帰納的述語定義を含む分離論理における循環証明による半自動証明

仲田 壮佑　中澤 巧爾

帰納的述語定義を含む分離論理によるプログラムの自動検証に向けて，論理式の半自動証明器を実装した．より具体的には，Brotherston らの提案した循環証明体系における証明探索を行なう．循環証明体系による証明では，特定の条件のもとで証明木に循環を許し，帰納的に定義された述語を含む論理式の証明を行う．証明探索の際には帰納的述語を定義に従って展開する必要があるが，素朴な方法では適用可能な推論規則の数が爆発的に増加する．展開する述語の選び方と適用する推論規則の順番を制限し，探索の枝刈りを行った．以上のアイディアに基づき，証明探索を行う自動証明器を実装した．本研究で実装した証明器を用いて，様々な帰納的述語定義を含む論理式の自動証明を行った．また，証明可能なときには，証明器に証明木を出力させ，証明探索によって発見された証明を可視化した．

We implement a semi-automated prover for implicational formulas (entailments) of the separation logic including inductive predicate definitions, toward automation of program verification by separation logic. More specifically, we use cyclic proof system, proposed by Brotherston et al., to construct proof. In cyclic proof system, we allow proof trees to contain cycles under a specific condition and prove entailments including inductive predicates. We must unfold inductive predicates in accordance with their definition. In a naïve way, the number of applicable inference rules increases explosively. For pruning, we restricted the selection method of predicates to be unfolded and the order of applications of the inference rules. Based on the above idea, we implemented an automated prover searching for a proof. We automatically proved several entailments with inductive predicates by using the prover implemented in this paper. Besides we let the prover output proof trees as LaTeX code if the entailment is provable, and we visualized the proofs found by proof search.

1 はじめに

プログラムの正当性を実行前に保証する手法の一つとして分離論理[6] がある．分離論理はホーア論理をポインタを操作するコマンドとヒープ状態を表す条件によって拡張したものである．現実のプログラムは，ヒープ領域のメモリ操作を含むものが多く，そのようなプログラムの正当性検証に即した分離論理は，近年盛んに研究されている．

ホーア論理およびその拡張の分離論理ではホーアの三つ組（Hoare triple）を用いてプログラムの性質を表現する．事前条件 ϕ が成立するときにプログラム C を実行し，停止すれば，必ず事後条件 ψ を満たすことを $\{\phi\}C\{\psi\}$ と表し，これをホーアの三つ組と呼ぶ．とくに分離論理では，条件 ϕ，ψ によってヒープの状態を表現できる．論理体系においてホーアの三つ組を証明する際，多くの場面で条件式の含意 $\phi_1 \models \phi_2$ の正しさを判定する必要がある．ここで，$\phi_1 \models \phi_2$ は，条件 ϕ_1 を満たす任意のヒープが条件 ϕ_2 を満たすことを表す式で，判断式（entailment）と呼ばれる．

本研究では，帰納的に定義された述語を含む分離論理において $\phi_1 \models \phi_2$ の形の判断式の妥当性を自動判定することを目標とし，Brotherston による循環証明体系 [2] [3] における半自動証明手続を提案する．さらに，その実装を与え，いくつかの実行例を示す．

帰納的に定義された述語に関する性質は帰納法によって証明されることが多い．帰納法では，証明しようとしている主張を帰納法の仮定として利用するが，これを循環する証明木の形で実現したものが循環証

Semi-automated Entailment Checker by Cyclic Proofs for Separation Logic with Inductive Predicates.

Sosuke Nakada and Koji Nakazawa, 名古屋大学大学院情報科学研究科情報システム学専攻, Dept. of Information Engineering, Graduate School of Information Science, Nagoya University.

明体系である．循環証明体系では帰納法の仮定に当たる部分を結論と繋ぐことでできる循環構造を証明木に含むことを許す．しかし，循環証明体系においては，述語を帰納的定義によって展開することによって証明探索を進めるため，どの述語をどのタイミングで展開するか，を考えなければならず，素朴な方法では考慮すべき場合の数が爆発することがある．

一般に，帰納的述語を含む判断式に対する循環証明を構成するためには，判断式の左辺と右辺において対応するメモリ領域について言及している述語を発見し，それらを帰納的定義に沿って展開することで循環構造を発見できることが多い．これは，場合分けと帰納法の仮定を用いた通常の帰納法による証明に相当する．本研究では，この判断式の両辺において対応する述語を探すために，推論規則の適用の順序を制御し，また，帰納的定義の展開を制限する．これにより，効率的な証明探索が実現できると考えられる．

さらに，本研究で提案する手続を実装した証明器によって，いくつかの判断式の例の妥当性を判定し，これらの判断式に対して確かに証明を与えられることを確認した．さらに，証明可能な判断式に対してはその証明を証明図の形で出力するようにし，証明の可視化を行なった．

本論文の構成は以下の通りである．第 2 節では分離論理及び帰納的述語定義を与える．第 3 節では循環証明体系を定義する．第 4 節では本研究が提案する証明探索の手続について解説し，第 5 節では第 4 節のアイディアにもとづいて実装した証明器による実行例を載せる．最後に第 6 節で本研究のまとめと今後の課題を述べる．

2 分離論理

分離論理の項（t, t_1, t_2,...）は，変数（x, y, z,...）または定数 nil である．変数全体の集合を Vars と書く．本論文で扱う論理式は，シンボリック・ヒープ（symbolic heap）と呼ばれるもので，以下のとおり定義される．

$$\begin{aligned} \text{ストア論理式 } \Pi &::= t = t \mid t \neq t \mid \Pi \wedge \Pi \\ \text{ヒープ論理式 } \Sigma &::= \mathbf{emp} \mid t \mapsto (\vec{t}) \mid \Sigma * \Sigma \mid P(\vec{t}) \\ \text{論理式 } \phi &::= \Pi \wedge \Sigma \end{aligned}$$

ここで，P は帰納的述語であり，各帰納的述語のアリティは固定とする．表記の便宜上，分離論理における結合（$*$）は論理積（$\wedge$）より記号としての結合が強いものとする．

述語 P の帰納的定義とは，

$$P(\vec{t}) \quad = \quad \bigvee_{i=0}^{n} \exists \vec{z_i}.\phi_i \quad (\vec{z_i} = FV(\phi_i) - \vec{t})$$

の形の式とする．各 $\exists \vec{z}.\phi_i$ を P の定義節と呼ぶ．述語の帰納的定義の有限集合 I が帰納的定義集合であるとは，I が各 P に対する帰納的定義を高々一つしか含まず，定義節中に出現する任意の述語 Q について，I が Q の帰納的定義を含んでいることをいう．以下では，一つの帰納的定義集合を固定して考えているものとする．

次に分離論理の意味論を与える．変数は，値としてメモリ上のロケーションまたは nil をとるが，ここではロケーションを 0 以外の自然数，nil を 0 として解釈する．

$$\begin{aligned} \text{Val} &:= N \\ \text{Locs} &:= N \backslash \{0\} \end{aligned}$$

メモリ状態を表すのは組 (s, h) である．s はストアと呼ばれ，変数に対する値（Val）の割当てを表現する．また h はヒープと呼ばれ，各ロケーションへの値（の組）の割当てを表現する．値が割当てられているメモリ領域は有限であるとする．すなわち，ストアとヒープは次のような集合の要素である．

$$\begin{aligned} \text{Stores} &:= \text{Vars} \to \text{Val} \\ \text{Heaps} &:= \text{Locs} \to_{fin} \bigcup_{n \geq 1} \text{Val}^n \end{aligned}$$

任意のストアは，$s(\text{nil}) = 0$ によって，項から Val への関数だと思える．h の定義域を $\text{Dom}(h)$ で表す．二つのヒープ h_1 と h_2 について，$h = h_1 + h_2$ とは，$\text{Dom}(h_1) \cap \text{Dom}(h_2) = \emptyset$ かつ h が次のようなヒープであることを意味する．

$$\begin{aligned} n \in \text{Dom}(h_1) \text{ のとき } h(n) &= h_1(n) \\ n \in \text{Dom}(h_2) \text{ のとき } h(n) &= h_2(n) \end{aligned}$$

メモリ状態 (s, h) が ϕ を満たすことを $s, h \models \phi$ と表し，以下で定義する．

$$\begin{aligned} s, h \models t_1 = t_2 \quad &\text{iff} \quad s(t_1) = s(t_2) \\ s, h \models t_1 \neq t_2 \quad &\text{iff} \quad s(t_1) \neq s(t_2) \\ s, h \models \Pi_1 \wedge \Pi_2 \quad &\text{iff} \quad s, h \models \Pi_1 \text{かつ } s, h \models \Pi_2 \end{aligned}$$

$$
\begin{array}{rll}
s,h \models \mathbf{emp} & \text{iff} & \mathrm{Dom}(h) = \emptyset \\
s,h \models t \mapsto (\vec{t}) & \text{iff} & \mathrm{Dom}(h) = \{s(t)\} \text{ かつ } h(s(t)) = s(\vec{t}) \\
s,h \models \Sigma_1 * \Sigma_2 & \text{iff} & \exists h_1, h_2 (h = h_1 + h_2 \text{ かつ} \\
 & & s,h_1 \models \Sigma_1 \text{ かつ } s,h_2 \models \Sigma_2) \\
s,h \models P(\vec{t}) & \text{iff} & \exists n (s,h \models P^n(\vec{t})) \\
s,h \models P^0(\vec{t}) & & \text{は成立しない} \\
s,h \models P^{k+1}(\vec{t}) & \text{iff} & P \text{ の定義節 } \exists \vec{z}.\phi_i \text{ と} \\
 & & \text{値の列 } \vec{n} \text{ が存在して} \\
 & & s[\vec{z} := \vec{n}], h \models \phi_i[P := P^k](\vec{t}) \\
s,h \models \Pi \wedge \Sigma & \text{iff} & s,h \models \Pi \text{ かつ } s,h \models \Sigma
\end{array}
$$

判断式 $\phi_1 \models \phi_2$ は「任意の (s,h) に対して，$s,h \models \phi_1$ ならば $s,h \models \phi_2$」と定義する．

3 循環証明体系

本論文では，与えられた ϕ_1, ϕ_2 における $\phi_1 \models \phi_2$ の妥当性を判定することを目標として，$\phi_1 \vdash \phi_2$ の証明探索を行い，自動証明を行う証明器を実装する．本節では，証明探索のための推論規則と循環証明の定義を行う．

以下の推論規則を用いる．

$$\frac{\Pi_1 \wedge \mathbf{emp} * \Sigma_1 \vdash \Pi_2 \wedge \Sigma_2}{\Pi_1 \wedge \Sigma_1 \vdash \Pi_2 \wedge \Sigma_2}\ (\mathbf{emp}\mathrm{L})$$

$$\frac{\Pi_1 \wedge \Sigma_1 \vdash \Pi_2 \wedge \mathbf{emp} * \Sigma_2}{\Pi_1 \wedge \Sigma_1 \vdash \Pi_2 \wedge \Sigma_2}\ (\mathbf{emp}\mathrm{R})$$

$$\frac{(\Pi_1 \wedge \Sigma_1 \vdash \Pi_2 \wedge \Sigma_2)[t_2 := t_1]}{t_1 = t_2 \wedge \Pi_1 \wedge \Sigma_1 \vdash \Pi_2 \wedge \Sigma_2}\ (=\mathrm{L})$$

$$\frac{\Pi_1 \wedge \Sigma_1 \vdash \Pi_2 \wedge \Sigma_2 \quad t_1 \neq t_2 \wedge \Pi_1 \wedge \Sigma_1 \text{ is } \mathit{UNSAT}}{\Pi_1 \wedge \Sigma_1 \vdash t_1 = t_2 \wedge \Pi_2 \wedge \Sigma_2}\ (=\mathrm{R})$$

$$\frac{\Pi_1 \wedge \Sigma_1 \vdash \Pi_2 \wedge \Sigma_2 \quad t_1 = t_2 \wedge \Pi_1 \wedge \Sigma_1 \text{ is } \mathit{UNSAT}}{\Pi_1 \wedge \Sigma_1 \vdash t_1 \neq t_2 \wedge \Pi_2 \wedge \Sigma_2}\ (\neq\mathrm{R})$$

$$\frac{}{\Pi_1 \wedge \mathbf{emp} \vdash \mathbf{emp}}\ (\mathrm{Id}) \qquad \frac{\Pi \wedge \Sigma_1 \vdash \Sigma_2}{\Pi \wedge \Sigma * \Sigma_1 \vdash \Sigma * \Sigma_2}\ (\mathrm{P\_Id})$$

$$\frac{\begin{array}{c}\Pi \wedge \phi_1\theta * \Sigma_1[\vec{t} := \vec{x}\theta] \vdash \Sigma_2[\vec{t} := \vec{x}\theta] \\ \cdots \\ \Pi \wedge \phi_n\theta * \Sigma_1[\vec{t} := \vec{x}\theta] \vdash \Sigma_2[\vec{t} := \vec{x}\theta]\end{array}}{\Pi \wedge P(\vec{t}) * \Sigma_1 \vdash \Sigma_2}\ (\mathrm{UnfL}P)$$

$$\frac{\Pi \wedge \Sigma_1 \vdash \phi_i[\vec{z} := \vec{y}] * \Sigma_2}{\Pi \wedge \Sigma_1 \vdash P(\vec{t}) * \Sigma_2}\ (\mathrm{UnfR}P_i)$$

ただし，(UnfLP) において $\theta = [\vec{z} := \vec{y}]$ であり，$\vec{y}$ は fresh であるとする．また，(UnfLP)，(UnfRP_i) において，P は $P(\vec{t}) = \bigvee_{i=1}^{n} \exists \vec{z}_i.\phi_i$ で定義されているとする．

(Id)，(P_Id)，(**emp**L)，(**emp**R) は [2] での (Id)，(∗)，(**emp**L)，(**emp**R) から導くことができる．(Id) は公理である．

(UnfLP)，(UnfRP_i) を展開規則と呼び，これらの規則を適用することを述語を展開すると呼ぶ．述語を展開することで，証明木の祖先と変数名以外が同じ判断式が現れることがある．次の例で示す．

$$
\begin{array}{rcll}
\mathrm{ls}(x,y) & = & x = y \wedge \mathbf{emp} & (\phi_1) \\
 & \vee & \exists x'.x \mapsto x' * \mathrm{ls}(x',y) & (\phi_2)
\end{array}
$$

図 1 において，結論の $\mathrm{ls}(x,x') * \mathrm{ls}(x',y) \vdash \mathrm{ls}(x,y)$ は波線部と名前替えをして等しい関係にある．この時，述語が帰納的に定義されていることから，ある条件下でこれを公理と同じように扱い，証明に循環を許すことができる．これを循環証明という．循環証明を適用できる条件を述べる前に，トレース（trace）について述べる．

証明木において対応する述語の列を考える．図 1 では，$\mathrm{ls}(x,x')$ に着目すると，右の枝では述語が展開されて $\mathrm{ls}(z,x')$ になる．次の推論規則適用では変わらずに $\mathrm{ls}(z,x')$ である．このような述語の列を $\mathrm{ls}(x,x')$ のトレースと呼ぶ．厳密な定義は [2] による．

循環証明は以下を満たすときに認められる．

判断式 $\Gamma_1 \vdash \Gamma_1'$ に対する推論規則の適用によって判断式 $\Gamma_2 \vdash \Gamma_2'$ となったとき，

- $\exists\theta.(\Gamma_1 \vdash \Gamma_1')\theta = \Gamma_2 \vdash \Gamma_2'$
- Γ_1, Γ_2 内の同じ位置での P の出現で，それぞれの P は同じトレースに属しており，そのトレース内で (UnfLP) が適用されているようなものが存在する．

葉が公理であるか，循環証明の条件を満たす証明木が構築できる場合，これを証明とする．

図 1 では結論中の $\mathrm{ls}(x,x')$ と 波線部中の $\mathrm{ls}(z,x')$ が同じトレース内にあり，トレース内の述語に (UnfLls) が適用されているので，$\mathrm{ls}(x,x') * \mathrm{ls}(x',y) \vdash \mathrm{ls}(x,y)$ の証明となっている．

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{emp} \vdash \mathbf{emp}}\,(\mathrm{Id})}{\mathbf{emp} * \mathrm{ls}(x,y) \vdash \mathbf{emp} * \mathrm{ls}(x,y)}\,(\mathrm{P\_Id})}{\mathbf{emp} * \mathrm{ls}(x,y) \vdash \mathrm{ls}(x,y)}\,(\mathbf{emp}\mathrm{R})}{x = x' \wedge \mathbf{emp} * \mathrm{ls}(x',y) \vdash \mathrm{ls}(x,y)}\,(=\mathrm{L}) \qquad \dfrac{\dfrac{\underbrace{\mathrm{ls}(z,x') * \mathrm{ls}(x',y) \vdash \mathrm{ls}(z,y)}}{x \mapsto z * \mathrm{ls}(z,x') * \mathrm{ls}(x',y) \vdash x \mapsto z * \mathrm{ls}(x,y)}\,(\mathrm{P\_Id})}{x \mapsto z * \mathrm{ls}(z,x') * \mathrm{ls}(x',y) \vdash \mathrm{ls}(x,y)}\,(\mathrm{UnfRls}_2)}{\mathrm{ls}(x,x') * \mathrm{ls}(x',y) \vdash \mathrm{ls}(x,y)}\,(\mathrm{UnfLls})$$

図 1

この条件は [2] の global trace condition の十分条件となっている．従って，我々の証明体系は [2] の部分体系となっている．循環証明の健全性は [2] で示されている．すなわち，$\phi_1 \vdash \phi_2$ が循環証明を持てば，$\phi_1 \models \phi_2$ である．

4 実装の考え方

本節では実装において必要な判断式の拡張と証明を高速に行うための推論規則の適用順序について述べる．

以下で，証明探索のための手続きを与える．以後，判断式 J と帰納的定義集合 I を入力とし，出力は「妥当である」「妥当でない」「不明」の 3 通りであるとする．

なお，実装ではヒープ論理式をリストで表し，**emp** は空リストとして扱う．よって，(**emp**L) 及び (**emp**R) については以後は考えない．

4.1 正規化

任意の判断式 $\Pi_1 \wedge \Sigma_1 \vdash \Pi_2 \wedge \Sigma_2$ は，3 節で述べた正規化をすることで $\Pi_1' \wedge \Sigma_1' \vdash \Sigma_2'$ の形に変形可能である．ただし，Π_1' は $t_1 = t_2$ の節を含まない．本論文では，これを正規形の判断式であると呼ぶ．

(=R)，(≠R) では充足可能性判定をし，論理式が充足可能でない場合にのみ規則を適用できるが，充足可能であるとした場合には即座に「妥当でない」を出力する．$\Pi_1 \wedge \Sigma_1 \vdash \Pi$ が妥当でなければ，その充分条件の $\Pi_1 \wedge \Sigma_1 \vdash \Pi \wedge \Pi_2 \wedge \Sigma_2$ も妥当でないからである．

正規化は正規形でない判断式が現れた場合に即座に行われる．(=L)，(=R)，(≠R) をそれ以上適用不可能であるまで適用する．

4.2 拡張判断式

2 節で定義した判断式を拡張する．入力 J が与えられたときに，J に含まれる帰納的述語それぞれを区別する番号を付ける．これをタグと呼ぶ．帰納的述語 P のタグが i である場合，P_i と表記する．次に，証明木において祖先となるノードの判断式の列である α と証明木において左辺の展開規則 (Unf$L P$) がいくつ前に適用されたかを示す π を定義する．α は空列，π は 0 が初期値であり，推論規則を適用する毎にそのときの判断式を α に加え，π を 1 増やす．ただし，(Unf$L P$) の場合は π を 0 にする．

循環証明に相当する規則 (Downlink) を以下で定義する．

$$\dfrac{|\alpha| > n > \pi \quad \exists\theta.\alpha_n = (\phi_1 \vdash \phi_3)[\theta] \quad \phi_2 \text{ matches } \phi_3}{(\alpha, \pi, \phi_1 \vdash \phi_2)}\,(\text{Downlink})$$

matches はタグを無視して等しいことを表す．これは，[2] の拡張判断式（augmented sequents）による．

次に，J の左辺及び右辺のタグ付き述語を 1 つずつ要素に持ったそれぞれの列 q_L, q_R と，次の展開規則を左辺と右辺どちらを優先するべきかを表す真偽値 f_L を定義する．これらによって述語の展開順序を制御する．q_L, q_R の初期値はそれぞれ左辺，右辺のタグ付き述語を 1 つずつ要素に持っているが，順序は任意に定めてよい．f_L の初期値は真である．

f_L が真の場合は q_L, q_R の順に，偽の場合は q_R, q_L の順に繋いだ新たな列 q を作る．q の先頭の述語を (Unf$L P$) または (Unf$R P_i$) によって展開する．(Unf$R P_i$) は全ての i について試みる．その後証明ができなかった場合は q の 2 番目の述語を同様に列の最後まで続ける．展開のとき，展開する述語のタグが q_L に含まれていれば，それを 削除し，展開した後に現れる述語を q_L の末尾に加える．展開する述語のタグが q_R に含まれていれば，それを削除し，展開した後に現れる述語を

q_R の末尾に加える．また，(UnfLP) を適用した場合は f_L を偽に，(UnfRP_i) を適用した場合は f_L を真にする．例えば，$q_L = [P_1; P_2; Q_3], q_R = [P_4; Q_5]$ のときに P_2 が展開された場合は，$q_L = [P_1; Q_3; P_2], q_R = [P_4; Q_5]$ となり，f_L は偽となる．

展開規則を適用する順や条件を細かく指定する理由を述べる．循環証明では，両辺の対応する述語を展開することが必要である．そのため左辺と右辺を交互に展開することをまず試みる．ただし，左辺と右辺それぞれどの述語が循環証明に対応する述語か分からないため，それを順繰りに試すために q_L, q_R を用いる．本論文の実装では，可能な展開順全てを試していることになる．

しかし，左辺や右辺を連続で展開しなくてはならない場合も存在する．例えば，3 節の ls と，

$$\begin{aligned} \mathrm{ls2}(x, y) \quad &= \quad x = y \wedge \mathbf{emp} \\ &\mid \quad \exists x', x''. x \mapsto x' * x' \mapsto x'' * \mathrm{ls}(x'', y) \end{aligned}$$

で定義される ls2 による $\mathrm{ls2}(x, y) \vdash \mathrm{ls}(x, y)$ は右辺を連続で展開する必要がある．そのため q_L, q_R を結合し，左辺や右辺を連続で展開する場合をも計算している．これは左辺や右辺を交互に展開することを試みた後に試みられる．

また，実行時には (UnfLP) を無限に適用可能であるので，証明器が有限で止まらなくなる可能性がある．展開回数の上限 $d_{\max}$ を与えて，証明器を有限で止まるようにする．ただし，展開回数の上限を超えて展開できずに証明ができなかった場合は結果を「不明」として出力する．そのために現在の展開回数を記憶する d を新たに定義する．

本論文では $J, \alpha, \pi, q_L, q_R, f_L, d$ を合わせて拡張判断式と呼ぶ．

4.3 (UnfRP_i) の適用の制限

(UnfLP) 規則によって帰納的述語を展開することで新しく現れる変数は判断式で既に使われているもの以外でなければならない．一方，(UnfRP_i) 規則によって帰納的述語を展開することで新しく得られる変数は制約はない．

実装においては，(UnfLP) 規則ではとにかく新たな変数を導入すればよいが，(UnfRP_i) 規則では循環証明を生成できるような変数の取り方をすべきである．

ここで，P の帰納的定義の定義節が以下の形で記述してあるとする．

$$\exists \vec{z}. \phi_i = \exists \vec{z}. x \mapsto (\vec{y}) * \phi_i'$$

左辺になんらかの項列 $\vec{v}$ による $x \mapsto (\vec{v})$ が含まれている場合に，$\vec{y}$ を $\vec{v}$ にマッチングする．ただし，マッチングの変数は $\vec{z}$ とする．マッチングが可能であった場合，$x \mapsto (\vec{y}) * \phi_i'$ に対してマッチング結果である $\vec{z}$ への代入を行い，(UnfRP_i) を適用する．マッチングができない場合は展開を行わない．

これは，左辺に $x \mapsto (\vec{v})$ が含まれていない場合，新たに右辺に現れる $x \mapsto (\vec{y})$ によって証明ができなくなるためである．左辺に $x \mapsto (\vec{y})$ を含む判断式を公理から導くには (P_Id) 規則を用いざるを得ない．よって，左辺にも $x \mapsto (\vec{y})$ が含まれているべきである．そのため左辺に $x \mapsto (\vec{v})$ が含まれている場合のみ (UnfRP_i) 規則を適用する．

4.4 適用順序

証明器は J から拡張判断式を作り，拡張判断式を推論規則に相当する操作によって変形していく．初期値は $\alpha = [], \pi = 0, q_L = [], q_R = [], f_L = \text{真}, c = 0$ である．ただし，[] は空列である．

「正規化」，「(Id)」,「(P_Id)」,「(Downlink)」,「(UnfLP) または (UnfRP_i) の適用の操作」をこの順序で適用し，新たに拡張判断式が得られた場合はその拡張判断式に対して繰り返してこれを適用する．「(UnfLP) または (UnfRP_i) の適用の操作」において $c > c_{\max}$ あるいは (UnfRP_i) が条件に合わずに展開されない場合は「不明」を出力する．正規化できなかったとき，または述語が一つもなくどのどの推論規則も適用できない場合は「妥当でない」を出力する．全ての葉で (Id) または (Downlink) が適用可能で，証明木が構築できた場合「妥当である」を出力する．

5 実行例

5.1 実行内容

4 節に従って自動証明器のプログラムを OCaml で記述し，表 1 で定義される述語を含む表 2 の判断式を入力として証明器を動作させた．$d_{\max}$ は 4 および 10 で

実行した．構文解析部及び，シンボリックヒープの充足可能性判定器は東邦大の木村による実装を利用した．この充足可能性判定は [4] のアルゴリズムの基づくものである．動作環境は DELL optiplex 9020, プロセッサは Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz 3.60GHz, メモリは 8GB，1600MHz DDR SDRAM デュアルインターリーブ， OS は Windows 10 Pro で仮想マシン VMware(R) Workstation 12 Player バージョン 12.1.0 build-3272444 を使って Ubuntu 15.04 64 ビットを起動し，RAM 2GB, プロセッサ数 2 の設定で，OCaml version 4.01.0 にて動作させた．実行を 5 回行い，それぞれの実行時間の平均値の 1/10 ミリ秒の位を四捨五入して実行時間とした．

また，証明可能だった場合は証明木を LaTeX ファイルに出力させて可視化した．

5.2 結果

表 2 の判断式を入力として証明器を動作させたときの実行時間を表 3 に示す．

$d = 4$ では表 2 の全ての例で証明ができた．$d = 10$ では全ての例で証明ができたが，全て実行時間がほぼ同じか増え，また証明木の形が $d = 4$ とは異なるものもあった．

表 2 の番号 1,4 の証明木を図 2,3 に示す．そのほかの証明木については付録 A に載せる．

5.3 考察

証明木の形が異なったのは循環証明に関与しない述語の展開を含んでいたためであった．図 4 は表 2 の番号 4 の $d = 10$ での証明木であり，図 3 と形が異なっている．

$d_{max} = 10$ では実行時間が増えているが，d_{max} が大きいほど適用可能な規則のパターンが多いためである．[2] の推論規則 $(*)$ をそのまま用いる場合，d_{max} が大きくなったときに推論規則の適用パターンが爆発的に多くなるが，本論文の定義ではその場合より時間の増加を抑えられているといえる．

証明木の形が異なったのは深さ優先で探索をしたため，$d_{max} = 4$ では打ち切られて発見されなかった，循環証明に関与しない述語の展開を含んでいる証明が先に発見されたためと考えられる．幅優先探索をするプログラムに変更することで改善されると思われる．

図 3 において，番号 1,5 は実行時間が $d_{max} = 4$ と $d_{max} = 10$ でほぼ変わらなかった．番号 2,3,4 は同じタグの述語を連続で展開する必要があるが，番号 1,5 は交互に展開をするだけなので，d_{max} によらず即座に証明を発見できたためであると考えられる．

6 おわりに

6.1 まとめ

本研究では帰納的述語を含む分離論理によるプログラム検証の自動化に向けて，含意を表す判断式の半自動証明手続を提案し，その実装を与えた．実装ではプログラムが有限で止まるように展開の回数の上限を定めた．述語展開の順序を制御することにより，効率のよい証明探索を目指した．また，証明可能な場合は証明図を出力するようにした．

例として様々なリストを含む判断式を入力として動作させ，正しい結果が得られることを確認した．その際，展開の回数の上限を大きくすると実行時間が増え，一部の例では出力させた証明木の形が異なった．展開の回数が増えたために循環証明に関係のない述語の展開をも行ったものが先に証明として発見されたためと思われる．

6.2 関連研究と今後の課題

制限のないシンボリック・ヒープに対する妥当性判定は決定不可能であることが知られている [1]．従って，妥当性判定を完全に決定するためには論理式に一定の制限を課すことが必要である．判断式の妥当性判定についての既存研究としては，[2] [3] [5] などがある．[2] [3] は本研究と同様に循環証明を用いた証明探索であり，[5] はオートマトンによる手法である．[2] [3] では妥当性判定は高速であるが述語定義に制約が設けられている．対して，[5] はより一般の述語定義において決定可能であるがより時間がかかる．本研究の証明器は多くの判断式について [2] より速く，[3] と同等程度の速度で証明をした．本研究で提案した手続きによって妥当性が決定できる判断式のクラスを明らかにすること，また，より高速な証明探索を実

表 1　述語の定義

述語	意味	定義
ls	空リストを含むリスト	$\mathrm{ls}(x,y) = x = y \wedge \mathbf{emp} \vee \exists x'.x \mapsto x' * \mathrm{ls}(x',y)$
lsE/lsO	空リストを含む，長さが（偶数／奇数）のリスト	$\mathrm{lsE}(x,y) = x = y \wedge \mathbf{emp} \vee \exists x'.x \mapsto x' * \mathrm{lsO}(x',y)$ $\mathrm{lsO}(x,y) = \exists x'.x \mapsto x' * \mathrm{lsE}(x',y)$
DLL	空リストを含まない双方向リスト	$\mathrm{DLL}(a,b,c,d) = a = b \wedge c = d \wedge \mathbf{emp} \vee \exists x.a \mapsto (x,d) * \mathrm{DLL}(x,b,c,a)$

表 2　証明した判断式

番号	意味	判断式
1	リストの結合	$\mathrm{ls}(x,y) * \mathrm{ls}(y,z) \vdash \mathrm{ls}(x,z)$
2	奇数の長さのリストと奇数の長さのリストの結合が偶数の長さのリストになること	$\mathrm{lsO}(x,y) * \mathrm{lsO}(y,z) \vdash \mathrm{lsE}(x,z)$
3	奇数の長さのリストと偶数の長さのリストの結合が奇数の長さのリストになること	$\mathrm{lsO}(x,y) * \mathrm{lsE}(y,z) \vdash \mathrm{lsO}(x,z)$
4	偶数の長さのリストと偶数の長さのリストの結合が偶数の長さのリストになること	$\mathrm{lsE}(x,y) * \mathrm{lsE}(y,z) \vdash \mathrm{lsE}(x,z)$
5	双方向リストの結合	$\mathrm{DLL}(x_5,x_2,x_3,x_6) * \mathrm{DLL}(x_1,x_5,x_6,x_4) \vdash \mathrm{DLL}(x_1,x_2,x_3,x_4)$

表 3　判断式の証明時間．d は展開回数の上限．

番号	$d = 4$ での実行時間（ミリ秒）	$d = 10$ での実行時間（ミリ秒）
1	3	4
2	4	7
3	0	22
4	6	43
5	22	21

現するし，既存の証明器との性能比較を行うことは今後の課題である．

また，本研究で考えている循環証明体系の完全性について考察することも今後の課題である．予想として，この体系は完全ではない，すなわち，妥当な判断式であって，証明不可能なものが存在する．一定の判断式のクラスに対して完全な循環証明体系を与えることは興味深い問題である．

参考文献

[1] Antonopoulos, T., Gorogiannis, N., Haase, C., Kanovich, M., and Ouaknine, J.: Foundations for decision problems in separation logic with general inductive predicates, *International Conference on Foundations of Software Science and Computation Structures*, Springer Berlin Heidelberg, 2014, pp. 411–425.

ls

実行時間 ： 0.003 秒

$$\dfrac{\dfrac{}{\mathbf{emp} \vdash \mathbf{emp}}\ (\mathrm{Id})}{\mathrm{ls}_2(x,z) \vdash \mathrm{ls}_3(x,z)}\ (\mathrm{P\_Id}) \quad \dfrac{\dfrac{\dfrac{(4\text{ 段下の式へ})}{\mathrm{ls}_1('1,y) * \mathrm{ls}_2(y,z) \vdash \mathrm{ls}_3('1,z)}\ (\mathrm{downlink})}{x \mapsto '1 * \mathrm{ls}_1('1,y) * \mathrm{ls}_2(y,z) \vdash x \mapsto '1 * \mathrm{ls}_3('1,z)}\ (\mathrm{P\_Id})}{x \mapsto '1 * \mathrm{ls}_1('1,y) * \mathrm{ls}_2(y,z) \vdash \mathrm{ls}_3(x,z)}\ (\mathrm{lsR})}{\mathrm{ls}_1(x,y) * \mathrm{ls}_2(y,z) \vdash \mathrm{ls}_3(x,z)}\ (\mathrm{lsL})$$

図 2　表 3 の番号 1 の $d = 4$ での証明木

lsEE

実行時間 ： 0.006 秒

$$\dfrac{\dfrac{}{\mathbf{emp} \vdash \mathbf{emp}}\ (\mathrm{Id})}{\mathrm{lsE}_2(x,z) \vdash \mathrm{lsE}_3(x,z)}\ (\mathrm{P\_Id}) \quad \dfrac{\dfrac{\dfrac{\dfrac{(7\text{ 段下の式へ})}{\mathrm{lsE}_1('2,y) * \mathrm{lsE}_2(y,z) \vdash \mathrm{lsE}_3('2,z)}\ (\mathrm{downlink})}{'1 \mapsto '2 * \mathrm{lsE}_1('2,y) * \mathrm{lsE}_2(y,z) \vdash '1 \mapsto '2 * \mathrm{lsE}_3('2,z)}\ (\mathrm{P\_Id})}{'1 \mapsto '2 * \mathrm{lsE}_1('2,y) * \mathrm{lsE}_2(y,z) \vdash \mathrm{lsO}_3('1,z)}\ (\mathrm{lsOR})}{\mathrm{lsO}_1('1,y) * \mathrm{lsE}_2(y,z) \vdash \mathrm{lsO}_3('1,z)}\ (\mathrm{lsOL}) \quad \dfrac{\dfrac{}{x \mapsto '1 * \mathrm{lsO}_1('1,y) * \mathrm{lsE}_2(y,z) \vdash x \mapsto '1 * \mathrm{lsO}_3('1,z)}\ (\mathrm{P\_Id})}{x \mapsto '1 * \mathrm{lsO}_1('1,y) * \mathrm{lsE}_2(y,z) \vdash \mathrm{lsE}_3(x,z)}\ (\mathrm{lsER})}{\mathrm{lsE}_1(x,y) * \mathrm{lsE}_2(y,z) \vdash \mathrm{lsE}_3(x,z)}\ (\mathrm{lsEL})$$

図 3　表 3 の番号 4 の $d = 4$ での証明木

lsEE

実行時間 ： 0.043 秒

$$\dfrac{\dfrac{}{\mathbf{emp} \vdash \mathbf{emp}}\ (\mathrm{Id})}{\mathrm{lsE}_2(x,z) \vdash \mathrm{lsE}_3(x,z)}\ (\mathrm{P\_Id}) \quad \dfrac{\dfrac{}{\mathbf{emp} \vdash \mathbf{emp}}\ (\mathrm{Id})}{\mathrm{lsO}_1('1,y) \vdash \mathrm{lsO}_3('1,y)}\ (\mathrm{P\_Id}) \quad \dfrac{\dfrac{\dfrac{}{\mathbf{emp} \vdash \mathbf{emp}}\ (\mathrm{Id})}{'2 \mapsto '4 * \mathrm{lsE}_2('4,z) \vdash '2 \mapsto '4 * \mathrm{lsE}_3('4,z)}\ (\mathrm{P\_Id})}{'2 \mapsto '4 * \mathrm{lsE}_2('4,z) \vdash \mathrm{lsO}_3('2,z)}\ (\mathrm{lsOR}) \quad \dfrac{\dfrac{\dfrac{}{'2 \mapsto '4 * \mathrm{lsE}_2('4,z) * '3 \mapsto '2 \vdash '3 \mapsto '2 * \mathrm{lsO}_3('2,z)}\ (\mathrm{P\_Id})}{'2 \mapsto '4 * \mathrm{lsE}_2('4,z) * '3 \mapsto '2 \vdash \mathrm{lsE}_3('3,z)}\ (\mathrm{lsER}) \quad \dfrac{\dfrac{\dfrac{\dfrac{(7\text{ 段下の式へ})}{'5 \mapsto '6 * \mathrm{lsE}_1('6,y) * '2 \mapsto '4 * \mathrm{lsE}_2('4,z) * y \mapsto '2 \vdash '5 \mapsto '6 * \mathrm{lsE}_3('6,z)}\ (\mathrm{downlink})}{'5 \mapsto '6 * \mathrm{lsE}_1('6,y) * '2 \mapsto '4 * \mathrm{lsE}_2('4,z) * y \mapsto '2 \vdash \mathrm{lsO}_3('5,z)}\ (\mathrm{lsOR})}{\mathrm{lsO}_1('5,y) * '2 \mapsto '4 * \mathrm{lsE}_2('4,z) * y \mapsto '2 \vdash \mathrm{lsO}_3('5,z)}\ (\mathrm{lsOL})}{'3 \mapsto '5 * \mathrm{lsO}_1('5,y) * '2 \mapsto '4 * \mathrm{lsE}_2('4,z) * y \mapsto '2 \vdash '3 \mapsto '5 * \mathrm{lsO}_3('5,z)}\ (\mathrm{P\_Id})}{'3 \mapsto '5 * \mathrm{lsO}_1('5,y) * '2 \mapsto '4 * \mathrm{lsE}_2('4,z) * y \mapsto '2 \vdash \mathrm{lsE}_3('3,z)}\ (\mathrm{lsER})}{'2 \mapsto '4 * \mathrm{lsE}_2('4,z) * '1 \mapsto '3 * \mathrm{lsE}_3('3,z)}\ (\mathrm{P\_Id}) \quad \dfrac{}{'3 \mapsto '5 * \mathrm{lsO}_1('5,y) * '2 \mapsto '4 * \mathrm{lsE}_2('4,z) * '1 \mapsto '3 * y \mapsto '2 \vdash '1 \mapsto '3 * \mathrm{lsE}_3('3,z)}\ (\mathrm{P\_Id})}{'2 \mapsto '4 * \mathrm{lsE}_2('4,z) * '1 \mapsto '3 * \mathrm{lsE}_1('3,y) * y \mapsto '2 \vdash '1 \mapsto '3 * \mathrm{lsE}_3('3,z)}\ (\mathrm{lsEL})}{'1 \mapsto '3 * \mathrm{lsE}_1('3,y) * y \mapsto '2 * \mathrm{lsO}_2('2,z) \vdash '1 \mapsto '3 * \mathrm{lsE}_3('3,z)}\ (\mathrm{lsOR})}{'1 \mapsto '3 * \mathrm{lsE}_1('3,y) * y \mapsto '2 * \mathrm{lsO}_2('2,z) \vdash \mathrm{lsO}_3('1,z)}\ (\mathrm{lsOL})}{y \mapsto '2 * \mathrm{lsO}_2('2,z) * \mathrm{lsO}_1('1,y) \vdash \mathrm{lsO}_3('1,z)}\ (\mathrm{lsEL})}{\mathrm{lsO}_1('1,y) * \mathrm{lsE}_2(y,z) \vdash \mathrm{lsO}_3('1,z)}$$
$$\dfrac{}{x \mapsto '1 * \mathrm{lsO}_1('1,y) * \mathrm{lsE}_2(y,z) \vdash x \mapsto '1 * \mathrm{lsO}_3('1,z)}\ (\mathrm{P\_Id})$$
$$\dfrac{}{x \mapsto '1 * \mathrm{lsO}_1('1,y) * \mathrm{lsE}_2(y,z) \vdash \mathrm{lsE}_3(x,z)}\ (\mathrm{lsER})$$
$$\mathrm{lsE}_1(x,y) * \mathrm{lsE}_2(y,z) \vdash \mathrm{lsE}_3(x,z)\ (\mathrm{lsEL})$$

図 4　表 3 の番号 4 の $d = 10$ での証明木

[2] Brotherston, J., Distefano, D. and Petersen, R. L.: Automated cyclic entailment proofs in separation logic, *Automated Deduction (CADE-23)*, Springer, 2011, pp. 131-146.

[3] Brotherston, J., Gorogiannis, N. and Petersen, R. L.: A generic cyclic theorem prover, *Programming Languages and Systems*, Springer, 2012, pp. 350-367.

[4] Brotherston, J., Fuhs, C., Pérez, J. A. N., and Gorogiannis, N.: A decision procedure for satisfiability in separation logic with inductive predicates, *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, ACM, 2014, p. 25.

[5] Iosif, R., Rogalewicz, A. and Vojnar, T.: Deciding entailments in inductive separation logic with tree automata, *Automated Technology for Verification and Analysis*, Springer, 2014, pp. 201-218.

[6] Reynolds, J. C.: Separation logic: A logic for shared mutable data structures, *Logic in Computer Science, 2002. Proceedings. 17th Annual IEEE Symposium on*, IEEE, 2002, pp. 55-74.

A　付録: 5 節の実行結果

5 節で使用した証明器によって LaTeX ファイルで出力された証明木をコンパイルして得られた証明木を載せる．5 章で載せたものについては省く．

ls

実行時間 ： 0.004 秒

$$\dfrac{\dfrac{}{\mathbf{emp} \vdash \mathbf{emp}}\ (\mathrm{Id})}{\mathrm{ls}_2(x,z) \vdash \mathrm{ls}_3(x,z)}\ (\mathrm{P\_Id}) \quad \dfrac{\dfrac{\text{(4 段下の式へ)}}{\mathrm{ls}_1('1,y) * \mathrm{ls}_2(y,z) \vdash \mathrm{ls}_3('1,z)}\ (\mathrm{downlink})}{x \mapsto '1 * \mathrm{ls}_1('1,y) * \mathrm{ls}_2(y,z) \vdash x \mapsto '1 * \mathrm{ls}_3('1,z)}\ (\mathrm{P\_Id})}{x \mapsto '1 * \mathrm{ls}_1('1,y) * \mathrm{ls}_2(y,z) \vdash \mathrm{ls}_3(x,z)}\ (\mathrm{lsR})$$
$$\mathrm{ls}_1(x,y) * \mathrm{ls}_2(y,z) \vdash \mathrm{ls}_3(x,z)\ (\mathrm{lsL})$$

図 5　表 **3** の番号 **1** の $d = 10$ での証明木

lsOO

実行時間 ： 0.004 秒

$$\dfrac{\dfrac{\dfrac{}{\mathbf{emp} \vdash \mathbf{emp}}\ (\mathrm{Id})}{\mathrm{lsO}_2('1,z) \vdash \mathrm{lsO}_3('1,z)}\ (\mathrm{P\_Id}) \quad \dfrac{\dfrac{\text{(7 段下の式へ)}}{\mathrm{lsO}_1('2,y) * \mathrm{lsO}_2(y,z) \vdash \mathrm{lsE}_3('2,z)}\ (\mathrm{downlink})}{'1 \mapsto '2 * \mathrm{lsO}_1('2,y) * \mathrm{lsO}_2(y,z) \vdash '1 \mapsto '2 * \mathrm{lsE}_3('2,z)}\ (\mathrm{P\_Id})}{'1 \mapsto '2 * \mathrm{lsO}_1('2,y) * \mathrm{lsO}_2(y,z) \vdash \mathrm{lsO}_3('1,z)}\ (\mathrm{lsOR})$$
$$\dfrac{\mathrm{lsE}_1('1,y) * \mathrm{lsO}_2(y,z) \vdash \mathrm{lsO}_3('1,z)}{x \mapsto '1 * \mathrm{lsE}_1('1,y) * \mathrm{lsO}_2(y,z) \vdash x \mapsto '1 * \mathrm{lsO}_3('1,z)}\ (\mathrm{lsEL})\ (\mathrm{P\_Id})$$
$$\dfrac{x \mapsto '1 * \mathrm{lsE}_1('1,y) * \mathrm{lsO}_2(y,z) \vdash \mathrm{lsE}_3(x,z)}{\mathrm{lsO}_1(x,y) * \mathrm{lsO}_2(y,z) \vdash \mathrm{lsE}_3(x,z)}\ (\mathrm{lsER})\ (\mathrm{lsOL})$$

図 6　表 **3** の番号 **2** の $d = 4$ での証明木

lsOO

実行時間 ： 0.007 秒

$$\dfrac{\dfrac{\dfrac{}{\mathbf{emp} \vdash \mathbf{emp}}\ (\mathrm{Id})}{\mathrm{lsO}_2('1,z) \vdash \mathrm{lsO}_3('1,z)}\ (\mathrm{P\_Id}) \quad \dfrac{\dfrac{\text{(7 段下の式へ)}}{\mathrm{lsO}_1('2,y) * \mathrm{lsO}_2(y,z) \vdash \mathrm{lsE}_3('2,z)}\ (\mathrm{downlink})}{'1 \mapsto '2 * \mathrm{lsO}_1('2,y) * \mathrm{lsO}_2(y,z) \vdash '1 \mapsto '2 * \mathrm{lsE}_3('2,z)}\ (\mathrm{P\_Id})}{'1 \mapsto '2 * \mathrm{lsO}_1('2,y) * \mathrm{lsO}_2(y,z) \vdash \mathrm{lsO}_3('1,z)}\ (\mathrm{lsOR})$$
$$\dfrac{\mathrm{lsE}_1('1,y) * \mathrm{lsO}_2(y,z) \vdash \mathrm{lsO}_3('1,z)}{x \mapsto '1 * \mathrm{lsE}_1('1,y) * \mathrm{lsO}_2(y,z) \vdash x \mapsto '1 * \mathrm{lsO}_3('1,z)}\ (\mathrm{lsEL})\ (\mathrm{P\_Id})$$
$$\dfrac{x \mapsto '1 * \mathrm{lsE}_1('1,y) * \mathrm{lsO}_2(y,z) \vdash \mathrm{lsE}_3(x,z)}{\mathrm{lsO}_1(x,y) * \mathrm{lsO}_2(y,z) \vdash \mathrm{lsE}_3(x,z)}\ (\mathrm{lsER})\ (\mathrm{lsOL})$$

図 7　表 **3** の番号 **2** の $d = 10$ での証明木

lsOE

実行時間 ： 0. 秒

$$\dfrac{\dfrac{\dfrac{}{\mathbf{emp} \vdash \mathbf{emp}}\ (\mathrm{Id})}{\mathrm{lsE}_2('1,z) \vdash \mathrm{lsE}_3('1,z)}\ (\mathrm{P\_Id}) \quad \dfrac{\dfrac{\text{(7 段下の式へ)}}{\mathrm{lsO}_1('2,y) * \mathrm{lsE}_2(y,z) \vdash \mathrm{lsO}_3('2,z)}\ (\mathrm{downlink})}{'1 \mapsto '2 * \mathrm{lsO}_1('2,y) * \mathrm{lsE}_2(y,z) \vdash '1 \mapsto '2 * \mathrm{lsO}_3('2,z)}\ (\mathrm{P\_Id})}{'1 \mapsto '2 * \mathrm{lsO}_1('2,y) * \mathrm{lsE}_2(y,z) \vdash \mathrm{lsE}_3('1,z)}\ (\mathrm{lsER})$$
$$\dfrac{\mathrm{lsE}_1('1,y) * \mathrm{lsE}_2(y,z) \vdash \mathrm{lsE}_3('1,z)}{x \mapsto '1 * \mathrm{lsE}_1('1,y) * \mathrm{lsE}_2(y,z) \vdash x \mapsto '1 * \mathrm{lsE}_3('1,z)}\ (\mathrm{lsEL})\ (\mathrm{P\_Id})$$
$$\dfrac{x \mapsto '1 * \mathrm{lsE}_1('1,y) * \mathrm{lsE}_2(y,z) \vdash \mathrm{lsO}_3(x,z)}{\mathrm{lsO}_1(x,y) * \mathrm{lsE}_2(y,z) \vdash \mathrm{lsO}_3(x,z)}\ (\mathrm{lsOR})\ (\mathrm{lsOL})$$

図 8　表 **3** の番号 **3** の $d = 4$ での証明木

lsOE

実行時間 ： 0.022 秒

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{emp} \vdash \mathbf{emp}}\ (\mathrm{Id})}{\mathrm{lsE}_2('1, z) \vdash \mathrm{lsE}_3('1, z)}\ (\mathrm{P\_Id}) \quad \dfrac{\dfrac{(7\text{ 段下の式へ})}{\mathrm{lsO}_1('2, y) * \mathrm{lsE}_2(y, z) \vdash \mathrm{lsO}_3('2, z)}\ (\mathrm{downlink})}{'1 \mapsto '2 * \mathrm{lsO}_1('2, y) * \mathrm{lsE}_2(y, z) \vdash '1 \mapsto '2 * \mathrm{lsO}_3('2, z)}\ (\mathrm{P\_Id})}{'1 \mapsto '2 * \mathrm{lsO}_1('2, y) * \mathrm{lsE}_2(y, z) \vdash \mathrm{lsE}_3('1, z)}\ (\mathrm{lsER})}{\mathrm{lsE}_1('1, y) * \mathrm{lsE}_2(y, z) \vdash \mathrm{lsE}_3('1, z)}\ (\mathrm{lsEL})}{x \mapsto '1 * \mathrm{lsE}_1('1, y) * \mathrm{lsE}_2(y, z) \vdash x \mapsto '1 * \mathrm{lsE}_3('1, z)}\ (\mathrm{P\_Id})}{x \mapsto '1 * \mathrm{lsE}_1('1, y) * \mathrm{lsE}_2(y, z) \vdash \mathrm{lsO}_3(x, z)}\ (\mathrm{lsOR})}{\mathrm{lsO}_1(x, y) * \mathrm{lsE}_2(y, z) \vdash \mathrm{lsO}_3(x, z)}\ (\mathrm{lsOL})$$

図 9　表 3 の番号 3 の $d = 10$ での証明木

dll

実行時間 ： 0.022 秒

$$\dfrac{\dfrac{\dfrac{}{\mathbf{emp} \vdash \mathbf{emp}}\ (\mathrm{Id})}{\mathrm{DLL}_1(x_1, x_2, x_3, x_6) \vdash \mathrm{DLL}_3(x_1, x_2, x_3, x_6)}\ (\mathrm{P\_Id}) \quad \dfrac{\dfrac{(4\text{ 段下の式へ})}{\mathrm{DLL}_2('1, x_5, x_6, x_1) * \mathrm{DLL}_1(x_5, x_2, x_3, x_6) \vdash \mathrm{DLL}_3('1, x_2, x_3, x_1)}\ (\mathrm{downlink})}{x_1 \mapsto '1, x_4 * \mathrm{DLL}_2('1, x_5, x_6, x_1) * \mathrm{DLL}_1(x_5, x_2, x_3, x_6) \vdash x_1 \mapsto '1, x_4 * \mathrm{DLL}_3('1, x_2, x_3, x_1)}\ (\mathrm{P\_Id})}{x_1 \mapsto '1, x_4 * \mathrm{DLL}_2('1, x_5, x_6, x_1) * \mathrm{DLL}_1(x_5, x_2, x_3, x_6) \vdash \mathrm{DLL}_3(x_1, x_2, x_3, x_4)}\ (\mathrm{DLLR})}{\mathrm{DLL}_1(x_5, x_2, x_3, x_6) * \mathrm{DLL}_2(x_1, x_5, x_6, x_4) \vdash \mathrm{DLL}_3(x_1, x_2, x_3, x_4)}\ (\mathrm{DLLL})$$

図 10　表 3 の番号 5 の $d = 4$ での証明木

dll

実行時間 ： 0.021 秒

$$\dfrac{\dfrac{\dfrac{}{\mathbf{emp} \vdash \mathbf{emp}}\ (\mathrm{Id})}{\mathrm{DLL}_1(x_1, x_2, x_3, x_6) \vdash \mathrm{DLL}_3(x_1, x_2, x_3, x_6)}\ (\mathrm{P\_Id}) \quad \dfrac{\dfrac{(4\text{ 段下の式へ})}{\mathrm{DLL}_2('1, x_5, x_6, x_1) * \mathrm{DLL}_1(x_5, x_2, x_3, x_6) \vdash \mathrm{DLL}_3('1, x_2, x_3, x_1)}\ (\mathrm{downlink})}{x_1 \mapsto '1, x_4 * \mathrm{DLL}_2('1, x_5, x_6, x_1) * \mathrm{DLL}_1(x_5, x_2, x_3, x_6) \vdash x_1 \mapsto '1, x_4 * \mathrm{DLL}_3('1, x_2, x_3, x_1)}\ (\mathrm{P\_Id})}{x_1 \mapsto '1, x_4 * \mathrm{DLL}_2('1, x_5, x_6, x_1) * \mathrm{DLL}_1(x_5, x_2, x_3, x_6) \vdash \mathrm{DLL}_3(x_1, x_2, x_3, x_4)}\ (\mathrm{DLLR})}{\mathrm{DLL}_1(x_5, x_2, x_3, x_6) * \mathrm{DLL}_2(x_1, x_5, x_6, x_4) \vdash \mathrm{DLL}_3(x_1, x_2, x_3, x_4)}\ (\mathrm{DLLL})$$

図 11　表 3 の番号 5 の $d = 10$ での証明木