A decision procedure for intuitionistic propositional tautologies

Proof Assistants – Exercise

January 17, 2022

The goal of this exercise is to implement and prove correct a tactic that proves intuitionistic propositional tautologies. It has been inspired by the `tauto` tactic.

1 Intuitionistic propositional tautologies

The formulae of intuitionistic propositional logic follows the syntax:

$$A, B, C ::= x \mid \top \mid \bot \mid A \Rightarrow B \mid A \wedge B \mid A \vee B$$

where x belongs to an infinite set of proposition variables.

The decision procedure consists in searching for a complete derivation using the following rules:

$$\frac{A \in \Delta}{\Delta \vdash A}(Ax) \qquad \frac{\bot \in \Delta}{\Delta \vdash A}(\bot - E) \qquad \frac{}{\Delta \vdash \top}(\top - I)$$

$$\frac{\Delta, A \vdash B}{\Delta \vdash A \Rightarrow B}(\Rightarrow -I) \qquad \frac{\Delta, B \vdash C \quad \Delta \vdash A}{\Delta, A \Rightarrow B \vdash C}(\Rightarrow -E)$$

$$\frac{\Delta \vdash A \quad \Delta \vdash B}{\Delta \vdash A \wedge B}(\wedge - I) \qquad \frac{\Delta, A, B \vdash C}{\Delta, A \wedge B \vdash C}(\wedge - E)$$

$$\frac{\Delta \vdash A}{\Delta \vdash A \vee B}(\vee - I_1) \qquad \frac{\Delta \vdash B}{\Delta \vdash A \vee B}(\vee - I_1) \qquad \frac{\Delta, A \vdash C \quad \Delta, B \vdash C}{\Delta, A \vee B \vdash C}(\vee - E)$$

A given sequent may match several rules. The resulting non-determinism is implemented by a backtracking algorithm. Since the premisses are uniquely determined by the conclusion, there is no other source of non-determinism.

Remark that order of hypotheses is irrelevant so, for instance, $\Delta, A \wedge B$ means that one the hypothesis sould be a conjunction (it does have to be the last one), and Δ is the set of remaining hypotheses.

2 Implementing the decision procedure

In this section, the decision procedure is implemented using Coq tactics and tacticals. We first introduce several tools to build powerful tactics.

2.1 Advanced tactic features

(this section contains no question.)

Macros A tactic macro is defined using the `Ltac` keyword:

```
Ltac mytactic := <tac>.
```

The macros are recursive, so `<tac>` may refer to `mytactic`. Macros may have parameters:

```
Ltac mytactic arg1 arg2 := ...
```

match goal Coq features a tactical that combines goal pattern-matching and backtracking:

```
match goal with
| <hyp-patt>* |- <patt> => <tac>
| ...
end
```

where `<patt>` is a pattern for terms (pattern variables are preceded by a “?”; beware that the ? is not used in the right-handside), and `<hyp-patt>` is of the form `H:<patt>`, and matches an hypothesis which type matches `<patt>`; the name of the selected hypothesis is substituted for `H` in the right-handside. For instance, `| H: A /\ ?B |- _ => destruct H` would break an hypothesis which is a conjunction and the left subterm is `A`.

The current goal is matched against each goal-pattern until it finds a match. The corresponding tactic (properly substituted) is then executed. If this tactic fails, then the goal pattern-matching process resumes, with the same branch if several hypotheses match the pattern, or with the subsequnt branches.

Failure levels Failures bear a level which is an non-negative integer. `fail` and regular tactics raise failures with level 0. It is possible to raise a failure at arbitrary level with `fail <n>`.

The point of higher levels of failure is that the backtracking mechanism only catches failures with level 0. When a failure with higher level is raised, the match goal will not catch it, but will reraise it with its level decremented.

It is generally admitted that failure at level 999 will never be caught...

Step-by-step debugging There exists a step-by-step debugging tool (not available in CoqIDE) triggered by the command `Set Ltac Debug`.

Debugging messages Debugging backtracking tactics may be easier if a trace of each step performed is printed, especially when the above step-by-step debugging is not available.

The `idtac` tactic accepts arguments (terms or strings) that are printed. For instance `idtac (S (S O)) "text"` will print “2 text”.

Generating fresh hypothesis names Tactics may introduce new hypothesis. It is useful to be able to control how these hypothesis are named, but it is in general not possible to know which names are already used. The tactic `fresh` generates an hypothesis name that is not used in the current goal. It is generally used in combination with the tactic-level let/in as in the following example:

```
let H := fresh in
assert (H: x=y); [ ... | apply thm with (1:=H) ]
```

This tactic proves the proposition $x = y$, condition required by a theorem `thm`. Without the `fresh` command, the tactic would fail in cases where `H` is already used.

2.2 Building the tactic

1- Write a tactic `tauto1` that searches for a derivation (where propositions are expressed using Coq’s standard connectives). The tactic shall leave a trace of all rules applied (axiom rules may be omitted). It is also recommended to print the search depth in order to figure out how the tactic backtracks. It is crucial that the tactic generate subgoals that correspond exactly to the given rules. In particular, elimination rules must clear the hypothesis that has been used, to avoid inifinite loops.
2- Try the tactic on examples of tautologies. Here is a list of tautologies that should cover most of the tactic code:

$$\bot \Rightarrow A \quad A \wedge B \Rightarrow A \quad A \wedge B \Rightarrow B \quad A \wedge B \Rightarrow B \wedge A \quad A \Rightarrow A \vee B \quad B \Rightarrow A \vee B$$

$$(A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow A \vee B \Rightarrow C \quad A \Rightarrow (A \Rightarrow B) \Rightarrow B$$

$$A \Rightarrow (A \Rightarrow B) \Rightarrow (B \Rightarrow C) \Rightarrow B \quad A \Rightarrow (A \Rightarrow B) \Rightarrow (B \Rightarrow C) \Rightarrow C$$

2.3 Backtrack control

A rule is said *reversible* if the conjunction of the premisses is valid if and only if the conclusion is valid. It is pointless to backtrack on those rules, since the failure of one premisse implies the non-validity of the conclusion.
1- Write a tactic `tauto2` that improves `tauto1` by disabling the backtracking when a reversible rule is applied. Hint: use failure level.
2- Give an example of lemma where we can see from the trace that the backtracking has been disabled on reversible rules.

3 Formalizing the tactic

In this section, the tactic written above will be encoded in Coq. The main difficulty is in the modelization of the backtracking.

3.1 Tactic steps

1- Write an inductive type `form` representing the propositions, and a type `seq` for the sequents. Hint: use the lists of the standard library to represent the hypotheses of the sequent. The command `Print List.` may be useful to find the needed definitions and lemmas.
2- Write a function (`is_leaf : seq -> bool`) that recognizes when a sequent is an instance of one of the rules (Ax), $(\bot - E)$ or $(\top - I)$.

To deal with rules with premisses and the non-determinism resulting from several rules matching the same sequent, we introduce the type of subgoals:

```
Definition subgoals := list (list seq).
```

The outer level of lists represents the applicable rules. For each of these rules, the list of premisses to satisfy is given.

For instance, consider the sequent $A \vee B \vdash C \Rightarrow D$. Two rules may apply, leading to the list of subgoals:

$$[[A \vee B, C \vdash D];\ [A \vdash C \Rightarrow D;\ B \vdash C \Rightarrow D]]$$

3- Write a function `step : seq -> subgoals` implementing the rules with premisses (without backtrack control, i.e. `tauto1`). Hint: write a function `pick_hyp : seq -> list (form * list form)` that produces all possible choices to pick an hypothesis (returning the selected hyp, and the remaining hyps).
4- Write the decision procedure `tauto : nat -> seq -> bool`, that, given the max search depth and a sequent, returns true if the sequent is valid (within the specified search depth). Either the sequent is an instance of a rule without premisses, or the step function is applied. It remains to interpret the subgoals to check whether one of applicable rules generated premisses that are all valid.

3.2 Termination

Find a size criterion on sequents such that for each rule, the size of each premisse is smaller than the size of the conclusion. This gives an upper bound to the search depth.
1- Write the size function.
2- Prove that the step function above produces sequents of size smaller than the input sequent. Hint: use the `lia` tactic (first `Require Import Lia`) to solve the arithmetic inequations.

3.3 Soundness

A sequent is valid if whenever its hypotheses hold, then its conclusion holds. A list of subgoals is valid of one of its elements is a list of sequents that are all valid.

1- Define the semantics of the formulae (`sem : form -> Prop`), and the validity of sequents and subgoals.

2- Prove the soundness of the leaf case: if `is_leaf s` returns true, then s is valid.

3- Prove the soundness of the step case: if `step s` is a valid list of subgoals, then s is valid.

4- Prove the soundness of `tauto`.