Linear Logic の Theorem Prover の使い方について (llprover)

神戸大学 工学部 情報知能工学科
田村直之 (tamura@kobe-u.ac.jp)
http://bach.seg.kobe-u.ac.jp/llprover/

1998 年 5 月 29 日

1 はじめに

このプログラムは，与えられたシーケントに対するカットなしの証明図を探します (ただし公理が設定されているときはカットも用います)．すなわち，適用可能な規則にしたがって，証明図を下から上へ作っていきます．利用している体系 CLL-X は，換の規則を陽に含んでいません (付録 A 参照)．したがって，表示される証明図中に換の規則は使われていません．

また，カットやコンラクションの使用回数が，定められた閾値以内の証明だけを探します．したがって，このプログラムで証明に失敗したという意味は，コントラクションおよびカットの使用回数が閾値以内の証明が存在しないということで，証明不可能ということではありません．

1.1beta 以前の版では変数条件の処理に誤りがありました．この版では直っているはずです…

2 起動方法

プログラム (llprover.pl) は，SICStus Prolog で書かれています．以下のようにして利用して下さい．

```
% prolog
SICStus 2.1 #3: Thu May 20 21:08:04 JST 1993
| ?- compile(llprover).
{compiling llprover.pl...}
{llprover.pl compiled, 10330 msec 120016 bytes}

yes
| ?- ll.
Linear Logic Prover ver 1.1 for SICStus Prolog
       by Naoyuki Tamura (tamura@kobe-u.ac.jp)
cll(full)> a->b->c --> b->a->c.
Trying to prove with threshold = 0
Succeed in proving a->b->c --> b->a->c (119 msec.)
twosided:pretty:1 =
            ------- Ax  ------- Ax
            b --> b     c --> c
------- Ax  ------------------- L->
a --> a        b,b->c --> c
--------------------------- L->
     a,b,a->b->c --> c
     ------------------ R->
```

```
     b,a->b->c --> a->c
     ------------------- R->
     a->b->c --> b->a->c
yes
cll(full)> quit.
yes
Exit from Linear Logic Prover...
Total CPU time = 270 msec.

yes
| ?- halt.
```

入力の最後に，ピリオド (“.”) を付けるのを忘れないで下さい! (Prolog でのお約束)
C-c (コントロール C) を入力すると，実行を停止できます.

```
Prolog interruption (h for help)? a
{ Execution aborted }
| ?-
```

3 記法

構文は，SICStus Prolog に従います．ただし，以下のように演算子が再定義されています.

```
:- op(1200, xfy, & ).
:- op(1190, xfx, [ -->, <--> ]).
:- op(1000, fx, @ ).
:- op( 910, xfy, -> ).
:- op( 500, xfy, [ +, /\, \/ ]).
:- op( 400, xfy, * ).
:- op( 300, fy,  [ ~, !, ? ]).
```

数値の小さいほうが強く (狭く) 結合します．また，`xfx` は挿入演算子，`xfy` は右結合的な挿入演算子 (すなわち $A+B+C=A+(B+C)$)，`fy` は前置演算子を表します.
変数などは以下のように記述して下さい.

- 個体変数: Prolog の変数を用いる `X`, `Y`, `X0`, ... など
- 定数: Prolog の定数を用いる `a`, `b`, `1`, ... など
- 関数記号，述語記号: Prolog の定数記号を用いる `a`, `b`, `c`, ... など

論理結合子は以下のように書きます (Girard の記法については [1],[2] を参照，Troelstra の記法については [3] を参照してください).

ここでの記法	Troelstra の記法	Girard の記法
`~A`	$\sim A$	$A^{\perp}$
`A*B`	$A \star B$	$A \otimes B$
`A+B`	$A + B$	$A \wp B$ $(A \maltese B)$
`1`	$\mathbf{1}$	$\mathbf{1}$
`0`	$\mathbf{0}$	$\perp$
`A/\B`	$A \sqcap B$	$A \,\&\, B$
`A\/B`	$A \sqcup B$	$A \oplus B$
`top`	$\top$	$\top$
`bot`	$\perp$	$\mathbf{0}$
`A->B`	$A \multimap B$	$A \multimap B$
`all(X,A)`	$\forall x.A$	$\bigwedge x.A$
`exists(X,A)`	$\exists x.A$	$\bigvee x.A$
`!A`	$!A$	$!A$
`?A`	$?A$	$?A$

`0` と`bot` が Girard の記法とは逆になっているので特に注意してください.

シーケントは以下のように書きます.

ここでの記法	Troelstra の記法	
`A1,A2,...,Am --> B1,B2,...,Bn`	$A_1, A_2, \ldots, A_m \Rightarrow B_1, B_2, \ldots, B_n$	$(m, n > 0)$
`A1,A2,...,Am --> []`	$A_1, A_2, \ldots, A_m \Rightarrow$	$(m > 0)$
`[] --> B1,B2,...,Bn`	$\Rightarrow B_1, B_2, \ldots, B_n$	$(n > 0)$
`[] --> []`	$\Rightarrow$	

以下，いくつか例を示します.

ここでの記法	Troelstra の記法
`1,a,b --> a*b,0`	$\mathbf{1}, A, B \Rightarrow A \star B, \mathbf{0}$
`a->b->c --> b->a->c`	$A \multimap (B \multimap C) \Rightarrow B \multimap (A \multimap C)$
`a->b/\c --> (a->b)/\(a->c)`	$A \multimap (B \sqcap C) \Rightarrow (A \multimap B) \sqcap (A \multimap C)$
`~ ~a --> a`	$\sim\sim A \Rightarrow A$
`!(a/\b) --> !a * !b`	$!(A \sqcap B) \Rightarrow !A \star !B$
`!all(X,a(X)) --> all(X,!a(X))`	$!\forall x.A(x) \Rightarrow \forall x.!A(x)$

4 体系の指定

以下のすべての体系を利用できます ([3] 参照)．厳密には CLL, ILZ, ILL はそれぞれ付録 A 中の CLL-X, ILZ-X, ILL-X のことです.

体系				ここでの表記			
CLL	$\mathbf{CLL}_{\mathrm{q}}$	$\mathbf{CLL}_{\mathrm{e}}$	$\mathbf{CLL}_0$	`cll(full)`	`cll(q)`	`cll(e)`	`cll(0)`
ILZ	$\mathbf{ILZ}_{\mathrm{q}}$	$\mathbf{ILZ}_{\mathrm{e}}$	$\mathbf{ILZ}_0$	`ilz(full)`	`ilz(q)`	`ilz(e)`	`ilz(0)`
ILL	$\mathbf{ILL}_{\mathrm{q}}$	$\mathbf{ILL}_{\mathrm{e}}$	$\mathbf{ILL}_0$	`ill(full)`	`ill(q)`	`ill(e)`	`ill(0)`

現在の体系は，プロンプトに表示されています．体系名を入力すれば，体系を変更することができます.

```
cll(full)> ill(0).
yes
ill(0)>
```

初期設定は，cll(full) です.

5 閾値の設定

この prover では，コントラクションおよびカットの使用回数の上限を定める必要があります (カットは公理に対してだけ用いられます)．すなわち，証明図の各枝に出てくる C!, C?, Cut の回数が 0 回の証明をまず探し，次に 1 回の証明を，次に 2 回の証明を…というように繰り返して，回数が閾値以下の証明だけを探します.

現在の閾値はthreshold(_) で調べることができます．また，threshold(n) により新しい閾値を設定できます．初期設定は 5 です (小さすぎると思うでしょうが，許して下さい…).

```
cll(full)> threshold(_).
threshold(5)
yes
cll(full)> threshold(10).
yes
cll(full)> threshold(_).
threshold(10)
yes
```

6 公理の設定

次のようにして公理を設定することができます.

cll(full)> axioms([公理$_1$, 公理$_2$, . . . , 公理$_n$]).

公理はすべて論理式です (シーケントは書けません)．また axiom schema を (Prolog の変数を用いて) 書くことはできません.

```
ill(0)> axioms([((A->0)->0)->A]).
Error ((_127->0)->0)->_127 is not a formula.
```

現在の公理はaxioms(_) で知ることができます．初期設定はaxioms([]) です.

公理が設定されていない場合は，カットなしの証明を探しますが，公理が設定されている場合はカットを使った証明も探します．したがって，命題論理の範囲でもかなり遅くなります.

```
cll(full)> ill(0).
yes
ill(0)> axioms(_).
axioms([])
yes
ill(0)> (a->0)->(b->0) --> b->a.
Trying to prove with threshold = 0 1 2 3 4 5
Fail to prove (a->0)->b->0 --> b->a (1170 msec.)
yes
ill(0)> axioms([ ((a->0)->0)->a ]).
yes
ill(0)> (a->0)->(b->0) --> b->a.
Trying to prove with threshold = 0 1
Succeed in proving (a->0)->b->0 --> b->a (7119 msec.)
twosided:pretty:2 =
                                        ------- Ax  ------- Ax
                                        b --> b     0 --> 0
```

```
                        ------------- Ax  ------------------- L->
                        a->0 --> a->0        b,b->0 --> 0
                        ---------------------------------- L->
                            a->0,b,(a->0)->b->0 --> 0
                            ---------------------------- R->  ------- Ax
                            b,(a->0)->b->0 --> (a->0)->0      a --> a
-------------------- Axiom  ---------------------------------------- L->
 --> ((a->0)->0)->a           ((a->0)->0)->a,b,(a->0)->b->0 --> a
-------------------------------------------------------------------- Cut
                       b,(a->0)->b->0 --> a
                       --------------------- R->
                       (a->0)->b->0 --> b->a
yes
```

7 スタイルの設定

以下のスタイルを選択できます.

命令	スタイル
`style(twosided)`	Two-sided calculus の指定です
`style(onesided)`	One-sided calculus の指定です
`style(proofnet)`	Proot Net PN1([1] 参照) の指定です

証明は two-sided で行い，出力の前に指定されたスタイルに変換します (付録 B 参照)．現在の出力形式は`style(_)` で調べることができます．初期設定は`style(twosided)` です.

`style(twosided)` の出力では，axiom link などに対する線は引きません (難しかったので)．かわりに，`Ax(1)` などのように対応する論理式に番号を付けます．また proof-box の表現も難しかったので，⊓ の規則は変換しません.

```
cll(full)> style(proofnet).
yes
cll(full)> a->b->c --> b->a->c.
Trying to prove with threshold = 0
Succeed in proving a->b->c --> b->a->c (199 msec.)
proofnet:pretty:3 =
       Ax(1)  Ax(3)       Ax(2)  Ax(3)
       ~a     c           b      ~c
Ax(2)  -------- +  Ax(1)  --------- *
~b       ~a+c      a        b* ~c
------------- +    -------------- *
  ~b+ ~a+c            a*b* ~c
yes
```

8 出力形式の設定

以下の出力形式を選択できます.

命令	出力形式
`output(pretty)`	これまでの例にあるような証明図の出力です
`output(tex)`	Makoto Tatsuta (`tatsuta@riec.tohoku.ac.jp`) さんによる`proof.sty` のための出力です
`output(standard)`	段下げによる出力です
`output(term)`	内部表現をそのまま出力します

現在の出力形式はoutput(_) で調べることができます．初期設定はoutput(pretty) です．

output(pretty) で証明図を出力するプログラムが，一番大変でした (自動証明を行うプログラムよりも!).

output(tex) は，杉山英二君 (eiji@grad306c.scitec.kobe-u.ac.jp) が作ってくれたプログラムを改良して用いています．個体変数は，x, y, z, x_1, y_1, z_1, ... に置き換えられます．述語記号は，最初の文字が大文字に置き換えられます．例えば，次のような出力が得られます．

```
cll(full)> output(tex).
yes
cll(full)> !all(X,a(X)) * !all(Y,a(Y)) --> all(Z,a(Z)).
Trying to prove with threshold = 0
Succeed in proving !all(_119,a(_119))*!all(_176,a(_176)) --> all(_523,a(_523)) (94 msec.)
twosided:tex:4 =
\begin{displaymath}
\infer[\LR{\tprod}]{! (\lall x.A(x)) \tprod ! (\lall y.A(y)) \drv \lall z.A(z)}{
  \infer[\RR{\lall}]{! (\lall x.A(x)), ! (\lall y.A(y)) \drv \lall z.A(z)}{
    \infer[\WR{!}]{! (\lall x.A(x)), ! (\lall y.A(y)) \drv A(z)}{
      \infer[\LR{!}]{! (\lall y.A(y)) \drv A(z)}{
        \infer[\LR{\lall}]{\lall y.A(y) \drv A(z)}{
          \infer[\Ax]{A(z) \drv A(z)}{}
        }
      }
    }
  }
}
\end{displaymath}
yes
```

これを，以下のようにしてLaTeX ファイル中に埋め込めば，美しい出力が得られます (Makoto Tatsuta さんによるproof.sty と，配布パッケージ中のlinear.sty が必要です).

```
\documentstyle[proof,linear]{article}
\begin{document}
\begin{displaymath}
  .....
\end{displaymath}
\end{document}
```

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A(z) \Rightarrow A(z)}\,\mathrm{Ax}}{\forall y.A(y) \Rightarrow A(z)}\,\mathrm{L}\forall}{!(\forall y.A(y)) \Rightarrow A(z)}\,\mathrm{L!}}{!(\forall x.A(x)), !(\forall y.A(y)) \Rightarrow A(z)}\,\mathrm{W!}}{!(\forall x.A(x)), !(\forall y.A(y)) \Rightarrow \forall z.A(z)}\,\mathrm{R}\forall}{!(\forall x.A(x)) \star !(\forall y.A(y)) \Rightarrow \forall z.A(z)}\,\mathrm{L}\star$$

\begin{displaymath}の前に\girardnotation を付けるか，あるいは前後を\begin{girardnotation}, \end{girardnotation}でくくると，Girard の記法で出力できます．Girard の記法の場合，シーケントの矢印として $\vdash$ が使われます.

9 ログ

log(yes) または log(' ファイル名') で，ファイルにログを記録できます．log(yes) の場合，ファイル名はllprover.log となります.

`log(no)` で，ログの記録を終了します．`log(_)` で現在のログファイル名を表示します．

ログ記録中，入力は必ずしも実際に入れたものと同じにはなりません (特に Prolog の変数を含んでいる場合)．たとえば，

```
cll(full)> !all(X,a(X)) --> all(X,!a(X)).
```

と入れても，ログには

```
cll(full)> !all(_290,a(_290))-->all(_290,!a(_290)).
```

などのように記録されます．

10 その他

- `quit` で終了します．
- `help` でコマンドの簡単な説明を表示します．
- `[`ファイル名`]` でファイルからの入力ができます．
- Γ`<-->`Δで，Γ`-->`ΔとΔ`-->`Γの両方を証明します (ΓとΔがそれぞれ一つの論理式でないとあまり意味がありません)．
- `init` で初期化をします (`cll(full)`, `threshold(5)`, `axioms([])`, `style(twosided)`, `output(pretty)`, `log(no)` および出力カウントの初期化を行う)．
- 正の整数 n で n 番目の出力を再表示します．$n < 0$ のときは，$|n|$ 個前の出力を再表示します．
- `&`で複数命令を記述できます．
- @G により，Prolog のゴール G を実行します．

入力ファイルのサンプルとして，`llproverex` を用意してあります．実行結果は `llproverex.log` です．

11 おわりに

まだまだ効率は悪いし，バグもあるかと思いますが，よろしければ使ってみてください．再配布，改変は自由です．

あと，できるとうれしいのは，

- `style(proofnet)` の出力中の link を描く．
- Cut elimination
- Natural deduction, Combinator への変換 (`style(natural)` など)

といったところでしょうか．

付録で，この Theorem Prover で使っている体系 CLL-X を説明します．

A CLL-X

まず，体系 CLLX を以下のように定義します．CLLX は換の規則を含んでいますが，あとでそれが不要であることを証明します．

- Logical axiom and Structural rules:

$$\frac{}{A \Rightarrow A}\ \mathrm{Ax}$$

$$\frac{\Gamma \Rightarrow \Delta_1, A, \Delta_2 \quad \Gamma_1', A, \Gamma_2' \Rightarrow \Delta'}{\Gamma_1', \Gamma, \Gamma_2' \Rightarrow \Delta_1, \Delta', \Delta_2}\ \mathrm{Cut}$$

$$\frac{\Gamma_1, B, A, \Gamma_2 \Rightarrow \Delta}{\Gamma_1, A, B, \Gamma_2 \Rightarrow \Delta}\ \mathrm{LX} \qquad \frac{\Gamma \Rightarrow \Delta_1, B, A, \Delta_2}{\Gamma \Rightarrow \Delta_1, A, B, \Delta_2}\ \mathrm{RX}$$

- Rules for the propositional connectives:

$$\frac{\Gamma_1, \Gamma_2 \Rightarrow A, \Delta}{\Gamma_1, \sim A, \Gamma_2 \Rightarrow \Delta}\ \mathrm{L}\sim \qquad \frac{A, \Gamma \Rightarrow \Delta_1, \Delta_2}{\Gamma \Rightarrow \Delta_1, \sim A, \Delta_2}\ \mathrm{R}\sim$$

$$\frac{\Gamma_1, A, \Gamma_2 \Rightarrow \Delta}{\Gamma_1, A \sqcap B, \Gamma_2 \Rightarrow \Delta}\ \mathrm{L}\sqcap_1$$

$$\frac{\Gamma_1, B, \Gamma_2 \Rightarrow \Delta}{\Gamma_1, A \sqcap B, \Gamma_2 \Rightarrow \Delta}\ \mathrm{L}\sqcap_2 \qquad \frac{\Gamma \Rightarrow \Delta_1, A, \Delta_2 \quad \Gamma \Rightarrow \Delta_1, B, \Delta_2}{\Gamma \Rightarrow \Delta_1, A \sqcap B, \Delta_2}\ \mathrm{R}\sqcap$$

$$\frac{\Gamma_1, A, B, \Gamma_2 \Rightarrow \Delta}{\Gamma_1, A \star B, \Gamma_2 \Rightarrow \Delta}\ \mathrm{L}\star \qquad \frac{\Gamma_1' \Rightarrow \Delta_1', A, \Delta_2' \quad \Gamma_1'' \Rightarrow \Delta_1'', B, \Delta_2''}{\Gamma_1 \Rightarrow \Delta_1, A \star B, \Delta_2}\ \mathrm{R}\star$$

$$\frac{\Gamma_1, A, \Gamma_2 \Rightarrow \Delta \quad \Gamma_1, B, \Gamma_2 \Rightarrow \Delta}{\Gamma_1, A \sqcup B, \Gamma_2 \Rightarrow \Delta}\ \mathrm{L}\sqcup \qquad \frac{\Gamma \Rightarrow \Delta_1, A, \Delta_2}{\Gamma \Rightarrow \Delta_1, A \sqcup B, \Delta_2}\ \mathrm{R}\sqcup_1$$

$$\frac{\Gamma \Rightarrow \Delta_1, B, \Delta_2}{\Gamma \Rightarrow \Delta_1, A \sqcup B, \Delta_2}\ \mathrm{R}\sqcup_2$$

$$\frac{\Gamma_1', A, \Gamma_2' \Rightarrow \Delta_1' \quad \Gamma_1'', B, \Gamma_2'' \Rightarrow \Delta_1''}{\Gamma_1, A + B, \Gamma_2 \Rightarrow \Delta_1}\ \mathrm{L}+ \qquad \frac{\Gamma \Rightarrow \Delta_1, A, B, \Delta_2}{\Gamma \Rightarrow \Delta_1, A + B, \Delta_2}\ \mathrm{R}+$$

$$\frac{\Gamma_1', \Gamma_2' \Rightarrow A, \Delta_1' \quad \Gamma_1'', B, \Gamma_2'' \Rightarrow \Delta_1''}{\Gamma_1, A \multimap B, \Gamma_2 \Rightarrow \Delta_1}\ \mathrm{L}\multimap \qquad \frac{A, \Gamma \Rightarrow \Delta_1, B, \Delta_2}{\Gamma \Rightarrow \Delta_1, A \multimap B, \Delta_2}\ \mathrm{R}\multimap$$

ただし，R$\star$，L+，L$\multimap$において，$\Gamma_i \in \mathrm{Merge}(\Gamma_i', \Gamma_i'')$，$\Delta_i \in \mathrm{Merge}(\Delta_i', \Delta_i'')$ である．ここで，$\mathrm{Merge}(\Gamma, \Delta)$ は以下のように定義される (列を要素とする) 集合である．

- Γが空列のとき，$\mathrm{Merge}(\Gamma, \Delta) = \{\Delta\}$.
- Δが空列のとき，$\mathrm{Merge}(\Gamma, \Delta) = \{\Gamma\}$.
- $\Gamma = A, \Gamma'$, $\Delta = B, \Delta'$のとき，
  $\mathrm{Merge}(\Gamma, \Delta) = \{A, X \mid X \in \mathrm{Merge}(\Gamma', \Delta)\} \cup \{B, Y \mid Y \in \mathrm{Merge}(\Gamma, \Delta')\}$.

- Rules for the propositional constants:

$$\frac{\Gamma_1, \Gamma_2 \ \Rightarrow \ \Delta}{\Gamma_1, \mathbf{1}, \Gamma_2 \ \Rightarrow \ \Delta}\ \mathrm{L}\mathbf{1} \qquad \frac{}{\Rightarrow \ \mathbf{1}}\ \mathrm{R}\mathbf{1}$$

$$\frac{}{\Gamma \ \Rightarrow \ \Delta_1, \top, \Delta_2}\ \mathrm{R}\top$$

$$\frac{}{\mathbf{0} \ \Rightarrow}\ \mathrm{L}\mathbf{0} \qquad \frac{\Gamma \ \Rightarrow \ \Delta_1, \Delta_2}{\Gamma \ \Rightarrow \ \Delta_1, \mathbf{0}, \Delta_2}\ \mathrm{R}\mathbf{0}$$

$$\frac{}{\Gamma_1, \bot, \Gamma_2 \ \Rightarrow \ \Delta}\ \mathrm{L}\bot$$

- Rules for the quantifiers:

$$\frac{\Gamma_1, A[x/t], \Gamma_2 \ \Rightarrow \ \Delta}{\Gamma_1, \forall x.A, \Gamma_2 \ \Rightarrow \ \Delta}\ \mathrm{L}\forall \qquad \frac{\Gamma \ \Rightarrow \ \Delta_1, A[x/y], \Delta_2}{\Gamma \ \Rightarrow \ \Delta_1, \forall x.A, \Delta_2}\ \mathrm{R}\forall$$

$$\frac{\Gamma_1, A[x/y], \Gamma_2 \ \Rightarrow \ \Delta}{\Gamma_1, \exists x.A, \Gamma_2 \ \Rightarrow \ \Delta}\ \mathrm{L}\exists \qquad \frac{\Gamma \ \Rightarrow \ \Delta_1, A[x/t], \Delta_2}{\Gamma \ \Rightarrow \ \Delta_1, \exists x.A, \Delta_2}\ \mathrm{R}\exists$$

ただし，R∀, L∃ において，yは下シーケント中で free ではない.

- Rules for the exponentials:

$$\frac{\Gamma_1, !A, !A, \Gamma_2 \ \Rightarrow \ \Delta}{\Gamma_1, !A, \Gamma_2 \ \Rightarrow \ \Delta}\ \mathrm{C}! \qquad \frac{\Gamma_1, \Gamma_2 \ \Rightarrow \ \Delta}{\Gamma_1, !A, \Gamma_2 \ \Rightarrow \ \Delta}\ \mathrm{W}!$$

$$\frac{\Gamma_1, A, \Gamma_2 \ \Rightarrow \ \Delta}{\Gamma_1, !A, \Gamma_2 \ \Rightarrow \ \Delta}\ \mathrm{L}! \qquad \frac{!\Gamma \ \Rightarrow ?\Delta_1, A, ?\Delta_2}{!\Gamma \ \Rightarrow ?\Delta_1, !A, ?\Delta_2}\ \mathrm{R}!$$

$$\frac{\Gamma \ \Rightarrow \ \Delta_1, A, \Delta_2}{\Gamma \ \Rightarrow \ \Delta_1, ?A, \Delta_2}\ \mathrm{W}? \qquad \frac{\Gamma \ \Rightarrow \ \Delta_1, ?A, ?A, \Delta_2}{\Gamma \ \Rightarrow \ \Delta_1, ?A, \Delta_2}\ \mathrm{C}?$$

$$\frac{!\Gamma_1, A, !\Gamma_2 \ \Rightarrow ?\Delta}{!\Gamma_1, ?A, !\Gamma_2 \ \Rightarrow ?\Delta}\ \mathrm{L}? \qquad \frac{\Gamma \ \Rightarrow \ \Delta_1, A, \Delta_2}{\Gamma \ \Rightarrow \ \Delta_1, ?A, \Delta_2}\ \mathrm{R}?$$

質問: Merge を，列の連結 (concatenation) とすれば，Exchange なしの体系になるでしょうか?

Proposition 1　CLLX は CLL と同値である.

(証明)　一方の規則は，他方の規則と LX, RX を用いれば表現できることから，明らか.

Proposition 2　CLLX で Cut の規則は不要である.

(証明)　CLL で Cut のない証明は，そのまま CLLX での Cut のない証明になる.

Proposition 3　CLLX で LX, RX の規則は不要である.

(証明) CLLX で$\Gamma \Rightarrow \Delta$が証明可能のとき，$\Gamma \Rightarrow \Delta$の任意の順列$\Sigma \Rightarrow \Phi$が CLLX で LX, RX なしに証明可能であることを，証明図の高さについての帰納法で示す．ただし，シーケント$\Sigma \Rightarrow \Phi$が$\Gamma \Rightarrow \Delta$の順列であるとは，$\Sigma$が$\Gamma$の順列であり，$\Phi$が$\Delta$の順列であることを意味するものとする．

- 高さ 1 の場合 (Ax, R$\mathbf{1}$, R$\top$, L$\mathbf{0}$, L$\bot$)
  Ax, R$\mathbf{1}$, L$\mathbf{0}$ については，異なる順列は生じない．

  R$\top$, L$\bot$の場合，どのような順列もそのまま LX, RX を用いずに R$\top$, L$\bot$の規則で証明できる．

- 最後の規則が LX, RX の場合
  帰納法の仮定により，上のシーケントの任意の順列が LX, RX なしで証明できるから，LX, RX を行ったシーケントも LX, RX なしで証明できる．

- 最後の規則が L$\sim$の場合
  $$\frac{\Gamma_1, \Gamma_2 \Rightarrow A, \Delta}{\Gamma_1, {\sim}A, \Gamma_2 \Rightarrow \Delta}\ \mathrm{L}\sim$$

  $\Gamma_1, {\sim}A, \Gamma_2 \Rightarrow \Delta$の任意の順列$\Sigma, {\sim}A, \Phi \Rightarrow \Psi$が証明可能であることを示す．

  Σ, ΦはΓ_1, Γ_2の順列，A, ΨはA, Δの順列であるから，帰納法の仮定より$\Sigma, \Phi \Rightarrow A, \Psi$は LX, RX なしで証明可能である．これに L$\sim$を適用すれば，$\Sigma, {\sim}A, \Phi \Rightarrow \Psi$は LX, RX なしで証明可能である．

- 最後の規則が L$\sqcap_1$の場合
  $$\frac{\Gamma_1, A, \Gamma_2 \Rightarrow \Delta}{\Gamma_1, A \sqcap B, \Gamma_2 \Rightarrow \Delta}\ \mathrm{L}\sqcap_1$$

  $\Gamma_1, A \sqcap B, \Gamma_2 \Rightarrow \Delta$の任意の順列$\Sigma, A \sqcap B, \Phi \Rightarrow \Psi$が証明可能であることを示す．

  Σ, A, ΦはΓ_1, A, Γ_2の，ΨはΔの順列であるから，帰納法の仮定より$\Sigma, A, \Phi \Rightarrow \Psi$は LX, RX なしで証明可能である．これに L$\sqcap_1$を適用すれば，$\Sigma, A \sqcap B, \Phi \Rightarrow \Psi$は LX, RX なしで証明可能である．

- 最後の規則が R$\sim$, L$\sqcap_2$, R$\sqcap$, L$\star$, L$\sqcup$, R$\sqcup_1$, R$\sqcup_2$, R$+$, R$\multimap$, L$\mathbf{1}$, R$\mathbf{0}$, Rules for the quantifiers, Rules for the exponentials の場合
  L$\sim$, L$\sqcap_1$の場合と同様である．

- 最後の規則が R$\star$の場合
  $$\frac{\Gamma_1' \Rightarrow \Delta_1', A, \Delta_2' \quad \Gamma_1'' \Rightarrow \Delta_1'', B, \Delta_2''}{\Gamma_1 \Rightarrow \Delta_1, A \star B, \Delta_2}\ \mathrm{R}\star$$

  $\Gamma_1 \Rightarrow \Delta_1, A \star B, \Delta_2$の任意の順列$\Sigma \Rightarrow \Phi, A \star B, \Psi$が証明可能であることを示す．

  Σ中のΓ_1'の要素を順に抜き出したものをΣ'，Γ_1''の要素を順に抜き出したものをΣ''とすると，$\Sigma \in \mathrm{Merge}(\Sigma', \Sigma'')$である．また，$\Phi$中の$\Delta_1'$または$\Delta_2'$の要素を順に抜き出したものを$\Phi'$，$\Delta_1''$または$\Delta_2''$の要素を順に抜き出したものを$\Phi''$とすると，$\Phi \in \mathrm{Merge}(\Phi', \Phi'')$である．同様に，$\Psi$中の$\Delta_1'$または$\Delta_2'$の要素を順に抜き出したものを$\Psi'$，$\Delta_1''$または$\Delta_2''$の要素を順に抜き出したものを$\Psi''$とすると，$\Psi \in \mathrm{Merge}(\Psi', \Psi'')$である．

  $\Sigma' \Rightarrow \Phi', A, \Psi'$は$\Gamma_1' \Rightarrow \Delta_1', A, \Delta_2'$の順列，$\Sigma'' \Rightarrow \Phi'', B, \Psi''$は$\Gamma_1'' \Rightarrow \Delta_1'', B, \Delta_2''$の順列であるから，帰納法の仮定により，$\Sigma' \Rightarrow \Phi', A, \Psi'$と$\Sigma'' \Rightarrow \Phi'', B, \Psi''$は LX, RX なしで証明可能である．これに R$\star$を適用すれば，$\Sigma \Rightarrow \Phi, A \star B, \Psi$は LX, RX なしで証明可能である．

- 最後の規則が L+, L $\multimap$ の場合
  R$\star$の場合と同様である.

CLLX で LX, RX が不要であることがわかったので，CLLX から LX, RX を取り除いた体系を CLL-X と定義します．この Theorem Prover で使っている体系は，この CLL-X です.
CLL-X について以下のことがわかります.

Proposition 4 CLL-X に $\Rightarrow A$ という形 (A は formula) の公理が付け加わっているとき，カットの規則を以下のような形に制限できる．また，公理がないときはカットの規則は不要である.

$$\dfrac{\dfrac{}{\Rightarrow A}\ \text{Axiom} \qquad A, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}\ \text{Cut}$$

(証明) カット除去定理の証明より明らか.

Proposition 5 CLL-X で C!の上の規則は R$\star$, L+, L $\multimap$, C!, C?, L!だけに制限できる．また，C?の上の規則は R$\star$, L+, L $\multimap$, C!, C?, R?だけに制限できる.

(証明) C!, C?の規則は以下のようになっている.

$$\dfrac{\Gamma_1, !A, !A, \Gamma_2 \Rightarrow \Delta}{\Gamma_1, !A, \Gamma_2 \Rightarrow \Delta}\ \text{C!} \qquad \dfrac{\Gamma \Rightarrow \Delta_1, ?A, ?A, \Delta_2}{\Gamma \Rightarrow \Delta_1, ?A, \Delta_2}\ \text{C?}$$

C!についてのみ証明する．C?についても同様にして証明できる.
C!の上に現れうる規則 R について場合分けする.

(1) R が Ax, Cut, R**1**, L**0** の場合．これらは C!の上の規則にはなりえない.

(2) R が R$\top$, L$\bot$の場合．C!の規則自体が不要である.

(3) R が Rules for the propositional connectives のうち，R$\star$, L+, L $\multimap$以外の場合.

$$\dfrac{\dfrac{\Gamma_1', !A, !A, \Gamma_2' \Rightarrow \Delta'}{\Gamma_1, !A, !A, \Gamma_2 \Rightarrow \Delta}\ \text{R}}{\Gamma_1, !A, \Gamma_2 \Rightarrow \Delta}\ \text{C!} \quad \text{または} \quad \dfrac{\Gamma_1', !A, !A, \Gamma_2' \Rightarrow \Delta' \quad \Gamma_1'', !A, !A, \Gamma_2'' \Rightarrow \Delta''}{\dfrac{\Gamma_1, !A, !A, \Gamma_2 \Rightarrow \Delta}{\Gamma_1, !A, \Gamma_2 \Rightarrow \Delta}\ \text{C!}}\ \text{R}$$

$!A$ は R の主論理式ではありえないから，R の上のシーケントにも $!A$ はそのまま現れている．したがって，以下のように変換すればよい.

$$\dfrac{\dfrac{\Gamma_1', !A, !A, \Gamma_2' \Rightarrow \Delta'}{\dfrac{\Gamma_1', !A, \Gamma_2' \Rightarrow \Delta'}{\Gamma_1, !A, \Gamma_2 \Rightarrow \Delta}\ \text{R}}\ \text{C!}}{\Gamma_1, !A, \Gamma_2 \Rightarrow \Delta}\ \text{R} \quad \text{または} \quad \dfrac{\dfrac{\Gamma_1', !A, !A, \Gamma_2' \Rightarrow \Delta'}{\Gamma_1', !A, \Gamma_2' \Rightarrow \Delta'}\ \text{C!} \quad \dfrac{\Gamma_1'', !A, !A, \Gamma_2'' \Rightarrow \Delta''}{\Gamma_1'', !A, \Gamma_2'' \Rightarrow \Delta''}\ \text{C!}}{\Gamma_1, !A, \Gamma_2 \Rightarrow \Delta}\ \text{R}$$

(4) R が Rules for the quantifiers の場合．(3) と同様である.

(5) R が R!, W?, L?, R?の場合．(3) と同様である.

(6) R が W!の場合.

(6a) $!A$ が W! の主論理式の場合.

$$\dfrac{\dfrac{\Gamma_1, !A, \Gamma_2 \ \Rightarrow\ \Delta}{\Gamma_1, !A, !A, \Gamma_2 \ \Rightarrow\ \Delta}\ \text{W!}}{\Gamma_1, !A, \Gamma_2 \ \Rightarrow\ \Delta}\ \text{C!}$$

これは，まとめて除去できる.

(6b) $!A$ が W! の主論理式でない場合.

$$\dfrac{\dfrac{\Gamma_1', !A, !A, \Gamma_2' \ \Rightarrow\ \Delta}{\Gamma_1, !A, !A, \Gamma_2 \ \Rightarrow\ \Delta}\ \text{W!}}{\Gamma_1, !A, \Gamma_2 \ \Rightarrow\ \Delta}\ \text{C!}$$

これは次のように変換できる.

$$\dfrac{\dfrac{\Gamma_1', !A, !A, \Gamma_2' \ \Rightarrow\ \Delta}{\Gamma_1', !A, \Gamma_2' \ \Rightarrow\ \Delta}\ \text{C!}}{\Gamma_1, !A, \Gamma_2 \ \Rightarrow\ \Delta}\ \text{W!}$$

よく考えれば，以下のようなさらに細かい制限を付けることができますが，現在のシステムではそこまで行っていません.

- $!A$ や $?A$ は，multiplicative な規則で別々のシーケントに別れなければならない.
- C! の上の L! は，以下のような場合だけを考えればよい．C? の上の R? についても同様.

$$\dfrac{\dfrac{\Gamma_1, A, !A, \Gamma_2 \ \Rightarrow\ \Delta}{\Gamma_1, !A, !A, \Gamma_2 \ \Rightarrow\ \Delta}\ \text{L!}}{\Gamma_1, !A, \Gamma_2 \ \Rightarrow\ \Delta}\ \text{C!}$$

B 他のスタイルへの変換

B.1 One-sided への変換

CLL-X の各シーケント $\Gamma \ \Rightarrow\ \Delta$ を $\sim\Gamma, \Delta$ に変換します．変換した結果の証明図中では，換の規則が明示されずに使われていることがあります

B.2 Proof Net への変換

説明を作成中です．とりあえずはプログラムを参照して下さい (^_^;)

C 証明アルゴリズムの概要

C.1 内部データ構造

シーケントの内部データ構造は以下の通りです.

〈 シーケント 〉 := 〈 論理式列 〉 --> 〈 論理式列 〉

〈 論理式列 〉 := [〈 論理式 〉 , ... 〈 論理式 〉]

証明の内部データ構造は以下の通りです.

〈証明〉の定義の右辺にある〈シーケント〉は下シーケントに，〈証明〉は上シーケントの証明に対応します．また，〈位置〉r(n) (または l(n)) は，シーケントの右辺 (または左辺) の左から n 番目の位置 ($n = 0$ のとき左端) を表します．

〈証明〉 := [〈規則名〉,〈論理式位置〉,〈シーケント〉,〈証明〉,...〈証明〉]
〈規則名〉 := ax | axiom | cut | l(~) | r(~) | l(/\,1) | l(/\,2) | r(/\) | l(*) | r(*) |
l(\/) | r(\/,1) | r(\/,2) | l(+) | r(+) | l(->) | r(->) |
l(1) | r(1) | r(top) | l(0) | r(0) | l(bot) |
l(all) | r(all) | r(exists) | l(exists) |
c(!) | w(!) | l(!) | r(!) | w(?) | c(?) | l(?) | r(?)
〈論理式位置〉 := [〈主論理式位置〉,〈副論理式位置〉,...〈副論理式位置〉]
〈主論理式位置〉 := [] | 〈位置〉
〈副論理式位置〉 := [] | 〈位置〉 | [〈位置〉,〈位置〉]
〈位置〉 := r(〈自然数〉) | l(〈自然数〉)

たとえば，次の証明

```
            ------- Ax  ------- Ax
            b --> b     c --> c
------- Ax  ------------------- L->
a --> a        b,b->c --> c
---------------------------- L->
     a,b,a->b->c --> c
     ------------------ R->
     b,a->b->c --> a->c
     ------------------- R->
     a->b->c --> b->a->c
```

の内部データ構造は以下のようになっています．

```
[r(->), [r(0),[l(0),r(0)]], ([a->b->c]-->[b->a->c]),
  [r(->), [r(0),[l(0),r(0)]], ([b,a->b->c]-->[a->c]),
    [l(->), [l(2),r(0),l(1)], ([a,b,a->b->c]-->[c]),
      [ax, [[]], ([a]-->[a])],
      [l(->), [l(1),r(0),l(0)], ([b,b->c]-->[c]),
        [ax, [[]], ([b]-->[b])],
        [ax, [[]], ([c]-->[c])]
      ]
    ]
  ]
]
```

C.2 アルゴリズムの概要

- 述語 prove($+S$, $-P$)

述語 prove(S,P) は，与えられたシーケント Sに対して，カットおよびコントラクションの回数が閾値以下の証明 Pを探す．一つでも証明が求まれば他の証明は探さない．

(1) 内部データベース proved/3, not_proved/2 を消去．

(2) Nを 0 から閾値まで 1 ずつ増やしながら，述語 prove(S,P,N) を呼び出す．

- 述語 prove($+S$, $-P$, $+N$)
  述語 prove(S,P,N) は，与えられたシーケント Sに対して，カットおよびコントラクションの回数が N以下の証明 Pを探す．一つでも証明が求まれば他の証明は探さない．また，prove(S,P,N) が失敗したときは，Sに変数が含まれていなければ，内部データベースに not_proved(S,N) を登録する

  (1) 内部データベースに not_proved(S,M) が登録されていて，$N \leq M$なら失敗する．

  (2) 内部データベースに proved(S,P,M) が登録されていて，$M \leq N$なら，求める証明を Pとして成功する．

  (3) Sがユーザ定義の公理のどれかと一致すれば成功する．

  (4) 体系が **ILZ** または **ILL** であり，Sの右辺の論理式が 2 個以上であれば失敗する．

  (5) 述語 select_rule(R,S,Ss,Pos) を呼び出すことにより，シーケント Sに適用可能な規則 Rを一つ選び，上シーケントのリスト Ss と主および副論理式の位置 Pos を求める．適用可能な規則がなくなれば prove(S,P,N) が失敗する．

  (6) Rが cut のとき，左上シーケントがユーザ定義の公理であるという制約条件を付ける．すなわち左上シーケントの証明の規則を axiom に単一化しておく．

  (7) Rが c(!) のとき，上シーケントに適用可能な規則が r(*), l(+), l(->), c(!), c(?), l(!) だけであるいう制約条件を付ける．現在は，SICStus Prolog の freeze 述語を用いて制約条件を付けている．

  (8) Rが c(?) のとき，上シーケントに適用可能な規則が r(*), l(+), l(->), c(!), c(?), r(?) だけであるいう制約条件を付ける．現在は，SICStus Prolog の freeze 述語を用いて制約条件を付けている．

  (9) Rが cut, c(!), c(?) のいずれかのときは Nを 1 減らす．$N \leq 0$ なら失敗し，select_rule にバックトラックする．

  (10) Ss 中の各シーケント S_iに対して prove(S_i,P_i,N) を再帰的に呼び出す．呼び出しが一つでも失敗すれば select_rule にバックトラックする．

  (11) $P = [R, Pos, S, P_1, \ldots, P_n]$ とし，内部データベースに proved(S,P,N) を登録する．

- 述語 select_rule($-R$, $+S$, $-Ss$, $-Pos$)
  述語 select_rule(R,S,Ss,Pos) は，シーケント Sに適用可能な規則 Rを一つ選び，上シーケントのリスト Ss と主および副論理式の位置 Pos を求める．バックトラックにより，すべての可能な Ss と Rを探す．ただし，invertible な規則 (すなわち l(/\,i), r(\/,i), r(*), l(+), l(->), l(all), r(exists), c(!), w(!), l(!), w(?), c(?), r(?) 以外の規則) が適用可能なときは，その一つの場合だけを決定的に選ぶ (バックトラックしてきても，他の Ss や Rは探さない)．変数条件は，述語 eigen_variable により制約条件として処理される．

- 述語 `eigen_variable(`$+V$, $+X$`)`
  V は **CLL-X** の R∀, L∃ の変数 y, X は下シーケントである．述語 `eigen_variable(`V`,`X`)` は X 中のすべての自由変数が V と異なるという制約条件を SICStus Prolog の `dif` 述語によって付与している．また，`freeze(`V`,fail)` として，V が他の項 (変数以外) と単一化されるのを防いでいる (`ax` によって単一化される可能性がある).

C.3 他プログラムからの利用

例えば以下のようにして利用できます.

```
:- compile(llprover).

use_llprover :-
    ll_init,
    set_system(ill,0),
    set_threshold(0),
    set_axioms([]),
    read(S0),
    convert_to_seq(S0, S),
    prove(S, P),
    set_style(onesided),
    set_output_form(pretty),
    print_proof(P).
```

参考文献

[1] J.-Y. Girard: Linear Logic. Theoretical Computer Science, 50:1–102, 1987.

[2] J.-Y. Girard, P. Taylor, Y. Lafont: Proofs and Types. Cambridge University Press, 1988.

[3] A. S. Troelstra: Lectures on Linear Logic. CSLI Lecture Notes No.29, 1992.

[4] 竹内外史: 線形論理入門．日本評論社，1995.