Midterm 2

15-317: Constructive Logic

November 5, 2009

Name: Andrew ID:

Instructions

- This exam is closed-book, but one two-sided sheet of notes is permitted.
- There are five problems, each with several parts. Not all problems are the same size or difficulty. You have 80 minutes to complete the exam.
- When writing proofs, remember to label each inference with the rule used.
- 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.
- Most importantly,

Don't Panic

Good luck!

	Problem 1	Problem 2	Problem 3	Problem 4	Problem 5	**Total**
Score						
Max	25	35	55	20	15	150
Grader						

1 Logic Programming (25 points)

Consider a predicate `intersect(+list, +list)` which holds of any two lists that share an element, defined using the `member` predicate on lists.

```
member(X, [X|Xs]).
member(X, [Y|Ys]) :-
    member(X, Ys).
```

```
intersect(L1, L2) :-
    member(X, L1),
    member(X, L2).
```

Task 1 (10 pts). What mode or modes must `member` satisfy in order for `intersect` to satisfy mode `(+, +)`? Explain in detail why `member` satisfies this mode or modes.

Task 2 (5 pts). Are there any other modes that `intersect` can be given, besides `(+, +)`? Explain your answer.

(Problem continues on next page)

Consider the following ways of introducing cuts into the code for `intersect` to cut off backtracking. We are only interested in the behavior on the given mode, where `L1` and `L2` are both inputs and valid lists.

Task 3 (5 pts). Is the following cut red or green? If it is red, exhibit a query provable *without* the cut that isn't provable *with* the cut. If it is green, explain why no such query exists.

```
intersect(L1, L2) :-
    member(X, L1),
    !,
    member(X, L2).
```

Task 4 (5 pts). Is the following cut red or green? If it is red, exhibit a query provable *without* the cut that isn't provable *with* the cut. If it is green, explain why no such query exists.

```
intersect(L1, L2) :-
    member(X, L1),
    member(X, L2),
    !.
```

2 G4ip (35 points)

Recall Dyckhoff's contraction-free sequent calculus **G4ip**.

Task 1 (10 pts). Give a **G4ip** derivation of the sequent $\cdot \longrightarrow \neg\neg(P \vee \neg P)$. Assume the proposition P is atomic and $\neg A$ is a notational definition, as usual.

(Problem continues on next page)

Consider a variant of **G4ip** with a "memoizing" right rule for conjunction.

$$\frac{\Gamma \longrightarrow A \quad \Gamma, A \longrightarrow B}{\Gamma \longrightarrow A \wedge B}\ \wedge R'$$

It can be difficult to tell whether such a rule represents an optimization or a complication of the decision procedure embodied by **G4ip**.

Task 2 (10 pts). Find a derivation of a sequent that "saves time" using $\wedge R'$. The smaller derivation should be identical to the larger aside from some work that is saved thanks to the memoizing rule.

(Problem continues on next page)

Task 3 (15 pts). Find a derivation of a sequent that "wastes time" using $\wedge R'$. The larger derivation should be identical to the smaller aside from some extra work that could be done thanks to the memoizing rule.

3 Classical Sequent Calculus (55 points)

We may define a classical sequent calculus by allowing a sequent to have multiple conclusions. The main judgement now becomes $\Gamma \ \# \ \Delta$, with the interpretation that if all of the propositions in Γ are true, then at least one of the propositions in Δ is true, or equivalently: if all of the propositions in Γ are true and all of the propositions in Δ are false, then they are in contradiction.

$$\frac{}{\Gamma, P \ \# \ P, \Delta} \ \text{init}$$

$$\frac{\Gamma \ \# \ A, B, \Delta}{\Gamma \ \# \ A \vee B, \Delta} \ \vee R \qquad \frac{\Gamma, A \ \# \ \Delta \quad \Gamma, B \ \# \ \Delta}{\Gamma, A \vee B \ \# \ \Delta} \ \vee L$$

$$\frac{\Gamma, A \ \# \ \Delta}{\Gamma \ \# \ \neg A, \Delta} \ \neg R \qquad \frac{\Gamma \ \# \ A, \Delta}{\Gamma, \neg A \ \# \ \Delta} \ \neg L$$

We do not need rules for $A \wedge B$, $A \supset B$, $\top$, or $\bot$ because these can be defined in classical logic.

Task 1 (5 pts). Give a derivation of the sequent $\cdot \ \# \ P \vee \neg P$ in this calculus. Assume the proposition P is atomic.

(Problem continues on next page)

The Cut Principle for this calculus is as follows:

Theorem (Cut). If $\Gamma \# A, \Delta$ and $\Gamma, A \# \Delta$, then $\Gamma \# \Delta$.

Task 2 (20 pts). Prove the principal case of the Cut Principle for $A \vee B$. Given

$$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1 \\ \Gamma \# A, B, \Delta\end{array}}{\Gamma \# A \vee B, \Delta} \vee R \qquad \text{and} \qquad \mathcal{E} = \dfrac{\begin{array}{cc}\mathcal{E}_1 & \mathcal{E}_2 \\ \Gamma, A \# \Delta & \Gamma, B \# \Delta\end{array}}{\Gamma, A \vee B \# \Delta} \vee L,$$

come up with a derivation of $\Gamma \# \Delta$. You may assume the following lemmas:

Lemma (Left Weakening). If $\Gamma \# \Delta$, then $\Gamma, A \# \Delta$ with a structurally identical deduction.

Lemma (Right Weakening). If $\Gamma \# \Delta$, then $\Gamma \# A, \Delta$ with a structurally identical deduction.

(Problem continues on next page)

Recall that an *invertible* rule is one whose conclusion entails its premises. For the next two Tasks, you may assume the following structural properties of the classical sequent calculus as lemmas:

Lemma (Left Weakening). If $\Gamma \# \Delta$, then $\Gamma, A \# \Delta$.

Lemma (Right Weakening). If $\Gamma \# \Delta$, then $\Gamma \# A, \Delta$.

Lemma (Identity). For any Γ, Δ, and A, we can derive $\Gamma, A \# A, \Delta$.

Lemma (Cut). If $\Gamma \# A, \Delta$ and $\Gamma, A \# \Delta$, then $\Gamma \# \Delta$.

It turns out that all rules in the calculus are invertible.

Task 3 (15 pts). Prove that the $\vee R$ rule is invertible.

(Problem continues on next page)

Task 4 (5 pts). We can interpret the inference rules as a proof search calculus, working bottom-up. Does proof search in this calculus always terminate? If so, explain why. If not, give a sequent that causes proof search to loop.

Task 5 (10 pts). Since we are in classical logic, we can define $A \supset B$ as $\neg A \vee B$. Give *derived* left and right rules for $A \supset B$ based on this definition.

4 Proof Search in Prolog (20 points)

Refer to the classical sequent calculus of Problem 3.

Task 1 (20 pts). Complete the following Prolog code implementing a proof search strategy for our classical sequent calculus. You may use the predicate `intersect` from Problem 1.

```
decide(Ps, Qs, G, [~ A | D]) :-
    decide(Ps, Qs, [A | G], D).

decide(Ps, Qs, [~ A | G], D) :-
    decide(Ps, Qs, G, [A | D]).

decide(Ps, Qs, G, [A \/ B | D]) :-

    ____________________________________________________________ .


decide(Ps, Qs, [A \/ B | G], D) :-

    ____________________________________________________________ ,

    ____________________________________________________________ .


decide(Ps, Qs, G, [? P | D]) :-

    ____________________________________________________________ .


decide(Ps, Qs, [? P | G], D) :-

    ____________________________________________________________ .


decide(Ps, Qs, [], []) :-

    ____________________________________________________________ .
```

5 Representing Proofs in LF (15 points)

Recall that the logical framework LF offers a clean and elegant representation of sequent calculi with weakening and contraction in which sequent calculus hypotheses are represented as LF hypotheses. If we are interested in proving things about the calculus from Problem 3 rather than using it for proof search, such a representation is very convenient.

Task 1 (15 pts). Imagine we permit implicit contraction in our classical sequent calculus. Recall that we interpret a sequent $\Gamma \# \Delta$ in the following way: if all the propositions in Γ are true and all the propositions in Δ are false, then they are in contradiction. We exploit this view in LF by representing every A in Γ as a hypothesis `true A`, and every B in Δ as a hypothesis `false B`.

Since every provable sequent represents a contradiction, the conclusion in LF is always a new judgment `contra`.

Complete the following LF representation of the new calculus using the ideas above.

```
contra : type.
true : prop -> type.
false : prop -> type.



init : ____________________________________________________________ .


~R : (true A -> contra)
  -> (false (~ A) -> contra).

~L : (false A -> contra)
  -> (true (~ A) -> contra).



\/R : ____________________________________________________________

   -> ____________________________________________________________ .


\/L : (true A -> contra)
   -> (true B -> contra)
   -> (true (A \/ B) -> contra).
```