Constructive Logic (15-317), Fall 2009
Assignment 8: Representing Proofs in Twelf

William Lovas (wlovas@cs)

Out: Thursday, October 29, 2009
Due: Thursday, November 5, 2009 (before class)

In this assignment, you'll learn hands-on how logics and their proofs can be represented in the logical framework LF using its implementation Twelf. You'll experiment with two different encodings, usually referred to as the *extrinsic* and *intrinsic* encodings. And you'll see how to transform a theorem prover into a *certifying* theorem prover that outputs not just a yes or no answer, but a proof of the proposition it claims to be true.

This assignment is all Twelf, all the time: there is no written portion. Your work should be submitted via AFS by copying your code to the directory

```
/afs/andrew/course/15/317/submit/<userid>/hw08
```

where <`userid`> is replaced with your Andrew ID. Submit files with the same names as the starter code, which is available from the course website.

Some tutorial notes are available via the web on using Twelf with Emacs and using Twelf without Emacs. These are part of the Twelf Wiki, a resource that you may find helpful while learning Twelf.

This assignment is meant to be straightforward so that you have time to study for the upcoming midterm exam, but you will have to become comfortable with using Twelf, so plan accordingly.

1 Warmup: Binary Numbers (8 points)

We represented binary numbers in Prolog two ways: first using the constructors `e`, `i(B)`, and `o(B)`, and second as lists of `0`s and `1`s. In Assignment 6, you wrote a predicate `convert(B, L)` that converted between these two representations in such a way that it could be run in either direction.

The following Twelf declarations (also found in the file `binary.elf`) give types for these representations.

```
binary : type.
```

```
e : binary.
i : binary -> binary.
o : binary -> binary.

bit : type.
0 : bit.
1 : bit.

bitlist : type.
nil : bitlist.
cons : bit -> bitlist -> bitlist.
```

Task 1 (8 pts). Port your `convert` predicate to Twelf using the above representations. Verify that it works in both directions by giving two `%mode` declarations for it. Verify also that it embodies a total function in both directions by giving two `%total` declarations.

Important Notes: You will have to turn off the `autoFreeze` parameter to allow multiple mode checks. In the Emacs mode, press `C-c <`, then type `autoFreeze`, and then type `false`; if interacting with the Twelf server directly, simply type `set autoFreeze false`.

Also, somewhere before your `%total` declaration, you will have to give a `%worlds` declaration of the form

```
%worlds () (convert B L).
```

2 Representing Proofs (20 points)

In lecture and recitation, we saw two different representations of proofs: a representation of raw proof terms `M : proof` with a proof-checking judgement `D : (M $ A)`, and an intrinsically well-formed representation `M : nd A`.[1] The following declarations are included in the files `prop.elf`, `nd.elf`, and `nd-intr.elf`, along with constants populating the types.

```
prop : type.

proof : type.
$ : proof -> prop -> type.

nd : prop -> type.
```

[1]In recitation, `nd A` was written as `true A`. We use the new name to distinguish between the case where we interpret the type family as a judgement on propositions and its inhabitants as derivations (`true A`) from the case where we interpret the type family and its inhabitants as the collection of well-formed proof terms (`nd A`).

Task 2 (10 pts). Extend the representation of propositions to include truth `tt` and falsehood `ff`. Add new raw proof terms, new inference rules for checking the proof terms, and new intrinsically well-formed proof terms.

Recall that when sanity-checking a new logic, we ensure that the elimination rules are not too strong by showing local soundness via local reductions, and we ensure that they are not too weak by showing local completeness via local expansions. We can represent local expansion as a relation in Twelf with the following declaration (also found in the file `nd-intr.elf`).

```
expand : {A:prop} nd A -> nd A -> type.
```

Given a proposition `A` and a proof term `M` of that proposition, `expand` relates the proof term `M` to one that introduces the main connective in `A`.

Task 3 (10 pts). Populate the type family `expand` with proofs of local expansions. Since you are working with an intrinsic representation, any accidentally invalid proofs will result in a Twelf type error. Verify that your `expand` has mode `+ + -` and that it embodies a total function: any proof of a proposition has an expansion.

Note: Remember to include a `%worlds` declaration for `expand`, just like in Task 1.

3 Certified Theorem Proving (12 points)

In lecture, we ported our Prolog implementation of the **G4ip**-plus-inversion theorem prover to Twelf for the purpose of making it a *certifying decision procedure*: whenever the Twelf version says a proposition is provable it also produces a proof that can be checked independently by a small, trusted checker. The code from lecture can be found in the file `g5ip-cert.elf`.

Task 4 (12 pts). Add cases to the theorem prover to handle propositions involving `tt` and `ff`. Again, since you will use the intrinsic representation, Twelf's type checker will help you maintain the invariant that your justifications are valid. (For reference, the **G4ip** rules can be found on page L12.5 of Lecture Notes 12.)

A List of Starter Files

The following files can be found on the course website.

File	Description
`binary.elf`	Binary numbers and bit lists
`prop.elf`	Propositions
`nd.elf`	Raw natural deduction proof terms and proof checking
`nd-intr.elf`	Intrinsically well-formed natural deduction proofs
`g5ip-cert.elf`	Certifying theorem prover from lecture
`task<`*n*`>.cfg`	Twelf configuration for Task *n*