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

田村直之[†] 平井崇晴[††] 吉川英男[††]
姜京順[††] 番原睦則[†††]

1987 年に J.-Y. Girard によって考え出された線形論理は，環境が動的に変化するソフトウェアを表現するための論理として非常に有望である．しかしながら，線形論理では「動的に変化する環境」を表現することはできるが,「時間経過とともに動的に変化する環境」を表現しきれているとは言えない．これは線形論理には時間の概念が直接的には入っていないからである．そこで，本論文では，線形論理と時相論理の特徴を融合した時相線形論理の体系について述べ，さらにこの体系に基づいた論理型プログラミング言語について述べる．この時相線形論理の体系は，著者らの一人によって考えられたものであり，直観主義線形論理の演算子に加え，次の時刻に一回だけ利用可能な資源を表す様相演算子 ○，現在以降の任意の時刻に一回だけ利用可能な資源を表す様相演算子 □，現在以降の任意の時刻に任意の回数利用可能な資源を表す様相演算子 ! を含んでいる．本論文では，Miller のユニフォーム証明の考えを適用することによって，この時相線形論理の体系に対する論理型言語を設計し，Hodas の IO モデルの考えを適用することによって，効率的な計算モデルを与える．

Logic Programming in an Intuitionistic Temporal Linear Logic

Naoyuki Tamura,[†] Takaharu Hirai,[††] Hideo Yoshikawa,[††]
Kyoung-Sun Kang[††] and Mutsunori Banbara[†††]

Linear logic developed by J.-Y. Girard in 1987 is very useful to describe the software in which the environment changes dynamically. Though it can represent “dynamically changing environment”, it can not naturally represent “time-dependently changing environment” because it lacks the concept of time. In this paper, we describe a system of temporal linear logic which integrates the features of linear logic and temporal logic, and a logic programming language based on the temporal linear logic system. This system, developed by one of the authors, has a modal operator ○ which represents a resource usable only once at the next clock, a modal operator □ which represents a resource usable only once at any clock of now and after, and a modal operator ! which represents a resource usable any times at any clock of now and after. In this paper, a logic programming language based on this temporal linear logic is designed by using the idea of Miller’s uniform proof, and its efficient computation model is given by using the idea of Hodas’s IO model.

1. はじめに

1987 年に J.-Y. Girard が発表した線形論理 [5] 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．

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[2], ACL[14], Lolli[11)4)], Lygon[6)7)], Forum[15], および著者らの LLP[3),12),17),19),20)] 等の研究がある．

しかしながら，線形論理では「動的に変化する環境」を表現することはできるが,「時間経過とともに動的に変化する環境」を表現しきれているとは言えない．こ

† 神戸大学工学部
Faculty of Engineering, Kobe University
†† 神戸大学大学院自然科学研究科
Graduate School of Science and Technology, Kobe University
††† 奈良工業高等専門学校
Nara National College of Technology

れは線形論理には時間の概念が直接的には入っていないからであり，線形論理に時間の概念を導入した時相線形論理の研究も始まってきている [8),13),18)]．

そこで，本論文では，そのような時相線形論理に基づいた論理型プログラミング言語である TLLP の設計と実装について述べる．

時相線形論理型言語 TLLP は，平井による時相線形論理 ITLL[8)] に基づいているので，まず，ITLL の体系と特徴について述べ，次に，Miller らのユニフォーム証明 [16)] の考え方をこの時相線形論理 ITLL に適用し，時相線形論理型言語 TLLP を設計する．

次に，Hodas の IO モデルの考えを適用することによって，TLLP の効率的な計算モデルを与える．

最後に，TLLP 処理系の実装方法について考察する．

2. 時相線形論理の体系 ITLL

線形論理と時相論理とを融合した論理体系については，Kanovich と伊藤による体系 [13)] や田辺による体系 [18)] が発表されている．

しかし，どちらも線形論理の論理結合子の ! を含んでおらず，カット除去定理も成立していないため，後述のユニフォーム証明の手法で論理型言語を設計するために利用することはできない．

一方，**図 2** に示す平井の直観主義時相線形論理の体系 ITLL[8)] は，直観主義線形論理 ILL(**図 1** 参照) の自然な拡張になっており，カット除去定理も成立する．したがって，Miller らのユニフォーム証明の手法を適用して，論理型言語を設計することが可能である．なお，図中で，シーケントの左辺は論理式のマルチ集合であり，Exchange 規則を暗黙のうちに仮定している．

ITLL は，ILL に (L □), (R □), (○) の三つの規則を付け加えた体系である．したがって，ILL で証明可能なシーケントはすべて ITLL でも証明可能である．

さらに，ITLL は「次の時刻」を意味する演算子 ○ と「現在時刻以降いつでも」を意味する演算子 □ を含んでいる．たとえば，論理式 A, $\bigcirc A$, $\Box A$, $!A$ などを資源として解釈した場合，それぞれの直観的な意味は次のようになる．

- A: 現在時刻に一度だけ利用できる資源
- $\bigcirc A$: 次の時刻に一度だけ利用できる資源
- $\Box A$: 現在時刻以降に一度だけ利用できる資源
- $!A$: 現在時刻以降，いつでも何度でも利用できる資源

また，A, $\bigcirc A$, $\Box A$, $!A$ などの間には，以下のような関係が成り立っている．

$$!A \longleftrightarrow !!A$$
$$!A \longrightarrow \Box A \otimes \cdots \otimes \Box A$$
$$\Box A \longleftrightarrow \Box\Box A$$
$$\Box A \longrightarrow \bigcirc^n A \qquad (n \geq 0)$$

さらに，他の線形論理の論理結合子との組み合わせれば，より様々な表現が可能である．

- $A \,\&\, \bigcirc A$: 現在または次の時刻に一度だけ利用できる資源
- $\bigcirc(1 \,\&\, A)$: 次の時刻に高々一度利用できる資源
- $\bigcirc^n \Box A$: 現在より n 時刻後以降に一度だけ利用できる資源
- $\bigcirc^n !A$: 現在より n 時刻後以降は，いつでも何度でも利用できる資源

3. 時相線形論理とユニフォーム証明

論理型言語 Prolog は，ホーン節に対する SLD 導出をその計算モデルとしている．しかし，線形論理および時相線形論理の場合は連言と選言が二通りずつあり，分配律が一般には成立しない．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，時相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[9)~11)] は，Miller らのユニフォーム証明 [16)] 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．

以下では，ユニフォーム証明の考え方について述べたあと，時相線形論理への適用について述べる．

3.1 ユニフォーム証明

Miller らのユニフォーム証明 [16)] 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．

  すなわち，**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

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

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

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

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

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

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

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

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

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

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

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

(y is not free in the lower sequent)

図 1 直観主義線形論理の体系 ILL
Fig. 1 System ILL for intuitionistic linear logic.

(Rules of ILL)

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

$$\frac{!\,\Gamma, \Box\,\Sigma, \Delta \longrightarrow C}{!\,\Gamma, \Box\,\Sigma, \bigcirc\,\Delta \longrightarrow \bigcirc\, C}\ (\bigcirc)$$

図 2 直観主義時相線形論理の体系 ITLL
Fig. 2 System ITLL for intuitionistic temporal linear logic.

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．

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 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．

3.2 線形論理でのユニフォーム証明

Hodas と Miller の Lolli 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ものである．

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，! の自由な使用を制限した．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に変更した．

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

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，Γ 中の論理式は 0 回以上何度でも利用できる仮定，Δ 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．

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 $\otimes$ 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，利用する論理式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，線形論理型言語 Lolli を設計した．

LLP[3),12),17),19),20)] も，Lolli の方針と同様にして設計された言語であり，Lolli のサブセットになっている．

以下に LLP のフラグメントを示す (A は原子論理式を表す)．

$$\begin{aligned} R &::= A \mid R_1 \,\&\, R_2 \mid G \multimap R \mid G \Rightarrow R \mid \forall x.R \\ G &::= 1 \mid \top \mid A \mid G_1 \otimes G_2 \mid G_1 \,\&\, G_2 \mid G_1 \oplus G_2 \mid \\ &\quad R \multimap G \mid R \Rightarrow G \mid !\,G \end{aligned}$$

ここで，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．

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，本論文では次のように変更した定義を用いる (A は原子論理式，$m \geq 1$)．

$$\begin{aligned} R &::= S_1 \,\&\, \cdots \,\&\, S_m \\ S &::= A \mid G \multimap A \mid \forall x.S \\ G &::= 1 \mid \top \mid A \mid G_1 \otimes G_2 \mid G_1 \,\&\, G_2 \mid G_1 \oplus G_2 \mid \\ &\quad R \multimap G \mid S \Rightarrow G \mid !\,G \end{aligned}$$

S は，Prolog の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)．

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

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で良い．

図 3 に LLP のフラグメントのための体系 HLL を示す☆．ここで，BC および BC!規則中の $\|R\|$ は，以下のように再帰的に定義される論理式の集合を表す (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，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 1 (Hodas and Miller) LLP のフラグメントに対して，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 をリソース節の集合，Δ をリソース論理式のマルチ集合とするとき以下が成

☆ 論文 11) 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．

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

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

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

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

$$\frac{}{\Gamma;R\longrightarrow A}\ (\mathrm{BC}_1)$$
(provided A is atomic and $A\in\|R\|$)

$$\frac{\Gamma;\Delta\longrightarrow G}{\Gamma;\Delta,R\longrightarrow A}\ (\mathrm{BC}_2)$$
(provided A is atomic and $G\multimap A\in\|R\|$)

$$\frac{}{\Gamma,S;\longrightarrow A}\ (\mathrm{BC!}_1)$$
(provided A is atomic and $A\in\|S\|$)

$$\frac{\Gamma,S;\Delta\longrightarrow G}{\Gamma,S;\Delta\longrightarrow A}\ (\mathrm{BC!}_2)$$
(provided A is atomic and $G\multimap A\in\|S\|$)

図 3 直観主義線形論理の体系 HLL
Fig. 3 System HLL for a fragment of intuitionistic linear logic.

立する．

$\mathrm{HLL}\vdash\Gamma;\Delta\longrightarrow G\Longleftrightarrow\mathrm{ILL}\vdash !\Gamma^*,\Delta^*\longrightarrow G^*$ 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 [10] を参照． □

3.3 時相線形論理でのユニフォーム証明

直観主義時相線形論理 ITLL に対しても，直観主義線形論理の場合と同様にユニフォーム証明の考えを適用できる．すなわち，ITLL に対しては，以下のようにフラグメントを選べば，ユニフォーム証明で完全となる (A は原子論理式, $m\geq 1$)．

$$R ::= S_1\,\&\cdots\&\,S_m \mid \Box(S_1\,\&\cdots\&\,S_m) \mid \bigcirc R$$
$$S ::= A \mid G\multimap A \mid \forall x.S$$
$$G ::= 1 \mid \top \mid A \mid G_1\otimes G_2 \mid G_1\,\&\,G_2 \mid G_1\oplus G_2 \mid R\multimap G \mid S\Rightarrow G \mid !\,G \mid \bigcirc G$$

したがって，この定義によって時相線形論理型言語 TLLP の構文を定める．

図 4 に TLLP のための体系 HTLL を示す．ただし，$\|R\|$ の定義に以下を追加する．

(5) $R=\Box R'$ の時，$\|R\|=\|R'\|$.
(6) $R=\bigcirc R'$ の時，$\|R\|=\emptyset$.

体系 HTLL について以下が成り立つ．

命題 2 TLLP のフラグメントに対して，体系 HTLL は体系 ITLL と同値である．すなわち，G をゴール論理式，Γ をリソース節の集合，Δ をリソース論理式のマルチ集合とするとき以下が成立する．

$$\mathrm{HTLL}\vdash\Gamma;\Delta\longrightarrow G\Longleftrightarrow \mathrm{ITLL}\vdash !\Gamma^*,\Delta^*\longrightarrow G^*$$

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 命題 1 と同様にして証明できる． □

TLLP を LLP と比較すると，$\bigcirc^n(S_1\,\&\cdots\&\,S_m)$ と $\bigcirc^n\Box(S_1\,\&\cdots\&\,S_m)$ の形のリソース論理式，および $\bigcirc G$ の形のゴール論理式が追加されている．

すなわち，n 時刻後に一度だけ利用できる資源，n 時刻以降に一度だけ利用できる資源を表現でき，また時刻を一つ進めた上でのゴールの実行が可能になっている．

資源としては単なる事実 (原子論理式) だけでなく，規則 (リソース節) も記述できるから，TLLP は時間とともに変化する環境を表現するための，最小限の記述性は備えていると考えられる．なお，具体的に記述できるプログラム例については 5 章で述べる．

4. 時相線形論理型言語 TLLP の計算モデル

本節では LLP の計算モデルである IO モデルについて述べたあと，TLLP への適用について述べる．

4.1 LLP の 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$\otimes$ 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．たとえば，$\Gamma;\Delta\longrightarrow G_1\otimes G_2$ 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

(Rules of HLL)

$$\frac{\Gamma; \Box\,\Sigma, \Delta \longrightarrow G}{\Gamma; \Box\,\Sigma, \bigcirc\,\Delta \longrightarrow \bigcirc\, G}\ (\bigcirc)$$

図 4 直観主義時相線形論理の体系 HTLL
Fig. 4 System HTLL for a fragment of intuitionistic temporal linear logic.

になるように，マルチ集合 Δ を二つのマルチ集合 Δ_1 と Δ_2 に分割しなければならない．この分割の仕方は，Δ 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，別の消費方法を探す．

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限量子の取り扱いとは無関係に議論できるので，この章での議論は簡単のため，対象とする論理を命題論理とする．

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する．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)，またはリソース節に ! 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節に対応する．

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．

ここで，$subcontext(O, I)$ は，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であり，以下のように定義される．

$subcontext([s_1, \ldots, s_n], [r_1, \ldots, r_n])$
$\Longleftrightarrow$ 各 $i = 1, 2, \ldots, n$ について，(1) r_i が有界リソース $S_1 \,\&\, \cdots \,\&\, S_m$ のときは $s_i = r_i$ または $s_i = 1$，(2) それ以外のときは $s_i = r_i$ となっている．

また述語 $pick(I, O, S)$ は，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．

$pick([r_1, \ldots, r_n], [s_1, \ldots, s_n], S)$
$\Longleftrightarrow$ (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 $r_i = !\,S'$ について $S' = S$ であり，$s_j = r_j$ $(1 \leq j \leq m)$ となっている．

体系 IO と HLL は同値である．IO と HLL の同値性を述べるために，差 $I - O$ を，$subcontext(O, I)$ を満たす 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，Γ は $!S$ 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節 S の集合，Δ 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．

なお，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．

命題 3 (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．

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

証明 Hodas の論文 [10] を参照． □

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

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

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

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

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

図 5 体系 IO: 命題 LLP に対する IO モデル
Fig. 5 System IO: IO-model for propositional LLP.

このIOモデルに従った処理系を作成すれば，リソースの分割を遅延的に行うという点で，単純にHLL上でユニフォーム証明探索を行う処理系よりも，はるかに効率的である．

4.2 時相線形論理 TLLP に対する IO モデル

前章で定義した時相線形論理型言語 TLLP に対しても，LLP の場合と同様に IO モデルを定義できる．

しかし，LLP の IO モデルと同様の IO コンテキストをそのまま用いたのでは非常に効率が悪くなる．

たとえば，入力コンテキスト $I = [p, \bigcirc q, \bigcirc\bigcirc r]$ のもとでゴール $\bigcirc G$ を実行する場合を考える．この場合，ゴール G のための入力コンテキスト $I' = [1, q, \bigcirc r]$ を I から作成する必要がある．このような IO コンテキストの書き換えは非常に効率が悪い．

そこで，IO コンテキスト中の各リソースに非負整数を付与し，消費できる時刻を表す方法を取る．このように拡張した IO コンテキストを IOT コンテキストと呼ぶ．すなわち，**IOT コンテキスト**とはリスト $[r_1, r_2, \ldots, r_n]$ で，各 r_i は $\langle S_1\,\&\,\ldots\,\&\,S_m, t\rangle$ または $\langle \Box(S_1\,\&\,\ldots\,\&\,S_m), t\rangle$ または $\langle !\,S, 0\rangle$ または 1 である (t は非負整数)．

このように表した時，IOT コンテキスト中で以下の形のリソースが，時刻 t の時点で利用可能である．

- $\langle S_1\,\&\,\ldots\,\&\,S_m, t\rangle$
- $\langle \Box(S_1\,\&\,\ldots\,\&\,S_m), t'\rangle$ (ただし $t' \le t$)
- $\langle !\,S, 0\rangle$

したがって，時刻 t 以降で利用可能なリソースは以下の形のものである．

- $\langle S_1\,\&\,\ldots\,\&\,S_m, t'\rangle$ (ただし $t' \ge t$)
- $\langle \Box(S_1\,\&\,\ldots\,\&\,S_m), t'\rangle$ (t' は任意)
- $\langle !\,S, 0\rangle$

また，式 $I\,\{G\}\,O$ にも現在時刻を表す非負整数 t を付加し，以下のように記述する．

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

このように IO モデルを変更した体系 IOT を**図 6** に示す．

ここで，述語 $subcontext_t(O, I)$ および $pick_t(I, O, S)$ は以下のように定義される述語である．

$subcontext_t([s_1, \ldots, s_n], [r_1, \ldots, r_n])$
$\Longleftrightarrow$ 各 $i = 1, 2, \ldots, n$ について，(1) r_i が $\langle S_1\,\&\,\cdots\,\&\,S_m,\ t'\rangle$ (ただし $t' \ge t$) または $\langle \Box(S_1\,\&\,\cdots\,\&\,S_m),\ t'\rangle$ (t' は任意) のときは，$s_i = r_i$ または $s_i = 1$，(2) それ以外のときは $s_i = r_i$ となっている．

$pick_t([r_1, \ldots, r_n], [s_1, \ldots, s_n], S)$
$\Longleftrightarrow$ (1) ある有界リソース $r_i = \langle S_1\,\&\,\cdots\,\&\,S_m,\ t'\rangle$ (ただし $t' = t$) あるいは $r_i = \langle \Box(S_1\,\&\,\cdots\,\&\,S_m),\ t'\rangle$ (ただし $t' \le t$) について $S_k = S$ であり，$s_i = 1$ かつ $s_j = r_j$ ($j \ne i$) となっている，または，(2) ある無限リソース $r_i = \langle !\,S',\ 0\rangle$ について $S' = S$ であり，$s_j = r_j$ ($1 \le j \le m$) となっている．

体系 IOT と HTLL は同値である．IOT と HTLL の同値性を述べるために，IO の場合と同様に，IOT コンテキストの差 $I -_t O$ を，$subcontext_t(O, I)$ を満

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

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

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

$$\frac{[\langle R, t+n\rangle \mid I]\,\{G\}_t\,[1 \mid O]}{I\,\{\bigcirc^n R \multimap G\}_t\,O}\ (\multimap)$$
(provided $R = (S_1 \,\&\, \cdots \,\&\, S_m)$ or $R = \Box(S_1 \,\&\, \cdots \,\&\, S_m)$)

$$\frac{[\langle !\,S, 0\rangle \mid I]\,\{G\}_t\,[\langle !\,S, 0\rangle \mid O]}{I\,\{S \Rightarrow G\}_t\,O}\ (\Rightarrow) \qquad \frac{I\,\{G\}_{t+1}\,O}{I\,\{\bigcirc G\}_t\,O}\ (\bigcirc)$$

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

図 6 体系 IOT: 命題 TLLP に対する IO モデル
Fig. 6 System IOT: IO-model for propositional TLLP.

たす IOT コンテキストに対して定義する (t は非負整数)．IOT コンテキストの差 $I -_t O$ は組 $\langle \Gamma, \Delta\rangle$ であり，Γ は $\langle !\,S, 0\rangle$ 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節 S の集合，Δ は $\langle R, t'\rangle$ が I の要素になっており (ただし R が $S_1 \,\&\, \cdots \,\&\, S_m$ のときは $t' \geq t$，$\Box(S_1 \,\&\, \cdots \,\&\, S_m)$ のときは t' は任意)，O では 1 になっているリソース論理式 R に対して $\max(0, t'-t)$ 個の $\bigcirc$ を付け加えたリソース論理式 (すなわち $\bigcirc^{\max(0,t'-t)} R$) のマルチ集合である．

なお，体系 IOT で $I\,\{G\}_t\,O$ が証明可能なとき，常に $subcontext_t(O, I)$ が成り立っていることに注意されたい．

命題 4 体系 IOT は体系 HTLL と同値である．すなわち，t を非負整数，G をゴール論理式，I, O を $subcontext_t(O, I)$ を満たす IOT コンテキストとし，$I -_t O = \langle \Gamma, \Delta\rangle$ であるとする．このとき以下が成立する．

$$\mathrm{IOT} \vdash I\,\{G\}_t\,O \Longleftrightarrow \mathrm{HTLL} \vdash \Gamma; \Delta \longrightarrow G$$

証明 命題 3 と同様にして証明できる． □

5. 時相線形論理型言語 TLLP

5.1 TLLP の構文

Prolog および LLP の記法に従い，時相線形論理型言語 TLLP のプログラム中での記法を以下のように定める (A は原子論理式, $m \geq 1$).

C ::= A. | A:-G.
R ::= S_1&$\cdots$&S_m | #(S_1&$\cdots$&S_m) | @R
S ::= A | G-<>A | forall X\S
G ::= true | top | A | G_1,G_2 | G_1&G_2 | G_1;G_2 | R-<>G | S=>G | !G | @G

ここで，C は Prolog と同様のプログラム節を表す．プログラム節は，それぞれ $\forall\vec{x}.A$ および $\forall\vec{x}.(G \multimap A)$ という閉じたリソース節を意味し，無限コンテキスト中に置かれる．その他は，第 3 章の TLLP の定義に対応する．たとえば，論理式 $\bigcirc((\Box R) \multimap G)$ はプログラム中では @((#R)-<>G) と記述する．

ゴールとして，true, A, G_1,G_2, G_1;G_2 だけを用いている TLLP プログラムは，構文的にも意味的にも Prolog と完全に一致する．

また，演算子#も@も使用していない TLLP プログラムは，構文的にも意味的にも LLP と完全に一致する．

5.2 TLLP のプログラム例

以下は，a, b, c, d の 4 頂点からなる完全グラフにおいて，頂点 a から頂点 d へのハミルトン経路 (すべての頂点を一度ずつ通る経路)P を探索する LLP プログラムである．

```
p(V,V,[V]) :- v(V).
p(U,V,[U|P]) :- v(U), e(U,W), p(W,V,P).
```

```
e(U,V).
goal(P) :- v(a) -<> v(b) -<>
           v(c) -<> v(d) -<> p(a,d,P).
```

すなわち，goal(P) を実行すると，各頂点を v(a) から v(d) というリソースとして追加し，それらをちょうど一回ずつ消費する道を探索することになる．

TLLP の場合，さらにどの時刻でリソースを消費するかを記述できる．たとえば，以下の TLLP プログラムは，辺を一つ通るたびに時刻を進めている．また，#v(a) は現時刻以降に消費できるリソース，@ @v(b) は，2 時刻後だけに消費できるリソース，@ #v(c) は，1 時刻以降に消費できるリソースを表しているから，このプログラムは，頂点 b を 3 番目に通り，頂点 c を 2 番目以降に通るハミルトン経路を探索している．

```
p(V,V,[V]) :- v(V).
p(U,V,[U|P]) :- v(U), e(U,W), @p(W,V,P).
e(U,V).
goal(P) :- #v(a) -<> @ @v(b) -<>
         @ #v(c) -<> #v(d) -<> p(a,d,P).
```

コンウェイのライフ・ゲームなどのように，ある時点での状態から次の時点での状態を生成するプログラムは，状態をリソースとして表現することにより，TLLP で簡潔に記述できる．**図 7** は，TLLP で記述したライフ・ゲームのプログラムである (出力のためのプログラム部分は省略してある)．

このプログラム中，リソース b(I,J) は，(I,J) の位置に現世代で石があることを表している．述語 next(I,J) は，次世代に石を出現すべき時に成功し，その場合はリソース@b(I,J) を追加する．ただし，述語 next の実行で周囲の石が消費されてしまうのを防ぐため，negation as failure による否定を二重に付け，消費せずにチェックだけを行っている．すなわち，next(I,J) が成功する時，\+ next(I,J) は失敗し，バックトラックによってすべてのリソース消費は取り消される．したがって，next(I,J) が成功する時，\+ \+ next(I,J) は成功し，リソースは消費されない．

この他，時間ペトリネットは，ITLL でエンコーディングできることが示されており [8]，TLLP で簡潔に記述できる問題の一例となっている．

6. TLLP の処理系

第 4.2 節で述べた IOT モデルに基づけば，TLLP のインタプリタ処理系を作成することは容易である．実装の一例として，付録に Prolog で記述したインタプリタ処理系を示す (ただし，∀ を含んだリソースは取り扱えない)．

```
life :- N = 20,
        b(1, 2) -<> b(2, 3) -<> b(3, 1) -<>
        b(3, 2) -<> b(3, 3) -<> n(N) =>
        loop.

loop :- loop(1, 1).

loop(I, J) :- n(N), I >= N, !, @loop.
loop(I, J) :- n(N), J >= N, !,
        I1 is I + 1, loop(I1, 1).
loop(I, J) :- \+ \+ next(I, J), !,
        J1 is J + 1,
        @b(I, J) -<> loop(I, J1).
loop(I, J) :-
        J1 is J + 1, loop(I, J1).

next(I, J) :- b(I, J), !,
        count(I, J, C), 2 =< C, C =< 3.
next(I, J) :- count(I, J, C), C = 3.

count(I1, J1, C) :-
        I0 is I1 - 1, I2 is I1 + 1,
        J0 is J1 - 1, J2 is J1 + 1,
        count_b([(I0,J0),(I0,J1),(I0,J2),
                 (I1,J0),        (I1,J2),
                 (I2,J0),(I2,J1),(I2,J2)], C).

count_b([], 0) :- !.
count_b([(I,J)|IJs], C) :- b(I, J), !,
        count_b(IJs, C1), C is C1 + 1.
count_b([(I,J)|IJs], C) :- count_b(IJs, C).
```

図 7 ライフ・ゲームの TLLP プログラム
Fig. 7 TLLP Program of life game.

このインタプリタ処理系は，リソースの分割を遅延的に行うという点で，単純に HTLL 上でユニフォーム証明探索を行う処理系よりも効率的だが，IOT コンテキストがリストで表現されいるという点では，効率が悪い．

一方，著者らの LLP 処理系は，リソースをハッシュ表で管理しており，また IO モデルを改良したレベル付き IO モデル [12),17)] と呼んでいる計算モデルを採用することにより，実行中ただ一つの IO コンテキストだけを保持すれば良いように工夫されている．

そこで，二つ目の実装方法として，TLLP から LLP へのトラスレート方式について考察する．なお，LLP 処理系には現在，WAM を拡張した抽象機械 LLPAM へのコンパイラ処理系☆と，Java へのトランスレータ処理系☆☆の二種類が存在する．コンパイラ処理系のほうが高速 (SICStus Prolog の compact code コンパイ

☆ http://bach.seg.kobe-u.ac.jp/llp/
☆☆ http://pascal.seg.kobe-u.ac.jp/~banbara/PrologCafe/

ラより，2〜3 倍遅く，SWI Prolog コンパイラより数割速い) ではあるが，現在の所，∀ を含んだリソースのコンパイルが実装されていない．

体系 IO と IOT の差異に注目すれば，基本的には TLLP の各述語に時刻を表す引数を追加することで，LLP の述語に変換できることがわかる．

まず，$R \multimap G'$ あるいは $S \Rightarrow G'$ の形でないゴール G について考えると，以下のような変換 $G[t]$ で，現在の時刻 t を追加すれば良い．

$$
\begin{aligned}
1[t] &= 1 \\
\top[t] &= \top \quad \text{(後述の説明に注意)} \\
p(\vec{x})[t] &= p(t, \vec{x}) \\
(G_1 \otimes G_2)[t] &= G_1[t] \otimes G_2[t] \\
(G_1 \,\&\, G_2)[t] &= G_1[t] \,\&\, G_2[t] \\
(G_1 \oplus G_2)[t] &= G_1[t] \oplus G_2[t] \\
(\bigcirc G)[t] &= G[t+1]
\end{aligned}
$$

ただし，R-<>G' あるいは S=>G' の形のゴール G については，以下のように，追加するリソースの形に応じて異なった変換を行う必要がある．

$$
\begin{aligned}
&(\bigcirc^n (S_1 \,\&\, \cdots \,\&\, S_m) \multimap G)[t] \\
&\quad = (S_1\{t+n\}_1 \,\&\, \cdots \,\&\, S_m\{t+n\}_1) \multimap G[t] \\
&(\bigcirc^n \Box (S_1 \,\&\, \cdots \,\&\, S_m) \multimap G)[t] \\
&\quad = (S_1\{t+n\}_2 \,\&\, \cdots \,\&\, S_m\{t+n\}_2) \multimap G[t] \\
&(S \Rightarrow G)[t] \\
&\quad = S\{t\}_3 \Rightarrow G[t]
\end{aligned}
$$

変換 $S\{t\}_1$ は，時刻 t だけに利用できるリソースの変換であるから次のように定義する．

$$
\begin{aligned}
p(\vec{x})\{t\}_1 &= p(t, \vec{x}) \\
(G \multimap p(\vec{x}))\{t\}_1 &= G[t] \multimap p(t, \vec{x}) \\
(\forall x.S)\{t\}_1 &= \forall x.S\{t\}_1
\end{aligned}
$$

変換 $S\{t\}_2$ は，時刻 t 以降に利用できるリソースの変換であるから次のように定義する．

$$
\begin{aligned}
p(\vec{x})\{t\}_2 &= \forall t'.(t' \geq t \multimap p(t', \vec{x})) \\
(G \multimap p(\vec{X}))\{t\}_2 &= \\
&\qquad \forall t'.((t' \geq t \otimes G[t']) \multimap p(t', \vec{x})) \\
(\forall x.S)\{t\}_2 &= \forall x.S\{t\}_2
\end{aligned}
$$

変換 $S\{t\}_3$ は，任意の時刻で利用できるリソースの変換であるから次のように定義する．

$$
\begin{aligned}
p(\vec{x})\{t\}_3 &= \forall t'.p(t', \vec{x}) \\
(G \multimap p(\vec{x}))\{t\}_3 &= \forall t'.(G[t'] \multimap p(t', \vec{x})) \\
(\forall x.S)\{t\}_3 &= \forall x.S\{t\}_3
\end{aligned}
$$

したがって，たとえば前節のハミルトン経路のプログラムは以下のように変換される．

```
p(T,V,V,[V]) :- v(T,V).
p(T,U,V,[U|P]) :- v(T,U), e(T,U,W),
  (T1 is T+1, p(T1,W,V,P)).
e(T,U,V).
goal(T,P) :-
  (forall T1 \ (T1>=T -<> v(T1,a)) -<>
  ( T2 is T+2,
    v(T2,b) -<>
    (forall T3 \ (T3>=T+1 -<> v(T3,c)) -<>
    (forall T4 \ (T4>=T -<> v(T4,d)) -<>
    p(T,a,d,P)
  ).
```

また，変換後の LLP プログラムは，論文 3) の方法を用いれば，拡張 WAM である抽象機械 LLPAM の機械語命令にすべてコンパイルできる．

しかし，実は上の変換は ⊤ を含んだゴールに対しては正しくない．ゴール ⊤ は，現時刻以降で消費可能なリソースのいくつかを暗黙的に消費するという動作を行う．したがって，$subcontext_t$ では，$t' < t$ となっているリソース $\langle S_1 \,\&\, \cdots \,\&\, S_m,\ t' \rangle$ を消費してはいけないが，上記の変換だと消費される可能性がある．

ただしゴール中に ⊤ が含まれていない場合は，変換後の LLP プログラムの体系 IO での証明可能性は，もとの TLLP プログラムの体系 IOT での証明可能性と同一になることが示せる．

最後に，抽象機械の機械語プログラムへのコンパイル方式について考察する．LLP 言語に対する抽象機械 LLPAM は，IO モデルを改良したレベル付き IO モデルに基づいて設計されており，非常に効率の良い実装を実現している．したがって，これを IOT モデルに基づいて拡張すれば，比較的小規模の変更だけで，効率の良い TLLP の実装が実現できると考えられる．

具体的には，式 $I\,\{G\}_t\,O$ に対応して現在時刻 t を表すレジスタを LLPAM に追加し，IOT コンテキスト中のリソース表現 $\langle R, t \rangle$ に対応して，LLPAM のリソースを表す構造体に時刻のためのフィールドを追加する．さらに，$subcontext_t$ および $pick_t$ に対応する処理の部分を変更すれば良い．ただし，利用可能なリソースを効率良く検索する方法を含め，詳細についてはさらに検討を進める必要がある．

7. おわりに

本論文では，時間と資源の概念を含んだ時相線形論理 ITLL に基づいたプログラミング言語である TLLP の設計と実装方法について述べた．

TLLP は，Prolog および線形論理型言語 LLP の拡張になっており，時間に依存した資源を表現することができる．

しかし，TLLP で取り扱える論理式は，ITLL の一

部でしかない．たとえば,「現在または次の時刻にちょうど一度だけ利用できる資源」を表す $A \,\&\, \bigcirc A$ や,「現在より n 時刻以降はいつでも何度でも利用できる資源」を表す $\bigcirc^n !A$ などは，TLLP のリソース論理式ではない．

Miller の元々のユニフォーム証明の考えに従う限りは，本論文で述べた以上に取り扱える論理式の範囲を広げることは困難なように見える．ユニフォーム証明を古典線形論理へ拡張する研究 (Harland and Pym[6], Andreoli[1]，Miller[15]) などの適用を考える必要があるだろう．

また本論文では，TLLP の実装方法について，まず，TLLP の効率良い実装を実現するための計算モデルである IOT モデルについて述べた．さらに，具体的な実装方法として，IOT モデルに基づくインタプリタ処理系，LLP へのトランスレータ処理系の実現方法について述べ，IOT モデルに基づいた抽象機械の命令へのコンパイル方式について考察を行った．

この抽象機械の詳細設計を進め，コンパイラ処理系を実現することが今後の課題となる．

参 考 文 献

1) Andreoli, J.-M.: Logic Programming with Focusing Proofs in Linear Logic, *Journal of Logic and Computation*, Vol. 2, No. 3, pp. 297–347 (1992).
2) Andreoli, J.-M. and Pareschi, R.: Linear Objects: Logical Processes with Built-In Inheritance, *New Generation Computing*, Vol. 9, pp. 445–473 (1991).
3) Banbara, M. and Tamura, N.: Compiling Resources in a Linear Logic Programming Language, *Proceedings of the JICSLP'98 Post Conference Workshop 7 on Implementation Technologies for Programming Languages based on Logic* (Sagonas, K.(ed.)), pp. 32–45 (1998).
4) 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, pp. 67–81 (1996).
5) Girard, J.-Y.: Linear Logic, *Theoretical Computer Science*, Vol. 50, pp. 1–102 (1987).
6) Harland, J. and Pym, D.: A Uniform Proof-Theoretic Investigation of Linear Logic Programming, *Journal of Logic and Computation*, Vol. 4, No. 2, pp. 175–207 (1994).
7) Harland, J. and Winikoff, M.: Deterministic Resource Management for the Linear Logic Programming Language Lygon, Technical Report TR 94/23, Melbourne University, Department of Computer Science (1994).
8) Hirai, T.: An Application of Temporal Linear Logic to Timed Petri Nets, *Proceedings of the Petri Nets'99 Workshop on Applications of Petri Nets to Intelligent System Development*, pp. 2–13 (1999).
9) Hodas, J. S.: Lolli: An Extension of λProlog with Linear Context Management, *Workshop on the λProlog Programming Language* (Miller, D.(ed.)), Philadelphia, Pennsylvania, pp. 159–168 (1992).
10) 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).
11) Hodas, J. S. and Miller, D.: Logic Programming in a Fragment of Intuitionistic Linear Logic, *Information and Computation*, Vol. 110, No. 2, pp. 327–365 (1994). Extended abstract in the Proceedings of the Sixth Annual Symposium on Logic in Computer Science, Amsterdam, July 15–18, 1991.
12) Hodas, J. S., Watkins, K., Tamura, N. and Kang, K.-S.: Efficient Implementation of a Linear Logic Programming Language, *Proceedings of the 1998 Joint International Conference and Symposium on Logic Programming* (Jaffar, J.(ed.)), MIT Press, pp. 145–159 (1998).
13) Kanovich, M. I. and Ito, T.: Temporal Linear Logic Specifications for Concurrent Processes (Extended Abstract), *Proceedings of 12th Annual IEEE Symposium on Logic in Computer Science (LICS'97)*, pp. 48–57 (1997).
14) 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, pp. 279–294 (1993).
15) Miller, D.: A Multiple-Conclusion Specification Logic, *Theoretical Computer Science*, Vol. 165, No. 1, pp. 201–232 (1996).
16) 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, pp. 125–157 (1991).
17) Tamura, N. and Kaneda, Y.: Extension of WAM for a linear logic programming language, *Second Fuji International Workshop on Func-*

tional and Logic Programming (Ida, T., Ohori, A. and Takeichi, M.(eds.)), World Scientific, pp. 33–50 (1996).
18) Tanabe, M.: Timed Petri Nets and Temporal Linear Logic, *Lecture Notes in Computer Science 1248: Proceedings of Application and Theory of Petri Nets*, pp. 156–174 (1997).
19) 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, pp. 25–30 (1996).
20) 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), pp. 1–16 (1999).

付録　TLLP のインタプリタ処理系

SICStus Prolog で記述した TLLP のインタプリタ処理系.

```
/*
        TLLP interpreter in Prolog
*/
:- op(1060, xfy, (&)).
:- op( 950, xfy, [-<>, =>]).
:- op( 900,  fy, [!, @, #]).

prove(G) :-
        prove(G, 0, [], []).

prove(true, T, I, I) :- !.
prove(top, T, I, O) :- !,
        subcontext(T, O, I).
prove((G1, G2), T, I, O) :- !,
        prove(G1, T, I, M),
        prove(G2, T, M, O).
prove((G1 & G2), T, I, O) :- !,
        prove(G1, T, I, O),
        prove(G2, T, I, O).
prove((G1 ; G2), T, I, O) :- !,
        (prove(G1, T, I, O) ;
         prove(G2, T, I, O)).
prove((R -<> G), T, I, O) :- !,
        count_next(R, N, R1),
        T1 is T + N,
        prove(G, T, [(R1,T1)|I], [1|O]).
prove((S => G), T, I, O) :- !,
        prove(G, T, [(!S,0)|I], [(!S,0)|O]).
prove(!G, T, I, I) :- !,
        prove(G, T, I, I).
prove(@G, T, I, O) :- !,
        T1 is T + 1,
        prove(G, T1, I, O).
prove(A, T, I, O) :-
        pick(T, I, O, A).
prove(A, T, I, O) :-
        pick(T, I, M, (G -<> A)),
        prove(G, T, M, O).

count_next(@R, N, R1) :- !,
        count_next(R, N1, R1),
        N is N1 + 1.
count_next(R, 0, R).

pick(T, I, O, S) :-
        pick1(T, I, O, S).
pick(T, I, I, S) :-
        rule(S).
pick(T, I, I, (G -<> A)) :-
        rule((A :- G)).

pick1(T, [(!S,0)|I], [(!S,0)|I], S).
pick1(T, [(#R,T0)|I], [1|I], S) :-
        T >= T0,
        select(R, S).
pick1(T, [(R,T)|I], [1|I], S) :-
        \+(R = (!_)), \+(R = (#_)),
        select(R, S).
pick1(T, [R|I], [R|O], S) :-
        pick1(T, I, O, S).

select((R1 & R2), R) :- !,
        (select(R1, R) ; select(R2, R)).
select(R, R).

subcontext(T, [], []).
subcontext(T, [(!S,0)|O], [(!S,0)|I]) :-
        subcontext(T, O, I).
subcontext(T, [R1|O], [(#R,T0)|I]) :-
        (R1 = (#R,T0) ; R1 = 1),
        subcontext(T, O, I).
subcontext(T, [R1|O], [(R,T0)|I]) :-
        \+(R = (!_)), \+(R = (#_)),
        T0 >= T,
        (R1 = (R,T0) ; R1 = 1),
        subcontext(T, O, I).

rule(( p(V,V,[V]) :- v(V) )).
rule(( p(U,V,[U|P]) :-
         v(U), e(U,W), @p(W,V,P) )).
rule(( e(U,V) )).
rule(( goal(P) :- #v(a) -<> @ @v(b) -<>
         @ #v(c) -<> #v(d) -<> p(a,d,P) )).
```