

# Programming in Intuitionistic Linear Logic

Alwen Tiu

Australian National University

ANU Logic Summer School  
Lecture 4, December 14, 2007

# Weakness of hereditary Harrop formulas

- In both Horn and HH languages, if a sequent  $\Sigma : \mathcal{P} \longrightarrow G$  is provable, then for every sequent  $\Sigma' : \mathcal{P}' \longrightarrow G'$  in the proof, we have that

$$\Sigma \subseteq \Sigma' \text{ and } \mathcal{P} \subseteq \mathcal{P}'.$$

- The changes of the states of the interpreter is thus monotonic, it can't model retractions of assumptions.
- There are cases where we want a dynamic context:
  - ▶ Implementation of theorem provers: can we make use of meta-logic context to manage object-logic context?
  - ▶ Parsing relative clauses: gap threading problems.
  - ▶ Database: addition/retraction of facts.
  - ▶ Representing an object's state in context: how to manage state updates.

# Intuitionistic linear logic

- We consider a restriction of linear logic to intuitionistic sequents. Since we have single-conclusion sequents, no contractions allowed on the right, hence no  $?$ .
- We consider first the fragment without disjunctions (and its associated units) and existential.
- The set of allowed connectives:  $\top$ ,  $\mathbf{1}$ ,  $\otimes$ ,  $\&$ ,  $\multimap$ ,  $!$  and  $\forall$ .
- Problem: intuitionistic linear logic (*ILL*) doesn't have uniform provability. Some counterexamples:

$$\begin{aligned} \Sigma : a \otimes b &\longrightarrow b \otimes a & \Sigma : !a &\longrightarrow !a \otimes !a \\ \Sigma : !a \&b &\longrightarrow !a & \Sigma : b \otimes (b \multimap !a) &\longrightarrow !a & \Sigma : \mathbf{1} &\longrightarrow \mathbf{1} \end{aligned}$$

## Some rules

$$\frac{}{\Sigma : \Delta \longrightarrow \top} \top_R \quad \frac{\Sigma : \Delta \longrightarrow B}{\Sigma : \Delta, \mathbf{1} \longrightarrow B} \mathbf{1}_L \quad \frac{}{\Sigma : \longrightarrow \mathbf{1}} \mathbf{1}_R$$

$$\frac{\Sigma : \Delta, B_i \longrightarrow C}{\Sigma : \Delta, B_1 \& B_2 \longrightarrow C} \&_L \quad \frac{\Sigma : \Delta \longrightarrow B \quad \Sigma : \Delta \longrightarrow C}{\Sigma : \Delta \longrightarrow B \& C} \&_R$$

$$\frac{\Sigma : \Delta, B_1, B_2 \longrightarrow \Gamma}{\Sigma : \Delta, B_1 \otimes B_2 \longrightarrow \Gamma} \otimes_L \quad \frac{\Sigma : \Delta_1 \longrightarrow B \quad \Sigma : \Delta_2 \longrightarrow C}{\Sigma : \Delta_1, \Delta_2 \longrightarrow B \otimes C} \otimes_R$$

$$\frac{\Sigma : \Delta_1 \longrightarrow B \quad \Sigma : \Delta_2, C \longrightarrow D}{\Sigma : \Delta_1, \Delta_2, B \multimap C \longrightarrow D} \multimap_L \quad \frac{\Sigma : \Delta, B \longrightarrow C}{\Sigma : \Delta \longrightarrow B \multimap C} \multimap_R$$

$$\frac{\Sigma : \Delta \longrightarrow C}{\Sigma : \Delta, !B \longrightarrow C} !W \quad \frac{\Sigma : \Delta, !B, !B \longrightarrow C}{\Sigma : \Delta, !B \longrightarrow C} !C$$

$$\frac{\Sigma : \Delta, B \longrightarrow C}{\Sigma : \Delta, !B \longrightarrow C} !D \quad \frac{\Sigma : !\Delta \longrightarrow B}{\Sigma : !\Delta \longrightarrow !B} !R$$

# A design for linear logic programming language

- Uniform provability doesn't hold for *ILL* because  $\mathbf{1}_R$ ,  $\otimes_R$  and  $!R$  do not permute down under left-intro rules.
- We omit  $\otimes$  and allow  $!$  only on certain places in formulas. This is easily defined via a *derived connective*:

$$A \Rightarrow B := !A \multimap B.$$

This is actually an intuitionistic implication. The structures of sequents need to be slightly modified:

$$\Sigma : \Gamma; \Delta \longrightarrow C.$$

$\Gamma$  is the unbounded part,  $\Delta$  the bounded part. In ordinary sequent, this is interpreted as follows

$$\Sigma : !\Gamma, \Delta \longrightarrow C.$$

# The proof system $\mathcal{L}$

$$\frac{}{\Sigma : \Gamma; A \longrightarrow A} \textit{id} \quad \frac{\Sigma : \Gamma, B; \Delta, B \longrightarrow C}{\Sigma : \Gamma, B; \Delta \longrightarrow C} \textit{absorb} \quad \frac{}{\Sigma : \Gamma; \Delta \longrightarrow \top} \top_R$$

$$\frac{\Sigma : \Gamma; \Delta, B_i \longrightarrow C}{\Sigma : \Gamma; \Delta, B_1 \& B_2 \longrightarrow C} \&_L \quad \frac{\Sigma : \Gamma; \Delta \longrightarrow B \quad \Sigma : \Gamma; \Delta \longrightarrow C}{\Sigma : \Gamma; \Delta \longrightarrow B \& C} \&_R$$

$$\frac{\Sigma : \Gamma; \Delta_1 \longrightarrow B \quad \Sigma : \Gamma; \Delta_2, C \longrightarrow E}{\Sigma : \Gamma; \Delta_1, \Delta_2, B \multimap C \longrightarrow E} \multimap_L \quad \frac{\Sigma : \Gamma; \Delta, B \longrightarrow C}{\Sigma : \Gamma; \Delta \longrightarrow B \multimap C} \multimap_R$$

$$\frac{\Sigma : \Gamma; \emptyset \longrightarrow B \quad \Sigma : \Gamma; \Delta, C \longrightarrow E}{\Sigma : \Gamma; \Delta, B \Rightarrow C \longrightarrow E} \Rightarrow_L \quad \frac{\Sigma : \Gamma, B; \Delta \longrightarrow C}{\Sigma : \Gamma; \Delta \longrightarrow B \Rightarrow C} \Rightarrow_R$$

# An abstract logic programming language

## Theorem

*If  $\Sigma : \Delta; C \longrightarrow$  has a proof in  $\mathcal{L}$  then it has a uniform proof in  $\mathcal{L}$ .*

Let  $\mathcal{F}$  be the set of all  $\mathcal{L}$  formulas. Then as a consequence of the above theorem,  $\langle \mathcal{F}, \mathcal{F}, \vdash_{\mathcal{L}} \rangle$  is an abstract logic programming language.

## Theorem

*The sequent  $\Sigma : \Gamma; \Delta \longrightarrow B$  is provable in  $\mathcal{L}$  if and only if  $\Sigma : !\Gamma, \Delta \longrightarrow B$  is provable in ILL.*

## Backchaining rules

Let's make the left-rules of  $\mathcal{L}$  more deterministic, by allowing application of left-rules only on specifically designated formulas.

$$\frac{\Sigma : \mathcal{P}, D; \Delta \xrightarrow{D} A}{\Sigma : \mathcal{P}, D; \Delta \longrightarrow A} \quad \frac{\Sigma : \mathcal{P}; \Delta \xrightarrow{D} A}{\Sigma : \mathcal{P}; \Delta, D \longrightarrow A} \quad \frac{}{\Sigma : \mathcal{P}; . \xrightarrow{A} A} id$$

$$\frac{\Sigma : \mathcal{P}; \Delta \xrightarrow{D_1} A}{\Sigma : \mathcal{P}; \Delta \xrightarrow{D_1 \& D_2} A} \&_L \quad \frac{\Sigma : \mathcal{P}; \Delta \xrightarrow{D_2} A}{\Sigma : \mathcal{P}; \Delta \xrightarrow{D_1 \& D_2} A} \&_L$$

$$\frac{\Sigma : \mathcal{P}; \Delta_1 \longrightarrow G \quad \Sigma : \mathcal{P}; \Delta_2 \xrightarrow{D} A}{\Sigma : \mathcal{P}; \Delta_1, \Delta_2 \xrightarrow{G \multimap D} A} \multimap_L$$

$$\frac{\Sigma : \mathcal{P}; . \longrightarrow G \quad \Sigma : \mathcal{P}; \Delta \xrightarrow{D} A}{\Sigma : \mathcal{P}; \Delta \xrightarrow{G \Rightarrow D} A} \Rightarrow_L \quad \frac{\Sigma \vdash t : \tau \quad \Sigma : \mathcal{P}; \Delta \xrightarrow{D[t/x]} A}{\Sigma : \mathcal{P}; \Delta \xrightarrow{\forall_{\tau} x. D} A} \forall_L$$

## Adding right introduction rules

- We can reintroduce the connectives  $\mathbf{1}$ ,  $\oplus$ ,  $\otimes$  and  $!$ , provided they are restricted to *positive occurrence*.
- Define the following class of formulas:

$$R := \top \mid A \mid R_1 \& R_2 \mid G \multimap A \mid G \Rightarrow A \mid \forall x.R$$

$$G := \top \mid A \mid G_1 \& G_2 \mid R \multimap G \mid R \Rightarrow G \mid G_1 \oplus G_2 \mid \mathbf{1} \mid G_1 \otimes G_2 \mid !G \mid \exists x.G$$

(Here  $A$  means atomic formula)

- $R$ -formulas are called *resource formulas*, and  $G$ -formulas are *goal formulas*.
- We call the proof system  $\mathcal{L}$  (with backchaining replacing left-rules) extended with right-rules for  $\mathbf{1}$ ,  $\oplus$ ,  $\otimes$ ,  $!$  system  $\mathcal{L}'$ .
- The triple  $\langle \mathcal{R}, \mathcal{G}, \vdash_{\mathcal{L}'} \rangle$  is an abstract logic programming language.

# Dynamics of bounded context

Resource formula can be added to the bounded context using linear implication.

$$\frac{\Sigma : \Gamma; \Delta, B \longrightarrow C}{\Sigma : \Gamma; \Delta \longrightarrow B \multimap C} \multimap_R$$

Resource formulas can be retracted using  $\otimes$ , e.g.,

$$\frac{\Sigma : \Gamma; B \longrightarrow B \quad \Sigma : \Gamma; \Delta \longrightarrow C}{\Sigma : \Gamma; B, \Delta \longrightarrow B \otimes C} \otimes_R$$

They can also be 'erased':

$$\frac{}{\Sigma : \Gamma; \Delta \longrightarrow \top} \top_R$$

and checked for emptiness:

$$\frac{\Sigma : \Gamma; . \longrightarrow B}{\Sigma : \Gamma; . \longrightarrow !B} !_R$$

# Embedding of hereditary Harrop formulas

$A^+ = A^- = A$ , when  $A$  is atomic

$$\top^+ = \mathbf{1} \quad \top^- = \top$$

$$(B \wedge C)^+ = B^+ \otimes C^+ \quad (B \wedge C)^- = B^- \& C^-$$

$$(B \supset C)^+ = B^- \Rightarrow C^+ \quad (B \supset C)^- = B^+ \multimap C^-$$

$$(\forall x.B)^+ = \forall x.(B^+) \quad (\forall x.B)^- = \forall x.(B^-)$$

## Theorem

$\Sigma : \Gamma \longrightarrow B$  has a proof in intuitionistic logic if and only if  $\Sigma : \Gamma^- ; . \longrightarrow B^+$  has a proof in  $\mathcal{L}'$ .

## A concrete syntax

Uniform proofs involving Horn clauses or hereditary Harrop formulas are essentially the same as  $\mathcal{L}'$ -proofs of their translations.

This suggests a concrete syntax for a linear logic programming language so that the interpretation of Prolog or  $\lambda$ Prolog programs remains unchanged when embedded into this new setting.

For example, Prolog's syntax  $A_0 :- A_1, \dots, A_n$  which traditionally denote  $(A_1 \wedge \dots \wedge A_n) \supset A_0$  under the translation becomes the linear formula

$$(A_1 \otimes \dots \otimes A_n) \multimap A_0.$$

The complete concrete syntax:

,	$\otimes$	:-	$\multimap$
true	<b>1</b>	=>	$\Rightarrow$
&	<b>&amp;</b>	bang	!
erase	$\top$	-o	$\multimap$

This new language is called Lolli and has been implemented [see Hodas's thesis].

## Example: Toggling a switch

Assume the state of a switch is stored in the bounded part of the proof context.  
`toggle G` is provable in a given context if  $G$  is provable when the switch is set to the opposite setting.

*toggle G :- on, (off -o G).*

*toggle G :- off, (on -o G).*

## Example: Permuting a list

```
load nil K :- unload K.  
load (X::L) K :- (item X -o load L K).  
unload nil.  
unload (X::L) :- item X, unload L.  
perm L K :- bang (load L K).
```

## Example: A data base query language

```
db :- write ‘‘Command: ‘‘, read Command, do Command.  
db :- write ‘‘Try again.’’, nl, db.  
do (enter Entry) :- entry Entry -o db.  
do (commit Entry) :- entry Entry => db.  
do (retract Entry) :- entry Entry, db.  
do (upd Old New) :- entry Old, (entry New -o db).  
do (check Q) :- (entry Q, erase, write Q,  
    write ‘‘ is an entry.’’, nl) & db.  
do (necessary Q) :- (bang (entry Q), erase,  
    write Q, write ‘‘ is a necessary entry’’, nl) & db.  
do quit :- erase.
```

## A sample session

Command: enter (enroll jane cs1).

Command: check (enroll jane X).

(enroll jane cs1) is an entry.

Command: upd (enroll jane cs1) (enroll jane cs2).

Command: check (enroll jane X).

(enroll jane cs2) is an entry.

Command: commit (student jane).

Command: enter (student bob).

Command: necessary (student X).

(student jane) is a necessary entry

Command: retract (student jane).

(student jane) is a necessary entry.

Command: necessary (student X).

(student jane) is a necessary entry.

## Example: 'reverse' revisited

Let  $F$  be the resource formula:

$\forall X, P, Q. \text{rv } (X :: P) Q :- \text{rv } P (X :: Q).$

Define 'reverse' as follows:

$\text{reverse } L K :-$

$\forall rv. F \Rightarrow rv \text{ nil } K -o rv L \text{ nil}.$

Prove that 'reverse' is symmetric, i.e., (reverse  $L K$ ) if and only if (reverse  $K L$ ).