古典線形論理型プログラミング言語の静的解析の一手法について

姜 京順[†] 番原睦則[††] 田村直之[†††]

本稿では古典線形論理に基づいた論理型プログラミング言語の静的解析方法を提案する．線形論理に基づく論理型プログラミング言語としては Lolli，LinLog，LO，Lygon，Forum などが提案されている．特に，Forum は古典線形論理に対し完全である論理型プログラミング言語である．しかし，Forum のプログラムの証明探索は非決定性が非常に大きいため，静的解析することでプログラムの実行前に証明不可能なシーケントを取り除くことは重要である．

Andreoli らは古典線形論理に基づく論理型プログラミング言語 LO の静的解析方法を提案している．しかし，その方法は「リーソスを意識した」線形論理型プログラミング言語で最も有用な演算子である乗法的論理積の解析には適用できないのである．

そこで，我々は乗法的論理和，含意に加えて乗法的論理積を解析できる静的解析方法を考案した．我々の方法ではまず，プログラムと証明すべきゴールが与えられたら，そのシーケントのあり得る全ての証明を 1 つの AND-OR グラフに表現する．そして，データフロー解析のための前方伝搬と後方伝搬を繰り返すことによって証明不可能な節点をグラフから取り除いていく．本稿の静的解析方法に基づくプロトタイプ解析機を開発し，実際に Forum で書かれたソーティングプログラムを解析した結果，6 要素のソーティングに対し，解析していない時の実行速度に比べ 1000 倍以上速くなった．

A Static Analysis Method for a Classical Linear Logic Programming Language

Kyoung-Sun Kang,[†] Mutsunori Banbara[††] and Naoyuki Tamura[†††]

We propose a new static analysis method applicable for a classical linear logic programming language.

Andreoli et al. proposed a static analysis method for a classical linear logic programming language LO, but their method can not handle multiplicative connectives, which can be seen as the most important connective for a resource-sensitive feature of linear logic.

Our method, in contract, covers multiplicative conjunctions in addition to multiplicative disjunctions and linear implications. Abstract proof graph, which is an AND-OR graph representing all possible sequent proofs, is constructed from a given program and goal sequent. The graph can be repeatedly refined by propagating information to remove unprovable nodes from the graph.

We applied the result of our prototype analyzer for a sorting program written in Forum. The sorting program was improved more than 1000 times faster for sorting 6 elements.

1. はじめに

論理型プログラミング言語でのプログラムの実行はシーケント計算体系での証明探索に対応していると考えることができる．実行の 1 ステップはシーケント計算体系において規則を下から上に適用することに対応している．この考え方は直観主義論理，古典論理，線形論理 (linear logic) など様々な論理体系に適用することのできる広い概念である．

線形論理 [4] は 1987 年に論理学者 J.-Y.Girard によって発表された新しい論理であり，論理概念の中に数量概念が入っていることから「リソースを意識した論理 (resource-conscious logic)」とも呼ばれている．近年，線形論理に基づく論理型プログラミング言語について盛んに研究されている．これまでに Lolli[7]，Lygon[5]，LO[2]，LinLog[1]，Forum[13]，HACL[11] な

† 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
††† 神戸大学工学部
Faculty of Engineering, Kobe University

どの言語が提案されている．

特に，Forum は古典線形論理に対して完全である論理型プログラミング言語である．Forum では，プログラムの実行は uniform proof[14] と呼ばれる “goal-directed” な証明の探索に対応している．この uniform proof は Miller が論理型プログラミングの理論上の基礎として提案した概念である．Uniform proof は推論規則 “cut” がない証明で右側の論理式 (ゴール) が原子論理式でないすべてのシーケントが右推論規則の結論になっている証明をいう．すなわち，ボトムアップに uniform proof を探索して行くときにはシーケントの右にあるゴールが原子論理式ではなかったらそのゴールを導くような右規則を適用し，ゴールが原子論理式になったときだけ左推論規則を適用して証明探索すればよいことになる．したがって，uniform proof は goal-directed なる証明だと言える．この goal-directed という概念は論理型言語を設計する上で非常に重要であり，実際に Prolog，λProlog，Lolli が goal-directed の概念をベースにしている．

しかし，Forum での証明探索は非常に非決定性が大きい．例えば，Forum で書かれたソーティングプログラム (第 6 章参照) の実行時間は，リストの長さの指数乗に比例して増加していく[8]．これは，実行時間の大部分が証明不可能なシーケントを証明する試みのために費されているからである．そのため，プログラムの実行前に証明不可能なシーケントを見つけることは非常に重要である．このアイデアはプログラムを静的解析することにより実現することができる．もちろん，線形論理の証明可能性は命題線形論理ですでに決定不能 (undecidable) であることが知られているため，証明不可能であるシーケントをすべてを取り除くのは不可能である．

Andreoli らは古典線形論理に基づいた論理型プログラミング言語 LO の静的解析方法[3] を提案している．しかし，Andreoli の解析方法では，リーソスの遅延分割を証明体系に取り入れていないため，線形論理の乗法的論理積 (⊗) を解析することができない☆．この演算子 ⊗ は,「リーソスを意識した」線形論理型プログラミング言語において最も有用な演算子である．

本稿では乗法的論理和 (℘)，含意 (—∘) に加えて乗法的論理積 (⊗) も解析できる静的解析方法を提案する．乗法的論理積の解析には，証明体系に Hodas と Miller が提案している I/O model[6),7)] を導入した．この I/O model はリーソス分割の非決定性を解消するために考案されたリソース消費モデルである．

本稿で述べる解析方法は，最初に与えられたプログラムとゴールから，シーケント計算体系においてあり得るすべての証明を表す 1 つの AND-OR グラフ (以降，抽象証明グラフと呼ぶ) を作成する．その後，データフロー解析のための解析の前方伝搬，後方伝搬を繰り返すことによって抽象証明グラフから証明不可能なノードを取り除いていく．実際に Forum のソーティングプログラムを本稿で述べる方法で解析した結果，6 要素のソーティングに対し，解析していない時の実行速度に比べ 1000 倍以上速くなった．

2. 古典線形論理型プログラミング言語

この節では，線形論理のシーケント計算体系 **LL** とその I/O model である **IO** について述べる．

2.1 言　　語

本稿で扱う言語は比較的小さいが，これは静的解析の議論をできるだけ簡潔に述べるためである．事実，本稿で述べる言語は 1 階 (first-order) の Forum[9] に簡単に拡張可能である．

定義 2.1.1 (節) 節とは

$$\forall \vec{x}(A_1 \wp \cdots \wp A_n :- G) \quad (\text{ただし } n \geq 1)$$

という形をした表現である．ただし，G は $\bot$ または **1** または A または $B \wp C$ または $B —\circ C$ または $B \otimes C$ である．また A, B, C, $A_i (1 \leq i \leq n)$ は原子論理式を表すものとする． □

定義 2.1.1 で演算子 “ :− ” は含意 (—∘) の逆を意味する．すなわち $A :- G$ は $G —\circ A$ と同値である．

定義 2.1.2 (シーケント) シーケントとは

$$\Psi; \Delta \longrightarrow \mathcal{A}$$

という形をした表現である．ただし，Ψ は節の集合であり，Δ と $\mathcal{A}$ は原子論理式のマルチ集合である． □

定義 2.1.2 において，Ψ 中の節にはすべてモダリティ “!” が付いていると考えることができる．すなわち，本稿で述べるシーケント $\Psi; \Delta \longrightarrow \mathcal{A}$ は，線形論理の標準的なシーケント計算体系の $!\Psi, \Delta \longrightarrow \mathcal{A}$ と同値である．**図 1** にシーケント計算体系 **LL** の命題論理部分を示す．

2.2 IO: LL のための I/O model

線形論理において，証明をボトムアップに探索をする場合，最も問題となるのがシーケントの右側 (ゴール) が $G_1 \otimes G_2$ の時である．この規則をボトムアップに適用する際，Δ_1 と Δ_2 の分け方は非常に非決定性が大きい．

☆ 事実，彼らは論文 3) において，リーソス遅延分割を導入した解析方法の開発を今後の課題としている

$$\frac{}{\Psi; A \longrightarrow A}\ (Id) \qquad \frac{\Psi; \Delta_1 \longrightarrow B, \mathcal{A}_1 \quad \Psi; \Delta_2 \longrightarrow C, \mathcal{A}_2}{\Psi; \Delta_1, \Delta_2 \longrightarrow A_1, \cdots, A_n, \mathcal{A}_1, \mathcal{A}_2}\ (\otimes)$$
$$(provided\ A_1 \wp \cdots \wp A_n :- B \otimes C \in \Psi)$$

$$\frac{}{\Psi; \longrightarrow A_1, \cdots, A_n}\ (\mathbf{1}) \qquad \frac{\Psi; \Delta \longrightarrow B, C, \mathcal{A}}{\Psi; \Delta \longrightarrow A_1, \cdots, A_n, \mathcal{A}}\ (\wp)$$
$$(provided\ A_1 \wp \cdots \wp A_n :- \mathbf{1} \in \Psi) \qquad (provided\ A_1 \wp \cdots \wp A_n :- B \wp C \in \Psi)$$

$$\frac{\Psi; \Delta \longrightarrow \mathcal{A}}{\Psi; \Delta \longrightarrow A_1, \cdots, A_n, \mathcal{A}}\ (\bot) \qquad \frac{\Psi; B, \Delta \longrightarrow C, \mathcal{A}}{\Psi; \Delta \longrightarrow A_1, \cdots, A_n, \mathcal{A}}\ (\multimap)$$
$$(provided\ A_1 \wp \cdots \wp A_n :- \bot \in \Psi) \qquad (provided\ A_1 \wp \cdots \wp A_n :- B \multimap C \in \Psi)$$

図 1 証明システム **LL**.
Fig. 1 The proof system **LL**.

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

Hodas と Miller はこの問題の解決法として，直観主義線形論理型言語 Lolli[7] に対して，IO-model と呼ばれるリソース消費モデルを提案した．この IO-model は，リソース分割を遅延することにより分割の非決定性を解決している．

$$\frac{\Gamma; \Delta_I \backslash \Delta' \longrightarrow G_1 \quad \Gamma; \Delta' \backslash \Delta_O \longrightarrow G_2}{\Gamma; \Delta_I \backslash \Delta_O \longrightarrow G_1 \otimes G_2}\ \otimes$$

Lolli の IO-model では，ゴールをリソースを消費する消費者として考える．例えば，$G_1 \otimes G_2$ 証明する場合，まずゴール G_1 を証明するために，入力リソース Δ_1, Δ_2 が与えられ，その後，G_1 の出力リソース (G_1 で消費されなかったリソース) Δ_2 がゴール G_2 を証明するための入力リソースとして与えられる．上の IO-model の $\otimes$ 規則において，“\” の左側がゴール G_i の入力リソース，右側が出力リソースを表している．Hodas はこのアイデアをシーケントの右側にも適用することによって，古典線形論理型プログラミング言語 Forum のための I/O model[8] を提案した．本稿では，これと同様の方法で証明システム **LL** の I/O model である **IO** を提案する (**図 2**).

定義 2.2.1 **IO** のシーケントとは

$\Psi; \Delta_I \backslash \Delta_O \longrightarrow \mathcal{A}_I \backslash \mathcal{A}_O$

という形をした表現である．ただし，Ψ は節の集合であり，Δ_I, Δ_O, $\mathcal{A}_I$, $\mathcal{A}_O$ は原子論理式のマルチ集合 (multiset) である． □

命題 2.1 **LL** において $\Psi; \Delta \longrightarrow \mathcal{A}$ が証明可能であることは，**IO** において $\Psi; \Delta, \Delta_O \backslash \Delta_O \longrightarrow \mathcal{A}, \mathcal{A}_O \backslash \mathcal{A}_O$ が証明可能であることと同値である．

(証明) Hodas の論文 9) と同じ方法で証明できる． ∎

第 1 章で述べた Forum で書かれたリストのソーティングプログラム☆とその実行に対応する **LL** での証明図と **IO** での証明図を付録に示す.

3. シーケントの抽象化

第 1 章で述べたように論理型プログラミング言語でのプログラムの実行はシーケント計算の体系での証明探索に対応していると考えることができる．

従来の命題論理ではシーケントが与えられた時，そのシーケントの証明中に現れる可能性があるシーケントの数は有限である．それは subformula property と論理積 ($\wedge$) と論理和 ($\vee$) に関する巾等律 (idempotent law) が成り立つからである．

一方，線形論理では論理式の数は有限であるが，シーケントの数は無限である．それは乗法的論理積 ($\otimes$) と乗法的論理和 ($\wp$) に関して巾等律が成り立たないからである．すなわち，線形論理ではシーケント $A, A \longrightarrow B, B$ と $A \longrightarrow B$ は同値ではない．線形論理の証明可能性に関しては，命題線形論理ですでに決定不能であることが知られている [12].

そのため，線形論理に対する抽象証明探索 (abstract proof search) を行うためにはシーケントの集合を有限集合に写像する必要がある．もちろん写像前と写像後では，シーケントの証明可能性は同値ではない．しかし，本稿の目的は証明不可能なシーケントを静的に検出することである．よって，我々はシーケントの証明可能性の性質を失わない，すなわち，写像前のシーケントが証明可能であれば，必ずその写像後のシーケントも証明可能であるような性質をもつ写像を定義していく．

証明すべきシーケント $\Psi; \Delta_I \backslash \Delta_O \longrightarrow \mathcal{A}_I \backslash \mathcal{A}_O$ が

☆ 本論文の言語で記述しても同じプログラムである.

$$\frac{}{\Psi; A, \Delta\backslash\Delta \longrightarrow A, \mathcal{A}\backslash\mathcal{A}}\ (Id) \qquad \frac{\Psi; \Delta_I\backslash\Delta' \longrightarrow B, \mathcal{A}_I\backslash\mathcal{A}' \quad \Psi; \Delta'\backslash\Delta_O \longrightarrow C, \mathcal{A}'\backslash\mathcal{A}_O}{\Psi; \Delta_I\backslash\Delta_O \longrightarrow A_1, \cdots A_n, \mathcal{A}_I\backslash\mathcal{A}_O}\ (\otimes)$$
$$(provided\ A_1 \wp \cdots \wp A_n : -B \otimes C \in \Psi\ and\ \mathcal{A}_O \subseteq \mathcal{A}' \subseteq \mathcal{A}_I)$$

$$\frac{}{\Psi; \Delta\backslash\Delta \longrightarrow A_1, \cdots A_n, \mathcal{A}\backslash\mathcal{A}}\ (\mathbf{1}) \qquad \frac{\Psi; \Delta_I\backslash\Delta_O \longrightarrow B, C, \mathcal{A}_I\backslash\mathcal{A}_O}{\Psi; \Delta_I\backslash\Delta_O \longrightarrow A_1, \cdots A_n, \mathcal{A}_I\backslash\mathcal{A}_O}\ (\wp)$$
$$(provided\ A_1 \wp \cdots \wp A_n : -\mathbf{1} \in \Psi) \qquad (provided\ A_1 \wp \cdots \wp A_n : -B \wp C \in \Psi)$$

$$\frac{\Psi; \Delta_I\backslash\Delta_O \longrightarrow \mathcal{A}_I\backslash\mathcal{A}_O}{\Psi; \Delta_I\backslash\Delta_O \longrightarrow A_1, \cdots A_n, \mathcal{A}_I\backslash\mathcal{A}_O}\ (\bot) \qquad \frac{\Psi; B, \Delta_I\backslash\Delta_O \longrightarrow C, \mathcal{A}_I\backslash\mathcal{A}_O}{\Psi; \Delta_I\backslash\Delta_O \longrightarrow A_1, \cdots A_n, \mathcal{A}_I\backslash\mathcal{A}_O}\ (\multimap)$$
$$(provided\ A_1 \wp \cdots \wp A_n : -\bot \in \Psi) \qquad (provided\ A_1 \wp \cdots \wp A_n : -B \multimap C \in \Psi)$$

図 2 証明システム **IO** .
Fig. 2 The proof system **IO** .

与えられると，まず与えられたシーケント (以降，ゴールシーケントと呼ぶ) から量化記号とすべての原子論理式の引数を取り除いたシーケントに写像する．

集合 **F** は命題シーケント中に出現するすべての論理式の部分論理式から構成される集合である．集合 **F** 上のマルチ集合は写像 $\mathbf{F} \longrightarrow \mathbf{N}$ で定義することができる．ただし，**N** は非負整数の集合である．

定義 3.1 (M-表現) $\langle M, +, 0\rangle$ が非負整数の集合 **N** と準同型である有限モノイドとする．すなわち，次の条件を満たす写像 $\phi : \mathbf{N} \longrightarrow M$ が存在する．

$$\phi(0) = 0$$
$$\phi(x + y) = \phi(x) + \phi(y) \quad \text{for all } x,\ y \in \mathbf{N}$$

X は **F** 上でのマルチ集合である．そのとき合成写像 $\phi \circ X$ を X の M-**表現** (M-representation) であるという． □

例題 3.1 集合 $M_n = \{0, 1, 2, \cdots, n\}$ に対し，有限モノイド $\langle M_n, +, 0\rangle$ は次の写像 ϕ により，**N** に準同型になる．

$$\phi(x) = \begin{cases} x & \text{if } x < n \\ n & \text{if } x \geq n \end{cases}$$

□

M-表現を利用して，マルチ集合に出現する論理式の数を有限的に数えることによってマルチ集合を有限的に表すことができる．我々は M-表現を導入したシーケント計算体系を定義する．

定義 3.2 (M-マルチ集合, M-シーケント) 写像 $\mathbf{F} \longrightarrow M$ を **F** 上の M-**マルチ集合** (M-multiset) という．また，M-**シーケント** (M-sequent) とは

$$\Psi; \Delta_I\backslash\Delta_O \longrightarrow \mathcal{A}_I\backslash\mathcal{A}_O$$

という形をした表現である．ただし，Ψ は **F** の元の集合，Δ_I, Δ_O, $\mathcal{A}_I$, A_O は各々**F** 上の M-マルチ集合である． □

F 上での M-マルチ集合を M 数が右上に書いてある論理式の列 (場合によって中カッコでくくる) で表現する．ただし，その右肩添字が 0 である論理式は省略してもよい．例えば，M-マルチ集合 $\{a^2, (a \otimes b)^1\}$ はマルチ集合 $\{a, a, a \otimes b\}$ の M_2-表現☆である．

定義 3.3 M-シーケント $\Psi; \Delta_I\backslash\Delta_O \longrightarrow \mathcal{A}_I\backslash\mathcal{A}_O$ 上での証明システム $\mathbf{IO}_M$ は**図 3** のように定義できる．ここで，$\uplus$ は M-マルチ集合の和集合を意味する．(例, $\{a^{m_1}, b^{m_1}\} \uplus \{b^{m_2}, c^{m_2}\} = \{a^{m_1}, b^{m_1+m_2}, c^{m_2}\}$) □

例題 3.2 プログラム Ψ が $a \wp a$:− $a \otimes b$ であり，ゴールシーケント S_1 が $\Psi; a, b\backslash \longrightarrow a, a\backslash$ のときの **IO** の証明と $\mathbf{IO}_{M_2}$ の証明 (モノイド M_2 での証明) を示す．

$$\dfrac{\dfrac{}{\Psi; a, b\backslash b \longrightarrow a\backslash b}\ (Id) \quad \dfrac{}{\Psi; b\backslash \ \longrightarrow b\backslash}\ (Id)}{\Psi; a, b\backslash \ \longrightarrow a, a\backslash}\ (\otimes)$$

$$\dfrac{\dfrac{}{\Psi; a^1, b^1\backslash b^1 \longrightarrow a^1\backslash b^1}\ (Id) \quad \dfrac{}{\Psi; b^1\backslash \ \longrightarrow b^1\backslash}\ (Id)}{\Psi; a^1, b^1\backslash \ \longrightarrow a^2\backslash}\ (\otimes)$$

□

IO システムの証明で，シーケント中の論理式の出現回数だけが違うものが複数存在する場合，それらの証明は，$\mathbf{IO}_M$ では同じ証明になる可能性が高い．

命題 3.1 Δ'_I, Δ'_O, $\mathcal{A}'_I$, $\mathcal{A}'_O$ が各々Δ_I, Δ_O, $\mathcal{A}_I$, $\mathcal{A}_O$ の M-表現であるとき，**IO** において $\Psi; \Delta_I\backslash\Delta_O \longrightarrow \mathcal{A}_I\backslash\mathcal{A}_O$ 証明可能であれば，$\mathbf{IO}_M$ において $\Psi; \Delta'_I\backslash\Delta'_O \longrightarrow \mathcal{A}'_I\backslash\mathcal{A}'_O$ が証明可能である．

(証明) 証明構造上での帰納法で証明できる． ∎

命題 3.1 の対偶を考えた場合，ゴールシーケントが

☆ 例題 3.1 での $M_2 = \{0, 1, 2\}$ のときである．

$$\frac{}{\Psi;\{A^1\}\uplus\Delta\backslash\Delta\longrightarrow\{A^1\}\uplus\mathcal{A}\backslash\mathcal{A}}\ (Id) \qquad \frac{\Psi;\Delta_I\backslash\Delta'\longrightarrow\{B^1\}\uplus\mathcal{A}_I\backslash\mathcal{A}'\quad\Psi;\Delta'\backslash\Delta_O\longrightarrow\{C^1\}\uplus\mathcal{A}'\backslash\mathcal{A}_O}{\Psi;\Delta_I\backslash\Delta_O\longrightarrow\{A_1{}^1,\cdots,A_n{}^1\}\uplus\mathcal{A}_I\backslash\mathcal{A}_O}\ (\otimes)$$
$$(provided\ A_1\wp\cdots\wp A_n:-B\otimes C\in\Psi\ and\ \mathcal{A}_O\subseteq\mathcal{A}'\subseteq\mathcal{A}_I)$$

$$\frac{}{\Psi;\Delta\backslash\Delta\longrightarrow\{A_1{}^1,\cdots,A_n{}^1\}\uplus\mathcal{A}\backslash\mathcal{A}}\ (\mathbf{1}) \qquad \frac{\Psi;\{B^1\}\uplus\Delta_I\backslash\Delta_O\longrightarrow\{C^1\}\uplus\mathcal{A}_I\backslash\mathcal{A}_O}{\Psi;\Delta_I\backslash\Delta_O\longrightarrow\{A_1{}^1,\cdots,A_n{}^1\}\uplus\mathcal{A}_I\backslash\mathcal{A}_O}\ (\multimap)$$
$$(provided\ A_1\wp\cdots\wp A_n:-\mathbf{1}\in\Psi) \qquad (provided\ A_1\wp\cdots\wp A_n:-B\multimap C\in\Psi)$$

$$\frac{\Psi;\Delta_I\backslash\Delta_O\longrightarrow\mathcal{A}_I\backslash\mathcal{A}_O}{\Psi;\Delta_I\backslash\Delta_O\longrightarrow\{A_1{}^1,\cdots,A_n{}^1\}\uplus\mathcal{A}_I\backslash\mathcal{A}_O}\ (\bot) \qquad \frac{\Psi;\Delta_I\backslash\Delta_O\longrightarrow\{B^1,C^1\}\uplus\mathcal{A}_I\backslash\mathcal{A}_O}{\Psi;\Delta_I\backslash\Delta_O\longrightarrow\{A_1{}^1,\cdots,A_n{}^1\}\uplus\mathcal{A}_I\backslash\mathcal{A}_O}\ (\wp)$$
$$(provided\ A_1\wp\cdots\wp A_n:-\bot\in\Psi) \qquad (provided\ A_1\wp\cdots\wp A_n:-B\wp C\in\Psi)$$

図 3 証明システム $\mathbf{IO}_M$.
Fig. 3 The proof system $\mathbf{IO}_M$.

与えられた時，その M-シーケントがシステム $\mathbf{IO}_M$ で証明不可能であれば，そのゴールシーケントもシステム **IO** で証明不可能であることがわかる．

命題 3.2 証明システム $\mathbf{IO}_M$ での証明可能性は決定可能である．

(証明) 証明システム $\mathbf{IO}_M$ で証明探索するとき，M-シーケントの数は有限なので証明可能性は決定可能である． ∎

4. 抽象証明グラフ

前章で提案したシステム $\mathbf{IO}_M$ での証明可能性は理論的に決定可能である．しかし，$\mathbf{IO}_M$ での純粋な証明探索は非効率的であり，実際に大きいプログラムに適用するには無理がある．

特に，乗法的論理積 $\otimes$ を実行する時，リソースの遅延分割のため，規則の上側に現れる得るシーケントの数が非常に多くなり，証明検索の効率低下の原因となる．以下に $\mathbf{IO}_{M_2}$ の証明例を示す (ただし，$a:-b\otimes c\in\Psi$ とする)．

$$\frac{\Psi;\ \backslash\ \longrightarrow b^1,d^2,e^2\backslash\Delta\qquad\Psi;\ \backslash\ \longrightarrow c^1,\Delta\backslash}{\Psi;\ \backslash\ \longrightarrow a^1,d^2,e^2\backslash}\ (\otimes)$$

ここで M_2-マルチ集合 Δ には $\emptyset$, $\{e^1\}$, $\{e^2\}$, $\{d^1\}$, $\{d^2\}$, $\{d^1,e^1\}$, $\{d^1,e^2\}$, $\{d^2,e^1\}$, $\{d^2,e^2\}$ の 9 つの場合がある．この問題を解決し，効率のよい証明探索を実現するために，M-マルチ集合のベキ集合 (power set) を利用する．これにより，複数の M-マルチ集合をただ 1 つのベキ集合で表すことができる．

$\mathcal{P}(M)$ は M のベキ集合であり，また演算子 $+$ を次のように定義する．

$$X+Y=\{x+y\mid x\in X,y\in Y\}$$

そのとき $\langle\mathcal{P}(M),+,\{0\}\rangle$ は有限モノイドになり，また $\langle\mathcal{P}(M),\cap,\cup\rangle$ は完全束になる．そして，演算子 $-$ も次のように定義する．

$$X-Y=\{z\in M\mid x\in X,y\in Y,x=y+z\}$$

表 1 $\mathcal{P}(M_2)$ の表記法.
Table 1 A denotation for $\mathcal{P}(M_2)$.

Notation	Stands For	Meaning
B	$B^{\{1\}}$	one B
B^2	$B^{\{2\}}$	two or more B's
B'	$B^{\{0,1\}}$	at most one B
B''	$B^{\{0,2\}}$	zero or more than one B's
B^+	$B^{\{1,2\}}$	one or more B's
B^*	$B^{\{0,1,2\}}$	any number of B's

定義 4.1 ($\mathcal{P}(M)$-表現，$\mathcal{P}(M)$-マルチ集合) X は $\mathbf{F}$ 上のマルチ集合であり，ψ は X の M-表現とする．また，X の $\mathcal{P}(M)$-**表現** ($\mathcal{P}(M)$-representation) は $[X]$ で示す．$[X]$ は次のように定義される写像 $\mathbf{F}\longrightarrow\mathcal{P}(M)$ である．

$$[X](B)=\{\psi(B)\}\ (\text{for all } B\in\mathbf{F})$$

また，$\mathbf{F}$ 上での $\mathcal{P}(M)$-**マルチ集合** ($\mathcal{P}(M)$-multiset) は写像 $\mathbf{F}\longrightarrow\mathcal{P}(M)$ である． □

$\mathbf{F}$ 上の $\mathcal{P}(M)$-マルチ集合を $\mathcal{P}(M)$ 数が右上に書いてある論理式の列で表現する．ただし，右肩添字が $\{0\}$ である論理式は省略してもよい．特に，第 6 章で述べる静的解析のプロトタイプのベースになる $\mathcal{P}(M_2)$ に関しては**表 1** のような表記法を使う．

モノイド M が大きい程，静的解析よって精密に解析できるが，そのコストは大きくなる．

$\mathcal{P}(M)$-マルチ集合は集合 $\mathbf{F}$ 中の各々の論理式 B に対してあり得る M 値の集合である．すなわち，$\mathcal{P}(M)$-マルチ集合は M-マルチ集合の集合を示す．

定義 4.2 X を $\mathcal{P}(M)$-マルチ集合とすると，$\|X\|$ は次のように定義される M-マルチ集合の集合である．

$$\|X\| = \{Y \mid Y \text{ is an } M\text{-multiset s.t } Y(B) \in X(B) \text{ for all } B \in \mathbf{F}\}$$

□

例題 4.1 $\mathcal{P}(M_2)$ において，
$X = \{a', b^*\}(= \{a^{\{0,1\}}, b^{\{0,1,2\}}\})$ とすると，$\|X\| = \{\{b^1\}, \{b^2\}, \{a^1, b^1\}, \{a^1, b^2\}\}$ になる． □

定義 4.3 X と Y を $\mathbf{F}$ 上での $\mathcal{P}(M)$-マルチ集合とすると，$X+Y$, $X-Y$, $X \cap Y$, $X \cup Y$ は各々すべての $B \in \mathbf{F}$ に対して次を満たすような $\mathcal{P}(M)$-マルチ集合である．

$$(X+Y)(B) = X(B) + Y(B)$$
$$(X-Y)(B) = X(B) - Y(B)$$
$$(X \cap Y)(B) = X(B) \cap Y(B)$$
$$(X \cup Y)(B) = X(B) \cup Y(B)$$

□

例題 4.2 $\mathcal{P}(M_2)$-マルチ集合に対して，

$$\begin{aligned}
\{a', b^*\} + \{a, b^2\} &= \{a^{\{0,1\}}, b^{\{0,1,2\}}\} + \{a^{\{1\}}, b^{\{2\}}\} \\
&= \{a^{\{0,1\}+\{1\}}, b^{\{0,1,2\}+\{2\}}\} \\
&= \{a^{\{1,2\}}, b^{\{2\}}\} \\
&= \{a^+, b^2\} \\
\{a', b^*\} - \{a, b^2\} &= \{a^{\{0,1\}}, b^{\{0,1,2\}}\} - \{a^{\{1\}}, b^{\{2\}}\} \\
&= \{a^{\{0,1\}-\{1\}}, b^{\{0,1,2\}-\{2\}}\} \\
&= \{a^{\{0\}}, b^{\{0,1,2\}}\} \\
&= \{b^*\} \\
\{a', b^*\} \cap \{a, b^2\} &= \{a^{\{0,1\}}, b^{\{0,1,2\}}\} \cap \{a^{\{1\}}, b^{\{2\}}\} \\
&= \{a^{\{0,1\}\cap\{1\}}, b^{\{0,1,2\}\cap\{2\}}\} \\
&= \{a^{\{1\}}, b^{\{2\}}\} \\
&= \{a, b^2\} \\
\{a', b^*\} \cup \{a, b^2\} &= \{a^{\{0,1\}}, b^{\{0,1,2\}}\} \cup \{a^{\{1\}}, b^{\{2\}}\} \\
&= \{a^{\{0,1\}\cup\{1\}}, b^{\{0,1,2\}\cup\{2\}}\} \\
&= \{a^{\{0,1\}}, b^{\{0,1,2\}}\} \\
&= \{a', b^*\}
\end{aligned}$$

□

定義 4.4 ($\mathcal{P}(M)$-シーケント) Ψ は $\mathbf{F}$ 上での集合であり，Δ_I, Δ_O, $\mathcal{A}_I$, $\mathcal{A}_O$ は $\mathbf{F}$ 上での $\mathcal{P}(M)$-マルチ集合とする．このとき $\mathcal{P}(M)$-**シーケント** ($\mathcal{P}(M)$-sequent) とは

$$\Psi; \Delta_I \backslash \Delta_O \longrightarrow \mathcal{A}_I \backslash \mathcal{A}_O$$

という形の表現である． □

定義 4.5 $S \equiv \Psi; \Delta_I \backslash \Delta_O \longrightarrow \mathcal{A}_I \backslash \mathcal{A}_O$ を $\mathcal{P}(M)$-シーケントとすると，$\|S\|$ は次のような M-シーケントの集合である．

$$\|S\| = \{\Psi; \Delta'_I \backslash \Delta'_O \longrightarrow \mathcal{A}'_I \backslash \mathcal{A}'_O \mid \Delta'_I \in \|\Delta_I\|, \Delta'_O \in \|\Delta_O\|, \mathcal{A}'_I \in \|\mathcal{A}_I\|, \mathcal{A}'_O \in \|\mathcal{A}_O\|\}$$

□

定義 4.6 V を節点 (node) の集合，E を AND 弧 (AND arc) の集合 (すなわち $E \subseteq V \cup V^2 \cup V^3$)，$v_0 \in V$ をルート節点 (root node) とするとき，組 $\langle V, E, v_0 \rangle$ をAND-OR **グラフ**☆と呼ぶ． □

組 $\langle v, v_1, v_2 \rangle \in E$ は AND 弧が v から v_1 と v_2 の両方に出ているを意味する．また，対 $\langle v, v_1 \rangle \in E$ は v から v_1 に弧が出ていることを意味し，$\langle v \rangle \in E$ は v から出て行き先の節点がない弧を意味する．同じ節点 v からの複数の弧は OR 選択を意味している．

定義 4.7 $G \equiv \langle V, E, v_0 \rangle$ を AND-OR グラフとする．また，T を節点の集合 U をもち，そのルート節点が $u_0 \in U$ であるような木であるとする．このとき，次の条件を満たす写像 $f : U \longrightarrow V$ が存在するならば，**木 T はグラフ G に含まれている**という．

(1) $f(u_0) = v_0$

(2) 子節点 (child nodes) $u_1, \cdots, u_n$ をもつすべての節点 $u \in U$ に対して，
$\langle f(u), f(u_1), \cdots, f(u_n) \rangle \in E$ ($n = 1$ *or* 2)

(3) すべての葉節点 (leaf node)$u \in U$ に対して，
$\langle f(u) \rangle \in E$ □

たとえ AND-OR グラフが有限であっても，サイクルをもつ可能性があるため，そのグラフ中には無限に多くの木を含む可能性がある．

定義 4.8 任意の節点 $v \in V$ が $\mathcal{P}(M)$-シーケントであるとき，AND-OR グラフ $\langle V, E, v_0 \rangle$ を**証明グラフ** (proof graph) と呼ぶ． □

定義 4.9 G を証明グラフとするとき，Π を $\mathbf{IO}_M$ の証明とするとき，次の条件を満たすならば $\mathbf{IO}_M$ **の証明** Π **は** G **に含まれている**という．

(1) 証明 Π を表す木が写像 f によって G に含まれている．

(2) 証明 Π の任意の M-シーケント S に対して，
$S \in \|f(S)\|$. □

定義 4.10
G をシーケント $\Psi; \Delta_I \backslash \Delta_O \longrightarrow \mathcal{A}_I \backslash \mathcal{A}_O$ の証明グラフとするとき，シーケント $\Psi; \Delta_I \backslash \Delta_O \longrightarrow \mathcal{A}_I \backslash \mathcal{A}_O$ のすべての $\mathbf{IO}_M$ 証明が G に含まれているならば，G を**抽象証明グラフ** (abstract proof graph) と呼ぶ． □

例題 4.3 Ψ が次のプログラムを表しているとする．

$$s \wp s \quad :- \quad s \otimes m$$
$$m \quad :- \quad \mathbf{1}$$

このとき，**図 4** は M_1-シーケント $\Psi; s^1 \backslash \longrightarrow s^1 \backslash$ に対する $\mathbf{IO}_{M_1}$ 証明の 1 部である．また，**図 5** は図 4 の 3 つの $\mathbf{IO}_{M_1}$ 証明に対する抽象証明グラフである．ただし，図 5 においてプログラム Ψ は省略している．
□

次に，ゴール $\mathcal{P}(M)$-シーケントが与えられていた

☆ 本稿の AND-OR グラフの定義は標準的なものとは異なる

$$\frac{}{\Psi; s^1\backslash \longrightarrow s^1\backslash}$$

$$\frac{\dfrac{}{\Psi; s^1\backslash \longrightarrow s^1\backslash} \quad \dfrac{}{\Psi; \backslash \longrightarrow m^1\backslash}}{\Psi; s^1\backslash \longrightarrow s^1\backslash}$$

$$\frac{\dfrac{\dfrac{}{\Psi; s^1\backslash \longrightarrow s^1\backslash} \quad \dfrac{}{\Psi; \backslash \longrightarrow m^1\backslash}}{\Psi; s^1\backslash \longrightarrow s^1\backslash} \quad \dfrac{}{\Psi; \backslash \longrightarrow m^1\backslash}}{\Psi; s^1\backslash \longrightarrow s^1\backslash}$$

図 4 $\mathbf{IO}_{M_1}$ 証明.
Fig. 4 $\mathbf{IO}_{M_1}$ proof.

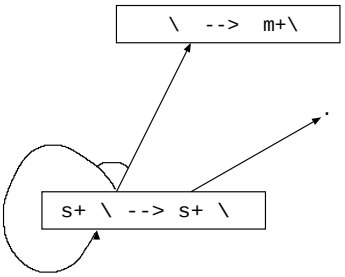


図 5 抽象証明グラフ.
Fig. 5 Abstract proof graph.

とき，その抽象証明グラフを作成するアルゴリズムについて述べる．以下のアルゴリズムは，与えられた $\mathcal{P}(M)$-シーケントから初期証明グラフ (initial proof graph) と呼ばれる証明グラフを作成するものである．また，この初期証明グラフが抽象証明グラフになることを証明する．

定義 4.11 (初期証明グラフ作成アルゴリズム) S を証明すべき $\mathcal{P}(M)$-シーケントであるとすると，初期証明グラフは次の (1)〜(6) のステップを，節点と弧がそれ以上加えられなくなるまで，繰り返すことによって生成される証明グラフである．S をルート節点 v_0 とする．任意の節点 $v \equiv \Psi; \Delta_I \backslash \Delta_O \longrightarrow \mathcal{A}_I \backslash \mathcal{A}_O$ に対して

(1) $\Delta_I \cap [\{A\}] \neq \emptyset$ かつ $\mathcal{A}_I \cap [\{A\}] \neq \emptyset$ であるような原子論理式 A が存在するならば，弧 $\langle v \rangle$ を加える．

(2) $(A_1 \wp \cdots \wp A_n :- \mathbf{1}) \in \Psi$ かつ $\mathcal{A}_I \cap [\{A_1, \cdots, A_n\}] \neq \emptyset$ ならば，弧 $\langle v \rangle$ を加える．

(3) $(A_1 \wp \cdots \wp A_n :- \bot) \in \Psi$ かつ $\mathcal{A}_I \cap [\{A_1, \cdots, A_n\}] \neq \emptyset$ ならば，節点 $v_1 \equiv \Psi; \Delta_I \backslash \Delta_O \longrightarrow \mathcal{A}_I - [\{A_1, \cdots, A_n\}] \backslash \mathcal{A}_O$ と弧 $\langle v, v_1 \rangle$ を加える．

(4) $(A_1 \wp \cdots \wp A_n :- B \multimap C) \in \Psi$ かつ $\mathcal{A}_I \cap [\{A_1, \cdots, A_n\}] \neq \emptyset$ ならば，節点 $v_1 \equiv \Psi; \Delta_I + [\{B\}] \backslash \Delta_O \longrightarrow \mathcal{A}_I - [\{A_1, \cdots, A_n\}] + [\{C\}] \backslash \mathcal{A}_O$ と弧 $\langle v, v_1 \rangle$ を加える．

(5) $(A_1 \wp \cdots \wp A_n :- B \wp C) \in \Psi$ かつ $\mathcal{A}_I \cap [\{A_1, \cdots, A_n\}] \neq \emptyset$ ならば，節点 $v_1 \equiv \Psi; \Delta_I \backslash \Delta_O \longrightarrow \mathcal{A}_I - [\{A_1, \cdots, A_n\}] + [\{B, C\}] \backslash \mathcal{A}_O$ と弧 $\langle v, v_1 \rangle$ を加える．

(6) $(A_1 \wp \cdots \wp A_n :- B \otimes C) \in \Psi$ かつ $\mathcal{A}_I \cap [\{A_1, \cdots, A_n\}] \neq \emptyset$ ならば，2 つの節点 $v_1 \equiv \Psi; \Delta_I \backslash Cl(\Delta_I) \longrightarrow \mathcal{A}_I - [\{A_1, \cdots, A_n\}] + [\{B\}] \backslash Cl(\mathcal{A}_I - [\{A_1, \cdots, A_n\}])$ と $v_2 \equiv \Psi; Cl(\Delta_I) \backslash \Delta_O \longrightarrow Cl(\mathcal{A}_I - [\{A_1, \cdots, A_n\}]) + [\{C\}] \backslash \mathcal{A}_O$ 及び，1 つの弧 $\langle v, v_1, v_2 \rangle$ を加える．

ただし，集合 $Cl(X)$ は次のように定義される．

$$Cl(X) = \{z \in M \mid x \in X, y \in M, x = y + z\}$$

□

このアルゴリズムは節点と弧が有限であるため，停止するのは明らかである．そして，定義 4.11 から次のような包含関係が成り立つ．

$$X = \{z \in M \mid x \in X, e \in M, x = e + z\} \subset \{z \in M \mid x \in X, y \in M, x = y + z\} = Cl(X)$$

ただし，e は恒等元 (identity element) である．すなわち，任意の $\mathcal{P}(M)$-マルチ集合 X に対して，$X \subset Cl(X)$ が成り立つ．

命題 4.1 初期証明グラフは抽象証明グラフである．

(証明) S を M-シーケントとし，

$$\frac{S_1 \cdots S_n}{S} \qquad (\text{ただし}, n = 0, 1, 2)$$

を $\mathbf{IO}_M$ における推論ステップとする．このとき，初期証明グラフ中の $S \in \|v\|$ であるような節点 v に対して，$S_i \in \|v_i\|$ $(n = 0, 1, 2)$ を満たすような弧 $\langle v, v_1, \cdots v_n \rangle$ が存在することを証明するのは容易である．また，$\otimes$ の場合も $\mathcal{A} \subset Cl(\mathcal{A})$ によって証明できる． ∎

5. 抽象証明グラフの精製

前節では，一般的な抽象証明グラフを生成するアルゴリズムについて述べた．しかし，この抽象証明グラフには，多くの証明不可能な節点が含まれている．

この節ではグラフが含んでいる証明不可能な節点を取り除く方法について述べる．本稿では，抽象証明グラフから証明不可能な節点を削除することを**抽象証明グラフの精製**と呼ぶことにする．また，これは抽象証明グラフの最適化とも考えることができる．なぜならば，節点，弧を取り除くことは無駄な推論ステップを

取り除くことを意味するからである．

抽象証明グラフの精製するために，本稿ではデータフロー解析のアイデアを使用する．すなわち，**F** 中の論理式 B を変数として考え，また B に対する $\mathcal{P}(M)$ 値をその変数がとり得る値の集合として考える．そしてデータフロー解析することにより，この集合は小さくなっていき，空集合になったときその節点は削除される．

ここで,「抽象証明グラフでデータフローとは何なのか?」について考える．I/O model では，殆どの場合，入力リソースは推論規則の下から上に流れ，出力リソースは上から下へ流れていく．$\otimes$ の推論規則に関しては，第 2.2 節で述べたように，上部の左側にあるシーケントの出力リソースが，右側にあるシーケントの入力リソースとなる．

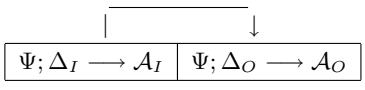
抽象証明グラフに対してデータフロー解析をおこなうために，証明グラフの各節点を 2 つの部分に分ける．以下で述べる精製アルゴリズムでは入力部分，出力部分の各々を節点とみなすので，それらをサブ節点と呼ぶことにする．すなわち，節点 $\Psi;\Delta_I\backslash\Delta_O \longrightarrow \mathcal{A}_I\backslash\mathcal{A}_O$ を入力部分 $\Psi;\Delta_I \longrightarrow \mathcal{A}_I$ と出力部分 $\Psi;\Delta_O \longrightarrow \mathcal{A}_O$ に分ける．これにより，抽象証明グラフの節点を次のように 2 つのサブ節点を並べて表す．

$\Psi;\Delta_I \longrightarrow \mathcal{A}_I$	$\Psi;\Delta_O \longrightarrow \mathcal{A}_O$

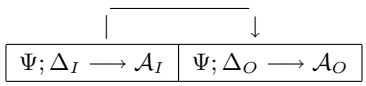
さらに，抽象証明グラフ中の各 AND 弧☆に対応してサブ節点間にデータフローリンクを加える．各リンクはその対応する推論ステップの条件 (種類) によって制約条件が与えられる．次に，各 AND 弧に対してどのようなリンクが加えられるについて述べる．

定義 5.1 G を抽象証明グラフとするとき，G 中の各 AND 弧に対応して加えられるデータフロリンクは以下のように定義される．ここで各々のリンクは制約条件 $C_1(\Delta_I,\Delta_O)$, $C_2(\mathcal{A}_I,\mathcal{A}_O)$ を持つ．

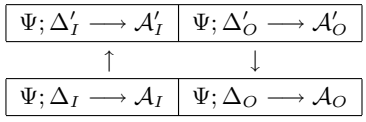
- AND 弧が原子論理式 A の (Id)-規則に対応しているならば，次のようなリンクが加えられる．



  制約：$C_1(\Delta_I,\Delta_O)$, $C_2(\mathcal{A}_I,\mathcal{A}_O)$
  - $(\Delta_I - [\{A\}]) \cap \Delta_O \neq \emptyset$
  - $(\mathcal{A}_I - [\{A\}]) \cap \mathcal{A}_O \neq \emptyset$
- AND 弧が節 $A_1 \wp \cdots \wp A_n :- \mathbf{1}$ を適用する $(\mathbf{1})$-規則に対応しているならば，次のようなリンクが加えられる．

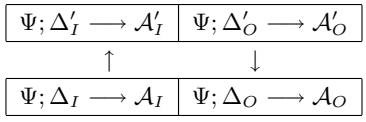


  制約：$C_1(\Delta_I,\Delta_O)$, $C_2(\mathcal{A}_I,\mathcal{A}_O)$
  - $\Delta_I \cap \Delta_O \neq \emptyset$
  - $(\mathcal{A}_I - [\{A_1,\cdots,A_n\}]) \cap \mathcal{A}_O \neq \emptyset$
- AND 弧が節 $A_1 \wp \cdots \wp A_n :- \bot$ を適用する $(\bot)$-規則に対応しているならば，次のような 2 つのリンクが加えられる．



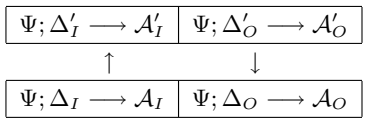
  左リンクの制約：$C_1(\Delta_I,\Delta'_I)$, $C_2(\mathcal{A}_I,\mathcal{A}'_I)$
  - $\Delta_I \cap \Delta'_I \neq \emptyset$
  - $(\mathcal{A}_I - [\{A_1,\cdots,A_n\}]) \cap \mathcal{A}'_I \neq \emptyset$

  右リンクの制約：$C_1(\Delta'_O,\Delta_O)$, $C_2(\mathcal{A}'_O,\mathcal{A}_O)$
  - $\Delta'_O \cap \Delta_O \neq \emptyset$
  - $\mathcal{A}'_O \cap \mathcal{A}_O \neq \emptyset$
- AND 弧が節 $A_1 \wp \cdots \wp A_n :- B \multimap C$ を適用する $(\multimap)$-規則に対応しているならば，次のような 2 つのリンクが加えられる．



  左リンクの制約：$C_1(\Delta_I,\Delta'_I)$, $C_2(\mathcal{A}_I,\mathcal{A}'_I)$
  - $(\Delta_I + [\{B\}]) \cap \Delta'_I \neq \emptyset$
  - $((\mathcal{A}_I - [\{A_1,\cdots,A_n\}]) + [\{C\}]) \cap \mathcal{A}'_I \neq \emptyset$

  右リンクの制約：$C_1(\Delta'_O,\Delta_O)$, $C_2(\mathcal{A}'_O,\mathcal{A}_O)$
  - $\Delta'_O \cap \Delta_O \neq \emptyset$
  - $\mathcal{A}'_O \cap \mathcal{A}_O \neq \emptyset$
- AND 弧が節 $A_1 \wp \cdots \wp A_n :- B \wp C$ を適用する $(\wp)$-規則に対応しているならば，次のような 2 つのリンクが加えられる．



  左リンクの制約：$C_1(\Delta_I,\Delta'_I)$, $C_2(\mathcal{A}_I,\mathcal{A}'_I)$
  - $\Delta_I \cap \Delta'_I \neq \emptyset$
  - $((\mathcal{A}_I - [\{A_1,\cdots,A_n\}]) + [\{B,C\}]) \cap \mathcal{A}'_I \neq \emptyset$

  右リンクの制約：$C_1(\Delta'_O,\Delta_O)$, $C_2(\mathcal{A}'_O,\mathcal{A}_O)$
  - $\Delta'_O \cap \Delta_O \neq \emptyset$
  - $\mathcal{A}'_O \cap \mathcal{A}_O \neq \emptyset$

☆ AND 弧は各推論ステップに対応している

- AND 弧が節 $A_1 \wp \cdots \wp A_n :- B \otimes C$ を適用する $(\otimes)$-規則に対応しているならば，**図 6** のような 3 つのリンクが加えられる．
  左リンクの制約：$C_1(\Delta_I, \Delta'_I)$, $C_2(\mathcal{A}_I, \mathcal{A}'_I)$
  - $\Delta_I \cap \Delta'_I \neq \emptyset$
  - $((\mathcal{A}_I - [\{A_1, \cdots, A_n\}]) + [\{B\}]) \cap \mathcal{A}'_I \neq \emptyset$

  中リンクの制約：$C_1(\Delta'_O, \Delta''_I)$, $C_2(\mathcal{A}'_O, \mathcal{A}''_I)$
  - $\Delta'_O \cap \Delta''_I \neq \emptyset$
  - $(\mathcal{A}'_O + [\{C\}]) \cap \mathcal{A}''_I \neq \emptyset$

  右リンクの制約：$C_1(\Delta''_O, \Delta_O)$, $C_2(\mathcal{A}''_O, \mathcal{A}_O)$
  - $\Delta''_O \cap \Delta_O \neq \emptyset$
  - $\mathcal{A}''_O \cap \mathcal{A}_O \neq \emptyset$

□

すなわち，リンクは (Id)-規則，$(\mathbf{1})$-規則では節点入力部分から出力部分に連結される．また，$(\otimes)$-規則以外に対応する AND 弧に対しては各々2 つのリンクが付けられる．その 1 つのリンクは AND 弧が出ていく節点の入力部分から AND 弧の先の節点の入力部分に付けられている．そして，もう 1 つのリンクは AND 弧の先の節点の出力部分から AND 弧が出ていく節点の出力部分に付けられている．規則 $(\otimes)$ に対応する AND 弧は 1 つの節点から左右に 2 つの節点に弧が出ている．この AND 弧に対しては AND 弧の左先の節点の出力部分から AND 弧の右先の節点の入力部分に付けられるデータフローリンクを含む 3 つのリンクが付けられる．

命題 5.1 G をルート節点が v_0 である抽象証明グラフとする．そのとき G が $\mathbf{IO}_M$ の証明 Π を含んでいるならば，v_0 の入力部分のサブ節点から始まり v_0 の出力部分のサブ節点で終るデータフローリンクの列が存在し，各データフローリンクはその制約条件を満たしている．

(証明) $\mathbf{IO}_M$ の正しい推論ステップは対応するリンクの制約条件を満たしていることから明らかである．

∎

次に，上で定義したデータフローリンクを使って抽象証明グラフの精製 (最適化) を行うアルゴリズムについて述べる．これには，データフローリンクを通して情報を伝搬することにより最適化を行う反復アルゴリズムを使用する．このようなアルゴリズムは，コンパイラの最適化などに広く用いられている．

本稿の抽象証明グラフの精製アルゴリズムは，前方伝搬アルゴリズムと後方伝搬アルゴリズムの 2 種類を組み合わせたものである．以下に，伝搬アルゴリズムの概要を示す．

伝搬アルゴリズムでは，各サブ節点 $\Psi; \Delta \longrightarrow \mathcal{A}$ について，Δ と $\mathcal{A}$ の取り得る値 ($\mathcal{P}(M)$-マルチ集合) を求める．そこで，サブ節点 u の Δ と $\mathcal{A}$ の取り得る値を保持する変数を各々$\Delta(u)$, $\mathcal{A}(u)$ で表す．

定義 5.2 (前方伝搬アルゴリズム)
G をルート節点が v_0 である抽象証明グラフとする．v_0 の入力部分のサブ節点を $u_0(\Psi; \Delta_I \longrightarrow \mathcal{A}_I)$，出力部分のサブ節点を $u_1(\Psi; \Delta_O \longrightarrow \mathcal{A}_O)$ とする．

(1) u_0, u_1 以外のすべてのサブ節点 u について，$\Delta(u) := \emptyset$, $\mathcal{A}(u) := \emptyset$ とする．また，$\Delta(u_0) := \Delta_I$, $\mathcal{A}(u_0) := \mathcal{A}_I$, $\Delta(u_1) := \Delta_O$, $\mathcal{A}(u_1) := \mathcal{A}_O$ とする．

(2) 節点 v_0 の入力部分のサブ節点から始めて，すべてのリンクを深さ優先で辿りながら次を実行する．ただし，辿っているリンクはサブ節点 u から，u' へのものとし，制約条件を $C_1(\Delta, \Delta')$, $C_2(\mathcal{A}, \mathcal{A}')$ とする．
- $C_1(\Delta(u), \Delta'')$ を満たす最小の Δ'' を求める．
- $C_2(\mathcal{A}(u), \mathcal{A}'')$ を満たす最小の $\mathcal{A}''$ を求める．
- $\Delta(u') := \Delta(u') \cup \Delta''$, $\mathcal{A}(u') := \mathcal{A}(u') \cup \mathcal{A}''$ とする．
- $\Delta(u')$, $\mathcal{A}(u')$ が変化していないか，$u' = u_1$ であれば u' からのリンクは辿らず前に戻る．
- それ以外の場合，u' からのリンクを辿る．

(3) 各サブ節点 u を $\Psi; \Delta(u) \longrightarrow \mathcal{A}(u)$ で更新する．

□

また，後方伝搬アルゴリズムも同様に定義される．

定義 5.3 (抽象証明グラフの精製アルゴリズム)
(1) 前方伝搬を実行する．
(2) 削除アルゴリズムを実行する．
- **削除アルゴリズム**
  入力部分あるいは出力部分の Δ あるいは $\mathcal{A}$ の部分が空になった節点と制約条件を満たさなくなったリンクを取り除く．

(3) 後方伝搬を実行する．
(4) 削除アルゴリズムを実行する．

□

この精製は何度でも繰り返し実行することができ，またいつ停止させてもかまわない．しかし，精製を繰り返す内に情報伝搬によって変化が生じなくなった場合，それ以上の精製は無意味であると言える．

6. 性 能 評 価

本論文で提案した静的解析手法での $\mathcal{P}(M_2)$ に基づくプロトタイプ解析機☆の性能評価について述べる．実行速度は Linux マシーン (MMX 266MHz, 128Mb

☆ SICStus Prolog 3.7.1(compact code) で記述した．

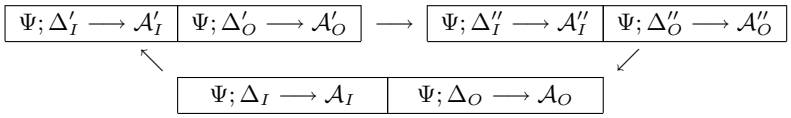


図 6 (⊗)-規則のリンク.
Fig. 6 Links for (⊗)-rule.

表 2 ソーティングプログラムの解析結果.
Table 2 Analysis result of sorting program in Forum.

	nodes	arcs	CPU(msec)
Initial proof graph	57	220	1530
1st refinement	25	53	11720
2nd refinement	15	29	2160
3rd refinement	15	29	2110

表 3 ソーティングプログラムの実行時間 (msec).
Table 3 Execution time of sorting program.

	解析前	解析後
5 要素のソーティング	120	0.940
6 要素のソーティング	3570	1.300
7 要素のソーティング	163080	1.730

表 4 生産者消費者プログラムの解析結果.
Table 4 Analysis result of producer-consumer program.

	nodes	arcs	CPU(msec)
Initial proof graph	78	143	1440
1st refinement	67	127	13860
2nd refinement	67	127	12310

Memory) 上で測定した．ベンチマークには Forum で書かれたソーティングプログラム [8](付録参照) を使用した．

解析機には，以下のようなプログラムが入力として与えられる．

$$\begin{array}{rcl} g & :- & s \multimap p \\ p & :- & s \\ p & :- & s \wp p \\ s \wp s & :- & s \otimes m \\ m & :- & \mathbf{1} \\ m & :- & m \end{array}$$

解析機はまず上のプログラムと証明すべきシーケント $\longrightarrow g$ から抽象証明グラフを作成する．精製 (最適化) される前のこの初期抽象証明グラフは，57 の節点と 220 の弧から構成されている．**表 2** に精製後の抽象証明グラフの節点の数，弧の数，及び解析時間を示す．

表 2 からわかるようにシーケント $\longrightarrow g$ による抽象証明グラフはとても大きく本稿中に描写することが困難である．そこで，上のプログラムとシーケント $s \longrightarrow s \wp s$ から生成される抽象証明グラフに関して，精製前 (**図 7**) と精製後 (**図 8**) のグラフを示す．ただし，両図ともプログラムの部分は変化がないので省略されており，シーケント $s \longrightarrow s \wp s$ は $s \longrightarrow s2$ で表されている．この例では，最初あった 10 個の節点が，精製により 4 個に最適化されていることがわかる．

次に静的解析による実行速度の向上について述べる．**表 3** は以下の 2 種類のソーティングプログラムを線形論理型言語 LLP[10),15)] に手動でトランスレートし，それを LLP コンパイラシステム☆ で実行した時間を示している．

(1) 静的解析をしていないソーティングプログラム
(2) プロトタイプ解析機による静的解析の結果を反映させたソーティングプログラム

(1) と (2) の大きな違いは，(1) が各推論のステップにおいてすべての規則をチェックするのに対して，(2) は静的解析によって削除された規則はチェックしない点である．

表 3 から，6 要素をソーティングに対して，本稿の静的解析により 1000 倍以上の実行速度の向上が実現されているのがわかる．また，ソーティングするリストが大きくなればなるほど，この実行速度の差はさらに大きくなる．

表 4 はリソースを生成するプログラムと消費するプログラムが並列的に動作する生産者消費者プログラムに対する解析結果を表している．生産者消費者プログラムの実行時間は，500 要素を生産して消費する場合，解析結果を適用していないプログラムが 1190 msec かかるのに対して，解析結果を適用したプログラムは 520 msec であった．

次に生産者と消費者システムを表すペトリネット (**図 9**) のシミュレーションプログラムの解析結果について述べる．図 9 はデータをバッファーに書き込む生産プロセスとバッファーからデータを読み込む消費プロセスがあるペトリネットである．生産プロセスは空のバッファーにしかデータを書き込むことはできず，空のバッファーがない場合は生産プロセスは待ち続け

☆ http://bach.seg.kobe-u.ac.jp/llp/

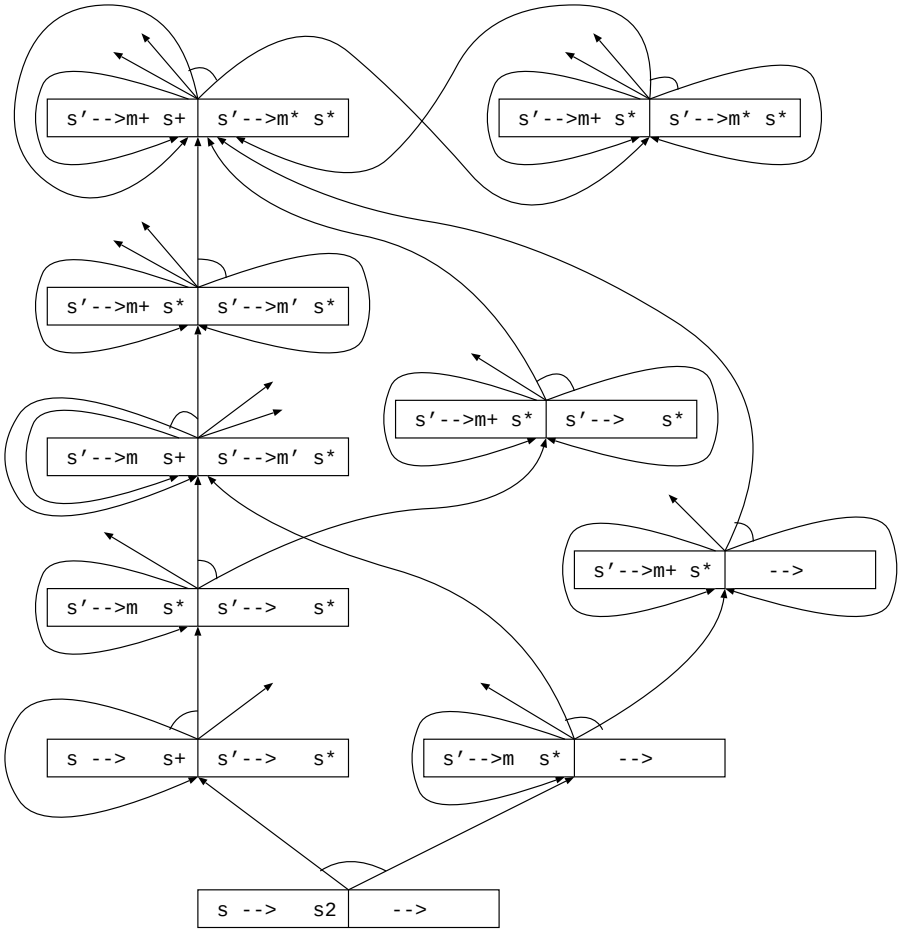


図 7 精製前の抽象証明グラフ.
Fig. 7 Initial Abstract Proof Graph.

る．そして，プレース “ Empty_buffers” のマーキングの初期値は空のバッファーの数である．すなわち，図 9 は 5 つの空バッファーが与えられた時の初期状態である．また，プレース “Semaphore” は任意のデータに対して 2 つのプロセスが同時に実行しないことを保証している．消費プロセスはバッファーからデータを読み込んだ後，バッファーを空にしている．

図 10 はこのペトリネットに対応する Forum のプログラムである．**表 5** は，このプログラムにおいて，初期状態が任意の数の空バッファーをもち (ただし，データが入っているバッファーは 0 個)，到達状態が任意の数のデータが入っているバッファーをもつ (ただし，空バッファーは 0 個) 場合の解析結果である．このペトリネットシミュレーションプログラムの実行時間は解析結果を適用していないプログラムの場合，3118 msec がかかるのに対して，解析結果を適用したプログラムは 2997 msec であり，実行速度はそれほど向上しなかった．その理由はプログラムが乗法的論理積 (⊗) を含んでいないからであると考えられる．

しかしながら，初期状態が任意の数の空バッファーをもち (ただし，データが入っているバッファーは 0 個)，到達状態がデータが入っているバッファーが 0 個である場合を解析した結果，ルート節点以外は全て取り除かれ，空のバッファーがない場合だけが証明可能となった．すなわち，空のバッファーが存在するのにデータが入っているバッファーは 0 個である場合は証明不可能であることが解析できた．これは，本稿の解析方法がペトリネットの到達可能性を調べる上で有効に利用できることを示している．

7. 結論と今後の課題

本稿では古典線形論理に基づく論理型言語の新しい静的解析方法を提案した．この静的解析方法の大きな特徴は，乗法的論理和 (℘)，含意 (—∘) に加えて，An-

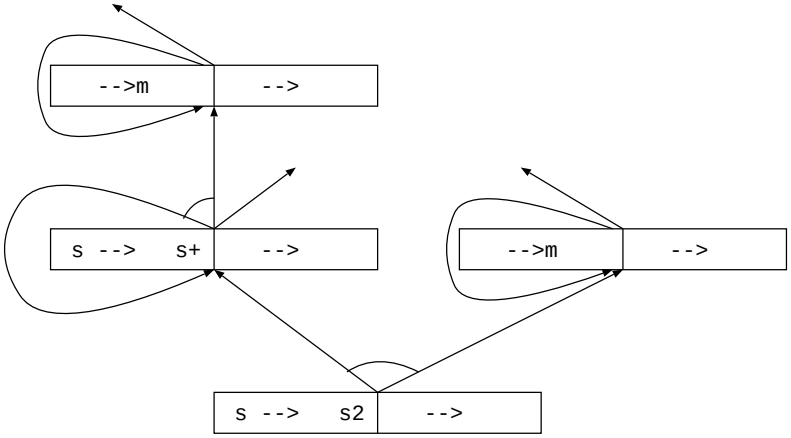


図 8 精製後の抽象証明グラフ.
Fig. 8 Refined Abstract Proof Graph.

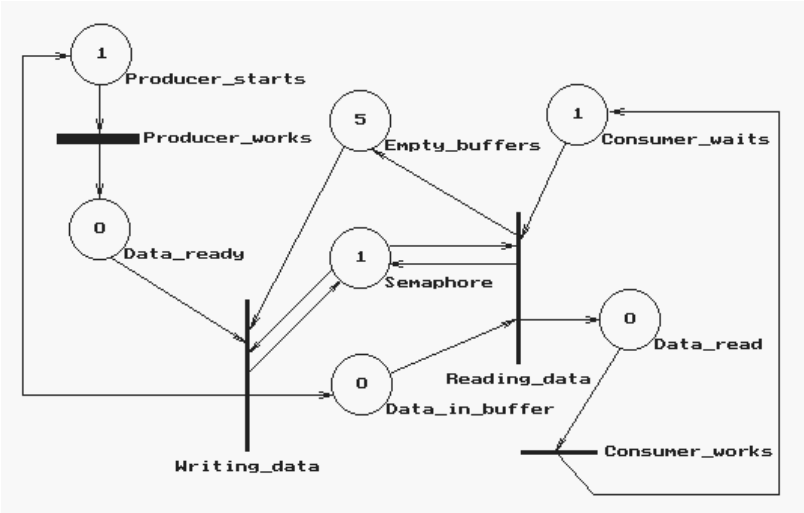


図 9 生産者と消費者システムを表すペトリネット.
Fig. 9 A Petri Net for a producer-consumer system.

```
producer_starts :- data_ready.

data_ready ℘ empty_buffers ℘ semaphore :-  data_in_buffer ℘ producer_starts ℘ semaphore.

data_in_buffer ℘ semaphore ℘ consumer_waits :- empty_buffers ℘ data_read ℘ semaphore.

data_read :- consumer_waits.
```

図 10 生産者と消費者システムを解く Forum プログラム.
Fig. 10 A Forum program for a producer-consumer system.

dreoli らの解析方法では取扱うことのできない乗法的論理積 (⊗) を解析できる点である.

我々の方法では,まずプログラムと証明すべきゴールが与えられたら,シーケントのあり得るすべての証明を 1 つの AND-OR グラフに表現する.その後,そのグラフに対して,データフロー解析のための前方伝搬と後方伝搬を繰り返すことによって,証明不可能な節点と弧をグラフから削除していく.プロトタイプ解

表 5 ペトリネットシミュレーションの解析結果.
Table 5 Analysis result of Petri Net simulation.

	nodes	arcs	CPU(msec)
Initial proof graph	28	49	630
1st refinement	25	46	6100
2nd refinement	25	46	5910

析機を開発し，実際に Forum で書かれたソーティングプログラムを静的解析した結果，6 要素のソーティングに対し，静的解析していない場合の実行速度に比べ 1000 倍以上速くなった.

今後の課題としては，現在の静的解析手法を加法的論理積 (&)，直観主義含意 (⇒) などさらに多くの演算子を解析できるように拡張することを考えている.

また，含意 (⊸) と乗法的論理積 (⊗) を含んでいないプログラムに関しては Andreoli らが提案した解析方法の方が精度が高くなる可能性がある．同じ論理式 A が 2 個出現した地点で，無条件に A^2 とする ($\mathcal{P}(M_2)$ に基づく) 我々方法と違って，Andreoli らが提案した解析方法は探索パス上のそれ以前に出てきたシーケント $\longrightarrow A$ を包含するシーケント $\longrightarrow A, B$ が出現した時点ではじめて $\longrightarrow A, B^*$ と近似するからである．Andreoli らが提案した解析方法を含意 (⊸) と乗法的論理積 (⊗) も解析できるように拡張する方法も含め，より効率良い解析方法を開発することは今後の課題となる.

謝辞 最後に，査読者の方々，並びにプログラミング研究会において貴重な御意見を下さった皆様に感謝いたします.

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) Andreoli, J.-M., Pareschi, R. and Castagnetti, T.: Static Analysis of Linear Logic Programming, *New Generation Computing,15*, pp. 449–481 (1997).
4) Girard, J.-Y.: Linear Logic, *Theoretical Computer Science*, Vol. 50, pp. 1–102 (1987).
5) Harland, J. and Pym, D.: The Uniform Proof-Theoretic Foundation of Linear Logic Programming, *Proceedings of the International Logic Programming Symposium* (Saraswat, V. and Ueda, K.(eds.)), San Diego, California, pp. 304–318 (1991).
6) 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).
7) 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.
8) Hodas, J.S. and Polakow, J.: Forum as a Logic Programming Language: Preliminary Results and Observations, *Proceedings of the Linear Logic '96 Meeting* (Okada, M.(ed.)), Vol. 3, Tokyo, Japan, Elsevier Electronic Notes in Theoretical Computer Science (1996).
9) Hodas, J. S. and Polakow, J.: Early Observation on Forum as a Logic Programming Language, Unpublised (1997).
10) Hodas, J. S., Watkins, K., Tamura, N. and Kang, K.-S.: Efficient Implementation of a Linear Logic Programming, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pp. 145–159 (1998).
11) Kobayashi, N. and Yonezawa, A.: Typed Higher-Order concurrent linear logic programming, Technical Report 94-12, University of Tokyo (1994).
12) Lincoln, P.: Linear Logic, *ACM SIGACT Notices*, Vol. 23, No. 2, pp. 29–37 (1992).
13) Miller, D.: FORUM: A Multiple-Conclusion Specification Logic, *Theoretical Computer Science*, Vol. 165, No. 1, pp. 201–232 (1996).
14) Miller, D., Nadathur, G., Pfenning, F. and Scedrov, A.: Uniform proofs as a foundation for logic programming, *Annals of Pure and Applied Logic,51*, pp. 125–157 (1991).
15) 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, pp. 33–50 (1996).

付 録

A.1 ソーティングプログラム

図 11 と**図 12** は各々リスト $[3, 2, 1]$ が与えられたとき，ソーティングされたリスト $[1, 2, 3]$ を求める実

$$\dfrac{\dfrac{\dfrac{\dfrac{\Psi;\ \longrightarrow \mathtt{m}([2,3],[1],[1,2,3]) \quad \dfrac{\mathtt{X}=[1,2,3]}{\Psi;\mathtt{s}(\mathtt{X})\longrightarrow\mathtt{s}([1,2,3])}}{\Psi;\ \longrightarrow \mathtt{m}([3],[2],[2,3]) \qquad \Psi;\mathtt{s}(\mathtt{X})\longrightarrow\mathtt{s}([2,3]),\mathtt{s}([1])}}{\Psi;\mathtt{s}(\mathtt{X})\longrightarrow\mathtt{s}([3]),\mathtt{s}([2]),\mathtt{s}([1]),\mathtt{s}([])}}{\Psi;\mathtt{s}(\mathtt{X})\longrightarrow\mathtt{p}([3,2,1])}}{\Psi;\ \longrightarrow \mathtt{sort}([3,2,1],\mathtt{X})}$$

図 11 ソーティングプログラムの実行に対応する **LL** の証明図.
Fig. 11 A **LL**-proof for sorting program.

$$\dfrac{\dfrac{\dfrac{\dfrac{\Psi;\mathtt{s}(\mathtt{X})\backslash\mathtt{s}(\mathtt{X})\longrightarrow\mathtt{m}([3],[2],[2,3]),\mathtt{s}([1])\backslash\mathtt{s}([1]) \quad \dfrac{\Psi;\mathtt{s}(\mathtt{X})\backslash\mathtt{s}(\mathtt{X})\longrightarrow\mathtt{m}([2,3],[1],[1,2,3])\backslash \quad \dfrac{\mathtt{X}=[1,2,3]}{\Psi;\mathtt{s}(\mathtt{X})\backslash\ \longrightarrow\mathtt{s}([1,2,3])\backslash}}{\Psi;\mathtt{s}(\mathtt{X})\backslash\ \longrightarrow\mathtt{s}([3]),\mathtt{s}([2]),\mathtt{s}([1]),\mathtt{s}([])\backslash}}{\Psi;\mathtt{s}(\mathtt{X})\backslash\ \longrightarrow\mathtt{p}([3,2,1])\backslash}}{\Psi;\ \backslash\ \longrightarrow\mathtt{sort}([3,2,1],\mathtt{X})\backslash}$$

図 12 ソーティングプログラムの実行に対応する **IO** の証明図.
Fig. 12 A **IO**-proof for sorting program.

行に対応する **LL** の証明図と **IO** の証明図の一例である．図 11 と図 12 で Ψ は以下のソーティングプログラムである．また，sorted は s，merge は m，spawn は p で簡略に表現している．そして，merge の実行などの部分は省略している．

ソーティングプログラム

```
sort(L,S) :- sorted(S) —○ spawn(L).

spawn([]) :- sorted([]).
spawn([H|T]) :- sorted([H]) ℘ spawn(T).

sorted(L1) ℘ sorted(L2) :-
            merge(L1,L2,L) ⊗ sorted(L).

merge([],L,L).
merge([H|T],[],[H|T]).
merge([H1|T1],[H2|T2],[H1|L]) :-
            H1 =< H2 ⊗ merge(T1,[H2|T2],L).
merge([H1|T1],[H2|T2],[H2|L]) :-
            H1 > H2 ⊗ merge([H2|T2],T2,L).
```