# Constructive Logic (15-317), Fall 2012
# Assignment 8: Logic Programming in Elf

Carlo Angiuli (`cangiuli@cs`)

Out: Thursday, November 1, 2012
Due: Thursday, November 8, 2012 (at the end of class)

In this assignment, you will explore how the types and directives in Elf can be used to reason about logic programs in ways we could not in Prolog. By the end, you should have a good understanding of mode, coverage, and termination checking.

The (short) written portions of this assignment may be submitted as comments in your code.

Your code should be submitted via AFS by copying it to the directory

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

where <`userid`> is replaced with your Andrew ID. Your solutions should work in the version of Twelf installed in the course directory.

## 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 following the directions at `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
```

To issue queries, type `top` and enter predicates at the prompt.

## 2 Representing Numbers (20 points)

Let's recap how we would write the predicates double and plus over natural numbers in Elf:

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

double : nat -> nat -> type.
double/z : double z z.
double/s : double (s N) (s (s N2))
            <- double N N2.

plus : nat -> nat -> nat -> type.
plus/z : plus z N N.
plus/s : plus (s N) M (s NM)
          <- plus N M NM.
```

To warm up:

**Task 1** (5 points). Add %mode, %worlds, and %total declarations to double and plus.

This representation of natural numbers can get a little tiresome, though. Let's shake things up by using *binary* instead.

```
binary : type.
e : binary.
1 : binary -> binary.
0 : binary -> binary.
```

e is the empty bitstring; the 1 constructor adds a 1 and the 0 constructor adds a 0. The bit closest to e should be thought of as the most significant bit. So, for example, the bitstring 10011 (19 in decimal) would be represented as "1 (1 (0 (0 (1 e))))".

**Task 2** (15 points). Starting with the file binary.elf, write a predicate
b2u : binary -> nat -> type. converting binary strings to unary natural numbers. Give it %mode, %worlds, and %total declarations.

## 3 Representing Logics (20 points)

Here is the beginning of an encoding of verifications and uses in Elf.

```
atom : type.
p : atom.
q : atom.
r : atom.

prop : type.
and : prop -> prop -> prop.
or : prop -> prop -> prop.
imp : prop -> prop -> prop.
t : prop.
f : prop.
atomic : atom -> prop.

verif : prop -> type.
use : prop -> type.

verif/and : verif (and A B)
             <- verif A
             <- verif B.
verif/or1 : verif (or A B)
             <- verif A.
verif/or2 : verif (or A B)
             <- verif B.
use/and1  : use A
             <- use (and A B).
use/and2  : use B
             <- use (and A B).
use/or    : verif C
             <- use (or A B)
             <- (use A -> verif C)
             <- (use B -> verif C).
```

**Task 3** (15 points)**.** Starting with the file `verif.elf`, fill in the remaining cases.

**Task 4** (5 points)**.** Recall from class that the `true` judgment we wrote was not well-moded. It turns out that the normal natural deduction judgments *can* be given modes. Figure out the modes for `verif` and `use` (thinking about it before running a guess through Twelf is more fun!) and add `%mode` declarations. In comments, give a brief explanation of why a mode can be assigned here while it could not for `true`.