b * x = a * x0.$$

We choose a variable

$$y$$

in

$$Hyp0$$

to obtain

$$a, b, c, y : natHyp0 : b = a * y.$$

We rewrite the goal using

$$Hyp0$$

to obtain

$$\exists x0 : nat, a * y * x = a * x0.$$

We shall prove

$$\exists x0 : nat, a * y * x = a * x0$$

by showing

$$a * y * x = a * (y * x).$$

This follows immediately from arithmetic.
This is done Now

$$a * y * x = a * (y * x)$$

means that

$$\exists x0 : nat, a * y * x = a * x0.$$

We have now proved

$$\exists x0 : nat, a * y * x = a * x0$$

and so

$$\exists x0 : nat, b * x = a * x0$$

follows. and so we have proved

$$\exists x0 : nat, b * x = a * x0.$$

We have now proved

$$\exists x0 : nat, b * x = a * x0$$

and so

$$\exists x0 : nat, c = a * x0$$

follows. and so we have proved

$$\exists x : nat, c = a * x.$$

Therefore we have showed

$$\exists x : nat, c = a * x$$

and so

$$a|c.$$

therefore we have

$$a|c.$$

therefore we have

$$a|c.$$

We are now done with

$$a|c.$$

We have now showed that if

$$Hyp : a|b \wedge b|c$$

then

$$a|c$$

a proof of

$$a|b \wedge b|c \Rightarrow a|c.$$

Since

$$c$$

was arbitrary this shows

$$\forall c : nat, a|b \wedge b|c \Rightarrow a|c.$$

Since

$$b$$

was arbitrary this shows

$$\forall bc : nat, a|b \wedge b|c \Rightarrow a|c.$$

Since

$$a$$

was arbitrary this shows

$$\forall abc : nat, a|b \wedge b|c \Rightarrow a|c.$$

Appendix D
Sets vs types

This is a rather subtle section. It deals with a primitive notion in Automated Theorem Provers, the concept of type. Reading through the book you might have wondered about the occurrence of things like this:

$P : Prop$
$Q : Prop$
$H : P- > Q$

...

The notation seems to be similar for $P : Prop$ and for $Hyp : P- > Q$.

Let us try some experiments. We first define some varibale, P and Q witll be propositions and h "will be' in $P \to Q$"

```
Variable P Prop. Variable Q:Prop.
Variable h:P->Q.
```

Now let us check them,

```
Check P.
Check (P->Q)
```

Nor surprises there, we get $P : Prop$ and "$P \to Q : Prop$" Now try

```
Check h.
```

The result is "$h : P \to Q$".

For much of the book one can look at the notation $a : U$ as a SpatchCoq version of $a \in U$. This is not quite correct. In fact $a : U$ denotes the statement "a is of type U". In particular, the notation

$Hyp : P \to Q$ and $P : Prop$ have the same kind of meaning. The first one means that Hyp is an object of the type $P \to Q$ i.e a witness of a proof of $P -> Q$ while the second means that P is an object of type $Prop$.

The point is that types are primitive objects in Coq (hence in SpatchCoq) and, more importantly,

Types are not Sets!

In Coq (and SpatchCoq) every object has a unique type. For example, 0 cannot represent both the natural number zero and the integer zero. The two objects are different and you need a conversion between them. try for example:

```
Check 0.
```

```
Check 0%Z.
```

Now consider the following:

```
Check Type.
```

you get "Type: Type"!!!!! What does that even mean? It seems that Type is of type Type, surely this must be some sort of Russell paradox.

This is in some sense, the crux of the matter. Modern type theory evolved out of an attempt, by Russell himself, to resolve the paradoxes of Set Theory. This was surpassed in popularity by the ZF Axiomatic Set Theory and waited, half forgotten, for Computer Scientists to rediscover it. The type system of Coq(and SpatchCoq) is based

In fact, the notation "$Type : Type$" is a small notational abuse. It really means that $Type_0 : Type_1$ or, more generally $Type_n : Type_{n+1}$. This is exactly how Russell imagined types, as an infinite series. At the bottom there are sets, that is types like nat or $\mathbb{Z}$ or bool or nat-¿nat. They are themselves types of type Set. The next layer is made of Set itself which of type Type($_0$) is the type Prop. Typ_0 is itself an objrct which is of type $Type_1$ and so on. Note for example:

```
Check Type:Type
```

which produces: $Type : Type : Type$.