APLL: A focusing-based automated prover for linear logic

Jui-Hsuan Wu

ENS Paris

December 18, 2018
1 Introduction
2 Proof search
3 Demonstration
4 Future work
5 References
Background
Some existing works:

- the focused inverse (forward) method by K. Chaudhuri
- llprover (http://bach.istc.kobe-u.ac.jp/llprover/) (without focusing, implemented in Prolog)
- another prover (https://gitlab.com/vcvpaiva/FLL-Prover) (backward method, implemented in Maude)
Overview of the prover
two different logical systems (LL and ILL)
two different proof methods (backward and forward)
proof certificates using Coq and the Yalla library (only available for the backward method)
\LaTeX\ code of proofs (only available for the backward method)
more details on https://github.com/wujuihsuan2016/LL_prover
Backward proof search
Our backward proof search consists of
- starting from the goal sequent,
- applying the rules backwards, and
- using the corresponding focused proof system

In LLF, there are two kinds of sequents: \( \vdash \Theta : \Gamma \uparrow L \) and \( \vdash \Theta : \Gamma \downarrow F \)

**Theorem 1** Proving \( \vdash \Delta \) in LL is equivalent to proving \( \vdash \cdots \downarrow \Delta \) in LLF.
Some rules:

\[
\begin{align*}
\Gamma \vdash \Theta & : \top \uparrow L, \top & \vdash \Theta : \top \downarrow \neg F & \vdash \Theta : \Gamma \uparrow L, F & \vdash \Theta : \Gamma \uparrow L, G \\
\vdash \Theta : \Gamma \downarrow F_1 & \vdash \Theta : \Gamma \downarrow F_1 \oplus F_2 & \vdash \Theta : \Gamma \downarrow F & \vdash \Theta : \Gamma' \downarrow G \\
\vdash \Theta : \Gamma \downarrow F & \vdash \Theta, F : \Gamma \uparrow & \vdash \Theta, F : \Gamma \uparrow & \\
\vdash \Theta : \Gamma, S \uparrow L & \vdash \Theta : \Gamma \uparrow L, S & \vdash \Theta : \Gamma \uparrow N & \vdash \Theta : \Gamma \downarrow N \\
\vdash \Theta : \Gamma \downarrow N &
\end{align*}
\]

(J is not a negative literal)

(S is not asynchronous)

(N is neither synchronous nor a positive atom)
Control of the $D_2$ rule

- The main factor that slows down the algorithm: the $D_2$ rule.
- Problem: the classic recursive depth-first design will choose the same formula every time we apply the $D_2$ rule.
- Solution: choose the formula in a round-robin style.
- Implementation: a list of candidates select_d2 and an integer max_d2.
In rules such as $\&$ or $\otimes$, there are two branches to prove.

whynot-height $wh(F)$ of a formula $F$:

$wh(1) = wh(0) = wh(\top) = wh(\bot) = wh(X) = wh(X^\bot) = 0$ where $X$ is an atom.

$wh(\?F) = 1 + wh(F)$, $wh(\!F) = wh(F)$, $wh(F \diamond G) = max(wh(F), wh(G))$, $\forall \diamond \in \{\otimes, \oplus, \&\}$

By choosing the branch with smaller whynot-height, we are likely to visit the branch that can be proved (or disproved) faster than the other one.
Forward proof search: the focused inverse method
Idea: keep a database (set) of intermediate sequents and generate new provable sequents and add them into the database.

Main issue:

- How to use the stored sequents to generate new sequents?
- How to minimize the number of "useless" sequents?
Definition (forward sequents): A forward sequent has one of the following forms:

- $\Theta ; [\Gamma]_0$ (strong)
- $\Theta ; [\Gamma]_1$ (weak)

Intuitively, weak sequents are used to deal with arbitrary linear zones.

Ex: the conclusion of the $\top R$ of ILLF can be expressed by $\Theta ; [\cdot]_1$ and even by $\cdot ; [\cdot]_1$
Definition (subsumption or ”weaker than”):
We define the subsumption relation between forward sequents as follows,

- \((\Theta ; [\Gamma]_0) \prec (\Theta' ; [\Gamma]_0)\) if \(\Theta \subseteq \Theta'\)
- \((\Theta ; [\Gamma]_1) \prec (\Theta' : [\Gamma']_w)\) if \(\Theta \subseteq \Theta'\) and \(\Gamma \subseteq \Gamma'\)
Definition (Additive composition):
Given two linear contexts with weakness flag, $[\Gamma]_w$ and $[\Gamma']_{w'}$, we define their additive composition as follows:

$$[\Gamma]_w + [\Gamma']_{w'} = \begin{cases} 
[\Gamma]_0 & \text{if } w = w' = 0 \text{ and } \Gamma = \Gamma' \\
[\Gamma]_0 & \text{if } w = 0, w' = 1 \text{ and } \Gamma' \subseteq \Gamma \\
[\Gamma']_0 & \text{if } w = 1, w' = 0 \text{ and } \Gamma \subseteq \Gamma' \\
[\Gamma \sqcup \Gamma']_1 & \text{if } w = w' = 1 
\end{cases}$$

where $\Gamma \sqcup \Gamma'$ denotes the least upper bound of $\Gamma$ and $\Gamma'$. Note that this function is partial.
Definition (Multiplicative composition):
Given two linear contexts with weakness flag, $[\Gamma]_w$ and $[\Gamma']_{w'}$, we define their multiplicative composition as follows:
$[\Gamma]_w \times [\Gamma']_{w'} = [\Gamma \cup \Gamma']_{w \lor w'}$
Here, $\Gamma \cup \Gamma'$ denotes the sum of the multisets $\Gamma$ and $\Gamma'$.

Definition (Insertion):
Given a linear context with weakness flag $[\Gamma]_w$ and a proposition $A$, we define the result context after insertion of $A$ as follows:
$[\Gamma]_w, A = [\Gamma, A]_w$
Derived rules of the forward calculus:

\[
\begin{align*}
  s_1 s_2 \cdots s_n & \Rightarrow \Theta ; D \quad (foc(P)[s_1 \cdot s_2 \cdots s_n] \Rightarrow \Theta ; D) \\
  \Theta ; D, P & \Rightarrow \Theta ; D, P \quad \text{foc} \\
  s_1 s_2 \cdots s_n & \Rightarrow \Theta ; D \quad (foc(A)[s_1 \cdot s_2 \cdots s_n] \Rightarrow \Theta ; D) \\
  \Theta, A ; D & \Rightarrow \Theta, A ; D \quad ?foc
\end{align*}
\]
Focal phase:

\[
\begin{align*}
\text{foc}(A)[\Sigma_1] &\hookrightarrow \Theta_1 ; D_1 \quad \text{foc}(B)[\Sigma_2] \hookrightarrow \Theta_2 ; D_2 \quad \otimes F \\
\text{foc}(A \otimes B)[\Sigma_1 \cdot \Sigma_2] &\hookrightarrow \Theta_1, \Theta_2 ; D_1 \times D_2
\end{align*}
\]

\[
\begin{align*}
\text{act}(\cdot ; \cdot ; A)[\Sigma] &\hookrightarrow \Theta ; [\cdot]_\wedge \quad \text{act}(\cdot ; \cdot ; L)[\Sigma] &\hookrightarrow s \quad \text{FA} \quad \text{where } L \text{ is asynchronous}
\end{align*}
\]

\[
\begin{align*}
\text{foc}(A_i)[\Sigma] &\hookrightarrow s \quad \oplus F_i \\
\text{foc}(A_1 \oplus A_2)[\Sigma] &\hookrightarrow s \\
\text{foc}(1)[\cdot] &\hookrightarrow \cdot ; [\cdot]_0 \\
\text{foc}(X)[\cdot] &\hookrightarrow \cdot ; [X^\perp]_0 \\
\text{foc}(!A)[\Sigma] &\hookrightarrow \Theta ; [\cdot]_{\wedge} \\
\end{align*}
\]

\[
\begin{align*}
\text{act(} \cdot ; \cdot ; A)[\Sigma] &\hookrightarrow \Theta ; [\cdot]_w \quad \text{act(} \cdot ; \cdot ; L)[\Sigma] &\hookrightarrow s
\end{align*}
\]
Active phase:

\[
\frac{\text{act}(\Theta ; \Gamma ; L, A)[\Sigma_1] \leftrightarrow \Theta_1 ; D_1}{\frac{\text{act}(\Theta ; \Gamma ; L, B)[\Sigma_2] \leftrightarrow \Theta_2 ; D_2}{\text{act}(\Theta ; \Gamma ; L, A & B)[\Sigma_1 \cdot \Sigma_2] \leftrightarrow \Theta_1, \Theta_2 ; D_1 + D_2}} \quad \& \ A
\]

\[
\frac{\text{act}(\Theta ; \Gamma ; L, A, B)[\Sigma] \leftrightarrow s}{\text{act}(\Theta ; \Gamma ; L, A \lozenge B)[\Sigma] \leftrightarrow s} \quad \not\exists \ A
\]

\[
\frac{\text{act}(\Theta ; \Gamma ; L, A_i)[\Sigma] \leftrightarrow \Theta' ; [\Gamma']_1}{\text{act}(\Theta ; \Gamma ; L, A_1 \lozenge A_2)[\Sigma] \leftrightarrow \Theta' ; [\Gamma']_1} \quad \not\exists \ A_i
\]

\[
\frac{\text{act}(\Theta \cup \{A\} ; \Gamma ; L)[\Sigma] \leftrightarrow s}{\text{act}(\Theta ; \Gamma ; L, ?A)[\Sigma] \leftrightarrow s} \quad \not? \ A
\]
Active phase:

\[
\begin{align*}
act(\Theta ; \Gamma ; L)[\Sigma] & \rightarrow s \\
act(\Theta ; \Gamma ; L, \bot)[\Sigma] & \rightarrow s \\
act(\Theta ; \Gamma, R ; L)[\Sigma] & \rightarrow s \\
act(\Theta ; \Gamma ; L, R)[\Sigma] & \rightarrow s \\
act(\Theta ; \Gamma ; \cdot)[\Theta, \Theta' ; \Gamma, \Gamma'] & \rightarrow \Theta' ; \Gamma'
\end{align*}
\]
Theorem (completeness):
If $\vdash \Theta : \Gamma \overset{\cdot}{\uparrow}$ is provable, then either
- $\Theta' ; [\Gamma]_0$ is provable for some $\Theta' \subseteq \Theta$, or
- $\Theta' ; [\Gamma']_1$ is provable for some $\Theta' \subseteq \Theta$ and $\Gamma' \subseteq \Gamma$.

Theorem (soundness):
If $\Theta ; [\Gamma]_0$ is provable, then $\vdash \Theta : \Gamma \overset{\cdot}{\uparrow}$ is provable.
If $\Theta ; [\Gamma]_1$ is provable, then $\vdash \Theta : \Gamma' \overset{\cdot}{\uparrow}$ is provable for every $\Gamma' \supseteq \Gamma$. 
The first issue in the implementation is to enumerate the propositions for which we need to derive inference rules. We first define two functions \( a \) (active) and \( f \) (focal) inductively:

\[
\begin{align*}
    f(X) &= X \\
    f(X^-) &= \emptyset \\
    f(!A) &= a(A) \\
    f(?A) &= a(?A) \\
    f(A \otimes B) &= f(A \oplus B) = f(A) \cup f(B) \\
    f(A \& B) &= f(A \Leftarrow B) = a(A \& B) = a(A \Leftarrow B) \\
    f(1) &= f(0) = f(\top) = f(\bot) = \emptyset \\
    a(X) &= \{X\} \\
    a(X^-) &= \{X^-\} \\
    a(!A) &= !A \cup f(A) \\
    a(?A) &= \{A?\} \cup f(A) \\
    a(A \otimes B) &= \{A \otimes B\} \cup f(A \otimes B) \\
    a(A \oplus B) &= \{A \oplus B\} \cup f(A \oplus B) \\
    a(A \& B) &= a(A \Leftarrow B) = a(A) \cup a(B) \\
    a(1) &= a(0) = a(\top) = a(\bot) = \emptyset
\end{align*}
\]
Definition (frontier): Given a goal sequent $\Theta ; \Gamma \uparrow \cdot$, its frontier contains:
- all (top-level) propositions in $\Theta$ and $\Gamma$
- $f(A)$ for all $A$ in $\Theta$ and $\Gamma$.

Theorem:
In any backward focused proof, all sequents of the form $\Theta ; \Gamma \uparrow \cdot$ consists only of frontier propositions of the goal sequent.
Our implementation is quite simple: For every frontier proposition, we consider all possible (multi-)lists we can form by using the sequents in the database and apply one of the derived rules (foc or ?foc). Note that we should set a bound on the number of copies of the same sequent we can have in these (multi-)lists. Hence, the completeness is not preserved but the soundness is.
An implementation technique about the unrestricted context (the \( ?\text{foc} \) rule): if the proposition \( A_? \) considered occurs in the unrestricted zone of the goal sequent \( \Theta_0 \); \( \Gamma_0 \), then it need not to be added into the unrestricted zone in the conclusion.

In the intuitionistic case, there is a \( \text{foc}^+ \) rule of the following form:

\[
\frac{s_1 \cdot s_2 \cdots s_n \ (\text{foc}^+(Q)[s_1 \cdot s_2 \cdots s_n] \hookrightarrow \Theta \ ; \ D \rightarrow \cdot ) \quad \text{foc}^+}{\Theta \ ; \ D \rightarrow Q}
\]

If every occurrence of \( Q \) in the goal sequent is of the form \( !^k Q \), then we can directly add \( !^k Q \) into the conclusion.
Tests and Results
Most of the tests are gathered from https://github.com/carlosolarte/Benchmarking-Linear-Logic/tree/master/TPTP which are obtained by translation from intuitionistic logic to linear logic.

In general, the inverse (forward) method works faster than the backward method. In some cases, the former works worse because of the redundancy of the database of sequents.
A web interface for the prover under construction
### Introduction

In this paper, we present APLL, an automated prover for linear logic. APLL is implemented in the Frama-C platform, which provides a framework for building tools for software verification.

### Proof search

APLL employs a focusing-based strategy to search for proofs in linear logic. Focusing allows for a more efficient search by breaking down the proof into smaller, more manageable steps.

### Demonstration

To demonstrate APLL's capabilities, we have developed a web interface that allows users to input logical formulas and attempt to prove them. The interface uses the WebAssembly technology to achieve real-time performance.

### Future work

Future work includes enhancing APLL's capabilities to handle more complex logical formulas and improving its integration with other verification tools.

### References


---

#### Input

<table>
<thead>
<tr>
<th>Example</th>
<th>Sequent</th>
</tr>
</thead>
<tbody>
<tr>
<td>!((!A → B) → !((!B → 0) → !(A → 0)))</td>
<td>!(A → 0) → !(B → 0) → !(A → 0)</td>
</tr>
</tbody>
</table>

#### Output

<table>
<thead>
<tr>
<th>Result</th>
<th>Provable</th>
<th>Not provable</th>
<th>Proof not found</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Your proof will be displayed here</td>
</tr>
</tbody>
</table>

Download Latex
Jui-Hsuan Wu

APLL: A focusing-based automated prover for linear logic
Jui-Hsuan Wu

APLL: A focusing-based automated prover for linear logic

APLL is developed in the LLIplIO project.
Demonstration
Future work
• Proof extraction (Coq export and Latex export) for the forward method.
• Extend benchmark tests and give more detailed analysis of the execution times.
• Define new Coq tactics in order to reduce the compile time.
• Use additional criteria to accelerate the unprovable cases.
References