Corneliu Hoffman

# Discrete Mathematics with Proof assistants

December 3, 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

SectionGeneral introduction 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 give numerous examples. The inpatient reader can skip quickly to Appendix C to get short descriptions of the tactics, respectively to Appendix D to see two detailed examples. You can also find some other examples of proofs at

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

## 0.1 Warning for teachers

Spatchcoq is developed on top of Coq and, by extension, will suffer from some of the difficulties inherent in using Coq (or any other proof assistant). Perhaps one of the most important one is the fact that proof assistants are based on type theory rather than set theory (see Appendix E for details). In particular every object in Coq has exactly one type and you need a conversion to move form one to the other. We have tried to keep this under the hood as much as possible and to mimic the usual notations and formulations.

This creates certain surprising complications. For example, in the number theory chapter, we have decided to work with natural numbers rather than integers (this is mainly so that induction will work as expected). In general this is ok but, because $\mathbb{N}$ is not a ring, subtraction is a delicate beast. This is because, if $m \leq n$ are two natural numbers then $m - n = 0$. So, for example a statement like $(m - n) + n = m$ is only true if $m \geq n$. For example $(2 - 3) + 3 = 0 + 3 = 3$. For this reason we advise to tilt your examples in the number theory section towards addition rather than subtraction. We will try to remind the reader about such issues as we go along. We will use the following format:

**!Warning!**

Watch for the complications related to minus.

We will also use some separate coloured boxes to separate txt. More neutral remarks are going to look like this:

**Remark**

This is a remark that should help you understand the paragraph better.

The text that is related to Spatchcoq will also be separated from usual text. The text that we input in to Spatchcoq will appear in input boxes like this:

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

The goals that Spatchcoq will return to us will look like this:

Goal

$Hyp : not\ (P \land (not\ Q))$
$Hyp0 : (((not\ P) \land Q) \lor (P \land (not\ Q)))$

$Q$

And the messages that Coq will offer look like this:

error

# Chapter 1

# Basic proof techniques.

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

## 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.

[1] My son Luca showed it to me.

**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.

### Of paradoxes

In Titus 1:12-13, Paul states: "One of themselves, even a prophet of their own, said, The Cretians are alway liars [...] This witness is true". Of course if this witness is true, then, since he is a Cretan, he must be a liar and therefore it cannot be true. This is a paradox due to the Cretan Philosopher Epimenides. Its most condensed logical form is "This statement is a lie" if the statement is true then must be false and if it is false then it is true. There are various similar variations of. This paradox, from Pinnochio's "My nose is growing!" to the I paradox.

" Once upon a time there was a. Small island with extremely radical trespassing laws. Everyone caught trespassing was interrogated and if they weer found truthful they were shot and if they were found lying they were hanged. A logician is caught trespassing and his sole statement during the interrogation is "I will be hanged!". Of course, in the classical logic of this island this person is either shot or hanged. If he is hanged then his statement is truthful and so he should have been shot. If he is shot then his statement is not truthful and so the law in question seems to be paradoxical (inconsistent).

Consider the following statement "This proposition is false"[2]. Is this true or false? If it is true then it must be false, if it is false then it must be true. This is the kind of self referential paradox that lead to Russell type paradoxes (see Subsection 1.2.4 and Chapter 2 as well as Appendix E)

[2] This is a very old story, it first occurred in the form of the liar paradox, in which the philosopher of Crete said "All Cretans are liars."

### *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")$\to$("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 $\neg$(" 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:

$$\frac{\begin{matrix} P \\ \vdots \\ Q \end{matrix}}{P \to Q}$$

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 proofs 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 combining that method with another 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

Goal

$Socrates \to Mortal$

Next we use

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

to get

Goal

$Hyp : Socrates$

$Mortal$

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

Goal

$Hyp : Socrates$
$H : Man$

$Mortal$

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.

*Proof.*

```
Apply result A2.
```

to get

Goal

$Hyp : Socrates$

$Man$

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

Goal

$Hyp : Socrates$

$Socrates$

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
```

This tells you 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

Goal

$PQ : Prop$

$P \wedge Q \rightarrow Q \wedge P$

We then use

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

to get

Goal

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

$Q \wedge P$

We know use

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

To get

Goal

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

$Q \wedge P$

Now we use

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

To get two goals

Goal

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

$Q$

and

Goal

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

$P$

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.

$$\frac{P \vee Q \quad \frac{P}{R} \quad \frac{Q}{R}}{R}$$

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.

Goal

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

$P \vee Q$

Goal

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

$P \vee Q$

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

Perhaps this is a little hard to digest at first but the negation of $P$ is the same thing as $P \to False$. Therefore the inference rules for negation are the same as those for implication. In particular, the negation introduction's logic statement is

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

Therefore th 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 insist here because $P \leftrightarrow Q$ is the same as $P \to Q \land Q \to P$ 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

Goal

$\cdots$

$P$

into

Goal

$\cdots$

$H : \neg P$

$False$

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

Goal

$P : Prop$
$Hyp : not(not P)$

$P$

so we will employ

```
Prove by contradiction.
```

to get

Goal

$P : Prop$
$Hyp : not(not P)$
$H : not P$

$False$

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:

Goal

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

$\neg P \vee Q$

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

Goal

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

$\neg P \vee Q$

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 1.5 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

Goal

$Hyp : onlyone$

$Q$

We now use the tactic that we used for not:

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

to get

Goal

$Hyp : (D1 \land (not\ D2)) \lor ((not\ D1) \land D2)$

$Q$

We now use

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

to get two new goals

Goal

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

$Q$

and

Goal

$Hyp1 : not\ D1 \wedge D2$

$Q$

```
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

Goal

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

$Q$

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

```
Prove by contradiction.
```

to get

Goal

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

$False$

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

Goal

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

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

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

Goal

$Hyp1 : not\ D1 \wedge D2$

$Q$

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:

Goal

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

$Q$

Since Hyp0 is a disjunction we do

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

To get again a case by case analysis.

Goal

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

$Q$

and

Goal

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

$Q$

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.
```

### *1.2.7 On the way to the barbershop*

[3]

In a paper published in 1894 called "A logical paradox" Lewis Carroll presents the following situation:

Two uncles want to go the barbershop. There are three barbers, Allen Brown and Carr. We are told that at least one of them has to be in at all times. Also we know that Allen "ever since he had that fever he?s been so nervous about going out alone, he always takes Brown with him. "

One of the uncles then argues that Carr has to be home. The argument is a proof by contradiction. Suppose that Carr is out. Then if Allen is out then Brown will have to be in since somebody should

[3] I'd like to thank Richard Kaye for showing me this

be in the shop. On the other hand if Allen is out the Brown will have to be out as well on account of Allen's nervousness. Therefore Carr being out generates two contradictory statements "if Allen is out then Brown is in" and "if Allen is out then Brown is out" and so Carr must be in.

This is a remarkable statement. It is reasonably easy to see what is wrong in today's terms (we shall write a careful argument in a moment) but at the end of the 19th century this was serious stumbling block for logicians. In fact in his 1903 book "the Principles of Mathematics", Bertrand Russell writes:

"The principle that false propositions imply all propositions solves Lewis Carroll's logical paradox in Mind, N. S. No. 11 (1894). The assertion made in that paradox is that, if p, q, r be propositions, and q implies r, while p implies that q implies not-r, then p must be false, on the supposed ground that q implies r and q implies not-r are incompatible. But in virtue of our definition of negation, if q be false both these implications will hold: the two together, in fact, whatever proposition r may be, are equivalent to not-q. Thus the only inference warranted by Lewis Carroll?s premisses is that if p be true, q must be false, i.e. that p implies not-q; and this is the conclusion, oddly enough, which common sense would have drawn in the particular case which he discusses".

Indeed the principle that if $p$ is false then $p \to q$ is true, as seen in the truth table of the implications was something that was only formalised by Russell. In fact he states that $p \vee q$ is equivalent to $(p \to q) \to q$ (see for example 1.2.7).

Note that B Russell already suggests the answer to the parable. The two statements only prove that if Carr is out then Allen must be in. Let us prove that ins Spatchcoq.

We will define 3 propositions, Allen, Brown and Carr to mean that the corresponding people are in and state two axioms, $notAllen- > notBrown$ and $Allen \vee Brown \vee Carr$ and we prove that $notCarr \to Allen$.

```
Variables Allen Brown Carr :Prop.
Axiom some: Allen ∨ Brown ∨ Carr.
Axiom AB: not Allen ->not Brown.
Lemma A: not Carr -> Allen.
```

Now we assume that Carr is out and prove that Allen must be in. We use the axiom some to get $Allen \vee (Brown \vee Carr)$.

```
Assume (not Carr) then prove Allen.
Claim (Allen ∨ Brown ∨ Carr).
Apply result some.
```

We get

Goal

$Hyp : notCarr$
$H : Allen \vee Brown \vee Carr$

$Allen$

The next step is to consider the two cases: either Allen is home or one of Brown or Carr must be home. If Allen is home then we are done.

```
Consider cases based on disjunction in hypothesis H .
This follows from assumptions.
```

We now have

Goal

$Hyp : notCarr$
$Hyp1 : Brown \lor Carr$

$Allen$

We will prove this by contradiction:

```
Prove by contradiction.
```

To get

Goal

$Hyp : notCarr$
$Hyp1 : Brown \lor Carr$
$H : notAllen$

$False$

We again consider two cases, either Brown happens or Carr happens. In the first case we use the axiom AB

```
Consider cases based on disjunction in hypothesis Hyp1 .
Apply result AB.
This follows from assumptions. This follows from assumptions.
```

We are left with the last case:

Goal

$Hyp : notCarr$
$Hyp2 : Carr$
$H : notAllen$

$False$

Ans so we can finish by applying modus ponies

```
Apply result (Hyp Hyp2).
```

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

triplenot $\neg(\neg(\neg P)) \leftrightarrow \neg P$

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

Russell $(P \vee Q) \leftrightarrow ((P \to Q) \to Q)$

## 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,$a|b \wedge a|c \to a|(b + c)$.
- 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 $P : U \to Prop$.

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 : $mathbbR^2 \to Prop$.

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 $P : U \to Prop$ 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

Goal

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

...

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

Goal

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

...

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

Goal

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

...

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[4].

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

Goal

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

$Pa$

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

[4] 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 [?]

Goal

$\cdots$

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

Then the tactic Fix an arbitrary element a.

produces the goal

Goal

$a : U$

$P(a)$

.

## 1.4 More Logic puzzles

### *1.4.1 Some examples*

Lewis Carroll is mostly known for the delightfully absurd stories of “Alice’s Adventures in Wonderland” and “Through the Looking-Glass”. The man behind the pseudonym was Charles Lutwidge Dodgson, an Oxford mathematician with interests in linear algebra, geometry and logic. Toward the end of his life he started to write a treaty of logic called “Symbolic Logic”. He had planned the treaty to have three parts but, unfortunately, he only lived to see “PART I ELEMENTARY” published. In the intro to the book he mentions that

> I have a quantity of MS. in hand for Parts II and III, and hope to be able – should life, and health, and opportunity, be granted to me, to publish them in the course of the next few years. Their contents will be as follows:
>
> PART II. ADVANCED. Further investigations in the subjects of Part I. Propositions of other forms (such as “Not-all x are y”). Triliteral and Multiliteral Propositions (such as “All abc are de”). Hypotheticals. Dilemmas. &c. &c.

Part III. TRANSCENDENTAL. Analysis of a Proposition into its Elements. Numerical and Geometrical Problems. The Theory of Inference. The Construction of Problems. And many other Curiosa Logica.

The book is rather technical and some of the methods exposed there have been overtaken by more modern notations. Nevertheless there are many wonderfully quirky logical puzzles (syllogisms). They come in the form of a number of propositions of type $P \to Q$ where $P$ and $Q$ are propositions including quantifiers. Here is an example:

```
All lions are fierce;
Some lions do not drink coffee.
```

    Some fierce creatures do not drink coffee.

The first two are hypotheses (axioms) and the third is the conclusion (lemma). We will formalise the statements in Spatchcoq and, while doing so we will introduce some the concept of notation. This will allow us to write more natural looking text.

We start, as before with defining some variables.We first define a "set of beings" and then a set of predicate on all beings. These predicates are: "Lion", "Fierce" and "Coffee".

```
Variable Beings:Set.
Variables Lion Fierce Coffee:Beings->Prop.
```

Next we introduce notations, note the use of ' ' to bound the words in those notations. The (at level 10) is a mandatory field, the lower the level the closer the brackets. For example if we use (at level 0) for "x is a lion" then it will be printed as "(x) is a lion". If we use a higher level it will be printed as "(x is a lion)".

```
Notation "x 'is' 'a' 'lion'":= (Lion x) (at level 10).
Notation "x 'is' 'fierce'":= (Fierce x) (at level 10).
Notation "x 'drinks' 'coffee'":= (Coffee x) (at level 10).
```

Finally we introduce the two axioms and the Lemma:

```
Axiom LF: forall x, x is a lion -> x is fierce.
Axiom LC: exists x, x is a lion ∧ not (x drinks coffee).
Lemma coffee: exists a, not(a drinks coffee) ∧ (a is fierce).
```

Now in order to prove the lemma we first will find a being that is a lion and does not drink coffee. To do that we use the axiom LC.

```
Claim (exists x, x is a lion ∧ not (x drinks coffee)).
Apply result LC.
```

The result is :

Goal

$(\exists x : Beings, (xisalion) \wedge (not(xdrinkscoffee)))$

$(\exists a : Beings, (not(adrinkscoffee)) \wedge (aisfierce))$

Now we shall fix the element that is lion and does not drink coffee and prove that is fierce and does not drink coffee. The proof is standard.

```
Fix b the existentially quantified variable in H .
Prove the existential claim is true for b.
Eliminate the conjuction in hypothesis H.
Prove the conjunction in the goal by first proving (not (b drinks coffee))
then (b is fierce).
This follows from assumptions.
Apply result LF.
This follows from assumptions.
```

### Exercises

For each of the following deduce wether the third statement follows from the other two and if it does write a formal proof.

1. `No doctors are enthusiastic;`
   `You are enthusiastic.`
       `You are not a doctor.`
2. `Dictionaries are useful;`
   `Useful books are valuable.`
       `Dictionaries are valuable.`
3. `No misers are unselfish;`
   `None but misers save egg-shells.`
       `No unselfish people save egg-shells.`
4. `Some epicures are ungenerous;`
   `All my uncles are generous.`
       `My uncles are not epicures.`
5. `Gold is heavy;`
   `Nothing but gold will silence him.`
       `Nothing light will silence him.`
6. `I saw it in a newspaper.`
   `All newspapers tell lies.`
       `It was a lie.`
7. `Some cravats are not artistic;`
   `I admire anything artistic.`
       `There are some cravats that I do not admire.`
8. `His songs never last an hour;`
   `A song, that lasts an hour, is tedious.`
       `His songs are never tedious.`
9. `Some candles give very little light;`

```
Candles are meant to give light.
    Some things, that are meant to give light, give very little.
```

10. ```
    All, who are anxious to learn, work hard;
    Some of these boys work hard.
        Some of these boys are anxious to learn.
    ```

## 1.5 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[5]. 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).
```

[5] This is a type rather than a set. The interested reader should read E
```

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:

Goal

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

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

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

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

to obtain two new goals:

Goal

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

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

and respectively

Goal

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

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

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

Goal

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

$False$

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

Goal

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

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

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

Goal

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

$False$

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

Goal

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

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

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

Goal

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

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

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.
```

# Chapter 2

# Set Theory

## 2.1 Introduction

The topic of Set Theory is at the very heart of Foundation of Mathematics. It is just about the simples construction in the world, a set is just a bunch of object right? Indeed, before the 20th century a set was rather informally considered to be just any collection of elements. This approach is now described as "naive set theory". The problem with this approach is that it allows for self referential definitions that quickly run into paradoxes. These paradoxes were first introduced by Bertrand Russell, based on the ancient Liar's paradox.

The paradoxes in the story mainly deal with constructions related to the "set of all sets". The commonly accepted solution is due to Ernst Zermelo and Abraham Fraenkel (and it is now called ZF (or ZFC) theory) and proposes an "axiomatic set theory". This means that we only allow sets that are built via a certain collection of axioms. We will describe this briefly bellow a little below but introducing all the subtleties of Axiomatic Set Theory would warrant a separate book. We choose to avoid such details by restricting to subsets of a given "predefined" set U. This avoids issues with any self referential sets and all paradoxes below. The interested reader is referred to citations for ZFC.

As a side remark, Russell's own solution to the problem that arose from "the set of all sets" type paradoxes was quite different from the ZFT approach. He proposed a "type theory" cite for types which introduced a theory of ever increasing Universes so that the objects of a type n are sets of objects of type n-1. His constructions were simplified over the years especially with the development Computer Science. They form the basis of most of the modern type systems, in particular of Thierry Coquand's calculus of constructions (CoC)which is the basis of Coq's (and SpatchCoq's) own construction of types. A more detailed exposure of this in Appendix E.

Our formal approach in SpatchCoq is developed in two steps. The first one (Section 2.2) uses bespoke definitions based on a small modification of the Library Ensembles from Coq. To keep up with the library we will call our formal versions of sets Ensembles (the French word for set). This uses direct definitions and it is very useful to understand the definitions and the connections between Sets and propositional calculus. Nevertheless it gets quite unruly if you need to do longer proof and so the second step is to use the library Ensembles and its inductive definitions.

**!Warning!**

Note that the definitions that we give in Section 2.2 are different from those in Section 2.2.3 . We use these definitions initially because they are slightly better for understanding the ideas behind some of our tactics. See the discussions in Section 2.2.3

Note that C Simpson and J Grim have formalised the Bourbaki version of Axiomatic Set theory in Coq. This is beyond the scope of this text but the interested reader can look at [2] and [1].

### Paradoxes

> Only a Sikh deals in absolutes.
>
> ObiWan.

Consider the popular version of Russell's paradox: In a village there is a barber who shaves those, and only those that do not shave themselves. The question is "Who shaves the barber?".

There are only two possibilities:

1. If he shaves himself then he should not shave himself because he only shaves those that do not shave themselves.
2. If he does not shave himself the he should because he shaves all those that do not shave themselves.

This is a modern version of an ancient paradox. It appears in various guises in the work of Epimenides (600 BC), St Jerome (400Ad), Bhart?hari (400AD), Ath?r al-D?n Mufa??al (10th century AD) and so on. Int its simplest form the paradox deals with the sentence: "This sentence is false". The question is wether this is a Mathematica proposition. Again there are two cases

1. The sentence is true in which case it is false
2. The sentence is false in which case it is not not true which means it is true.

These paradoxes can be modified to show what is wrong with the naive idea of sets. Suppose we define a set as a collection of elements. We also have a predicate $a \in B$ to mean that $a$ is an element of the set $B$. There are two kind of sets.

For most sets the proposition $X \in X$ is obviously false (note that we use $\in$ and not $\subseteq$ see the difference bellow) for example $\{1, 2, 3\} \notin \{1, 2, 3\}$.

However since any collection of objects is a set we can form "sets of sets" and for example if $X$ is the set of all sets that have at least 3 elements must be a set. Now of course $X$ is an infinite set and so it has a lot more than 3 elements. This means in particular that $X \in X$.

We shall then consider $B$ to be the set of all sets that do not contain themselves as an element. In set notation:

$$B = \{A \mid A \notin A\}.$$

There are now again two possibilities:

1. If $B \in B$ then, by the definition of $B$, we get that $B \notin B$, a contradiction.
2. If $B \notin B$ then, again by the definition of $B$, we get that $B \in B$, a contradiction.

This means that the "naive" version of set theory cannot be consistent.

## 2.2 Sets

### *2.2.1 First definitions*

Despite the above mentioned issues, from now on a set will be a collection of elements. In order to do so we shall consider an "universal" set $U$ and for the most part, our sets will be subsets of $U$. More precisely, we a set $U$ which either has some formal construction via ZFT or it is a concretely constructed set such as the set of all people. A set will now be a collection of element of $U$. We normally try to use lower letters for elements and capital letters for sets. We denote by $x \in A$ the fact that $x$ is an element of the set $A$ and by $x \notin A$ the fact that $x$ is not an element of $A$. Note that if $x$ is a fixed element and $A$ is a fixed set then both $x \in A$ and $x \notin A$ are propositions and $\neg(x \in A)$ is the same as $x \notin A$. We will also use the notation $\{a, b, c\}$ for the set whose elements are $a, b$ and $c$. Another common notation is (here $P : U \to Prop$ is a predicate ):

$$A = \{x \in U \mid P(x)\}$$

This means that $A$ is the set of all elements (in U) that satisfy the property $P$. In fact if the universe $U$ can be deduced from the context then we ignore it in the notation. Here are some examples noting that these representations are not unique:

$$\{0, 1, 2\} = \{x \in \mathbb{N} \mid x < 3\} = \{x \in \mathbb{N} \mid x \leq 2\}$$

$$\{x \in \mathbb{R} | x \geq 0\} = \{x \in \mathbb{R} || x| = x\} = \{x \in \mathbb{R} | \exists y \in \mathbb{R}, x = y^2\}$$

Note that while set $A$ is defined by the predicate $P$, the converse is also true. Indeed $P(x)$ is the predicate $x \in A$[1] and so the notation $A = \{x \mid P(x)\}$ is not a special case, it includes all subsets of $U$.

As a consequence, in Spatchcoq we identify the two. More precisely we define a set as a function $A : U \to$ Prop, the function that take the value True if $x \in A$ and False otherwise. For example if $U = \mathbb{N}$ then the set $\{0, 1, 2\}$ is identified with the function $A : \mathbb{N} \to Prop$ so that $A(0) = A(1) = A(2) =$True and $A(x) =$ False otherwise (or by the function $A(x) = x < 3$. Here are the precise definitions. Note that $U$ is a type and not a set, a subtlety that we do discuss here.

[1] In particular, the following two predicates $P : \mathbb{R}-> Prop, P(x) = x > 0$ and $Q : \mathbb{R}-> Prop, P(x) = existy \in \mathbb{R}, x = y^2$ are logically equivalent.

```
Variable U : Type.
Definition Ensemble := U -> Prop.
Definition In (A:Ensemble) (x:U):= A x.
Notation "x ∈ A":= (In A x) (at level 10).
```

Note the definition of In and the notation $\in$.

**!Warning!**

The syntagm (at level 10) is a bit confusing. It establishes the Precedence level of the operator. For example $\to$ has precedence 99 while $\vee$ has precedence 85 and $\wedge$ has precedence 80. This means that if you are careless and write $A \to B \vee C \wedge D$, Spatchcoq will interpret this as $A \to (B \vee (C \wedge D))$.

From now on we will almost forget the function definition and think about sets as collections of elements. To do so however we still need to define the various operations we can do with sets.

The fist concept we discuss is the concept of subset. We say that a set $A$ is a subset of a set $B$ (we write $A \subseteq B$ if any element of $A$ is also an element of $B$. Formally this means

$$A \subseteq B := \forall x \in U, x \in A \to x \in B.$$

Not so surprising the Spatchcoq definition is exactly this:

```
Definition Included (B C:Ensemble) : Prop := forall x:U, x ∈ B -> x ∈ C.
Notation "A ⊆ B":= (Included A B)(at level 10).
```

Note the notation $\subseteq$. It is different from $\in$ and you need to carefully understand the difference. The symbol $\in$ connects elements to the sets that contain them and the symbol $\subseteq$ connects sets. For example

$$\{3\} \subseteq \{1, \{2\}, 3\} \text{ and } 3 \in \{1, \{2\}, 3\}$$
$$2 \notin \{1, \{2\}, 3\} \text{ and } \{2\} \nsubseteq \{1, \{2\}, 3\},$$
$$\{2\} \in \{1, \{2\}, 3\} \text{ and } \{\{2\}\} \subseteq \{1, \{2\}, 3\}$$

We now introduce the tow operation with sets. If $A$ and $B$ are two sets then we can define the union of $A$ and $B$ as the set of all the elements that either belong to $A$ or belong to $B$. That is:

$$A \cup B = \{x | x \in A \vee x \in B\}.$$

The corresponding Spatchcoq definition is a bit stranger, recall that we are defining a new set, that is a predicate $A \cup B : U \to Prop$. The format is perhaps a bit overcharged, you need to use the form (fun x: type $\Rightarrow$ P(x)) that will define a predicate.

```
 Definition Union (B C:Ensemble):Ensemble:=fun x:U => (x∈B) ∨(x∈C).
Notation "A ∪ B":= (Union A B)(at level 8).
```

**Remark**

Note that the precedence here is 8 so $A \subseteq B \cup C$ means $A \subset (B \cup C)$.

**!Warning!**

Note that these definition are definitions of functions, therefore unfolding definitions are a bit weird. For example if your goal looks like

Goal

$\ldots$

$x \in A \cup B$

And you try

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

You will get an unpleasant surprise:

Goal

$\ldots$

$x \in (\lambda x0 : U, x0 \in A \vee x0 \in B))$

The solution is to use a slightly modified form and unfold the actual definition of In the union.

```
Rewrite goal using the definition of (In,Union).
```

To get

Goal

$\ldots$

$((x \in A) \vee (x \in B))$

Similarly the intersection of two sets $A$ and $B$ is the set of the element they have in common. An element is in $A \cap B$ if it is in both $A$ and $B$. This means:

$$A \cap B = \{x | x \in A \wedge x \in B\}.$$

```
 Definition Intersection (B C:Ensemble):Ensemble :=fun x:U=> (x∈B)∧(x∈C).
Notation "A ∩ B" := (Intersection A B) (at level 10).
```

The complement of a set $A$ is the set of element of U that do not belong to $A$. That is:

$$\complement A = \{x | x \notin A\}.$$

Or in spatchcoq

```
Definition Complement (A:Ensemble) : Ensemble := fun x:U =>not In A x.
Notation " ∁A" := (Complement A) (at level 5).
```

**Remark**

note that the precedence is 5 so $\complement A \cup B$ means $(\complement A) \cup B$.

And also $A \setminus B$ is the set of elements in $A$ and not in $B$. This can be also defined as $A \cap (\complement B)$.

```
Definition Setminus (A B :Ensemble) : Ensemble := A∩(∁B).
Notation " A\B" := (Setminus A B) (at level 10).
```

We will make use of the following technical axiom

```
 Axiom Extensionality_Ensembles : forall A B:Ensemble, A ⊆ B ∧ B ⊆ A → A =
B.
```

Finally we introduce the empty set

```
Definition EmptySet:Ensemble:= fun x:U=>False.
Notation ''∅'':=Emptyset
```

### 2.2.2 Operations on Sets

This naive construction of set theory is an ideal ground to try out our proof techniques. The proofs will be very similar to those in predicate calculus. Let us start with transitivity of inclusion:

**Lemma 1.** *trans_incl (A B C:Ensemble U):* $A \subseteq B \wedge B \subseteq C \rightarrow A \subseteq C$.

We will provide three proofs. The first two will be informal and the last will be formal (in Spatchcoq). The first proof is a direct proof and the other two are backward proofs.

*Proof (informal direct proof).*

Recall that $X \subseteq Y$ if and only if $\forall x : U, x \in X \to x \in Y$. In order to prove that $A \subseteq C$, we will pick an element $x$, assume it is in $A$ and prove that it is in fact also in $C$. We will first do a direct proof. Since we know that $A \subseteq B$, it follows by the definition of $\subseteq$ and by *modus ponens* that $x \in B$. Similarly since $B \subseteq$ it also follows from definition and modus ponens that $x \in C$.

*Proof (informal backward proof).*

Recall that $X \subseteq Y$ if and only if $\forall x : U, x \in X \to x \in Y$. In order to prove that $A \subseteq C$, we will pick an element $x$, assume it is in $A$ and prove that it is in fact also in $C$. Since $B \subseteq C$, by the definition of inclusion and by implication elimination, in order to prove that $x \in C$, it suffices to prove that $x \in B$. Moreover since $A \subseteq B$, in order to prove that $x \in B$ it is sufficient to show that $x \in A$. This is however our assumption and so we finish the proof.

*Proof (formal proof).* We start by assuming $A \subseteq B \wedge B \subseteq C$ and aiming to prove $A \subseteq C$.

```
Assume (A ⊆ B ∧ B ⊆ C) then prove (A ⊆ C )
```

. We now split the hypothesis $(A \subseteq B \wedge B \subseteq C)$ into ($A \subseteq B$ and $B \subseteq C$).

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

Next we expand the definition of included everywhere.

```
Rewrite hypothesis Hyp0 using the definition of Included.
Rewrite hypothesis Hyp1 using the definition of Included.
Rewrite goal using the definition of Included.
```

Now we know that

$$Hyp0 : (\forall x : U, (x \in A) \to (x \in B))$$

and

$$Hyp1 : (\forall x : U, (x \in B) \to (x \in C))$$

And we want to show

$$(\forall x : U, (x \in A) \to (x \in C))$$

We pick a random $x \in U$ and try to prove $(x \in A) \to (x \in C)$. To do so assume that $(x \in A)$ and prove $(x \in C)$).

```
Fix an arbitrary element x.
Assume (x ∈ A) then prove (x ∈ C).
```

Now by Hyp1 if we want to prove $x \in C$ it is enough to prove $x \in B$.

```
Apply result Hyp1 .
```

Similarly by Hyp1 if we want to prove $x \in B$ it is enough to prove $x \in A$.

```
Apply result Hyp0 .
```

We already know $x \in A$, finishing the proof.

```
This follows from assumptions.
```

Did that look familiar? Recall the backward proof at page 9. The two proofs are basically identical one you abstract a bit.

The sound proof will involve the empty set.

```
Lemma emptyid(A :Ensemble): A ∪ ∅ = A.
```

*Proof (informal proof).*

We first prove $A \cup \emptyset \subseteq A$.

To do so pick an element $x \in A \cup \emptyset$. There are two cases to consider:

Case 1 $x \in A$ : in this case the conclusion is equal to one of the assumptions so we are done.

Case 2 $x \in \emptyset$ : In this case by the definition of the empty set, $x \in \emptyset$ implies $False$ and so a proof by contradiction finishes this.

We now prove $A \subseteq A \cup \emptyset$. To do so let $x \in A$. By disjunction introduction it means that $x \in A \cup \emptyset$, a proof of the second step.

*Proof (formal proof).*

We first apply the axiom

```
Apply result Extensionality_Ensembles.
```

To get that we need to show

Goal

$A : Ensemble$

$(A \cup \emptyset \subseteq A) \wedge (A \subseteq A \cup \emptyset)$

We now split the goal in two

```
Prove the conjunction in the goal by first proving (A ∪ ∅ ⊆ A) and then (A ⊆
A ∪ ∅).
```

This part of the proof is for $A \cup \emptyset \subseteq A$: We start with the standard "opening moves:"

```
Rewrite goal using the definition of Included.
Fix an arbitrary element x.
```

To get

Goal

$A : Ensemble$
$x : U$

$(x \in (A \cup \emptyset)) \to (x \in A)$

We now apply the standard tactic for implication

```
Assume (x ∈ (A ∪ ∅)) then prove (x ∈ A).
```

And then unfold the definition of the Union.

```
Rewrite hypothesis Hyp using the definition of (In, Union).
```

To obtain

Goal

$A : Ensemble$
$x : U$
$Hyp : (x \in A) \vee (x \in \emptyset)$

$x \in A$

Which requires a case by case analysis.

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

The first case is

Goal

$A : Ensemble$
$x : U$
$Hyp0 : (x \in A)$

$x \in A$

Which follows immediately from assumption:

```
This follows from assumptions.
```

The second case is a bit stranger:

Goal

$A : Ensemble$
$x : U$
$Hyp1 : (x \in \emptyset)$

$x \in A$

And so we unfold the definition of the EmptySet to get

```
Rewrite hypothesis Hyp1 using the definition of (In, EmptySet).
```

Goal

$A : Ensemble$
$x : U$
$Hyp1 : False$

$x \in A$

This is an immediate proof by contradiction.

```
Prove by contradiction.
This follows from assumptions.
```

Next we need to show that $A \subseteq A \cup \emptyset$: We use again the usual moves:

```
 Rewrite goal using the definition of Included.
Fix an arbitrary element x.
```

To get

Goal

$A : Ensemble$
$x : U$

$(x \in A) \to (x \in (A \cup \emptyset))$

We now apply the standard tactic for implication

Assume $(x \in A)$ then prove $(x \in (A \cup \emptyset))$.

And then unfold the definition of the Union:

```
 Rewrite goal using the definition of (In, Union).
```

To get

Goal

$A : Ensemble$
$x : U$
$Hyp : x \in A$

$(x \in A) \vee (x \in \emptyset)$

W will now prove the left hand side of the disjunction:

```
 Prove (x ∈ A) in the disjunction.
This follows from assumptions.
```

Note that the Spatchcoq proof is roughly twice as long as the informal proof. The more complicated the proof the more the formal proof will increase.

Let us write a more involved proof: another such proof:

**Lemma 2 (distr).** $\forall$ *A B C: Ensemble,* $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

I will start with a proof as you would see in a any Mathematics book followed by an Spatchcoq based proof.

*Proof (informal proof).* In order to prove the equality of two sets $X$ and $Y$ we will show that $X \subseteq Y$ and that $Y \subseteq X$.

$\mathbf{A} \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C)$

We take an arbitrary $x \in A \cap (B \cup C)$ and show that $x \in (A \cap B) \cup (A \cap C)$. By definition of the intersection it means that $x \in A$ and $x \in B \cup C$. Since $x \in B \cup C$ there are two cases to consider.

Case 1: $x \in B$, in this case we know that $x$ is an element of $A$ and that $x$ is an element of $B$. This means that $x \in A \cap B$ and so it is also in $x \in (A \cap B) \cup (A \cap C)$.

Case 2: $x \in C$, this case is very similar, we swap $C$ for $B$ in the previous proof, that is we know that $x$ is an element of $A$ and that $x$ is an element of $C$. This means that $x \in A \cap C$ and so it is also in $x \in (A \cap B) \cup (A \cap C)$.

Since $x \in (A \cap B) \cup (A \cap C)$ in both cases, this finishes the proof of the first inclusion.

$(\mathbf{A} \cap \mathbf{B}) \cup (\mathbf{A} \cap \mathbf{C}) \subseteq \mathbf{A} \cap (\mathbf{B} \cup \mathbf{C})$

For this we will take an element $x \in (A \cap B) \cup (A \cap C)$ and show that $x \in A \cap (B \cup C)$. Since $x$ is in a union of two sets we again get to do a case by case analysis.

Case 1: $x \in (A \cap B)$, In this case we know that $x$ is an element of $A$ and $x$ is an element of $B$. Since $x$ is an element of $B$ it follows that $x$ is also an element of $B \cup C$. Now we put together $x \in A$ and $x \in B \cup C$ to get $x \in A \cap (B \cup C)$.

Case 2: $x \in (A \cap C)$, this case is very similar, we swap $C$ for $B$ in the previous proof, that is we know that $x$ is an element of $A$ and $x$ is an element of $C$. Since $x$ is an element of $C$ it follows that $x$ is also an element of $B\ cupC$. Now we put together $x \in A$ and $x \in B \cup C$ to get $x \in A \cap (B \cup C)$.

*Proof (formal proof).* We first pick A, B and C to be sets and we prove $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

```
Fix an arbitrary element A.
Fix an arbitrary element B.
Fix an arbitrary element C.
```

We are now going to employ the Extensionality axiom and so in order to prove the equality it suffices to prove both inclusions.

```
Apply result Extensionality_Ensembles .
```

To get

Goal

$ABC : Ensemble$

$((A \cap (B \cup C)) \subseteq ((A \cap B) \cup (A \cap C))) \wedge (((A \cap B) \cup (A \cap C)) \subseteq (A \cap (B \cup C)))$.

We will now prove the two inclusions separately.

```
Prove the conjunction in the goal by first proving ((A ∩ (B ∪ C)) ⊆ ((A ∩ B) ∪
(A ∩ C))) then (((A ∩ B) ∪ (A ∩ C)) ⊆ (A ∩ (B ∪ C))).
```

The proof of $((A \cap (B \cup C)) \subseteq ((A \cap B) \cup (A \cap C)))$:

We expand the definition of included, pick a random $x \in (A \cap (B \cup C))$ and prove $x \in (A \cap B) \cup (A \cap C))$.

```
Rewrite goal using the definition of Included.
Fix an arbitrary element x.
Assume (x ∈ (A ∩ (B ∪ C))) then prove (x ∈ (A ∩ B) ∪ (A ∩ C)).
```

To get

Goal

$ABC : Ensemble$
$x : U$
$x \in (A \cap (B \cup C))$

$x \in (A \cap B) \cup (A \cap C))$

Next we explicitly expand the definitions of union and intersection in the goal.

```
Rewrite goal using the definition of (In, Union).
Rewrite goal using the definition of (In, Intersection).
```

Note the format, (In, Union) respectively (In, Intersection), this is a technical trick, you could have done this in two steps but it would have looked ugly. The result is that we need to show:

Goal

$x : U$
$x \in (A \cap (B \cup C))$

$(((x \in A) \wedge (x \in B)) \vee ((x \in A) \wedge (x \in C)))$

Similarly expanding the definition in the hypothesis:

```
Rewrite hypothesis Hyp using the definition of (In, Intersection).
Eliminate the conjuction in hypothesis Hyp .
Rewrite hypothesis Hyp1 using the definition of (In, Union).
```

To get the two hypotheses $Hyp0 : (x \in A)$ and $Hyp1 : (x \in B) \vee (x \in C)$. We now consider the two possible cases in Hyp1, that is either $x \in B$ or $x \in C$.

```
 Consider cases based on disjunction in hypothesis Hyp1 .
```

In the first case we know that $x \in A$ and $x \in B$ so we can prove the left hand side of the goal.

```
 Prove left hand side.
Prove the conjunction in the goal by first proving (x ∈ A) then (x ∈ B).
This follows from assumptions.
This follows from assumptions.
```

In the second case we know that $x \in A$ and $x \in C$ so we can prove the right hand side of the goal.

```
 Prove right hand side.
Prove the conjunction in the goal by first proving (x ∈ A) then (x ∈ C).
This follows from assumptions.
This follows from assumptions.
```

The proof of $((A \cap B) \cup (A \cap C))) \subseteq ((A \cap (B \cup C))$:

We follow the same standard procedure:

```
 Rewrite goal using the definition of Included.
Fix an arbitrary element x.
Assume (x ∈ (A ∩ B) ∪ (A ∩ C)) then prove (x ∈ (A ∩ (B ∪ C))) .
```

Now we do know that $x \in (A \cap B) \cup (A \cap C)$ so by using the definition of (In Union) we need to consider the two cases, either $x \in (A \cap B)$ or $x \in (A \cap C)$

```
 Rewrite hypothesis Hyp using the definition of (In, Union).
Consider cases based on disjunction in hypothesis Hyp .
```

If $x \in (A \cap B)$ then using the definition of union we know that $x \in A$ and $x \in B$.

```
Rewrite hypothesis Hyp0 using the definition of (In, Intersection).
Eliminate the conjuction in hypothesis Hyp0 .
```

So we now know that need to show that

$$((x \in A) \wedge (x \in (B \cup C)))$$

This follows using standard introduction rules:

```
 Rewrite goal using the definition of (In, Intersection).
Prove the conjunction in the goal by first proving (x ∈ A) then (x ∈ (B ∪
C)).
This follows from assumptions.
Rewrite goal using the definition of (In, Union).
Prove left hand side.
This follows from assumptions.
```

```
 Rewrite hypothesis Hyp using the definition of (In, Intersection).
Rewrite goal using the definition of (In, Intersection).
```

The other case is similar.

```
Rewrite hypothesis Hyp0 using the definition of (In, Intersection).
Eliminate the conjuction in hypothesis Hyp0 . Rewrite goal using the
definition of (In, Intersection).
Prove the conjunction in the goal by first proving (x ∈ A) then (x ∈
(B ∪ C)).
This follows from assumptions.
Rewrite goal using the definition of (In, Union).
Prove right hand side.
This follows from assumptions.
```

Note again that this proof is not so much different from the proof of distributivity rule for and and or.

**Exercises**

**distr** ∀ A B C: Ensemble, $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

**intuni** ∀ A B:Ensemble, $A \cap (A \cup B) = A$.

**intuni** ∀ A B:Ensemble, $A \cup (A \cap B) = A$.

**uniitro** ∀ A B:Ensemble, $A \cap (A \cup B) = A$.

**incl** ∀ A B:Ensemble, $A \subset B \leftrightarrow A = A \cup B$.

**union** ∀ A B:Ensemble, $A \subset B \leftrightarrow B = A \cap B$.

**diff** ∀ A B:Ensemble, $A \subset B \leftrightarrow A \cup \complement B = \emptyset$.

### *2.2.3 Sets in Coq*

**!Warning!**

The constructions of union, intersection, complement and Empty Set form Section 2.2 are easy enough to deal with but if the theorems are complicated they begin to be slightly unruly. We used them to exemplify the use of the "rewrite using the definition" tactics but from now on we abandon them in favour of the built in definitions in Coq.

We do start by importing the Ensemble package.

```
Require Import Ensembles.
```

Note that we can now also introduce the notations:

```
 Notation "x ∈A":= (In _ A x) (at level 10).
Notation "A ⊆ B":= (Included _ A B)(at level 10).
Notation "A ∪ B":= (Union _ A B)(at level 1).
Notation "A∩ B" := (Intersection _ A B) (at level 10).
Notation "∁ A" := (Complement _ A) (at level 10).
Notation " A \ B" := (Setminus _ A B) (at level 10).
Notation "∅ ":= (Empty_set _).
```

Note that in Section 2.2 the definition of union was:

```
Definition Union (B C:Ensemble):Ensemble:=fun x:U => (x∈B) ∨(x∈C).
```

The package Ensembles in Coq defines the union inductively. Therefore if you look at

```
Print Union.
```

You get

```
 Inductive Union (B C:Ensemble) : Ensemble :=
| Union_introl : forall x:U, In B x -> In (Union B C) x
| Union_intror : forall x:U, In C x -> In (Union B C) x.
```

Therefore, you get two different theorems (Union_introl, Union_intro4) that allow you to "introduce a union" if you want to prove $x \in A \cup B$ you can either prove $x \in A$ and use Union_introl or prove $x \in B$ and use Union_intror.

For example if you want to prove

```
Lemma distr (A B:Ensemble U): A ⊆ (A ∪ B).
```

We do the same few standard things

```
Rewrite goal using the definition of Included.
Fix an arbitrary element x.
Assume (x ∈ A) then prove (x ∈ (A ∪ B)).
```

To get

Goal

$U : Type$
$A, B : Ensemble\ U$
$x : U$
$Hyp : x \in A$

$x \in (A \cup B)$

At this point however we can do

```
Apply result Union_introl.
```

To get

Goal

$U : Type$
$A, B : Ensemble\ U$
$x : U$
$Hyp : x \in A$

$x \in A$

And finish with:

```
This follows from assumptions.
```

At the same time, if you want to use a hypothesis of type $Hyp : x \in A \cup B$, you will use at the same cases tactic that you use for disjunction.

For example, suppose you want to prove

```
Lemma a (U:Type) ( A B:Ensemble U): (A ∪ B)⊆(B ∪ A).
```

We do the usual

```
Rewrite goal using the definition of Included.
Fix an arbitrary element x.
Assume (x ∈ (A ∪ B)) then prove (x ∈ (B ∪ A)).
```

At which point your goal is:

Goal

$U : Type$
$A, B : EnsembleU$
$x : U$
$Hyp : x \in A \cup B$

$x \in B \cup A$

We now apply the tactic:

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

To get two cases:

Goal

$U : Type$
$A, B : EnsembleU$
$x : U$
$Hyp : x \in A$

$x \in B \cup A$

and

Goal

$U : Type$
$A, B : EnsembleU$
$x : U$
$Hyp : x \in B$

$x \in B \cup A$

We can now finish the proofs of the two cases by using:

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

Respectively

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

.

Similarly, the command

```
Print Intersection
```

. will give us:

Inductive Intersection (U : Type) (B C : Ensemble U) : Ensemble U := Intersection_intro : $\forall x : U, x \in B \to x \in C \to x \in (B \cap C)$

Which means that if you want to prove a goal of type $x \in A \cup B$, you can use the tactic:

```
Apply resuly Intersection_intro.
```

In order to use a hypothesis of the type $Hyp : x \in A \cup B$, you can use the tactic:

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

For example, in order to prove the lemma:

```
Lemma b (U:Type) ( A B:Ensemble U): (AcapB) ⊆ (B ∩ A).
```

```
Rewrite goal using the definition of Included.
Fix an arbitrary element x.
Assume (x ∈ (A ∩ B)) then prove (x ∈ (B ∩ A)).
Eliminate the conjunction in hypothesis Hyp.
Apply result Intersection_intro.
This follows from assumptions.
This follows from assumptions.
```

Perhaps more interesting is the definition of the empty set. As before you can try

```
Print Empty_set.
```

To get

```
Inductive Empty_set (U : Type) : Ensemble U :=
```

That is the empty set has an empty intro constructor. You cannot really prove $x \in \emptyset$. Nevertheless if you look for it

```
Search Empty_set.
```

You get (a few similar results)among other things)

Empty_set_ind: $\forall (U : Type)(P : U \to Prop)(u : U), \emptyset u \to Pu$

Which means that if you know x is an element of the empty set then you can prove anything. For example, to prove that $A \cup \emptyset \subseteq A$

```
Lemma emptyid (U:Type) (A :Ensemble U): A ∪ ∅ ⊆ A.
```

We do the following:

```
Rewrite goal using the definition of Included.
Fix an arbitrary element x.
Assume (x ∈ A ∪ ∅) then prove (x ∈ A).
Consider cases based on disjunction in hypothesis Hyp.
This follows from assumptions.
Apply result Empty_set_ind.
This follows from assumptions.
```

De definition of Complement and Setminus are identical to those of Sectionsec:sets.

**!Warning!**

Finally note that in the Ensembles package the axiom Extensionality_Ensembles is defined using Same_set:

$$\forall (U : Type)(AB : EnsembleU), Same\_setUAB \rightarrow A = B$$

were

$$Same\_setUBC : B \subseteq C \wedge C \subseteq B$$

An d so usually equality proofs in sets start by doing

```
Apply result Extensionality_Ensembles.
Rewrite goal using the definition of Same_set.
```

Here is a quick cheatsheet for set theory:

| Hypothesis contains | Tactic |
|---|---|
| $x \in \emptyset$ | Apply result Empty_Set_ind. |
| $x \in A \cup B$ | Consider cases based on disjunction in hypothesis Hyp. |
| $x \in A \cap B$ | Eliminate the conjuction in hypothesis Hyp. |

| Goal contains | Tactic |
|---|---|
| $x \in A \cup B$ | Apply result Union_introl.<br>Apply result Union_introl. |
| $x \in A \cap B$ | Apply result Intersection_intro. |

**Exercises**

Do the same exercises as in Section 2.2 with the inductive constructions for sets.

**distr** ∀ A B C: Ensemble, $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

**intuni** ∀ A B:Ensemble, $A \cap (A \cup B) = A$.

**intuni** ∀ A B:Ensemble, $A \cup (A \cap B) = A$.

**uniitro** ∀ A B:Ensemble, $A \cap (A \cup B) = A$.

**incl** ∀ A B:Ensemble, $A \subset B \leftrightarrow A = A \cup B$.

**union** ∀ A B:Ensemble, $A \subset B \leftrightarrow B = A \cap B$.

**diff** ∀ A B:Ensemble, $A \subset B \leftrightarrow A \cup \complement B = \emptyset$.

## Chapter 3

# Number Theory

> Mathematics is the queen of the sciences and number theory is the queen of mathematics.
>
> C.F. Gauss.

### 3.1 Natural numbers, operations, order

Number Theory is one of the oldest branches of Mathematics. We have some evidence of prime numbers are as old as the Ishango bones, 22000 years ago. A formal definition of natural numbers however was only done rather recently. Surprisingly, it is not different to the intuition that you get all natural numbers by keep adding ones to 0 (or 1). Our approach will be quite formal, following Peano and some of the questions will feel a bit strange.

We will definitely the natural numbers as a set $\mathbb{N}$ containing an element $0$[1] and with a map (called the successor map) $S : \mathbb{N} \to \mathbb{N}$ so that

1. $\forall x \in \mathbb{N}, S(x) \neq 0$. (0 is not the successor of any number)
2. $\forall x, y \in \mathbb{N}, S(x) = S(y) \to x = y$. (S is injective)
3. (induction axiom.) If $A \subseteq \mathbb{N}$ is such that $o \in A$ and $S(A) \subseteq A$ then $A = \mathbb{N}$.

Note that a few other properties are implicit. For example $S$ being a function will tell you that $x = y \to Sx = Sy$[2].

We also employ syntactic sugar in denoting $1 := S\ 0, 2 := S\ (S\ 0), 3 := S\ (S\ (S\ 0)), \cdots$.

[1] Note that Peano himself considered 1 instead of zero as the first natural number. That might have been historically accurate (humankind too a long time to discover zero) but created a lot of inconsistency. We made the more modern choice here, partly influenced by Coq.

[2] this is called eq_S in Coq/spatchcoq.

Axiom 3 allows us to prove things by induction. More precisely if we are trying to prove a statement of the type $\forall x \in \mathbb{N}, P(x)$ and we consider (as in Section refch:settheory) the set $A \subseteq \mathbb{N}$ given by $P$ then the statement is equivalent to showing that $A = \mathbb{N}$ and, by Axiom 3, it is enough to show that $0 \in p$ (or in other words P(0) holds) and then if $k \in A$ (I.e. P(k) holds) then $S(k) \in A$ (i.e. P(S(k)) holds). This is the usual way of doing induction

**First Step** Prove P(0)

**Induction step** Assume P(k) holds and show P(k+1) holds.

Before giving some examples, we need to define some more concepts. The first thing to define is the concept of addition. Addition in $\mathbb{N}$ is defined inductively, as follows

1. $0 + m = m$,[3]
2. $S(n) + m = S(n + m)$.[4]

In Coq the corresponding definition looks a bit strange but quite readable (aside from the "fix" which represents the fact that the definition is inductive):

```
Nat.add =
fix add (n m : nat) {struct n} : nat :=
  match n with
  | 0 => m
  | S p => S (add p m)
  end
```

**!Warning!**

Note that this definition is not a set in the sense of Section 2. It is an inductively defined type. We will allow ourself a bit of liability here, we view the set nat as the universe for much of the section. All subsets of natural numbers will be subsets of $\mathbb{N}$ as before.

This definition has introduced two properties that tell us how to add integers, we will use those very often:

**plus_O_n:** ∀ n : nat, 0 + n = n

and

**plus_Sn_m:** ∀ n m : nat, S n + m = S (n + m)

This tells us how to add two numbers. If we want to compute $n + m$ we first look at n. If it is zero, we use the first method and get m as the sum. If n is not zero then it can be obtained as the successor of somebody else say $n = S\ p$. To obtain the result we now add p and m and then take the successor of the result.

If you forget the name of the statements perhaps it is time you learn how to search for them. First try

[3] this is called plus_O_n in Coq/spatchcoq

[4] this is called plus_Sn_m in Coq/spatchcoq

```
Search (0+_=_).
```

That is search for statements that look like 0+? =?. The result will be longer than you want:

```
Nat.add_0_l: forall n : nat, 0 + n = n

plus_O_n: forall n : nat, 0 + n = n
```

Similarly if you need to find things that look like $S$ ?+? =? you use “”

```
Search (S _+_).
```

To obtain“”

```
Nat.add_1_l: forall n : nat, 1 + n = S n

Nat.add_succ_comm: forall n m : nat, S n + m = n + S m

Nat.add_succ_l: forall n m : nat, S n + m = S (n + m)

plus_Sn_m: forall n m : nat, S n + m = S (n + m)
========================
plus_Snm_nSm: forall n m : nat, S n + m = n + S m
```

Without further ado let us prove the thing you always wanted to prove:

```
Lemma first: 1+1=2.
```

Note that if we wrote the lemma as

```
Lemma first:1+1= S ( S 0).
```

The result would have been the same.

*Proof (informal).* The proof is indeed very simple once we understand what we need to show. We first note that 1+1 means in fact that $(S0)+1$. The statement plus_Sn_m, tells us how to add those two, that is we need to add 0 and 1 and take the successor. That is $1+1 = S(0+1)$. Now plus_O_n tells us that $0+1 = 1$ and so becomes $1+1 = S\ (0+1) = S\ 1 = S\ (S\ 0) = 2$ which finishes the proof.

To follow the proof in Spatchcoq do:

```
 Lemma first:1+1=2.
Rewrite the goal using plus_Sn_m.
Rewrite the goal using plus_O_n.
This follows from reflexivity.
Qed.
```

Note that if we had to do such a complicated proof each time we would never achieve anything. Note the following :

```
 Lemma b: 3+4=7.
Rewrite goal using the definition of Nat.add.
This follows from reflexivity.
```

or even

```
 Lemma b: 3+4=7.
This is trivial.
```

We will now prove the first theorem that is based by induction, the associativity of addition:

```
Lemma add_assoc (n m p:nat ): n+(m+p ) = (n+m)+p.
```

To prove that we will use induction on n. Therefore we need to show

**First Step** $0+(m+p)=(0+m)+p$ Here we first use plus_O_n to show that the left hand side equals $m+p$ and then again to show the right hand side also equals $m+p$.

**Induction Step** Assume $n+(m+p)=(n+m)+p$ and prove $S\ n+(m+p)=(S\ n+m)+p$.

We now use plus_Sn_m to show the left hand side can be written as $S\ (n+(m+p))$ then use the induction hypothesis to show you can write than as $S\ ((n+m)+p)$ Using plus_Sn_m we rewrite the latter as $S\ ((n+m))+p$ and using it again we can write it as $(S\ n+m)+p$ which equals the right hand side.

The complete proof is bellow[5].

[5] This is a standard lemma, we will use it henceforth as Nat.add_assoc

```
 Lemma add_assoc (n m p:nat ): n+(m+p ) = (n+m)+p.
Apply induction on n.
Rewrite the goal using plus_O_n.
Rewrite the goal using plus_O_n.
This follows from reflexivity.
Rewrite the goal using plus_Sn_m.
Rewrite the goal using IHn .
Rewrite the goal using plus_Sn_m.
Rewrite the goal using plus_Sn_m.
This follows from reflexivity.
```

We also note the following lemma for further use[6]. Note the use of the new tactic Replace (n + 1) by (S n) in the goal, in this tactic it is assumed that one of the statements (a=b) or (b=a) are among your hypotheses. It can be used in a more general form but it will hurt latex export.

```
Lemma add_1_r: ∀ n : nat, n + 1 = S n.
Fix an arbitrary element n.
Apply induction on n.
Rewrite the goal using plus_O_n.
This follows from reflexivity.
Rewrite the goal using plus_Sn_m.
Replace (n + 1) by (S n) in the goal.
This follows from reflexivity.
Qed.
```

We are nor ready to prove commutativity[7].:

[6] This is a standard Lemma, we will use it as Nat.add_1_r

[7] This is a standard Lemma, we will use it as Nat.add_comm

```
 Lemma comm(n m:nat):n+m=m+n.
Apply induction on n.
Rewrite the goal using plus_O_n.
Apply induction on m.
Rewrite the goal using plus_O_n.
This follows from reflexivity.
Rewrite the goal using plus_Sn_m.
Replace (m + 0) by (m) in the goal.
This follows from reflexivity.
Rewrite the goal using plus_Sn_m.
Replace (n + m) by (m + n) in the goal.
Rewrite the goal using Nat.add_1_r.
Rewrite the goal using Nat.add_assoc.
Rewrite the goal using Nat.add_1_r.
This follows from reflexivity.
```

Hopefully by now you are starting to get used to such exercises. Any of the Lemmas above could have be proved in one line using the tactic True by arithmetic properties. We proffered to use the longer way in order to show you some easy examples of induction.

The next thing we will define is the subtraction. If you try

```
Print Nat.sub
```

in spatchcoq you will se the construction:

```
Nat.sub =
fix sub (n m : nat) {struct n} : nat :=
  match n with
  | 0 => n
  | S k => match m with
           | 0 => n
           | S l => sub k l
           end
```

We can slowly decipher the above. The first rule says that $0 - m = 0$ regardless on what $m$ is. This is perhaps surprising but a moment thought will suffice to see that there is no other choice since we do not have negative numbers. The second rule said that if $n = Sk$ then we look at $m$. If $m = 0$ we get the familiar $n - 0 = n$. Otherwise $m = Sl$ and so we can compute $n - m = k - l$. Note the following useful properties of subtraction:

Nat.sub_0_l: ∀ n : nat, 0 - n = 0

Nat.sub_0_r: ∀ n : nat, n - 0 = n

Nat.sub_succ: ∀ n m : nat, S n - S m = n - m

minus_plus: ∀ n m : nat, n + m - n = m

minus_plus_simpl_l_reverse: ∀ n m p : nat, n - m = p + n - (p + m)

Here is a proof of the third based on the first two:

```
 Lemma minus_plus: ∀ n m : nat, n + m - n = m.
Fix an arbitrary element n.
Fix an arbitrary element m.
Apply induction on n.
Rewrite the goal using plus_O_n.
Rewrite the goal using Nat.sub_0_r.
This follows from reflexivity.
Rewrite the goal using plus_Sn_m.
Rewrite the goal using Nat.add_comm.
Rewrite the goal using Nat.sub_succ.
Rewrite the goal using Nat.add_comm.
Apply result IHn .
```

Note also that the definition of subtraction limits the use of the tactic “True by arithmetic properties.”. Indeed almost no step from the Lemma above can be solved by that tactic.

We now define the multiplication. We define it inductively as before, we have two rules:

Nat.mul_0_l: ∀ n : nat, 0 * n = 0

and

Nat.mul_succ_l: ∀ n m : nat, S n * m = n * m + m

Indeed the definition of multiplication has a first step that tells you that $0*n = 0$ and another that says that $(n+1)*m = n*m+m$.

Let us prove that $n*0 = 0$ as well.

To do so we will use induction on n. If $n = 0$ we need to show $0*0 = 0$ and that follows from Nat.mul_0_l. The induction step assumes that $n*0 = 0$ and we will need to show that $S\ n*0 = 0$. Now we do know from Nat.mul_succ_l: that $S\ n*0 = n*0+0$ and so using the induction hypothesis we only need to show that $0+0 = 0$.

```
 Lemma a(n:nat): n*0=0.
Apply induction on n. Rewrite the goal using Nat.mul_0_l.
This follows from reflexivity.
Rewrite the goal using Nat.mul_succ_l.
Rewrite the goal using IHn .
This follows from reflexivity.
```

We will also have the right hand property of Successor:

Nat.mul_succ_r: ∀ n m : nat, n * S m = n * m + n

We leave this to the reader and use it to prove commutativity of multiplication.

```
 Lemma mul_comm ( n m : nat): n * m = m * n.
Apply induction on n.
Rewrite the goal using Nat.mul_0_l.
Rewrite the goal using Nat.mul_0_r.
This follows from reflexivity.
Rewrite the goal using Nat.mul_succ_l.
Replace (n * m) by (m * n) in the goal.
Rewrite the goal using Nat.mul|_succ_r.
This follows from reflexivity.
```

Finally we will define the order relation on $\mathbb{N}$. We will define "$\leq$" the less than or equal relation first. We define this also inductively, that is we ask that $n \leq n$ and that if $n \leq m$ then $n \leq S\ m$.

Inductive le (n : nat) : nat $\rightarrow$ Prop := $le\_n : n \leq n | le\_S : \forall m : nat, n \leq m \rightarrow n \leq Sm$

Let us prove the antisymmetry of le.

**Lemma 3 (antisym).**

$$\forall ab \in \mathbb{N}, a \leq b \wedge b \leq a \rightarrow a = b.$$

And of course the relation $<$ is defined as $n < m$ if $Sn \leq m$.

We have an equivalent definition of the $\leq$ relation

Nat.sub_0_le: $\forall$ n m : nat, n - m = 0 $\leftrightarrow$ n $\leq$ m.

## 3.2 More induction

The previous section gave some easy examples of induction. This section will have some more involved examples. We will start with some strange examples. We will have various definitions of even and odd numbers and prove that a number is either even or odd.

The first definition will be

**Definition 1.** A natural number n is even if there its a natural number $k$ with $n = 2k$.

```
Definition Nat.Even (n:nat):= exists k, n=2*k
Definition Nat.odd (n:nat):= exists k, n=2*k+1
```

```
 Lemma even_or_odd(n:nat): Nat.Even n ∨ Nat.odd n.
```

*Proof (informal).* We will prove this by induction on $n$. The first step is quit easy, we need to prove Nat.Even 0 $\vee$ Nat.odd 0. We in fact prove 0 is even, by noting that $0 = 2 * 0$.

The induction step assumes Nat.Even n $\vee$ Nat.odd n and tries to prove Nat.Even ( S n) $\vee$ Nat.odd (S n). To do so we consider the two cases in the

To show this we consider the two cases in the induction hypothesis:

Case 1 Nat.Even n

This means that there exists $k$ so that $n = 2k$. We then note that $Sn = 2k + 1$ and so S n is odd and Nat.Even (S n) $\vee$ Nat.odd (S n) holds.

Case 2 Nat.odd n

In this case there exists a $k$ so that $n = 2k + 1$ and so $Sn = 2k + 2 = 2(k + 1)$ and so S n is even and Nat.Even (S n) $\vee$ Nat.odd (S n) holds.

The formal proof is slightly longer but follows the same structure.

```
 Apply induction on n.
Prove left hand side.
Rewrite goal using the definition of Nat.Even.
Prove the existential claim is true for 0.
True by arithmetic properties.
Consider cases based on disjunction in hypothesis IHn .
Rewrite hypothesis H using the definition of Nat.Even.
Prove right hand side.
Rewrite goal using the definition of Nat.odd.
Fix k the existentially quantified variable in H .
Prove the existential claim is true for k.
Replace (2 * k) by (n) in the goal.
This is trivial.
Rewrite hypothesis H using the definition of Nat.odd.
Fix k the existentially quantified variable in H .
Prove (Nat.Even (S n)) in the disjunction.
Rewrite goal using the definition of Nat.Even.
Prove the existential claim is true for (k+1).
Replace (n) by ((2 * k) + 1) in the goal.
True by arithmetic properties.
```

We now arrive at the first standard induction proof you probably see in a textbook:

**Lemma 4.**

$$\forall n \in \mathbb{N}, \sum_{I=0}^{n} i = \frac{n(n+1)}{2}$$

In order to prove this we will need first to define the above sum. This is a little strange in Spatchcoq, we will define it as:

```
Fixpoint sum (n : nat) : nat :=
match n with
| 0 => 0
| S p => n + (sum p)
end.
```

Which allows out to prove things by induction. Since dividing by two presents complications we shall in fact prove:

**Lemma 5.**

$$\forall n \in \mathbb{N}, 2\sum_{I=0}^{n} i = n(n+1)$$

Then we prove the Lemma:

*Proof (informal).* We will prove this by induction on $n$. Recall that the statement we shall prove is

$$P(n) : 2\sum_{I=0}^{n} i = n(n+1)$$

**First Step**

We need to prove

$$P(0) = 2\sum_{i=0}^{0} i = 0 * (0+1)$$

Which is quite trivial.

**Induction step**

We now assume that

$$P(n) : 2\sum_{i=0}^{n} i = n(n+1)$$

Holds and we prove

$$P(n+1) : 2\sum_{i=0}^{n+1} i = (n+1)(n+1+1).$$

Now note that, by definition and then by distributivity of multiplication,

$$2\sum_{I=0}^{n+1} i = 2((n+1) + \sum_{i=0}^{n}) = 2(n+1) + 2\sum_{i=0}^{n}$$

Using the induction hypothesis we get that

$$2(n+1)+2\sum_{i=0}^{n} = 2(n+1)+n(n+1) = (n+1(n+2)$$

Which is what we needed to prove.

The formal proof is very similar. Note the fact that we use Natṁul_add_distr_l which states

Natṁul_add_distr_l : ∀ n m p : nat, n * (m + p) = n * m + n * p

and which can be found using the pattern (_*(_+_)=_). We also use the tactic "True by arithmetic properties." To finish the computations.

```
 Lemma a (n:nat): 2*(sum n) = n*(n+1).
Apply induction on n.
This is trivial.
Rewrite goal using the definition of sum.
Rewrite the goal using Natmul_add_distr_l.
Replace (2 * (sum n)) by (n * (n + 1)) in the goal.
Replace (S n) by (n+1) in the goal.
True by arithmetic properties.
This is trivial.
```

Here is another example:

**Lemma 6.**

$$\forall n \in \mathbb{N}, \sum_{i=0}^{n} 2^i = 2^{(n+1)} - 1$$

This is again a slightly complicated problem because of the minus in the story. At one point in the proof we will need to use the result

Nat.add_sub_assoc : $\forall nmp : nat, p \leq m \rightarrow n + (m - p) = n + m - p$

For that reason we will start by proving the following Lemma (which is another example of induction)

**Lemma 7 (onepow).**

$$\forall a\ n \in \mathbb{N}, a \neq 0 \rightarrow 1 \leq a\hat{\ }n$$

Recall that the definition of exponential in the natural numbers is also inductive:

```
Nat.pow = fix pow (n m : nat) struct m : nat :=
match m with
| 0 => 1
| S m0 => n * pow n m0
end
```

That is we know that $a\hat{}0 = 1$ and that $a\hat{}Sn = a * a\hat{}n$.

So in order to prove Lemma 7 we again need to use induction on $n$.

*Proof (informal).* Let us fix $a$ and assume $a \neq 0$.The statement we need to show is

$$P(n) : 1 \leq a\hat{}n.$$

**Step one**

The case $n = 0$ is rather easy

$$P(0) : 1 \leq a\hat{}0$$

and since, by definition $a\hat{}0 = 1$ this is equivalent to showing $1 \leq 1$ which is trivial.

**Induction Step**

Let us assume that

$$P(n) : 1 \leq a\hat{}n$$

holds and prove

$$P(n+1) : 1 \leq a\hat{}(Sn).$$

By definition, $a\hat{}(Sn) = a * a\hat{}n$ and we now employ the result

Nat.pow_le_mono_r : $\forall abc : nat, a \neq 0 \rightarrow b \leq c- > a\hat{}b \leq a\hat{}c$

In order to prove that $a\hat{}n \leq a\hat{}Sn$ an then use transitivity of $\leq$.

The formal proof is not much more difficult.

```
Lemma onepow(a n:nat): (a <> 0)-> 1 <= a^n.
Assume (a ≠ 0) then prove (1 ≤ a^n).
Apply induction on n.
This is trivial.
Rewrite goal using the definition of Nat.pow.
Claim (a^n ≤ a^(Sn)).
Apply result Nat.pow_le_mono_r.
This follows from assumptions.
This is trivial.
Apply result (Nat.le_trans  1(a^n)(a^(Sn))).
This follows from assumptions.
This follows from assumptions.
Qed.
```

Do not forget the Qed so we can use the Lemma.

We are now ready for the other lemma. First define the other sum:

```
Fixpoint sum2 (n : nat) : nat :=
match n with
| 0 => 1
| S p => 2^n + (sum2 p)
end.
```

And state the Lemma.

```
Lemma s2 (n:nat): sum2 n = 2^(S n)-1
```

We will not give an informal proof but rather a annotated formal proof. We start by using induction.

```
Apply induction on n.
```

The first step is of course

Goal

$(sum2\ 0 = (2\hat{}1) - 1)$

Which is trivial

```
This is trivial.
```

We now need to look at the indiction step

Goal

$n : nat$
$IHn : sum2\ n = (2\hat{}(Sn)) - 1$

$sum2S\ n = (2\hat{}(S(S\ n))) - 1$

We use the definition of sum2 to modify the goal:

```
Rewrite goal using the definition of sum2.
```

Goal

$n : nat$
$IHn : sum2\ n = (2\hat{}(S\ n)) - 1$

$((2\hat{}(S\ n)) + (sum2\ n) = (2\hat{}(S(S\ n))) - 1)$

We can now use the induction hypothesis IHn.

```
Replace (sum2 n) by ((2^(S  n )) - 1) in the goal.
```

To get

Goal

$n : nat$
$IHn : sum2\ n = (2\hat{}(S\ n)) - 1$

$((2\hat{}(Sn)) + ((2\hat{}(Sn)) - 1) = (2\hat{}(S(Sn))) - 1)$

We now manipulate the right hand side of the equality by using

Nat.pow_succ_r': $\forall ab : nat, a\hat{}Sb = a * a\hat{}b$

```
Rewrite the goal using (Nat.pow_succ_r' 2 (S n)).
```

to obtain

Goal

$n : nat$
$IHn : sum2\ n = (2\hat{}(S\ n)) - 1$

$((2\hat{}(Sn)) + ((2\hat{}(Sn)) - 1) = (2 * (2\hat{}(Sn))) - 1)$

Toi clean up a bit we will use

```
Denote (2^( S n)) by x.
```

The result is

Goal

$n : nat$
$IHn : sum2\ n = (2\hat{}(S\ n)) - 1$

$x + (x - 1) = (2 * x) - 1$

Which seems that it should be immediate but we run into a bit of troubles because of the minus. We rewrite using

```
Rewrite the goal using Nat.add_sub_assoc.
```

This changes the goal to

Goal

$n : nat$
$IHn : sum2\ n = (2\hat{}(S\ n)) - 1$

$(x + x) - 1 = (2 * x) - 1$

But also creates a new goal:

Goal

$n : nat$
$IHn : sum2\ n = (2\hat{}(S\ n)) - 1$

$1 \leq x$

The first goal is easily solvedd by

```
Replace (x+x) by (2*x) in the goal. This follows from reflexivity. This is
trivial.
```

And we are left with

Goal

$n : nat$
$IHn : sum2\ n = (2\hat{}(S\ n)) - 1$

$1 \leq x$

We remember what $x$ is and apply onepow:

```
Replace (x) by (2^(S n)) in the goal. Rewrite the goal using onepow.
```

And we are left to show that $1 \leq 1$ and $not(0 = 2)$ both of which are trivial tasks.

```
This is trivial. This is trivial.
```

**Exercises:**

Prove by induction:

1. The product of two consecutive integers is even.
2. The sum of two consecutive integers is odd.
3. $\forall n\ x \in \mathbb{N}, 1 + nx \leq (1 + x)^n$.(Hint: you might want to use Nat.pow_mul_l, Nat.mul_le_mono_l, Nat.le_trans and le_plus_trans)
4. $\forall n\ m \in \mathbb{N}, 2^{n+m} = 2^n 2^m$
5. $\forall n\ m \in \mathbb{N}, 2^{n*m} = (2^n)^m$ (Hint: you might want to use Nat.mul_succ_l, Nat.pow_add_r Nat.pow_mul_l.)
6. $\forall n \in \mathbb{N}, n \leq 2^n$ (Hint: you might want to use Nat.add_le_mono).
7. $\forall n \in \mathbb{N}, \sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i^2)^2$.

## 3.3 Divisibility

We start with a definition and a notation

```
Definition div a b := (exists n, b = a*n).
Notation "a | b" := (div a b) (at level 10).
```

Let us first show something easy:

```
 Lemma a: forall n, n |0.
Fix an arbitrary element n.
Rewrite goal using the definition of div.
Prove the existential claim is true for 0.
This is trivial.
Qed.
```

And also

```
 Lemma a: forall, 1| n.
Fix an arbitrary element n.
Rewrite goal using the definition of div.
Prove the existential claim is true for n.
This is trivial.
```

Let us prove an induction statement about divisibility:

**Lemma 8.**

$$\forall n \in \mathbb{N}, 3|4\,\hat{}n - 1$$

```
 Lemma c(n:nat) : 3 | (4^n -1).
Apply induction on n.
```

The first induction step is easy to do, you quickly unfold the definitions and it becomes trivial.

```
 Rewrite goal using the definition of Nat.pow.
Rewrite goal using the definition of div.
Prove the existential claim is true for 0.
This is trivial.
```

We are now left with the induction step:

Goal

$n : nat$
$IHn : 3|4\hat{}n - 1$

$3|4\hat{}(S\ n) - 1$

We unfold the definition of divides in IHn and find the s so that $4^n - 1 = 3s$

```
 Rewrite hypothesis IHn using the definition of div.
Fix s the existentially quantified variable in IHn .
```

We now unfold de definition of power :

```
Rewrite goal using the definition of Nat.pow.
```

To get

Goal

$n\ s : nat$
$IHn : 4\hat{}n - 1 = 3s$

$3|(4 * 4^n) - 1$

We now modify a bit the statement IHn to be useful. That is we try to show $3s = 4^n + 1$.

```
 Claim ( 3*s+1 = 4 ^n ).
Replace (3 * s) by ((4 ^n) - 1) in the goal.
```

We therefore need to show

Goal

$n\ s : nat$
$IHn : 4\hat{}n - 1 = 3s$

$(((4\hat{}n) - 1) + 1 = 4\hat{}n)$

This, sadly is not completely trivial. We need to use our old friend:

```
Apply result Nat.sub_add.
```

And we still have to show

Goal

$n\ s : nat$
$IHn : 4\hat{}n - 1 = 3s$

$1 \leq 4\hat{}n$

Which is exactly the statement of onepow (and the trivial statement that $not(0 = 4)$ .

```
Apply result onepow.
This is trivial.
```

We now make use of our newfound equality

```
Replace (4 ^n) by ((3 * s) + 1) in the goal.
```

To get

Goal

$n\ s : nat$
$IHn : 4\hat{\ }n - 1 = 3s$
$H : (3 * s) + 1 = 4\hat{\ }n$

$3|4 * (3s + 1) - 1$

We now unfold the definition of div

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

And do some calculations (to be proved later) in the goal.

```
Replace (4 * ((3 * s) + 1)) by (4 * (3 * s) + 4) in the goal.
```

Now we make Ann educated guess was to what the value of $n0$ should be:

```
Prove the existential claim is true for (4*s+1).
```

And we start proving that equality.

```
Replace (3 * (4 * s + 1)) by (4*(3*s) +3) in the goal.
```

Clean the computation a bit with a notation

```
Denote (4 * (3 * s)) by x.
```

And a few standard moves

```
 Replace 4 by (3+1) in the goal.
Rewrite the goal using plus_assoc.
Rewrite the goal using Nat.add_sub.
This follows from reflexivity.
```

finishes the main goal. We still have to show a few easy assertions that we made on the way:

$3 + 1 = 4$,

```
This follows from reflexivity.
```

$((4 * (3 * s)) + 3 = 3 * ((4 * s) + 1))$

```
True by arithmetic properties.
```

And finally $((4 * (3 * s)) + 4 = 4 * ((3 * s) + 1))$

```
 True by arithmetic properties.
```

Note that most of our troubles stemmed from the subtraction issues.

Let us prove another standard statement.

**Lemma 9.** *The product of any 3 consecutive natural numbers is divisible by 3.*

```
Lemma consecutive (n:nat): 3 | n*(n+1)*(n+2).
```

We unfold the definition of div, start an induction and easily eliminate the first case:

```
 Rewrite goal using the definition of div.
Apply induction on n.
Prove the existential claim is true for 0.
True by arithmetic properties.
```

We are left with

Goal

$n : nat$
$IHn : \exists n0 : nat, (n * (n + 1)) * (n + 2) = 3 * n0)$

$(\exists n0 : nat, ((Sn) * ((Sn) + 1)) * ((Sn) + 2) = 3 * n0)$

This is really hard to read and so we fix the value of $s$ in IHn and replace $S\ n$ by $n+1$ (to be shown later).

```
 Fix s the existentially quantified variable in IHn .
Replace (S n) by (n+1) in the goal.
```

We now make some more trivial replacement to be proved later:

```
 Replace (n+1 + 1) by (n+2) in the goal. Replace (n + 1 +2) by (n+3) in the
goal.
```

and one slightly more complicated:

```
 Replace (((n + 1) * (n + 2)) * (n + 3)) by (n*(n + 1) * (n + 2) +
3*(n+1)*(n+2)) in the goal.
```

Now we can use the induction Hypothesis:

```
 Rewrite the goal using IHn.
```

To get

Goal

$n\ s : nat$
$IHn : (n * (n+1)) * (n+2) = 3 * s)$

$\exists n0 : nat, (3 * s) + ((3 * (n+1)) * (n+2)) = 3 * n0$

W can now guess what $n0$ should be:

```
 Prove the existential claim is true for (s+ (n+1)*(n+2)).
```

The rest of the proof is rather trivial:

```
 True by arithmetic properties.
True by arithmetic properties.
True by arithmetic properties.
True by arithmetic properties.
True by arithmetic properties.
```

**Exercises:**

1. Prove by induction the product of 4 consecutive integers is divisible by 4.
2. Prove by induction that for any $n \in \mathbb{N}$, $5|6^n - 1$.
3. Prove by induction that for any $n \in \mathbb{N}$, $5|7^n - 2^n$.
4. Prove that if $a, b, c \in \mathbb{N}$ then $a|b$ and $a|c$ implies that $a|b + c$.
5. prove or disprove the converse of the previous question, that is if $a, b, c \in \mathbb{N}$ then $a|b + c$ implies that $a|b$ and $a|c$ .
6. Prove or disprove the following if $a, b, c \in \mathbb{N}$ and $a|bc$ then $a|b$ and $a|c$.

# Chapter 4

# Cartesian products, relations, functions

In this section we will discuss the cartesian product of two sets and their subsets with we will call (binary) relations. In most mathematics textbooks functions are defined to be special kinds of relations. Just like in Section 2.2 we define functions as relations initially in order for the reader to become comfortable with the concept. In order to take advantage of the fact that, from the point of view of type theory, functions are primitive concepts we will we will modify the definitions later on.

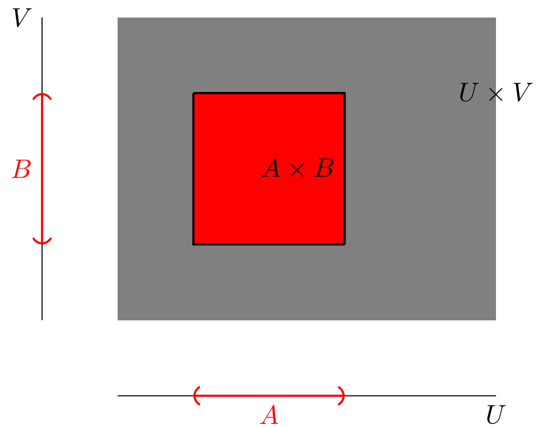
## 4.1 Cartesian products, relations

The notion of Cartesian product of two sets is very natural, it is the set of pairs of element one from $A$ and one from $B$.

**Definition 2.** If $A$ and $B$ are sets then:

$$A \times B = \{(a, b) \mid a \in A \wedge B \in B\}$$

.



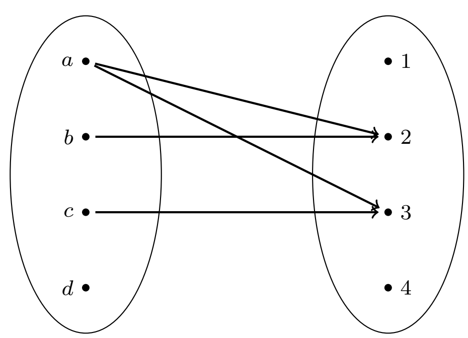

For example if $A = \{1,2,3\}$ and $B = \{a\}$ then $A \times B = \{(1,a),(2,a),(3,a)\}$. Note that from the point of view described in sections 2.2 and 2.2.3, if $A$ is of type Ensemble $U$ and $B$ is of type Ensemble $V$ then $A \times B : Ensemble(U * V)$. Note also that $A * B$ is endowed with two projections fst:$A * B \to A$ and snd :$A * B \to B$. We use these to define:

```
Definition prod ( U V:Type) (A :Ensemble U)(B:Ensemble V): Ensemble
(U*V):=fun x=> ((fst x) ∈ A ∧ (snd x) ∈ B) .
```

and

```
Notation "A 'X' B":=(prod _ _ A B)( at level 40).
```

We also define a (binary relation) $A$ with domain $A$ and range $B$ as subset of $A \times B$. One can think of a relations as a way of associating elements form $A$ and $B$. For example, assume that $A = \{a,b,c,d\}$ and $B = \{1,2,3,4\}$ and $R = \{(a,2),(a,3),(b,2),(c,3)\}$. We can represent this as:




Relations can represent various real life connections. For example one can think of a relation with domain the set of all people and range the set of all cars where a pair $(a,b) \in R$ if the person $a$ has ever been in car $b$. We can also think of more mathematical relations, for example the relation $R$

with domain $\mathbb{R}$ and range $\mathbb{R}$ given by $(x, y) \in R$ and only if $y = x^2$. We sometimes call this relation the graph of the function $f : \mathbb{R} \to \mathbb{R}, f(x) = x^2$.

```
Definition isrel ( U V:Type) (A :Ensemble U)(B:Ensemble V) (R:Ensemble (U*
V)) : R ⊆ (AXB).
```

Here is an example, let us define the relation less than with domain natural numbers and range all numbers larger than 1 and prove it is a relation. Note that we will not go into details about natural numbers here as we will do so in Chapter 3.

```
 Definition allnat:Ensemble nat: fun x=> True.
Definition strictlypos:Ensemble nat := fun x=> x>0.
Definition mylt:Ensemble nat*nat:= fun a => fst a < snd a.
```

Let us prove that melt is a relation with domain allnat and range strictlypos.

```
Lemma a: isrel nat nat allnat strictlypos mylt.
```

The first few steps are just unfolding of definitions:

```
 Rewrite goal using the definition of isrel.
Rewrite goal using the definition of Included.
Fix an arbitrary element a.
Assume (a ∈ melt) then prove (a ∈ (allnat X strictlypos)).
Rewrite hypothesis Hyp using the definition of (In, mylt).
Rewrite goal using the definition of (In, prod).
Rewrite goal using the definition of (In, allnat).
Prove the conjunction in the goal by first proving True then (strictlypos
(snd a)). This is trivial.
Rewrite goal using the definition of strictlypos.
```

We are now left with the goal

Goal

$a : nat * nat$
$Hyp : fst\ a < snd\ a$

$0 < snd\ a$

We now try to search for some useful theorem, one that implies that 0 is smaller than something:

```
SearchPattern (( _ -> 0 < _ )).
```

The result includes

Nat.lt_lt_0: $\forall nm : nat, n < m \rightarrow 0 < m$

And so if we do

```
Apply result (Nat.lt_lt_0 (fst a) (snd a)).
```

We only need to use the assumptions.

On the other hand

On the other hand, mylt is not a relation with domain strictlypositive.

```
Lemma a: not ( isrel nat nat strictlypos allnat mylt).
```

We can prove this by first unfolding some definitions:

```
Rewrite goal using the definition of not.
Assume (isrel nat nat strictlypos allnat mylt) then prove False.
Rewrite hypothesis Hyp using the definition of isrel.
Rewrite hypothesis Hyp using the definition of Included.
```

To get

Goal

$Hyp : \forall x : nat * nat, (x \in mylt) \rightarrow (x \in (strictlypos \times allnat)))$

$False$

Now we choose the element $(0, 1)$ and we show that $(0, 1) \in mylt$ and $(0, 1) \notin (strictlypos \times allnat))$ obtaining a contradiction.

The proof of $(0, 1) \in mylt$ is very easy:

```
Claim ((0,1) ∈ mylt). Rewrite goal using the definition of (In, mylt). This
is trivial.
```

The proof of $(0, 1) \notin$ (strictlypos $\times$ allnat)) is a bit harder:

```
Claim (not ((0,1) ∈ (strictlypos × allnat))).
Rewrite goal using the definition of (In, prod).
Rewrite goal using the definition of (In, strictlypos).
Rewrite goal using the definition of allnat.
Rewrite goal using the definition of not.
Assume (0 < fst (0, 1) ∧ True ) then prove False.
Eliminate the conjuction in hypothesis Hyp0.
Claim (0<0) by rewriting H0 using (fst (0,1)).
```

At this point we have:

Goal

$Hyp : (\forall x : nat * nat, (x \in mylt) \to (x \in (strictlypos \times allnat)))$
$H : (0, 1) \in mylt$
$H0 : 0 < fst(0, 1)$
$H1 : True$
$H2 : 0 < 0$

$False$

And we look for a theorem about not (_¡_)

```
SearchPattern (not (_< _))
```

.

To get

Nat.nlt_0_r: ∀ n : nat, $\neg n < 0$

We then apply this theorem and the rest is straightforward.

```
Apply result (Nat.nlt_0_r 0 H2).
Apply result H0 .
Apply result Hyp .
This follows from assumptions.
```

## 4.2 Binary relations on a set

There is an especially important subclass of general relations. If A is a type, a binary relation on $A$ is a subset of $A \times A$. This concept includes many examples you already know. Here are some natural examples:

1. if $A$ is any set the relation of equality can be viewed as the set
$$\{(a, a)|a \in A\}$$
.
2. If the set $A$ is some subset of real numbers you can define the relation of order
$$\{(a, b) \mid a, b \in A \wedge a < b\}$$
.
3. If $A = \mathbb{Z}$ you can define divisibility relation
$$\{(a, b) \mid a, b \in \mathbb{Z} \wedge a|b\}$$
4. if $A = \mathbb{Z}$ you can define the "= mod 3" relation as the set:
$$\{(a, b) \mid alb \in \mathbb{Z} \wedge 3|a - b\}$$
5. if $A$ is the set of all people you define two people to be related if they are from the same family.

We now describe things in Spatchcoq. For simplicity, in this section the set $A$ will be fixed and we will see relations not as in 4.1 but as predicates of two variables, that is as elements of type $A \to A \to Prop$. For that end we will use directly the package Relations in Coq.

```
Require Import Relation
```

For example, we can define the equality relation on a type $A$ as

```
Definition eq (A:Type): relation A: fun a b => a=b.
```

You can (re)define the order on natural numbers as :

```
Definition mylt:relation nat: fun a b => a<b.
```

Divisibility can be defined as:

```
Local Open Scope Z_scope.
Definition div:relation Z: fun a b => exists c:nat, b= a*c.
```

And we can define the mod 3 on $\mathbb{Z}$ as

```
Definition mod3:relation Z: fun a b => div 3 (a-b).
```

**!Warning!**

Note that for the definitions above I needed the integers and not the natural numbers, if we defined this in the natural numbers you will run into troubles regarding minus as in in 0.1

For simplicity very often we will also use the following common notation: if $R$ is a relation on the set $A$ and $(a, b) \in R$ we will write $aRb$.

Some relations have certain properties that are useful. We list them in the definition bellow.

**Definition 3 (reflexive).** A relation $R \subseteq A \times A$ is *reflexive* if $\forall a \in A, aRa$, that is any element is related to itself.

Note that this definition is already there

```
Print reflexive.
```

gives

reflexive = λ (A : Type) (R : relation A), ∀ x : A, R x x

For example equality and =mod3 are reflexive relation but " ¡ " is not. Here is a proof for mod3

```
Lemma a:reflexive Z mod3.
```

*Proof (informal).* We need to show that $forallx \in \mathbb{Z}, mod3xx$. We fix a $x$ and rewrite the definition of mod3. We therefore need show that $div3(x - x)$. If we rewrite the definition of divide we need to show that $\exists c \in \mathbb{Z}, (x - x) = 3c$. It remains to pick $c = 0$.

*Proof (formal).*

```
 Rewrite goal using the definition of reflexive.
Fix an arbitrary element x.
Rewrite goal using the definition of mod3.
Rewrite goal using the definition of div.
Prove the existential claim is true for 0.
True by arithmetic properties.
```

**Definition 4 (symmetric).** A relation $R \subseteq A \times A$ is *symmetric* if $\forall ab \in A, aRb \to bRa$.

As before:

```
Print symmetric.
```

Gives

symmetric = $\lambda$ (A : Type) (R : relation A), $\forall$ x y : A, R x y $\to$ R y x

We will now prove that mod3 is also symmetric:

```
 Lemma a:symmetric Z mod3.
```

*Proof (informal).* We need to show that $forallxy \in \mathbb{Z}, mod3xy \to mod3yx$. To do so we fix $x$ and $y$ and use the definitions of mod3 respectively div. We are left to prove that $\exists c : Z, x - y = 3 * c \to \exists c : Z, y - x = 3 * c$ and so we assume that $\exists c : Z, x - y = 3 * c$ and prove that $\exists c : Z, y - x = 3 * c$ if we pick the c so that $x - y = 3c$ then it is not too hard to see that $y - x = 3(-c)$.

The formal proof is slightly harder.

*Proof (formal).*

```
 Rewrite goal using the definition of symmetric.
Fix an arbitrary element x.
Fix an arbitrary element y.
Rewrite goal using the definition of mod3.
Rewrite goal using the definition of div.
Assume (∃ c : Z, x - y = 3 * c) then prove (∃ c : Z, y - x = 3 * c).
Fix c the existentially quantified variable in Hyp .
Prove the existential claim is true for (-c).
Replace (3 * c) by (x - y) in the goal.
Claim (3*(-c)= -(3*c)).
True by arithmetic properties.
Rewrite the goal using H .
Replace (3 * c) by (x - y) in the goal.
True by arithmetic properties.
```

Note that the “Claim (3*(-c)= -(3*c)).” is needed here while it was implicitely used in our informal proofs.

**Definition 5 (transitive).** A relation $R \subseteq A \times A$is *transitive* if $\forall abc \in A, aRb \wedge bRc \rightarrow aRc$.

However

```
Print transitive.
```

Gives

```
transitive= λ (A : Type) (R : relation A), ∀ x y z : A, R x y → R y z → R x z
```

This looks a bit different but the two are equivalent. See for example exercise andimp ar page 26.

Let us prove that mod3 is transitive

```
Lemma trans: transitive Z mod3.
```

```
 Rewrite goal using the definition of transitive.
Fix an arbitrary element x.
Fix an arbitrary element y.
Fix an arbitrary element z.
Rewrite goal using the definition of mod3.
Rewrite goal using the definition of div.
Assume (∃ c : Z, x - y = 3 * c) then prove
```
$((\exists c : Z, y - z = 3 * c) \to (\exists c : Z, x - z = 3 * c))$.
```
Assume (∃ c : Z, y - z = 3 * c) then prove (∃ c : Z, x - z = 3 * c).
Fix c the existentially quantified variable in Hyp .
Fix d the existentially quantified variable in Hyp0 .
Prove the existential claim is true for (c+d).
Claim (3*(c+d)= 3*c+3*d).
True by arithmetic properties.
Replace (3 * (c + d)) by ((3 * c) + (3 * d)) in the goal.
Replace (3 * c) by (x - y) in the goal.
Replace (3 * d) by (y - z) in the goal.
True by arithmetic properties.
```

## 4.3 Functions

## Chapter 5

# Algebraic Structures

### 5.1 Groups

We will briefly define abstract groups.

**Definition 6.** A group is a triple $(G, mult, id, inv)$ where $G$ is a set, mult is a binary operation that associates to every pair of elements $ab$ an element $mult\ a\ b$, $id$ is an element of $G$ and $inv : G \to G$ is a function such that.

mult_closure G is closed under multiplication, that is $\forall ab \in G, mult\ a\ b \in G$.

- **assoc** $\forall x\ y\ z, mult\ (mult\ x\ y)z = mult\ x(mult\ yz)$
- **right_id** $\forall x, mult\ x\ id = x$
- **inv_closure** $\forall x \in G, (inv\ x) \in G$
- **right_inverse** $\forall x, mult\ x(inv\ x) = id$

Note that we gave been deliberately vague in assoc, right_id and right_inverse. This is because we will be somewhat light on this in our definition. We also did not require left inverses or left identity, we shall prove these later.

To introduce the definition of a group we will use some new Coq notions, modules and records. We start by Importing sets and recalling the set notations

```
 Require Import Ensembles.
Notation "x ∈ A":= (In _ A x) (at level 10).
Notation "A ⊆ B":= (Included _ A B)(at level 10).
Notation "A ∪ B":= (Union _ A B)(at level 8).
Notation "A ∩ B" := (Intersection _ A B) (at level 10).
```

We now start a module. This is convenient for polymorphisms.

```
Module gps.
```

The definition of a groups resembles Definition 6 closely.

```
Record Group : Type := group
{U:Type;
setG : Ensemble U;
mult : U -> U -> U;
inv : U -> U;
id : U;
mult_closure : ∀xy : U, x ∈ setG → y ∈ setG → (multxy) ∈ setG;
assoc : ∀xyz : U, mult (mult x y) z = mult x(mult y z);
id_closure : id ∈ setG;
right_id :∀x : U, mult x id = x;
inv_closure : ∀x : U, (inv x) ∈ setG;
right_inverse: ∀x : U, mult x(inv x) = id; }.
```

Note the format, this is in fact an inductive constructor, not unlike *nat*.

We introduce some convenient notations:

```
Notation "x {*} y":=(mult _ x y) (at level 50).
Notation "'e'":=(id _ ) (at level 50).
Notation "x ^-1'":=(inv _ x) (at level 30).
```

And we are now ready to prove the first group theory lemma, the uniqueness of identity (we shall prove the theorem inside the module gps so we can use it with any group.

```
Lemma unit_uniq (U:Type)(G:Group): forall x:gps.U G, x {*} x = x -> x = e.
```

The "gps:U G" notation looks a bit strange but this automatically generated.

The resulting goal is

Goal

$U : Type$
$G : Group$

$\forall x : gps.U\ G, x\{*\}x = x \to x = e)$

The first two moves are standard.

```
Fix an arbitrary element x.
Assume (x {*} x = x ) then prove (x=e).
```

And we arrive at

Goal

$U : Type$
$G : Group$
$x : gps.U\ G$
$Hyp : x\{*\}x = x$

$x = e$

We will now use the fact that $x = x * e$ and we replace it in the goal (and prove it later)

```
Replace x by (x * e) in the goal.
```

Goal

$U : Type$
$G : Group$
$x : gps.U\ G$
$Hyp : x\{*\}x = x$

$x\{*\}e = e$

Respectively

Goal

$U : Type$
$G : Group$
$x : gps.U\ G$
$Hyp : x\{*\}x = x$

$x\{*\}e = x$

We now know that x {*} x ˆ-1 = e and so

Goal

$U : Type$
$G : Group$
$x : gps.U\ G$
$Hyp : x\{*\}x = x$

$x{*}(x{*}x\hat{}-1) = x{*}x\hat{}-1$

Respectively

Goal

$U : Type$
$G : Group$
$x : gps.U\ G$
$Hyp : x\{*\}x = x$

$x{*}x\hat{}-1 = e$

```
 Replace (id G) by (x * x ^ -1) in the goal.
```

We now apply associativity and Hyp.

```
 Rewrite the goal using (assoc G).
Rewrite the goal using Hyp .
This follows from reflexivity.
```

And we now only need to show the two goals that we introduced.

```
 Apply result (right_inverse).
Apply result (right_id G).
Qed.
End gps.
```

Now we show how to prove that $Z$ is a group under addition:

```
 Open Scope Z_scope.
Import gps.
Definition gZ:Group.
Apply result (group Z (fun x=> True) Z.add Z.opp 0).
Rewrite goal using the definition of In.
This is trivial.
Fix an arbitrary element x.
Fix an arbitrary element y.
Fix an arbitrary element z.
True by arithmetic properties.
Rewrite goal using the definition of In.
This is trivial.
Fix an arbitrary element x.
True by arithmetic properties.
Rewrite goal using the definition of In.
This is trivial.
Fix an arbitrary element x.
True by arithmetic properties.
Qed.
```

And we can see that one can apply unit_uniq immediately.

```
 Lemma b: forall x:gps.U gZ, x * x = x -> x = e.
Apply result (unit_uniq Z s).
Qed.
```

# Appendix A
# The software

## A.1 Installation

### *A.1.1 Online*

1. Go to http://app.spatchcoq.co.uk/
2. enjoy

### *A.1.2 Mac OSX*

1. You might need to install gtk, the simplest way to do this is via homebrew.

   If do not have homebrew installed, install it from https://brew.sh

   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+
   ```

2. 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.

3. Download and unpack spatchocq.app from http://app.spatchcoq.co.uk/ , move the spatchcoq.app to Applications and start it.
4. 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.

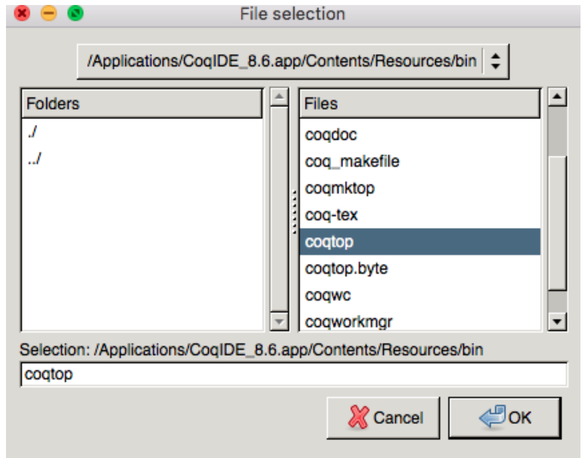



**Fig. A.1** Choose the Coq app in a Mac env

5. enjoy

### *A.1.3 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.1.4 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 Online*

Figure A.2 is the main window in the online version.

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 white window : this is the Coq feedback window.
4. The grey 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.

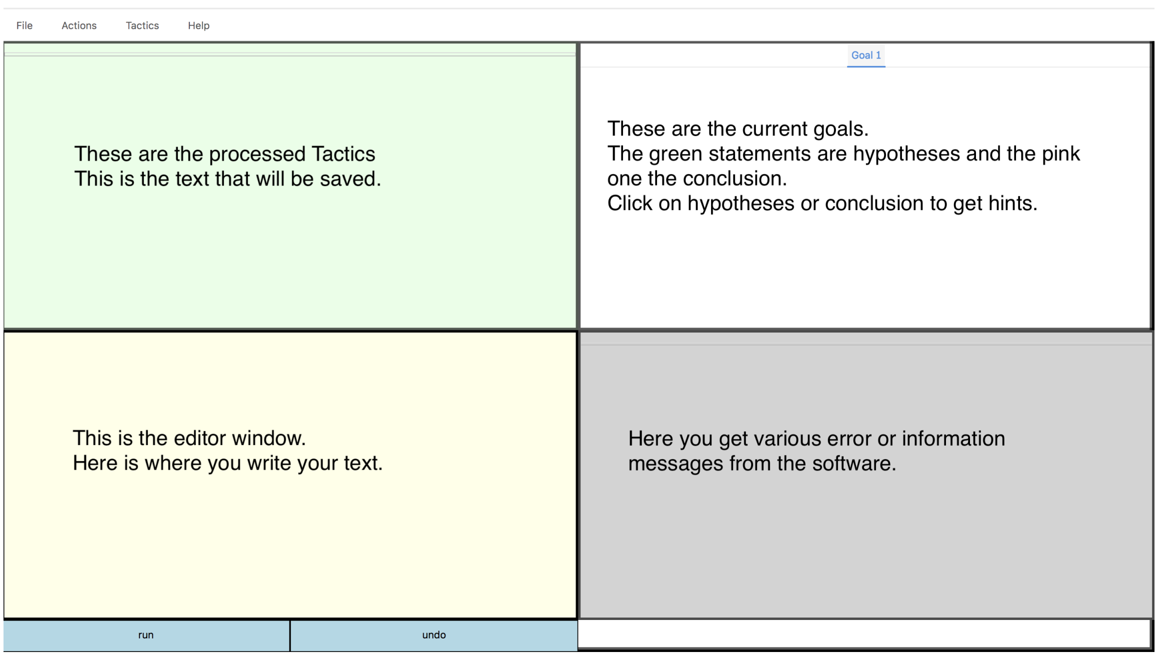



**Fig. A.2** the online GUI

### A.2.2 The menus

The File menu (Figure A.7) is quite standard:

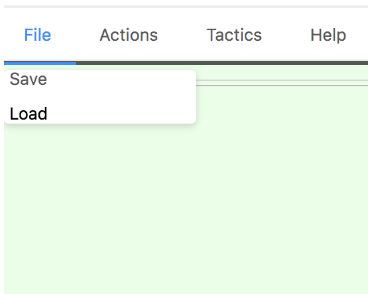



**Fig. A.3** the File Menu

The action menu allows you to pick run, undo or print tree:

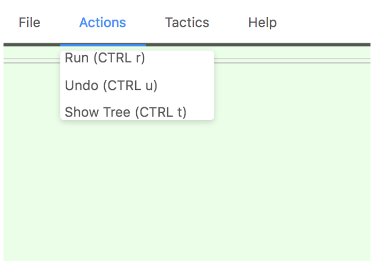



**Fig. A.4** the Action Menus

The Tactics menu (Figure A.5) allows one to pick one of the predefined tactics. Note the place keeper VAR. These can be modified. More on these later.

### A.2.3 Keyboard shortcuts

- CTRL-Space autocompletion
- CTRL-r Run
- CTRL-u Undo
- CTRL-t Draw tree.

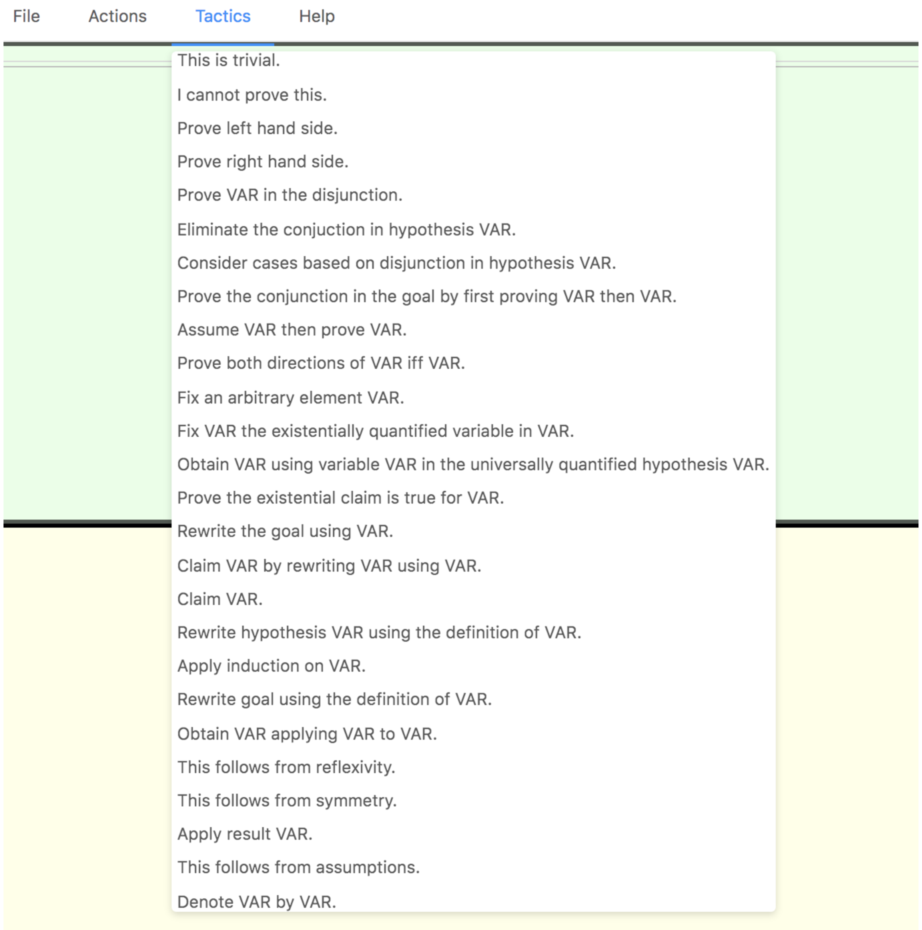



**Fig. A.5** the Tactics Menus

## *A.2.4 Desktop*

### A.2.4.1 Main windows

Figure A.6 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.6** the GUI

### *A.2.5 The menus*

The File menu (Figure A.7) is quite standard:

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

### *A.2.6 Keyboard shortcuts*

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

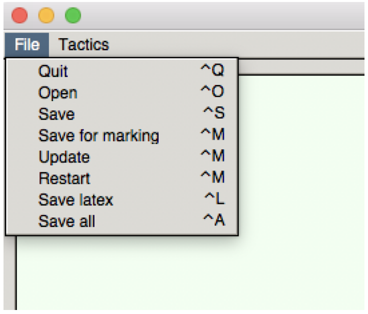



**Fig. A.7** the File Menu

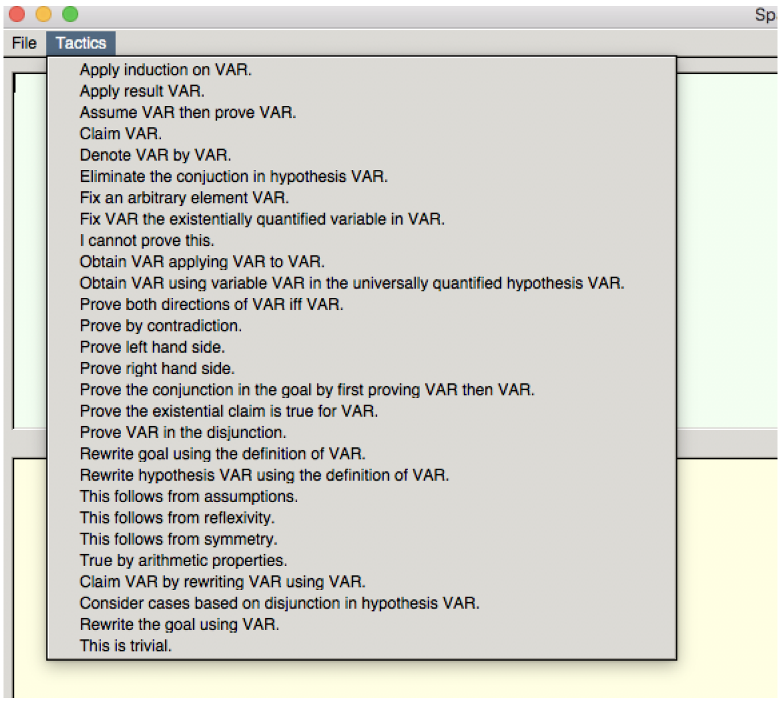



**Fig. A.8** the Tactics/Environment Menus

## Appendix B

# A brief overview of the notations in the language

We will list the formats of the various constructions we use. For more details see https://coq.inria.fr/refman/language/gallina-specification-language.html.

### Check

```
Check X.
```

This will print the type of the statement $X$. For example the command

```
Check le.
```

Will produce

le : nat $\to$ nat $\to$ Prop

---

### Print

```
Print X.
```

This will print the full definition (proof) of of the statement $X$. For example the command

```
Print le.
```

Will produce

le inductive le (n : nat) : nat → Prop := le_n : n ≤ n | le_S : ∀ m : nat, n ≤ m → n ≤ S m

Which describes the two ways to check that $n \leq m$,

1. if $n = m$ then $n \leq m$.
2. otherwise you look at the predecessor of m and retry.

---

**Varible/Axiom**

```
Variable name:Type.
```

The two keywords can be used interchangeably but, to keep with normal mathematical notation, one should use Axiom to define variables of type Prop and Variable for all other types. For example one can use

```
Variable myax:3=1+2.
```

or Axiom myax:3=1+2. To mean that max is a "proof" of 3=1+2. However in practice we should only use the latter. Similarly we can use

```
Variable n:nat.
```

or

```
Axiom n:nat
```

To introduce a natural number $n$ but we should really only use the former.

**Definition**

```
Definition name vars := term.
```

or

```
definition name vars : Type:= term.
```

This names an object of a certain type.

```
Definition thenumber3:=3.
```

or

```
Definition thenumber3:nat:=3.
```

Both define the object thenumber3 to be the natural number 3. The second one is more precise since it forces Spatchcoq to verify that it has the type you want. For example

```
Definition thenumber3:Prop:=3.
```

Will give the message

Error: The term "3" has type "nat" while it is expected to have type "Prop".

Definitions can of course depend on parameters. For example

```
Definition mult a b:= a*b.
```

Defines the multiplication of two numbers, it does assume that the numbers are natural numbers so if you do

```
Check mult.
```

You will get

mult: nat $\rightarrow$ nat $\rightarrow$ nat

Of course you can be precise:

```
Definition mult a b :Z:= a*b.
```

To get

mult : Z $\rightarrow$ Z $\rightarrow$ Z

Definitions are useful for writing shorter text and can be unfolded using the tactics “Rewrite goal using the definition of VAR.” or “Rewrite hypothesis VAR using the definition of VAR.”

### Notation

```
 Notation ‘‘ notation ’’:= (term)( at level x).
```

This introduces notations. For example we might want to write things nicely let us assume that we have two predicates on the type U one is called P and the other Q.

```
Variables U:Type Variables: P Q:U → Prop
```

We now decide to say that $U$ is the set of beings and that $Px$ means that “x is human” and $Qx$ means that “x will eventually die” then We can have the following notations:

```
 Notation " ’beings’ ":= U.
Notation "x ’is’ ’a’ ’human’ ":= (Q x) (at level 10).
Notation "x ’will’ ’eventually’ ’die’ ":= (Q x) (at level 10).
```

Now let us assume we want to prove that $\forall x, Px \rightarrow Qx$.

```
Lemma z:forall x, P x → Q x.
```

Note that the response is

Goal

$\forall x$ : beings, (x is a human) $\rightarrow$ (x will eventually die)

Which reads a quite a bit better.

### Lemma/Proposition/Theorem

```
Lemma name (vars1: Type)(Vars2:Type2)··· : statement .
```

A lemma, proposition or theorem is a statement that needs to be proved. They all have the same shape. You start with a either Lemma, Proposition or Theorem. The next entry is the name of the theorem. This can be anything. Here is a very easy example:

```
Theorem the_easiest_theorem: 1=1.
```

Note that this theorem does not depend on any variables. This is perfectly ok, in fact in mathematics we are used with not defining theorems that have parameters.

By comparison, the statement

```
Theorem the_second_easiest_theorem (n:nat): n=n.
```

Is a theorem that contains the parameter $n$ which is defined to be a natural number.

# Appendix C
# 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:

| Goal |
|---|
| $\cdots$ |
| $P \vee Q$ |

The above mentioned tactic will the produce a the following goal

Goal

$P$

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:

Goal

...

$P \lor Q$

The above mentioned tactic will the produce a the following goal

Goal

...

$Q$

**Prove VAR in the disjunction.**

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

Goal

...

$P \lor Q$

Then applying

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

will produce the goal

Goal

$\cdots$

$P$

while applying

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

will produce the goal

Goal

$\cdots$

$Q$

**Eliminate the conjuction in hypothesis VAR.**

Suppose your goal looks like

Goal

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

$\cdots$

Then applying

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

will produce a goal similar to the one below:

Goal

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

$\cdots$

allowing you to use the parts of Hyp independently.

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

Suppose your goal looks like

Goal

$\dots$
$Hyp : P \vee Q$
$\dots$

$\dots$

Then applying

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

will produce two separate goals similar to the one below:

Goal

$\dots$
$Hyp : P$
$\dots$

$\dots$

Goal

$\dots$
$Hyp : Q$
$\dots$

$\dots$

obtaining a proof by cases.

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

Suppose your goal looks like

Goal

$\cdots$

$P \wedge Q$

Then

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

will separate the proof in two different goals

Goal

$\cdots$

$P$

Goal

$\cdots$

$Q$

**Assume VAR then prove VAR.**

Suppose your goal looks like

Goal

$\cdots$

$P \to Q$

then

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

will modify the goal to

Goal

$\dots$

$P$

$Q$

**Prove both directions of VAR iff VAR.**

Suppose your goal looks like

Goal

$\dots$

$P \leftrightarrow Q$

then

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

will split the goal into two different goals

Goal

$\dots$

$P \to Q$

Goal

$\dots$

$Q \to P$

**Fix an arbitrary element VAR.**

Suppose your goal looks like

Goal

$\ldots$

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

then

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

will modify the goal to

Goal

$\ldots$

$a : S$

$P(a)$

**Fix VAR the existentially quantified variable in VAR.**

Suppose your goal looks like

Goal

$\ldots$

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

$\ldots$

then

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

will modify the goal to

Goal

$\dots$

$a : S$

$Hyp : P(a)$

$\dots$

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

Suppose your goal looks like

Goal

$\dots$

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

$\dots$

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

Goal

$\dots$

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

then

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

will modify the goal to

Goal

$\ldots$

$P(a)$

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

Suppose your goal looks like

Goal

$\ldots$

$Hyp : x = f$

$P(x)$

then

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

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

Goal

$\ldots$

$Hyp : x = f$

$P(f)$

```
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

Goal

$\ldots$

$P(x)$

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

Goal

$\dots$

$Q$

into

Goal

$\ldots$

$P$

and

Goal

$\ldots$

$Hyp : P$

$Q$

**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

Goal

$\ldots$

$P(n)$

then

```
Apply induction on n.
```

will split the proof into two goals

Goal

$\dots$

$P(0)$

and

Goal

$\dots$

$IHn : P(n)$

$P(Sn)$

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

Goal

$\dots$

$P(n)$

will be split into 3 cases

Goal

$\dots$

$P(0)$

Goal

$\dots$

$n : positive$

$P(Z.pos\ n)$

Goal

$\ldots$

$n : negative$

$P(Z.neg\ n)$

**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:

Goal

$\ldots$

$P$

then

```
Prove by contradiction.
```

will transform the goal to

Goal

$\dots$

$\neg P$

$False$

**This follows from reflexivity.**

Assume the goal is

Goal

$\dots$

$a = a$

then

```
This follows from reflexivity.
```

will finish the proof.

This follows from symmetry.

**Apply result VAR.**

Assume that the goal is

Goal

$\dots$

$Hyp : P- > Q$

..

$Q$

Then

```
Apply result Hyp
```

will transform the goal to

Goal

$\ldots$
$Hyp : P- > Q$
$..$

$P$

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

```
Apply result thm
```

will transform the goal to

Goal

$\ldots$
$Hyp : P- > Q$
$..$

$P$

**This follows from assumptions.**

if the goal is

Goal

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

$P$

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 D
# 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.

### *D.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

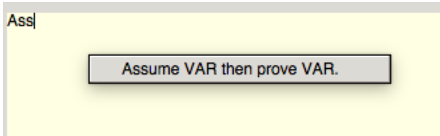
Goal

$P, Q : Prop$

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

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.

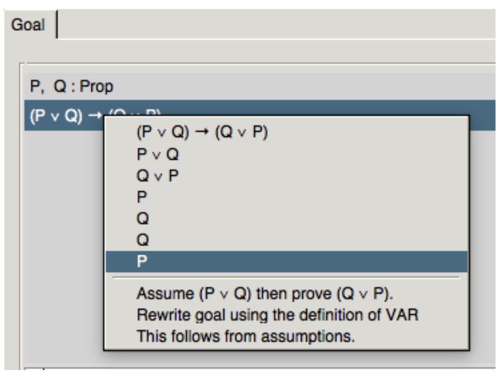
write $(P \vee Q)$ to replace the first VAR. Repeat CTRL-r and and replace the second VAR by $(Q \vee P)$.

The text in the yellow window should now be

Assume $(P \vee Q)$ then prove $(Q \vee P)$.

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 $(P \vee Q)$ then prove $(Q \vee P)$.

and click run.

The response from Coq is

Goal

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

$Q \vee P$

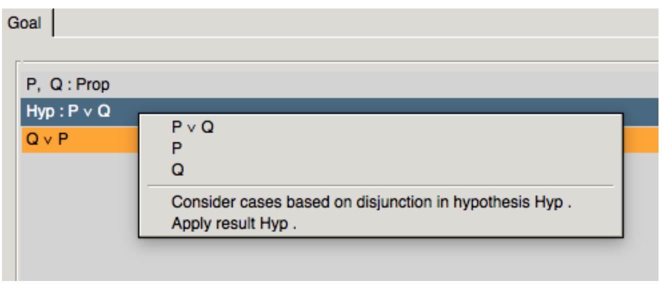
This reflects the fact that we have a new hypothesis tabled Hyp and a new conclusion.

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:

Goal

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

$Q \vee P$

and

Goal

$P\ Q$ : 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)

Goal

$P\ Q$ : 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

Goal

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. D.1** the tree window

### *D.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 $= \lambda ab : nat, \exists x : nat, b = a * x$
`: 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 $\lambda$, 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).
```

Goal

$a, b, c : nat$

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

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

Goal

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

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

Goal

$a, b, c : nat$

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

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

Goal

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

$a|c$

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

Goal

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

$a|c$

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.
```

Goal

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

$a|c$

then

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

Goal

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

$a|c$

and

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

Goal

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

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

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

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

to get

Goal

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

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

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

Goal

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

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

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

Goal

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

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

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

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

to obtain

Goal

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

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

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)** $\forall\, a\, b\, c, \, (a \,|\, b) \, \wedge (b \,|\, c) \; \Rightarrow \; (a \,| c).$
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 E

# Sets vs types

## E.1 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:

Goal

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

$\ldots$

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

Let us try some experiments. We first define some variables: P and Q will 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

$h : P \to Q.$

Note that, in particular h is NOT a proposition but an object of type $P- > Q$, I.e. a witness(proof) of the implication $P- > Q$. Similarly if you define

```
Axiom aaa:2=1+1.
```

then

```
Check aaa.
```

will produce

aaa : $2 = 1 + 1$

That means that aaa is a witness of the equality $2 = 1 + 1$ and that you can refer to aaa in other proofs (for example using rewrite).

For much of the book one can look at the notation $a : nat$ as a SpatchCoq version of $a \in \mathbb{N}$. 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 $P \to Q$ 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 object which is of type $Type_1$ and so on. Note for example:

```
Check Type:Type
```

which produces: $Type : Type : Type$.

# References


1. José Grimm. Implementation of bourbaki's elements of mathematics in coq: Part one, theory of sets. Research Report RR-6999, INRIA, 2011.

2. C. Simpson. Set-theoretical mathematics in Coq. *ArXiv Mathematics e-prints*, February 2004.