プログラムの正しさを証明する

分離論理入門

中澤 巧爾　(名古屋大学大学院情報学研究科)*

概　　要

本講演では，プログラムの性質を数理論理学，とくに，証明論的手法で検証するための枠組みであるホーア論理，および，その変種である分離論理を紹介する．これらの体系においては，プログラムの性質はホーア・トリプルと呼ばれる三つ組 $\{A\}P\{B\}$ によって表現される．これは，事前条件として論理式 A が成立している状況で，プログラム P が実行されれば，事後条件 B が成立する状況に至ることを意味する．ホーア論理は，このホーア・トリプルを導出する証明体系である．さらにこの対象となるプログラムを C 言語のようなヒープ・メモリを操作するプログラムに拡張し，仕様記述の論理式をヒープ・メモリの状態を表現する特別な結合子などで拡張したものが分離論理である．本講演では，分離論理の基本を解説した上で，講演者を含むプロジェクトで現在進行中の，分離論理を帰納的述語で拡張した論理体系に関する研究について紹介する．

1. ホーア論理

本章では，ホーア論理 [8] を紹介する．より詳しい解説は，[2, 16] などを参照のこと．

まず，簡単な手続型プログラミング言語とその意味論を与える．プログラム実行中の状態は変数に対する値割当てとして表現され，プログラムの意味は，その状態を遷移させるものと考えられる．

定義 1.1 (プログラムと状態)　1. Prog_H を以下で定義されるプログラムの集合とする．

$$P ::= \texttt{skip} \mid x := a \mid P_1;P_2 \mid \texttt{if}(b)P_1\ \texttt{else}\ P_2 \mid \texttt{while}(b)\ \texttt{do}\ P$$

ここで，x は変数，a は算術式，b は真偽式であり，算術式と真偽式は以下で定義される．

$$a ::= n\text{（整数定数）} \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2$$
$$b ::= \texttt{true} \mid a_1 = a_2 \mid a_1 < a_2 \mid \neg b \mid b_1 \wedge b_2$$

2. Stat_H を，変数から整数値への有限関数全体からなる集合とする．Stat_H の要素をストアと呼ぶ．$[x_1{\to}i_1, \ldots, x_n{\to}i_n]$ によって，$\mathrm{dom}(s) = \{x_1, \ldots, x_n\}$，かつ，各 $j = 1, \ldots n$ について $s(x_j) = i_j$ であるようなストア s を表す．とくに，$[\,]$ によって，定義域が空集合であるようなストアを表す．また，ストア s に対して，$s[x{\to}n]$ は，次で定義されるストアである．

$$s[x{\to}n](y) = \begin{cases} n & (x \text{ と } y \text{ は同一の変数}) \\ s(y) & (x \text{ と } y \text{ は異なる変数}) \end{cases}$$

* Koji Nakazawa, Graduate School of Informatics, Nagoya University
e-mail: knak@i.nagoya-u.ac.jp
web: http://www.sqlab.jp/~nakazawa/

算術式や真偽式の意味は標準的な整数の構造上で与える．算術式 a の値は，各変数の値が決まれば計算できる．すなわち，a の意味 $[\![a]\!]$ は，ストアから整数への関数として与えられる．真偽式 b の意味 $[\![b]\!]$ は，b を真とするストア s 全体の集合とする．

定義 1.2 (プログラムの操作的意味論) プログラム $P \in \mathsf{Prog}_H$ に対して，Stat_H 上の二項関係 $[\![P]\!]$ を次の公理と規則によって帰納的に定義する．

$$\frac{}{s[\![\mathtt{skip}]\!]s}\ \mathsf{OHskip} \qquad \frac{}{s[\![x := a]\!]s[x{\to}[\![a]\!](s)]}\ \mathsf{OHasgn} \qquad \frac{s[\![P_1]\!]s' \quad s'[\![P_2]\!]s''}{s[\![P_1;P_2]\!]s''}\ \mathsf{OHcomp}$$

$$\frac{s \in [\![b]\!] \quad s[\![P_1]\!]s'}{s[\![\mathtt{if}(b)P_1\ \mathtt{else}\ P_2]\!]s'}\ \mathsf{OHif\text{-}t} \qquad \frac{s \notin [\![b]\!] \quad s[\![P_2]\!]s'}{s[\![\mathtt{if}(b)P_1\ \mathtt{else}\ P_2]\!]s'}\ \mathsf{OHif\text{-}f}$$

$$\frac{s \in [\![b]\!] \quad s[\![P]\!]s' \quad s'[\![\mathtt{while}(b)\ \mathtt{do}\ P]\!]s''}{s[\![\mathtt{while}(b)\ \mathtt{do}\ P]\!]s''}\ \mathsf{OHwhile\text{-}t} \qquad \frac{s \notin [\![b]\!]}{s[\![\mathtt{while}(b)\ \mathtt{do}\ P]\!]s}\ \mathsf{OHwhile\text{-}f}$$

次に，状態の性質を表現する論理式である，表明を定義する．ここでは，表明は真偽式を全称量化 ($\forall$) と存在量化 ($\exists$) によって拡張したものとする．表明の集合を Assn_H とし，表明を表すメタ変数には A, B などを用いる．表明の意味は，真偽式と同様 $[\![A]\!] \subseteq \mathsf{Stat}_H$ として与えられる．量化子に対する意味は

$$s \in [\![\forall x A]\!] \overset{\text{def}}{\iff} \text{任意の整数 } n \text{ について，} s \in [\![A[x := n]]\!],$$
$$s \in [\![\exists x A]\!] \overset{\text{def}}{\iff} \text{ある整数 } n \text{ が存在して，} s \in [\![A[x := n]]\!].$$

で定義される．ここで，$A[x := n]$ は，A 中の変数 x を整数定数 n に構文的に置換して得られる表明である．一般の算術式に対して，$A[x := a]$ も同様に定義される．

表明の間の意味論的な含意関係は，それらの意味を表す状態集合の間の部分集合関係で表すことができる．つまり，表明 A と B について，$[\![A]\!] \subseteq [\![B]\!]$ は「A ならば B」であることを意味する．

以上の要素に対して，ホーア・トリプルを次のように定義する．

定義 1.3 (ホーア・トリプル) 表明 A, B と，プログラム P について，$\{A\}P\{B\}$ を（部分正当性を表す）ホーア・トリプルという．ホーア・トリプル $\{A\}P\{B\}$ が正しいとは，「任意の $s, s' \in \mathsf{Stat}_H$ について，$s \in [\![A]\!]$ かつ $s[\![P]\!]s'$ ならば $s' \in [\![B]\!]$」であることとする．

ホーア論理は，以上で定義されたホーア・トリプルを推論するための証明体系である．意味論的にホーア・トリプルの正しさを検証するためには，事前条件を満たす全ての状態について議論しなければならないが，一般にこのような状態は無限個あるため，これを効率的に確かめるためには何らかの抽象化が必要である．ホーア論理では，これらの表明を表す論理式を状態集合を抽象化したものと見做し，この論理式に対する記号的操作によってホーア・トリプルの正しさを証明という有限の構造によって示すことができる．

定義 1.4 (ホーア論理) 部分正当性のためのホーア論理の公理と推論規則は以下のとおりである．

$$\frac{}{\{A\}\texttt{skip}\{A\}}\ \mathsf{Hskip} \qquad \frac{}{\{A[x := a]\}x := a\{A\}}\ \mathsf{Hasgn}$$

$$\frac{\{A\}P_1\{B\} \quad \{B\}P_2\{C\}}{\{A\}P_1;P_2\{C\}}\ \mathsf{Hcomp} \qquad \frac{\{A \wedge b\}P_1\{B\} \quad \{A \wedge \neg b\}P_2\{B\}}{\{A\}\texttt{if}(b)P_1\ \texttt{else}\ P_2\{B\}}\ \mathsf{Hif}$$

$$\frac{\{A \wedge b\}P\{A\}}{\{A\}\texttt{while}(b)\ \texttt{do}\ P\{A \wedge \neg b\}}\ \mathsf{Hwhile} \qquad \frac{[\![A]\!] \subseteq [\![A']\!] \quad \{A'\}P\{B'\} \quad [\![B']\!] \subseteq [\![B]\!]}{\{A\}P\{B\}}\ \mathsf{Hcsq}$$

ホーア論理の証明において自明でないのは，規則 Hcsq における表明間の含意関係の確認と，規則 Hwhile におけるループ不変式 A の発見である．ホーア論理における証明を自動化するためには，これらの問題を解決する必要がある．

2. 分離論理

分離論理 [12] は，メモリの割当てや参照，解放などのヒープ・メモリを操作するコマンドを含むプログラムを検証するためにホーア論理を拡張したものである．分離論理に関する解説は [11] などがある．

分離論理は，項定数として nil はヌル・ポインタを表を nil をもつとする．考察の対象とする状態集合は，ストアの集合 Stat_H をヒープ・メモリの状態を表すよう拡張したものである．ヒープ・メモリは，アドレスを表す正整数からそのアドレスに格納されている整数値への有限部分関数で表現する．ここで，$\mathrm{dom}(h)$ はヒープ中で値が割当てられているアドレスの集合を表す．ヌル・ポインタは整数値 0 で表現する．すなわち，ストア s は常に $s(\mathsf{nil}) = 0$ によって拡張されると考える．ストアとヒープの組をヒープ・モデルと呼ぶ．また，メモリへのアクセス・エラーを表す特別な状態 abort を用意する．

定義 2.1 (ヒープモデル) Stat_S は，ストア $s \in \mathsf{Stat}_H$ とヒープ h（$\mathbb{Z}_+$ から $\mathbb{Z}$ への有限関数）の組 (s, h)，および，特別な状態 abort からなる集合とする．

ストアの時と同様，記法 $[n_1{\to}m_1, \ldots n_k{\to}m_k]$ および $[\,]$ を利用する．ヒープ h に対して，$h[n{\to}m]$ および $h[n{\to}\uparrow]$ は次で定義されるヒープである．

$$h[n{\to}m](k) = \begin{cases} m & (k = n) \\ h(k) & (k \neq n) \end{cases} \qquad h[n{\to}\uparrow](k) = \begin{cases} \text{未定義} & (k = n) \\ h(k) & (k \neq n) \end{cases}$$

二つのヒープ h_1 と h_2 について，$\mathrm{dom}(h_1) \cap \mathrm{dom}(h_2) = \emptyset$ であるとき $h_1 \perp h_2$ と書く．$h_1 \perp h_2$ であるとき，この二つのヒープの和 $h_1 + h_2$ は，$\mathrm{dom}(h_1 + h_2) = \mathrm{dom}(h_1) \cup \mathrm{dom}(h_2)$ であり，

$$(h_1 + h_2)(n) = \begin{cases} h_1(n) & (n \in \mathrm{dom}(h_1)) \\ h_2(n) & (n \in \mathrm{dom}(h_2)) \end{cases}$$

であるようなヒープのこととする．

分離論理が対象とするプログラムの集合 Prog_S は Prog_H のプログラムをヒープ操作のコマンドによって拡張したプログラムの集合とする．ただし，算術式および真偽式の中でヒープ・メモリのアクセスはないものとする．すなわち，これらの定義および意味はホーア論理と同様である．

定義 2.2 (プログラム) Prog_S は Prog_H を以下で拡張したプログラムの集合とする．

$$P ::= \cdots \mid x := \mathtt{cons}(\vec{a}) \mid x := [a] \mid [a] := a' \mid \mathtt{free}(a)$$

$x := \mathtt{cons}(\vec{a})$ は，ヒープ・メモリに（非決定的に）ベクトル $\vec{a}$ の長さ分の連続領域を確保して $\vec{a}$ の値を格納し，x に先頭のアドレスを代入するコマンドである．$x := [a]$ はヒープ・メモリ中のアドレス a に格納されている値を x に代入し，$[a] := a'$ はアドレス a の値を a' の値に変更するコマンドである．$\mathtt{free}(a)$ はアドレス a のブロックを解放するコマンドである．これらの意味は以下のように形式的に定義される．

定義 2.3 (プログラムの操作的意味論) プログラム $P \in \mathsf{Prog}_S$ に対して，Stat_S 上の二項関係 $[\![P]\!]$ を次の公理と規則によって帰納的に定義する．

$$\frac{}{\mathsf{abort}[\![P]\!]\mathsf{abort}}\ \mathsf{OSabort} \qquad \frac{}{(s,h)[\![\mathtt{skip}]\!](s,h)}\ \mathsf{OSskip}$$

$$\frac{}{(s,h)[\![x := a]\!](s[x{\to}[\![a]\!](s)],h)}\ \mathsf{OSasgn}$$

$$\frac{(s,h)[\![P_1]\!](s',h') \quad (s',h')[\![P_2]\!](s'',h'')}{(s,h)[\![P_1;P_2]\!](s'',h'')}\ \mathsf{OScomp}$$

$$\frac{s \in [\![b]\!] \quad (s,h)[\![P_1]\!](s',h')}{(s,h)[\![\mathtt{if}(b)P_1\ \mathtt{else}\ P_2]\!](s',h)}\ \mathsf{OSif\text{-}t} \qquad \frac{s \notin [\![b]\!] \quad (s,h)[\![P_2]\!](s',h')}{(s,h)[\![\mathtt{if}(b)P_1\ \mathtt{else}\ P_2]\!](s',h')}\ \mathsf{OSif\text{-}f}$$

$$\frac{s \in [\![b]\!] \quad (s,h)[\![P]\!](s',h') \quad (s',h')[\![\mathtt{while}(b)\ \mathtt{do}\ P]\!](s'',h'')}{(s,h)[\![\mathtt{while}(b)\ \mathtt{do}\ P]\!](s'',h'')}\ \mathsf{OSwhile\text{-}t}$$

$$\frac{s \notin [\![b]\!]}{(s,h)[\![\mathtt{while}(b)\ \mathtt{do}\ P]\!](s,h)}\ \mathsf{OSwhile\text{-}f}$$

$$\frac{\vec{a} = (a_1,\ldots,a_k) \quad n \in \mathbb{Z}_+ \quad \{n,\ldots,n+k-1\} \cap \mathrm{dom}(h) = \emptyset}{(s,h)[\![x := \mathtt{cons}(\vec{a})]\!](s[x{\to}n], h[n{\to}[\![a_1]\!](s),\ldots n+k-1{\to}[\![a_k]\!](s)])}\ \mathsf{OScons}$$

$$\frac{[\![a]\!](s) \in \mathrm{dom}(h)}{(s,h)[\![x := [a]]\!](s[x{\to}h([\![a]\!](s))],h)}\ \mathsf{OSlook} \qquad \frac{[\![a]\!](s) \notin \mathrm{dom}(h)}{(s,h)[\![x := [a]]\!]\mathsf{abort}}\ \mathsf{OSlook\text{-}f}$$

$$\frac{[\![a]\!](s) \in \mathrm{dom}(h)}{(s,h)[\![[a] := a']\!](s,h[[\![a]\!](s){\to}[\![a']\!](s)])}\ \mathsf{OSmutat} \qquad \frac{[\![a]\!](s) \notin \mathrm{dom}(h)}{(s,h)[\![[a] := a']\!]\mathsf{abort}}\ \mathsf{OSmutat\text{-}f}$$

$$\frac{[\![a]\!](s) \in \mathrm{dom}(h)}{(s,h)[\![\mathtt{free}(a)]\!](s,h[[\![a]\!](s){\to}\uparrow])}\ \mathsf{OSfree} \qquad \frac{[\![a]\!](s) \notin \mathrm{dom}(h)}{(s,h)[\![\mathtt{free}(a)]\!]\mathsf{abort}}\ \mathsf{OSfree\text{-}f}$$

分離論理の表明集合 Assn_S は，ホーア論理の表明言語 Assn_H を分離連言 $A * B$，分離含意 $A \mathrel{-\!\!*} B$，空ヒープ emp，および，単位ヒープ $a \mapsto a'$ で拡張したものである．表明の意味は以下で定義される．

定義 2.4 (分離論理の表明) 表明 A の意味 $[\![A]\!] \subseteq \mathsf{Stat}_S$ は以下で定義される．

$$
\begin{aligned}
[\![A_1 * A_2]\!] &= \{(s,h) \mid \exists h_1 h_2 (h = h_1 + h_2 \text{かつ} (s,h_1) \in [\![A_1]\!] \text{かつ} (s,h_2) \in [\![A_2]\!])\} \\
[\![A_1 \mathrel{-\!\!*} A_2]\!] &= \{(s,h) \mid \forall h_1 ((h \perp h_1 \text{かつ} (s,h_1) \in [\![A_1]\!]) \text{ならば} (s, h+h_1) \in [\![A_2]\!])\} \\
[\![\mathsf{emp}]\!] &= \{(s,h) \mid \mathrm{dom}(h) = \emptyset\} \\
[\![a \mapsto a']\!] &= \{(s,h) \mid \mathrm{dom}(h) = \{s(a)\} \text{かつ} h(s(a)) = s(a')\}
\end{aligned}
$$

その他の表明についてはホーア論理の時と同様定義されるが，状態 abort については，任意の表明 A に対して $\mathsf{abort} \notin [\![A]\!]$ とする．とくに，$\mathsf{abort} \notin [\![\mathtt{true}]\!]$ である．

単位ヒープ $a \mapsto a'$ は，ヒープ中のちょうど一つのアドレスのみが割当てられていることを意味する表明である．例えば，$([x{\to}10], [10{\to}20]) \in [\![x \mapsto 20]\!]$ であるが，$([x{\to}10], [10{\to}20, 11{\to}30]) \notin [\![x \mapsto 20]\!]$ である．分離連言 $A_1 * A_2$ は，ヒープが A_1 と A_2 を満たす二つの互いに素な部分に分割できることを意味する表明である．例えば，$([x{\to}10, y{\to}11], [10{\to}20, 11{\to}20]) \in [\![x \mapsto 20 * y \mapsto 20]\!]$ である．通常の連言 $x \mapsto 20 \wedge y \mapsto 20$ は，ヒープが $x \mapsto 20$ と $y \mapsto 20$ を同時に満たすことを意味するので，ヒープ中のちょうど一つのアドレスのみが割当てられており，x と y ともにそのアドレス値をもっていることを意味する．分離含意 $A_1 \mathrel{-\!\!*} A_2$ を満たすヒープは，それと共通部分を持たない A_1 を満たすヒープと結合すると A_2 を満たすヒープになる．従って，$[\![A_1 * (A_1 \mathrel{-\!\!*} A_2)]\!] \subseteq [\![A_2]\!]$ と $[\![A_2]\!] \subseteq [\![A_1 \mathrel{-\!\!*} (A_1 * A_2)]\!]$ が常に成立する．

以下では略記として，

$$
\begin{gathered}
a \mapsto (a_1, a_2, \ldots, a_n) = a \mapsto a_1 * a+1 \mapsto a_2 * \cdots * a+(n-1) \mapsto a_n \\
a \mapsto - = \exists y (a \mapsto y) \qquad (y \text{は} a \text{に現れない})
\end{gathered}
$$

を用いる．また，任意の表明 A について $[\![A * \mathsf{emp}]\!] = [\![\mathsf{emp} * A]\!] = [\![A]\!]$ であることは自明なので，$A * \mathsf{emp}$ および $\mathsf{emp} * A$ と A は同一視することにする．

以上の要素に対して，ホーア・トリプルを次のように定義する．

定義 2.5 (分離論理のホーア・トリプル) 表明 A, B と，プログラム P について，$\{A\}P\{B\}$ を（部分正当性を表す）ホーア・トリプルという．ホーア・トリプル $\{A\}P\{B\}$ が正しいとは，「任意の $S, S' \in \mathsf{Stat}_S$ について，$S \in [\![A]\!]$ かつ $S[\![P]\!]S'$ ならば $S' \in [\![B]\!]$」であることとする．

とくに，任意の表明が abort に対して偽であることにより，部分正当性 $\{A\}P\{B\}$ が成立しているとき，$(s,h) \in [\![A]\!]$ ならば $(s,h)[\![P]\!]\mathsf{abort}$ とはならない，つまりメモリへのアクセス・エラーが発生しないことが分かる．

分離論理はホーア論理の拡張として，追加したヒープ・メモリ操作のためのコマンドに対する公理を追加することによって得られる．ここでは，[13] で与えられている公理を紹介する．

定義 2.6 (分離論理) 部分正当性のための分離論理の公理と推論規則は，ホーア論理のものに以下の公理を加えたものである．

$$\frac{y\text{は}\vec{a}, A\text{に現れない}}{\{\forall y((y \mapsto \vec{a}) \mathrel{-\!\!*} A[x := y])\}x := \mathtt{cons}(\vec{a})\{A\}}\ \mathsf{Scons}$$

$$\frac{y\text{は}a, A\text{に現れない}}{\{\exists y(a \mapsto y * ((a \mapsto y) \mathrel{-\!\!*} A[x := y]))\}x := [a]\{A\}}\ \mathsf{Slook}$$

$$\frac{y\text{は}a\text{に現れない}}{\{a \mapsto - * ((a \mapsto a') \mathrel{-\!\!*} A)\}[a] := a'\{A\}}\ \mathsf{Smutat} \qquad \frac{}{\{a \mapsto - * A\}\mathtt{free}(a)\{A\}}\ \mathsf{Sfree}$$

公理 Scons では，cons コマンドで割り当てられるアドレスの任意性が全称量化によって表現されている．規則 Slook，Smutat，Sfree における結論の事前条件の形により，これらのコマンド実行前にアドレス a のメモリが確保されていることが要求されていることが分かる．

3. 帰納的述語による拡張

表明集合 Assn_S では，ヒープ構造に関する再帰的な性質が表現できないため，リストや木などの再帰的構造をもつデータを操作するプログラムの性質をうまく表現できない．そこで，Assn_S をヒープ構造の性質に関する帰納的述語によって拡張する．帰納的述語 $P(\vec{x})$ の意味 $[\![P(\vec{x})]\!] \subseteq \mathsf{Assn}_S$ は最小不動点を用いて定義される．

まず，分離論理における証明例として，連結リストを反転するプログラムの性質の証明を与える．

例 3.1 (リストの反転) 以下では，連結リストの構造を表す述語を考える．リストの各ノードには整数値が格納されているとする．整数リストは自然数でエンコードすることができ，それらに関する操作の多くは Assn_S において表現可能である．例えば，

$$\begin{aligned}
\mathsf{Nil}(l) &= 「l\text{は空の整数リスト}」\\
\mathsf{Cons}(l, x, l') &= 「l\text{は，}l'\text{が表現する整数リストの先頭に整数}x\text{を追加したリスト}」\\
\mathsf{Rev}(l, l') &= 「l\text{は，}l'\text{が表現する整数リストを反転したリスト}」\\
\mathsf{App}(l, l', l'') &= 「l\text{は，}l', l''\text{が表現する整数リストを結合したリスト}」
\end{aligned}$$

などの意味をもつ整数上の述語は Assn_S で表現することができる．これらの述語によって，リストに関する要素の追加 $v : l$，反転 l^{-1}，結合 $l \cdot l'$ といった関数を表明において用いることにする．

以上の準備のもとで，帰納的に定義される 2 引数述語 list を考える．

$$\mathsf{list}(x, l) = (x = 0 \wedge \mathsf{Nil}(l)) \vee \exists v l' z(l = v : l' \wedge x \mapsto (v, z) * \mathsf{list}(z, l'))$$

$\mathsf{list}(x, l)$ を満たすヒープは，アドレス x から始まる連結リストである．各ノードは 2 つのフィールド (v, z) からなり，1 つ目のフィールド v にはそのノードの整数値，2 つ目のフィールド z には次のノードのアドレスがそれぞれ格納されている．最後のノードの 2

つ目のフィールドにはヌル・ポインタを表現する0が格納されている．lはこのリストが表す整数リストを表現する自然数である．形式的な$(s,h) \in [\![\mathsf{list}(x,l)]\!]$の意味は，通常の最小不動点を用いた方法で定義される．

$\mathsf{list}(x,l)$を満たすリストを反転する以下のプログラムRを考える．

$$\texttt{while}(\neg(x=0))\ \texttt{do}\ (z := [x+1]; [x+1] := y; y := x; x := z)$$

このRについて，分離論理においてホーア・トリプル$\{y=0 \wedge \mathsf{list}(x,l)\}R\{\mathsf{list}(y,l^{-1})\}$が証明可能であることを確かめよう．ループ不変式としては，

$$A = \exists l_x l_y (l = l_y^{-1} \cdot l_x \wedge (\mathsf{list}(x,l_x) * \mathsf{list}(y,l_y)))$$

を取ることができる．

まず，Aにおいてl_xをl，l_yを空リストとすれば，$\mathsf{list}(0,l_y)$が成り立つので，$[\![y=0 \wedge \mathsf{list}(x,l)]\!] \subseteq [\![A]\!]$が分かる．また，$x=0$かつ$A$を仮定すると，$A$中の$l_x$は空リストとなるので$l = l_y^{-1}$，つまり，ヒープ中で$y$から始まるリストは$l$を反転したリストを表すことが分かるので，$[\![x=0 \wedge A]\!] \subseteq [\![\mathsf{list}(y,l^{-1}))]\!]$が成り立つ．したがって，以下のように証明を構成できる．

$$\dfrac{\dfrac{\vdots\ \Pi}{\dfrac{\{\neg(x=0)\wedge A\}R'\{A\}}{\{A\}R\{x=0\wedge A\}}\ \mathsf{Hwhile}}}{\{y=0\wedge \mathsf{list}(x,l)\}R\{\mathsf{list}(y,l^{-1})\}}\ \mathsf{Hcsq}$$

ここで，R'はループの本体$z := [x+1]; [x+1] := y; y := x; x := z$である．

証明Πを与えよう．まず，$\neg(x=0)$かつ$\mathsf{list}(x,l_x)$のとき，xは非空なリストであることが分かるから，

$$[\![\neg(x=0)\wedge A]\!] \subseteq [\![\exists l'_x l_y v x' (l = l_y^{-1} \cdot (v : l'_x) \wedge x \mapsto (v,x') * \mathsf{list}(x',l'_x) * \mathsf{list}(y,l_y))]\!]$$

が成り立つ．この右辺の論理式を$\exists x' B(x')$と書くことにする．$B(x')$は$x+1 \mapsto x'$を含む（略記$x \mapsto (v,x')$に含まれる）ので，分離含意の意味より，

$$[\![B(x')]\!] \subseteq [\![x+1 \mapsto x' * (x+1 \mapsto x' -\!\!* B(x'))]\!]$$

が成立する．従って，

$$[\![\exists x' B(x')]\!] \subseteq [\![\exists x' (x+1 \mapsto x' * (x+1 \mapsto x' -\!\!* B(x')))]\!]$$

である．公理Slookによって，

$$\{\exists x' (x+1 \mapsto x' * (x+1 \mapsto x' -\!\!* B(x')))\} z := [x+1] \{B(z)\}$$

であるから，規則Hcsqより，

$$\{\neg(x=0)\wedge A\} z := [x+1] \{B(z)\}$$

が証明できる．同様に，規則Hcsqと公理Smutatによって，

$$\{B(z)\}[x+1] := y \{\exists l'_x l_y v (l = l_y^{-1} \cdot (v : l'_x) \wedge x \mapsto (v,y) * \mathsf{list}(z,l'_x) * \mathsf{list}(y,l_y))\}$$

が証明できる．この事後条件を C と置くと，リストの性質として $l_y^{-1} \cdot (v : l'_x) = (v : l_y)^{-1} \cdot l'_x$ であることと，list の定義によって，

$$\begin{aligned}[\![C]\!] &\subseteq [\![\exists l'_x l_y v (l = (v : l_y)^{-1} \cdot l'_x \wedge x \mapsto (v, y) * \mathsf{list}(z, l'_x) * \mathsf{list}(y, l_y))]\!] \\ &\subseteq [\![\exists l'_x l''_x (l = l''^{-1}_x \cdot l'_x \wedge \mathsf{list}(z, l'_x) * \mathsf{list}(x, l''_x))]\!] = [\![A[x := z][y := x]]\!]\end{aligned}$$

が成り立つ．公理 Hasgn を 2 回適用することにより，$\{A[x := z][y := x]\}y := x; x := z\{A\}$ が証明できるので，以上より $\{\neg(x = 0) \wedge A\}R'\{A\}$ が分離論理において証明できることが分かる．

4. エンテイルメント判定問題のための証明体系

ホーア論理のときと同様，分離論理の証明（の自動化）において重要なポイントは，規則 Hsq における含意関係 $[\![A]\!] \subseteq [\![B]\!]$ の判定と，規則 Hwhile におけるループ不変式の発見である．ここではとくに前者の問題（エンテイルメント判定問題）について考えることにする．算術式を含む一般の分離論理において，エンテイルメント判定問題は明らかに決定不可能であるため，以下のように制限された論理式に対する判定問題を考える．とくに，算術演算などは考慮せず，項は変数または定数 nil のみとし，t, t' などで項を表す．

定義 4.1 (記号ヒープと帰納的述語) 以下でストア論理式とヒープ論理式を定義する．

$$\begin{aligned}\Pi &::= (t = t') \mid (t \neq t') \mid \Pi \wedge \Pi' && \text{（ストア論理式）} \\ \Sigma &::= \mathsf{emp} \mid x \mapsto (\vec{t}) \mid P(\vec{t}) \mid \Sigma * \Sigma' && \text{（ヒープ論理式）}\end{aligned}$$

ここで，P は述語記号である．$\exists \vec{x}(\Pi \wedge \Sigma)$ の形の論理式を記号ヒープと呼ぶ．各述語記号 P に対しては，$P(\vec{x}) = \bigvee_j A_j(\vec{x})$（各 A_j は記号ヒープ）の形の帰納的定義が与えられているとする．各 A_j は P の定義節と呼ばれる．

記号ヒープ A, B に対して，判断式 $A \vdash B$ をエンテイルメントと呼ぶ．このエンテイルメントが正しい，とは，含意関係 $[\![A]\!] \subseteq [\![B]\!]$ が成り立つこととする．

ここで与えた記号ヒープは，算術式も含まない非常に制限されたものであるが，一般の帰納的述語を許した場合，エンテイルメントの正しさを判定する問題は決定不可能であることが知られている [1]．このため，エンテイルメント判定問題が決定可能になるような帰納的述語への制限に関して様々な提案がされている．最も単純なものは，特定の述語に限定する方法である．例えば，リスト・セメグント

$$\mathsf{ls}(x, y) ::= (x \neq y \wedge x \mapsto y) \ \vee \ \exists z(x \neq y \wedge x \mapsto z * \mathsf{ls}(z, y))$$

のみを考える場合，単純な証明体系で完全かつ決定可能なものが知られている [3]．同様に，特定の述語のみを考える場合は，完全かつ決定可能な証明体系における証明探索によるエンテイルメント判定アルゴリズムが与えらている [4, 10]．一方，より広い述語のクラスに対しては，有界木幅グラフの単項二階論理に帰着する方法で決定アルゴリズムが与えられている [9]．このクラスが，現在知られている中では，エンテイルメント判定が決定可能である最も広いクラスである．

このように，記号ヒープ間のエンテイルメント判定問題に対しては様々な手法が提案されているが，一般の述語に対する証明論的な手法，とくに，完全かつ決定可能な

証明体系は知られていない．証明体系 CSLID_ω[14] は，一般の述語クラス（[9] の部分クラス）のエンテイルメントに対する完全かつ決定可能な証明体系である．ここでは，この証明体系の概要を解説する．

CSLID_ω が対象とする帰納的述語 $P(x,\vec{y})$ の各定義節は

$$\exists u_1 \cdots u_n(\Pi \wedge x \mapsto (\vec{v}) * P_1(u_1,\vec{w}_1) * \cdots P_n(u_n,\vec{w}_n))$$

の形で，条件 $\{u_1,\ldots,u_n\} \subseteq \vec{v}$ を満たすものとする．このとき，$(s,h) \models P(x,\vec{y})$ を満たすヒープ (s,h) は，$s(x)$ を根とする木に葉から他のノードへの逆辺を追加した構造をもつ．このため，引数 x を $P(x,\vec{y})$ の根と呼ぶ．また，このヒープ中の各ノードはある（再帰的に現れる）述語が表現する部分ヒープの根になっている．この構造のため，以下のような証明探索が有効である．エンテイルメント

$$\mathsf{ls}(x,y) * \mathsf{ls}(y,\mathsf{nil}) \vdash \mathsf{ls}(x,\mathsf{nil})$$

を例にとって，探索アルゴリズムを簡単に説明する．

1. 両辺に共通の根をもつ述語を探す．この例においては，左辺の $\mathsf{ls}(x,y)$ と右辺の $\mathsf{ls}(x,\mathsf{nil})$ が共通の根 x をもつ．

2. 共通の根をもつ述語を定義節に従って展開する．左辺の展開は定義節の選択に関する場合分けに相当する．このとき，存在量化された変数は，fresh な変数で具体化する．

$$\frac{x \neq y \wedge x \mapsto y * \mathsf{ls}(y,\mathsf{nil}) \vdash \mathsf{ls}(x,\mathsf{nil}) \quad x \neq y \wedge x \mapsto z * \mathsf{ls}(z,y) * \mathsf{ls}(y,\mathsf{nil}) \vdash \mathsf{ls}(x,\mathsf{nil})}{\mathsf{ls}(x,y) * \mathsf{ls}(y,\mathsf{nil}) \vdash \mathsf{ls}(x,\mathsf{nil})}\ (\mathrm{PL})$$

さらに，左辺の形に合わせて右辺を展開する．例の二つの上式のうち右のものについて示すと，

$$\frac{x \neq y \wedge x \mapsto z * \mathsf{ls}(z,y) * \mathsf{ls}(y,\mathsf{nil}) \vdash x \neq \mathsf{nil} \wedge x \mapsto z * \mathsf{ls}(z,\mathsf{nil})}{x \neq y \wedge x \mapsto z * \mathsf{ls}(z,y) * \mathsf{ls}(y,\mathsf{nil}) \vdash \mathsf{ls}(x,\mathsf{nil})}\ (\mathrm{PR})$$

3. 両辺に共通する単位ヒープを削除する．このとき，左辺には x がヒープ中に存在しないことを表す補助的な述語 $x\uparrow$ を追加する．

$$\frac{x\uparrow \wedge x \neq y \wedge \mathsf{ls}(z,y) * \mathsf{ls}(y,\mathsf{nil}) \vdash x \neq \mathsf{nil} \wedge \mathsf{ls}(z,\mathsf{nil})}{x \neq y \wedge x \mapsto z * \mathsf{ls}(z,y) * \mathsf{ls}(y,\mathsf{nil}) \vdash x \neq \mathsf{nil} \wedge x \mapsto z * \mathsf{ls}(z,\mathsf{nil})}\ (* \mapsto)$$

4. 正規化する．ヒープ論理式中に現れない変数に関する $x\uparrow$ や等号などを削除する．

$$\frac{\mathsf{ls}(z,y) * \mathsf{ls}(y,\mathsf{nil}) \vdash \mathsf{ls}(z,\mathsf{nil})}{x\uparrow \wedge x \neq y \wedge \mathsf{ls}(z,y) * \mathsf{ls}(y,\mathsf{nil}) \vdash x \neq \mathsf{nil} \wedge \mathsf{ls}(z,\mathsf{nil})}$$

ここで，得られたエンテイルメント $\mathsf{ls}(z,y) * \mathsf{ls}(y,\mathsf{nil}) \vdash \mathsf{ls}(z,\mathsf{nil})$ は，入力のエンテイルメントを名前替えしただけのものになっており，証明木中の二つのエンテイルメント間では左辺の述語が一度展開されている．従って，直観的には $\mathsf{ls}(z,y) * \mathsf{ls}(y,\mathsf{nil}) \vdash \mathsf{ls}(z,\mathsf{nil})$ を帰納法の仮定とすれば，帰納法によって証明は完了していると考えることができる．

このアイディアを形式化したものが，循環証明である [5, 6, 7]．循環証明体系では，証明木において公理ではない葉に対して，同じエンテイルメントをもつノードへのリンクを許し，大域トレース条件と呼ばれる条件を満たすものを証明と認める．この条件によって，証明体系の健全性が保証される．上記の手順に従えば，大域トレース条件を満たす循環証明が得られる．

CSLID_ω に対して，[14] では上記のアイディアをもとにした証明探索手続きが与えられ，その完全性と決定可能性が証明されている．証明の上で解決すべき大きな問題としては以下が挙げられる．

- 選言による拡張：例として以下の述語を考える．

$$\mathsf{lso}(x,y) = (x \neq y \wedge x \mapsto y) \ \vee \ \exists z(x \neq y \wedge x \mapsto z * \mathsf{lse}(z,y))$$
$$\mathsf{lse}(x,y) = \exists z(x \neq y \wedge x \mapsto z * \mathsf{lso}(z,y))$$
$$\mathsf{lsx}(x,y) = \exists z(x \neq y \wedge x \mapsto z * \mathsf{lso}(z,y)) \ \vee \ \exists z(x \neq y \wedge x \mapsto z * \mathsf{lse}(z,y))$$

  すなわち，lso は奇数長のリスト，lse は偶数長のリスト，lsx はそのいずれか（ただし長さは2以上）を表す．このとき，エンテイルメント $x \mapsto y{*}\mathsf{ls}(y,\mathsf{nil}) \vdash \mathsf{lsx}(x,\mathsf{nil})$ は正しいが，上記の証明探索において右辺の lsx を展開する段階で二つの定義節のいずれを選ぶべきか決定できない．そのため，完全な証明体系のためには右辺を選言によって拡張する必要がある．一般に，$A \vdash B_1, \ldots, B_n$ の形を考え，その意味を $[\![A]\!] \subseteq \bigcup_i [\![B_i]\!]$ で与える．

- 分離連言規則の一般化：通常の分離連言規則は

$$\frac{A_1 \vdash A_2 \quad B_1 \vdash B_2}{A_1 * A_2 \vdash B_1 * B_2}$$

  という形をしているが，右辺を選言で拡張したことにより以下のような一般化が必要である．

$$\frac{A_1 \vdash (B_1^i)_{i\in I_1} \quad \text{or} \quad A_2 \vdash (B_2^i)_{i\in I_2} \quad (\text{for any } I_1 \uplus I_2 = I)}{A_1 * A_2 \vdash (B_1^i * B_2^i)_{i\in I}}$$

- 帰納的分離含意の導入：証明探索手続の最初のステップにおいて，両辺に共通の根をもつ述語が存在しないことがある．例えば，双方向リストの二つの定義を考える．

$$\mathsf{dll1}(h,t,p,n) = (h = t \wedge h \mapsto (p,n)) \ \vee \ \exists z(h \mapsto (p,z) * \mathsf{dll1}(z,t,h,n))$$
$$\mathsf{dll2}(h,t,p,n) = (h = t \wedge h \mapsto (n,p)) \ \vee \ \exists z(h \mapsto (z,p) * \mathsf{dll1}(z,t,h,n))$$

  このとき，$\mathsf{dll1}(h,t,p,n) \vdash \mathsf{dll2}(t,h,n,p)$ は正しいエンテイルメントであるが，両辺に共通の根は存在しない．$\mathsf{dll2}(t,h,n,p)$ が表すヒープの部分ヒープとして h を根としてもつ部分を切り出せば，共通の根 h を見つけることができる．dll2 の定義は dll2 にしか依存していないので，この h を根とする部分ヒープも dll2 を満たすはずであるが，残りの部分も帰納的述語によって表現することができる．$\mathsf{dll2}(t,h,n,p)$ を満たすヒープから $\mathsf{dll2}(h',t',p',n')$ を満たすヒープを取り除いた部分は

$$\mathsf{dll2}(h',t',p',n') {-\!\!*}^i \mathsf{dll2}(t,h,n,p) = \quad t' = t \wedge p' = h \wedge n' = n \wedge h \mapsto (h',p)$$
$$\vee \ \exists z(h \mapsto (z,p) * (\mathsf{dll2}(h',t',p',n') {-\!\!*}^i \mathsf{dll2}(z,t,h,n)))$$

で定義される帰納的述語 $\mathsf{dll2}(h', t', p', n') \mathrel{-\!\!*^i} \mathsf{dll2}(t, h, n, p)$ で表される．これを帰納的分離含意と呼ぶ．ここで，この述語の根は t である．帰納的分離含意を利用すれば，ヒープ論理式分解規則

$$\frac{\mathsf{dll1}(h, t, p, n) \vdash \exists t' p' n' ((\mathsf{dll2}(h, t', p', n') \mathrel{-\!\!*^i} \mathsf{dll2}(t, h, n, p)) * \mathsf{dll2}(h, t', p', n'))}{\mathsf{dll1}(h, t, p, n) \vdash \mathsf{dll2}(t, h, n, p)} \text{ (Fact)}$$

を適用することができ，両辺共通の根 h を見出すことができる．これらの一般化・追加した規則により，正しいエンテイルメントに対しては必ず証明探索を進めることができる．

- 証明探索手続の停止性：さらに，証明中に現れ得るエンテイルメントが有限であることが分かれば，いずれは循環証明が完成するか，または，エンテイルメントが正しくないことが分かる．このために，ある種の正規形を定め，可能な正規形が有限であることを示す．ここで重要な事実は，与えられたエンテイルメントに対し，その証明に必要な帰納的分離含意の入れ子の深さの上限が定められることである．

参考文献

[1] T. Antonopoulos, N. Gorogiannis, C. Haase, M. Kanovich, and J. Ouaknine, Foundations for Decision Problems in Separation Logic with General Inductive Predicates, In: Proceedings of FoSSaCS 2014, *LNCS* 8412 (2014) 411–425.

[2] K.R. Apt (1981) Ten Years of Hoare's Logic: A Survey—Part I. *ACM Transactions on Programming Languages and Systems*, Vol. 3, No. 4, 431-483.

[3] J. Berdine, C. Calcagno, P. W. O'Hearn, A Decidable Fragment of Separation Logic, In: Proceedings of FSTTCS 2004, *LNCS* 3328 (2004) 97–109.

[4] J. Berdine, C. Calcagno, and P. W. O'Hearn, Symbolic Execution with Separation Logic, In: Proceedings of APLAS 2005, *LNCS* 3780 (2005) 52–68.

[5] J. Brotherston, A Simpson, Sequent calculi for induction and infinite descent, *Journal of Logic and Computation 21* (6) (2011) 1177–1216.

[6] J. Brotherston, D. Distefano, and R. L. Petersen, Automated cyclic entailment proofs in separation logic, In: *Proceedings of CADE-23* (2011) 131–146.

[7] J. Brotherston, N. Gorogiannis, and R. L. Petersen, A Generic Cyclic Theorem Prover, In: Proceedings of APLAS 2012, *LNCS* 7705 (2012) 350–367.

[8] C.A.R. Hoare (1969) An Axiomatic Basis for Computer Programming. *Communications of the ACM*, Vol. 12, No. 10, 576–583.

[9] R. Iosif, A. Rogalewicz, and J. Simacek, The Tree Width of Separation Logic with Recursive Definitions, In: Proceedings of CADE-24, *LNCS* 7898 (2013) 21–38.

[10] D. Kimura and M. Tatsuta (2018) Decidability for Entailments of Symbolic Heaps with Arrays. Available at `https://arxiv.org/abs/1802.05935`.

[11] P.W. O'Hearn (2012) A Primer on Separation Logic (and Automatic Program Verification and Analysis). *Software Safety and Security; Tools for Analysis and Verification, Volume 33 of NATO Science for Peace and Security Series*, 286–318.

[12] J.C. Reynolds (2002) Separation Logic: A Logic for Shared Mutable Data Structures. *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LiCS 2002)*, 55–74.

[13] M.Tatsuta, W.N. Chin, and M.F. Al Ameen (2009) Completeness of Pointer Program Verification by Separation Logic. *Proceedings of 7th IEEE International Conference on Software Engineering and Formal Methods (SEFM 2009)*, 179–188.

[14] M. Tatsuta, K. Nakazawa, and D. Kimura (2018) Completeness of Cyclic Proofs for Symbolic Heaps. Available at `https://arxiv.org/abs/1804.03938`.

[15] G. Winskel (2001) *The Formal Semantics of Programming Languages (Fifth ed.)*, MIT Press.

[16] 林 晋 (1995) プログラム検証論（情報数学講座 8），共立出版.