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:

- \( \Gamma; \Omega \xrightarrow{\text{g4ip}_R} C \) Decompose \( C \) on the right
- \( \Gamma; \Omega \xrightarrow{\text{g4ip}_L} C^+ \) Decompose \( \Omega \) on the left
- \( \Gamma \xrightarrow{\text{g4ip}_S} C^+ \) 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

\[
\frac{\Gamma; \Omega \xrightarrow{\text{g4ip}_R} A \quad \Gamma; \Omega \xrightarrow{\text{g4ip}_R} B}{\Gamma; \Omega \xrightarrow{\text{g4ip}_R} A \land B} \quad \wedge R
\]

\[
\frac{\Gamma; \Omega, A \xrightarrow{\text{g4ip}_R} B}{\Gamma; \Omega \xrightarrow{\text{g4ip}_R} A \supset B} \quad \supset R
\]

\[
\frac{\Gamma; \Omega \xrightarrow{\text{g4ip}_R} \top}{\Gamma; \Omega \xrightarrow{\text{g4ip}_R} \top} \quad \top R
\]

### Switching Mode

\[
\frac{\Gamma; \Omega \xrightarrow{\text{g4ip}_L} C^+}{\Gamma; \Omega \xrightarrow{\text{g4ip}_R} C^+} \quad LR^+_R
\]

### Left Inversion
\[
\Gamma \vdash \Omega, A, B \xrightarrow{\text{g4ip}}_{gL} C^+ \quad \land
\]
\[
\Gamma \vdash \Omega, A \land B \xrightarrow{\text{g4ip}}_{gL} C^+
\]
\[
\Gamma \vdash \Omega, A \lor B \xrightarrow{\text{g4ip}}_{gL} C^+ \quad \lor
\]
\[
\Gamma \vdash \Omega, A \lor B \xrightarrow{\text{g4ip}}_{gL} C^+
\]
\[
\Gamma \vdash \Omega, \bot \xrightarrow{\text{g4ip}}_{gL} C^+ \quad \bot
\]
\[
\Gamma \vdash \Omega, \top \xrightarrow{\text{g4ip}}_{gL} C^+
\]

**Compound Left Invertible Rules**

\[
\Gamma \vdash \Omega, B \xrightarrow{\text{g4ip}}_{gL} C^+ \Rightarrow
\]
\[
\Gamma \vdash \Omega, B \xrightarrow{\text{g4ip}}_{gL} C^+ \quad \top \Rightarrow
\]
\[
\Gamma \vdash \Omega, A \lor B \xrightarrow{\text{g4ip}}_{gL} C^+ \Rightarrow
\]
\[
\Gamma \vdash \Omega, (A \land B) \xrightarrow{\text{g4ip}}_{gL} C^+ \quad \bot \Rightarrow
\]

**Shift and Search**

\[
\Gamma \vdash \Omega, A \xrightarrow{\text{g4ip}}_{gL} C^+ \quad \text{shift}
\]
\[
\Gamma \vdash \Omega, A \xrightarrow{\text{g4ip}}_{gL} C^+ \quad \text{search}
\]

**Search Rules**

\[
\frac{P \in \Gamma}{\Gamma \xrightarrow{\text{g4ip}}_{S} P} \quad \text{init}
\]
\[
\frac{A \in \Gamma}{\Gamma \xrightarrow{\text{g4ip}}_{S} A} \quad \lor R_1
\]
\[
\frac{A \in \Gamma}{\Gamma \xrightarrow{\text{g4ip}}_{S} A} \quad \lor R_2
\]

**Compound Left Search Rules**

\[
\frac{P \in \Gamma}{\Gamma, P \supset B \xrightarrow{\text{g4ip}}_{S} C^+} \quad \supset \Rightarrow
\]
\[
\frac{A_2 \supset B, A_1 \xrightarrow{\text{g4ip}}_{S} A_2 \supset B \xrightarrow{\text{g4ip}}_{S} C^+}{\Gamma, (A_1 \supset A_2) \supset B \xrightarrow{\text{g4ip}}_{S} C^+} \quad \supset \Rightarrow
\]

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. 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 =
    Atom of string (* A ::= *)
  | 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, ∨R₁, ∨R₂, P⊢L and ⊢L. Among these, init and P⊢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⊢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, \( \subseteq \), is defined by the following rules:

\[
\frac{L \subseteq L}{R} \\
\frac{S \subseteq L}{X | S \subseteq |X|L} \subseteq B \\
\frac{S \subseteq L}{S \subseteq |X|L} \subseteq C
\]

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

\[
\begin{align*}
\emptyset \oplus 0 & \oplus E \\
S \oplus N & \quad X + N = M \\
|X|S \oplus M & \oplus +
\end{align*}
\]

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 \( T \subseteq V \) (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 $T \models P$, 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.

$$
\frac{T \vdash A \quad T \vdash B}{T \vdash \text{and}(A, B)} \quad \frac{T \vdash A}{T \vdash \text{or}(A, B)} \quad \frac{T \vdash B}{T \vdash \text{or}(A, B)} \quad \frac{\text{atom}(A)}{T \vdash A}
$$

You may also find the predicate $\text{atom}/1$ to be useful.

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

Below are proposed rules for $\not\models$:

$$
\frac{T \not\models A}{T \not\models \text{and}(A, B)} \quad \frac{T \not\models B}{T \not\models \text{and}(A, B)} \quad \frac{T \not\models A \quad T \not\models B}{T \not\models \text{or}(A, B)} \quad \frac{T \models A}{T \not\models \text{imp}(A, B)}
$$

Below is a proposed translation from a language with $\text{imp}$ and generalized $\text{neg}$ to one with no $\text{imp}$ and $\text{neg}$ applied only to atoms.

$$
\frac{A \leadsto A' \quad B \leadsto B'}{A \leadsto \text{and}(A, B) \leadsto \text{and}(A', B')} \quad \frac{A \leadsto A' \quad B \leadsto B'}{A \leadsto \text{or}(A, B) \leadsto \text{or}(A', B')} \quad \frac{\text{atom}(A)}{A \leadsto A}
$$

$$
\frac{\text{neg}(A) \leadsto C \quad B \leadsto B'}{\text{imp}(A, B) \leadsto \text{or}(C, B')} \quad \frac{\text{neg}(A) \leadsto C \quad \text{neg}(B) \leadsto D}{\text{imp}(\text{and}(A, B)) \leadsto \text{or}(C, D)} \quad \frac{\text{neg}(A) \leadsto C \quad \text{neg}(B) \leadsto D}{\text{neg} \quad \text{and}} \quad \frac{\text{neg}(A) \leadsto C \quad \text{neg}(B) \leadsto D}{\text{neg} \quad \text{or}} \quad \frac{A \leadsto A'}{\text{neg}(\text{neg}(A)) \leadsto A'} \quad \frac{A \leadsto A'}{\text{neg} \quad \text{neg} \quad \text{atom}}
$$

Below is a proposed translation from a language with $\text{imp}$ and generalized $\text{neg}$ to one with no $\text{imp}$ and $\text{neg}$ applied only to atoms.

$$
\frac{A \leadsto A' \quad B \leadsto B'}{A \leadsto \text{and}(A, B) \leadsto \text{and}(A', B')} \quad \frac{A \leadsto A' \quad B \leadsto B'}{A \leadsto \text{or}(A, B) \leadsto \text{or}(A', B')} \quad \frac{\text{atom}(A)}{A \leadsto A}
$$

$$
\frac{\text{neg}(A) \leadsto C \quad B \leadsto B'}{\text{imp}(A, B) \leadsto \text{or}(C, B')} \quad \frac{\text{neg}(A) \leadsto C \quad \text{neg}(B) \leadsto D}{\text{imp}(\text{and}(A, B)) \leadsto \text{or}(C, D)} \quad \frac{\text{neg}(A) \leadsto C \quad \text{neg}(B) \leadsto D}{\text{neg} \quad \text{and}} \quad \frac{\text{neg}(A) \leadsto C \quad \text{neg}(B) \leadsto D}{\text{neg} \quad \text{or}} \quad \frac{A \leadsto A'}{\text{neg}(\text{neg}(A)) \leadsto A'} \quad \frac{A \leadsto A'}{\text{neg} \quad \text{neg} \quad \text{atom}}
$$

Below is a proposed translation from a language with $\text{imp}$ and generalized $\text{neg}$ to one with no $\text{imp}$ and $\text{neg}$ applied only to atoms.

$$
\frac{A \leadsto A' \quad B \leadsto B'}{A \leadsto \text{and}(A, B) \leadsto \text{and}(A', B')} \quad \frac{A \leadsto A' \quad B \leadsto B'}{A \leadsto \text{or}(A, B) \leadsto \text{or}(A', B')} \quad \frac{\text{atom}(A)}{A \leadsto A}
$$

$$
\frac{\text{neg}(A) \leadsto C \quad B \leadsto B'}{\text{imp}(A, B) \leadsto \text{or}(C, B')} \quad \frac{\text{neg}(A) \leadsto C \quad \text{neg}(B) \leadsto D}{\text{imp}(\text{and}(A, B)) \leadsto \text{or}(C, D)} \quad \frac{\text{neg}(A) \leadsto C \quad \text{neg}(B) \leadsto D}{\text{neg} \quad \text{and}} \quad \frac{\text{neg}(A) \leadsto C \quad \text{neg}(B) \leadsto D}{\text{neg} \quad \text{or}} \quad \frac{A \leadsto A'}{\text{neg}(\text{neg}(A)) \leadsto A'} \quad \frac{A \leadsto A'}{\text{neg} \quad \text{neg} \quad \text{atom}}
$$

Below is a proposed translation from a language with $\text{imp}$ and generalized $\text{neg}$ to one with no $\text{imp}$ and $\text{neg}$ applied only to atoms.

$$
\frac{A \leadsto A' \quad B \leadsto B'}{A \leadsto \text{and}(A, B) \leadsto \text{and}(A', B')} \quad \frac{A \leadsto A' \quad B \leadsto B'}{A \leadsto \text{or}(A, B) \leadsto \text{or}(A', B')} \quad \frac{\text{atom}(A)}{A \leadsto A}
$$

$$
\frac{\text{neg}(A) \leadsto C \quad B \leadsto B'}{\text{imp}(A, B) \leadsto \text{or}(C, B')} \quad \frac{\text{neg}(A) \leadsto C \quad \text{neg}(B) \leadsto D}{\text{imp}(\text{and}(A, B)) \leadsto \text{or}(C, D)} \quad \frac{\text{neg}(A) \leadsto C \quad \text{neg}(B) \leadsto D}{\text{neg} \quad \text{and}} \quad \frac{\text{neg}(A) \leadsto C \quad \text{neg}(B) \leadsto D}{\text{neg} \quad \text{or}} \quad \frac{A \leadsto A'}{\text{neg}(\text{neg}(A)) \leadsto A'} \quad \frac{A \leadsto A'}{\text{neg} \quad \text{neg} \quad \text{atom}}
$$

We would also add the following rule to $\models$ to deal with atomic $\text{not}$.

$$
\frac{\text{atomic}(A)}{T \models \text{not}(A)}
$$

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:

\[ \begin{align*}
\frac{\Gamma \vdash M : A \uparrow}{\Gamma \vdash (M, N) : A \land B \uparrow} \quad &\quad \land I \\
\frac{(u : A \downarrow) \cdot \Gamma \vdash M : B \uparrow}{\Gamma \vdash (\text{fn } u \Rightarrow M) : A \supset B \uparrow} \quad &\quad \supset I_u \\
\frac{\Gamma \vdash R : A \downarrow}{\Gamma \vdash R : A \uparrow} \quad &\quad \downarrow \uparrow \\
\frac{\Gamma \vdash R : A \land B \downarrow}{\Gamma \vdash \text{snd } R : B \downarrow} \quad &\quad \land E_2 \\
\frac{\Gamma \vdash R : A \supset B \downarrow}{\Gamma \vdash R \cdot M : B \downarrow} \quad &\quad \supset E \\
\frac{\Gamma \vdash M : B \uparrow}{\Gamma \vdash \text{inl } M : A \lor B \uparrow} \quad &\quad \lor I_2 \\
\frac{\Gamma \vdash M : B \uparrow}{\Gamma \vdash \text{inr } M : B \uparrow} \quad &\quad \lor E \\
\frac{\Gamma \vdash (): \top \uparrow}{\Gamma \vdash \text{abort}(R) : C \uparrow} \quad &\quad \perp E \\
\frac{\Gamma \vdash M : A \uparrow (u : A \downarrow) \cdot \Gamma \vdash N : C \uparrow}{\Gamma \vdash \text{let } u : A = M \end{\text{in } N} : C \uparrow} \quad &\quad \text{cut}
\end{align*} \]

**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.