Constructive Logic (15-317), Spring 2020
Assignment 7:

Instructor: André Platzer
TAs: Avery Cowan, Cameron Wong, Carter Williams, Klaas Pruiksma

Due: Tuesday, March 24, 2020, 11:59 pm

In this assignment, you will complete the g4ip prover developed in Homework 6, and also get your feet wet with Prolog. There is no written component this week.

1 G4ip

Last week, we implemented inversion calculus for the implication-free fragment of propositional logic. However, implications are important! Hence, this week we will be implementing Roy Dyckhoff's contraction-free sequent calculus. This 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 recitation notes for this week. They have an excellent description for the rules of **g4ip**, along with directions for implementing them as a decision procedure in a functional programming language.

Our forms of judgment are as follows:

$\Gamma^-; \Omega \xrightarrow{\text{g4ip}}_{\mathsf{R}} C$	Decompose C on the right
$\Gamma^-; \Omega \xrightarrow{\text{g4ip}}_{\mathsf{L}} C^+$	Decompose Ω on the left
$\Gamma^- \xrightarrow{\text{g4ip}}_{\mathsf{S}} C^+$	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.

Right Inversion

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

Switching Mode

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

Left Inversion

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

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

Compound Left Invertible Rules

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

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

Shift and Search

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

Search Rules

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

Compound Left Search Rules

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

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.

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

Task 1 (40 points)**.** Implement a proof search procedure based on **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] 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.

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

```
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
```

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, $\vee R_1$, $\vee R_2$, $P\supset L$ and $\supset\supset L$. Among these, init and $P\supset L$ 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 ($P\supset L$).

  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!

2 Things Add Up

The *subset sum problem* is a popular programming challenge for introductory computer science courses. As we will see, it can be solved quite elegantly with Prolog.

Task 2 (2 points)**.** Write the predicate `sublist_sum(+L,+N,-S)` that takes in a list of integers L and an integer N that synthesizes a new list S which is a *sublist* of L that sums to N. Feel free to make use of the prolog predicates that can be found at `http://www.gprolog.org/manual/html_node/gprolog044.html`.

Your solution must also work correctly when moded at `sublist_sum(+L,+N,+S)`.

The *sublist relation*, $\sqsubseteq$, is defined by the following rules:

$$\frac{}{L \sqsubseteq L} \sqsubseteq R \qquad \frac{S \sqsubseteq L}{[X|S] \sqsubseteq [X|L]} \sqsubseteq B \qquad \frac{S \sqsubseteq L}{S \sqsubseteq [X|L]} \sqsubseteq C$$

You may also find the *sums to* predicate, written below as $\oplus$, to be useful:

$$\frac{}{[\,] \oplus 0} \oplus E \qquad \frac{S \oplus N \qquad X + N = M}{[X|S] \oplus M} \oplus +$$

Task 3 (3 points)**.** Implement the predicate `subset_sum(+L,+N,-S)`. It should behave the same as `sublist_sum`, but without the requirement that the synthesized list is in the correct order. Instead, we will just require that every element of S appears in L at least once per time it appears in S.

Your solution must also work correctly when moded at `subset_sum(+L,+N,+S)`.

Hint: Don't overthink it! The reference solutions fit on one line each.

3 Can't Get No Satisfaction

A major distinction between constructive and classical logic is the validity of truth tables as proofs. One can think of a truth table as applying the Law of the Excluded Middle to every atom, then casing on every disjunction and finishing each clause with a proof of every atom or its negation.

A common problem in logic is determining whether any boolean proposition (formally defined in a moment) with variables has any assignment of truth values to variables such that the formula evaluates to true. This is known as *boolean satisfiability*, or the SAT problem. The task of finding all assignments satisfying a given clause is a perfect match for Prolog's implicit backtracking and exhaustive search capabilities.

We will define boolean propositions P according to the following grammar:

$$P, Q := x \mid and(P, Q) \mid or(P, Q) \mid imp(P, Q) \mid neg(P)$$

with the usual meanings.

Task 4 (5 points)**.** Write the predicate `sat(+P,+V,-T)`, where P is a proposition and V is an ordered list of all variable atoms used in P. You should synthesize $T \sqsubseteq V$ (order matters!) such that P is true when:

- all atoms occurring in T are set to true, and
- all atoms occurring in V but not T are set to false.

We will say that T *satisfies* P when this is the case.

Finally, your implementation should also be correct when moded at `sat(+P,+V,+T)`.

To assist you with this task, we've included some inference rules defining the judgment $T \vDash P$, which is derivable when T satisfies P. Note that this judgment does not use the atom list V, so you may want to generate sublists to be checked against $\vDash$ for yourself.

$$\frac{T \vDash A \qquad T \vDash B}{T \vDash and(A,B)}\ and \qquad \frac{T \vDash A}{T \vDash or(A,B)}\ or1 \qquad \frac{T \vDash B}{T \vDash or(A,B)}\ or2 \qquad \frac{atom(A) \qquad A \in T}{T \vDash A}\ atom$$

$$\frac{T \nvDash A}{T \vDash neg(A)}\ neg \qquad \frac{T \vDash B}{T \vDash imp(A,B)}\ imp1 \qquad \frac{T \nvDash A}{T \vDash imp(A,B)}\ imp2$$

You may also find the predicate `atom/1` to be useful.

You may be concerned by the appearance of a novel, undefined judgment $T \nvDash A$. There are a few ways to resolve this. Firstly, we could define $T \nvDash A$ to mean that T satisfies the negation of A. Alternatively, we could translate our predicate language into a form without implication that only applies neg directly to atoms. The latter approach is closer to the standard presentation of the satisfiability problem in conjuctive normal form. You are welcome to handle $\nvDash$ in any way you'd like – the refsol uses neither of these approaches (hint: SAT is decidable).

Below are proposed rules for $\nvDash$:

$$\frac{T \nvDash A}{T \nvDash and(A,B)}\ and1 \qquad \frac{T \nvDash B}{T \nvDash and(A,B)}\ and2 \qquad \frac{T \nvDash A \qquad T \nvDash B}{T \nvDash or(A,B)}\ or \qquad \frac{T \vDash A \qquad T \nvDash B}{T \nvDash imp(A,B)}\ imp$$

$$\frac{T \vDash A}{T \nvDash neg(A)}\ neg$$

Below is a proposed translation from a language with imp and generalized neg to one with no imp and neg applied only to atoms.

$$\frac{A \rightsquigarrow A' \qquad B \rightsquigarrow B'}{and(A,B) \rightsquigarrow and(A',B')}\ and \qquad \frac{A \rightsquigarrow A' \qquad B \rightsquigarrow B'}{or(A,B) \rightsquigarrow or(A',B')}\ or \qquad \frac{atom(A)}{A \rightsquigarrow A}\ atom$$

$$\frac{neg(A) \rightsquigarrow C \qquad B \rightsquigarrow B'}{imp(A,B) \rightsquigarrow or(C,B')}\ imp \qquad \frac{neg(A) \rightsquigarrow C \qquad neg(B) \rightsquigarrow D}{neg(and(A,B)) \rightsquigarrow or(C,D)}\ neg-and$$

$$\frac{neg(A) \rightsquigarrow C \qquad neg(B) \rightsquigarrow D}{neg(or(A,B)) \rightsquigarrow and(C,D)}\ neg-or \qquad \frac{A \rightsquigarrow A'}{neg(neg(A)) \rightsquigarrow A'}\ neg-neg$$

$$\frac{A \rightsquigarrow A' \qquad neg(B) \rightsquigarrow C}{neg(imp(A,B)) \rightsquigarrow and(A',C)}\ neg-imp \qquad \frac{atom(A)}{neg(A) \rightsquigarrow neg(A)}\ neg-atom$$

We would also add the following rule to $\vDash$ to deal with atomic not.

$$\frac{atomic(A) \qquad A \notin T}{T \vDash not(A)}\ not$$

4 Bidirectional Typechecking

Your next task is to complete a proof checker written in Prolog. The checker given to you in the starter code is currently able to check proof terms using conjunctions and implications only.

Task 5 (10 points)**.** Extend the given proof checker to correctly check terms containing disjunctions, verum, falsum, and cut (let-expressions). That is, complete the `check` and `synth` predicates.

The algorithmic typing rules for this checker are given below:

$$\frac{\Omega \vdash M : A\uparrow \qquad \Omega \vdash N : B\uparrow}{\Omega \vdash (M, N) : A \wedge B\uparrow}\ \wedge I \qquad \frac{(u : A\downarrow) \cdot \Omega \vdash M : B\uparrow}{\Omega \vdash (\mathsf{fn}\ u \Rightarrow M) : A \supset B\uparrow}\ \supset I^u \qquad \frac{\Omega \vdash R : A\downarrow}{\Omega \vdash R : A\uparrow}\ \downarrow\uparrow$$

$$\frac{}{(u : A\downarrow) \cdot \Omega \vdash u : A\downarrow}\ \mathsf{var}_= \qquad \frac{w \neq u \quad \Omega \vdash u : A\downarrow}{(w : B\downarrow) \cdot \Omega \vdash u : A\downarrow}\ \mathsf{var}_\neq \qquad \frac{\Omega \vdash R : A \wedge B\downarrow}{\Omega \vdash \mathsf{fst}\, R : A\downarrow}\ \wedge E_1$$

$$\frac{\Omega \vdash R : A \wedge B\downarrow}{\Omega \vdash \mathsf{snd}\, R : B\downarrow}\ \wedge E_2 \qquad \frac{\Omega \vdash R : A \supset B\downarrow \qquad \Omega \vdash M : A\uparrow}{\Omega \vdash R\, M : B\downarrow}\ \supset E \qquad \frac{\Omega \vdash M : A\uparrow}{\Omega \vdash \mathsf{inl}\, M : A \vee B\uparrow}\ \vee I_1$$

$$\frac{\Omega \vdash M : B\uparrow}{\Omega \vdash \mathsf{inr}\, M : B \vee B\uparrow}\ \vee I_2 \qquad \frac{\Omega \vdash R : A \vee B\downarrow \qquad (u : A\downarrow) \cdot \Omega \vdash N : C\uparrow \qquad (v : B\downarrow) \cdot \Omega \vdash M : C\uparrow}{\Omega \vdash \mathsf{case}\ R\ \mathsf{of}\ u.N \mid v.M : C\uparrow}\ \vee E$$

$$\frac{}{\Omega \vdash (\,) : \top\uparrow}\ \top I \qquad \frac{\Omega \vdash R : \bot\downarrow}{\Omega \vdash abort(R) : C\uparrow}\ \bot E \qquad \frac{\Omega \vdash M : A\uparrow \quad (u : A\downarrow) \cdot \Omega \vdash N : C\uparrow}{\Omega \vdash \mathsf{let}\ u : A = M\ \mathsf{in}\ N\ \mathsf{end} : C\uparrow}\ cut$$

How to Submit

Your code should be in two files:

- `hw7.pl`, and
- `inversion_calculus.sml`

Please submit a tar archive containing `hw7.pl` to the "Homework 7 - Prolog" assignment on autolab, and a tar archive containing `inversion_calculus.sml` to "Homework 7 - G4ip (SML)". If there are other files in the archive submitted, they will be ignored.