Midterm II

15-317: Constructive Logic

November 6, 2014

Name: Andrew ID:

Instructions

- This exam is closed book and closed Internet. A two-sided sheet of handwritten notes is permitted.
- There are three problems. Not all problems are the same size or difficulty, so it may help to read through the whole exam first. You have ninety minutes to complete the exam.
- You may find it helpful to construct your proofs on scratch paper (such as the back of a page) before writing it clearly in the space provided.
- Good luck!

	Problem 1	Problem 2	Problem 3	**Total**
Score				
Max	30	40	30	100

1 Invertibility

Problem 1: Below are three rules governing a connective $\spadesuit(A, B, C)$ in natural deduction. Which of the rules are invertible, and which are non-invertible? Explain.

$$\dfrac{\begin{array}{cc} A\ \mathsf{true} & A\ \mathsf{true} \\ \vdots & \vdots \\ B\ \mathsf{true} & C\ \mathsf{true} \end{array}}{\spadesuit(A, B, C)\ \mathsf{true}}\ \spadesuit I \qquad \dfrac{\spadesuit(A, B, C)\ \mathsf{true} \quad A\ \mathsf{true}}{B\ \mathsf{true}}\ \spadesuit E1 \qquad \dfrac{\spadesuit(A, B, C)\ \mathsf{true} \quad A\ \mathsf{true}}{C\ \mathsf{true}}\ \spadesuit E2$$

Problem 2: Below are three rules governing a connective $\clubsuit(A, B, C)$ in sequent calculus. Which of the rules are invertible, and which are non-invertible? Explain. (You may assume that cut admissibility and identity expansion hold.)

$$\frac{\Delta \longrightarrow A \quad \Delta \longrightarrow B}{\Delta \longrightarrow \clubsuit(A, B, C)}\ \clubsuit R1 \qquad \frac{\Delta \longrightarrow A \quad \Delta \longrightarrow C}{\Delta \longrightarrow \clubsuit(A, B, C)}\ \clubsuit R2$$

$$\frac{\Delta, A, B \longrightarrow D \quad \Delta, A, C \longrightarrow D}{\Delta, \clubsuit(A, B, C) \longrightarrow D}\ \clubsuit L$$

Problem 3: Explain the role that invertibility plays in automated theorem proving. Use complete sentences.

2 Logic Programming

Suppose we are given a collection of rules of the form `edge`$(\mathit{from}, \mathit{to}, \mathit{cost})$ that define a directed acyclic graph with weighted edges. All costs are greater than zero. For example:

```
edge(a, b, 1).
edge(a, c, 1).
edge(b, d, 2).
edge(c, d, 2).
```

defines a graph with four nodes, in which one can move from `a` to `b` or `c` at a cost of 1, and from `b` or `c` to `d` at a cost of 2.

Problem 1: In Prolog, implement a predicate `path`(V, W, N) that holds when there exists a path (consisting of at least one edge) from V to W with a total cost of **exactly** N. Use Prolog's built-in arithmetic for cost arithmetic. Do not use cut.

Problem 2: Suppose the graph **may** contain cycles, all costs remain greater than zero, and `path` is always called with a ground integer as its third argument. Re-implement `path` so that it never enters an infinite loop. Use Prolog's built-in arithmetic for cost arithmetic. Do not use cut.

3 Twelf

Below is Twelf code for natural numbers and addition, with mode, termination, and coverage checking.

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

plus : nat -> nat -> nat -> type.
%mode plus +M +N -P.

plus/z : plus z N N.

plus/s : plus (s M) N (s P)
          <- plus M N P.

%worlds () (plus _ _ _).
%total M (plus M _ _).
```

Problem 1: Below is Twelf code for multiplication. Either mode checking, termination checking, or coverage checking fails. Explain which one fails, and why. Then correct the program. Do not change the mode declaration.

```
mult : nat -> nat -> nat -> type.
%mode mult +M +N -P.

mult/z : mult z N z.

mult/s : mult (s M) N Q
          <- mult M N P
          <- plus P N Q.

%worlds () (mult _ _ _).
%total N (mult _ N _).
```

Problem 2: Below is Twelf code for exponentiation. Either mode checking, termination checking, or coverage checking fails. Explain which one fails, and why. Then correct the program. Do not change the mode declaration. You may assume that the implementation of `mult` is correct.

```
exp : nat -> nat -> nat -> type.
%mode exp +M +N -P.

exp/z : exp N z (s z).

exp/s : exp M (s N) Q
         <- mult P M Q
         <- exp M N P.

%worlds () (exp _ _ _).
%total N (exp _ N _).
```

Problem 3: Below is Twelf code for factorial. Either mode checking, termination checking, or coverage checking fails. Explain which one fails, and why. Then correct the program. Do not change the mode declaration. You may assume that the implementation of `mult` is correct.

```
fact : nat -> nat -> type.
%mode fact +M -N.

fact/1 : fact (s z) (s z).

fact/s : fact (s N) Q
          <- fact N P
          <- mult (s N) P Q.

%worlds () (fact _ _).
%total N (fact N _).
```