Logic Programming in a Fragment of Intuitionistic Temporal Linear Logic

Mutsunori BANBARA[1], Kyoung-Sun KANG[2],
Takaharu HIRAI[3], and Naoyuki TAMURA[4]

[1] Department of Mathematics, Nara National College of Technology,
22 Yata, Yamatokoriyama, 639-1080, Japan
banbara@libe.nara-k.ac.jp
[2] Department of Computer Engineering, Pusan University of Foreign Studies,
55-1, Uam-dong, Nam-gu, Pusan, 608-738, Korea
kskang@taejo.pufs.ac.kr
[3] Graduate School of Science and Technology, Kobe University,
1-1 Rokkodai, Nada, Kobe, 657-8501, Japan
hirai@pascal.cs.kobe-u.ac.jp
[4] Department of Computer and Systems Engineering, Kobe University,
1-1 Rokkodai, Nada, Kobe, 657-8501, Japan
tamura@kobe-u.ac.jp

Abstract. Recent development of logic programming languages based on linear logic suggests a successful direction to extend logic programming to be more expressive and more efficient. The treatment of formulas-as-resources gives us not only powerful expressiveness, but also efficient access to a large set of data. However, in linear logic, whole resources are kept in one context, and there is no straight way to represent complex data structures as resources. For example, in order to represent an ordered list and time-dependent data, we need to put additional indices for each resource formula. This paper describes a logic programming language, called TLLP, based on intuitionistic temporal linear logic. This logic, an extension of linear logic with some features from temporal logics, allows the use of the modal operators '$\bigcirc$'(next-time) and '$\Box$'(always) in addition to the operators used in intuitionistic linear logic. The intuitive meaning of modal operators is as follows: $\bigcirc\, B$ means that B can be used exactly once at the next moment in time; $\Box\, B$ means that B can be used exactly once any time; $!\, B$ means that B can be used arbitrarily many times (including 0 times) at any time. We first give a proof theoretic formulation of the logic of the TLLP language. We then present a series of resource management systems designed to implement not only interpreters but also compilers based on an extension of the standard WAM model.

1 Introduction

Linear logic was introduced by J.-Y.Girard in 1987 [4] as a resource-conscious refinement of classical logic. Since then a number of logic programming languages

based on linear logic have been proposed: LO[1], ACL[12], Lolli[3][8][9], Lygon[5], Forum[13], and LLP[2][15].

These languages suggest a direction to extend logic programming to be more expressive and more efficient. The treatment of formulas-as-resources gives us not only powerful expressiveness, but also efficient access to a large set of data. However, in linear logic, whole resources are kept in one context, and there is no straight way to represent complex data structures as resources. For example, in order to represent an ordered list and time-dependent data, we need to put additional indices for each resource formula.

Temporal Linear Logic (TLL) is an extension of linear logic with some features of temporal logic. TLL was first studied by Kanovich and Itoh [11], and a cut-free sequent system has been proposed by Hirai [6]. The semantics model of TLL consists an infinite number of phase spaces linearly ordered by the time clock. Each phase space is the same as that of linear logic.

This paper describes a logic programming language, called TLLP, based on intuitionistic temporal linear logic [6]. We first give a proof theoretic formulation of the logic of the TLLP language. We then present a series of resource management systems designed to implement not only interpreters but also compilers based on an extension of the standard WAM model. Finally, we describe some implementation methods based on our systems.

2 Intuitionistic Temporal Linear Logic

In this paper, we will focus on the sequent system *ITLL* [6] of intuitionistic temporal linear logic developed by Hirai. The expressive power of *ITLL* is shown by a natural encoding of Timed Petri Nets. It is this logic that we shall use to design and implement the logic programming language described below.

ITLL allows the use of the modal operators '$\bigcirc$'(next-time) and '$\Box$'(always) in addition to the operators used in intuitionistic linear logic. Compared with the sequent system *ILL* (see Fig. 1) of intuitionistic linear logic, three rules (L $\Box$), (R $\Box$), and ($\bigcirc$) are added. The entire set of *ITLL* sequent rules is given in Fig. 2. Here, the left-hand side of sequents are multisets of formulas, and the structural rule for exchange need not be explicitly stated. The structural rule for weakening (W !) and contraction (C !) are available only for assumptions marked with the modal operator '!'. This means that, in general, formulas not !-marked can be used exactly once. Limited-use formulas can represent time-dependent resources in *ITLL*. The intuitive meaning of these modal operators is as follows:

- $\bigcirc\, B$ means that B can be used exactly once at the next moment in time.
- $\Box\, B$ means that B can be used exactly once any time.
- $!\, B$ means that B can be used arbitrarily many times (including 0 times) at any time.

By combining these modalities with binary operators in linear logic, several resources can be expressed. For example, $B \,\&\, \bigcirc\, B$ means that B can be used

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

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

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

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

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

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

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

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

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

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

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

provided, in each case, y does not appear free in the conclusion.

Fig. 1. The proof system *ILL* for intuitionistic linear logic

exactly once either at the present time or at the next moment in time. $\bigcirc(1 \,\&\, B)$ means that B can be used at most once at the next moment in time.

Two formulas B and C are equivalent, denoted $B \equiv C$, if the sequents $B \longrightarrow C$ and $C \longrightarrow B$ are provable in *ITLL*. The notation $\bigcirc^n$ means n multiplicity of $\bigcirc$. We note the following sequents that are provable in *ITLL*.

$$!\,B \equiv !\,!\,B, \qquad \Box\,B \equiv \Box\,\Box\,B, \qquad !\,B \equiv \Box\,!\,B,$$
$$!\,B \longrightarrow \Box\,B \otimes \cdots \otimes \Box\,B, \qquad \Box\,B \longrightarrow \bigcirc^n B \quad (n \geq 0)$$

The main differences from other temporal linear logic systems [11][16] are that *ITLL* includes the modal operator '!', and it satisfies a cut elimination theorem. Both of these additions are very important for the design of a language based on the notion of *Uniform Proofs*.

3 Language Design

The idea of uniform proofs [14], proposed by Miller et. al, is a simple and powerful notion for designing logic programming languages. Uniform proof search is a cut-

(Rules of *ILL*)

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

Fig. 2. The proof system *ITLL* for intuitionistic temporal linear logic

free, *goal-directed proof search* in which a sequent $\Gamma \longrightarrow G$ denotes the state of the computation trying to solve the goal G from the program Γ. Goal-directed proof search is characterized operationally by the bottom-up construction of proofs in which right-introduction rules are applied first and left-introduction rules are applied only when the right-hand side is atomic. This means that the operators in the goal G are executed independently from the program Γ, and the program is only considered when its goal is atomic. A logical system is an *abstract logic programming language* if restricting it to uniform proofs retains completeness. The logics of Prolog, λProlog, and Lolli are examples of abstract logic programming language.

Clearly, intuitionistic linear logic (even over the connectives: $\top$, $\&$, $\otimes$, $\multimap$, $!$, and $\forall$) is not an abstract logic programming language. For example, the sequents $a \otimes b \longrightarrow b \otimes a$ and $!\,a \,\&\, b \longrightarrow !\,a$ are both provable in *ILL* but do not have uniform proofs.

Hodas and Miller have designed the linear logic programming language Lolli [7][8] by restricting formulas so that the above counterexamples do not appear, although it retains desirable features of linear logic connectives such as ! and $\otimes$. The Lolli language is based on the following fragment of linear logic:

$$R ::= \top \mid A \mid R_1 \,\&\, R_2 \mid G \multimap R \mid G \Rightarrow R \mid \forall x.R$$
$$G ::= 1 \mid \top \mid A \mid G_1 \otimes G_2 \mid G_1 \,\&\, G_2 \mid G_1 \oplus G_2 \mid R \multimap G \mid R \Rightarrow G \mid !\,G \mid \forall x.G \mid \exists x.G$$

Here, R-formulas and G-formula are called *resource* and *goal formulas* respectively. The connective $\Rightarrow$ is called *intuitionistic implication*, and it is defined as $B \Rightarrow C \equiv (!\,B) \multimap C$.

The sequent of Lolli is of the form $\Gamma; \Delta \longrightarrow G$ where Γ is a set of resource formulas, Δ is a multiset of resource formulas, and G is a goal formula. Γ and Δ are called *intuitionistic* and *linear context* respectively, and they correspond to the *program*. G is called the *goal*. The sequent $\Gamma; \Delta \longrightarrow G$ can be mapped to the linear logic sequent $!\,\Gamma, \Delta \longrightarrow G$. Thus, the right introduction rule for $\multimap$ adds its assumption (called a *linear resource*) to the linear context, in which every formula can be used exactly once. The right introduction rule for $\Rightarrow$ adds its assumption (called an *intuitionistic resource*) to the intuitionistic context, in which every formula can be used arbitrarily many times (including 0 times).

Hodas and Miller developed a series of proof systems $\mathcal{L}$ (see Fig. 3) and $\mathcal{L}'$ in [7]. They proved that $\mathcal{L}$ is sound and complete with respect to the *ILL* rules restricted to the Lolli language. They also proved $\mathcal{L}$ preserves completeness even if probability is restricted to uniform proofs. $\mathcal{L}'$ is the proof system that results

$$\frac{}{\Gamma; A \longrightarrow A}\ (\text{Identity}) \qquad \frac{\Gamma, B; \Delta, B \longrightarrow C}{\Gamma, B; \Delta \longrightarrow C}\ (\text{absorb})$$

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

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

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

$$\frac{\Gamma; \emptyset \longrightarrow C_1 \quad \Gamma; \Delta, B \longrightarrow C_2}{\Gamma; \Delta, C_1 \multimap B \longrightarrow C_2}\ (\text{L}\Rightarrow) \qquad \frac{\Gamma, B; \Delta \longrightarrow C}{\Gamma; \Delta \longrightarrow B \Rightarrow C}\ (\text{R}\Rightarrow)$$

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

provided that y is not free in the conclusion.

$$\frac{\Gamma; \emptyset \longrightarrow C}{\Gamma; \emptyset \longrightarrow\, !\, C}\ (\text{R}\,!) \qquad \frac{\Gamma; \Delta_1 \longrightarrow C_1 \quad \Gamma; \Delta_2 \longrightarrow C_2}{\Gamma; \Delta_1, \Delta_2 \longrightarrow C_1 \otimes C_2}\ (\text{R}\otimes)$$

$$\frac{\Gamma; \Delta \longrightarrow C[t/x]}{\Gamma; \Delta \longrightarrow \exists x.C}\ (\text{R}\exists) \qquad \frac{\Gamma; \Delta \longrightarrow C_i}{\Gamma; \Delta \longrightarrow C_1 \oplus C_2}\ (\text{R}\oplus_i)$$

Fig. 3. The proof system $\mathcal{L}$ for the Lolli language

(Rules of $\mathcal{L}$)

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

Fig. 4. $\mathcal{TL}$: A proof system for the connectives $\top$, $\&$, $\multimap$, $\Rightarrow$, $\forall$, 1, !, $\otimes$, $\oplus$, $\exists$, $\bigcirc$, and $\Box$.

from replacing the Identity, L$\multimap$, L$\Rightarrow$, L$\&$, and L$\forall$ rules in $\mathcal{L}$ with a single rule, called *backchaining*.

In this paper, we will use a more restrictive definition for resource and goal formulas. Let A be atomic and $m \geq 1$:

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

Here, S-formulas are called *resource clauses* in which A and G are called the *head* and the *body* respectively. S-formulas correspond to program clauses. Although this simplification does not change expressiveness of the language, it makes the presentation of *backchaining* simpler, as is discussed below.

Since full intuitionistic linear logic is not an abstract logic programming language, it is obvious that intuitionistic temporal linear logic is not as well. For example, in addition to the counterexamples in *ILL*, the sequents $\Box \bigcirc a \longrightarrow \bigcirc a$,

$!\bigcirc a \longrightarrow \bigcirc a$, and $a \,\&\, \bigcirc a \longrightarrow \bigcirc a$ are all provable in *ITLL*, but they do not have uniform proofs.

Fig. 4 presents a proof system $\mathcal{TL}$ for the connectives $\top$, $\&$, $\multimap$, $\Rightarrow$, $\forall$, 1, $!$, $\otimes$, $\oplus$, $\exists$, $\bigcirc$, and $\Box$. Two rules, L$\Box$ and $\bigcirc$, are added in addition to those that arise in $\mathcal{L}$. This system has been designed to support the logic programming language TLLP over the following formulas: If A is atomic and $m \geq 1$,

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

Let D be a $\&$-product of resource clauses $S_1 \,\&\, \cdots \,\&\, S_m$. Compared with Lolli, $\bigcirc^n D$ and $\bigcirc^n \Box D$ are added to resource formulas, and $\bigcirc G$ is added to goal formulas. The intuitive meaning of these formulas is as follows: $\bigcirc^n D$ means that the resource clause S_i $(1 \leq i \leq m)$ in D can be used exactly once at time n; $\bigcirc^n \Box D$ means that the resource clause S_i $(1 \leq i \leq m)$ in D can be used exactly once any time at and after time n; $\bigcirc G$ adjusts time one clock ahead and then executes G.

The proofs of propositions in this paper are based on Hodas and Miller's results in [7] for the Lolli language, and we will only give proof outlines.

Proposition 1. Let G be a goal formula, Γ a set of resource clauses, and Δ a multiset of resource formulas. Let D^* be the result of replacing all occurrences of $B \Rightarrow C$ in D with $(!\,B) \multimap C$, and let $\Gamma^* = \{B^* \mid B \in \Gamma\}$. Then the sequent $\Gamma; \Delta \longrightarrow G$ is provable in $\mathcal{TL}$ if and only if $!(\Gamma^*), \Delta^* \longrightarrow G^*$ is provable in *ITLL*.

Proof (sketch). The proof of this proposition can be shown by giving a simple conversion between proofs in the two systems. The cases of $\bigcirc$ and L$\Box$ are also immediate. □

Proposition 2. Let G be a goal formula, Γ a set of resource clauses, and Δ a multiset of resource formulas. Then the sequent $\Gamma; \Delta \longrightarrow G$ has a proof in $\mathcal{TL}$ if and only if it has a uniform proof in $\mathcal{TL}$.

Proof (sketch). The proof in the reverse direction is immediate, since a uniform proof in $\mathcal{TL}$ is a proof in $\mathcal{TL}$. The forward direction can be proved by showing that any proof in $\mathcal{TL}$ can be converted to a uniform proof of the same endsequent by permuting the rules to move occurrences of the left-rule up, though, and above instances of the right-rule. We explicitly show one case, that is when L$\Box$ occurs below R$\&$:

$$\dfrac{\dfrac{\begin{matrix}\Xi_1 \\ \Gamma; \Delta, B \longrightarrow C_1\end{matrix} \quad \begin{matrix}\Xi_2 \\ \Gamma; \Delta, B \longrightarrow C_2\end{matrix}}{\Gamma; \Delta, B \longrightarrow C_1 \,\&\, C_2}\,(\mathrm{R}\,\&)}{\Gamma; \Delta, \Box\, B \longrightarrow C_1 \,\&\, C_2}\,(\mathrm{L}\,\Box)$$

$$\frac{}{\Gamma; D \longrightarrow A}\ (\mathrm{BC}_1) \qquad \frac{}{\Gamma, D; \emptyset \longrightarrow A}\ (\mathrm{BC!}_1)$$

provided, in each case, A is atomic and $A \in \|D\|$.

$$\frac{\Gamma; \Delta \longrightarrow G}{\Gamma; \Delta, D \longrightarrow A}\ (\mathrm{BC}_2) \qquad \frac{\Gamma, D; \Delta \longrightarrow G}{\Gamma, D; \Delta \longrightarrow A}\ (\mathrm{BC!}_2)$$

provided, in each case, A is atomic and $G \multimap A \in \|D\|$.

Fig. 5. Backchaining for the proof system $\mathcal{TL}'$

where Ξ_1 and Ξ_2 are uniform proofs of their endsequents respectively. The above proof structure can be converted to the following:

$$\frac{\dfrac{\begin{matrix}\Xi_1\\ \Gamma; \Delta, B \longrightarrow C_1\end{matrix}}{\Gamma; \Delta, \Box B \longrightarrow C_1}\ (\mathrm{L}\Box) \quad \dfrac{\begin{matrix}\Xi_2\\ \Gamma; \Delta, B \longrightarrow C_2\end{matrix}}{\Gamma; \Delta, \Box B \longrightarrow C_2}\ (\mathrm{L}\Box)}{\Gamma; \Delta, \Box B \longrightarrow C_1 \,\&\, C_2}\ (\mathrm{R}\,\&)$$

□

As with $\mathcal{L}$ and $\mathcal{L}'$, the left-hand rules can be restricted to a form of backchaining. Let us consider the following definition: Let R be a resource formula. $\|R\|$ is defined as a set of resource clauses (S-formulas):

1. if $R = A$ then $\|R\| = \{A\}$,
2. if $R = G \multimap A$ then $\|R\| = \{G \multimap A\}$,
3. if $R = \forall x.S$ then for all closed terms t, $\|R\| = \|S[t/x]\|$,
4. if $R = S_1 \,\&\, \cdots \,\&\, S_m$ then $\|R\| = \|S_1\| \cup \cdots \cup \|S_m\|$,
5. if $R = \Box R'$ then $\|R\| = \|R'\|$,
6. if $R = \bigcirc R'$ then $\|R\| = \emptyset$.

Let $\mathcal{TL}'$ be a proof system that results from replacing the Identity, absorb, L$\multimap$, L$\Rightarrow$, L&, L$\forall$, and L$\Box$ rules in $\mathcal{TL}$ with the backchaining rules in Fig. 5. These backchaining rules (especially the definition of $\|\cdot\|$) are simpler than the original rule for Lolli because of the restrictive definition of resource formulas. It is noticed that the absorb rule is integrated into (BC!$_1$) and (BC!$_2$).

Proposition 3. Let G be a goal formula, Γ a set of resource clauses, and Δ a multiset of resource formulas. Then the sequent $\Gamma; \Delta \longrightarrow G$ has a proof in $\mathcal{TL}$ if and only if it has a proof in $\mathcal{TL}'$.

Since uniform proofs are complete for $\mathcal{TL}$, this proposition can be proved by showing that there is a uniform proof in $\mathcal{TL}$ if and only if there is a proof in $\mathcal{TL}'$. We do not present the proof here. A similar proof has been given by Hodas and Miller in [7] for the Lolli language.

3.1 TLLP Example Programs

We will present simple TLLP examples here. For the syntax, we use '`:-`' for the inverse of $\multimap$, '`,`' for $\otimes$, '`-<>`' for $\multimap$, '`=>`' for $\Rightarrow$, '`@`' for $\bigcirc$, and '`#`' for $\Box$.

We first consider a Lolli program that finds a Hamilton path through the complete graph of four vertices. Since each vertex is represented as a linear resource, the constraints such that each vertex must be used exactly can be expressed.

```
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).
```

When the goal `goal(P)` is executed, the vertices are added as resources, and the goal `p(a,d,P)` will search a path from `a` to `d` by consuming each vertex exactly once.

In addition to the resource-sensitive features of Lolli, TLLP can describe the time-dependent properties of resources, in particular, the precise order of the moments when some resources are consumed. For example, `#v(a)` denotes the vertex a that can be used exactly once at and after present. `@ #v(c)` denotes the vertex c that can be used exactly once at and after the next moment in time. So, the following TLLP program finds a Hamilton path that satisfies such constraints. It is noticed that time is adjusted one clock ahead every time the path crosses an arc.

```
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).
```

Our next example is a simple Timed Petri Nets reachability emulator. This program checks the reachability of a Timed Petri Net in Fig. 6 from the initial marking (one token in p) to the final marking (one token in p and two tokens in r). Each d_i, a non-negative integer, is the delay time for the transition t_i.

```
tpn :- #p -<> (goal :- p, r, r) => tpn(1, 100).
tpn(Dep, Lim) :- Dep =< Lim, fire(Dep).
tpn(Dep, Lim) :- Dep =< Lim, Dep1 is Dep + 1, tpn(Dep1, Lim).
next(D) :- D1 is D - 1, D1 > 0, fire(D1).
fire(D) :- goal.
fire(D) :- p, @ #p -<> @ #q -<> next(D).
fire(D) :- q, q, q, @ #r -<> next(D).
fire(D) :- @next(D).
```

Since the proof search of TLLP is depth-first and is not complete, we use a *iterative deepening* search, a combination of depth-fisrt and breadth-first search. First, the predicate `tpn(Dep, Lim)` checks the reachability at depth 1, and then it increases the depth by one if the check fails.

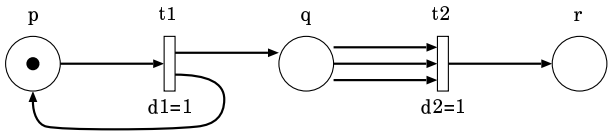

Fig. 6. An example of Timed Petri Nets

Besides the examples presented above, the latest TLLP package includes programs for the flow-shop scheduling problem and Conway's Game of Life.

4 Resource Management Model

The resource management during a proof search in $\mathcal{TL}'$ is a serious problem for the implementor. Let us consider, for example, the execution of the goal $G_1 \otimes G_2$:

$$\frac{\Gamma; \Delta_1 \longrightarrow G_1 \quad \Gamma; \Delta_2 \longrightarrow G_2}{\Gamma; \underbrace{\Delta_1, \Delta_2}_{\Delta} \longrightarrow G_1 \otimes G_2}\ \mathrm{R}\otimes$$

When the system applies this rule during bottom-up search, the linear context Δ must be divided into Δ_1 and Δ_2. If Δ contains n resource formulas, all 2^n possibilities might need to be tested to find a desirable partition.

For Lolli, Hodas and Miller solved this problem by splitting resources lazily, and they proposed a new execution model called the I/O model [8].

In this model, the sequent $I\,\{G\}\,O$ means that the goal G can be executed given the *input context* I so that the *output context* O remains. The input and output context, together called *IO*-context, are lists of resource formulas, !-marked resource formulas, or the special symbol 1 that denotes a place where a resource formula has been consumed. In the the execution of the goal $G_1 \otimes G_2$:

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

First, $I\,\{G_1\}\,M$ tries to execute G_1 given the input context I. If this succeeds, the output context M is forwarded to G_2, and then $M\,\{G_2\}\,O$ is attempted. If this second attempt fails, $I\,\{G_1\}\,M$ retries to find a different, more desirable consumption pattern.

We will extend the I/O model for the TLLP language. The additional problem here is that the bottom-up application of the rule for $\bigcirc$ in $\mathcal{TL}'$ requires manipulating large dynamic data structures.

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

For example, when the system executes the goal $\bigcirc\,G$ given input context $I = [p, \bigcirc\,q, \bigcirc\bigcirc\,r, !\,s]$, we need to reconstruct and create a new input context $I' = [1, q, \bigcirc\,r, !\,s]$ before the execution of the goal G.

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

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

$$\frac{I\,\{G_i\}_T\,O}{I\,\{G_1 \oplus G_2\}_T\,O}\ (\oplus) \qquad \frac{[\langle !\,S, 0\rangle \mid I]\,\{G\}_T\,[\langle !\,S, 0\rangle \mid O]}{I\,\{S \Rightarrow G\}_T\,O}\ (\Rightarrow)$$

$$\frac{[\langle R, T+n\rangle \mid I]\,\{G\}_T\,[1 \mid O]}{I\,\{\bigcirc^n R \multimap G\}_T\,O}\ (\multimap)$$

provided that R is a formula of the form: $S_1 \,\&\, \cdots \,\&\, S_m$ or $\Box(S_1 \,\&\, \cdots \,\&\, S_m)$.

$$\frac{I\,\{G\}_T\,I}{I\,\{!\,G\}_T\,I}\ (!) \qquad \frac{I\,\{G\}_{T+1}\,O}{I\,\{\bigcirc\,G\}_T\,O}\ (\bigcirc)$$

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

Fig. 7. *IOT*: An I/O model for propositional TLLP

We introduce a *time index* to solve this problem. Fig. 7 presents an extension of the I/O model for the TLLP language, called *IOT*. *IOT* makes use of a time index T. The sequent is of the form $I\,\{G\}_T\,O$. T, non-negative integer, is the *current time*. At a given point in the proof, only resources that can be used at that time may be used. T is also used to set a *consumption time* of newly added resources.

Each element in *IOT*-context is a pair $\langle R, t\rangle$ where R is a resource formula or !-marked resource formula, and t is its consumption time, or the special symbol 1. Linear resources have the form $\langle S_1 \,\&\, \ldots \,\&\, S_m, t\rangle$ or $\langle \Box(S_1 \,\&\, \ldots \,\&\, S_m), t\rangle$, where t is its consumption time calculated from the value of T, and its multiplicity of $\bigcirc$. Intuitionistic resources have the form $\langle !\,S, 0\rangle$, where S is a resource clause. For example, the consumable resources at time T have the following forms in the context: $\langle S_1 \,\&\, \ldots \,\&\, S_m, T\rangle$, $\langle \Box(S_1 \,\&\, \ldots \,\&\, S_m), t\rangle$ where $t \leq T$, and $\langle !\,S, 0\rangle$

The relation $pick_T(I, O, S)$ holds if S occurs in the context I and is consumable at time T, and O results from replacing that occurrence of S in I with 1. The relation also holds if $!S$ occurs in I, and I and O are equal. The relation $subcontext_T(O, I)$ holds if O arises from replacing arbitrarily many (including 0) non-!-marked elements of I that are consumable any time at and after time T with 1.

To prove that *IOT* is logically equivalent to $\mathcal{TL}'$, we need to define the notion of difference $I -_T O$ for two *IOT*-context I and O that satisfy the relation $subcontext_T(O, I)$. $I -_T O$ is a pair $\langle \Gamma, \Delta\rangle$, where Γ is a set of all formulas S such that $\langle !\,S, 0\rangle$ is an element of I (and O), and Δ is a multiset of all formulas $\bigcirc^{\max(0,t-T)} R$ such that $\langle R, t\rangle$ occur in I (If R is of the form $S_1 \,\&\, \cdots \,\&\, S_m$, then $t \geq T$. If R is of the form $\Box(S_1 \,\&\, \cdots \,\&\, S_m)$, then t is arbitrary), and the corresponding place in O is the symbol 1.

Proposition 4. Let T be a non-negative integer. Let I and O be IOT-contexts that satisfy $subcontext_T(O, I)$. Let $I -_T O$ be the pair $\langle \Gamma, \Delta \rangle$ and let G be a goal formula. $I\,\{G\}_T\,O$ is provable in IOT if and only if $\Gamma; \Delta \longrightarrow G$ is provable in $\mathcal{TL}'$.

Proof (sketch). This proposition, in both directions, can be proved by induction on proof structure. □

5 Level-Based Resource Management Model

The I/O model provides an efficient computation model for proof search. The I/O model has been refined several times. Cervesato et. al recently have proposed a refinement designed to eliminate the non-determinism in management of linear context involving & and $\top$ [3]. However, the I/O model and its refinements still require copying and scanning large dynamic data structures to control the consumption of linear resources. Thus, they are more suited to develop interpreters in high-level languages rather than compilers.

We point out two problems here. First, during the execution of $I\,\{G\}\,O$ (especially *pickR*), the context O is reconstructed from the context I by replacing linear consumed resources with 1. This will slow down the execution speed. Secondly, let us consider the execution of the goal $G_1 \,\&\, G_2$:

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

This rule means that the goal G_1 and G_2 must use the same resources. In a naive implementation, the system first copies the input context and executes the two conjuncts separately, and then it compares their output contexts. This leads to unnecessary backtracking.

To solve these problems, Tamura et. al have introduced a refinement of the I/O model with *level indices* [10][15], called the *IOL* model [1]. Hodas et. al recently proposed the refinement of *IOL* for the complete treatment of $\top$ in [9].

IOL makes use of two level indices L and U to manage the consumption of resources. The sequent is of the form $\vdash_{L,U} I\,\{G\}\,O$. L, a positive integer, is the *current consumption level*. At a given point in the proof, only linear resources labeled with that consumption level (and intuitionistic resources labeled with 0) can be used. U, a negative integer, is the *current consumption maker*. When a linear resource is consumed, its consumption level is changed to the value of U.

Each element in *IOL*-context is a pair $\langle R, \ell \rangle$, where R is a resource formula, and ℓ is its consumption level. Linear resources have the form $\langle R, \ell \rangle$, where ℓ is the value of L at which the resource can be consumed. Intuitionistic resources have the form $\langle S, 0 \rangle$ where S is a resource clause.

$$\frac{\vdash_{L,U-1} I\,\{G_1\}\,M \quad change_{U-1,L+1}(M, N) \quad \vdash_{L+1,U} N\,\{G_2\}\,O \quad thinable_{L+1}(O)}{\vdash_{L,U} I\,\{G_1 \,\&\, G_2\}\,O}\ (\&)$$

[1] In this paper, we use the notation in [10] to explain the *IOL* model.

$$\frac{}{\vdash^T_{L,U} I\,\{1\}\,I}\ (1) \qquad \frac{\mathit{subcontext}^T_{U,L}(O,I)}{\vdash^T_{L,U} I\,\{\top\}\,O}\ (\top)$$

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

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

$$\frac{\vdash^T_{L,U} I\,\{G_i\}\,O}{\vdash^T_{L,U} I\,\{G_1 \oplus G_2\}\,O}\ (\oplus_i) \qquad \frac{\vdash^T_{L,U} [\langle S,0,0\rangle \mid I]\,\{G\}\,[\langle S,0,0\rangle \mid O]}{\vdash^T_{L,U} I\,\{S \Rightarrow G\}\,O}\ (\Rightarrow)$$

$$\frac{\vdash^T_{L,U} [\langle R,T+n,L\rangle \mid I]\,\{G\}\,[\langle R,T+n,U\rangle \mid O]}{\vdash^T_{L,U} I\,\{\bigcirc^n R \multimap G\}\,O}\ (\multimap)$$

provided that R is a formula of the form: $S_1 \,\&\, \cdots \,\&\, S_m$ or $\Box(S_1 \,\&\, \cdots \,\&\, S_m)$.

$$\frac{\vdash^T_{L+1,U} I\,\{G\}\,O}{\vdash^T_{L,U} I\,\{!\,G\}\,O}\ (!) \qquad \frac{\vdash^{T+1}_{L,U} I\,\{G\}\,O}{\vdash^T_{L,U} I\,\{\bigcirc\,G\}\,O}\ (\bigcirc)$$

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

Fig. 8. *IOTL*: A level-based I/O model for propositional TLLP

For example, the outline of the execution of the goal $G_1 \,\&\, G_2$ is as follows:

1. $\vdash_{L,U-1} I\,\{G_1\}\,M$ Decrement U so that we know which resources are consumed during the execution of G_1, and then execute G_1.
2. $\mathit{change}_{U-1,L+1}(M,N)$ Change the level of resources that have been consumed in G_1 to $L+1$.
3. $\vdash_{L+1,U} N\,\{G_2\}\,O$ Increment L and U, and then execute G_2.
4. $\mathit{thinable}_{L+1}(O)$ Check whether none of resources in O have $L+1$ as their consumption level.

IOL is logically equivalent to $\mathcal{L}'$. In *IOL*, all resources are kept in a single table, called *resource table*, during execution. The consumption of resources can be achieved easily by changing their consumption level destructively. The idea of this model has already been used as a basis for a compiler system for a useful fragment of first-order Lolli, in which the resource table is implemented as an array, and the speed access to resources is achieved by using a hash table.

For TLLP, we give a refinement of *IOT*, called *IOTL* in Fig. 8, with level indices of *IOL*. The sequent of *IOTL* is of the form $\vdash^T_{L,U} I\,\{G\}\,O$, where T is the current time, L is the current consumption level, and U is the current consumption maker.

Each element in *IOTL*-contexts is a tuple $\langle R,t,\ell\rangle$, where R is a resource formula, t is its consumption time, and ℓ is its consumption level. Linear resources have the form $\langle S_1 \,\&\, \ldots \,\&\, S_m,t,\ell\rangle$ or $\langle \Box(S_1 \,\&\, \ldots \,\&\, S_m),t,\ell\rangle$, where t is calculated

from the value of T and its multiplicity of $\bigcirc$, and ℓ is the value of L at which the resource can be consumed. Intuitionistic resources have the form $\langle S, 0, 0\rangle$, where S is a resource clause.

When the system executes $\vdash^{T}_{L,U} I\,\{G\}\,O$, the consumable resources in the context I have the following forms: $\langle S_1 \,\&\, \ldots \,\&\, S_m, T, L\rangle$, $\langle \Box(S_1 \,\&\, \ldots \,\&\, S_m), t, L\rangle$ where $t \leq T$, and $\langle S, 0, 0\rangle$.

The relation $pick^{T}_{L,U}(I, M, S)$ selects a consumable resource clause S from the input context I. The output context M is the same as I, except that the consumption level of the selected clause is changed to the value of U if it is a linear resource. The relation $change_{\ell,\ell'}(M, N)$ modifies the context M so that any resources in M with level ℓ have their level changed to ℓ' in the context N. The relation $thinable_{\ell}(O)$ checks whether none of resources in O have ℓ as their consumption level. The relation $subcontext^{T}_{U,L}(O, I)$ then consumes some resources. The output context O is the same as I, except that the consumption levels of some resources are changed to the value of U, if they are linear resources.

6 Implementation Design

In this section, we discuss implementation issues for the TLLP language.

For Lolli, Hodas has developed the I/O model-based interpreters both in Prolog and SML. Tamura et. al have designed an extension of standard WAM model (called LLPAM) and have developed a compiler system [2]. This compiler system supports the first-order Lolli language, except for the goal $\forall x.G$. LLPAM was first designed, based on *IOL* [10][15], and recently refined with the *top flag* in $\mathcal{LRM}$ [9], for the complete treatment of $\top$. LLPAM has also been improved to incorporate the resources compilation technique in [2].

The main differences between LLPAM and WAM is as follows:

- Three new date areas are added: `RES` (the resource table), `SYMBOL` (the symbol table), and `HASH` (the hash table). `HASH` and `SYMBOL` are used for speed access to resources, and the entries in `RES` are hashed on the predicate symbol and the value of the first argument in the current implementation.
- Six new registers are added: `R`, `L`, `U`, `T`, `R1`, and `R2`. `R` is the current top of `RES`. `L` and `U` are the current values of L (current consumption level) and U (current consumption maker) in *IOL* respectively. `T` is the *top flag* in $\mathcal{LRM}$. `R1` and `R2` are used for picking up consumable resources quickly.
- New instructions for newly added connectives are added.

For TLLP, there seem to be at least three approaches to develop efficient implementations: TLLP interpreter, translator from TLLP to Lolli, and TLLP compiler. First, it is easy to implement a TLLP interpreter based on *IOT* and *IOTL* in high-level languages like Prolog, but the resulting systems are slow. Secondly, it is possible to translate TLLP programs into Lolli programs by adding

[2] The latest package (version 0.5) including all source code can be obtained from `http://bach.seg.kobe-u.ac.jp/llp/`.

a new argument for the current time T in IOT to each predicate in Lolli. The drawback of this simple translation is that the goal $\top$ in TLLP can not be correctly translated into that in Lolli. Finally, it is also possible to extended LLPAM to support the TLLP language. We summarize important points that have been improved:

- Two new fields `time` and `box` have been added to each entry in `RES`. The `time` field denotes the consumption time in $IOTL$. The `box` flag is set to false if the newly added resource is not prefixed by $\Box$, otherwise true.
- A new register `TI` has been added. `TI` denotes the current time T in $IOTL$. Choice instructions such as *try* in WAM therefore need to set and restore the value of `TI`. `TI` is used to set the `time` field of newly added resource. `TI` is also used for hash key for speed access to the resources.
- In LLPAM, the instruction `add_res` $\mathtt{A}_i$, $\mathtt{A}_j$ is used to add linear resource clauses, where $\mathtt{A}_i$ is its head, $\mathtt{A}_j$ is its *closure* that consists of the compiled code and a set of bindings for free variables. We replaced this instruction with two new instructions `add_exact_timed_res` $\mathtt{A}_i$, $\mathtt{A}_j$, n and `add_timed_res` $\mathtt{A}_i$, $\mathtt{A}_j$, n. The former is used to add a resource clause S_i $(1 \leq i \leq m)$ in $\bigcirc^n(S_1 \,\&\, \cdots \,\&\, S_m)$, where $\mathtt{A}_i$ is its head, $\mathtt{A}_j$ is its closure, and n is the multiplicity of $\bigcirc$. The latter is used to add a resource clause S_i $(1 \leq i \leq m)$ in $\bigcirc^n \Box(S_1 \,\&\, \cdots \,\&\, S_m)$, $\mathtt{A}_i$ is its head, $\mathtt{A}_j$ is its closure, and n is the multiplicity of $\bigcirc$.
- In LLPAM, the instruction `pickup_resource` p/n, $\mathtt{A}_i$, L finds a consumable resource with predicate symbol p/n by checking its consumption level, and then it sets its index value to $\mathtt{A}_i$. If there are no consumable resources, it jumps to L. We need to improve this instruction so that it checks not only the level condition but also the time condition by comparing the consumption time (the `time` field) of resources with the current time (the current value of `TI`).

The specification of LLPAM have been shown in the papers [9] and [2].

7 Conclusion and Future Work

Recent development of logic programming languages based on linear logic suggests a successful direction to extend logic programming to be more expressive and more efficient. In this paper, we have designed the logic programming language TLLP based on intuitionistic temporal linear logic and have discussed some implementation issues for TLLP. The following points are still remaining:

- TLLP supports a small fragment of intuitionistic temporal linear logic.
- $IOTL$ needs to be refined with the idea of top flag in $\mathcal{LRM}$ [9] for the complete treatment of $\top$.
- The goal $\forall x.G$ is not supported in the current implementation.

Currently, we have developed a prototype of TLLP compiler system based on the extension presented in this paper. The latest TLLP package (version 0.1) including TLLP interpreters, a translator from TLLP into Lolli, and a prototype compiler is available from `http://kaminari.scitec.kobe-u.ac.jp/tllp/`.

References

1. Jean-Marc Andreoli and Remo Pareschi. Linear objects: Logical processes with built-in inheritance. *New Generation Computing*, 9:445–473, 1991.
2. Mutsunori Banbara and Naoyuki Tamura. Compiling Resources in a Linear Logic Programming Language. In Konstantinos Sagonas, editor, *Proceedings of the JICSLP'98 Post Conference Workshop 7 on Implementation Technologies for Programming Languages based on Logic*, pages 32–45, June 1998.
3. Iliano Cervesato, Joshua S. Hodas, and Frank Pfenning. Efficient resource management for linear logic proof search. In R. Dyckhoff, H. Herre, and P. Schroeder-Heister, editors, *Proceedings of the Fifth International Workshop on Extensions of Logic Programming — ELP'96*, pages 67–81, Leipzig, Germany, 28–30 March 1996. Springer-Verlag LNAI 1050.
4. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
5. James Harland and David Pym. A uniform proof-theoretic investigation of linear logic programming. *Journal of Logic and Computation*, 4(2):175–207, April 1994.
6. Takaharu Hirai. An Application of Temporal Linear Logic to Timed Petri Nets. In *Proceedings of the Petri Nets'99 Workshop on Applications of Petri Nets to Intelligent System Development*, pages 2–13, June 1999.
7. Joshua S. Hodas. *Logic Programming in Intuitionistic Linear Logic: Theory, Design and Implementation*. PhD thesis, University of Pennsylvania, Department of Computer and Information Science, 1994.
8. Joshua S. Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994. Extended abstract in the Proceedings of the Sixth Annual Symposium on Logic in Computer Science, Amsterdam, July 15–18, 1991.
9. Joshua S. Hodas, Kevin Watkins, Naoyuki Tamura, and Kyoung-Sun Kang. Efficient Implementation of a Linear Logic Programming Language. In Joxan Jaffar, editor, *Proceedings of the 1998 Joint International Conference and Symposium on Logic Programming*, pages 145–159. The MIT Press, June 1998.
10. Kyoung-Sun Kang, Mutsunori Banbara, and Naoyuki Tamura. Efficient resource management model for linear logic programming languages. *Computer Software*, 18(0):138–154, 2001. (in Japanese).
11. Max I. Kanovich and Takayasu Ito. Temporal linear logic specifications for concurrent processes (extended abstract). In *Proceedings of 12th Annual IEEE Symposium on Logic in Computer Science (LICS'97)*, pages 48–57, 1997.
12. Naoki Kobayashi and Akinori Yonezawa. ACL — A concurrent linear logic programming paradigm. In D. Miller, editor, *Proceedings of the 1993 International Logic Programming Symposium*, pages 279–294, Vancouver, Canada, October 1993. MIT Press.
13. Dale Miller. A multiple-conclusion specification logic. *Theoretical Computer Science*, 165(1):201–232, 1996.
14. Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.
15. Naoyuki Tamura and Yukio Kaneda. Extension of WAM for a linear logic programming language. In T. Ida, A. Ohori, and M. Takeichi, editors, *Second Fuji International Workshop on Functional and Logic Programming*, pages 33–50. World Scientific, Nov. 1996.
16. Makoto Tanabe. Timed petri nets and temporal linear logic. In *Lecture Notes in Computer Science 1248: Proceedings of Application and Theory of Petri Nets*, pages 156–174, June 1997.