Constructive Logic (15-317), Fall 2009
Assignment 5: Sequent Calculus for Proof Search

William Lovas (wlovas@cs)

Out: Thursday, October 8, 2009
Due: Thursday, October 15, 2009 (before class)

In this assignment, you will explore the **G4ip** sequent calculus and see how it may be used to build a simple yet realistic theorem prover for intuitionistic propositional logic. By the end of the assignment, you will have implemented a sound and complete proof search procedure capable of proving automatically any of the propositional theorems you've proven manually this semester using Tutch.

The written portion of your work (Sections 1 and 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 3) should be submitted via AFS by copying your code to the directory

```
/afs/andrew/course/15/317/submit/<userid>/hw05
```

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

1 Manual Theorem Proving (10 points)

During the last few lectures, we have discussed various forms of sequent calculi and considered how they might be used to build a decision procedure for intuitionistic propositional logic. One particularly simple formulation amenable to implementation is Dyckhoff's contraction-free sequent calculus **G4ip**. (See Appendix A for a recap of the rules.)

First, let's get some practice using **G4ip** as a proof system.

Task 1 (5 pts)**.** Give a proof of the sequent $\cdot \longrightarrow ((P \supset Q) \supset R) \wedge ((P \supset Q) \supset S) \supset (P \supset Q) \supset R$ using the rules of **G4ip**. (P, Q, R, and S all stand for atomic propositions.)

Task 2 (5 pts)**.** Give a proof of the sequent $\cdot \longrightarrow ((P \vee Q) \wedge R \supset S) \supset R \wedge (Q \vee P) \supset S$ using the rules of **G4ip**. (Again, P, Q, R, and S all stand for atomic propositions.)

2 Cut Elimination (5 points)

The Cut Admissibility theorem is a crucial tool that helps us demonstrate the expressivity of our logic: if we can prove something, then we can use it as a lemma towards proving something else. We usually prove this theorem by a nested induction, first on the structure of the cut formula, and then on the structure of the two given derivations.

The **G4ip** calculus also admits such a theorem, and it is proven by roughly similar means (though some additional structural properties are required for the cases involving compound left rules).

Theorem (Cut Admissibility for **G4ip**)**.** If $\Gamma \longrightarrow A$ and $\Gamma, A \longrightarrow C$, then $\Gamma \longrightarrow C$.

Proof. By nested induction, first on the *weight* of the cut formula A, and then on the structure of the derivations $\mathcal{D}$ of $\Gamma \longrightarrow A$ and $\mathcal{E}$ of $\Gamma, A \longrightarrow C$. □

Task 3 (5 pts)**.** Prove the case of **G4ip**'s Cut Admissibility theorem where A is $P \supset B$, $\mathcal{D}$ is derived by $\supset R$, and $\mathcal{E}$ is derived by $P \supset L$. You may assume without proof the following structural properties of **G4ip**:

Lemma (Weakening)**.** If $\Gamma \longrightarrow C$, then $\Gamma, A \longrightarrow C$.

Lemma (Contraction)**.** If $\Gamma, A, A \longrightarrow C$, then $\Gamma, A \longrightarrow C$.

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

3 Automated Theorem Proving (25 points)

Because **G4ip**'s rules all reduce the "weight" of the formulas making up the sequent when read bottom-up, it is straightforward to see that it represents a decision procedure even without the benefit of loop checking. The rules themselves are non-deterministic, though, so one must invest some effort in extracting a deterministic implementation from them.

Task 4 (25 pts)**.** Implement a proof search procedure based on the **G4ip** calculus. Efficiency should not be a primary concern, but see the hints below regarding invertible rules. Strive instead for *correctness* and *elegance*, in that order.

You should write your implementation in Standard ML.[1] Some starter code is provided in the file `prop.sml` to clarify the setup of the problem and give you some basic tools for debugging (see Figure 1). Implement a structure `G4ip` matching the signature `G4IP`. A simple test harness assuming this structure is given in the structure `Test` in the file `test.sml`. Feel free to post any additional interesting test cases you encounter to the course bulletin board.

Here are some hints to help guide your implementation:

- Be sure to apply all invertible rules before you apply any non-invertible rules. Recall that the only non-invertible rules in **G4ip** are $\vee R_1$, $\vee R_2$, and $\supset\supset L$, but that $P\supset L$ and the init rule cannot always be applied asynchronously. One simple way to ensure that you do inversions first is to maintain a second context of non-invertible propositions and to process it only when the invertible context is exhausted.

- When it comes time to perform non-invertible search, you'll have to consider all possible choices you might make. Many theorems require you to use your non-invertible hypotheses in a particular order, and unless you try all possible orders, you may miss a proof.

- The provided test cases can help you catch many easy-to-make errors. Test your code early and often! If you come up with any interesting test cases of your own that help you catch other errors, we encourage you to share them via the course bulletin board.

There are many subtleties and design decisions involved in this task, so don't leave it until the last minute!

[1] If you are not comfortable writing in Standard ML, you should contact the instructors and the TA to work out an alternate arrangement.

```
signature PROP =
  sig
    datatype prop =                 (* A ::=            *)
        Atom of string              (*       P          *)
      | True                        (*     | T          *)
      | And of prop * prop          (*     | A1 & A2    *)
      | False                       (*     | F          *)
      | Or of prop * prop           (*     | A1 | A2    *)
      | Implies of prop * prop      (*     | A1 => A2   *)

    val Not : prop -> prop          (* ~A := A => F     *)

    val toString : prop -> string
  end

structure Prop :> PROP = ...

signature G4IP =
  sig
    (* [decide A = true] iff . ===> A has a proof,
       [decide A = false] iff . ===> A has no proof *)
    val decide : Prop.prop -> bool
  end
```

Figure 1: SML starter code for **G4ip** theorem prover.

A Complete G4ip Rules

Init Rule

$$\frac{}{\Gamma, P \longrightarrow P}\ \text{init}$$

Ordinary Rules

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

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

(no $\bot R$ rule) $\qquad \dfrac{}{\Gamma, \bot \longrightarrow C}\ \bot L$

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

$$\frac{\Gamma, A \longrightarrow B}{\Gamma \longrightarrow A \supset B}\ \supset R$$

Compound Left Rules

$$\frac{\Gamma, P, B \longrightarrow C}{\Gamma, P, P \supset B \longrightarrow C}\ P \supset L$$

$$\frac{\Gamma, B \longrightarrow C}{\Gamma, \top \supset B \longrightarrow C}\ \top \supset L \qquad \frac{\Gamma, D \supset E \supset B \longrightarrow C}{\Gamma, D \wedge E \supset B \longrightarrow C}\ \wedge \supset L$$

$$\frac{\Gamma \longrightarrow C}{\Gamma, \bot \supset B \longrightarrow C}\ \bot \supset L \qquad \frac{\Gamma, D \supset B, E \supset B \longrightarrow C}{\Gamma, D \vee E \supset B \longrightarrow C}\ \vee \supset L$$

$$\frac{\Gamma, D, E \supset B \longrightarrow E \quad \Gamma, B \longrightarrow C}{\Gamma, (D \supset E) \supset B \longrightarrow C}\ \supset \supset L$$