Constructive Logic (15-317), Fall 2020
Assignment 9: Proof search in G4ip

Instructor: Karl Crary
TAs: Avery Cowan, Matthew McQuaid, Long Pham, Ariel Uy

Due: Friday, November 6, 2020, 11:59 pm

There is no written component this week. Submit your programming homework as a single `g4ip.sml` file to Autolab. **After submitting via Autolab, please check the submission's contents to ensure it contains what you expect. No points can be given to a submission that isn't there.**

1 Implementing a theorem prover

Two weeks ago we say how **g4ip** could be used to ensure a proof search terminates. Last week we introduced a method, inversion, for directing proof search to a few critical decision points. Hence, this week we will be extending Roy Dyckhoff's contraction-free sequent calculus. His calculus, called **g4ip**, relies on distinguishing the type of antecedent on an implication on the left. To perform efficient proof search, we are extending the inversion calculus with a new form of judgment to apply synchronous (non-invertible) rules; these include the right rules of right-synchronous connectives and the left rules of left-synchronous connectives. For reference, the rules are below. For further information, please see the notes posted along with the homework. They have an excellent description of combining **g4ip** with the inversion calculus, along with suggestions for implementing these rules as a decision procedure in a functional programming language.

Our forms of judgment are as follows:

$\Delta^-; \Omega \xrightarrow{\text{g4ip}}_{\mathsf{R}} C$	Decompose C on the right
$\Delta^-; \Omega \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+$	Decompose Ω on the left
$\Delta^- \xrightarrow{\text{g4ip}}_{\mathsf{S}} C^+$	Apply non-invertible rules

The rules of our version of **g4ip** are divided roughly into the inversion phases (right and left) and the search phase. Unlike in the inversion calculus, we have the search rule to make a formal distinction between $\Delta^-; \cdot \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+$ and $\Delta^- \xrightarrow{\text{g4ip}}_{\mathsf{S}} C^+$. Purposefully, the search phase does not have the secondary Ω context anymore, because search should only happen once all left-invertible propositions in Ω are processed and all left-noninvertible propositions in Ω are shifted into Δ.

Right Inversion

$$\frac{\Delta^-;\Omega \xrightarrow{\text{g4ip}}_{\mathsf{R}} A \quad \Delta^-;\Omega \xrightarrow{\text{g4ip}}_{\mathsf{R}} B}{\Delta^-;\Omega \xrightarrow{\text{g4ip}}_{\mathsf{R}} A \wedge B}\ \wedge\mathsf{R} \qquad \frac{\Delta^-;\Omega, A \xrightarrow{\text{g4ip}}_{\mathsf{R}} B}{\Delta^-;\Omega \xrightarrow{\text{g4ip}}_{\mathsf{R}} A \supset B}\ \supset\mathsf{R} \qquad \frac{}{\Delta^-;\Omega \xrightarrow{\text{g4ip}}_{\mathsf{R}} \top}\ \top\mathsf{R}$$

Switching Mode

$$\frac{\Delta^-;\Omega \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}{\Delta^-;\Omega \xrightarrow{\text{g4ip}}_{\mathsf{R}} C^+}\ \mathsf{LR}_+$$

Left Inversion

$$\frac{\Delta^-;\Omega, A, B \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}{\Delta^-;\Omega, A \wedge B \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}\ \wedge\mathsf{L} \qquad \frac{\Delta^-;\Omega, A \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+ \quad \Delta^-;\Omega, B \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}{\Delta^-;\Omega, A \vee B \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}\ \vee\mathsf{L} \qquad \frac{}{\Delta^-;\Omega, \bot \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}\ \bot\mathsf{L}$$

$$\frac{\Delta^-;\Omega \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}{\Delta^-;\Omega, \top \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}\ \top\mathsf{L}$$

Compound Left Invertible Rules

$$\frac{\Delta^-;\Omega, B \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}{\Delta^-;\Omega, \top \supset B \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}\ \top\supset\mathsf{L} \qquad \frac{\Delta^-;\Omega, A_1 \supset A_2 \supset B \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}{\Delta^-;\Omega, (A_1 \wedge A_2) \supset B \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}\ \wedge\supset\mathsf{L}$$

$$\frac{\Delta^-;\Omega, A_1 \supset B, A_2 \supset B \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}{\Delta^-;\Omega, (A_1 \vee A_2) \supset B \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}\ \vee\supset\mathsf{L} \qquad \frac{\Delta^-;\Omega \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}{\Delta^-;\Omega, \bot \supset B \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}\ \bot\supset\mathsf{L}$$

Shift and Search

$$\frac{\Delta^-, A^-;\Omega \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}{\Delta^-;\Omega, A^- \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}\ \text{shift} \qquad \frac{\Delta^- \xrightarrow{\text{g4ip}}_{\mathsf{S}} C^+}{\Delta^-;\cdot \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}\ \text{search}$$

Search Rules

$$\frac{P \in \Delta^-}{\Delta^- \xrightarrow{\text{g4ip}}_{\mathsf{S}} P}\ \text{init} \qquad \frac{\Delta^-;\cdot \xrightarrow{\text{g4ip}}_{\mathsf{R}} A}{\Delta^- \xrightarrow{\text{g4ip}}_{\mathsf{S}} A \vee B}\ \vee\mathsf{R}_1 \qquad \frac{\Delta^-;\cdot \xrightarrow{\text{g4ip}}_{\mathsf{R}} B}{\Delta^- \xrightarrow{\text{g4ip}}_{\mathsf{S}} A \vee B}\ \vee\mathsf{R}_2$$

Compound Left Search Rules

$$\frac{P \in \Delta^- \quad \Delta^-;B \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}{\Delta^-, P \supset B \xrightarrow{\text{g4ip}}_{\mathsf{S}} C^+}\ P\supset\mathsf{L} \qquad \frac{\Delta^-;A_2 \supset B, A_1 \xrightarrow{\text{g4ip}}_{\mathsf{R}} A_2 \quad \Delta^-;B \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+}{\Delta^-, (A_1 \supset A_2) \supset B \xrightarrow{\text{g4ip}}_{\mathsf{S}} C^+}\ \supset\supset\mathsf{L}$$

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.

Task 1. Implement a proof search procedure based on the **inversion extended g4ip**. 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][2]

Tip The rules themselves are non-deterministic, so one must invest some effort in extracting a deterministic implementation from them.

Some starter code is provided in the file `prop.sml`, included in this homework's handout, to clarify the setup of the problem and give you some basic tools for debugging.

```
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    *)
    | Imp of prop * prop         (*     | A1 => A2   *)

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

  val toString : prop -> string
  val toStringList : prop list -> string
  val eq : prop * prop -> bool
end

structure Prop :> PROP
```

Your task is implement a structure `G4ip` matching the signature `G4IP`. The signature has been provided below, and is included in the handout materials.

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

[1] If you are not comfortable writing in Standard ML, you should contact the instructors and the TAs for help.

[2] If you have concerns about the Standard ML language, you should study 15-312 and contact their instructor instead.

A simple test harness assuming this structure is given in the structure `Test` in the file `test.sml`, also included in the handout. Feel free to post any additional interesting test cases you encounter to Piazza.

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 non-invertible rules in **g4ip** are init, $\vee\mathsf{R}_1$, $\vee\mathsf{R}_2$, $P\supset\mathsf{L}$ and $\supset\supset\mathsf{L}$. Among these, init and $P\supset\mathsf{L}$ have somewhat special status: if they apply, we don't need to look back because there is no premise (init), or the sequent in the premise is provable whenever the conclusion is ($P\supset\mathsf{L}$).

  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 rules have been 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 on Piazza.

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