Report for the *proof assistant project*

Rémi Germe

This report is written without giving much context and will not provide a clear explanation to a wandering person. If the person is not wandering and still doesn't know what to expect in this repository, I can only strongly suggest that they visit the course website by Samuel Mimram (link). They can then check the project template (link). While coding, I often referred to other labs, especially lab 3 on lambda calculus.

1 What was implemented in the project

I'll review what was implemented of course, but also some little details of the implementation.

General considerations. Everywhere throughout the project, I kept ASCII characters for both input and output strings instead of fancy UTF-8 characters such as 'λ' and '$\Rightarrow$'.

1.1 Simple types

Type inference for a simply typed calculus. This part was pretty straightforward. I also relied on labs 1 and 3 which were good inspirations. For example, I defined `infer_type` and `check` as mutual recursive functions, in order to use `check` inside `infer_type` to (perhaps) simplify it. Everything was implemented.

Interactive prover. This part was fun ! Note that I did not implement the "save commands to file" feature, but everything else was implemented and all proofs were done (see `simple/proofs/`). Using the prover at this point was fun and gave me a better intuition on tactics and their use (`intro`, `elim`, ...). I think these benefits were provided by the interactive aspect of the prover.

Natural numbers. Everything was implemented.

Small extensions. Unfortunately, I did not add any of the proposed extensions. I planned to do it after dependent types, but I had no time left by the deadline.

1.2 Dependent types

All mandatory expressions were supported by the prover.

```
type expr =
  | Type
  | Var of var
  | App of expr * expr
  | Abs of var * expr * expr
  | Pi of var * expr * expr
  | Nat
  | Z
  | S of expr
  | Ind of expr * expr * expr * expr
  | Eq of expr * expr
  | Refl of expr
  | J of expr * expr * expr * expr * expr
```

However, not all the mandatory proofs have been made : associativity and commutativity of both the addition and the multiplication are still missing to this day. The rest can be found in `dependent/proofs/dnat.proof`.

Once again, none of the proposed extensions were implemented due to a lack of time before the deadline.

2 Difficulties encountered

Time management. More could have been done if time had been managed more wisely on my side. This isn't that trivial as the first period was very busy.

2.1 Simple types

Save commands to file feature. One easy solution would have been to dump every command in a log file created by the prover. However, I only wanted to log valid commands. This would have been possible by writing to the file only when returning successfully from a command. However the naive implementation would have required to put instructions on every case of the prover, so I did not implement it as I found it very inelegant.

Elim rule for natural numbers. I skipped this specific part and only came back to it later. I struggled with the treatment of the recursion hypothesis. Indeed, if I asked the user to prove a `TAbs(Nat, TAbs(a, a))` (with `a` denoting the goal), he would later introduce two variables corresponding to a natural and the recurrence hypothesis. But I had to know these two variables names in order to generate a correct `Rec` instance. The solution I came up with consists in proving the same goal type (ie `a`) and automatically introducing the recurrence hypothesis `prec : a` and the natural `nrec : Nat` with fixed names.

```
...
print_endline "base case : ";
let t = prove env a in
```

```
print_endline "rec case : ";
let u = prove (("prec", a) :: ("nrec", Nat) :: env) a in
Rec (Var x, t, "nrec", "prec", u)
...
```

2.2 Dependent types

Oopsies. I made a few huge mistakes that - I hope - were all hopefully fixed fast (for example forgetting to use λ-reduction when trying to reduce expressions... hem).

Term or type ? In general I was easily confused between expressions of which I should infer the type and expressions which already represented a given type. I struggled with the `infer` function which performs type inference, especially when dealing with `Ind` expressions (the induction principle). Paper and small examples were required, but in the end I have gained a better understanding.

Properties of addition (and multiplication). For previous proofs (definitions of `pred` and `add`, proofs `Seq`, `zadd`, `addz`) it was possible to write it one-shot - potentially with a bit of trial-and-error with the prover. But for the proofs of associativity and commutativity, it became quite difficult to write them down at once. Unlike Agda, the prover is not interactive (I have not extended it) and there is no "hole" possibility to slowly progress through the proof. This feature would have been very helpful.

3 Implementation choices

Tests and witnesses for the simple types. Various tests and some witnesses can be found below the prover itself in `simple/prover.ml`. They can be printed by setting `let debug = true` at the top of the file.

Normalizing expressions. Rather than defining a recursive function `normalize : context -> expr -> expr` which would call itself, I defined a function `red : context -> expr -> expr option` which returns `None` if the expression is already in normal form, or a reduced expression if possible. Then `normalize` is simply :

```
let rec normalize ctx e =
  match red ctx e with
  | None -> e
  | Some u -> normalize ctx u
```

It turns out that `red` is pretty verbose and that choice might have complexified the overall normalization process.

4 Possible extensions

Those proposed would be a great start.

5 Conclusion

It was nice to use Ocaml for a bigger project than just a single lab. I really loved the interactive section, and wish I had the time to make the dependent types prover also interactive. The dependent types part really forced me to manipulate these expressions and gain a better understanding of “terms” vs “types”, which sometimes caused me trouble even using Agda.