Constructive Logic (15-317), Fall 2014
Assignment 8: Elf Programming

Joe Tassarotti (jtassaro@andrew), Evan Cavallo (ecavallo@andrew)

Out: Thursday, October 30, 2014
Due: November 6, 2014 (before class)

Your work should be submitted electronically before the beginning of class. Please name your homework `hw08.elf` and put the file in:

```
/afs/andrew/course/15/317/submit/<your andrew id>
```

If you are familiar with LaTeX, you are encouraged to use this document as a template for typesetting your solutions, but you may alternatively write your solutions *neatly* by hand and scan them.

1 Running Twelf

To run Twelf, execute

```
/afs/andrew/course/15/317/bin/twelf-server
```

from any Andrew machine. Alternatively, you may download and install a copy locally from `http://twelf.org/`, but please test your code a final time on an Andrew machine to ensure it works there, as that is what we will use to grade.

You can load a file `foo.elf` at the prompt by typing

```
loadFile foo.elf
```

You can clear out the definitions you've loaded by running `reset`. It's good practice to do this before re-loading a file. Once a file is loaded, you can type `top` to bring up the query prompt. Then, you can issue queries by typing predicates at the prompt as you have seen in class.

If you see Twelf print out an error mentioning an identifier `%foo%`, this is referring to some definition of `foo` that has been shadowed by later redefinitions. You should try resetting and then loading your file again to get a more useful error message.

2 Programming (9 points)

Recall how we defined the natural numbers in Elf:

```
nat: type.
z: nat.
s: nat -> nat.
```

Task 1 (3 points)**.** Define a predicate `lte: nat -> nat -> type`, where `lte M N` holds if and only if `M` is less than or equal to `N`. Give your predicate `%mode` and `%terminates` declarations.

We can also define the type of lists of natural numbers, like so:

```
natlist : type.
nil: natlist.
cons: nat -> natlist -> natlist.
```

Task 2 (3 points)**.** Using your `lte` predicate, define a predicate `sorted: natlist -> type` which holds when the input list is sorted. Give your predicate `%mode` and `%terminates` declarations.

Task 3 (3 points)**.** Define a predicate `map: natlist -> (nat -> nat) -> natlist -> type`, where `map L1 F L2` holds if and only if the list `L2` is the result of applying `F` to each of the elements of `L1` Give your predicate `%mode` and `%total` declarations.

3 Proof search (6 points)

In class we showed how to encode natural deduction into Elf, and we saw that Twelf could search for proofs in natural deduction. In this section we will explore how Twelf searches when we execute a query. First, let's recall the encoding of natural deduction we gave:

```
pr : type.
at : type.

a : at.
b : at.

atomic : at -> pr.
t : pr.
f : pr.
```

```
and : pr -> pr -> pr.
or : pr -> pr -> pr.
imp : pr -> pr -> pr.

true : pr -> type.
%mode true *P.

true/ti : true t.

true/andi : true (and A B)
             <- true A
             <- true B.

true/ori1 : true (or A B)
             <- true A.

true/ori2 : true (or A B)
             <- true B.

true/impi : true (imp A B)
             <- (true A -> true B).

true/ande1 : true A
              <- true (and A B).

true/ande2 : true B
              <- true (and A B).


true/impe : true B
             <- true (imp A B)
             <- true A.

true/ore : true C
            <- true (or A B)
            <- (true A -> true C)
            <- (true B -> true C).

true/fe : true C
           <- true f.
```

If we load this file and run the query `true X`, we get output like the following:

```
?- true X.
Solving...
X = t.
More? ;
X = and t t.
More? ;
X = and t (and t t).
More? ;
X = and t (and t (and t t)).
More? ;
X = and t (and t (and t (and t t))).
More? ;
X = and t (and t (and t (and t (and t t)))).
More? ;
X = and t (and t (and t (and t (and t (and t t))))).
...
```

Like Prolog, Twelf tries clauses in the order they appear in the source file. When it tries to match against `true/ti: true t`, it succeeds, giving us our first solution X = t. When we ask for another solution, it back tracks and matches against `true/andi1`, generating the constraint that `X = and A B`. It first tries to prove the subgoal `true A`, which succeeds via `true/ti`, and then it tries to prove `true B`, which succeeds the same way, giving us the solution `X = and t t`. When we ask for yet another solution, Twelf back tracks in the goal `true B`, trying to generate another prooof of this. This process can be repeated indefinitely.

If we instead do the query:

```
?- true (imp X X).
Solving...
X = X.
More ? ;
X = t.
More ? ...
```

Our first solution is `X = X`, because we will match against `true/impi`, which gives us the constraint `imp X X = imp A B` and generates the subgoal `true X -> true X`. This adds the assumption `true X` as we try to prove `true B`. Twelf tries these local assumptions *before* trying the global clauses. In this case, this assumption solves our goal, giving us the solution `X = X` before `X = t`.

You can find a more thorough explanation of Twelf's proof search by looking here in the manual: `http://www.cs.cmu.edu/~twelf/guide-1-4/twelf_5.html#SEC27`.

Task 4 (2 points)**.** When we run the query `true (imp (and X X) X)`, why don't we get `X = X` as a solution? Explain why in a comment in your Elf file.

Task 5 (2 points)**.** Suppose we modify the definition of `true/andi` to:

```
true/andi : true (and A B)
             <- true B
             <- true A.
```

What happens when we now run the query `true X`? Explain why, in a manner similar to the explanations given above. Again, write your answer as a comment.

Task 6 (2 points)**.** What happens when we run the query `true X` after reordering the clauses of the `true` judgment so that `true/fe` appears first? Explain why.