# Constructive Logic (15-317), Spring 2021
# Assignment 10: Focusing and Chaining

Instructor: Karl Crary
TAs: Avery Cowan, Katherine Cordwell, Matias Scharager, Antian Wang

Due: Wednesday, November 17, 11:59 pm

The assignments in this course must be submitted electronically through Gradescope. For this homework, you will be submitting both written pdf files and Dcheck coding files:

- `hw.deriv` (your coding solutions)
- `hw.pdf` (your written solutions)

The coding portion will use the experimental Dcheck derivation checker. You can find documentation and examples on the Software page at the course web site (`cs.cmu.edu/~crary/317-f21/software.html`). That document has been updated with information on preparing focused logic derivations.

## Focusing and Chaining

A major theme of this course has been the discovery of theory through practice: strategies for efficient proof search in the concrete conditions of real-world implementations are transformed into razor-edged intellectual weapons, entirely new logics which sharpen the principal contradiction of proof theory: the dialectic of the *positive* and *negative* (polarity).

The decomposition of *truth* into *verification* and *use* was our first encounter with the scientific law, "One Divides Into Two". By studying invertibility in the context of the sequent calculus (when does a conclusion imply its premises?), we were able to achieve a firmer grasp of the fault-lines at play, summarized in a dangerously over-simplified[1] form below:

| | LEFT RULE | RIGHT RULE |
|---|---|---|
| POSITIVE | invertible | **non-invertible** |
| NEGATIVE | **non-invertible** | invertible |

**Inversion** Invertible rules can always be applied without any need for backtracking: since the conclusion of an invertible rule implies its premises, the "future truth" of the goal is preserved under free application of such rules. This practical insight, which is crucial for implementing a performant proof search engine, can be codified by sharpening the logic to include deterministic inversion phases $\Gamma; \Omega \xrightarrow{L} C$ and $\Gamma; \Omega \xrightarrow{R} C$ (where $\Omega$ is an ordered context of propositions).

**Chaining** While the above gives a clear and deterministic account of invertible rules, the non-invertible ones beg for something similar. In this week's lecture, we began to study *chaining*, which fixes a dynamics for the non-invertible rules based on two forms of judgment, $\Gamma \longrightarrow [A^+]$ and $\Gamma; [A^-] \longrightarrow C$. Chaining is a technique to minimize backtracking by applying a sequence of non-invertible rules in one go.

## 1 Polarization

**Task 1** (5 pts)**.** Consider the following depolarized formula:

$$((A^+ \vee B^-) \supset F) \supset ((A^+ \supset F) \wedge (B^- \supset F))$$

Come up with two *distinct* polarizations of the formula, adding shifts in appropriate places. You do not need to prove them. (This is a written problem. Include your answer in `hw.pdf`.)

## 2 Focusing

Provide derivations of the following Focused Logic judgements using Dcheck syntax.

**Task 2** (5 pts)**.** Define a derivation named `task2` that derives:

$$\cdot; \cdot \xrightarrow{R} \downarrow((P^+ \supset Q^-) \wedge^- (P^+ \supset R^-)) \supset (P^+ \supset (Q^- \wedge^- R^-))$$

[1] In *structural* or *persistent* logic, some rules which ought to be non-invertible turn out to be invertible; polarity arises properly from the proof search dynamics of *linear logic*, and casts an imperfect shadow in persistent logic.

**Task 3** (5 pts)**.** Define a derivation named `task3` that derives:

$$\cdot;\cdot \xrightarrow{R} \downarrow P^- \supset \downarrow(\downarrow P^- \supset Q^-) \supset \downarrow((\downarrow P^- \wedge^+ \downarrow Q^-) \supset R^-) \supset R^-$$

**Task 4** (5 pts)**.** Define a derivation named `task4` that derives:

$$\cdot;\cdot \xrightarrow{R} \downarrow(\uparrow P^+ \wedge^- (P^+ \supset \uparrow Q^+)) \supset \downarrow((Q^+ \vee R^+) \supset \uparrow R^+) \supset \uparrow R^+$$

# 3 Saturation

Consider the following grammar of ground terms representing binary numbers:

$$n \quad ::= \quad \epsilon \mid \mathsf{b0}(n) \mid \mathsf{b1}(n)$$

In class, we learned to write forward logic programs using inference rules; a forward logic programming engine will apply these inference rules until saturation is reached, and then the result of our program can be read from the saturated proof state. In the tasks that follow, you are free to introduce any auxiliary predicates that you require. You need to ensure that your rules *saturate* when new facts of the indicated form are added to the database.

In the problems that follow, you are required to implement forward logic programs by writing down systems of inference rules. You may find it useful to experiment with **DLV**, an implementation of forward logic programming which can be downloaded here: `http://www.dlvsystem.com/dlv/`. **DLV** can be used to test your ideas on specific cases and quickly determine if they are likely to work; but it is not required.

**Task 5** (5 pts)**.** Implement a forward logic program $\mathsf{std}(n)$ which derives the atom `no` iff it is not the case that $n$ is in standard form. You may assume that $n$ is ground (i.e. not subject to unification).

**Task 6** (5 pts)**.** Next, implement a forward logic program $\mathsf{succ}(m, n)$ which derives `no` when it is not the case that $m + 1 = n$. For the purpose of this exercise, you may assume that $m$ and $n$ are ground. You may also assume that $m$ and $n$ are in standard form.