Constructive Logic (15-317), Fall 2020
Assignment 11: Linear Logic, Focusing, and Chaining

TAs: Avery Cowan, Matthew McQuaid, Long Pham, Ariel Uy

Due: Friday, December 4, 11:59 pm

Submit your homework to Gradescope as a PDF.

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 $\Gamma; \Omega \longrightarrow_{\mathsf{L}} C$ and $\Gamma; \Omega \longrightarrow_{\mathsf{R}} C$ (where Ω 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] 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.

1 Practicing focusing

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

$$\neg(a^+ \vee b^-) \supset \neg a^+ \wedge \neg b^-$$

Come up with two *distinct* polarizations of the formula, adding shifts in the appropriate places; you do not need to prove them. Remember that in ordinary depolarized constructive logic, $\neg A$ means $A \supset F$.

Task 2 (10 pts)**.** Construct a derivation in focused logic for the following sequent:

$$\cdot;\cdot \longrightarrow_{\mathsf{R}} \downarrow\big(\big(a^+ \supset b^-\big) \wedge^- \big(a^+ \supset c^-\big)\big) \supset \big(a^+ \supset \big(b^- \wedge {}^-c^-\big)\big)$$

2 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 3 (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 4 (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.

3 Practicing Linear Logic

For your convenience, we supply a self-contained presentation of the rules of linear logic.

Structural Rules

In the presentation of linear logic that we use in this course, contexts Γ should be taken as unordered lists; therefore, the principle of *exchange* is automatic. Weakening and contraction are *not* included in linear logic.

$$\frac{}{u : A \textit{ true} \Vdash A \textit{ true}}\ \mathit{lhyp}$$

Multiplicative Conjunction

$$\frac{\Gamma \Vdash A\ true \quad \Gamma' \Vdash B\ true}{\Gamma, \Gamma' \Vdash A \otimes B\ true}\ \otimes\mathsf{I} \qquad \frac{\Gamma \Vdash A \otimes B\ true \quad \Gamma', u : A\ true, v : B\ true \Vdash C\ true}{\Gamma, \Gamma' \Vdash C\ true}\ \otimes\mathsf{E}^{u,v}$$

$$\frac{}{\cdot \Vdash \mathbf{1}\ true}\ \mathbf{1}\mathsf{I} \qquad \frac{\Gamma \Vdash \mathbf{1}\ true \quad \Gamma' \Vdash C\ true}{\Gamma, \Gamma' \Vdash C\ true}\ \mathbf{1}\mathsf{E}$$

Additive Conjunction

$$\frac{\Gamma \Vdash A\ true \quad \Gamma \Vdash B\ true}{\Gamma \Vdash A \mathbin{\&} B\ true}\ \&\mathsf{I} \qquad \frac{\Gamma \Vdash A \mathbin{\&} B\ true}{\Gamma \Vdash A\ true}\ \&\mathsf{E}_1 \qquad \frac{\Gamma \Vdash A \mathbin{\&} B\ true}{\Gamma \Vdash B\ true}\ \&\mathsf{E}_2$$

$$\frac{}{\Gamma \Vdash \top\ true}\ \top\mathsf{I}$$

There is no elimination rule for the unit $\top$.

Additive Disjunction

$$\frac{\Gamma \Vdash A\ true}{\Gamma \Vdash A \oplus B\ true}\ \oplus\mathsf{I}_1 \qquad \frac{\Gamma \Vdash B\ true}{\Gamma \Vdash A \oplus B\ true}\ \oplus\mathsf{I}_2$$

$$\frac{\Gamma \Vdash A \oplus B\ true \quad \Gamma', u : A\ true \Vdash C\ true \quad \Gamma', v : B\ true \Vdash C\ true}{\Gamma, \Gamma' \Vdash C\ true}\ \oplus\mathsf{E}^{u,v}$$

$$\frac{\Gamma \Vdash \mathbf{0}\ true}{\Gamma, \Gamma' \Vdash C\ true}\ \mathbf{0}\mathsf{E}$$

There is no introduction rule for the unit $\mathbf{0}$.

Implication

$$\frac{\Gamma, u : A\ true \Vdash B\ true}{\Gamma \Vdash A \multimap B\ true}\ \multimap\mathsf{I}^u \qquad \frac{\Gamma \Vdash A \multimap B\ true \quad \Gamma' \Vdash A\ true}{\Gamma, \Gamma' \Vdash B\ true}\ \multimap\mathsf{E}$$

Task 5 (15 pts)**.** Prove the following judgments in linear natural deduction, or state that they do not hold.

1. $f : A \multimap B \multimap C\ true \Vdash A \otimes B \multimap C\ true$
2. $f : (A \mathbin{\&} B) \multimap C\ true \Vdash A \multimap (B \multimap C)\ true$
3. $f : ((A \otimes \top) \mathbin{\&} (B \otimes \top)) \multimap C\ true \Vdash A \multimap (B \multimap C)\ true$

4 Proof-Theoretic Harmony

Just like we did in the beginning of the course, we can check a local correctness condition for the rules of linear natural deduction: proof-theoretic harmony.[2] Hint: exhibiting local reductions and expansions in linear logic is subtle: you must be sure to not constrain the contexts Γ in your local reductions and expansions any more than is warranted by the rules of linear logic.

Remark 1 (Linear substitution principle)**.** When exhibiting local reductions and expansions, you will need to use substitutions $[\mathcal{D}/u]\mathcal{E}$. These are governed by the *linear substitution principle*, which states:

$$\text{If } \Gamma \Vdash \overset{\mathcal{D}}{A \textit{ true}} \text{ and } \Gamma', u : A \textit{ true} \Vdash \overset{\mathcal{E}}{B \textit{ true}}, \text{ then } \Gamma, \Gamma' \Vdash \overset{[\mathcal{D}/u]\mathcal{E}}{B \textit{ true}}.$$

You *must* ensure that your resulting derivations have the correct contexts.

Task 6 (10 pts)**.** Verify that the rules for the tensor $\otimes$ are harmonious.

Task 7 (10 pts)**.** Verify that the rules for the unit $\mathbf{1}$ are harmonious.

Task 8 (10 pts)**.** Verify that the rules for the unit $\top$ are harmonious.

5 Applications

In class, we looked *Blocks World*, an example of encoding *state* in linear logic; Blocks World is a class of scenarios in which there is a table, some number of blocks which can be stacked on top of each other, and a robotic arm which can pick up and move blocks. The following atomic predicates are used:

1. empty means that the robotic arm's hand is empty.
2. $\mathsf{clear}(x)$ means that the block x does not have anything on top of it.
3. $\mathsf{on}(x, y)$ means that the block x is directly on top of the block y.
4. $\mathsf{on\_table}(x)$ means that the block x is sitting directly on the table.

The possible state transitions for Blocks World are given by the following axioms in linear logic:

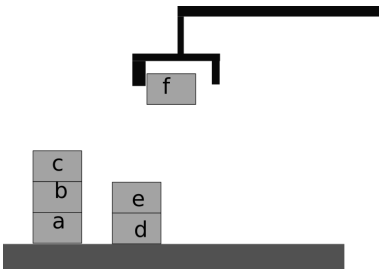
1. $\forall x, y.\ \mathsf{empty} \otimes \mathsf{clear}(x) \otimes \mathsf{on}(x, y) \multimap \mathsf{holds}(x) \otimes \mathsf{clear}(y)$
2. $\forall x.\mathsf{empty} \otimes \mathsf{clear}(x) \otimes \mathsf{on\_table}(x) \multimap \mathsf{holds}(x)$
3. $\forall x, y.\ \mathsf{holds}(x) \otimes \mathsf{clear}(y) \multimap \mathsf{empty} \otimes \mathsf{on}(x, y) \otimes \mathsf{clear}(x)$
4. $\forall x.\ \mathsf{holds}(x) \multimap \mathsf{empty} \otimes \mathsf{on\_table}(x) \otimes \mathsf{clear}(x)$

[2] Harmony is a necessary condition for the correctness of rules, but not a sufficient condition.

Task 9 (10 pts)**.** We have assumed that the table is infinitely broad and can therefore accomodate any number of blocks. **Consider the case that the table in fact only has a finite number of spaces for blocks on it, and modify the axioms of Blocks World above in order to preserve this invariant.**

You should use an atomic predicate space, which will mean that there is an available space on the table. Your solution to this task will not depend on how large the table is in the Blocks World configuration.

Task 10. Consider the following Blocks World scenario:



Write a formula in linear logic which expresses this configuration, assuming that the table can fit **four** blocks total directly on it.