A Gentle Introduction to the Coq Proof Assistant, from a Teaching Perspective

Nicolas Magaud

Lab. ICube UMR 7357 CNRS Université de Strasbourg

Université de Strasbourg

PAT 2023 Thematic School

19 - 23 June 2023

Val d'Ajol, France

https://dpt-info.u-strasbg.fr/~magaud/PAT2023

Three-part Course (Nicolas Magaud and Yves Bertot)

1. Monday 11:00-12:30 : Logic and Computation (Nicolas Magaud)
2. Tuesday 14:00-15:30 : Trusting Proof Automation (Nicolas Magaud)
3. Wednesday 17:00-18:00 : Numbers in Coq (Yves Bertot)

What you will learn in this course:

1. Better understand what a proof is, carry out proofs more carefully.
2. Discover the field of formal proofs.
3. Practical aspects: using Coq.
4. More theoretical aspects : the Curry-Howard isomorphism, ...

Acknowlegements

This course is built upon the lectures that Julien Narboux and I give yearly to students of the computer science master at the University of Strasbourg. This is greatly inspired by lectures by other people including:

- Yves Bertot
- Gilles Dowek
- Hugo Herbelin
- Pierre Lescanne
- David Pichardie
- Benjamin Werner
- Laurent Théry
- ...

What is a proof ?

- something convincing,
- a sequence of deductions from the axioms,
- an algorithm (Curry-Howard isomorphism)

Yes but. . .

It may be difficult to be sure that a proof is actually correct:

- the number of statements involved
- the occurence of computations
- too many technical details, too many subcases
- the size of the proof

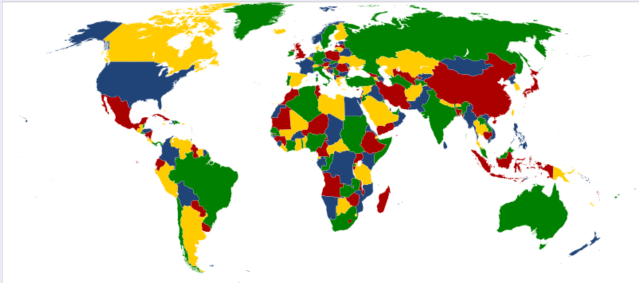
Verifying that a proof is correct

When computations occur

Four color theorem

No more than four colors are required to color the regions of any map so that no two adjacent regions have the same color.

- 1976 Appel and Hake (1478 configurations, 1200 hours of computations)
- 2004 Formalized in Coq by Gonthier and Werner



Verifying that a proof is correct

When computations occur

Kepler conjecture/Hales theorem

For a packing of equally-sized spheres, the maximum density is obtained by a face-centered cubic arrangement.

- **1998** Mathematical proof by Thomas Hales
- **2004 - 2014** Projet Flyspeck: formalizing the theorem using HOL-light with contributions in Coq and Isabelle (more than 300 000 lines)

Photo by Robert Cudmore

Robert MacPherson, editor, wrote:

> *"The news from the referees is bad, from my perspective. They have not been able to certify the correctness of the proof, and will not be able to certify it in the future, because they have run out of energy to devote to the problem. This is not what I had hoped for. The referees put a level of energy into this that is, in my experience, unprecedented. "*

Verifying that a proof is correct

The proof size

Théorème de Feit-Thompson

```
Theorem Feit_Thompson (gT:finGroupType) (G:{group gT}):
odd ##|G| -> solvable G.
```

Proof in Coq by Georges Gonthier et al. (september 2012)[a]:
170 000 lines, 15 000 definitions, 4 200 theorems

[a] https://mathlesstraveled.com/2012/11/11/a-computer-checked-proof-of-the-odd-order-theorem/

Verifying that a proof is correct

Some success stories

- CompCert: a C compiler proved correct in Coq (Xavier Leroy et. al.)
- seL4: a micro kernel proved correct in Isabelle (Gerwin Klein et. al.)
- A payment system (Gemalto, Andronick et. al.)
- The automation of some underground lines (e.g. line 14 in Paris)
- A hash function (SHA 256, Andrew Appel, 2015)
- A crypotographic protocol (OpenSSL HMAC, Andrew Appel et. al., 2015)

Improving the quality of proofs

1. Make the *hypotheses* clearer
   (as precise as possible, not too restrictive)
2. Make it clear what a *good proof* actually is
3. Be as precise as possible so that we do not need to understand the proof to check it.
4. Automate some parts of the proofs

The Coq Proof Assistant

What is Coq ?

- A Proof Assistant, developped and distributed by INRIA
- Try it easily ! `https://coq.vercel.app/`
- Install with opam: `https://coq.inria.fr/opam-using.html`

It allows :

- to define mathematical notions/programs,
- and to prove some properties of these objects.

ACM Software System Award

2015 GCC
2014 Mach
2013 **Coq**
2012 LLVM
2011 Eclipse
2010 GroupLensCFRS
2009 VMware
2008 Gamma Parallel Database System
2007 Statemate
2006 Eiffel
2005 The Boyer-Moore Theorem Prover
2004 Secure Network Programming
2003 Make
2002 Java
...
1991 TCP-IP

Why do we (Need to) Formalize Mathematical Results?

- The Example of the Finite Projective Space PG(3,3)
  - Projective Incidence Geometry only features points and lines, together with an incidence relation ($\in$).
  - Projective Incidence Geometry can be captured by a set of axioms.
  - PG(3,3) is a finite projective space with 35 points and 130 lines. It is a model of Projective Incidence Geometry.
  - Each line contains exactly 4 points.
  - Lines are easily represented as sets of points, as Alan R. Prince did in a journal article.[1]
  - The specification is actually wrong (this is a minor error, but still an error).

[1] Projective planes of order 12 and PG(3,3). Discrete Mathematics, 208-209 :477-483, 1999.

PG(3,3) - description of the incidence relation

L1	0	1	4	13	*L14*	1	2	5	14	*L27*	1	7	12	33	*L40*	1	8	24	26	*L53*	1	10	37	38
L2	0	2	24	17	*L15*	2	3	6	15	*L28*	2	19	26	4	*L41*	2	22	12	32	*L54*	2	33	29	13
L3	0	3	12	39	*L16*	3	5	27	20	*L29*	3	38	32	24	*L42*	3	10	26	28	*L55*	3	4	7	16
L4	0	5	26	34	*L17*	5	6	9	18	*L30*	5	30	28	12	*L43*	5	33	32	36	*L56*	5	24	19	13
L5	0	6	32	11	*L18*	6	27	35	1	*L31*	6	16	36	26	*L44*	6	4	28	21	*L57*	6	12	38	17
L6	0	27	28	31	*L19*	27	9	25	2	*L32*	27	13	21	32	*L45*	27	24	36	23	*L58*	27	26	30	39
L7	0	9	36	37	*L20*	9	35	14	3	*L33*	9	17	23	28	*L46*	9	12	21	8	*L59*	9	32	16	34
L8	0	35	21	29	*L21*	35	25	15	5	*L34*	35	39	8	36	*L47*	35	26	23	22	*L60*	35	28	13	11
L9	0	25	23	7	*L22*	25	14	20	6	*L35*	25	34	22	21	*L48*	25	32	8	10	*L61*	25	36	17	31
L10	0	14	8	19	*L23*	14	15	20	6	*L36*	14	11	10	23	*L49*	14	28	22	33	*L62*	14	21	39	37
L11	0	15	22	38	*L24*	15	20	1	9	*L37*	15	31	33	8	*L50*	15	36	10	4	*L63*	15	23	34	29
L12	0	20	10	30	*L25*	20	18	2	35	*L38*	20	37	4	22	*L51*	20	21	33	24	*L64*	20	8	11	7
L13	0	18	33	16	*L26*	18	1	3	25	*L39*	18	29	24	10	*L52*	18	23	4	12	*L65*	18	22	31	19

L66	1	11	21	31	*L79*	1	16	23	39	*L92*	1	17	19	34	*L105*	1	22	30	36	*L118*	1	28	29	32
L67	2	31	23	37	*L80*	2	13	8	34	*L93*	2	39	38	11	*L106*	2	10	16	21	*L119*	2	36	7	28
L68	3	37	8	29	*L81*	3	17	22	11	*L94*	3	34	30	31	*L107*	3	33	13	23	*L120*	3	21	19	36
L69	5	29	22	7	*L82*	5	39	10	31	*L95*	5	11	16	37	*L108*	5	4	17	8	*L121*	5	23	38	21
L70	6	7	10	19	*L83*	6	34	33	37	*L96*	6	31	13	29	*L109*	6	24	39	22	*L122*	6	8	30	23
L71	27	19	33	38	*L84*	27	11	4	29	*L97*	27	37	17	7	*L110*	27	12	34	10	*L123*	27	22	16	8
L72	9	38	4	30	*L85*	9	31	24	7	*L98*	9	29	39	19	*L111*	9	26	11	33	*L124*	9	10	13	22
L73	35	30	24	16	*L86*	35	37	12	19	*L99*	35	7	34	38	*L112*	35	32	31	4	*L125*	35	33	17	10
L74	25	16	12	13	*L87*	25	29	26	38	*L100*	25	19	11	30	*L113*	25	28	37	24	*L126*	25	4	39	33
L75	14	13	26	17	*L88*	14	7	32	30	*L101*	14	38	31	16	*L114*	14	36	29	12	*L127*	14	24	34	4
L76	15	17	32	39	*L89*	15	19	28	16	*L102*	15	30	37	13	*L115*	15	21	7	26	*L128*	15	12	11	24
L77	20	39	28	34	*L90*	20	38	36	13	*L103*	20	16	29	17	*L116*	20	23	19	32	*L129*	20	26	31	12
L78	18	34	36	11	*L91*	18	30	21	17	*L104*	18	13	7	39	*L117*	18	8	38	28	*L130*	18	32	37	26

PG(3,3) - description of the incidence relation

*L*1	0	1	4	13	*L*14	1	2	5	14	*L*27	1	7	12	33	*L*40	1	8	24	26	*L*53	1	10	37	38
*L*2	0	2	24	17	*L*15	2	3	6	15	*L*28	2	19	26	4	*L*41	2	22	12	32	*L*54	2	33	29	13
*L*3	0	3	12	39	*L*16	3	5	27	20	*L*29	3	38	32	24	*L*42	3	10	26	28	*L*55	3	4	7	16
*L*4	0	5	26	34	*L*17	5	6	9	18	*L*30	5	30	28	12	*L*43	5	33	32	36	*L*56	5	24	19	13
*L*5	0	6	32	11	*L*18	6	27	35	1	*L*31	6	16	36	26	*L*44	6	4	28	21	*L*57	6	12	38	17
*L*6	0	27	28	31	*L*19	27	9	25	2	*L*32	27	13	21	32	*L*45	27	24	36	23	*L*58	27	26	30	39
*L*7	0	9	36	37	*L*20	9	35	14	3	*L*33	9	17	23	28	*L*46	9	12	21	8	*L*59	9	32	16	34
*L*8	0	35	21	29	*L*21	35	25	15	5	*L*34	35	39	8	36	*L*47	35	26	23	22	*L*60	35	28	13	11
*L*9	0	25	23	7	*L*22	25	14	20	6	*L*35	25	34	22	21	*L*48	25	32	8	10	*L*61	25	36	17	31
*L*10	0	14	8	19	*L*23	14	15	20	6	*L*36	14	11	10	23	*L*49	14	28	22	33	*L*62	14	21	39	37
*L*11	0	15	22	38	*L*24	15	20	1	9	*L*37	15	31	33	8	*L*50	15	36	10	4	*L*63	15	23	34	29
*L*12	0	20	10	30	*L*25	20	18	2	35	*L*38	20	37	4	22	*L*51	20	21	33	24	*L*64	20	8	11	7
*L*13	0	18	33	16	*L*26	18	1	3	25	*L*39	18	29	24	10	*L*52	18	23	4	12	*L*65	18	22	31	19

18 27

30

*L*66	1	11	21	31	*L*79	1	16	23	39	*L*92	1	17	19	34	*L*105	1	22	30	36	*L*118	1	28	29	32
*L*67	2	31	23	37	*L*80	2	13	8	34	*L*93	2	39	38	11	*L*106	2	10	16	21	*L*119	2	36	7	28
*L*68	3	37	8	29	*L*81	3	17	22	11	*L*94	3	34	30	31	*L*107	3	33	13	23	*L*120	3	21	19	36
*L*69	5	29	22	7	*L*82	5	39	10	31	*L*95	5	11	16	37	*L*108	5	4	17	8	*L*121	5	23	38	21
*L*70	6	7	10	19	*L*83	6	34	33	37	*L*96	6	31	13	29	*L*109	6	24	39	22	*L*122	6	8	30	23
*L*71	27	19	33	38	*L*84	27	11	4	29	*L*97	27	37	17	7	*L*110	27	12	34	10	*L*123	27	22	16	8
*L*72	9	38	4	30	*L*85	9	31	24	7	*L*98	9	29	39	19	*L*111	9	26	11	33	*L*124	9	10	13	22
*L*73	35	30	24	16	*L*86	35	37	12	19	*L*99	35	7	34	38	*L*112	35	32	31	4	*L*125	35	33	17	10
*L*74	25	16	12	13	*L*87	25	29	26	38	*L*100	25	19	11	30	*L*113	25	28	37	24	*L*126	25	4	39	33
*L*75	14	13	26	17	*L*88	14	7	32	30	*L*101	14	38	31	16	*L*114	14	36	29	12	*L*127	14	24	34	4
*L*76	15	17	32	39	*L*89	15	19	28	16	*L*102	15	30	37	13	*L*115	15	21	7	26	*L*128	15	12	11	24
*L*77	20	39	28	34	*L*90	20	38	36	13	*L*103	20	16	29	17	*L*116	20	23	19	32	*L*129	20	26	31	12
*L*78	18	34	36	11	*L*91	18	30	21	17	*L*104	18	13	7	39	*L*117	18	8	38	28	*L*130	18	32	37	26

Proof Process in Coq

Developing a proof in Coq is achieved in two successive steps:

- first a proof is *interactively built* by *the user*;
- then the proof is *automatically checked* by *the system*.

The user does the proof work,
the system simply checks that the proof is actually correct.

Useful Ressources

- Coq web site:
  - Download:

    http://coq.inria.fr/
  - Coq reference manual:

    http://coq.inria.fr/doc/
- Books and Exercices :
  - *Coq'Art* by Y. Bertot and P. Castéran
    (available in French, English and Chinese)

    http://www.labri.fr/perso/casteran/CoqArt/
  - *Software Foundations* par Benjamin C. Pierce, Chris Casinghino, Michael Greenberg,Vilhelm Sjöberg, Brent Yorgey

    http://www.cis.upenn.edu/~bcpierce/sf/

Syntax

Logic	Coq
$\bot$	`False`
$\top$	`True`
$a = b$	`a = b`
$a \neq b$	`a <> b`
$\neg A$	`~ A`
$A \vee B$	`A \/ B`
$A \wedge B$	`A /\ B`
$A \Rightarrow B$	`A -> B`
$A \Leftrightarrow B$	`A <-> B`
$f(x, y, z)$	`(f x y z)`
$\forall xy, P(x, y)$	`forall (x y:A), P x y`
$\exists xy, P(x, y)$	`exists (x:A) (y:B), P x y`

Sequent

Formal deduction systems, used to modelize logics, often rely on a language based on **sequents**. It is a couple (Γ, F) with:

- a multi-set of formula Γ (the order is not relevant, some elements may be repeated) and
- a formula F.

This couple is usually denoted by

$$\Gamma \vdash F$$

Intuitively, a sequent represents the fact that from the hypotheses of Γ, one can deduce F.

Interaction with Coq

In Coq, instead of writing $\{A_1, A_2, \ldots, A_n\} \vdash P$, we write:

```
H_1 : A_1
H_2 : A_2

H_n : A_n
___________________________________ (1/1)
P
```

Natural Deduction

- We use sequents.
- We only handle hypotheses.

Rules for Minimal Logic

$$\frac{}{\Gamma \vdash A} \text{ if } A \in \Gamma$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \text{ Intro} \to \qquad \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B} \text{ Elim} \to$$

Proof of the formula K

$$\dfrac{\dfrac{A, B \vdash A}{A \vdash B \to A}\,\text{Intro} \to}{\vdash A \to B \to A}\,\text{Intro} \to$$

Proof of the formula S

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A \to B \to C, A \to B, A \vdash A \to B \to C \qquad A \to B \to C, A \to B, A \vdash A}{A \to B \to C, A \to B, A \vdash B \to C}\,\text{MP} \qquad \ldots\ \text{X}\ \ldots}{A \to B \to C, A \to B, A \vdash C}\,\text{MP}}{A \to B \to C, A \to B \vdash A \to C}\,\text{Intro} \to}{A \to B \to C \vdash (A \to B) \to A \to C}\,\text{Intro} \to}{\vdash (A \to B \to C) \to (A \to B) \to A \to C}\,\text{Intro} \to
$$

X:

$$
\dfrac{A \to B \to C, A \to B, A \vdash A \to B \qquad A \to B \to C, A \to B, A \vdash A}{A \to B \to C, A \to B, A \vdash B}\,\text{MP}
$$

Rules for ∧ (and)

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q}\ \text{Intro}\ \wedge$$

$$\frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash P}\ \text{Elim}\ \wedge\text{l}$$

$$\frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash Q}\ \text{Elim}\ \wedge\text{r}$$

Rules for $\vee$ (or)

$$\frac{\Gamma \vdash P}{\Gamma \vdash P \vee Q} \text{ Intro } \vee \text{l}$$

$$\frac{\Gamma \vdash Q}{\Gamma \vdash P \vee Q} \text{ Intro } \vee \text{r}$$

$$\frac{\Gamma \vdash P \vee Q \qquad \Gamma, P \vdash R \qquad \Gamma, Q \vdash R}{\Gamma \vdash R} \text{ Elim } \vee$$

Rule for $\neg$ (not)

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\ \text{Intro } \neg$$

$$\frac{\Gamma \vdash \neg A \qquad \Gamma \vdash A}{\Gamma \vdash \bot}\ \text{Elim } \neg$$

Note

We can also write $\neg A$ as $A \rightarrow \bot$.

Rule for $\perp$ (bottom)

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash P}\text{Elim } \perp$$

Rules for ∀ (forall)

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x\, A}\ \forall\ \text{intro } (x \text{ is not free in } \Gamma)$$

$$\frac{\Gamma \vdash \forall x\, A}{\Gamma \vdash A[x \leftarrow t]}\ \forall\ \text{elim}$$

Rules of ∃ (exist)

$$\frac{\Gamma \vdash A[x \leftarrow t]}{\Gamma \vdash \exists x\, A}\ \exists \text{ intro}$$

$$\frac{\Gamma \vdash \exists x\, A(x) \qquad \Gamma, A(x) \vdash B}{\Gamma \vdash B}\ \exists \text{ elim } (x \text{ is not free neither in } \Gamma \text{ nor in } B)$$

Names

Note

Rules are named after their behavior when read *from top to bottom*.

Summary (cheat sheet)

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \text{Intro} \to \qquad \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ \text{Elim} \to \qquad \frac{}{\Gamma \vdash A}\ \text{if } A \in \Gamma$$

$$\frac{\Gamma \vdash P}{\Gamma \vdash P \vee Q}\ \text{Intro} \vee\text{l} \qquad \frac{\Gamma \vdash Q}{\Gamma \vdash P \vee Q}\ \text{Intro} \vee\text{r} \qquad \frac{\Gamma \vdash P \vee Q \qquad \Gamma, P \vdash R \qquad \Gamma, Q \vdash R}{\Gamma \vdash R}$$

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q}\ \text{Intro} \wedge \qquad \frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash P}\ \text{Elim} \wedge\text{l} \qquad \frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash Q}\ \text{Elim} \wedge\text{r}$$

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\ \text{Intro} \neg \qquad \frac{\Gamma \vdash \neg A \qquad \Gamma \vdash A}{\Gamma \vdash \bot}\ \text{Elim} \neg \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash P}\ \text{Elim} \bot$$

$$\frac{\Gamma \vdash \exists x\, A(x) \qquad \Gamma, A(x) \vdash B}{\Gamma \vdash B}\ \begin{array}{c} \exists elim \\ x \notin FV(\Gamma) \cup FV(B) \end{array} \qquad \frac{\Gamma \vdash A[x \leftarrow t]}{\Gamma \vdash \exists x\, A}\ \exists \text{ intro}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x\, A}\ \forall \text{ intro } (x \notin FV(\Gamma)) \qquad \frac{\Gamma \vdash \forall x\, A}{\Gamma \vdash A[x \leftarrow t]}\ \forall \text{ elim}$$

Coq Tactics for Natural Deduction

Provided an inference rule R with n premisses of the shape:

$$\frac{\Gamma_1 \vdash P_1 \;\; \ldots \;\; \Gamma_n \vdash P_n}{\Gamma \vdash G}\ \mathrm{R}$$

Applying the corresponding Coq tactic transforms the current goal:

$$\Gamma \vdash G$$

into n new subgoals:

$$\Gamma_1 \vdash P_1$$

$$\ldots$$

$$\Gamma_n \vdash P_n$$

Axiom Rule

$$\frac{}{\Gamma \vdash A} \text{ if } A \in \Gamma$$

```
 X : A
============
 A
```

Proof completed.

`assumption.` or `apply X.`

Intro Rule for $\rightarrow$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B}\,\text{Intro} \rightarrow$$

```
....
=====================
A -> B
```

```
X : A
=====================
B
```

`intro X.` or `intros.` to introduce several hypotheses.

Elimination Rule for $\rightarrow$

$$\frac{\Gamma \vdash A \rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B} \text{ Elim} \rightarrow$$

```
....
======================
B
```

```
======================
A -> B
```

```
======================
A
```

`cut A.`

An Alternative Eliminate Rule for $\rightarrow$

$$\frac{\Gamma \vdash A \qquad \Gamma, A \vdash B}{\Gamma \vdash B}$$

```
....
======================
B
```

```
...
======================
A
```

```
...
A
======================
B
```

```
assert (A).
```

Another Alternative for Elim $\rightarrow$

$$\frac{\Gamma, H : A \rightarrow B \vdash A}{\Gamma, H : A \rightarrow B \vdash B}$$

```
H: A -> B
=====================
B
```

```
H: A -> B
=====================
A
```

```
apply H.
```

Another Alternative for Elim $\rightarrow$

$$\frac{\Gamma, H : H_1 \rightarrow H_2 \rightarrow B \vdash A}{\Gamma, H : A \rightarrow B \vdash B}$$

```
H: H1 -> H2 -> B
======================
B
```

```
H: H1 -> H2 -> B
======================
H1
```

```
H: H1 -> H2 -> B
======================
H2
```

`apply H.`

Introduction Rule for $\wedge$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \text{Intro}\ \wedge$$

```
....
======================
A /\ B
```

```
======================
A
```

```
======================
B
```

```
split.
```

Elimination Rule for $\wedge$ I

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \text{Elim } \wedge\text{l}$$

```
....
======================        ======================
A                             A /\ B
```

`assert (T: A /\ B);[idtac|elim T;intros;assumption].`

Introduction Rule for $\vee$ r

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \text{Intro } \vee\text{r}$$

```
....
=====================
A \/ B
```

```
=====================
B
```

```
right.
```

Introduction Rule for $\vee$ I

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \text{Intro } \vee\text{I}$$

```
....
=====================          =====================
A \/ B                         A
```

`left.`

Elimination Rule for $\vee$

$$\frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash G \qquad \Gamma, B \vdash G}{\Gamma \vdash G}\ \text{Elim } \vee$$

$$\frac{\Gamma, H : A \vee B, A \vdash G \qquad \Gamma, H : A \vee B, B \vdash G}{\Gamma, H : A \vee B \vdash G}$$

```
....
H : A \/ B
=====================
G
```

```
H : A \/ B
H0 : A
=====================
G
```

```
H : A \/ B
H0 : B
=====================
G
```

`elim H;intro.` or `destruct H.` or `decompose [or] H.`

An Alternative Elimation Rule for $\wedge$

$$\frac{\Gamma, H : A \wedge B, H_0 : A, H_1 : B \vdash G}{\Gamma, H : A \wedge B \vdash G}$$

```
....
H : A /\ B
=====================
G
```

```
H : A /\ B
H0 : A
H1 : B
=====================
G
```

`elim H;intro.` or `destruct H.` or `decompose [and] H.`

Elimination Rule for $\perp$

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash P}\ \mathrm{Elim}\ \perp$$

```
....
=====================
P
```

```
....
=====================
False
```

`exfalso.`

Elimination Rule for $\perp$

$$\dfrac{\overline{\Gamma, H : \bot \vdash \bot}\ \text{Ax}}{\Gamma, H : \bot \vdash P}\ \text{Elim}\ \bot$$

```
....
H : False
=====================
P
```

Proof completed.

`elim H.`

Introduction Rule for $\neg$

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\ \mathsf{Intro}\ \neg$$

```
....
======================
~ A
```

```
H : A
======================
False
```

```
intro.
```

Elimination Rule for $\perp$

$$\cfrac{\cfrac{\Gamma \vdash \neg A \qquad \Gamma \vdash A}{\Gamma \vdash \perp}\ \text{Elim}\ \neg}{\Gamma \vdash G}\ \text{Elim}\ \perp$$

```
....
======================
G
```

```
...
======================
A
```

```
...
======================
~ A
```

```
absurd A.
```

Some Coq tactics (to remember)

- `intro` (introduces hypotheses in the context)
- `assert` (assumes a statement holds)
- `apply` (applies a theorem)
- `exists` (provides a witness for a 'exists')
- `decompose [ex] H`
  (provides witnesses for all 'exists' in the hypothesis H)
- `decompose [and] H` (splits all 'and' of H)
- `decompose [or] H` (carries out case reasoning on all 'or' of H)
- `unfold t` (unfolds the definition of t)
- `simpl`
- `reflexivity`, `symmetry`, `transitivity`
- `rewrite`, replace ... with

Exercises (exercises_logic.v)

- `l0:` $\forall A\ B\ C : \texttt{Prop}, ((A \to B) \land (B \to C)) \to (A \to C)$
- `l1:` $\forall A\ B : \texttt{Prop}, A \lor B \to B \lor A$
- `l2:` $\forall A\ B\ C : \texttt{Prop}, ((A \land B) \to C) \to A \to B \to C$
- `l3:` $\forall A\ B\ C : \texttt{Prop}, (A \to B \to C) \to (A \land B) \to C$
- `l4:` $\forall A\ B\ C : \texttt{Prop}, (A \land (B \lor C)) \to ((A \land B) \lor (A \land C))$
- `l5:` $\forall A\ B\ C : \texttt{Prop}, ((A \land B) \lor (A \land C)) \to (A \land (B \lor C))$

Composing tactics

`tac0; tac1` applies the tactic `tac0` to the current goal, and then the tactic `tac1` to the n subgoals generated by tactic `tac0`.

Some notations

Logic	Coq
$\wedge$	`/\`
$\vee$	`\/`
$\neg$	`~`
$\Rightarrow$	`->`
$\Leftrightarrow$	`<->`
$\forall$	`forall`
$\exists$	`exists`

Parentheses

The arrow is right-associative: $(A \rightarrow B \rightarrow C)$ corresponds to $(A \rightarrow (B \rightarrow C))$.

Prop vs bool

- **Prop** is the type of propositions that we prove.
  Examples: `True`, `False`, 2<3, `forall x, x=x`,
  `leb 2 3 = true`.
- **bool** is the type of boolean values, which can be used in programs (in `if-then-else` constructs).
  Examples: `true`, `false`, `leb 2 3`

```
Compute (leb 2 3).
    = true
    : bool
Compute (2 <= 3).
    = 2 <= 3
    : Prop
```

One may prove that `forall b: bool, b=true \/ b=false`.
But we do not always consider that `forall P, P \/ ~P` holds.

Intuitionist vs Classical Logic

The following formulae are valid in classical logic:

excluded middle $P \vee \neg P$

elimination of double negation $\neg\neg P \rightarrow P$

Pierce's law $((P \rightarrow Q) \rightarrow P) \rightarrow P$

These propositions acknowledge that there are some proofs which do not build an object satisfying the considered statement.
Some mathematicians rejected these propositions:

- Brouwer,
- Heyting, ...

A proof is said to be constructive if it does not require the excluded middle principle.

Properties of intuitionist logic

Disjunction

From a proof of $A \vee B$, we can extract a proof of A or a proof of B.

Witness

From a proof of $\exists x, A(x)$ we can extract a witness t and a proof of $A(t)$.

Example of a classic proof

Let us show that:

$$\exists x, y \notin \mathbb{Q}, x^y \in \mathbb{Q}$$

Consider $\sqrt{2}^{\sqrt{2}}$.

a If $\sqrt{2}^{\sqrt{2}} \in \mathbb{Q}$.
  We choose $x = \sqrt{2}$ et $y = \sqrt{2}$.

b Otherwise $\sqrt{2}^{\sqrt{2}} \notin \mathbb{Q}$.
  We choose $x = \sqrt{2}^{\sqrt{2}}$ and $y = \sqrt{2}$.

$$x^y = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2} \times \sqrt{2}} = \sqrt{2}^2 = 2 \in \mathbb{Q}$$

A theorem in analysis actually states that $\sqrt{2}^{\sqrt{2}}$ is irrational and that we must choose the case “b“ , but the proof relying on the excluded middle **does not tell this**.

Classic propositional logic

We add the rule:

Reductio ad absurdum

$$\frac{\Gamma, \neg P \vdash \bot}{\Gamma \vdash P}\,\text{RAA}$$

Note

We could have written:

$$\frac{\Gamma \vdash \neg\neg P}{\Gamma \vdash P}\,\text{RAA'}$$

Double Negation

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\neg A, \neg A \vdash \neg\neg A} \qquad \dfrac{}{\neg\neg A, \neg A \vdash \neg A}}{\neg\neg A, \neg A \vdash \bot}\,\text{elim } \neg}{\neg\neg A \vdash A}\,\text{RAA}}{\vdash \neg\neg A \to A}\,\text{intro } \to
$$

Law of Excluded Middle

$$
\dfrac{
\dfrac{
\neg(A \vee \neg A) \vdash \neg(A \vee \neg A)
\qquad
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\neg(A \vee \neg A), A \vdash A}
}{\neg(A \vee \neg A), A \vdash A \vee \neg A}\,\text{intro } \vee\text{l}
\qquad
\dfrac{}{\neg(A \vee \neg A), A \vdash \neg(A \vee \neg A)}
}{\neg(A \vee \neg A), A \vdash \bot}
}{\neg(A \vee \neg A) \vdash \neg A}\,\text{intro } \bot
}{\neg(A \vee \neg A) \vdash A \vee \neg A}\,\text{intro } \vee\text{r}
}{\neg(A \vee \neg A) \vdash \bot}
}{\vdash A \vee \neg A}\,\text{RAA}
$$

Pierce law

$$
\begin{prooftree}
\AxiomC{}
\UnaryInfC{$(A \to B) \to A, \neg A \vdash \neg A$}
\AxiomC{}
\UnaryInfC{$(A \to B) \to A, \neg A \vdash (A \to B) \to A$}
\AxiomC{}
\UnaryInfC{$\Gamma, \neg A, A \vdash A$}
\AxiomC{}
\UnaryInfC{$\Gamma, \neg A, A \vdash \neg A$}
\RightLabel{elim $\neg$}
\BinaryInfC{$(A \to B) \to A, \neg A, A \vdash \bot$}
\RightLabel{elim $\bot$}
\UnaryInfC{$(A \to B) \to A, \neg A, A \vdash B$}
\RightLabel{intro $\to$}
\UnaryInfC{$(A \to B) \to A, \neg A \vdash A \to B$}
\RightLabel{elim $\to$}
\BinaryInfC{$(A \to B) \to A, \neg A \vdash A$}
\RightLabel{elim $\bot$}
\BinaryInfC{$(A \to B) \to A, \neg A \vdash \bot$}
\RightLabel{RAA}
\UnaryInfC{$(A \to B) \to A \vdash A$}
\RightLabel{Intro $\to$}
\UnaryInfC{$\vdash ((A \to B) \to A) \to A$}
\end{prooftree}
$$

Exercises

```
Definition EM := (forall A:Prop, A\/~A).
Definition DN := (forall A:Prop, ~~A->A).
Definition contrap := (forall A B:Prop,(~B->~A) -> (A->B)).
Definition Pierce := (forall A B:Prop, ((A->B)->A)  -> A).
Definition neg_impl := forall P Q:Prop, (P->Q)->(~P\/Q).

Definition all := [EM; DN; contrap; Pierce; neg_impl].

Lemma all_equiv :
  forall x y, In x all -> In y all -> x <-> y.
```

Coq is an intuitionist system

By default, Coq works in intuitionist logic.
To use the excluded middle, we must explicit require it by the command:

```
Require Export Classical.
```

Introduction to currying

Prove that the following formula holds:

$$(A \to (B \to C)) \leftrightarrow (A \wedge B \to C)$$

Note: As $\to$ is right-associative, we could have writtten:

$$(A \to B \to C) \leftrightarrow (A \wedge B \to C)$$

Currying

Definition

Currying consists in transforming a function with takes several arguments into a function with a single argument returning a function which takes as arguments all the remaining arguments.

Note: this operation is named after Haskell Curry (1900-1982).

Example

In OCaml

Instead of writing:

```
# let f(x,y) = x + y;;
val f : int * int -> int = <fun>
```

We write:

```
# let f x = fun y -> x + y;;
val f : int -> int -> int = <fun>
```

or

```
# let f x y = x + y;;
val f : int -> int -> int = <fun>
```

Example

In Coq

We usually curry all functions and statements. We shall rather write

```
Lemma foo : forall p q : R, p > 0 -> q > 0 -> p*q > 0.
```

than:

```
Lemma foo : forall p q : R, p > 0 /\ q > 0 -> p*q > 0.
```

Why use currying?

To be able to carry out *partial applications* more easily.

Transition: Sorts

A sort is a type for types.
Propositions A, B, etc. are types (those of their proof terms).
These types are of type `Prop`.
We say that A, B, etc. are of sort `Prop`.
On the other side, boolean `bool`, integers `nat` are types whose type is `Set`.
`Set` et `Prop` are of type `Type`.

`Set` : where we compute

`Prop` : where we reason

Inductive Datatypes

Coq relies on a formalism called the *Calculus of Inductive Constructions*.

Main features

- Based on type theory
- Higher-order logic (functions are first-class citizens)
- Data-structures can be represented by Inductive Types
- There is no distinction between terms and types:
  `bool` is the type of `true` and `false` and `bool` is also a term of type `Set`.

Inductive Definitions

Inductive definitions consist in:

- providing the basic elements,
- providing rules to build new elements from the already-known elements.

Examples

- Natural Numbers
- Lists, Trees, ...
- Inductive Predicates

Peano integers (natural numbers)

1. The element `zero`, denoted by 0 is a natural number.
2. If n is a natural number, then its successor $S(n)$ is a natural number.

Alternative notation: `P ::= O | S P`

Coq vs OCaml

Coq

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.
```

Caml

```
type nat =
  O
| S of nat
```

Recursion/Induction on nat

```
forall P : nat -> Prop,
       P 0 ->
       (forall n : nat, P n -> P (S n)) ->
       forall n : nat, P n
```

- Universal Quantification on a property `P: nat -> Prop`
- Conclusion : `forall n :  nat, P n`
- 2 cases:
  1. one base case
  2. one induction case

Induction Principle

Applying the induction principle: `elim n` generates 2 subgoals.

```
forall n : nat, P n
```

```
----------------
P 0
```

```
n : nat
IHn : P(n)
----------------
P(S(n))
```

In Coq, an induction principle is automatically generated after each inductive definition.

Constructors are distinct (free)

To use the specific property, we can use the tactic `discrimate`.

```
Lemma true_false : true<>false.
intro H; discriminate.
Qed.
```

```
Lemma zero_succ : forall n:nat, ~(S n)=O.
intros n H; discriminate H.
Qed.
```

Constructors are injective

To use the specific property, we can use the tactic `injection`.

```
Lemma test_injection: forall x y, S x = S y -> x=y.
Proof.
intros.
injection H.
intro.
assumption.
Qed.
```

Aparté: equality in Coq

```
Check eq.
eq  :  forall A : Type, A -> A -> Prop

Check refl_equal.
refl_equal : forall (A : Type) (x : A), x = x
```

- It is a polymorphic type `(A:Type)`. Thus the equality relation is generic and its first argument is the type of elements to be compared.
- Equality is a reflexive, symmetric and transitive relation. These properties can be used through the tactics `reflexivity`, `symmetry` and `transitivity t`.

Reasoning about equality

Principle of Leibniz equality

```
Check eq_ind.
  eq_ind : forall (A : Type) (x : A) (P : A -> Prop),
P x -> forall y : A, x = y -> P y
```

Tactics for equality

- `rewrite`
- `rewrite in`
- `replace with`
- `replace with in`
- `subst`

Recursive Function: fixpoint definition

Adding natural numbers:

```
Fixpoint plus (n m : nat) {struct n} : nat :=
match n with
  | O   => m
  | S p => S (plus p m)
end.

Eval compute in (plus O 56).
Eval compute in (plus 12 67).
Eval compute in (plus 67 12).
```

Termination

In Coq, functions must always terminate!

The annotation {`struct n`} tells the system which argument decreases structurally at each recursive call. Recursive calls are carried out on strict subterms. This enables to ensure the termination of the function.

Example

```
Fixpoint bar (n:nat) : nat := bar n.
```

This definition is rejected by the system and leads to the following error:

```
Error: Recursive definition of bar is ill-formed.
```

Otherwise we get ...

... a contradiction!

```
Fixpoint foo (b :bool) : bool := negb (foo b).
```

We have `foo true = negb (toto true)`

- If `foo true = false` then `foo true = true`.
- If `foo false = true` then `foo false = false`.

Functions are total!

In Coq, all functions must be total!

What to do when we need to non total function?

We can use the `option` type:

```
Inductive option (A:Type) : Type :=
  | Some : A -> option A
  | None : option A.
```

The depth of filtering can be more than 1.
This is useful for checking parity or computing Fibonacci:

```
Fixpoint even (n:nat) : Prop :=
match n with O => True
| (S O) => False
| (S (S p)) => pair p
end.

Eval compute in (even 8789).
Eval compute in (even 8790).
```

Example: Fibonacci

```
Fixpoint fib n {struct n}  :=
  match n with
   O => 1
 | S O => 1
 | S ((S p) as p1) => fib p + fib p1
 end.
```

Computing with Functions

Once a fixpoint definition is entered, additional computational rules are added to the system (one by match branches).

- tactics `compute`, `vm_compute` , `simpl` are useful to compute in the current goal.
- `simpl in H`, to compute in a specific hypothesis H.

Example:

Reductions for `plus`:

$$\begin{array}{rcl} \texttt{plus O m} & \longrightarrow_\iota & \texttt{m} \\ \texttt{plus (S n) m} & \longrightarrow_\iota & \texttt{S (plus n m)} \end{array}$$

Associativity of addition $+$

```
Lemma plus_assoc: forall n m p : nat,
   (plus (plus n m) p) = (plus n (plus m p))
```

Proof by induction : `intros n m p; elim n.`

1. Base case : `0 + (m + p) = 0 + m + p`
   - simplification; reflexivity of equality.
2. induction case:...

Three Examples of Proofs

- proving a statement about natural numbers (by induction)

$$6 \quad | \quad n^3 - n$$

- sum of the n first integers (interactive example)

$$2 * \Sigma_{i=0}^{n} \ i = n * (n+1).$$

- making an amount greater than 8 with coins of 3 and 5

$$\forall m : \texttt{nat}, \exists i : \texttt{nat}, \exists j : \texttt{nat}, 8 + m = 5 * i + 3 * j.$$

Tarski Semantics

Truth Tables

p	0	0	1	1
q	0	1	0	1
$p \wedge q$	0	0	0	1
$p \vee q$	0	1	1	1
$p \rightarrow q$	1	1	0	1

Heyting-Kolmogorov Semantics

Heyting-Kolmogorov semantics consist in providing a functional interpretation to proofs.

- A proof of $A \to B$ is a *function* which, from a proof of A produces a proof of B.
- A proof of $A \wedge B$ is a *couple* composed of a proof of A and of a proof of B.
- A proof of $A \vee B$ is a couple (i, p) with ($i = 0$ and p a proof of A) or ($i = 1$ and p a proof of B).
- A proof of $\forall x.A$ is a function which, for each object t builds an object of type $A[x := t]$.

This interpretation consists in *computing with proofs*. This is very close to functional programming and λ-calculus.

Example

Example from a type point of view:

If H has type $A \to B$ et H' has type A

then

$H\ H'$ has B.

Example from a proof point of view:

If H is a proof of $A \to B$ and H' is a proof of A

then

$H\ H'$ is a proof of B.

Curry-Howard Isomorphism I

logic	programming
$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}$	$\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash (\text{fun } x : A \mapsto t) : A \to B}$
$\dfrac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$	$\dfrac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B}$

Curry-Howard Isomorphism II

$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$	$\dfrac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash (a, b) : A \times B}$
$\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}$	$\dfrac{\Gamma \vdash t : A \times B}{\Gamma \vdash \mathit{fst}\ t : A}$
$\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$	$\dfrac{\Gamma \vdash t : A \times B}{\Gamma \vdash \mathit{snd}\ t : B}$

Curry-Howard Isomorphism III

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \quad \Bigg| \quad \frac{\Gamma \vdash a : A}{\Gamma \vdash inl\ a : A + B}$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \quad \Bigg| \quad \frac{\Gamma \vdash b : B}{\Gamma \vdash inr\ b : A + B}$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}$$

$$\frac{\Gamma \vdash m : A \vee B \quad \Gamma, x : A \vdash t : C \quad \Gamma, x : B \vdash u : C}{\Gamma \vdash case\ m\ of\ inl(a) => t \mid inr(a) => u : C}$$

Simplification Rules

$$\begin{array}{rcl} fst(A,B) & = & A \\ snd(A,B) & = & B \\ case\ (inl\ m)\ of\ inl(a) => t | inr(a) => u & = & t[x := m] \\ case\ (inr\ m)\ of\ inl(a) => t | inr(a) => u & = & u[x := m] \end{array}$$

Curry-Howard Isomorphism

Logic	λ-**calculus/programming**
formula	type
proof	term/program
proof checking	type checking
proof normalization	β-reduction

Example : building a proof term

Building a proof, as a λ-term for the formula: $A \to (A \to B) \to B$.

- Make it a closed formula,i.e. $\forall AB : Prop, A \to (A \to B) \to B$.
- We must now build a λ-term whose type is:
  $\forall A\ B : Prop, A \to (A \to B) \to B$.
  - This shall be a function whose arguments are A, B, H_1 and H_2, its body of type B must be built from A, B, H_1 et H_2.
    `fun (A:Prop) (B:Prop) (H1:A) (H2:A->B) => ...:B`
  - A way to build a term of type B is to take the term H_1 of type A and to apply the (functional) term H_2 to it. This yields the application $(H_2\ H_1)$.
  - A possible proof term for $\forall AB : Prop, A \to (A \to B) \to B$ is `fun (A:Prop) (B:Prop) (H1:A) (H2:A->B) => (H2 H1)`.

Exercises

Assume we have the following terms available:

```
and_ind   : forall A B P : Prop,
              (A -> B -> P) -> A /\ B -> P
conj      : forall A B : Prop, A -> B -> A /\ B
or_ind    : forall A B P : Prop,
              (A -> P) -> (B -> P) -> A \/ B -> P
or_introl : forall A B : Prop, A -> A \/ B
or_intror : forall A B : Prop, B -> A \/ B
```

Build a proof term for the following statements:

1. $(A \to B \to C) \to (A \to B) \to A \to C$
2. $(A \to B) \to (B \to C) \to (A \to C)$
3. $A \land B \to B \land A$
4. $A \land B \to A \lor B$
5. $A \lor B \to B \lor A$

Checking Proof Automation

Proof checking

We can use arbitrary tool to generate proofs. In the end, it should produce a proof term (a trace), which is type-checked by the system to ensure that it really is a proof of the statement at stake.

Examples of Proof Automation (`lia_example.v`)

- Automated Tactics for logic: intuition, first_order, ...
- Decision Procedures for numbers: lia
- Going further: using external SMT solvers and importing traces

In Coq, some logic connectives are primitive and others are defined using inductive types:

Defined ones:

- $\top$
- $\bot$
- $\wedge$
- $\vee$
- $\exists$
- $\neg A \equiv A \Rightarrow \bot$

Primitive ones:

- $\forall$
- $\Rightarrow$

True

```
Inductive True : Prop :=
  I : True.
```

```
True_ind
     : forall P : Prop, P -> True -> P
```

Useless...

False

```
Inductive False : Prop := .

False_ind
     : forall P : Prop, False -> P
```

Example

```
Theorem ex_falso_quodlibet : forall (P:Prop),
   False -> P.
Proof.
  intros P F.
  inversion F.
Qed.
```

AND

```
Inductive and (A B : Prop) : Prop :=
  conj : A -> B -> and A B.
```

```
   and_ind
      : forall A B P : Prop,
    (A -> B -> P) -> and A B -> P
```

OR

```
Inductive or (A: Prop) (B : Prop) : Prop :=
 | or_introl : A -> or A B
 | or_intror : B -> or A B.
```

```
or_ind
     : forall A B P : Prop,
  (A -> P) -> (B -> P) -> or A B -> P
```

Existential quantification

```
Inductive ex (A: Set) (P: A -> Prop) : Prop :=
 | ex_intro : forall x : A, P x -> ex A P.
```

```
ex_ind
     : forall (A : Set) (P : A -> Prop) (P0 : Prop),
       (forall x : A, P x -> P0) -> ex A P -> P0
```

Equality

```
Inductive eq (A:Type) (x:A) : A -> Prop :=
   refl_equal : eq A x x.

eq_ind: forall (A:Type)(x:A)(P:A->Prop),
      P x -> forall y: A, x=y -> P y
```

Function vs Predicate

We can use an inductive *predicate* to describe a function in a Prolog-like style:

Example: Syracuse

$$U_0 = N \qquad\qquad U_{n+1} = \begin{cases} \frac{U_n}{2} & \text{if} \quad U_n \quad \text{is even} \\ 3U_n + 1 & \text{if} \quad U_n \quad \text{is odd} \end{cases}$$

```
Inductive syracuse (N:nat) : nat -> nat -> Prop :=
  done : syracuse N 0 N
| even_case : forall n p,even p ->
   syracuse N n p -> syracuse N (S n) (div2 p)
| odd_case : forall n p , odd p ->
   syracuse N n p -> syracuse N (S n) (S(p+p+p)).
```

Syracuse :how to carry out a proof?

```
Lemma example : syracuse 15 1 46.
replace (46) with (S(15+15+15)) by reflexivity.
apply odd_case.
repeat constructor.
constructor.
Qed.
```