# Constructive Logic (15-317), Spring 2022
# Assignment 7: Theorem Proving

Instructor: Klaas Pruiksma
TAs: Runming Li, Onyekachi Onyeador, Viraj Puri, Xiao Yu

Submit to Gradescope by Thursday, 17 March, 2022, 23:59

The assignments in this course must be submitted electronically through Gradescope. This homework is a two-week long assignment.

We recommend that you start this work well before the deadline since it does represent more work than a normal assignment.

For this homework, you will only be submitting SML files:

- `hw7.sml` (your coding solutions)

## Implementing a theorem prover

You might have noticed, after some practice, that proving a theorem in a calculus becomes quite a mechanical task. Wouldn't it be great if we could have the computer do that for us? That is exactly our goal for this homework: to implement an automatic theorem prover for propositional intuitionistic logic.

The first thing to think about is which calculus we will use. The holy grail of constructive logic would be a prover that could automatically generate a natural deduction proof for an arbitrary proposition. However, it should be clear by now that natural deduction is not the best choice for search, as it is too non-deterministic. The verification calculus could be a bit better, as it avoids the redundant steps that eliminate and introduce the same connective over and over again, but it still has the problem of keeping track of the right assumptions at the right places. Maybe we should try the context-style presentation of natural deduction, since this keeps the context in place. In this case we need to be very smart about which direction to work at each step, since we can either go upwards or downwards. Instead of trying to come up with heuristics for that, why don't we use the sequent calculus itself, where proof construction always happens from the bottom up?

Indeed, sequent calculi are much better behaved for proof search. But we need to be careful about it. Think about the first sequent calculus we have seen. In this first version, the formulas on the left side of the sequent were persistent. This means we can *always* choose to decompose those formulas. In fact, any sequent calculus that has what we call *implicit contraction*[1] of some formulas runs into the same problem. The inversion calculus avoids these problems. This calculus refines the restricted sequent calculus into two mutually dependent forms of sequents.

$$\Delta^- \,;\, \Omega \xrightarrow{R} C \qquad \text{Decompose } C \text{ on the right}$$
$$\Delta^- \,;\, \Omega \xrightarrow{L} C^+ \qquad \text{Decompose } \Omega \text{ on the left}$$

Above, $\Omega$ is a plain context containing any kind of formula. $\Delta^-$ is a context restricted to those formulas whose left rules are *not* invertible, and $C^+$ in the second sequent is a formula whose right rule is *not* invertible. Both types of sequents can also contain atoms. All the rules of the inversion calculus have the property that the "weight" of the sequents decrease when the rules are read bottom-up, *except* the left rule for implication. Use of the implication left rule can lead to looping because we can continue to try using the left implication rule on the same implication in the premise. As such, we will be extending inversion to use 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 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:

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

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 $\Delta^- ; \cdot \xrightarrow{\mathsf{g4ip}}_{\mathsf{L}} C^+$ and $\Delta^- \xrightarrow{\mathsf{g4ip}}_{\mathsf{S}} C^+$. Purposefully, the search phase does not have the secondary

[1] Usually in the form of applying a rule to decompose a formula and keeping a copy of the original formula in the context.

$\Omega$ context anymore, because search should only happen once all left-invertible propositions in $\Omega$ are processed and all left-noninvertible propositions in $\Omega$ are shifted into $\Delta$. For clarity, we can formally define the polarities mentioned in the rules. Positive and negative correspond exactly with non-left invertible propositions and non-right invertible propositions.

$$C^+ ::= A \vee B \mid \bot \mid P$$
$$C^- ::= (A_1 \supset A_2) \supset B \mid P \supset B \mid P$$

Above, $P$ is considered atomic while all other metavariables are arbitrary propositions. Based on these polarizations, the negative context $\Delta^-$ is composed entirely of negative propositions.

For reference, the rules for **g4ip** are below.

**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**. This algorithm should take in a right inversion judgement and produce an inversion with **g4ip** proof option. 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 the file `hw7.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 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 ($P \supset L$).
- You may find it helpful to implement each judgement as its own function, rather than trying to write a single function that handles all three judgements together.
- 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.
- Since you will be creating your own proof trees algorithmically, make sure to read the `inversion_g4ip.sml` file carefully. We also won't have the usual macros to create a proof tree so looking at `logic.sml` might also be helpful.
- Generating a proof can be difficult and create incredibly nested cases. As such, we highly suggest using continuation passing style (CPS) to simplify your code. For a refresher on CPS, please feel free to look at the 15-150 notes on the topic. In particular, you may find the concept of *failure continuations* useful here.
- Your code should be elegant and use good style. Please look at the style guide from the pre-requisite course 15-150 to guide your stylistic decisions in SML.
- The rules themselves are non-deterministic, so one must invest some effort in extracting a deterministic implementation from them. Planning before coding will likely save time.
- Test your code early and often! If you come up with any interesting test cases that help you catch errors, we encourage you to share them on Piazza. We may assign small amounts of extra credit for particularly interesting test cases that are shared.

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