線形論理型言語の効率的なリソース管理モデル

姜 京順　番原 睦則　田村 直之

1987年にJ.-Y. Girardが発表した線形論理は，計算機科学への応用が期待されている新しい論理体系である．線形論理の論理型言語への応用は，特に活発な分野の一つであり，LO, ACL, Lolli, Lygon, Forum, およびLLP等の研究がある．これらのほとんどの言語では，リソース概念はプログラミング上で重要な位置を占めている．したがって，リソースを効率的良く取り扱う計算モデルの構築は重要な研究課題となっている．本論文で述べるレベル付きIOモデルは，効率的なリソース管理を目的とした計算モデルであり，実行中はただ一つのリソース表を保持すれば良いように工夫されている．本モデルは，LLPコンパイラ処理系で採用され，その有効性が示されている．

1 はじめに

1987年にJ.-Y. Girardが発表した線形論理[9][10][25]は，以下のような特徴を持ち，計算機科学への応用が期待されている新しい論理体系である．

- **資源を意識した論理**: 古典論理や直観主義論理では仮定は何度でも利用できるが，線形論理では基本的には仮定は一度しか利用できない．これが資源(リソース)の消費性を表していると考えられることから，資源を意識した論理と呼ばれる．
- **豊富な表現力**: 古典論理や直観主義論理は線形論理に埋め込むことができる．すなわち，線形論理はこれまでの論理を含んでいる．また，命題線形論理の範囲内でカウンタ機械を構成できるほど大きな表現力を持っている．
- **計算モデルとの対応**: π計算やペトリネットとの自然な対応が知られている．

線形論理の論理型言語への応用は，特に活発な分野の一つであり，LO [3], ACL [18], Lolli [16], Lygon [12], Forum [20], 著者らのLLP [4][5][17][22][26][28]およびTLLP [27]等の研究がある．

これらのうち，特にLolli[†1]は以下のような特徴を持っており，線形論理型言語の一つの方向性を示している．

- Prologの自然な拡張になっており，さらに線形論理の資源性を生かしたプログラミングが可能である．たとえばLolliでは，実行過程でPrologの節に相当する論理式を追加したり利用(消費)するようなプログラムを記述できる．
- Millerらのユニフォーム証明 [19]の考え方を，直観主義線形論理に適用した言語になっており，理論的なベースがしっかりしている．

Efficient Resource Management Model for Linear Logic Programming Languages.

Kyoung-Sun Kang, 釜山外国語大学校(韓国)コンピュータ電子工学部コンピュータ工学科, Department of Computer Engineering, Division of Electronic and Computer Engineering, Pusan University of Foreign Studies.

Mutsunori Banbara, 奈良工業高等専門学校, Nara National College of Technology.

Naoyuki Tamura, 神戸大学工学部情報知能工学科, Department of Computer and Systems Engineering, Kobe University.

コンピュータソフトウェア, Vol.18, No.0(2001), pp.138–154.

[論文]2000年1月6日受付.

†1 http://www.cs.hmc.edu/~hodas/research/lolli/

- LO, Lygon, Forumなどのように，基礎となる論理を古典線形論理まで広げておらず，効率的な実装が可能である．特に，著者らのLLP処理系は，Lolli言語のコンパイラによる実装を目指したものであり，広く用いられているPrologコンパイラ処理系と同程度の速度(SICStus Prologより2〜3倍遅く，SWI Prologより約2倍速い)を実現しており，さらにリソースの概念をうまく利用したプログラミングを行えば，より高速なプログラムを記述できる．

Lolliに限らず，他の線形論理型言語においても，リソース概念はプログラミング上で重要な位置を占めている．したがって，リソースを効率良く管理するための計算モデルの設計が重要な研究テーマとなっている[7][12][16].

しかしこれらの研究で述べられている計算モデルは，証明論上の問題点だけに着目しており，コンパイラ処理系でそのまま採用するには不十分であった．すなわち，これらの計算モデルでは，リソースを管理するための巨大なリスト構造を複数保持することを要求しており，これはSMLやPrologで記述したインタプリタ処理系では自然ではあるが，実際には効率が悪い．

本論文で述べるレベル付きIOモデルは，リソースを管理するためのテーブルを一つだけ保持するので良いように考えられた計算モデルである．この計算モデルは，著者らのLLPAM処理系[†2]で採用され，有効性が示されている．

本論文では，線形論理型言語Lolliの言語について述べた後，Lolliの元々の計算モデルであるIOモデルについて述べる．次に，IOモデルの問題点を明らかにし，それを解決するレベル付きIOモデルについて述べる．

2 線形論理型言語Lolliについて

論理型言語Prologは，ホーン節に対してのSLD導出をその計算モデルとしている．しかし，線形論理の場合は連言と選言が二通りずつあり，一般には分配律が成立しない．つまり節形式のような簡明な標準形が存在していない．したがって，線形論理に基づいた論理型言語を設計する場合，単純に導出原理を拡張することはできない．

HodasとMillerによる線形論理型言語Lolli [13][15][16]は，Millerらのユニフォーム証明 [19]の考え方を，直観主義線形論理に適用したものであり，ホーン節に対するSLD導出とは全く異なった考えで設計されている．

本章では，ユニフォーム証明の考え方，およびLolliの言語について述べる．本章の内容は基本的にはHodasらの論文[13][15][16]に従っている．説明のために一部変更している箇所については，脚注で述べる．

2.1 ユニフォーム証明

Millerらのユニフォーム証明[19]は，論理プログラミングの計算モデルの基礎を与えるものであり，以下のような考え方に基づいている．

- **計算＝ユニフォーム証明の探索**: 論理プログラミングでの計算は，直観主義シーケント計算のカットのない証明の探索であり，かつゴール主導(goal-directed)の探索とみなすことができる．ここで，ゴール主導の証明探索とは，証明を下から上に構築していく過程でゴール論理式(すなわちシーケントの右辺の論理式)が原子論理式でない場合は，必ず右導入規則を用いている証明だけを探索することを意味する．

  今，**ユニフォーム証明**(uniform proof)を，カットのない証明で，右辺が原子論理式でないシーケントは必ず右導入規則の結論になっているような証明と定義すれば，論理プログラミングでの計算はユニフォーム証明の(下から上への)探索とみなせる．
- **言語＝ユニフォーム証明探索が完全であるフラグメント**: もちろん，探索する証明の形を制限したのだから，すべての真な論理式が証明可能なわけではない．例えば直観主義論理で，$a \vee b \longrightarrow b \vee a$はユニフォーム証明を持たない．しかし，使用する論理式の形をうまく制限すれば，ユニフォーム証明だけの探索で完全になる．

  逆に，ユニフォーム証明探索が完全になるような論理式のフラグメントが論理プログラミング言語を定めると考えることができる．

†2 http://bach.seg.kobe-u.ac.jp/llp/
http://cs32.scitec.kobe-u.ac.jp/~banbara/PrologCafe/

Millerらは，直観主義論理において，$\top$, $\wedge$, $\supset$, $\forall$からなるフラグメントは，ユニフォーム証明の探索だけで完全であることを示した．また，それに基づいた論理型言語λ-Prolog [21] を設計している．

λ-Prologは，ホーン節を含んでおり，さらに強力である．例えば，ゴール$D \supset G$の実行は，仮定(すなわちプログラム)Dを追加した上でゴールGを実行することを意味する．

2.2 直観主義線形論理でのユニフォーム証明

HodasとMillerはユニフォーム証明の考え方を，一階の直観主義線形論理に適用した．

一階の直観主義線形論理の体系ILLを図1に示す(カット除去定理が成立するので，Cut規則は省いても良い)．ここで，シーケントの左辺は論理式のマルチ集合とし，Exchange規則を暗黙のうちに仮定する．

体系ILLでのユニフォーム証明を考えた場合，$\otimes$と!を自由に使用することはできない．たとえば，$a \otimes b \longrightarrow b \otimes a$, $!a \,\&\, b \longrightarrow !a$などはILLで証明可能であるがユニフォーム証明を持たない．しかし，$\otimes$と!は，線形論理で最も特徴的な論理結合子であり，これらを全く含まないフラグメントを考える意味は少ない．

そこでHodasらは，まず新しい含意記号$\Rightarrow$を導入することで，!を間接的に利用可能にした[15][16]．ここで，$B \Rightarrow C$は$(!B) \multimap C$を意味する．

また$\Rightarrow$の導入に伴い，シーケントを以下のような形式で表している．

$$\Gamma; \Delta \longrightarrow C$$

ここで，Γは論理式の集合(**無限コンテキスト**と呼ぶ)，Δは論理式のマルチ集合(**有界コンテキスト**と呼ぶ)，Cは論理式である．このシーケントは，ILLでの$!\Gamma, \Delta \longrightarrow C$に対応する．すなわち，$\Gamma$中の論理式は0回以上何度でも利用できる仮定，$\Delta$中の論理式はちょうど1回だけ利用できる仮定を意味する．なお，Γは論理式の集合であるので，Contraction規則およびExchange規則が暗黙のうちに仮定されており，Δは論理式のマルチ集合であるので，Exchange規則が暗黙のうちに仮定されている．

このように含意記号$\Rightarrow$を導入しシーケントの形式を変更した時，$\top$, $\&$, $\multimap$, $\Rightarrow$, $\forall$からなるフラグメントは，ユニフォーム証明の探索だけで完全になる．

しかし，やはり$\otimes$と!を直接的には含んでいないという問題点がある．

そこでHodasらは，利用できる論理式の制限をシーケントの左辺と右辺で別々にすることで，この問題を解決した．具体的には，以下のBNF定義で，Rがシーケントの左辺に現れる論理式(**リソース論理式**と呼ぶ)，Gが右辺に現れる論理式(**ゴール論理式**と呼ぶ)である．また，Aは原子論理式を表すものとする．

$$\begin{aligned} R &::= A \mid R_1 \,\&\, R_2 \mid G \multimap R \mid G \Rightarrow R \mid \forall x.R \\ G &::= 1 \mid \top \mid A \mid G_1 \otimes G_2 \mid G_1 \,\&\, G_2 \mid G_1 \oplus G_2 \mid \\ &\quad\; !G \mid R \multimap G \mid R \Rightarrow G \mid \forall x.G \mid \exists x.G \\ A &::= P(t_1, \cdots, t_n) \\ &\quad\; (\text{ただし}, P\text{はアリティ}n\text{の述語記号}, t_i\text{は項}) \end{aligned}$$

Hodasらの論文では，リソース論理式に$\top$が含まれているが，実質的な意味を持たないので削除している．体系ILLには$\top$の左側規則は存在しないため，リソース論理式としての$\top$は他に現れない原子論理式で代用できるからである．

このフラグメントに対して，ユニフォーム証明探索は完全である．したがって，この定義をプログラミング言語の構文として採用するので良いのだが，論理体系の記述を簡潔にするために，以下での論理体系の記述中では次のように変更したリソース論理式の定義を用いる(Aは原子論理式，$m \geq 1$)．

$$\begin{aligned} R &::= S_1 \,\&\, \cdots \,\&\, S_m \\ S &::= A \mid G \multimap A \mid \forall x.S \end{aligned}$$

Sは，Prologでのプログラム節に相当する形をしているので，**リソース節**と呼ぶことにする．S中の，Aがヘッド部，Gがボディ部に相当する．

このように変更しても，線形論理では以下が成立するので，表現力は全く変わっていない(ただしzは，Gに自由な出現を持たない変数とする)．

$$\begin{aligned} G \Rightarrow R &\equiv (!G) \multimap R \\ G \multimap (R_1 \,\&\, R_2) &\equiv (G \multimap R_1) \,\&\, (G \multimap R_2) \\ G_1 \multimap (G_2 \multimap R) &\equiv (G_1 \otimes G_2) \multimap R \\ G \multimap \forall x.R &\equiv \forall z.(G \multimap R[z/x]) \\ \forall x.(R_1 \,\&\, R_2) &\equiv (\forall x.R_1) \,\&\, (\forall x.R_2) \\ (R_1 \,\&\, R_2) \,\&\, R_3 &\equiv R_1 \,\&\, (R_2 \,\&\, R_3) \end{aligned}$$

リソース論理式のこのような変換は，言語処理系が自動的に行うものとすれば，言語の記述範囲はもとのまま

$$\frac{}{B \longrightarrow B}\ (\text{Identity}) \qquad \frac{\Delta_1 \longrightarrow B \quad \Delta_2, B \longrightarrow C}{\Delta_1, \Delta_2 \longrightarrow C}\ (\text{Cut})$$

$$\frac{}{\Delta, 0 \longrightarrow C}\ (\text{L0}) \qquad \frac{}{\Delta \longrightarrow \top}\ (\text{R}\top)$$

$$\frac{\Delta \longrightarrow C}{\Delta, 1 \longrightarrow C}\ (\text{L1}) \qquad \frac{}{\longrightarrow 1}\ (\text{R1})$$

$$\frac{\Delta, B_i \longrightarrow C}{\Delta, B_1 \,\&\, B_2 \longrightarrow C}\ (\text{L}\,\&_i) \qquad \frac{\Delta \longrightarrow C_1 \quad \Delta \longrightarrow C_2}{\Delta \longrightarrow C_1 \,\&\, C_2}\ (\text{R}\,\&)$$

$$\frac{\Delta, B_1, B_2 \longrightarrow C}{\Delta, B_1 \otimes B_2 \longrightarrow C}\ (\text{L}\otimes) \qquad \frac{\Delta_1 \longrightarrow C_1 \quad \Delta_2 \longrightarrow C_2}{\Delta_1, \Delta_2 \longrightarrow C_1 \otimes C_2}\ (\text{R}\otimes)$$

$$\frac{\Delta, B_1 \longrightarrow C \quad \Delta, B_2 \longrightarrow C}{\Delta, B_1 \oplus B_2 \longrightarrow C}\ (\text{L}\oplus) \qquad \frac{\Delta \longrightarrow C_i}{\Delta \longrightarrow C_1 \oplus C_2}\ (\text{R}\oplus_i)$$

$$\frac{\Delta_1 \longrightarrow C_1 \quad \Delta_2, B \longrightarrow C_2}{\Delta_1, \Delta_2, C_1 \multimap B \longrightarrow C_2}\ (\text{L}\multimap) \qquad \frac{\Delta, B \longrightarrow C}{\Delta \longrightarrow B \multimap C}\ (\text{R}\multimap)$$

$$\frac{\Delta, B[t/x] \longrightarrow C}{\Delta, \forall x.B \longrightarrow C}\ (\text{L}\forall) \qquad \frac{\Delta \longrightarrow C[y/x]}{\Delta \longrightarrow \forall x.C}\ (\text{R}\forall)$$

$$\frac{\Delta, B[y/x] \longrightarrow C}{\Delta, \exists x.B \longrightarrow C}\ (\text{L}\exists) \qquad \frac{\Delta \longrightarrow C[t/x]}{\Delta \longrightarrow \exists x.C}\ (\text{R}\exists)$$

$$\frac{\Delta, B \longrightarrow C}{\Delta, !\,B \longrightarrow C}\ (\text{L}\,!) \qquad \frac{!\,\Delta \longrightarrow C}{!\,\Delta \longrightarrow !\,C}\ (\text{R}\,!)$$

$$\frac{\Delta \longrightarrow C}{\Delta, !\,B \longrightarrow C}\ (\text{W}\,!) \qquad \frac{\Delta, !\,B, !\,B \longrightarrow C}{\Delta, !\,B \longrightarrow C}\ (\text{C}\,!)$$

(変数 y は下シーケントで自由な出現を持たない)

図1 直観主義線形論理の体系ILL

で良い.

図2にこのフラグメントに対応する体系HLLを示す†3. ここで, BCおよびBC!規則中の$\|R\|$は, Aまたは$G \multimap A$の形をしたリソース論理式を要素とする集合で, 以下のように再帰的に定義される(ただし, $\bigcup_t$でtはすべての項を動く).

1. $R = A$の時, $\|R\| = \{A\}$.
2. $R = G \multimap A$の時, $\|R\| = \{G \multimap A\}$.
3. $R = \forall x.S$の時, $\|R\| = \bigcup_t \|S[t/x]\|$.
4. $R = S_1 \,\&\, \cdots \,\&\, S_m$の時, $\|R\| = \|S_1\| \cup \cdots \cup \|S_m\|$.

HLLには左導入規則が全く現れていない. これは, Identity規則を含めてすべての左導入規則がBCおよびBC!規則(これらはバックチェイン規則と呼ばれる)に置き換えられるからである. また, 明らかにHLLでの証明はすべてユニフォーム証明になる.

体系HLLについて以下が成り立つ.

命題 2.1 (Hodas and Miller) 体系HLLは体系ILLと同値である. すなわち, Gをゴール論理式, Γをリソース論理式の集合, Δをリソース論理式のマルチ

†3 論文[16]中での$\mathcal{L}''$に相当するが, absorb 規則に替えてBC!規則を導入している. またBC規則も変更している. これは後述のIOモデルとの対応を明確にするためである.

集合とする．このとき以下が成立する．

$$\mathrm{HLL} \vdash \Gamma; \Delta \longrightarrow G \iff \mathrm{ILL} \vdash\, !\Gamma^*, \Delta^* \longrightarrow G^*$$

ただし，Γ^* は Γ 中の $B \Rightarrow C$ の形の論理式を $(!\,B) \multimap C$ ですべて置き換えたものを表す．Δ^*, G^* も同様である．

証明 Hodasの論文[15]の $\mathcal{L}''$ とILLの同値性の証明と同様にして証明できる．ただし，$\mathcal{L}''$ のabsorb規則は，必ずBC規則のすぐ下だけに出てくるように制限できるので，HLLでは以下のようにBC!規則に対応することになる．

$$\dfrac{\dfrac{}{\Gamma, R; R \longrightarrow A}\ (BC)}{\Gamma, R; \longrightarrow A}\ (absorb)\ \text{(ただし，} A \in \|R\|\text{)} \iff \dfrac{}{\Gamma, R; \longrightarrow A}\ (\mathrm{BC!}_1)\ \text{(ただし，} A \in \|R\|\text{)}$$

$$\dfrac{\dfrac{\Gamma, R; \Delta \longrightarrow G}{\Gamma, R; \Delta, R \longrightarrow A}\ (BC)}{\Gamma, R; \Delta \longrightarrow A}\ (absorb)\ \text{(ただし，} G \multimap A \in \|R\|\text{)} \iff \dfrac{\Gamma, R; \Delta \longrightarrow G}{\Gamma, R; \Delta \longrightarrow A}\ (\mathrm{BC!}_2)\ \text{(ただし，} G \multimap A \in \|R\|\text{)}$$

□

ユニフォーム証明を古典線形論理に拡張する研究も行われているが(Harland and Pym [11], Andreoli [2], Miller [20])，本論文は直観主義線形論理を対象としているので詳細は述べない．

2.3 線形論理型言語Lolli

HodasとMillerによる線形論理型言語Lolliは，前述のHLLに基づいた言語である．

LolliはPrologおよびλ-Prologを含んでいる(ただしλ-Prologの高階単一化機能は含んでいない)．また，バックチェイン規則BC, BC!が，PrologでのSLD導出による推論に相当する．すなわち，バックチェイン規則の適用(下から上への)は，ヘッド部 A を持つリソース R を使用することによって，ボディ部 G を呼び出すことに対応する．

Prologのプログラム節に相当する論理式は，閉じたリソース論理式であり，何度でも使用できるから無限コンテキストに置かれることになる．

今，Prologプログラム中のプログラム節を $C_1, \ldots, C_m$ とし，実行するゴールを G とする．各 C_i は $\forall \vec{x}.(A_1 \wedge \cdots \wedge A_n \supset A)$ の形をしているから(A_j, A は原子論理式)，これを $\forall \vec{x}.(A_1 \otimes \cdots \otimes A_n \multimap A)$ と書き換え，C_i^* で表す．また G も $\exists \vec{x}.(A_1 \wedge \cdots \wedge A_n)$ の形をしているから，これを $\exists \vec{x}.(A_1 \otimes \cdots \otimes A_n)$ と書き換え，G^* で表す．この時，HLLでの $C_1^*, \ldots, C_n^*; \longrightarrow G^*$ のユニフォーム証明の探索がPrologでの実行過程に対応する．

例 2.1 規則 $q \wedge r \supset p$，事実 q，事実 r からなるPrologプログラムに対してのゴール $p \wedge q$ の実行は，以下のようなユニフォーム証明の探索に対応する(ただし，Γ はリソース論理式の集合 $\{q \otimes r \multimap p, q, r\}$ を表す)．

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma; \longrightarrow q}\ (\mathrm{BC!}_1) \quad \dfrac{}{\Gamma; \longrightarrow r}\ (\mathrm{BC!}_1)}{\Gamma; \longrightarrow q \otimes r}\ (\mathrm{R}\otimes)}{\Gamma; \longrightarrow p}\ (\mathrm{BC!}_2) \qquad \dfrac{}{\Gamma; \longrightarrow q}\ (\mathrm{BC!}_1)}{\Gamma; \longrightarrow p \otimes q}\ (\mathrm{R}\otimes)$$

この考察に従い，プログラム節 C，リソース論理式 R，ゴール論理式 G の記法を，Prologの記法に従って以下のように定める†4．ただし，演算子の優先順位は弱いものから順に，“`:-`”, “`;`”, “`&`”, “`,`”, “`-<>`”, “`=>`”, “`!`”とし，“`:-`”以外の2項演算子は右結合的とする．

$$\begin{array}{lcl} C & ::= & A\,. \mid A\texttt{:-}G\,. \\ R & ::= & S_1\texttt{\&}\cdots\texttt{\&}S_m \\ S & ::= & A \mid G\texttt{-<>}A \mid A\texttt{:-}G \mid \texttt{forall(}X\texttt{,}S\texttt{)} \\ G & ::= & \texttt{true} \mid \texttt{erase} \mid A \mid G_1\texttt{,}G_2 \mid G_1\texttt{\&}G_2 \mid \\ & & G_1\texttt{;}G_2 \mid \texttt{!}G \mid R\texttt{-<>}G \mid G\texttt{:-}R \mid R\texttt{=>}G \mid \\ & & \texttt{forall(}X\texttt{,}G\texttt{)} \mid \texttt{exists(}X\texttt{,}G\texttt{)} \end{array}$$

ただし，プログラム中の記法と論理式の対応は，以下の通りである．

†4 Lolliでは，本論文での記法とは異なり，ML風の記法を用いている．

$$\frac{}{\Gamma;\longrightarrow 1}\ (\mathrm{R1}) \qquad \frac{}{\Gamma;\Delta\longrightarrow \top}\ (\mathrm{R}\top)$$

$$\frac{\Gamma;\Delta_1\longrightarrow G_1 \quad \Gamma;\Delta_2\longrightarrow G_2}{\Gamma;\Delta_1,\Delta_2\longrightarrow G_1\otimes G_2}\ (\mathrm{R}\otimes) \qquad \frac{\Gamma;\Delta\longrightarrow G_1 \quad \Gamma;\Delta\longrightarrow G_2}{\Gamma;\Delta\longrightarrow G_1\,\&\,G_2}\ (\mathrm{R}\,\&)$$

$$\frac{\Gamma;\Delta\longrightarrow G_i}{\Gamma;\Delta\longrightarrow G_1\oplus G_2}\ (\mathrm{R}\oplus_i) \qquad \frac{\Gamma;\longrightarrow G}{\Gamma;\longrightarrow !\,G}\ (\mathrm{R}\,!)$$

$$\frac{\Gamma;\Delta,R\longrightarrow G}{\Gamma;\Delta\longrightarrow R\multimap G}\ (\mathrm{R}\multimap) \qquad \frac{\Gamma,R;\Delta\longrightarrow G}{\Gamma;\Delta\longrightarrow R\Rightarrow G}\ (\mathrm{R}\Rightarrow)$$

$$\frac{\Gamma;\Delta\longrightarrow G[y/x]}{\Gamma;\Delta\longrightarrow \forall x.G}\ (\mathrm{R}\forall) \qquad \frac{\Gamma;\Delta\longrightarrow G[t/x]}{\Gamma;\Delta\longrightarrow \exists x.G}\ (\mathrm{R}\exists)$$

$$\frac{}{\Gamma;R\longrightarrow A}\ (\mathrm{BC}_1)\ (\text{ただし，}\ A\in\|R\|) \qquad \frac{\Gamma;\Delta\longrightarrow G}{\Gamma;\Delta,R\longrightarrow A}\ (\mathrm{BC}_2)\ (\text{ただし，}\ G\multimap A\in\|R\|)$$

$$\frac{}{\Gamma,R;\longrightarrow A}\ (\mathrm{BC!}_1)\ (\text{ただし，}\ A\in\|R\|) \qquad \frac{\Gamma,R;\Delta\longrightarrow G}{\Gamma,R;\Delta\longrightarrow A}\ (\mathrm{BC!}_2)\ (\text{ただし，}\ G\multimap A\in\|R\|)$$

(変数 y は下シーケントで自由な出現を持たない)

図2 体系HLL

プログラム中の記法	論理式
`true`	1
`erase`	$\top$
B`,`C	$B\otimes C$
B`&`C	$B\,\&\,C$
B`;`C	$B\oplus C$
B`-<>`C	$B\multimap C$
C`:-`B	$B\multimap C$
B`=>`C	$B\Rightarrow C$
`!`B	$!\,B$
`forall(`X`,`B`)`	$\forall x.B$
`exists(`X`,`B`)`	$\exists x.B$

また，プログラム節Cは，Prologと同様に，すべての自由変数を全称限量子で束縛した閉じた論理式を表すとする．したがって，プログラム節Aは$\forall\vec{x}.A$を表し，A`:-`Gは，$\forall\vec{x}.(G\multimap A)$を表している．

ゴール論理式として，AおよびG_1`,`G_2だけをもちいたLolliプログラムは，構文的にも操作意味論的にも純Prologと全く同一になる．さらに，リソース論理式を用いれば，線形論理のリソース消費性の特徴を生かしたプログラムを記述できる．

2.4 Lolliでのプログラミング

本節では，Lolliプログラムの操作的な意味について，直観的な説明を行う．

ゴールR`-<>`G (あるいはG`:-`R)は，リソースRを有界コンテキストに追加した上でゴールGを実行する．Rは一度だけ使用可能であり，ちょうど一度だけしか使用できない(BC規則によって)．Rは一般にS_1`&`$\cdots$`&`S_mの形をしているが，S_iのどれか一つだけが使用できる．S_iがAの形の場合は，Aとマッチするゴールの実行によって消費される(BC_1規則)．S_iがG`-<>`A (あるいはA`:-`G)の形の場合も同様に，Aとマッチするゴールの実行によって消費されるが，同時にGが実行される(BC_2規則)．

ゴールR`=>`Gは，リソースRを無限コンテキストに追加した上でゴールGを実行する．したがって，Rは0

回以上何度でも使用できる(BC!規則によって)．これは，Prologでの`assert`に近いが，追加されているスコープはG中に限られており，バックトラックすれば消去される点，Rが自由変数を含むことが可能な点で異なっている．たとえばゴール`forall(X,p(X))=>`Gの実行は，事実`p(X)`を`assert`した上でGを実行するのと同様である．一方，ゴール`p(X)=>`Gの実行の場合は，変数`X`が全称限量子で束縛されていないため，Gの実行中に`X`の値を決めることが可能である．

各ゴール論理式およびリソース論理式についての説明は以下のようになる．

- 原子論理式のゴールAは，リソースの消費かプログラム節の呼出しを意味する(これらの選択は非決定的である．すなわちバックトラックによりすべての可能性が実行される)．
- ゴールR`-<>`G (あるいはG`:-`R)は，リソースRを有界コンテキストに追加した後，ゴールGを実行する．RはG中ですべて消費されなければならない．たとえばゴール`p(1)-<>p(X)`は，`X=1`となって成功する．
- ゴールR`=>`GはリソースRを無限コンテキストに追加した後，ゴールGを実行する．したがって，リソースRは0回以上何回でも利用できる．
- ゴールG_1`,`G_2はPrologのG_1`,`G_2と同様に実行される．G_1中で消費されたリソースはG_2中では消費できない．たとえばゴール`p(1)-<>p(2)-<>(p(X),p(Y))`は，リソース`p(1)`と`p(2)`を追加したのち，ゴール`(p(X),p(Y))`を実行する．これは，`X=1`, `Y=2`または`X=2`, `Y=1`となって成功する．
- ゴールG_1`&`G_2もまたPrologのG_1`,`G_2と同様であるが，実行の前にリソースがコピーされ，G_1とG_2で消費されるリソースは同一でなければならない．たとえばゴール`p(1)-<>p(2)-<>((p(X)&p(Y)),p(Z))`では，ゴール`p(X)`と`p(Y)`は，同一のリソース(`p(1)`または`p(2)`)を消費しなければならない．したがって，このゴールは`X=Y=1,Z=2`または`X=Y=2`, `Z=1`となって成功する．
- ゴールG_1`;`G_2は，PrologのゴールG_1`;`G_2と同様であり，G_1の実行が失敗した後にG_2を実行する．
- ゴール`!`Gは，ゴールGと同様であるが，有界コンテキストのリソースは使用できない．
- ゴール`true`はPrologの`true`と同様であり，直ちに成功する．
- ゴール`erase`もPrologの`true`と同様だが，いくつかのリソースが消費されないまま残ってもよいことを表す．すなわち`erase`は，いくつかのリソースを暗黙的に消費する．たとえばゴール`p(1)-<>true`は，リソース`p(1)`が消費されないまま残っているので失敗するが，ゴール`p(1)-<>erase`は成功する．
- リソースS_1`&`$\cdots$`&`S_mは，選択可能なリソースを表す．S_iのどれか一つだけが消費可能である．たとえばゴール`(p(1)&p(2))-<>p(X)`は`X=1`または`X=2`となって成功する．
- リソースG`-<>`AあるいはA`:-`Gは，ルールタイプのリソースを表す．すなわちリソース消費の際に，ボディ部Gが実行される．たとえばゴール`(q-<>p)-<>q-<>p`は，リソース`q-<>p`と`q`を追加したのち，ゴール`p`を実行する．これは，リソース`q-<>p`の消費の際に，`q`が消費され，成功する．

```
queens(N,Q) :-
    result(Q)-<> place(N,N).
place(1,N) :-
    c(1)-<>u(2)-<>d(0)-<> solve(N,[]).
place(I,N) :-
    I > 1, I1 is I-1,
    U1 is 2*I, U2 is 2*I-1,
    D1 is I-1, D2 is 1-I,
    c(I)-<>u(U1)-<>u(U2)-<>d(D1)-<>d(D2)-<>
    place(I1,N).
solve(0,Q) :-
    result(Q), erase.
solve(I,Q) :-
    I > 0, c(J),
    U is I+J, u(U),
    D is I-J, d(D),
    I1 is I-1, solve(I1,[J|Q]).
```

図3　Lolliで記述したN-クイーンのプログラム

例 2.2 図3は，Lolliで記述したN-クイーンのプログラムである．

たとえば，`queen(4,Q)`を実行した場合，リソース`result(Q)`が追加され，`place(4,4)`が実行される．

```
queens(N,Q) :-
    gen(1,N,Js),
    N2 is 2*N-1, gen(2,N2,Us),
    D0 is 1-N, gen(D0,N2,Ds),
    sol(N,Js,Us,Ds,Q).
sol(0,_,_,_,[]).
sol(I,Js0,Us0,Ds0,[J|Q]) :-
    I > 0, del(J,Js0,Js),
    U is I+J, del(U,Us0,Us),
    D is I-J, del(D,Ds0,Ds),
    I1 is I-1, sol(I1,Js,Us,Ds,Q).
del(X,[X|Xs],Xs).
del(X,[Y|Ys],[Y|Zs]) :- del(X,Ys,Zs).
gen(_,0,[]).
gen(I,N,[I|Ns]) :-
    N>0, I1 is I+1, N1 is N-1,
    gen(I1,N1,Ns).
```

図4 Prologで記述したN-クイーンのプログラム

`result(Q)`中の変数`Q`は，結果を表す変数であり，`solve`の中で結果の配置と単一化される．

`place(4,4)`は，リソース`c(1)`, ..., `c(4)`, `u(2)`, ..., `u(8)`, `d(-3)`, ..., `d(3)`を追加してから`solve(4,[])`を実行する．これらのリソース`c`, `u`, `d`は，各列，各右上がりのライン，各右下がりのラインにそれぞれ対応している．`solve`中で`I`行目にクイーンを置くときに，`c(J)`, `u(I+J)`, `d(I-J)`を消費することによって，各列，各右上がりのライン，各右下がりのラインに高々一つしかクイーンを置けないという制約条件を表している．

また，図4は，Prologで記述したN-クイーンのプログラムであり(文献[6]と同様のもの)，Lolliでリソースを利用して行っていた制約条件のチェックをリストを用いて行っている．したがって，各制約条件のチェックには，リストの長さに比例する(したがって，Nに比例する)計算量がかかる．

この外，論文[14], [16], [23]には，Lolliのプログラム例として，自然言語のパーザ，直観主義命題論理の証明系，実験計画でのBIBD配置の探索プログラム，ハミルトン経路の探索，覆面算，ペントミノなどのパズルプログラムが記載されている．

2.5 IOモデル

HLLでのユニフォーム証明を探索する場合，一番問題となるのがR⊗規則である．

$$\frac{\Gamma;\Delta_1 \longrightarrow G_1 \quad \Gamma;\Delta_2 \longrightarrow G_2}{\Gamma;\Delta_1,\Delta_2 \longrightarrow G_1 \otimes G_2}\ (\mathrm{R}\otimes)$$

この規則を下から上へ適用しようとするとき，有界コンテキストをΔ_1とΔ_2に分割する必要がある．すなわち，$\Gamma;\Delta \longrightarrow G_1 \otimes G_2$を証明しようとするとき，$\Gamma;\Delta_1 \longrightarrow G_1$と$\Gamma;\Delta_2 \longrightarrow G_2$がそれぞれ証明可能になるように，マルチ集合$\Delta$を二つのマルチ集合$\Delta_1$と$\Delta_2$に分割しなければならない．この分割の仕方は，$\Delta$のサイズを$n$とすると$2^n$になり，非決定性が大きい．

Hodasらは，この問題の解決法として，リソース分割を遅延的に行うための**IOモデル**を提案した．このモデルでは，ゴールをリソースを消費する消費者と考える．すなわち，$G_1 \otimes G_2$の実行では，まずΔのうちのいくつかを使ってG_1を証明し，その後，残ったリソースを使ってG_2を証明する．残ったリソースでのG_2の証明が失敗すれば，G_1の証明にバックトラックし，別の消費方[redacted]を探す．

以下でIOモデルについて説明するが，リソース分割の問題は線形論理の乗法的論理演算子($\otimes$, $\multimap$など)の取り扱いの問題であり，限量子の取り扱いとは無関係に議論できるので，簡単のため対象とする論理を直観主義命題線形論理とする．

IOモデルでは，HLLのシーケントの左辺をIOコンテキストと呼ばれる論理式のリストで表現している．すなわち，**IOコンテキスト**とは，リスト$[r_1, r_2, \ldots, r_n]$で，各r_iは，リソース論理式(**有界リソース**と呼ぶ)，またはリソース論理式に!をつけた論理式(**無限リソース**と呼ぶ)，または1である．1は，リソースがそれまでの実行で消費されたことを示す特別の印として使用する．有界リソースは，HLLのシーケント中の有界コンテキスト中のリソース論理式に対応し，無限リソースは無限コンテキスト中のリソースに対応する．

IOコンテキストI, Oについて，OがIの部分コンテキストになっていること，すなわちOは，I中のいくつかの有界リソースを1に置き換えたリストになっていることを意味する述語$subcontext(O, I)$を以下のように定義する．

$subcontext([s_1, \ldots, s_n], [r_1, \ldots, r_n]) \Longleftrightarrow$

各$i = 1, 2, \ldots, n$について，(1) r_iが有界リソース

$$\frac{}{I\,\{1\}\,I}\ (1) \qquad \frac{subcontext(O,I)}{I\,\{\top\}\,O}\ (\top)$$

$$\frac{I\,\{G_1\}\,M \quad M\,\{G_2\}\,O}{I\,\{G_1 \otimes G_2\}\,O}\ (\otimes) \qquad \frac{I\,\{G_1\}\,O \quad I\,\{G_2\}\,O}{I\,\{G_1 \,\&\, G_2\}\,O}\ (\&)$$

$$\frac{I\,\{G_i\}\,O}{I\,\{G_1 \oplus G_2\}\,O}\ (\oplus_i) \qquad \frac{I\,\{G\}\,I}{I\,\{!\,G\}\,I}\ (!)$$

$$\frac{[R\,|\,I]\,\{G\}\,[1\,|\,O]}{I\,\{R \multimap G\}\,O}\ (\multimap) \qquad \frac{[!\,R\,|\,I]\,\{G\}\,[!\,R\,|\,O]}{I\,\{R \Rightarrow G\}\,O}\ (\Rightarrow)$$

$$\frac{pick(I,O,A)}{I\,\{A\}\,O}\ (\mathrm{BC}_1) \qquad \frac{pick(I,M,G \multimap A) \quad M\,\{G\}\,O}{I\,\{A\}\,O}\ (\mathrm{BC}_2)$$

図5 IOモデルの体系

のときは$s_i = r_i$または$s_i = 1$，(2) 有界リソース以外のときは$s_i = r_i$となっている．

また，IOコンテキストIからリソース節Sを取り出し，それを消費したIOコンテキストがOになっていることを意味する述語$pick(I,O,S)$を以下のように定義する．

$pick([r_1,\ldots,r_n],[s_1,\ldots,s_n],S) \Longleftrightarrow$

(1) ある有界リソース$r_i = S_1 \,\&\cdots\&\, S_m$について$S_k = S$であり，$s_i = 1$かつ$s_j = r_j$ $(j \neq i)$となっている，または，(2) ある無限リソース$r_i = !(S_1 \,\&\cdots\&\, S_m)$について$S_k = S$であり，$s_j = r_j$ $(1 \leq j \leq n)$となっている．

HLLでの各シーケントは，IOモデルでは，以下のような式で表現される．

$$I\,\{G\}\,O$$

ここで，Gはゴール論理式，I, OはIOコンテキストである．特に，Iは**入力コンテキスト**と呼ばれ，ゴールGの実行過程で消費できるリソースを表し，Oは**出力コンテキスト**と呼ばれ，ゴールGの実行後まで残ったリソースを表す．

IOモデルの体系IOを図5に示す．

IOモデルとHLLの同値性を述べるために，差$I - O$を，$subcontext(O,I)$が成り立っているIOコンテキストに対して定義する．IOコンテキストの**差**$I - O$は組$\langle \Gamma, \Delta \rangle$であり，$\Gamma$は$!R$が$I$の要素(したがって$O$の要素にも)になっているリソース論理式$R$の集合，$\Delta$は$R$が$I$の要素になっており$O$では1になっているリソース論理式$R$のマルチ集合である．体系IOで$I\,\{G\}\,O$が証明可能なとき，常に$subcontext(O,I)$が成り立っていることに注意されたい．

$$\dfrac{\dfrac{\dfrac{\dfrac{}{p;\longrightarrow p}\ (\mathrm{BC}!_1) \quad \dfrac{}{p;q\,\&\,r \longrightarrow q}\ (\mathrm{BC}_1)}{p;q\,\&\,r \longrightarrow p \otimes q}\ (\mathrm{R}\otimes)}{p;\longrightarrow (q\,\&\,r) \multimap (p \otimes q)}\ (\mathrm{R}\multimap)}{;\longrightarrow p \Rightarrow ((q\,\&\,r) \multimap (p \otimes q))}\ (\mathrm{R}\Rightarrow)$$

図6 HLLにおける$p \Rightarrow ((q\,\&\,r) \multimap (p \otimes q))$の証明図

命題 2.2 (Hodas and Miller) 体系IOは体系HLLと同値である．すなわち，Gをゴール論理式，I, Oを$subcontext(O,I)$を満たすIOコンテキストとし，$I - O = \langle \Gamma, \Delta \rangle$であるとする．このとき以下が成立する．

$$\mathrm{IO} \vdash I\,\{G\}\,O \iff \mathrm{HLL} \vdash \Gamma; \Delta \longrightarrow G$$

証明 Hodasの論文[16]と同様にして証明できる． □

例 2.3 Γ, Δが$\emptyset$, Gが$p \Rightarrow ((q\,\&\,r) \multimap (p \otimes q))$のとき，HLLでの証明は図6のようになる．R$\otimes$規則でのリソース分割の非決定性があり，うまくリソースを分割しないと(つまり，$q\,\&\,r$を右のシーケントに持ってこないと)，このような証明図は得られないことがわかる．

一方，IOモデルでの証明は図7のようになり，リソース分割が遅延的に行われている．

$$\dfrac{\dfrac{\dfrac{pick([q\,\&\,r,!\,p],[q\,\&\,r,!\,p],p)}{[q\,\&\,r,!\,p]\,\{p\}\,[q\,\&\,r,!\,p]}\ (\mathrm{BC}_1)\quad \dfrac{pick([q\,\&\,r,!\,p],[1,!\,p],q)}{[q\,\&\,r,!\,p]\,\{q\}\,[1,!\,p]}\ (\mathrm{BC}_1)}{[q\,\&\,r,!\,p]\,\{p\otimes q\}\,[1,!\,p]}\ (\otimes)}{[!\,p]\,\{(q\,\&\,r)\multimap(p\otimes q)\}\,[!\,p]}\ (\multimap)}{[\,]\,\{p\Rightarrow((q\,\&\,r)\multimap(p\otimes q))\}\,[\,]}\ (\Rightarrow)$$

図7 IOモデルにおける$p\Rightarrow((q\,\&\,r)\multimap(p\otimes q))$の証明図

3 リソース管理モデルについて

3.1 IOモデルの問題点

HodasらのIOモデルは，リソース分割を遅延的に行うという点で，優れたリソース管理モデルといえるが，問題も残っている．

1. ***pick*におけるIOコンテキストの探索の問題:**
   BC_1, BC_2規則中の*pick*では，IOコンテキストI中からヘッド部がAに一致するリソース節Sを探索する必要がある．IOコンテキストをリスト構造で表現している場合，リストの長さに比例する時間がかかってしまう．
2. ***pick*におけるIOコンテキストの再構築の問題:**
   さらに$pick(I,O,S)$では，新しいIOコンテキストOを構築する必要がある．たとえば，前節の例2.3では，IOコンテキスト$[q\,\&\,r,!\,p]$の第一要素を変更し，新たに$[1,!\,p]$を作成する必要がある．これも，リスト構造を用いている場合，IOコンテキストのサイズに比例した時間と領域を必要とする．

これらは，*pick*を用いているBC_1, BC_2規則が，Prologでの述語呼び出しに相当することを考えれば，到底看過できない問題点である．

第1の「*pick*におけるIOコンテキストの探索」という問題点は，IOコンテキストをリスト構造ではなく，述語名と引数などをキーとしたハッシュ表で実現すれば解決できると考えられる．実際，我々の実装したコンパイラ処理系ではそのようにしている[22]．

第2の「*pick*におけるIOコンテキストの再構築」という問題点は，IOコンテキストを破壊的に書き換えれば解決できるように見える(もちろん，バックトラック時には元に戻すようにする)．たとえば，例2.3の場合，IOコンテキスト$[q\&r,!p]$の第一要素を破壊的に書き換えて，$[1,!p]$を作成する操作は定数時間で行うことが可能である(IOコンテキストをハッシュ表で実行した場合)．しかし，この場合，$G_1\,\&\,G_2$の実行で問題が生じる．

$$\dfrac{I\,\{G_1\}\,O\quad I\,\{G_2\}\,O}{I\,\{G_1\,\&\,G_2\}\,O}\ (\&)$$

上の規則を見るとわかるように，G_1の実行とG_2の実行とでは，同一の入力コンテキストIを必要とする．しかし，IOコンテキストを破壊的に書き換えた場合，G_1の実行によってIが書き換えられてしまうため，G_1の実行の前に，Iのコピーを保存しておく必要がある．また，G_1の実行によって書き換えられた結果であるOも，G_2の結果と一致するかどうかを調べる必要があるから，やはりコピーを保存しておく必要がある．

しかし，Cervesatoらが論文[8]で述べているように，G_2の実行は，G_1で消費されたリソースだけを用い，かつそれらをすべて消費すると考えることができる．したがって，各リソースに消費済みかどうかのフラグを付与し，G_1で消費されたリソースが判別できるようにする方法が考えられる．

しかし，単なるフラグだけでは，ゴール中に&がネストしている場合にうまく働かない．そこで本論文では，各リソースにフラグではなく整数値を付与した計算モデル(レベル付きIOモデルと呼ぶ)を提案する．

3.2 レベル付きIOモデル

レベル付きIOモデルの体系IOLでは，各リソース論理式に，現在使用可能かどうかを表すための整数値を割り当てる．すなわち，**レベル付きリソース論理式**とは，リソース論理式Rとレベルを表す整数ℓの対であり，$\langle R,\ell\rangle$で表わされる．特に，無限リソースに対しては$\ell=0$とし，有界リソースに対しては$\ell\neq 0$とする．

また，レベル付きIOモデルでは，二つの整数L, Uを付与したシーケントを用いる．

$$\vdash_{L,U} I\,\{G\}\,O$$

ここで，Lは正整数で，消費可能なリソース論理式のレベルを表しており，**消費可能レベル**と呼ばれる．初期値は1である．Uは負整数で，消費後に割り当てられるレベルを表しており，**消費後レベル**と呼ばれる．初期値は-1である．Gはゴール論理式である．I, Oはレベル付きリソース論理式のリストであり，**IOL コンテキスト**と呼ばれる．特に，すべてのレベルがU以上L以下であるとき，(L,U)-コンテキストと呼ぶことにする．

$\vdash_{L,U} I\,\{G\}\,O$の実行においては，I中でレベルがL(または0)であるリソースだけが消費可能と考え，消費された場合にはレベルはU (または0のまま) となるようにする．このようにすることで，$G_1 \,\&\, G_2$中のG_1の実行でどのリソースが消費されたかを判別できるようになる．すなわち，$\vdash_{L,U} I\,\{G_1 \,\&\, G_2\}\,O$は以下のようなステップで実行される．

1. G_1でどのリソースが消費されたかわかるように，消費後レベルを$U-1$としG_1を実行する．
2. IOLコンテキスト中で，レベルが$U-1$となっているのはG_1で消費されたリソースであるから，それらのレベルを$L+1$に変更する．
3. 消費可能レベルを$L+1$としてG_2を実行する．
4. G_2は，G_1で消費したリソースすべてを消費しなければならない．すなわち，IOLコンテキスト中にレベル$L+1$のリソースが残っていてはならないので，それをチェックする．

また，$\vdash_{L,U} I\,\{!G\}\,O$は以下のようなステップで実行される．

1. $!G$の実行の成功は入力コンテキスト中の有界リソースを消費せずにGが証明できることであるため，Gが有界リソースを消費できないように，消費可能レベルを$L+1$にする．
2. Gを実行する．

このように考えた，レベル付きIOモデルの体系IOLを図8に示す．

図中の，*subcontext*, *pick*, *change*, *thinable*はそれぞれ以下のように定義される述語である．

$subcontext_{U,L}([s_1,\ldots,s_n],[r_1,\ldots,r_n]) \Longleftrightarrow$
各$i=1,2,\ldots,n$について，(1) $r_i=\langle R,L\rangle$のときは$s_i=r_i$または$s_i=\langle R,U\rangle$，(2) その他のときは$s_i=r_i$となっている．

$pick_{L,U}([r_1,\ldots,r_n],[s_1,\ldots,s_n],S) \Longleftrightarrow$
(1) ある$r_i=\langle S_1 \,\&\cdots\&\, S_m,L\rangle$について$S_k=S$であり，$s_i=\langle S_1 \,\&\cdots\&\, S_m,U\rangle$かつ$s_j=r_j$ $(j\neq i)$となっている，または，(2) ある$r_i=\langle S_1 \,\&\cdots\&\, S_m,0\rangle$について$S_k=S$であり，$s_j=r_j$ $(1\leq j\leq n)$となっている．

$change_{\ell,\ell'}([r_1,\ldots,r_n],[s_1,\ldots,s_n]) \Longleftrightarrow$
各$i=1,2,\ldots,n$について，(1) $r_i=\langle R,\ell\rangle$のときは$s_i=\langle R,\ell'\rangle$，(2) その他のときは$s_i=r_i$となっている．

$thinable_{\ell}([r_1,\ldots,r_n]) \Longleftrightarrow$
すべての$r_i=\langle R,\ell'\rangle$について，$\ell\neq\ell'$となっている．

体系IOLは体系HLL，すなわち体系IOと同値である．このことを示すために，必要となるいくつかの定義と補題を示しておく．

まず，IOLコンテキストのスライスを次のように定義する．ℓが正整数のとき，IOLコンテキスト$I=[\langle R_1,\ell_1\rangle,\ldots,\langle R_n,\ell_n\rangle]$の**スライス**$I^{(\ell)}$は次の条件を満たすIOコンテキスト$[s_1,\ldots,s_n]$である：各$i=1,2,\ldots,n$について，(1) $\ell_i=0$のとき$s_i=!R_i$，(2) $\ell_i=\ell\neq 0$のとき$s_i=R_i$，(3) それ以外のとき$s_i=1$．消費可能レベルがLのとき，スライス$I^{(L)}$は，I中の消費可能なリソースから成るIOコンテキストを表している．

補題 3.1 Lを正整数，Uを負整数，Gをゴール論理式，Iを(L,U)-コンテキストとする．このときIOL $\vdash_{L,U} I\,\{G\}\,O$ならば，Oは$subcontext_{U,L}(O,I)$を満たす(L,U)-コンテキストである．したがって，レベルがUより大きくL未満のレベル付きリソース論理式はIとOで一致している．

証明 証明図の高さに関する帰納法で証明する．ここでは，規則(!)と規則(&)の場合についてだけ述べる．
規則(!)の場合：仮定よりIは(L,U)-コンテキストであり，レベル$L+1$の論理式は存在していない．また帰納法の仮定より，Oは$(L+1,U)$-コンテキストで$subcontext_{U,L+1}(O,I)$である．したがって，Oにもレベル$L+1$の論理式は存在せず，Oは(L,U)-コンテキストで$subcontext_{U,L}(O,I)$が成り立つ．
規則(&)の場合：仮定よりIは(L,U)-コンテキスト

$$\frac{}{\vdash_{L,U} I\,\{1\}\,I}\ (1) \qquad \frac{\mathit{subcontext}_{U,L}(O,I)}{\vdash_{L,U} I\,\{\top\}\,O}\ (\top)$$

$$\frac{\vdash_{L,U} I\,\{G_1\}\,M \quad \vdash_{L,U} M\,\{G_2\}\,O}{\vdash_{L,U} I\,\{G_1 \otimes G_2\}\,O}\ (\otimes)$$

$$\frac{\vdash_{L,U-1} I\,\{G_1\}\,M \quad \mathit{change}_{U-1,L+1}(M,N) \quad \vdash_{L+1,U} N\,\{G_2\}\,O \quad \mathit{thinable}_{L+1}(O)}{\vdash_{L,U} I\,\{G_1 \,\&\, G_2\}\,O}\ (\&)$$

$$\frac{\vdash_{L,U} I\,\{G_i\}\,O}{\vdash_{L,U} I\,\{G_1 \oplus G_2\}\,O}\ (\oplus_i) \qquad \frac{\vdash_{L+1,U} I\,\{G\}\,O}{\vdash_{L,U} I\,\{!\,G\}\,O}\ (!)$$

$$\frac{\vdash_{L,U} [\langle R,L\rangle \mid I]\,\{G\}\,[\langle R,U\rangle \mid O]}{\vdash_{L,U} I\,\{R \multimap G\}\,O}\ (\multimap) \qquad \frac{\vdash_{L,U} [\langle R,0\rangle \mid I]\,\{G\}\,[\langle R,0\rangle \mid O]}{\vdash_{L,U} I\,\{R \Rightarrow G\}\,O}\ (\Rightarrow)$$

$$\frac{\mathit{pick}_{L,U}(I,O,A)}{\vdash_{L,U} I\,\{A\}\,O}\ (\mathrm{BC}_1) \qquad \frac{\mathit{pick}_{L,U}(I,M,G \multimap A) \quad \vdash_{L,U} M\,\{G\}\,O}{\vdash_{L,U} I\,\{A\}\,O}\ (\mathrm{BC}_2)$$

図8 IOLモデルの体系

であって，帰納法の仮定より，Mは$(L,U-1)$-コンテキストで$\mathit{subcontext}_{U-1,L}(M,I)$である．また，$\mathit{change}$の定義からレベルが$U-1$である$M$の論理式は$N$では$L+1$になり，帰納法の仮定より$O$は$(L+1,U)$-コンテキストで$\mathit{subcontext}_{U,L+1}(O,N)$である．したがって，レベルが$U$より大きく$L$未満のレベル付きリソース論理式は$I$と$O$で一致している．また，$I$中でレベルが$U$の論理式は$O$中でもレベル$U$であることも容易に確かめられる．$I$中でレベルが$L$の論理式は$O$中ではレベルが$L$または$L+1$または$U$となるが，$\mathit{thinable}_{L+1}(O)$であるので，レベルが$L+1$の論理式は存在しない．したがって，$O$は$(L,U)$-コンテキストで，$\mathit{subcontext}_{U,L}(O,I)$が成り立つ． □

定理 3.1 体系IOLは体系HLLと同値である．すなわち，Lを正整数，Uを負整数，Gをゴール論理式，I, Oを$\mathit{subcontext}_{U,L}(O,I)$を満たす任意の$(L,U)$-コンテキストとし，$I^{(L)} - O^{(L)} = \langle \Gamma, \Delta\rangle$であるとする．このとき以下が成立する．

$$\mathrm{IOL} \vdash_{L,U} I\,\{G\}\,O \iff \mathrm{HLL} \vdash \Gamma; \Delta \longrightarrow G$$

証明 ($\Longrightarrow$) IOLの証明図の高さに関する帰納法で証明する．ここでは，規則(&)，規則(!)，規則(BC_2)の場合について示す．

規則(&)の場合: 補題3.1およびchangeと$\mathit{thinable}$の定義より，I中のレベルLのいくつかのリソースはM中ではレベル$U-1$となり，それらはN中ではレベル$L+1$，さらにO中ではレベルUとなる．すなわち，Mは$\mathit{subcontext}_{U-1,L}(M,I)$を満たす$(L,U-1)$-コンテキスト，$N$は$\mathit{subcontext}_{U,L+1}(O,N)$を満たす$(L+1,U)$-コンテキストであり，$I^{(L)} - M^{(L)} = N^{(L+1)} - O^{(L+1)} = I^{(L)} - O^{(L)} = \langle \Gamma, \Delta\rangle$が成り立つ．したがって，帰納法の仮定より$\Gamma; \Delta \longrightarrow G_1$と$\Gamma; \Delta \longrightarrow G_2$がHLLで証明可能だから，$\Gamma; \Delta \longrightarrow G_1 \,\&\, G_2$はHLLで証明可能である．

規則(!)の場合: 補題3.1の適用により，Oは$\mathit{subcontext}_{U,L+1}(O,I)$を満たす．また，$I$, Oにはレベル$L+1$のリソースは存在しないため，$I = O$である．したがって，$I^{(L+1)} - O^{(L+1)} = I^{(L)} - O^{(L)} = \langle \Gamma, \emptyset\rangle$が成り立ち，帰納法の仮定より$\Gamma; \longrightarrow G$がHLLで証明可能だから，$\Gamma; \longrightarrow\, !\,G$はHLLで証明可能である．

規則(BC_2)の場合: 入力コンテキスト$I = [r_1, \cdots, r_n]$とする．$\mathit{pick}_{L,U}(I,M,G \multimap A)$なので，ある$r_i = \langle S_1 \& \cdots \& S_m, \ell\rangle$について$S_k = (G \multimap A)$となっており，$\ell$は$L$または0である．

$\ell = L$の場合，$M^{(L)} - O^{(L)} = \langle \Gamma, \Delta\rangle$とすると$I^{(L)} - O^{(L)} = \langle \Gamma, \Delta \uplus \{S_1 \& \cdots \& S_m\}\rangle$ †5であ

†5 $\uplus$はマルチ集合の和集合である．

り，$G \multimap A \in \|S_1 \& \cdots \& S_m\|$である．帰納法の仮定より$\Gamma; \Delta \longrightarrow G$がHLLで証明可能であるから$\Gamma; \Delta, S_1 \& \cdots \& S_m \longrightarrow A$もHLLで証明可能である．

$\ell = 0$の場合，$M^{(L)} - O^{(L)} = \langle \Gamma, \Delta \rangle$とすると$I^{(L)} - O^{(L)} = \langle \Gamma, \Delta \rangle$，$(S_1 \& \cdots \& S_m) \in \Gamma$，$G \multimap A \in \|S_1 \& \cdots \& S_m\|$である．帰納法の仮定より$\Gamma; \Delta \longrightarrow G$がHLLで証明可能であるから$\Gamma; \Delta \longrightarrow A$もHLLで証明可能である．

($\Longleftarrow$) HLLの証明図の高さに関する帰納法で証明する．ここでは，規則(R$\otimes$)，規則(R&)，規則(R!)，規則(BC_2)の場合についてだけ示す．

規則(R$\otimes$)の場合: I，Oを$I^{(L)} - O^{(L)} = \langle \Gamma, \Delta_1 \uplus \Delta_2 \rangle$かつ$subcontext_{U,L}(O, I)$を満たす任意の$(L, U)$-コンテキストとする．$I$中の$\Delta_1$に対応するリソースのレベルを$L$から$U$に変更したものを$M$とする．$M$は，$I^{(L)} - M^{(L)} = \langle \Gamma, \Delta_1 \rangle$かつ$subcontext_{U,L}(M, I)$を満たす$(L, U)$-コンテキストだから，帰納法の仮定より$\vdash_{L,U} I\{G_1\}M$はIOLで証明可能である．また，同様に$\vdash_{L,U} M\{G_2\}O$もIOLで証明可能であるから，$\vdash_{L,U} I\{G_1 \otimes G_2\}O$はIOLで証明可能である．

規則(R&)の場合: I, Oを$I^{(L)} - O^{(L)} = \langle \Gamma, \Delta \rangle$かつ$subcontext_{U,L}(O, I)$を満たす任意の$(L, U)$-コンテキストとする．$I$中の$\Delta$に対応するリソースのレベルを$L$から$U-1$に変更したものを$M$とする．$M$は，$I^{(L)} - M^{(L)} = \langle \Gamma, \Delta \rangle$かつ$subcontext_{U-1,L}(M, I)$を満たす$(L, U-1)$-コンテキストだから，帰納法の仮定より$\vdash_{L,U-1} I\{G_1\}M$はIOLで証明可能である．$M$中のレベル$U-1$のリソースをレベル$L+1$に変更したもの(すなわち，$I$中の$\Delta$に対応するリソースのレベルを$L$から$L+1$に変更したもの)を$N$とする．$N$は，$N^{(L+1)} - O^{(L+1)} = \langle \Gamma, \Delta \rangle$かつ$subcontext_{U,L+1}(O, N)$を満たす$(L+1, U)$-コンテキストだから，帰納法の仮定より$\vdash_{L+1,U} N\{G_2\}O$はIOLで証明可能である．また，$change_{U-1,L+1}(M, N)$かつ$thinable_{L+1}(O)$も成り立つから，$\vdash_{L,U} I\{G_1 \,\&\, G_2\}O$はIOLで証明可能である．

規則(R!)の場合: I, Oを$I^{(L)} - O^{(L)} = \langle \Gamma, \emptyset \rangle$かつ$subcontext_{U,L}(O, I)$を満たす任意の$(L, U)$-コンテキストとする．$I$, Oは，$I^{(L+1)} - O^{(L+1)} = \langle \Gamma, \emptyset \rangle$かつ$subcontext_{U,L+1}(O, I)$を満たす$(L+1, U)$-コンテキストだから，帰納法の仮定より$\vdash_{L+1,U} I\{G\}O$はIOLで証明可能である．したがって，$\vdash_{L,U} I\{!G\}O$はIOLで証明可能である．

規則(BC_2)の場合: I，Oを$I^{(L)} - O^{(L)} = \langle \Gamma, \Delta \uplus \{R\} \rangle$かつ$subcontext_{U,L}(O, I)$を満たす任意の$(L, U)$-コンテキストとする．$R = (S_1 \& \cdots \& S_m)$とすると，ある$S_k$について，$S_k = (G \multimap A)$である．$I$はレベル付きリソース$\langle R, L \rangle$を含んでいるので，$pick_{L,U}(I, M, G \multimap A)$を満たす$M$が存在し，$M$は$M^{(L)} - O^{(L)} = \langle \Gamma, \Delta \rangle$かつ$subcontext_{U,L}(O, M)$を満たす$(L, U)$-コンテキストになる．したがって，帰納法の仮定より，$\vdash_{L,U} M\{G\}O$はIOLで証明可能であり，$\vdash_{L,U} I\{A\}O$もIOLで証明可能である． □

IOLが効率的なリソース管理を行う計算モデルとして最も重要なのは，「実行中に保持すべきIOLコンテキストをただ一つにできる」という点である．すなわち，IOLの各規則を見ると，下の入力コンテキストが上の入力コンテキストに，左の出力コンテキストが右の入力コンテキストに，上の出力コンテキストが下の出力コンテキストに渡されていることが分かる．したがって，IOLの証明を下から上かつ左から右に探索していった時，単一のIOLコンテキストを保持しているだけで良い．

ただ一つのIOLコンテキストを保持するだけで良いので，コンテキストの実装にハッシュ表を利用することも容易である．IOモデルの場合は，ハッシュ表を用いたとしても，複数のハッシュ表を管理し，それらの間のコピーや比較を実現する必要があり，実装(特にメモリ管理)は複雑になる．

ただし，規則(&)については，IOLモデルの場合でも，全リソースを調べる必要があり，メモリ使用量は小さくなるが，実行時間はIOモデルの場合と同じく，リソースの個数に比例する時間がかかる．

表1 N-クイーンプログラムの実行結果

N-Queens	Prolog (msec)	LLP (msec)	速度向上比
8 (全解)	164	50	3.28
9 (全解)	780	226	3.45
10 (全解)	3810	1006	3.79
11 (全解)	20426	4912	4.16
12 (全解)	113938	25754	4.42
13 (全解)	680984	143234	4.75
15 (第一解)	290	50	5.80
20 (第一解)	63308	8322	7.61
25 (第一解)	22453	2340	9.60

表2 ナイト・ツアー (5×5) の実行結果

	Prolog (msec)	LLP (msec)	速度向上比
Knight Tour	241404	89042	2.71

表3 Prologベンチマークの実行結果

プログラム名	SICStus (msec)	LLP (msec)	SWI (msec)
`browse`	770	1510	2352
`cal`	142	311	645
`chat_parser`	29	40	63
`ham`	580	1492	1286
`poly_10`	43	60	145
`queens_8` (all)	56	100	288
`queens_10` (all)	1271	2320	6978
`queens_16` (one)	777	1460	4567
`zebra`	39	68	66
実行時間比の平均	1.00	1.86	3.73

4 処理系について

レベル付きIOモデルの体系IOLに基づいたLolliのインタープリタ処理系を図9に示す(Prologで記述してある). ただし，リソース論理式およびゴール論理式中に現れる限量子には対応していない(すなわち，`forall(`X`,`S`)`, `exists(`X`,`G`)` などの記法は使用できない).

もちろんこの処理系は，IOLコンテキストがリストで表現されているため，IOモデルの問題点(*pick* におけるIOコンテキストの探索の問題，*pick* におけるIOコンテキストの再構築の問題)を解決したことにはなっていない.

著者らのLLPコンパイラ処理系[4][17][22][26][28]は，Prologコンパイラで広く用いられている抽象機械であるWAM [1][24]を拡張し，リソースを保持するためのリソース表と，高速にリソースを探索するためのハッシュ表を付け加えた抽象機械LLPAMを用いている.

LLPAMの命令セットは，本論文で述べたレベル付きIOモデルIOLに基づいて設計されており，実行中はただ一つのリソース表を保持し，そのレベル値を書き換えることで計算が進んで行く．また，リソース論理式中の限量子にも対応している.

そこで，本論文では，レベル付きIOモデルのリソース管理方式の有効性を示すために，2.4節で述べたN-クイーンのプログラムをベンチマークとして，実行時間を計測した．

一つ目は，図4のPrologで記述したN-クイーンのプログラムについて，SICStus Prologコンパイラ処理系上(バージョン3.7.1, コンパクトコード)での実行速度を計測した．この場合，リソースはリストで表現されており，もともとのIOモデルでの実装に対応するとみなせる．

二つ目は，図3のLolliで記述したN-クイーンのプログラムについて，LLPコンパイラ処理系上(バージョン0.43)での実行速度を計測した．この場合，リソースはLLPAMのリソース表に登録され，ハッシュ表で検索される．

表1に，これら二つのプログラムの実行結果を示す．なお，計測はCPUがMMX Pentium 266MHz, メモリが128Mバイト，OSがLinuxの計算機上で行った．Nが8以上13以下については全解探索の実行時間(単位はミリ秒)，Nが15, 20, 25については第一解探索までの実行時間(単位はミリ秒)を示している．Nが8の場合ですでに3倍以上の速度であり，Nが大きくなるにつれて速度向上比も大きくなっている．

また，5×5のチェス盤上でのナイト・ツアー問題(ハミルトン経路探索の一種で，ナイトの駒が全てのマスをちょうど一度ずつ訪れる経路を探索する問題) についても実行時間を計測した．表2は，5×5のチェス盤上での全解探索の実行時間(単位はミリ秒)を示しており，LLPの実行速度は，SICStus Prologと比較して，2.71倍速くなっている．

また，LLPAMの拡張によるオーバーヘッドを計測するために，標準的なPrologプログラムのベンチマークも実行し，既存のPrologコンパイラ処理系との性能比較も行った(表3参照)．商用であるSICStus Prolog (コンパクトコード) と比較すると，1.86倍遅い程度に抑えられており，また広く利用されているフリーのコンパイラ処理系のSWI-Prologと比較して，約2倍速くなっている．

5 おわりに

本論文では，直観主義線形論理型言語Lolliに対して，効率的なリソース管理を行うための計算モデルであるレベル付きIOモデルについて述べた．

レベル付きIOモデルを採用すれば，実行中はただ一つのコンテキスト(リソース表)を保持すれば良く，その実装にハッシュ表を用いるなどすることで，非常に効率良い処理系の実現が可能になる．

実際，レベル付きIOモデルを採用した著者らのLLPコンパイラ処理系で計測した所，8-クイーンの全解探索について3倍以上，25-クイーンの第一解探索について10倍近い速度向上を得，レベル付きIOモデルの有効性を実証できた．

なお，同一のリソース管理方式は，IOモデルを利用した他の線形論理型言語に対しても適用可能であると考えている．たとえば，MillerのForum言語についても，IOモデルを利用した計算モデルがHodasらによって提案されているが，本論文での方法をそのまま適用可能である．

最後に，ゴール⊤の取り扱いについて述べる．ゴール⊤は，利用可能なリソースのうちいくつかを暗黙的に消費するという意味を持っている．IOモデルおよびレベル付きIOモデルでは，これをそのまま実装しており($subcontext$および$subcontext_{U,L}$によって)，非決定性が大きい．したがって，Cervesatoらの論文[7]の手法をレベル付きIOモデルに適用して，ゴール⊤の実行を遅延的に行う方法が考えられるが，これについては，Hodasらとの共同研究[17]で解決済みである．

参考文献

[1] Aït-Kaci, H.: *Warren's Abstract Machine*, The MIT Press, 1991. (`http://www.isg.sfu.ca/~hak/documents/wam.html`).

[2] Andreoli, J.-M.: Logic Programming with Focusing Proofs in Linear Logic, *Journal of Logic and Computation*, Vol. 2,No. 3(1992), pp. 297–347.

[3] Andreoli, J.-M. and Pareschi, R.: Linear Objects: Logical Processes with Built-In Inheritance, *New Generation Computing*, Vol. 9(1991), pp. 445–473.

[4] Banbara, M. and Tamura, N.: Compiling Resources in a Linear Logic Programming Language, *Proceedings of Post-JICSLP'98 Workshop on Par-*

```
:- op(1060, xfy, (&)).
:- op( 950, xfy, (-<>)).
:- op( 950, xfy, (=>)).
:- op( 900,  fy, (!)).

prove(G) :- I = [], O = [], L = 1, U = -1, prove(G, I, O, L, U).

prove(true,      I, I, L, U) :- !.
prove(erase,     I, O, L, U) :- !, subcontext(O, I, U, L).
prove((G1,G2),   I, O, L, U) :- !, prove(G1, I, M, L, U), prove(G2, M, O, L, U).
prove((G1&G2),   I, O, L, U) :- !, L1 is L+1, U1 is U-1,  prove(G1, I, M, L, U1),
                                   change(M, N, U1, L1),  prove(G2, N, O, L1, U),
                                   thinable(O, L1).
prove((G1;G2),   I, O, L, U) :- !, (prove(G1, I, O, L, U) ; prove(G2, I, O, L, U)).
prove(!G,        I, O, L, U) :- !, L1 is L+1, U1 is U-1, prove(G, I, O, L1, U1).
prove((R -<> G), I, O, L, U) :- !, prove(G, [(R,L)|I], [(R,U)|O], L, U).
prove((R => G),  I, O, L, U) :- !, prove(G, [(R,0)|I], [(R,0)|O], L, U).
prove(A,         I, O, L, U) :- pick(I, O, A, L, U). % A is an atomic formula
prove(A,         I, O, L, U) :- pick(I, M, (G -<> A), L, U), % A is an atomic formula
                                prove(G, M, O, L, U).

subcontext([],        [],        U, L).
subcontext([(R,U)|O], [(R,L)|I], U, L) :- subcontext(O, I, U, L).
subcontext([X|O],     [X|I],     U, L) :- subcontext(O, I, U, L).

change([],         [],         L1, L2) :- !.
change([(R,L1)|I], [(R,L2)|O], L1, L2) :- !, change(I, O, L1, L2).
change([(R,L)|I],  [(R,L)|O],  L1, L2) :- L=\=L1, !, change(I, O, L1, L2).

thinable([],         L) :- !.
thinable([(R,L1)|I], L) :- L=\=L1, !, thinable(I, L).

pick(I,         I,         S, L, U) :- rule(S).
pick([(R,0)|I], [(R,0)|I], S, L, U) :- pick_S(R, S).
pick([(R,L)|I], [(R,U)|I], S, L, U) :- pick_S(R, S).
pick([X|I],     [X|O],     S, L, U) :- pick(I, O, S, L, U).

pick_S((R1&R2), S) :- !, (pick_S(R1, S); pick_S(R2, S)).
pick_S(S,       S).

rule( append([], Zs, Zs) ).
rule(( append(Xs, Ys, Zs) -<> append([X|Xs], Ys, [X|Zs]) )).
```

図9 IOLモデルに基づいたLolliインタプリタ

allelism and Implementation Technology for Logic Programming Languages, June 1998, pp. 32–45.

[5] Banbara, M. and Tamura, N.: Translating a Linear Logic Programming Language into Java, *Proceedings of ICLP'99 Workshop on Parallelism and Implementation Technology for (Constraint) Logic Programming Languages*, November 1999, pp. 19–39.

[6] Bratko, I.: *Prolog programming for artificial intelligence*, Addison-Wesley, 1986.

[7] Cervesato, I., Hodas, J. S., and Pfenning, F.: Efficient resource management for linear logic proof search, *Proceedings of the 1996 International Workshop on Extensions of Logic Programming*, Springer-Verlag LNAI 1050, March 1996, pp. 67–81.

[8] Cervesato, I., Hodas, J. S., and Pfenning, F.: Efficient Resource Management for Linear Logic Proof Search, *Proceedings of the Fifth International Workshop on Extensions of Logic Programming — ELP'96*(Dyckhoff, R., Herre, H., and Schroeder-Heister, P.(eds.)), Leipzig, Germany,

Springer-Verlag LNAI 1050, 28–30 March 1996, pp. 67–81.

[9] Girard, J.-Y.: Linear Logic, *Theoretical Computer Science*, Vol. 50(1987), pp. 1–102.

[10] Girard, J.-Y.: Linear Logic: Its Syntax and Semantics, *Advances in Linear Logic*(Girard, J.-Y., Lafont, Y., and Regnier, L.(eds.)), Cambridge University Press, 1995, pp. 1–42. Proceedings of the Workshop on Linear Logic, Ithaca, New York, June 1993.

[11] Harland, J. and Pym, D.: A Uniform Proof-Theoretic Investigation of Linear Logic Programming, *Journal of Logic and Computation*, Vol. 4,No. 2(1994), pp. 175–207.

[12] Harland, J. and Winikoff, M.: Implementing the Linear Logic Programming Language Lygon, *Proceedings of the 1995 International Logic Programming Symposium*(Lloyd, J.(ed.)), Portland, Oregon, 1995, pp. 66–80.

[13] Hodas, J. S.: Lolli: An Extension of λProlog with Linear Context Management, *Workshop on the λProlog Programming Language*(Miller, D.(ed.)), Philadelphia, Pennsylvania, August 1992, pp. 159–168.

[14] Hodas, J. S.: Specifying Filler-Gap Dependency Parsers in a Linear-Logic Programming Language, *Proceedings of the Joint International Conference and Symposium on Logic Programming*(Apt, K.(ed.)), Washington, DC, November 1992, pp. 622–636.

[15] Hodas, J. S.: *Logic Programming in Intuitionistic Linear Logic: Theory, Design and Implementation*, PhD Thesis, University of Pennsylvania, Department of Computer and Information Science, 1994.

[16] Hodas, J. S. and Miller, D.: Logic Programming in a Fragment of Intuitionistic Linear Logic, *Information and Computation*, Vol. 110,No. 2(1994), pp. 327–365. Extended abstract in the Proceedings of the Sixth Annual Symposium on Logic in Computer Science, Amsterdam, July 15–18, 1991.

[17] Hodas, J. S., Watkins, K., Tamura, N., and Kang, K.-S.: Efficient Implementation of a Linear Logic Programming Language, *Proceedings of the 1998 International Conference and Symposium on Logic Programming*, June 1998, pp. 145–159.

[18] Kobayashi, N. and Yonezawa, A.: ACL — A Concurrent Linear Logic Programming Paradigm, *Proceedings of the 1993 International Logic Programming Symposium*(Miller, D.(ed.)), Vancouver, Canada, MIT Press, October 1993, pp. 279–294.

[19] Miller, D., Nadathur, G., Pfenning, F., and Scedrov, A.: Uniform proofs as a foundation for logic programming, *Annals of Pure and Applied Logic*, Vol. 51(1991), pp. 125–157.

[20] Miller, D.: A Multiple-Conclusion Specification Logic, *Theoretical Computer Science*, Vol. 165,No. 1(1996), pp. 201–232.

[21] Nadathur, G. and Miller, D.: An Overview of λProlog, *Logic Programming: Proceedings of the Fifth International Conference and Symposium, Volume 1*(Kowalski, R. A. and Bowen, K. A.(eds.)), Cambridge, Massachusetts, MIT Press, August 1988, pp. 810–827.

[22] Tamura, N. and Kaneda, Y.: Extension of WAM for a linear logic programming language, *Second Fuji International Workshop on Functional and Logic Programming*(Ida, T., Ohori, A., and Takeichi, M.(eds.)), World Scientific, Nov. 1996, pp. 33–50.

[23] Tamura, N. and Kaneda, Y.: A Compiler System of a Linear Logic Programming Language, *Proceedings of the IASTED International Conference Artificial Intelligence and Soft Computing*, July 1997, pp. 180–183.

[24] Warren, D. H. D.: An abstract Prolog instruction set, Technical Report Technical Note 309, SRI International, Menlo Park, CA, Oct. 1983.

[25] 竹内外史: **線形論理入門**, 日本評論社, 1995.

[26] 田村直之, 池田雄一: 線形論理型言語のコンパイラ処理系でのリソース管理方式について, **情報処理学会 プログラミング研究会報告** *No. 7*, 5月 1996, pp. 25–30.

[27] 田村直之, 平井崇晴, 吉川英男, 姜京順, 番原睦則: 直観主義時相線形論理における論理プログラミングについて, **情報処理学会論文誌: プログラミング**, Vol. 41,No. SIG 4 (PRO 7)(2000), pp. 11–23.

[28] 番原睦則, 姜京順, 田村直之: 線形論理型言語の Java 言語による処理系の設計と実装, **情報処理学会論文誌: プログラミング**, Vol. 40,No. SIG 10 (PRO 5)(1999), pp. 1–16.