Assignment 7:
Theorem Proving

15-317: Constructive Logic

Out: Thursday, October 23, 2008
Due: Thursday, October 30, 2008, before class

1 Sequent Calculus

1.1 Rules

$$\frac{P \in \Gamma}{\Gamma \Longrightarrow P}\ \mathit{init}$$

$$\frac{\Gamma \Longrightarrow A \quad \Gamma \Longrightarrow B}{\Gamma \Longrightarrow A \wedge B}\ \wedge R \qquad \frac{\Gamma, A, B \Longrightarrow C}{\Gamma, A \wedge B \Longrightarrow C}\ \wedge L$$

$$\frac{}{\Gamma \Longrightarrow \top}\ \top R \qquad \frac{\Gamma \Longrightarrow C}{\Gamma, \top \Longrightarrow C}\ \top L$$

$$\frac{\Gamma \Longrightarrow A}{\Gamma \Longrightarrow A \vee B}\ \vee R_1 \quad \frac{\Gamma \Longrightarrow B}{\Gamma \Longrightarrow A \vee B}\ \vee R_2 \qquad \frac{\Gamma, A \Longrightarrow C \quad \Gamma, B \Longrightarrow C}{\Gamma, A \vee B \Longrightarrow C}\ \vee L$$

$$\text{(no rule } \bot R\text{)} \qquad \frac{}{\Gamma, \bot \Longrightarrow C}\ \bot L$$

$$\frac{\Gamma, A \Longrightarrow B}{\Gamma \Longrightarrow A \supset B}\ \supset R \qquad \frac{\Gamma, A \supset B \Longrightarrow A \quad \Gamma, B \Longrightarrow C}{\Gamma, A \supset B \Longrightarrow C}\ \supset L$$

Note that we treat Γ as a set of hypotheses, so the notation Γ, A means that A occurs somewhere in the set of hypotheses.

This sequent calculus satisfies weakening, identity, and cut:

- **Weakening:** If $\Gamma \Longrightarrow C$ then $\Gamma, A \Longrightarrow C$.
- **Identity:** For all A, $\Gamma, A \Longrightarrow A$.
- **Cut:** If $\Gamma \Longrightarrow A$ and $\Gamma, A \Longrightarrow C$ then $\Gamma \Longrightarrow C$.

1.2 Differences with class

This sequent calculus differs from the one presented in class on Thursday, Oct. 23. in a few ways:

- Conjunction is given a single left rule, decomposing the conjunction into two assumptions, rather than two left rules corresponding to `fst` and `snd`.
- The rules $\top L, \wedge L, \vee L, \supset L$ remove the assumption you're using from the context (when going from the conclusion of the rule to the premise). For example, in class, we wrote:

$$\frac{\Gamma, A \vee B, A \Longrightarrow C \quad \Gamma, A \vee B, B \Longrightarrow C}{\Gamma, A \vee B \Longrightarrow C} \vee L_{class}$$

  which keeps the assumption $A \vee B$ in both premises. This is problematic if you want to implement a theorem prover by typing the rules into Prolog: you can apply the $\vee L_{class}$ rule again in the premises, and again in their premises, etc., causing Prolog's proof search to diverge.

However, the above sequent calculus is still correct, because neither of these changes affects which sequents are provable. We ask you to show one case:

Task 1 (5 pts). Show that the rules $\vee L_{class}$ and $\vee L$ are equivalent by proving the following:

1. $\Gamma, A \vee B, A \Longrightarrow C$ iff $\Gamma, A \Longrightarrow C$
2. $\Gamma, A \vee B, B \Longrightarrow C$ iff $\Gamma, B \Longrightarrow C$

Hint: use weakening, identity, and cut!

1.3 Prolog Implementation

In Figure 1, we present a Prolog implementation of these rules. The predicate `choose(G,A,G')` means that A is in G and that removing A from G gives G' (i.e., $G = (G', A)$, keeping in mind that order doesn't matter). We'll use this predicate to write the left rules, so they apply to any assumption in the context.

Task 2 (0 pts). Read and understand this Prolog code.

2 Contraction-Free Sequent Calculus

2.1 The Problem

Unfortunately, the above Prolog program does not always terminate. In almost every rule, a connective is decomposed as you go from the conclusion to the premises. However, this is not the case for the implication left rule, where the assumption $A \supset B$ is copied to the first premise:

$$\frac{\Gamma, \boxed{A \supset B} \Longrightarrow A \quad \Gamma, B \Longrightarrow C}{\Gamma, A \supset B \Longrightarrow C} \supset L$$

Task 1 (3 pts). Explain why running the above Prolog program to `prove` the sequent

$$[] \Longrightarrow ((P \vee (P \supset \bot)) \supset \bot) \supset \bot$$

loops: find a cycle in the series of goals that Prolog tries to prove. (For example, if the rules were

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

% == initial ==
prove(G0,atom(P)) :- choose(G0,atom(P),_).

% == right rules ==
prove(_,top).
prove(G,and(A,B)) :- prove(G,A),prove(G,B).
%% no rule for bot
prove(G,or(A,_)) :- prove(G,A).
prove(G,or(_,B)) :- prove(G,B).
prove(G,implies(A,B)) :- prove([A|G],B).

% == left rules ==
prove(G0,C) :-
        choose(G0,top,G),
        prove(G,C).
prove(G0,C) :-
        choose(G0,and(A,B),G),
        prove([A,B|G],C).
prove(G0,_) :-
        choose(G0,bot,_).
prove(G0,C) :-
        choose(G0,or(A,B),G),
        prove([A|G],C),
        prove([B|G],C).
prove(G0,C) :-
        choose(G0,implies(A,B),G),
        prove(G0,A),
        prove([B|G],C).
```

Figure 1: Prolog code for sequent calculus rules

```
foo(X) :- bar(s(X)).
bar(s(X)) :- foo(X).
```

your answer would be "When Prolog tries to prove `foo(z)` using the first rule, it tries to prove `bar(s(z))`, but using the second rule yields the subgoal `foo(z)`, so we've cycled back to where we started.")

Task 2 (2 pts). Suppose we did not copy the implication to the first premise:

$$\frac{\Gamma \Longrightarrow A \quad \Gamma, B \Longrightarrow C}{\Gamma, A \supset B \Longrightarrow C} \supset L'$$

Argue that this sequent calculus is not complete by explaining why the following sequent is not provable:

$$[] \Longrightarrow ((P \vee (P \supset \bot)) \supset \bot) \supset \bot$$

2.2 The Solution

Thus, we need to keep the implication assumption in the first premise, but the naïve way of doing this leads to non-termination. We can fix this problem using *Dyckhoff's contraction-free sequent calculus*. The key idea is to replace the above implication left-rule with a set of better-behaved rules. These rules break down the *left-hand-side* of an implication assumption, so something gets smaller every time a rule is applied.

We will have one implication-left-rule for each possible left-hand-side. Four of these rules are based on equivalences, many of which you have proved over the course of the semester:

$$\begin{aligned}
(\top \supset B) &\equiv B \\
(A_1 \wedge A_2) \supset B &\equiv A_1 \supset (A_2 \supset B) \\
(\bot \supset B) &\equiv \top \\
(A_1 \vee A_2) \supset B &\equiv (A_1 \supset B) \wedge (A_2 \supset B)
\end{aligned}$$

Read left-to-right, these equivalences simplify the proposition to the left of the $\supset$.

This gives the following four rules, which replace an assumption with an equivalent but simpler one:

$$\frac{\Gamma, B \Longrightarrow C}{\Gamma, \top \supset B \Longrightarrow C} \supset L_\top \quad \frac{\Gamma, A_1 \supset (A_2 \supset B) \Longrightarrow C}{\Gamma, (A_1 \wedge A_2) \supset B \Longrightarrow C} \supset L_\wedge \quad \frac{\Gamma \Longrightarrow C}{\Gamma, \bot \supset B \Longrightarrow C} \supset L_\bot \quad \frac{\Gamma, A_1 \supset B, A_2 \supset B \Longrightarrow C}{\Gamma, (A_1 \vee A_2) \supset B \Longrightarrow C} \supset L_\vee$$

We need two additional rules, for atomic propositions and implications:

$$\frac{P \in \Gamma \quad \Gamma, B \Longrightarrow C}{\Gamma, P \supset B \Longrightarrow C} \supset L_P \quad \frac{\Gamma, (A_2 \supset B), A_1 \Longrightarrow A_2 \quad \Gamma, B \Longrightarrow C}{\Gamma, (A_1 \supset A_2) \supset B \Longrightarrow C} \supset L_\supset$$

The first rule is the usual function application rule, but restricted to atoms that are already in the context.

The second rule merits some explanation: if we were applying the original implication left rule $\supset L$, the first premise would be:

$$\Gamma, ((A_1 \supset A_2) \supset B) \Longrightarrow A_1 \supset A_2$$

Because we can always introduce an implication on the right, it is equivalent to prove

$$\Gamma, ((A_1 \supset A_2) \supset B), A_1 \Longrightarrow A_2$$

But *under the assumption of* A_1, $(A_1 \supset A_2) \equiv A_2$, so we can simplify the assumption to $(A_2 \supset B)$.

We do not give a formal termination proof for running these rules, but the idea is that each rule simplifies the implication assumption, so it is no longer possible to loop.

Task 3 (15 pts). Implement the contraction-free sequent calculus in Prolog:

- **Remove** the clause corresponding to $\supset L$.
- **Add** six new clauses for $\supset L_\top, \supset L_\wedge, \supset L_\bot, \supset L_\vee, \supset L_P, \supset L_\supset$.

Your implementation should work for the tests provided in the support code, including the law-of-the-excluded-middle example that loops above.

3 Inversion

3.1 Invertible Left Rules

Even though it is sound and complete, your implementation is inefficient in a couple of ways. We will use the idea of *invertibility* to fix this. A rule is *invertible* if the conclusion implies the premise. Because the conclusion of an invertible rule is true iff the premises are, one can eagerly apply such a rule whenever it applies, without having to backtrack.

Task 1 (5 pts). Determine whether each left rule is invertible (hand in only your yes/no answers):

$\wedge L$ If $\Gamma, A \wedge B \Longrightarrow C$ then $\Gamma, A, B \Longrightarrow C$

$\top L$ If $\Gamma, \top \Longrightarrow C$ then $\Gamma \Longrightarrow C$.

$\vee L$ If $\Gamma, A \vee B \Longrightarrow C$ then $\Gamma, A \Longrightarrow C$ and $\Gamma, B \Longrightarrow C$.

$\bot L$ If $\Gamma, \bot \Longrightarrow C$ then true.

$\supset L_\top$ If $\Gamma, \top \supset B \Longrightarrow C$ then $\Gamma, B \Longrightarrow C$.

$\supset L_\wedge$ If $\Gamma, (A_1 \wedge A_2) \supset B \Longrightarrow C$ then $\Gamma, A_1 \supset (A_2 \supset B) \Longrightarrow C$.

$\supset L_\bot$ If $\Gamma, \bot \supset B \Longrightarrow C$ then $\Gamma \Longrightarrow C$.

$\supset L_\vee$ If $\Gamma, (A_1 \vee A_2) \supset B \Longrightarrow C$ then $\Gamma, A_1 \supset B, A_2 \supset B \Longrightarrow C$.

$\supset L_P$ If $\Gamma, P \supset B \Longrightarrow C$ then $P \in \Gamma$ and $\Gamma, B \Longrightarrow C$.

$\supset L_\supset$ If $\Gamma, (A_1 \supset A_2) \supset B \Longrightarrow C$ then $\Gamma, (A_2 \supset B), A_1 \Longrightarrow A_2$ and $\Gamma, B \Longrightarrow C$.

You should use weakening/cut/identity to try to prove these rules invertible. When that fails, you should be able to construct a simple counterexample. Hint: two rules are non-invertible, and the remainder are invertible.

3.2 Prolog Implementation

We can use invertibility to make our theorem prover more efficient. The idea is this: Whenever we add an assumption to the context, we immediately apply all invertible left rules to it.

This solves two sources of inefficiency:

1. **Unnecessary backtracking:** The above Prolog code will try applying the invertible left rules in many different orders, when in fact one will suffice.

2. **Linear context lookups:** Because the above code uses the `choose` relation to pick an assumption to work on, checking each left rule to see whether it applies can take time linear in the length of the context Γ. For example, to see if $\top L$ applies, you have to look through all of the assumptions for an occurrence of $\top$.

Here's how we will exploit invertibility:

- In `prove(G,A)`, restrict the context `G` to *stable* propositions, those whose left rules are **not** invertible.
- Define an auxiliary relation `invert(G,D,A)`, where `D` is a list of arbitrary propositions (some stable and some not). `invert` should process each proposition in `D`:
  - If its left-rule is invertible, apply it. For example, the case of `invert` for $\wedge L$ looks like this:

    ```
    invert(G,[and(A,B)|D],C) :-
            invert(G,[A,B|D],C).
    ```

  - If its left-rule is not invertible, move the proposition to `G` and process the rest of `D`.
- `prove` calls `invert` whenever it needs to add a potentially-invertible proposition to the context. `invert` calls prove when all propositions in `D` have been processed.
- The non-invertible left rules are still applied non-deterministically in `prove` to a proposition `choosen` from `G`, as in the code above. (There are no clear rules about when to use these non-invertible hypotheses.)

Task 2 (10 pts). Revise your theorem prover to handle the left-invertible rules in this manner.

4 Handin Instructions

- GNU Prolog is installed in

  ```
  /afs/andrew/course/15/317/bin/gprolog
  ```

  To run it, you must have this `bin` directory in your path.

  In `tcsh`, do this:

  ```
  setenv PATH "/afs/andrew/course/15/317/bin:$PATH"
  ```

In `bash`, do this:

```
export PATH="/afs/andrew/course/15/317/bin:$PATH"
```

You may want to add this to your startup files so that you don't have to reset your path every time you want to run Prolog.

Once you run Prolog, load your file using the `consult` command:

```
| ?- consult('yourfile.pl').
```

- To hand in your code, copy two files to your handin directory:

  Problem 2 (Dyckhoff):

  ```
  /afs/andrew/course/15/317/submit/<yourid>/hw07-2.pl
  ```

  Problem 3 (Dyckhoff + inversion):

  ```
  r/afs/andrew/course/15/317/submit/<yourid>/hw07-3.pl
  ```

  Submit the written problems in class or in `hw06.pdf`.