Proseminar on computer-assisted mathematics

Session 8 - The Natural Number Game

```
example (P Q R : Type) : (P → (Q → R)) → ((P → Q) → (P → R)) :=

begin
  intro h,
  intro hPQ,
  intro p,
  apply h p,
  apply hPQ,
  exact p,
```

39:8: goal

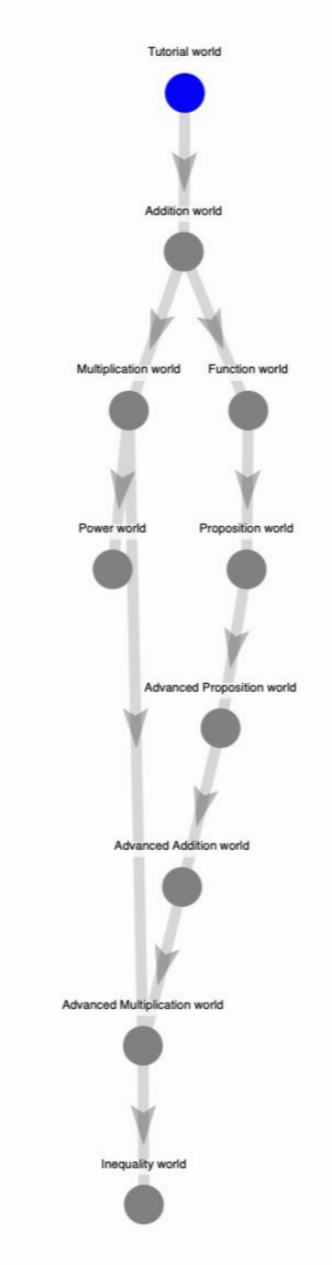
```
P Q R : Type,
h : P → Q → R,
hPQ : P → Q,
p : P
⊢ R
```

$h(p) : (Q \to R)$

Florent Schaffhauser
Heidelberg University, Summer semester 2023

Today we will:

- Play the Natural Number Game, created by Kevin Buzzard and Mohammad Pedramfar.
- Learn a lot of useful Lean tactics along the way!



The Natural Number Game is available online at

https://www.ma.imperial.ac.uk/~buzzard/xena/natural_number_game/

It consists of ten worlds, each one with various levels (and of course, final bosses).

It is a pretty addictive game, as you will see!

The objective is to learn about tactics through a game-like approach. Keep your eye on the goal and try to close it using the assumptions at your disposal.

reflexivity, exact, apply,
use, intro, cases, split,
have, by_cases, ...

However, try to also focus on the mathematics.

It is for instance recommendable to write a pen-and-paper proof of the result before formalizing the proof!

Finally, it is also advisable to study the syntax of the statements, and to take notes of the way each tactic is used, for future record.

If you want to put your first tactics at use and prove basic propositions in Lean, check out the modus ponens file in the Practice folder of the GitHub repository of the seminar!

Modus ponens

Let us use tactic mode to write a proof of the rule of deductive reasoning known as *modus ponens*, which says that if we have a proof of proposition `P` and a proof of the implication `P → Q`, then we have a proof of the proposition `Q`.

```
def MP {P Q : Prop} (hP : P) (hPQ : P → Q) : Q :=
begin
  apply hPQ,
  exact hP,
end
```

The proof we give can be understood as follows. First, let us make it clear that `P` and `Q` are propositions, that we have a proof of `P` and of `P → Q`, and that our goal is to prove `Q`.

Since by assumption we have a proof of the implication `P → Q`, it suffices to prove `P`. This is what the `apply` tactic enables us to do. More precisely, the term `hPQ`, being of type `P → Q`, is a function from `P` to `Q`, so it sends a term of type `P` (i.e. a proof a `P`) to a term of type `Q`, i.e. a proof of `Q`.

Now, after the `apply` tactic, the goal is changed to `P`. And since, again by assumption, we have a proof of `P`, we can close the goal using the `exact` tactic.

Alternately, we can write the proof using just the `exact` tactic, because `hPQ hP` is the result of applying the function `hPQ : P → Q` to the term `hP`, which is of type `P`, so it is a term of type `Q`.

```
def MP_bis {P Q : Prop} (hP : P) (hPQ : P → Q) : Q :=
begin
  exact hPQ hP,
end
```

There, you will also see how different ways of stating the result can affect the formal proof.

```
def MP_no_var {P Q : Prop}: P → (P → Q) → Q :=
begin
  intro hP,
  intro hPQ,
  apply hPQ,
  exact hP,
end
```

```
def MP_no_var_bis {P Q : Prop}: P ∧ (P → Q) → Q :=
begin
  intro h,
  cases h with hP hPQ,
  apply hPQ,
  exact hP,
end
```

$$\frac{hP : P \qquad hPQ : P \to Q}{hP\ hPQ : Q}$$