# Constructive Logic (15-317), Fall 2009
# Assignment 6: Logic Programming and Inversion

William Lovas (`wlovas@cs`)

Out: Thursday, October 15, 2009
Due: Thursday, October 22, 2009 (before class)

This assignment serves to introduce you to Prolog programming by having you play around with a more realistic representation of natural arithmetic. You will also have a chance to review the key concepts of *invertibility* in the context of the quantifiers of first-order logic.

The written portion of your work (Section 2) should be submitted at the beginning of class. As usual, 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.

The programming portion of your work (Section 1) should be submitted via AFS by putting your code in a file `binary.pl` and copying it to the directory

```
/afs/andrew/course/15/317/submit/<userid>/hw06
```

where <`userid`> is replaced with your Andrew ID.

## 1 Building a Better Number

Although we like to use natural numbers defined by `z` and `s` for logic programming examples in a classroom setting, such a unary representation sacrifices a great deal in asymptotic performance. But a binary representation with good complexity characteristics is just as possible in a logic programming language like Prolog.

Consider representing binary numbers as strings of bits using the following constants:

- `e` stands for the empty string,
- `i(B)` stands for a one bit followed by the string `B`, and
- `o(B)` stands for a zero bit followed by the string `B`.

**Task 1 (4 pts).** Define a type predicate `binary(B)` which is true for all bit strings and false for everything else.

We interpret these bit strings in a little-endian fashion: the bit exposed at the front is the *least significant bit*. Under this interpretation, many algorithms can be written in a relatively natural inductive fashion.

**Task 2 (10 pts).** Define a predicate `add(B1, B2, B3)` with mode `(+, +, -)` that computes the sum `B3` of binary numbers `B1` and `B2`.

An alternative representation of binary numbers is Prolog lists of Prolog integers, where each integer in the list is either `0` or `1`.

**Task 3 (6 pts).** Define a predicate `convert(B, L)` that converts between bit strings and lists of `1`s and `0`s. Write the clauses in such a way that the conversion can be run in either direction.

**Extra Credit Task 1 (3 extra credit pts).** Can you define a conversion between bit strings and unary natural numbers that can be run in either direction?

## 2 Quantifiers and Inversion (20 points)

Recall the sequent calculus rules for quantifiers:

$$\frac{(\Sigma, c{:}\tau)\,;\,\Gamma \Longrightarrow A(c)}{\Sigma\,;\,\Gamma \Longrightarrow \forall x{:}\tau.\,A(x)}\ \forall R \qquad \frac{\Sigma \vdash t : \tau \quad \Sigma\,;\,(\Gamma, \forall x{:}\tau.\,A(x), A(t)) \Longrightarrow C}{\Sigma\,;\,(\Gamma, \forall x{:}\tau.\,A(x)) \Longrightarrow C}\ \forall L$$

$$\frac{\Sigma \vdash t : \tau \quad \Sigma\,;\,\Gamma \Longrightarrow A(t)}{\Sigma\,;\,\Gamma \Longrightarrow \exists x{:}\tau.\,A(x)}\ \exists R \qquad \frac{(\Sigma, c{:}\tau)\,;\,(\Gamma, \exists x{:}\tau.\,A(x), A(c)) \Longrightarrow C}{\Sigma\,;\,(\Gamma, \exists x{:}\tau.\,A(x)) \Longrightarrow C}\ \exists L$$

### 2.1 Identity

The Identity theorem states that assumptions are strong enough to become conclusions, even when we limit the "init" rule to atomic formulas $P$.

**Theorem (Identity).** For all $A$, we can derive $\Sigma\,;\,(\Gamma, A) \Longrightarrow A$.

*Proof.* By induction on the structure of $A$. □

**Task 4 (2 pts).** Prove the case of the Identity theorem for $\forall x{:}\tau.\,A(x)$.

**Task 5 (2 pts).** Prove the case of the Identity theorem for $\exists x{:}\tau.\,A(x)$.

Remember that for any term $t$, the proposition $A(t)$ is considered to be a subformula of the propositions $\forall x{:}\tau.\,A(x)$ and $\exists x{:}\tau.\,A(x)$.

## 2.2 Invertibility

A rule is said to be *invertible* if its conclusion entails its premises. As you learned in the last assignment, the value of invertibility is that it culls the search space of a goal-directed theorem prover considerably: invertible rules can often be applied asynchronously since they simplify the sequent in some way without changing its provability.

**Task 6 (4 pts).** Is the $\forall R$ rule invertible? If so, prove it; if not, give a counterexample.

**Task 7 (4 pts).** Is the $\forall L$ rule invertible? If so, prove it; if not, give a counterexample.

**Task 8 (4 pts).** Is the $\exists R$ rule invertible? If so, prove it; if not, give a counterexample.

**Task 9 (4 pts).** Is the $\exists L$ rule invertible? If so, prove it; if not, give a counterexample.

A proof of invertibility for a rule should be an step-by-step argument that given a deduction of the rule's conclusion, you can construct deductions of its premises. A counterexample should be a deduction of an instance of the rule's conclusion along with an argument that no deduction exists for one of its premises.

For each task, you may assume the following structural properties of the sequent calculus:

**Lemma (Weakening).** If $\Sigma\,;\Gamma \Longrightarrow C$, then $\Sigma\,;(\Gamma, A) \Longrightarrow C$.

**Lemma (Term Weakening).** If $\Sigma\,;\Gamma \Longrightarrow C$, then $(\Sigma, c{:}\tau)\,;\Gamma \Longrightarrow C$.

**Lemma (Contraction).** If $\Sigma\,;(\Gamma, A, A) \Longrightarrow C$, then $\Sigma\,;(\Gamma, A) \Longrightarrow C$.

**Lemma (Identity).** For all $A$, we can derive $\Sigma\,;(\Gamma, A) \Longrightarrow A$.

**Lemma (Cut).** If $\Sigma\,;\Gamma \Longrightarrow A$ and $\Sigma\,;(\Gamma, A) \Longrightarrow C$, then $\Sigma\,;\Gamma \Longrightarrow C$.

**Extra Credit Task 2 (3 extra credit pts).** If you were adding quantifiers to the restrictive sequent calculus $\Sigma\,;\Gamma \longrightarrow C$, how would you change the above rules? How would you justify these changes?