Constructive Logic (15-317), Fall 2017
Assignment 6: Proof Checking and Certification

Frank Pfenning

Due Thursday, October 26, 2017

This assignment is due at the beginning of class on the above date and must be submitted electronically via autolab. Submit your homework as a **tar** archive containing only `check.sml` and `g4ip.sml`.

There is no written portion of this assigment, so you do not have to submit a pdf file.

Theorem provers and decision procedures are generally complicated beasts. This increases the likelihood of errors in code that we would like to hold to the highest standards of correctness. What to do?

One technique is for theorem provers to produce *proof certificates* (in case they succeed, of course) that can be externally checked. This will represent a step forward in particular if we can make such checkers small. Very small.

The goal of this assignment is for you (1) to implement a proof checker (worth 20 points) and then (2) instrument your G4ip decision procedure to produce proof objects that can be certified by your checker (and our checker!) (worth 30 points).

Since propositions correspond to types and constructive proofs correspond to programs, you are also implementing a type checker for a small functional programming language. It has the form of a so-called *bidirectional* checker.

1 Implementing a proof checker

We present the syntax and checking system for a language of proof terms.

$$
\begin{array}{lrcl}
\text{Checkable terms} & N & ::= & \mathsf{fn}\ x \Rightarrow N \mid (N_1, N_2) \mid (\,) \mid \mathsf{inl}\ N \mid \mathsf{inr}\ N \mid R \\
& & & \mid (\mathsf{case}\ R\ \mathsf{of}\ \mathsf{inl}\ x_1 \Rightarrow N_1 \mid x_2 \Rightarrow N_2) \mid \mathsf{abort}\ R \\
& & & \mid \mathsf{let}\ x : A = N_1\ \mathsf{in}\ N_2 \\
\text{Synthesizing terms} & R & ::= & x \mid N\ R \mid \mathsf{fst}\ R \mid \mathsf{snd}\ R \\
& & & \mid (N : A) \\
\text{Ordered contexts} & \Omega & ::= & \cdot \mid (x{:}A{\downarrow}) \cdot \Omega
\end{array}
$$

We are using *ordered context* Ω here for two reasons: (1) they are likely closer to the implementation than sets would be, and (2) they allow us to check terms such as $(\mathsf{fn}\ x \Rightarrow \mathsf{fn}\ x \Rightarrow x) : A \supset B \supset B \downarrow$ without ambiguity. An occurrence of a variable always refers to the *leftmost* variable in the context with the same name.

The checking and synthesis judgments are the following, defined in Figure 1

Checking: $\Omega \vdash N : A \uparrow$

Synthesis: $\Omega \vdash R : A \downarrow$

Given types `term` for terms and `prop` for propositions as defined in the starter code, operationalize these as two mutually recursive functions in ML

```
val check : term -> prop -> bool
val synth : term -> prop option
```

where `check N A` returns `true` if $\cdot \vdash N : A \uparrow$ and `false` otherwise, and `synth R` returns `SOME(A)` if $\cdot \vdash R : A \downarrow$ and `NONE` otherwise. Note that the contexts here are empty at the interface, but that you will need to maintain them internally.

Your code must respect the distinction between checkable terms N and synthesizing terms R, even though in ML they are represented within the same type `term`. This representation makes it much easier to write out concrete examples by hand.

Your implementation does not need to print error messages, although it may be a significant aid in debugging if it does. Some printing functions are provided with the starter code for this homework.

$$\frac{}{(x{:}A{\downarrow}) \cdot \Omega \vdash x : A \downarrow}\ \mathsf{var}_{=} \qquad \frac{y \neq x \quad \Omega \vdash x : A \downarrow}{(y{:}A{\downarrow}) \cdot \Omega \vdash x : A \downarrow}\ \mathsf{var}_{\neq}$$

$$\frac{(x{:}A{\downarrow}) \cdot \Omega \vdash N : B \uparrow}{\Omega \vdash (\mathsf{fn}\ x \Rightarrow N) : A \supset B \uparrow} \supset I \qquad \frac{\Omega \vdash R : A \supset B \downarrow \quad \Omega \vdash N : A \uparrow}{\Omega \vdash R\,N : B \downarrow} \supset R$$

$$\frac{\Omega \vdash N_1 : A \uparrow \quad \Omega \vdash N_2 : B \uparrow}{\Omega \vdash (N_1, N_2) : A \wedge B \uparrow} \wedge I \qquad \frac{\Omega \vdash R : A \wedge B \downarrow}{\Omega \vdash \mathsf{fst}\,R : A \downarrow} \wedge E_1 \qquad \frac{\Omega \vdash R : A \wedge B \downarrow}{\Omega \vdash \mathsf{snd}\,R : B \downarrow} \wedge E_2$$

$$\frac{}{\Omega \vdash (\,) : \top \uparrow} \top I \qquad \text{no } \top E \text{ rule}$$

$$\frac{\Omega \vdash N : A \uparrow}{\Omega \vdash \mathsf{inl}\,N : A \vee B \uparrow} \vee I_1 \qquad \frac{\Omega \vdash N : B \uparrow}{\Omega \vdash \mathsf{inr}\,N : A \vee B \uparrow} \vee I_2 \qquad \frac{\Omega \vdash R : A \vee B \downarrow \quad (x{:}A{\downarrow}) \cdot \Omega \vdash N_1 : C \uparrow \quad (y{:}B{\downarrow}) \cdot \Omega \vdash N_2 : C \uparrow}{\Omega \vdash (\mathsf{case}\ R\ \mathsf{of}\ \mathsf{inl}\,x \Rightarrow N_1 \mid \mathsf{inr}\,y \Rightarrow N_2) : C \uparrow} \vee E$$

$$\text{no } \bot I \text{ rule} \qquad \frac{\Omega \vdash R : \bot \downarrow}{\Omega \vdash \mathsf{abort}\,R : C \uparrow} \bot E$$

$$\frac{\Omega \vdash R : A' \downarrow \quad A = A'}{\Omega \vdash R : A \uparrow} \downarrow\uparrow \qquad \frac{\Omega \vdash N : A \uparrow}{\Omega \vdash (N : A) : A \downarrow} \uparrow\downarrow$$

$$\frac{\Omega \vdash N : A \uparrow \quad (x{:}A{\downarrow}) \cdot \Omega \vdash N' : C \uparrow}{\Omega \vdash (\mathsf{let}\,x : A = N\ \mathsf{in}\ N') : C \uparrow}\ \mathsf{let}$$

Figure 1: Type checking and synthesis judgments for proof terms

2 Instrumenting a decision procedure

Recall that in Assignment 5 we asked you to implement a function

```
val decide : prop -> bool
```

In this assignment, modify your implementation to satisfy the interface

```
val certify : prop -> term option
```

where `certify A` returns `SOME(N)` with $\cdot \vdash N : A \uparrow$ if a proof of A exists, and `NONE` if no proof of A exists.

We recommend that you follow the judgment $\Gamma \longrightarrow N : A$ given in Figure 2 to assign proof terms to G4ip. This judgment uses contexts where every antecedent is annotated with a synthesizing proof term R, and the succedent with a checkable proof term N

$$\Gamma \quad ::= \quad \cdot \mid \Gamma, R : A$$

We have deliberately left some gaps for you to fill in the nested left rules, indicated by "??".

Identity

$$\frac{R : P \in \Gamma}{\Gamma \longrightarrow R : P}\ \mathsf{id}_P$$

Ordinary Rules

$$\frac{}{\Gamma \longrightarrow () : \top}\ \top R \qquad \frac{\Gamma \longrightarrow N : C}{\Gamma, R : \top \longrightarrow N : C}\ \top L$$

$$\frac{\Gamma \longrightarrow N_1 : A \quad \Gamma \longrightarrow N_2 : B}{\Gamma \longrightarrow (N_1, N_2) : A \wedge B}\ \wedge R \qquad \frac{\Gamma, (\mathsf{fst}\, R : A), (\mathsf{snd}\, R : B) \longrightarrow N : C}{\Gamma, R : A \wedge B \longrightarrow N : C}\ \wedge L$$

(no $\bot R$ rule)

$$\frac{}{\Gamma, R : \bot \longrightarrow \mathsf{abort}\, R : C}\ \bot L$$

$$\frac{\Gamma \longrightarrow N : A}{\Gamma \longrightarrow \mathsf{inl}\, N : A \vee B}\ \vee R_1 \qquad \frac{\Gamma, x : A \longrightarrow N_1 : C \quad \Gamma, y : B \longrightarrow N_2 : C}{\Gamma, R : A \vee B \longrightarrow (\mathsf{case}\, R\, \mathsf{of}\, \mathsf{inl}\, x \Rightarrow N_1 \mid \mathsf{inr}\, y \Rightarrow N_2) : C}\ \vee L$$

$$\frac{\Gamma \longrightarrow N : B}{\Gamma \longrightarrow \mathsf{inr}\, N : A \vee B}\ \vee R_2$$

$$\frac{\Gamma, x : A \longrightarrow N : B}{\Gamma \longrightarrow (\mathsf{fn}\ x \Rightarrow N) : A \supset B}\ \supset R$$

Compound Left Rules

$$\frac{R_2 : P \in \Gamma \quad \Gamma, ?? : B \longrightarrow N : C}{\Gamma, R_1 : P \supset B \longrightarrow N : C}\ P{\supset}L$$

$$\frac{\Gamma, ?? : B \longrightarrow N : C}{\Gamma, R : \top \supset B \longrightarrow N : C}\ \top{\supset}L \qquad \frac{\Gamma, ?? : A_1 \supset (A_2 \supset B) \longrightarrow N : C}{\Gamma, R : (A_1 \wedge A_2) \supset B \longrightarrow N : C}\ \wedge{\supset}L$$

$$\frac{\Gamma \longrightarrow N : C}{\Gamma, R : \bot \supset B \longrightarrow N : C}\ \bot{\supset}L \qquad \frac{\Gamma, ?? : A_1 \supset B, ?? : A_2 \supset B \longrightarrow N : C}{\Gamma, R : A_1 \vee A_2 \supset B \longrightarrow N : C}\ \vee{\supset}L$$

$$\frac{\Gamma, ?? : A_2 \supset B \longrightarrow N_2 : A_1 \supset A_2 \quad \Gamma, ?? : B \longrightarrow N : C}{\Gamma, R : (A_1 \supset A_2) \supset B \longrightarrow N : C}\ \supset{\supset}L$$

Figure 2: G4ip with proof terms

Some advice. Here is some potentially useless advice, since it very much depends on the structure of your current G4ip prover. Nevertheless, we suggest you read it carefully before you embark on your implementation.

- It should be easy to generalize context Γ from lists (or a similar data structure) of types A to lists of pairs (R, A).

- If your internal functions implementing `decide` return booleans (a perfectly good design), you might consider modifying them to return a `term option` instead. You will then have to replace idioms such as `e orelse f` by code such as

```
case e
  of NONE => (case f
                of NONE => NONE
                 | SOME(N) => SOME(N))
   | SOME(N) => SOME(N)
```

  or write higher-order functions ("combinators") to achieve this effect.

- If your internal functions implementing `decide` use success continuation (functions to call if proving some subgoal *succeeds*), you should pass the proof term to the success continuation instead. Failure continuations (functions to call if some subgoal *fails*) probably don't have to change, since we are not producing certificates to witness unprovability.

- To fill in the missing proof terms in the rules, write some explicit ML functions to call. For example, in case of the rule $\wedge\supset L$ we discussed in lecture, you might have an ML function which given and $R : (A_1 \wedge A_2) \supset B$ returns an $R' : A_1 \supset (A_2 \supset B)$, and similarly in the other missing cases. In this case, presumably R' will be some version of of $(\mathsf{fn}\ x_1 \Rightarrow \mathsf{fn}\ x_2 \Rightarrow R\,(x_1, x_2))$ for some *fresh variables* x_1 and x_2. You have to be mindful of the fact that R' must be a synthesizing term and that x_1 and x_2 must be fresh (see next bullet).

- In certain rules, such as $\supset R$, $\vee L$, or $\wedge\supset L$ from the previous bullet, you will need to generate fresh variables, that is, variables that are distinct from any variables you already use. They need to be fresh

so they do not accidentally shadow (and thereby conflict with) variables already declared in in the context. For this purpose we have provided you with the function `new : (var -> 'a) -> 'a`. This function maintains local state so that if you write `new (fn x => e)` then it calls `(fn x => e)` with a fresh variable x_i every time it is called. In the example above, you might write something like

```
new (fn x1 => new (fn x2 =>
  Fun(x1,Fun(x2,App(R,Pair(Var(x1),Var(x2)))))))
```

Note that it would be a bug to simply write something like

```
Fun("x1",Fun("x2",
  App(R,Pair(Var("x1"),Var("x2"))))) (* bug! *)
```

because if `"x1"` or `"x2"` happen to be already be used in R the two new binders would capture them and very likely render the result no longer well-typed.

- If you find that you are not passing your local tests, please note that either your proof search engine *or* your proof checker could be at fault. Therefore, it is a good idea to thoroughly debug your proof checker prior to instrumenting your G4ip implementation.

Submitting. Please generate a tarball containing your solution files by running

```
$ tar cf hw6.tar check.sml g4ip.sml
```

and submit the resulting `hw6.tar` file to Autolab.