Constructive Logic (15-317), Fall 2012
Assignment 6: Sequent Calculus for Proof Search

Carlo Angiuli (`cangiuli@cs`)

Out: Thursday, October 18, 2012
Due: Thursday, October 25, 2012 (before class)

In this assignment, you will use the contraction-free, or **G4ip**, sequent calculus to build a simple yet realistic theorem prover for propositional constructive 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.

Your submission must include:

- A `README` file including your answer to Task 1, and a brief description of your solution to Task 2.
- Your implementation of the `G4ip` structure, as a solution to Task 2.

Your work should be submitted via AFS by copying your code to the directory

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

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

1 Automated Theorem Proving (40 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. The rules themselves are non-deterministic, though, so one must invest some effort in extracting a deterministic implementation from them.

Task 1 (5 pts)**.** Explain briefly why the **G4ip** calculus is more suitable for automated theorem proving than the original sequent calculus we presented in class. In particular, what do we gain from:

- distinguishing between invertible and non-invertible rules, and
- splitting the $\supset L$ rule into a specialized set of rules?

Task 2 (35 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.

In `README`, you must also briefly describe your implementation strategy.

We recommend writing your implementation in Standard ML. If you would like to use a different language, you **must** clear your choice with Carlo before submission.

Some starter SML 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`.

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!

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

```
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

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

Ordinary Rules

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

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

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

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

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

Compound Left Rules

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

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

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

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