Completeness of Cyclic Proofs for Symbolic Heaps

Makoto Tatsuta
National Institute of Informatics / Sokendai, Japan
tatsuta@nii.ac.jp

Koji Nakazawa
Nagoya University, Japan
knak@i.nagoya-u.ac.jp

Daisuke Kimura
Toho University, Japan
kmr@is.sci.toho-u.ac.jp

Abstract

Separation logic is successful for software verification in both theory and practice. Decision procedure for symbolic heaps is one of the key issues. This paper proposes a cyclic proof system for symbolic heaps with general form of inductive definitions, and shows its soundness and completeness. The decision procedure for entailments of symbolic heaps with inductive definitions is also given. Decidability for entailments of symbolic heaps with inductive definitions is an important question. Completeness of cyclic proof systems is also an important question. The results of this paper answer both questions. The decision procedure is feasible since it is nondeterministic double-exponential time complexity.

1 Introduction

Separation logic is successful for software verification [22, 4, 5]. Several systems based on this idea have been actively investigated and implemented. One of the keys in these systems is the entailment checker that decides the validity for a given entailment of symbolic heaps.

The paper [17] proposed the system SLRD_{btw}, which is the first decidable system for entailment of symbolic heaps with general form of inductive definitions. We call the conditions imposed in [17] for restricting the class of inductive definitions by a bounded treewidth condition. The inductive definitions without any restriction cause undecidability [1]. The bounded treewidth condition is one of the most flexible conditions for a decidable system.

A cyclic proof system [7] can give us efficient implementation of theorem provers. On the other hand, the completeness and the decidability of provability are not known for any cyclic proof system. Hence it is a challenging problem to find some complete cyclic proof system and some decidable system for symbolic heaps with general form of inductive definitions.

Our contribution is to solve these problems, namely, to propose a cyclic proof system CSLID_ω for symbolic heaps with inductive definitions, to prove its soundness theorem and its completeness theorem, and to give its decision procedure.

Our first ideas are as follows: (1) We define inductive definition clauses so that the unfolding determines the root cells of children. (2) We use unfold-match and split, namely, first unfold inductive predicates on both sides (antecedent and succedent), next remove the same $x \mapsto (\_)$ on both sides, then split the entailment by separating conjunction. (3) We do proof-search by going up from the conclusion to the assumption in an inference rule. We will show the termination by defining normal forms and showing the set of those possibly used during computation is finite. This shows the termination since (a) every path of potentially infinite length contains infinite number of normal forms, (b) the set of normal forms is finite, (c) hence there is some repetition of normal forms in the path, (d) hence in the path some subgoal of the repetition is eventually discharged by cyclic proof mechanism. (4) We also show (selective) local completeness of each application of rules used in each step. By this and termination, we will show the completeness.

Our main ideas are as follows. Each of them is new. All of these techniques together make an algorithm based on those first ideas a real algorithm. (1) We will introduce atomic formulas $t \downarrow$ and $t \uparrow$
which mean $t$ is in the heap and $t$ is not in the heap respectively, and put $\downarrow$ or $\uparrow$ for each variable in both antecedent and succedent. (2) When we unfold the succedent, to keep validity, we need disjunction in the succedent. So we will introduce disjunction in the succedent. (3) We will propose a new $(\ast)$-split rule for disjunction. As far as we know, no $(\ast)$-split rule has not been proposed for disjunction. (4) We will introduce a factor rule. Roughly speaking, if the candidate of a common root is $x$ but it does not appear in some disjunct $P(y)$, we transform this disjunct into $(Q(x) \rightarrow P(y)) \ast Q(x)$ so that the disjunct has the root $x$. (5) For splitting existential scopes, we transform $\exists w((P(x,w) \wedge w \downarrow) \ast Q(y,w))$ into $\exists w((P(x,w) \wedge w \downarrow) \ast (Q(y,w) \wedge w \uparrow))$ and it into $\exists w((P(x,w) \wedge w \downarrow)) \ast Q(y,w) \wedge w \uparrow)).$ We will show these transformations keep equivalence. (6) We eliminate a disjunct that is a renaming of another disjunct, and moreover we will show that this elimination keeps validity.

For unfold-match and removing $\rightarrow$, we need some conditions (strong connectivity, decisiveness) to the class of inductive definitions besides the bounded treewidth condition. The establishment condition in the bounded treewidth condition is checked by considering the set of inductive definitions, and it is not locally checked by the shape of each definition clause. Our condition is a local version of the bounded treewidth condition. These additional conditions are not so restrictive and our class of inductive definitions is still quite large, since our class contains doubly-linked lists, skip lists, and nested lists.

The decision procedure is feasible since it is double-exponential time complexity.

Several entailment checkers for symbolic heaps with inductive definitions have been discussed. Most of them do not have general form of inductive definitions and have only hard-coded inductive predicates [19, 24, 23, 3] [13, 15, 16]. The entailment checkers for general form of inductive definitions are studied in [12, 13, 17, 18, 24, 8, 9]. The engines of the system SLRD [17, 18] and the system in [24] are both model theoretic, and they are decidable systems. The systems in [13, 8, 9] use cyclic proofs, but neither of them is a complete system. [12] is based on ordinary sequent calculus and is not complete.

The cyclic proofs have been intensively investigated for the first-order predicate logic [7, 9, 2, 23, 4], a bunched implication system [8, 9], and a symbolic heap system [9, 9].

Section 2 defines separation logic with inductive definitions. In Section 3, we extend our language, in particular, we give definition clauses for the strong wand. Section 4 defines the system CSLID$^\omega$. Section 5 shows soundness of the factor rule. Section 6 proves soundness of the existential amalgamation rules. Section 7 proves soundness of the $(\ast)$-split rule. Section 8 proves the soundness of the system. Section 9 gives a satisfiability checker. In Section 10, we define normal forms and groups, give the decision algorithm of validity, and shows its partial correctness, loop invariants and termination. Section 11 proves a property for constant store validity. Section 12 explains cones. Section 13 proves local completeness of the factor rule. Section 14 shows local completeness of the unrelated introduction. Section 15 shows selective local completeness of the rule $(\ast)$-split. Section 16 shows properties for elimination of fresh variables in the succedent. Section 17 proves completeness of CSLID$^\omega$. We conclude in Section 18.

## 2 Symbolic Heaps with Inductive Definitions

This section defines symbolic heaps, inductive definitions, and their semantics.

We will use vector notation $\overline{r}$ to denote a sequence $x_1, \ldots, x_k$ for simplicity. $|\overline{r}|$ denotes the length of the sequence. Sometimes we will also use a notation of a sequence to denote a set for simplicity. We write $\equiv$ for the syntactical equivalence.

### 2.1 Language

Our language is a first order logic with a new connective $\ast$ and inductive predicates, and defined as follows.

- First-order variables $\text{Vars} ::= x, y, z, w, v, \ldots$. Terms $t, u, p, q, r ::= x \mid \text{nil}$.
- Inductive Predicate Symbols $P, Q, R$.

We define formulas $P, G$ of separation logic as those of the first-order language generated by the constant nil, the propositional constant emp, predicate symbols $\equiv, \rightarrow, P, Q, \ldots$, and an additional logical connective $\ast$. We write $t \neq u$ for $\neg t = u$. We assume some number $n_{\text{cell}}$ for the number of elements in a cell.

- Pure formulas $\Pi ::= t = t \mid t \neq t \mid \Pi \wedge \Pi$.
- Spatial formulas $\Sigma ::= \text{emp} \mid t \rightarrow (t_1, \ldots, t_{n_{\text{cell}}}) \mid P(\overline{r}) \mid \Sigma \ast \Sigma$. 

2
We suppose $\ast$ binds more tightly than $\land$. We will sometimes write $P(t)$ for $P(t, \bar{v})$. We write $\ast_{i \in [1..n]} P_i(x_i)$ for $P_1(x_1) \ast \ldots \ast P_n(x_n)$. Similarly we write $\ast_{i \in I} P_i(x_i)$. We write $\Pi \subseteq \Pi'$ when all the conjuncts of $\Pi$ are contained in those of $\Pi'$. 

Eq-Symbolic Heaps $A, B ::= \Pi \land \Sigma \mid \Sigma$. Symbolic Heaps $\phi ::= \exists \bar{v} A$. Entailments $A \vdash B_1, \ldots, B_n$.

Inductive Definitions $P(x, \bar{y}) =_{\text{def}} \bigvee_i \phi_i(x, \bar{y})$ where $\phi_i$ is a definition clause.

Definition Clauses $\phi(x, \bar{y}) \equiv \exists \bar{v} (\Pi \land x \mapsto (\bar{u}) \ast \ast_{i \in I} P_i(z_i, \bar{t}_i))$, where

- $\{ z_i \mid i \in I \} = \bar{v}$ (strong connectivity),
- $\bar{v} \subseteq \bar{u}$ (decisiveness).

We call the first argument of a spatial atomic formula except $\text{emp}$ a root.

The strong connectivity implies the bounded tree width condition. The decisive condition is that in every definition clause, all existential variables must occur in $\bar{u}$ where the definition clause has $x \mapsto (\bar{u})$. It is similar to the constructively valued condition in [11]. This condition guarantees that the cell at address $x$ decides the content of every existential variable.

We give some examples of the inductive definitions in the following. The list segment is definable: $\text{ls}(x, y) =_{\text{def}} x \mapsto y \lor \exists z (x \mapsto (z \ast \text{ls}(z, y)))$.

The doubly-linked list is definable:

\[
\text{dll}(h, p, n, t) =_{\text{def}} h = t \land h \mapsto (p, n) \lor \exists z (h \mapsto (p, z) \ast \text{dll}(z, h, n, t)).
\]

The nested list is definable:

\[
\text{lnest}(x) =_{\text{def}} \exists z. x \mapsto (z, \text{nll}) \ast \text{ls}(z, \text{nll}) \lor \exists z_1 z_2 (x \mapsto (z_1, z_2) \ast \text{ls}(z_1, \text{nll}) \ast \text{lnest}(z_2)).
\]

The following nested list segment is also definable:

\[
\text{lnest}(x, y) =_{\text{def}} \exists z (x \mapsto (z, \text{nll}) \ast \text{ls}(z, y)) \lor \exists z_1 z_2 (x \mapsto (z_1, z_2) \ast \text{ls}(z_1, y) \ast \text{lnest}(z_2, y)).
\]

The skip list is definable:

\[
\begin{align*}
\text{skl1}(x, y) &=_{\text{def}} x \mapsto (\text{nll}, y) \lor \exists z (x \mapsto (\text{nll}, z) \ast \text{skl1}(z, y)), \\
\text{skl2}(x, y) &=_{\text{def}} \exists z (x \mapsto (y, z) \ast \text{skl1}(z, y)) \lor \exists z_1 z_2 (x \mapsto (z_1, z_2) \ast \text{skl1}(z_2, z_1) \ast \text{skl2}(z_1, y)).
\end{align*}
\]

The examples in [8] are definable in our system as follows: List, ListE, ListO are definable, RList is not definable. DLL, PeList, SLL, BSLL, BinTree, BinTreeSeg, BinListFirst, BinListSecond, BinPath are not definable but will be definable in a straightforward extension of our system by handling emp in the base cases.

We prepare some notions. We define $P^{(m)}$ by

\[
P^{(0)}(\bar{v}) = (\text{nll} \neq \text{nll}), \\
P^{(m+1)}(\bar{v}) =_{\text{def}} \bigvee_i \phi_i[P := P^{(m)}],
\]

where $P(\bar{v}) =_{\text{def}} \bigvee_i \phi_i$. $P^{(m)}$ is $m$-time unfold of $P$. We define $F^{(m)}$ as obtained from a formula $F$ by replacing every inductive predicate $P$ by $P^{(m)}$.

We define $(\neq (T_1, T_2))$ as $\bigwedge_{t_1 \in T_1, t_2 \in T_2, t_1 \neq t_2} t_1 \neq t_2$. We write $x \neq T$ for $(\neq (\{x\}, T))$.

We define $(\neq (T))$ as $(\neq (T \cup \{\text{nll}\}, T \cup \{\text{nll}\}))$.

2.2 Semantics

This subsection gives semantics of the language.

We define the following structure: $\text{Val} = N$, $\text{Locs} = \{ x \in N \mid x > 0 \}$, $\text{Heaps} = \text{Heaps} \rightarrow_{\text{fin}} \text{Val}^n$, $\text{Stores} = \text{Vars} \rightarrow \text{Val}$. Each $s \in \text{Stores}$ is called a store. Each $h \in \text{Heaps}$ is called a heap, and $\text{Dom}(h)$ is the domain of $h$, and $\text{Range}(h)$ is the range of $h$. We write $h = h_1 + h_2$ when $\text{Dom}(h_1)$ and $\text{Dom}(h_2)$ are disjoint and the graph of $h$ is the union of those of $h_1$ and $h_2$. A pair $(s, h)$ is called a heap model, which means a memory state. The value $s(x)$ means the value of the variable $x$ in the model $(s, h)$. Each value $a \in \text{Dom}(h)$ means an address, and the value of $h(a)$ is the content of the memory cell at address $a$ in the heap $h$. We suppose each memory cell has $n_{\text{cell}}$ elements as its content.

The interpretation $s(t)$ for any term $t$ is defined as 0 for $\text{nll}$ and $s(x)$ for the variable $x$.

For a formula $F$ we define the interpretation $s, h \models F$ as follows.
s, h \models t_1 = t_2 \text{ if } s(t_1) = s(t_2),
\text{s, h} \models \neg F \text{ if } s, h \not\models F.
\text{s, h} \models t \rightarrow (t_1, \ldots, t_{n_{	ext{var}}}) \text{ if } \text{Dom}(h) = \{s(t)\} \text{ and } h(s(t)) = (s(t_1), \ldots, s(t_{n_{	ext{var}}})),$
\text{s, h} \models F_1 \land F_2 \text{ if } s, h \models F_1 \text{ and } s, h \models F_2,
\text{s, h} \models F_1 \lor F_2 \text{ if } s, h \models F_1 \text{ or } s, h \models F_2,$
\text{s, h} \models F \text{ if } \text{Dom}(h) = \emptyset.

We write $\text{Cells}(F, \Phi)$ if $\bigcup_{x.} x \in \text{V} \wedge \neg \Phi.$

For saving space, we identify some syntactical objects that have the same meaning, namely, we use $\text{Cells}(F, \Phi) = \text{Cells}(F, \Phi)$ if $s[h] := b, h \models F$ for some $b \in \text{Val}.$

We write $A \models B_1, \ldots, B_n$ for $\forall$h$(s, h \models A \rightarrow ((s, h \models B_1) \lor \ldots \lor (s, h \models B_n))).$ The entailment $A \models B_1, \ldots, B_n$ is said to be valid if $A \models B_1, \ldots, B_n$ holds. Our goal in this paper is to decide the validity of a given entailment.

For saving space, we identify some syntactical objects that have the same meaning, namely, we use $\text{Cells}(F, \Phi) = \text{Cells}(F, \Phi)$ if $s[h] := b, h \models F$ for some $b \in \text{Val}.$

We write $A \models B_1, \ldots, B_n$ for $\forall$h$(s, h \models A \rightarrow ((s, h \models B_1) \lor \ldots \lor (s, h \models B_n))).$ The entailment $A \models B_1, \ldots, B_n$ is said to be valid if $A \models B_1, \ldots, B_n$ holds. Our goal in this paper is to decide the validity of a given entailment.

3 Language Extension

3.1 Extended Language

In this section, we extend our language from symbolic heaps by $\downarrow$ and $L$ and $\neg \ast^\downarrow$, which is necessary to show the completeness.

We extend inductive predicate symbols with $Q_1 \ast \ldots \ast \ast \ast \ast \ast Q_m \ast \ast \ast P$ where $Q_1, \ldots, Q_m, P$ are original inductive predicate symbols. We call $m$ the depth of wands. We write $Q_1(T_1) \ast \ldots \ast \ast \ast Q_m(T_m) \ast \ast \ast P(T)$ for $(Q_1 \ast \ldots \ast \ast \ast Q_m \ast \ast \ast P)(T, T_1, \ldots, T_m).$ For a sequence $R = R_1, \ldots, R_n$ of predicates, we write $\vec{R} \ast P$ for $R_1 \ast \ldots \ast \ast \ast R_n \ast P.$

We extend our first-order language with the extended inductive predicate symbols and unary predicate symbols $\downarrow$ and $L$. $t \downarrow$ means that $t$ is in $\text{Dom}(h)$ and $L(t)$ means that $t$ is in $\text{Range}(h) - \text{Dom}(h)$ (the leaves of $h$). We write $t \uparrow$ for $\neg t \downarrow$. We write $t \uparrow$ for $t \uparrow \wedge \neg L t$.

We write $\Sigma, A, B, \phi$ for the same syntactical objects with the extended inductive predicate symbols.

We use $X, Y$ for a finite set of variables and write $X \uparrow$ for $\bigwedge \{t \uparrow \mid t \in X\}$. $X \downarrow$ and $X \uparrow$ are similarly defined. We write $\exists \varphi \downarrow$ for $\exists \varphi(\varphi \downarrow \wedge \ldots)$. Similarly we write $\exists \varphi \uparrow$.

We define:

$\varphi ::= \rightarrow \mid P$ where $P$ varies in inductive predicate symbols,
$\Delta ::= P(T) \wedge X \downarrow$, and $\Gamma ::= \Delta \mid \Gamma \ast \Gamma$, and $\psi ::= Y \uparrow \wedge \Pi \wedge \Gamma$, and $\Phi ::= \exists \varphi \exists \varphi \uparrow \uparrow \Pi \wedge \Gamma,$
$F, G$ a separation logic formula with $\downarrow, L$.

We define entailments as $\psi \models \Phi_1, \ldots, \Phi_n.$

We write $J$ for an entailment. In $\psi, \Phi$, we call $\Gamma$ a spatial part and $\Pi$ a pure part.

We define $\text{Roots}(X \uparrow \wedge Y \uparrow \wedge \Pi \wedge \ast i \in I(P_i(x_i, \overrightarrow{t_i}) \wedge X_i \downarrow)) = \{x_i \mid i \in I\}$. Then we define $\text{Roots}(\exists x \Phi) = \text{Roots}(\Phi)$ if $x \not\in \text{Roots}(\Phi)$, and undefined otherwise. We define $\text{Cells}(X \uparrow \wedge Y \uparrow \wedge \Pi \wedge \ast i \in I(P_i(x_i) \wedge X_i \downarrow)) = \bigcup_{i \in I} X_i$. Then we define $\text{Cells}(\exists x \Phi) = \text{Cells}(\Phi) - \{x\}$.

We write $(\text{Roots} + \text{Cells})(F)$ for $\text{Roots}(F) + \text{Cells}(F)$ and call them address variables of $F$.

We define a substitution as a map from the set of variables to the set of terms. For a substitution $\theta$, we define $\text{Dom}(\theta) = \{x | \theta(x) \neq x\}$ and $\text{Range}(\theta) = \{\theta(x) | x \in \text{Dom}(\theta)\}$. We define a variable renaming as a substitution that is a bijection among variables with a finite domain.

Definition 3.1 $s, h \models t \downarrow$ iff $s(t) \in \text{Dom}(h)$.
\text{s, h} \models L(t) \text{ iff } s(t) \in \text{Range}(h) - \text{Dom}(h),$
3.2 Strong Wand

This section gives the definition clauses for inductive predicates that contain the strong wand.

**Definition 3.2** The definition clauses of \( Q(y, \overline{x}) \rightarrow s^*P(x, \overline{y}) \) are as follows:

Case 1. \( \exists (\overline{w} - z_i)((\overline{w} = \overline{t_i} \wedge \Pi \wedge x \rightarrow (\overline{w}) \ast s_{\overline{f}_i}P_i(z_i, \overline{t_i}))\) where \( Q = P_i \) and \( \exists \overline{x}(\Pi \wedge x \rightarrow (\overline{w}) \ast s_{\overline{f}_i}P_i(z_i, \overline{t_i})) \) is a definition clause of \( P(x, \overline{y}) \).

Case 2. \( \exists \overline{x}(\Pi \wedge x \rightarrow (\overline{w}) \ast s_{\overline{f}_i}P_i(z_i, \overline{t_i}) \ast (Q(y, \overline{w}) \rightarrow s^*P_i(z_i, \overline{t_i}))) \) where \( i \in L \) and \( \exists \overline{x}(\Pi \wedge x \rightarrow (\overline{w}) \ast s_{\overline{f}_i}P_i(z_i, \overline{t_i})) \) is a definition clause of \( P(x, \overline{y}) \).

\( Q(y) \rightarrow s^*P(x) \) is inductively defined by the definition clauses obtained by removing \( Q(y) \) from the definition clauses of \( P(x) \). \( Q(y) \rightarrow s^*P(x) \) plays a similar role to the ordinary magic wand \( Q(y) \rightarrow s*P(x) \), but it is stronger than the ordinary magic wand and it is defined syntactically. Roughly speaking, it is defined to be false if it cannot be defined syntactically.

**Example 3.3** \( ls(y, v) \rightarrow s^*ls(x, w) =_{\text{def}} w = v \wedge x \rightarrow (y) \vee \exists z(x \rightarrow (z) \ast (ls(y, v) \rightarrow s^*ls(z, w))) \).

Note that \( Q_1(\overline{t_1}) \rightarrow s^*Q_2(\overline{t_2}) \rightarrow s^*P(\overline{t}) \) and \( Q_2(\overline{t_2}) \rightarrow s^*Q_1(\overline{t_1}) \rightarrow s^*P(\overline{t}) \) are equivalent, which will be shown by Lemma 5.1.

4 System CSLID\( ^\omega \)

This section defines our logical system CSLID\( ^\omega \).

4.1 Inference Rules

This subsection gives the set of inference rules.

We define \( \text{Dep}(P) \) as the set of inductive predicates symbols that appear in the unfolding of \( P \). We write \( F[F'] \) to explicitly display the subformula \( F' \) in \( F \). We write \( T \) for a finite set of terms. We say \( \Phi \) is equality-full when \( \Pi \) contains \( (\neq, \overline{y}, V \cup \overline{y} \cup \{\text{nil}\}) \) where \( \Phi = \exists \overline{x} \overline{\Psi} \upharpoonright (\Pi \wedge \Gamma) \) and \( V = \text{FV}(\Phi) \).

We write \( \Pi' \subseteq \exists \overline{x} \exists \overline{\Psi} \upharpoonright (\Pi \wedge \Gamma) \) when \( \Pi' \subseteq \Pi \). We call a context \( F[\ ] \) positive when the number of \( (\ ) \rightarrow F' \) that contain \( [ ] \) is even. We call a context \( F[\ ] \) existential when it is of the form \( \exists \overline{x}[\ ] \ast F' \).

We assume a number \( d_{\text{rand}} \) for the maximum depth of wands.

The inference rules are given in the figure 1.

The rule (Factor) derives \( P(t) \) from \( (Q(y) \rightarrow s^*P(t)) \ast Q(y) \) where we list up all the cases for \( Q(y) \) in the disjunction. The rules \( (\exists \text{Amalg1,2}) \) amalgamate \( \exists x \)'s under some condition, which guarantees that existentials have the same values. The rule (\( * \)) is a new split rule since it handles disjunction in the succedent. The other rules are standard.

4.2 Proofs in CSLID\( ^\omega \)

This subsection defines a proof in CSLID\( ^\omega \).

**Definition 4.1** We define a bud and a companion in the same way as [7]. For CSLID\( ^\omega \), we define a cyclic proof to be a proof figure by the inference rules without any open assumptions where each bud has a companion below it and there is some rule \( (* \rightarrow) \) between them.

Instead of the global trace condition in ordinary cyclic proof systems [7], CSLID\( ^\omega \) requires some \( (* \rightarrow) \) rule between a bud and its companion.

4.3 Preparation for Soundness Proof

The following is a key notion for simple soundness proof.

**Definition 4.2** We write \( F \models_m \overline{G} \) when for all \( s, h, |\text{Dom}(h)| \leq m \) and \( s, h \models F \) imply \( s, h \models \overline{G} \). We say \( F \vdash \overline{G} \) is \( m \)-valid and write \( \models_m F \vdash \overline{G} \) when \( F \models_m \overline{G} \).

We call a rule \( m \)-sound when the following holds: if all the assumptions are \( m \)-valid then the conclusion is \( m \)-valid.
Figure 1: Inference Rules
Lemma 4.3 If $x \notin T$, then
\[ F \rightarrow \exists x \uparrow ((x \neq T) \wedge F). \]

Proof. Assume
\[ s, h \models F. \]

There is $a$ such that $a \neq s(T)$ and $\notin \text{Dom}(h) \cup \text{Range}(h)$ since the set of addresses is infinite. Then
\[ s[x := a]_!, h \models x \uparrow \wedge (x \neq T). \]

Hence
\[ s, h \models \exists x \uparrow ((x \neq T) \wedge F). \]
\[ \Box \]

Definition 4.4 For a heap $h$, $\text{Leaves}(h)$ is defined as $\text{Range}(h) \setminus \text{Dom}(h)$.

Lemma 4.5 If
\[ s, h \models \Gamma \]
then $\text{Leaves}(h) \subseteq \bigcup \{ s(\overrightarrow{t} \cup \{\text{nil}\}) \mid P(x, \overrightarrow{t}) \in \Gamma \}$.

Proof. By induction on $|h|$.

Let $\Gamma$ be $\ast_i \Delta_i$. Assume $a \in \text{Leaves}(h)$. We have $h = \Sigma_i h_i$ such that $s, h_i \models \Delta_i$. We have $i$ such that $a \in \text{Leaves}(h_i)$. Let $\Delta_i$ be $P(x, \overrightarrow{t}) \wedge X \downarrow$. Let
\[ s[\overrightarrow{t} := \overrightarrow{p}], h \models \Pi \wedge x \mapsto (\overrightarrow{u}) \ast_{i} P_i(z_i, \overrightarrow{t}_i). \]

$u_i$ is included in $x$, $\overrightarrow{t}$, nil, $\overrightarrow{p}$. Hence $s(\overrightarrow{u}) \setminus \text{Dom}(h)$ is included in $s(\overrightarrow{t}, \text{nil})$.

Let $s'$ be $s[\overrightarrow{p} := \overrightarrow{p}]$ and $h_i = h' + h''$ such that
\[ s', h' \models x \mapsto (\overrightarrow{u}), \]
\[ s', h'' \models \ast_{i} P_i(z_i, \overrightarrow{t}_i). \]

By IH, $\text{Leaves}(h'') \subseteq \bigcup \{ s'(\overrightarrow{t}_i \cup \{\text{nil}\}) \mid l \}$. Since $\overrightarrow{t}_i$ are included in $x$, $\overrightarrow{t}$, nil, $\overrightarrow{p}$, $s'(\overrightarrow{t}_i) \setminus \text{Dom}(h)$ are included in $s(\overrightarrow{t}, \text{nil})$.

Hence $a$ is included in $s(\overrightarrow{t}, \text{nil})$. Hence $\text{Leaves}(h) \subseteq \bigcup \{ s(\overrightarrow{t} \cup \{\text{nil}\}) \mid P(x, \overrightarrow{t}) \in \Gamma \}$. \[ \Box \]

5 Soundness of Rule (Factor) and Properties for Strong Wands

We prove some properties for strong wands.

The order in $\overrightarrow{R}$ is not important in $\overrightarrow{R} \rightarrow^{*} P$, since $\overrightarrow{R} \rightarrow^{*} P$ and $\overrightarrow{R}' \rightarrow^{*} P$ are equivalent when $\overrightarrow{R}'$ is a permutation of $\overrightarrow{R}$. This is shown in the next lemma.

Lemma 5.1 The definition clauses of $\overrightarrow{R}(v, \overrightarrow{u}) \rightarrow^{*} P(x, \overrightarrow{z})$ are
\[ \exists \overrightarrow{w}(\Pi \wedge x \mapsto (\overrightarrow{t}) \ast_{i} R_i(v_i, \overrightarrow{u}_i) \rightarrow^{*} Q_i(v_i, \overrightarrow{s}_i))\] for each definition clause $\exists \overrightarrow{w}(\Pi \wedge x \mapsto (\overrightarrow{t}) \ast Q(w, \overrightarrow{s}))$ of $P(x, \overrightarrow{z})$, and divisions $\overrightarrow{R}(v, \overrightarrow{u}) = (\overrightarrow{R}_i(v_i, \overrightarrow{u}_i))_{i \in I}, \overrightarrow{R}_2(v_2, \overrightarrow{u}_2),$ (with some permutation) and $Q(w, \overrightarrow{s}) = (Q_i(v_i, \overrightarrow{s}_i))_{i \in I}, Q_2(w_2, \overrightarrow{s}_2)$ such that $\overrightarrow{R}_2 = Q_2$. Hence, if $\overrightarrow{R}$ is a permutation of $\overrightarrow{R}$, then $\overrightarrow{R} \rightarrow^{*} P$ is equivalent to $\overrightarrow{R}' \rightarrow^{*} P$. Note that each $\overrightarrow{R}_i$ can be empty.
Proof. By induction on the length of $\overrightarrow{R}$.

Case 0. In this case, $\overrightarrow{Q} = (Q_{1i})$, and all of $(\overrightarrow{R}_{1i})$ and $\overrightarrow{R}_2$ are empty. Hence, (#) is the same as

$\exists \overrightarrow{w}(\Pi \wedge x \mapsto (\overrightarrow{t}^*) * Q(w, \overrightarrow{s}))$.

Case $R'$, $\overrightarrow{R} \rightleftharpoons *P(= R' \rightleftharpoons *P(\overrightarrow{R} \rightleftharpoons *P))$. By the induction hypothesis, a definition clause of $\overrightarrow{R} \rightleftharpoons *P$ is

$\exists(\overrightarrow{w}_{1i})_{i \in I}(\Pi \wedge \overrightarrow{w}_2 = \overrightarrow{s}_2 \wedge x \mapsto (\overrightarrow{t}^*) * _{i \in I}(R_{1i}(v_{i1}, \overrightarrow{w}_{1i}) \cdots _{i \in I}Q_{1i}(w_{i1}, \overrightarrow{s}_{1i})))[\overrightarrow{w}_2 := \overrightarrow{w}_2]$.

for a definition clause $\exists \overrightarrow{w}(\Pi \wedge x \mapsto (\overrightarrow{t}^*) * Q(w, \overrightarrow{s}))$ of $P$, and some divisions $\overrightarrow{R} = (R_{1i}(v_{i1}, \overrightarrow{w}_{1i}), \overrightarrow{R}_2(v_{2}, \overrightarrow{w}_2))$ and $Q = (Q_{1i}(w_{i1}, \overrightarrow{s}_{i1}))_{i \in I}$, $\overrightarrow{Q}_2(v_{2}, \overrightarrow{s}_2)$ such that $\overrightarrow{R}_2 = Q_2$.

By the definition of the strong wand, we have two cases for the definition clauses of $R' \rightleftharpoons *P(\overrightarrow{R} \rightleftharpoons *P)$.

Subcase 1. $R' = \overrightarrow{R}_1 \rightleftharpoons *P(\overrightarrow{R}_1)$ for some $j$, that is, $\overrightarrow{R}_1$ is empty and $R' = Q_{1j}$, and the clause is

$\exists(\overrightarrow{w}_{1i})_{i \neq j}(\Pi \wedge \overrightarrow{w}_2 = \overrightarrow{s}_2 \wedge u_{ij} = s_{ij} \wedge x \mapsto (\overrightarrow{t})^*) * _{i \neq j}(R_{1i}(v_{i1}, \overrightarrow{w}_{1i}) \cdots _{i \neq j}Q_{1i}(w_{i1}, \overrightarrow{s}_{1i})))[\overrightarrow{w}_2 := \overrightarrow{w}_2, w_{1j} := v_{1j}]$.

This is (#) for the division $R', \overrightarrow{R} = ((\overrightarrow{R}_{1i}), \overrightarrow{R}_2)$.

Subcase 2. The clause is

$\exists(\overrightarrow{w}_{1i})_{i \neq j}(\Pi \wedge \overrightarrow{w}_2 = \overrightarrow{s}_2 \wedge x \mapsto (\overrightarrow{t})^*) * _{i \neq j}(R_{1i}(v_{i1}, \overrightarrow{w}_{1i}) \cdots _{i \neq j}Q_{1i}(w_{i1}, \overrightarrow{s}_{1i})) * (R', \overrightarrow{R}_1 \rightleftharpoons *P(\overrightarrow{R}_1))[\overrightarrow{w}_2 := \overrightarrow{w}_2, w_{1j} := v_{1j}]$.

This is (#) for the division $R', \overrightarrow{R} = ((\overrightarrow{R}_{1i}), \overrightarrow{R}_2)$, where $\overrightarrow{R}_{1i} = \overrightarrow{R}_1$ for $i \neq j$, and $\overrightarrow{R}_1 = R', \overrightarrow{R}_1$ for $i = j$. ⊢

The following shows what is derived from the strong wand.

Lemma 5.2 (Strong Wand Elimination) $(Q(y, \overrightarrow{w}) \rightleftharpoons *P(x, \overrightarrow{z})) \models (\overrightarrow{R}(v, \overrightarrow{u}) \rightleftharpoons *P(y, \overrightarrow{w}))$.

Proof. We prove $s, h \models LHS$ implies $s, h \models RHS$ by induction on the size of $h$.

Suppose $s, h \models LHS$, then we have some $h_1 + h_2 = h$ such that

$s, h_1 \models Q(y, \overrightarrow{w}) \rightleftharpoons *P(x, \overrightarrow{z})$

$s, h_2 \models R(v, \overrightarrow{u}) \rightleftharpoons *Q(y, \overrightarrow{w})$.

By the definition of the strong wand, we have the following two cases.

Case 1. We have

$s, h_1 \models \exists(\overrightarrow{r}_Q)(\Pi \wedge x \mapsto (\overrightarrow{t})^*) * s_{l1}S_{1}(r_{l1}, \overrightarrow{t}_{1})))[r_Q := y]$ for a definition clause $\exists \overrightarrow{r}_Q(\Pi \wedge x \mapsto (\overrightarrow{t})^*) * Q(r_Q, \overrightarrow{s}_Q) * s_{l1}S_{1}(r_{l1}, \overrightarrow{t}_{1}))$ of $P(x, \overrightarrow{z})$. Then, we have some $\overrightarrow{t}$ such that

$s[\overrightarrow{r}_Q := \overrightarrow{t}, r_Q := s(y)], h_1 \models \overrightarrow{w} = \overrightarrow{s}_Q \wedge \Pi \wedge x \mapsto (\overrightarrow{t})^*) * s_{l1}S_{1}(r_{l1}, \overrightarrow{t}_{1})$.

Let $s' = s[\overrightarrow{r}_Q := \overrightarrow{t}, r_Q := s(y)]$. We also have

$s', h_2 \models \overrightarrow{r}(v, \overrightarrow{u}) \rightleftharpoons *Q(y, \overrightarrow{w})$,

since none of $\overrightarrow{t}, r_Q$ occurs in $\overrightarrow{R}(v, \overrightarrow{u}) \rightleftharpoons *Q(y, \overrightarrow{w})$. Hence we have

$s', h \models \Pi \wedge x \mapsto (\overrightarrow{t})^*) * (\overrightarrow{R}(v, \overrightarrow{u}) \rightleftharpoons *Q(y, \overrightarrow{w})) * s_{l1}S_{1}(r_{l1}, \overrightarrow{t}_{1})$.

Since $s', h \models \overrightarrow{w} = \overrightarrow{s}_Q$ and $s'(r_Q) = s(y)$, we have

$s', h \models \Pi \wedge x \mapsto (\overrightarrow{t})^*) * (\overrightarrow{R}(v, \overrightarrow{u}) \rightleftharpoons *Q(r_Q, \overrightarrow{s}_Q)) * s_{l1}S_{1}(r_{l1}, \overrightarrow{t}_{1})$,

and hence,

$s, h \models \exists(\overrightarrow{r}_Q)(\Pi \wedge x \mapsto (\overrightarrow{t})^*) * (\overrightarrow{R}(v, \overrightarrow{u}) \rightleftharpoons *Q(r_Q, \overrightarrow{s}_Q)) * s_{l1}S_{1}(r_{l1}, \overrightarrow{t}_{1}),$
which is a definition clause of $\overline{R(v, \overline{u})}\overset{s'}{\rightarrow} P(x, \overline{z})$ by Lemma 5.1

Case 2. We have

$$s, h_1 \models \exists(\overline{\varphi}r)(\Pi \land x \mapsto (\overline{u}) \overset{s}{\rightarrow} (Q(y, \overline{w}) \overset{s}{\rightarrow} S(r, \overline{s})) \overset{*}{\rightarrow} S_l(r_l, \overline{r}_l))$$

for a definition clause $\exists(\overline{\varphi}r)(\Pi \land x \mapsto (\overline{r}) \overset{s}{\rightarrow} (S(r, \overline{s}) \overset{s}{\rightarrow} S_l(r_l, \overline{r}_l))$ of $P(x, \overline{z})$. Then, we have some $\overline{b}$ and $\overline{b}$ such that

$$s[\overline{r} := \overline{b}, r := b], h_1 \models \Pi \land x \mapsto (\overline{u}) \overset{s}{\rightarrow} (Q(y, \overline{w}) \overset{s}{\rightarrow} S(r, \overline{s})) \overset{*}{\rightarrow} S_l(r_l, \overline{r}_l)).$$

Let $s' = s[\overline{r} := \overline{b}, r := b]$, and we have some $h_1 = h_x + h_S + \Sigma_l h_l$ such that

$$s', h_x \models x \mapsto (\overline{r})$$

$$s', h_S \models Q(y, \overline{w}) \overset{s}{\rightarrow} S(r, \overline{s})$$

$$s', h_l \models S_l(r_l, \overline{r}_l) \quad \text{(for each } l).$$

We also have

$$s', h_2 \models \overline{R(v, \overline{u})}\overset{s}{\rightarrow} Q(y, \overline{w}),$$

since none of $\overline{\varphi}, r$ occurs in $\overline{R(v, \overline{u})}\overset{s}{\rightarrow} Q(y, \overline{w})$. Therefore, we have

$$s', h_S + h_2 \models \overline{R(v, \overline{u})}\overset{s}{\rightarrow} S(r, \overline{s})$$

by IH. Hence, we have

$$s', h \models \Pi \land x \mapsto (\overline{r}) \overset{s}{\rightarrow} (\overline{R(v, \overline{u})}\overset{s}{\rightarrow} S(r, \overline{s})) \overset{*}{\rightarrow} S_l(r_l, \overline{r}_l),$$

and then

$$s, h \models \exists(\overline{\varphi}r)(\Pi \land x \mapsto (\overline{r}) \overset{s}{\rightarrow} (\overline{R(v, \overline{u})}\overset{s}{\rightarrow} S(r, \overline{s})) \overset{*}{\rightarrow} S_l(r_l, \overline{r}_l)),$$

which is a definition clause of $\overline{R(v, \overline{u})}\overset{s}{\rightarrow} P(x, \overline{z})$ by Lemma 5.1

By the previous two lemmas, we have

$$(Q(y, \overline{w}), \overline{R(v, \overline{u})}\overset{s}{\rightarrow} P(x, \overline{z})) \overset{*}{\rightarrow} (\overline{R'(v', \overline{u'})}\overset{s}{\rightarrow} Q(y, \overline{w})) \models \overline{R(v, \overline{u})} \overset{s}{\rightarrow} \overline{R'(v', \overline{u'})} \overset{s}{\rightarrow} P(x, \overline{z}).$$

The following shows what derives the strong wand.

**Lemma 5.3 (Strong Wand Introduction)**

$$(\overline{\varphi}, x \neq y \land y \downarrow \land (\overline{R(v, \overline{u})}\overset{s}{\rightarrow} P(x, \overline{z})) \models \exists(\overline{\varphi}(Q(y, \overline{w}), \overline{R_1}(v_1, \overline{u_1})\overset{s}{\rightarrow} P(v_2, \overline{u_2})\overset{s}{\rightarrow} Q(y, \overline{w})) \models \overline{R} = (\overline{R_1}, \overline{R_2}), Q \in \text{Dep}(P), \overline{R_2} \subseteq \text{Dep}(Q))$$

**Proof.** We prove $s, h \models \text{LHS}$ implies $s, h \models \text{RHS}$ by induction on the size of $h$.

Suppose $s, h \models \text{LHS}$. Since $s, h \models \overline{R(v, \overline{u})}\overset{s}{\rightarrow} P(x, \overline{z})$, we have

$$s, h \models \exists(r_{A_1})(\Pi \land \overline{\overline{R}} B = \overline{S_B} \land x \mapsto (\overline{r}) \overset{s}{\rightarrow} (\overline{R_{A_1}(v_{A_1}, \overline{u_{A_1}})}, \overline{S_{A_1}(r_{A_1}, \overline{s_{A_1}})})[\overline{B} := \overline{B}]$$

for some definition clause

$$\exists(r_{A_1}), \overline{B} \models (\Pi \land x \mapsto (\overline{r}) \overset{s}{\rightarrow} (\overline{R_{A_1}(v_{A_1}, \overline{u_{A_1}})}, \overline{S_{A_1}(r_{A_1}, \overline{s_{A_1}})}))$$

of $P$ and some division $\overline{R}(v, \overline{u}) = (\overline{R_{A_1}(v_{A_1}, \overline{u_{A_1}})}), \overline{S_B}(r_B, \overline{s_B})$ such that $\overline{R} = \overline{S_B}$. Then, we have some $(b_{A_1})$, such that

$$s[(r_{A_1} := b_{A_1}), \overline{B} := s(\overline{B})], h \models \Pi \land \overline{\overline{B}} = \overline{S_B} \land x \mapsto (\overline{r}) \overset{s}{\rightarrow} (\overline{R_{A_1}(v_{A_1}, \overline{u_{A_1}})}, \overline{S_{A_1}(r_{A_1}, \overline{s_{A_1}})}).$$

Let $s' = s[(r_{A_1} := b_{A_1}), \overline{B} := s(\overline{B})]$. We have $h = h_x + \Sigma h_{A_1}$ such that

$$s', h_x \models x \mapsto (\overline{r})$$

$$s', h_{A_1} \models \overline{R_{A_1}(v_{A_1}, \overline{u_{A_1}})}\overset{s}{\rightarrow} S_{A_1}(r_{A_1}, \overline{s_{A_1}}).$$
Since \( s' \models x \neq y \), we have \( y \in \text{dom}(h_{Ak}) \) for some \( k \), and hence we have
\[
s', h_{Ak} \models y \downarrow \widehat{R}_{Ak}(v_{Ak}, \overrightarrow{u}_{Ak}) \rightarrow ^{*} S_{Ak}(r_{Ak}, \overrightarrow{s}_{Ak}).
\]

Case \( s'(y) = b_{Ak}(= s'(r_{Ak})) \). Let \( Q = S_{Ak}, \quad \overrightarrow{R}_1 = \overrightarrow{R} - \overrightarrow{R}_{Ak}, \) and \( \overrightarrow{R}_2 = \overrightarrow{R}_{Ak} \). Note that \( Q \in \text{Dep}(P) \) and \( \overrightarrow{R}_2 \subseteq \text{Dep}(Q) \). Then, we have
\[
s', h_{Ak} \models \overrightarrow{R}_2(v_2, \overrightarrow{u}_2) \rightarrow ^{*} Q(y, \overrightarrow{s}_{Ak}).
\]

We show
\[
s', \Sigma_{i \neq k} h_{Ai} + h_x \models Q(y, \overrightarrow{s}_{Ak}), \quad \overrightarrow{R}_1(v_1, \overrightarrow{u}_1) \rightarrow ^{*} P(x, \overrightarrow{v}).
\]

Now,
\[
\exists (r_{Ai})_i \overrightarrow{R}_B(\Pi \times x \mapsto (\overrightarrow{R})*i \neq k S_{Ai}(r_{Ai}, \overrightarrow{s}_{Ai}) \rightarrow Q(r_{Ak}, \overrightarrow{s}_{Ak}) \rightarrow S_{B}(r_{B}, \overrightarrow{s}_{B}))
\]
is a definition clause of \( P \). Consider the division \( \overrightarrow{R} = ((\overrightarrow{R}_{Ai})_{i \neq k}), (Q, \overrightarrow{R}_B) \), and it is sufficient to show that
\[
s', \Sigma_{i \neq k} h_{Ai} + h_x \models \exists (r_{Ai})_i \overrightarrow{R}_B(\Pi \times x \mapsto (\overrightarrow{R})*i \neq k S_{Ai}(r_{Ai}, \overrightarrow{s}_{Ai}) \rightarrow Q(r_{Ak}, \overrightarrow{s}_{Ak})) \rightarrow \overrightarrow{R}_B := \overrightarrow{v}_B, r_{Ak} := y \]
which follows from
\[
s' \models \overrightarrow{v}_B := \overrightarrow{s}_B \quad \overrightarrow{R}_B := \overrightarrow{s}_B \quad \overrightarrow{R}_B := \overrightarrow{s}_B \quad \overrightarrow{R}_B := \overrightarrow{s}_B
ds'(r_{Ak}) = s'(y)
\]
\[
s', h_x \models x \mapsto (\overrightarrow{R})
\]
\[
s', h_i \models \overrightarrow{R}_{Ai}(v_{Ai}, \overrightarrow{u}_{Ai}) \rightarrow ^{*} S_{Ai}(r_{Ai}, \overrightarrow{s}_{Ai}) \quad \text{(for } i \neq k)\]

Therefore, we have
\[
s', h \models (Q(y, \overrightarrow{s}_{Ak}), \overrightarrow{R}_A \rightarrow ^{*} P(x, \overrightarrow{v})) \rightarrow (\overrightarrow{R}_B(v_B, \overrightarrow{u}_B) \rightarrow ^{*} Q(y, \overrightarrow{s}_{Ak})),
\]
so we have
\[
s', h \models \exists \overrightarrow{v}(Q(y, \overrightarrow{v}), \overrightarrow{R}_A \rightarrow ^{*} P(x, \overrightarrow{v})) \rightarrow (\overrightarrow{R}_B(v_B, \overrightarrow{u}_B) \rightarrow ^{*} Q(y, \overrightarrow{v})),
\]
and hence
\[
s, h \models \exists \overrightarrow{v}(Q(y, \overrightarrow{v}), \overrightarrow{R}_A \rightarrow ^{*} P(x, \overrightarrow{v})) \rightarrow (\overrightarrow{R}_B(v_B, \overrightarrow{u}_B) \rightarrow ^{*} Q(y, \overrightarrow{v})),
\]

since \( (r_{Ai})_i, \overrightarrow{R}_B \not\in \text{RHS} \).

Case \( s'(y) \neq b_{Ak} \). In this case, we have
\[
s', h_{Ak} \models r_{Ak} \neq y \downarrow y \downarrow \widehat{R}_{Ak}(v_{Ak}, \overrightarrow{u}_{Ak}) \rightarrow ^{*} S_{Ak}(r_{Ak}, \overrightarrow{s}_{Ak}).
\]

By IH, there exist \( Q \in \text{Dep}(S_{Ak}) \subseteq \text{Dep}(P) \) and a division \( \overrightarrow{R}_{Ak} = \overrightarrow{R}_{1Ak}, \overrightarrow{R}_{2Ak} \) such that \( \overrightarrow{R}_{2Ak} \subseteq \text{Dep}(Q) \) and
\[
s', h_{Ak} \models \exists \overrightarrow{v}(\overrightarrow{Q}(y, \overrightarrow{v}), \overrightarrow{R}_{1Ak}(v_{1Ak}, \overrightarrow{u}_{1Ak}) \rightarrow ^{*} S_{Ak}(r_{Ak}, \overrightarrow{s}_{Ak})) \rightarrow (\overrightarrow{R}_{2Ak}(v_{2Ak}, \overrightarrow{u}_{2Ak}) \rightarrow ^{*} Q(y, \overrightarrow{v})).
\]

There exists \( \overrightarrow{v} \) such that
\[
s'[\overrightarrow{v} := \overrightarrow{v}], h_{1Ak} \models Q(y, \overrightarrow{v}), \overrightarrow{R}_{1Ak}(v_{1Ak}, \overrightarrow{u}_{1Ak}) \rightarrow ^{*} S_{Ak}(r_{Ak}, \overrightarrow{s}_{Ak})
\]
\[
s'[\overrightarrow{v} := \overrightarrow{v}], h_{2Ak} \models \overrightarrow{R}_{2Ak}(v_{2Ak}, \overrightarrow{u}_{2Ak}) \rightarrow ^{*} Q(y, \overrightarrow{v}),
\]
and then we have
\[
s'[\overrightarrow{v} := \overrightarrow{v}], h_x + \Sigma_{i \neq k} h_i + h_{1Ak} \models \Pi \downarrow \overrightarrow{v}_B := \overrightarrow{s}_B \downarrow
\]
\[
\rightarrow (\overrightarrow{R}) \rightarrow ^{*} i \neq k S_{Ai}(r_{Ai}, \overrightarrow{s}_{Ai}) \rightarrow Q(y, \overrightarrow{v}), \overrightarrow{R}_{1Ak}(v_{1Ak}, \overrightarrow{u}_{1Ak}) \rightarrow ^{*} S_{Ak}(r_{Ak}, \overrightarrow{s}_{Ak}))
\]

Let \( \overrightarrow{R}_1 = ((\overrightarrow{R}_{Ai})_{i \neq k}), \overrightarrow{R}_{1Ak}, \) and \( \overrightarrow{R}_2 = \overrightarrow{R}_{2Ak} \subseteq \text{Dep}(Q) \), and then we have
\[
s'[\overrightarrow{v} := \overrightarrow{v}], h_x + \Sigma_{i \neq k} h_i + h_{1Ak} \models Q(y, \overrightarrow{v}), \overrightarrow{R}_1 \rightarrow ^{*} P
\]
and hence
\[ s'[\overrightarrow{v} := \overrightarrow{v'}], h \models (Q(y, \overrightarrow{v}), \overrightarrow{R}_1 \rightarrow \overrightarrow{v} P) \ast (\overrightarrow{R}_2 \rightarrow \overrightarrow{v} Q(y, \overrightarrow{v})). \]

Therefore, we have
\[ s, h \models \exists \overrightarrow{v} ((Q(y, \overrightarrow{v}), \overrightarrow{R}_1 \rightarrow \overrightarrow{v} P) \ast (\overrightarrow{R}_2 \rightarrow \overrightarrow{v} Q(y, \overrightarrow{v}))). \]

\[ \square \]

**Lemma 5.4 (Soundness of Rule (Factor))** The rule (Factor) is m-sound.

**Proof.** Assume the antecedent of the conclusion is true at \((s, h)\) and \(|\text{Dom}(h)| \leq m\) in order to show the succedent of the conclusion is true at \((s, h)\). Then the antecedent of the assumption is true at \((s, h)\). Then the succedent of the assumption is true at \((s, h)\). By Lemma 5.2 the succedent of the conclusion is true at \((s, h)\). \(\square\)

### 6 Soundness of Rules (\(\exists\) Amalg1, 2)

**Lemma 6.1** (1) If \(F\) is a formula constructed from \(=, \neq, \text{emp}, \rightarrow, \land, \lor, \ast, \exists x, \downarrow\), and \(T = \text{FV}(F) \cup \{\text{nil}\} - \overrightarrow{y}, \) and \(\overrightarrow{v} \notin s(T) \cup \text{Dom}(h) \cup \text{Range}(h), \) and \(\overrightarrow{v} \) are different from each other, and
\[ s, h \models \overrightarrow{y} \uparrow \land (\neq (\overrightarrow{y}, T \cup \overrightarrow{y})) \land F \]
then
\[ s[\overrightarrow{y} := \overrightarrow{v}], h \models F. \]

(2) If \(\exists y \uparrow \Phi\) is equality-full, and \(a \notin s(\text{FV}(\exists y \uparrow \Phi) \cup \{\text{nil}\}) \cup \text{Dom}(h) \cup \text{Range}(h)\) and
\[ s, h \models \exists y \uparrow \Phi, \]
then
\[ s[y := a], h \models \Phi. \]

(3) If \(\exists x \uparrow \Phi_1\) and \(\exists x \uparrow \Phi_2\) are equality-full, then \(\exists x \uparrow \Phi_1 \ast \exists x \uparrow \Phi_2 \rightarrow \exists x \uparrow (\Phi_1 \ast \Phi_2).\)

(4) If \(x \in \text{Cells}(\Phi_1),\) and \((x \neq \text{FV}(\exists x \uparrow \Phi_2)) \subseteq \Phi_1,\) and \(\exists x \uparrow \Phi_2\) is equality-full, then \(\exists x \Phi_1 \ast \exists x \uparrow \Phi_2 \rightarrow \exists x(\Phi_1 \ast (x \uparrow \land \Phi_2)).\)

**Proof.** (1) We show the claim by induction on \(F.\)

If \(F\) is \(F_1 \ast F_2\) or \(F_1 \land F_2\) or \(F_1 \lor F_2\), the claim immediately follows from IH.

Case 1. \(\exists x \downarrow F_1,\)
We have \(b \in \text{Dom}(h)\) such that
\[ s[x := b], h \models F_1. \]
Since \(\overrightarrow{v} \notin \text{Dom}(h),\) we have \(\overrightarrow{v} \neq b\) and \(s(\overrightarrow{y}) \neq b.\) From
\[ s[x := b], h \models \overrightarrow{y} \uparrow \land (\neq (\overrightarrow{y}, T \cup \{x\} \cup \overrightarrow{y})) \land F_1, \]
by IH
\[ s[x := b, \overrightarrow{y} := \overrightarrow{v}], h \models F_1. \]
Hence
\[ s[\overrightarrow{v} := \overrightarrow{v'}], h \models \exists x \downarrow F_1. \]

Case 2. \(\text{emp}.\) The claim immediately follows.
Case 3. \(t \rightarrow (\overrightarrow{y}).\)
\[ t \notin \overrightarrow{y}\) since \(s(t) \in \text{Dom}(h).\) \(\overrightarrow{v} \neq \overrightarrow{y}\) since \(s(\overrightarrow{v}) \subseteq \text{Range}(h).\) Since \(F\) does not contain \(\overrightarrow{y},\) the claim holds.
Case 4. \(t \downarrow.\)
\( t \not\in \overline{y} \) since \( s(t) \in \text{Dom}(h) \). Since \( F \) does not contain \( \overline{y} \), the claim holds.

Case 5. \( u = t \).

If \( u, t \not\in \overline{y} \), the claim immediately follows. If \( u = t \), the claim immediately follows. If \( u, t \in \overline{y} \) and \( u \neq t \), it contradicts with \( (\overline{y}, T \cup \overline{y}) \). If \( u \in \overline{y} \) and \( t \notin \overline{y} \), then \( t \in T \), which contradicts with the assumption \( \overline{y} \neq T \).

Case 6. \( u \neq t \).

If \( u, t \not\in \overline{y} \), the claim immediately follows. If \( u = t \), it contradicts with \( u \neq t \). If \( u, t \in \overline{y} \) and \( u \neq t \), \( s[\overline{y} := \overline{a}] \models u \neq t \) since \( \overline{a}, \overline{y} \) are different. If \( u \in \overline{y} \) and \( t \notin \overline{y} \), then \( t \in T \), so \( s[\overline{y} := \overline{a}] \models u \neq t \) since \( \overline{a} \notin s(T) \).

(2) Let \( T \) be \( \text{FV}(\exists y \uparrow \Phi) \cup \{ \text{nil} \} \). Let \( \Phi \) be \( \exists \overline{x} \exists \overline{y} \uparrow (\Pi \land \Gamma) \). Note that \( \overline{a} \in \text{Cells}(\Gamma) \). We have \( m \) such that

\[ s, h \models \exists y \uparrow \Phi(m). \]

We have \( \overline{a}, \overline{b} \in \text{Dom}(h) \) and \( \overline{b}, c \) such that

\[ s[\overline{x} := \overline{a}, \overline{y} := \overline{b}, y := c], h \models \overline{y} \uparrow \Pi \land \Gamma(m). \]

Since \( \overline{a} \in \text{Dom}(h) \), we have \( a \neq \overline{a}. \) From the equality-fullness,

\[ s[\overline{x} := \overline{a}, \overline{y} := \overline{b}, y := c] \models (\neq (y \overline{y}, T \cup \{ \overline{x}, \overline{y} \})). \]

Choose \( \overline{b}' \) such that \( \overline{b}' \notin s(T) \cup \text{Dom}(h) \cup \text{Range}(h), \) and \( \overline{b}' \neq a, \) and \( \overline{b}', \overline{b} \) are different from each other. By taking \( \overline{y} \) to be \( y \overline{y}, \overline{a} \) to be \( a \overline{b}' \) and \( s \) to be \( s[\overline{x} := \overline{a}, \overline{y} := \overline{b}, y := c] \) in (1),

\[ s[\overline{x} := \overline{a}, \overline{y} := \overline{b}', y := a], h \models \Pi \land \Gamma(m). \]

Hence

\[ s[y := a], h \models \exists \overline{x} \exists \overline{y} \uparrow (\Pi \land \Gamma(m)). \]

Hence

\[ s[y := a], h \models \Phi. \]

(3) Assume

\[ s, h \models \exists x \uparrow \Phi_1 \land \exists x \uparrow \Phi_2. \]

We have \( h_1 + h_2 = h \) such that

\[ s, h_1 \models \exists x \uparrow \Phi_1, \]
\[ s, h_2 \models \exists x \uparrow \Phi_2. \]

There is \( a \) such that \( a \notin s(\text{FV}(\Phi_1, \Phi_2) \cup \{ \text{nil} \} - \{ x \}) \cup \text{Dom}(h) \cup \text{Range}(h) \). By (2),

\[ s[x := a], h_1 \models \Phi_1, \]
\[ s[x := a], h_2 \models \Phi_2. \]

Hence

\[ s[x := a], h_1 + h_2 \models \Phi_1 \land \Phi_2. \]

Hence

\[ s, h \models \exists x \uparrow (\Phi_1 \land \Phi_2). \]

(4) Assume

\[ s, h \models \exists x \Phi_1 \land \exists x \uparrow \Phi_2. \]

Then we have \( h_1 + h_2 = h \) such that

\[ s, h_1 \models \exists x \Phi_1, \]
\[ s, h_2 \models \exists x \uparrow \Phi_2. \]
Then we have \( a \in \text{Dom}(h_1) \), \( b \) such that
\[
\begin{align*}
  s[x := a], h_1 &\models \Phi_1, \\
  s[x := b], h_2 &\models x \uparrow \land \Phi_2.
\end{align*}
\]

Let \( \Phi_2 \) be \( \exists x \exists \bar{y} \uparrow (\Pi_2 \land \Gamma_2) \). Then we have \( \bar{a} \in \text{Dom}(h_2) \) and \( \bar{b} \notin \text{Dom}(h_2) \cup \text{Range}(h_2) \) such that by letting \( s' \) be \( s[x := b], \bar{x}_2 := \bar{a}, \bar{y}_2 := \bar{b} \),
\[
  s', h_2 \models \Pi_2 \land \Gamma_2.
\]

We will show
Claim 1: \( s[x := a], h_2 \models x \uparrow \land \Phi_2 \).
If \( a = b \), the claim immediately follows. Assume \( a \neq b \).
We can show \( a \notin \text{Range}(h_2) \) as follows. Assume \( a \in \text{Range}(h_2) \) in order to show contradiction.
\( a \neq \bar{a} \) since \( a \in \text{Dom}(h_1) \). \( a \neq \bar{b} \) since \( \bar{b} \notin \text{Range}(h_2) \). \( a \neq s(\text{FV}(\exists x \uparrow \Phi_2)) \) since \( x \neq \text{FV}(\exists x \uparrow \Phi_2) \). By Lemma 4.5, \( a \in s'(\text{FV}(\Gamma_2) \cup \{\text{nil}\}) \), which contradicts.

We have shown \( a \notin \text{Range}(h_2) \). Hence \( a \notin s(\text{FV}(\exists x \uparrow \Phi_2) \cup \{\text{nil}\}) \cup \text{Dom}(h_2) \cup \text{Range}(h_2) \). By (2),
\[
  s[x := a], h_2 \models x \uparrow \land \Phi_2.
\]
Hence we have shown the claim 1.

Hence
\[
  s[x := a], h \models \Phi_1 \ast (x \uparrow \land \Phi_2).
\]
Hence
\[
  s, h \models \exists x (\Phi_1 \ast (x \uparrow \land \Phi_2)).
\]

\( \square \)

**Lemma 6.2 (Soundness of Rules (\( \exists \text{Amalg1}, 2) \))** The rules \( \exists \text{Amalg1} \) and \( \exists \text{Amalg2} \) are m-sound.

**Proof.** We consider both rules simultaneously. Assume the antecedent of the conclusion is true at \((s, h)\) and \(|\text{Dom}(h)| \leq m\) in order to show the succedent of the conclusion is true at \((s, h)\). Then the antecedent of the assumption is true at \((s, h)\). Then the succedent of the assumption is true at \((s, h)\). By Lemma 6.1 (3) and (4) for \( \exists \text{Amalg1} \) \( \exists \text{Amalg2} \) respectively, the succedent of the conclusion is true at \((s, h)\). \( \square \)

7 **Soundness of Rule (*)**

We will show the soundness of the rule (*). It is short but one of the most interesting parts in our contribution.

**Lemma 7.1** For propositional variables \( A^i_k \) \((k = 1, 2, i \in I)\), the following is true in the propositional logic:
\[
\bigwedge_{l_1 + l_2 = I} \left( \left( \bigvee_{i \in l_1} A^i_1 \right) \lor \left( \bigvee_{i \in l_2} A^i_2 \right) \right) \leftrightarrow \bigvee_{i \in I} (A^i_1 \land A^i_2).
\]

**Proof.** \( \Rightarrow \): Assume the negation of the right-hand side
\[
\neg \bigvee_{i \in I} (A^i_1 \land A^i_2)
\]
in order to show the negation of the left-hand side. It is
\[
\bigwedge_{i \in I} (\neg A^i_1 \lor A^i_2)
\]

13
Take \( I_1 \) to be \( \{ i \mid \neg A^1_i \} \) and \( I_2 \) be \( I - I_1 \). Then
\[
\left( \bigwedge_{i \in I_1} \neg A^1_i \right) \land \left( \bigwedge_{i \in I_2} \neg A^2_i \right).
\]

Hence
\[
\neg \left( \left( \bigvee_{i \in I_1} A^1_i \right) \lor \left( \bigvee_{i \in I_2} A^2_i \right) \right).
\]

Hence the negation of the left-hand side is true.

\[ \Leftarrow: \text{Assume the } i \text{-th disjunct is true. \ For any } I = I_1 + I_2, i \in I_1 \text{ or } i \in I_2, \text{ so the conjunct of the left-hand side is true. } \square \]

**Lemma 7.2 (Split Lemma)** The rule
\[
\frac{F_1 \vdash \{ G^1_i \mid i \in I' \} \text{ or } F_2 \vdash \{ G^2_i \mid i \in I ' \} \text{ (}\forall I' \subseteq I\text{)}\!
}{F_1 \ast F_2 \vdash \{ G^1_i \ast G^2_i \mid i \in I \}} (\ast)
\]
is \( m \)-sound.

**Proof.** For each \( I' \subseteq I \), assume
\[ F_1 \models_m \{ G^1_i \mid i \in I' \} \]
or
\[ F_2 \models_m \{ G^2_i \mid i \in I' \} \]
We will show \( F_1 \ast F_2 \models_m \{ G^1_i \ast G^2_i \mid i \in I \} \).

Assume \( s, h \models F_1 \ast F_2 \) and \( |\text{Dom}(h)| \leq m \).

We have \( h = h_1 + h_2 \) such that \( s, h_i \models \Gamma_i \) \((i = 1, 2)\). Then
\[ s, h_1 \models F_1, \]
\[ s, h_2 \models F_2. \]

Since \(|\text{Dom}(h_1)|, |\text{Dom}(h_2)| \leq m\), \( \bigvee_{i \in I_1} s, h_1 \models G^1_i \) or \( \bigvee_{i \in I - I'} s, h_2 \models G^2_i \). Since this holds for each \( I' \subseteq I \), by letting \( I_1 = I' \) and \( I_2 = I - I' \),
\[
\bigwedge_{i_1 + i_2 = I} \left( \bigvee_{i \in I_1} s, h_1 \models G^1_i \right) \lor \left( \bigvee_{i \in I_2} s, h_2 \models G^2_i \right).
\]

By Lemma 7.1 \( \bigvee_{i \in I} (s, h_1 \models G^1_i \land s, h_2 \models G^2_i) \). Therefore \( s, h \models G^1_i \ast G^2_i \) for some \( i \in I \). \( \square \)

## 8 Soundness of CSLID\( ^\omega \)

This section proves the soundness theorem of CSLID\( ^\omega \). The soundness proof uses the fact that \(|\text{Dom}(h)|\) decreases upwardly by the rule \((\ast \rightarrow)\).

**Lemma 8.1** (1) For the rule \((\ast \rightarrow)\), if the assumptions are \( m \)-valid then the conclusion is \((m + 1)\)-valid.

(2) Every rule except \((\ast \rightarrow)\) is \( m \)-sound.

**Proof.** (1) Assume the antecedent of the conclusion is true at \((s, h)\) and \(|\text{Dom}(h)| \leq m + 1\) in order to show the succedent of the conclusion is true at \((s, h)\). We have \( h_1 + h_2 = h \) such that the first conjunct is true at \((s, h_1)\) and the second conjunct is true at \((s, h_2)\). Then the antecedent of the assumption is true at \((s, h_1)\). Since \(|\text{Dom}(h_1)| \leq m\), the succedent of the assumption is true at \((s, h_1)\). Hence the succedent of the conclusion is true at \((s, h)\).

(2) Lemmas 6.4, 6.2 and 7.2 show \( m \)-soundness of (Factor), (∃Amalg) and (\(\ast\)) respectively. \((\dagger R)\) is \( m \)-sound by Lemma 14.3 The claim for the other rules apparently holds. \( \square \)

When we allow open assumptions in a proof, we call it an open proof.
Lemma 8.2  (1) For any bud $J$ in a given proof, $\models_m J$.
(2) If $F \vdash G$ has a proof then $F \models_m \overline{G}$ for any $m$.

Proof. (1) Assume $D$ is a proof. For a bud $J$ in $D$, we define the height $|J|$ as the number of judgments in the path from the conclusion to its companion in $D$. Note that the path is not to the bud. For a bud $J$ in $D$, we will show $\models_m J$ by induction on $(m, |J|)$.

Let $J'$ be the companion of $J$. Consider the open subproof $\pi_J$ of $J'$ in $D$.

For a bud $J_1$ in $\pi_J$ such that $|J_1| \geq |J|$, the companion of $J_1$ is between $J'$ and $J_1$. Hence there is a rule application ($* \mapsto \rightarrow$) between $J'$ and $J_1$. We remove the path from the assumption of the rule to $J_1$. We do this for every such bud to obtain an open proof of $\pi'_{J'}$ where the open assumptions are buds of smaller heights or the conclusions of rule applications ($* \mapsto \rightarrow$). For an open assumption $J_2$ of a bud in $\pi'_{J'}$, by IH for $|J_2| < |J|$, $\models_m J_2$. Let $J_3$ be an open assumption of the conclusion of the rule ($* \mapsto \rightarrow$) in $\pi'_{J'}$. By case analysis of $m$, we show $\models_m J_3$.

Case $m = 0$. Since the antecedent of $J_3$ is false with the empty heap, $\models_m J_3$.

Case $m > 0$. By IH for $m - 1$, the assumption of the rule ($* \mapsto \rightarrow$) is $(m - 1)$-valid. By Lemma 8.1 (1), $J_3$ is $m$-valid.

Hence, in both cases, $\models_m J_3$. Since $J_3$ is arbitrary, all the open assumptions of $\pi'_{J'}$ are $m$-valid. By Lemma 8.1 (1)(2), $\models_m J'$ so $\models_m J$.

(2) Assume a proof $D$ of $F \vdash G$. By (1), every bud of $D$ is $m$-valid. By Lemma 8.1 (1)(2), $F \models_m \overline{G}$. □

Theorem 8.3 (Soundness) If $J$ is provable in CSLIDω, then $J$ is valid.

Proof. Let $J$ be $F \vdash \overline{G}$. Assume $s, h \models F$. Let $m$ be $|\Dom(h)|$. By Lemma 8.2 (2), $F \models_m \overline{G}$. Hence $s, h \models \overline{G}$. □

9 Satisfiability Checking

This section gives a satisfiability checking procedure for $\psi$.

For each inductive predicate symbol $P$ of arity $m$ and each $n \geq 0$, we add an inductive predicate symbol $\hat{P}^n$ of arity $m + n$. For simplicity, we write $\hat{P}$ for every $\hat{P}^n$, since $n$ is determined by the number of arguments.

We define the definition clauses for $\hat{P}(x, \overline{\tau}, \overline{\gamma})$ as the following (1) and (2) for each definition clause of $P(x, \overline{\tau})$. Let the definition clause of $P(x, \overline{\tau})$ be

$\exists \overline{\tau'}(\Pi \land x \mapsto (\overline{\tau'}) * \hat{s}, \hat{P}(t_i, \overline{\tau'}))$.

(1) For each $y_i((\overline{\gamma})_i)$ such that the sequence $y((\overline{\gamma})_i)$ is a permutation of the sequence $\overline{\gamma}$ and the sequence $\overline{\gamma}_i$ are in the same order as the sequence $\overline{\gamma}$, we define the definition clause of $\hat{P}(x, \overline{\tau}, \overline{\gamma})$ by

$\exists \overline{\tau'}(\Pi \land x \mapsto y \land x \mapsto (\overline{\tau'}) * \hat{s}, \hat{P}(t_i, \overline{\tau'}))$.

(2) For each $(\overline{\gamma}_i)_i$ such that the sequence $(\overline{\gamma}_i)_i$ is a permutation of the sequence $\overline{\gamma}$ and the sequence $\overline{\gamma}_i$ are in the same order as the sequence $\overline{\gamma}$, we define the definition clause of $\hat{P}(x, \overline{\tau}, \overline{\gamma})$ by

$\exists \overline{\tau'}(\Pi \land x \mapsto (\overline{\tau'}) * \hat{s}, \hat{P}(t_i, \overline{\tau'}))$.

Lemma 9.1 We have $P(x, \overline{\tau}) \land \overline{\gamma} \downarrow \iff \hat{P}(x, \overline{\tau}, \overline{\gamma})$.

Proof. →:

It is sufficient to show the following claim for all $m$.

Claim 1: $P^{(m)}(x, \overline{\tau}) \land \overline{\gamma} \downarrow \iff P^{(m)}(x, \overline{\tau}, \overline{\gamma})$.

We show this claim by induction on $m$.

Assume

$s, h \models P^{(m)}(x, \overline{\tau}) \land \overline{\gamma}$

in order to show $s, h \models \hat{P}^{(m)}(x, \overline{\tau}, \overline{\gamma})$. □
If \( m = 0 \), the immediately holds. Assume \( m > 0 \).
We have a definition clause of \( P(x, \overline{x}) \)
\[
\exists \overline{x} (\Pi \land x \mapsto (\overline{x}) \ast *_{!} P(t_{i}, \overline{t}_{i}))
\]
such that
\[
s, h \models \exists \overline{x} (\Pi \land x \mapsto (\overline{x}) \ast *_{!} P^{(m-1)}(t_{i}, \overline{t}_{i})).
\]

Consider cases according to \( s(x) \in s(\overline{y}) \).
Case \( s(x) \in s(\overline{y}) \).
Let \( y \in \overline{y} \) such that \( s(x) = s(y) \).
We have \((\overline{y}_{i})_{i} \) such that \( y(\overline{y}_{i})_{i} \) is a permutation of \( \overline{y} \), \( \overline{y}_{i} \) are in the same order as \( \overline{y} \) and
\[
s, h \models \exists \overline{x} (\Pi \land x = y \land x \mapsto (\overline{x}) \ast *_{i} (P^{(m-1)}(t_{i}, \overline{t}_{i}) \land \overline{y}_{i}\downarrow)).
\]

By IH,
\[
s, h \models \exists \overline{x} (\Pi \land x = y \land x \mapsto (\overline{x}) \ast *_{i} P^{(m-1)}(t_{i}, \overline{t}_{i}, \overline{y}_{i})).
\]

Hence \( s, h \models \hat{P}^{(m)}(x, \overline{x}, \overline{y}). \)

Case \( s(x) \notin s(\overline{y}) \).
We have \((\overline{y}_{i})_{i} \) such that \( (\overline{y}_{i})_{i} \) is a permutation of \( \overline{y} \), \( \overline{y}_{i} \) are in the same order as \( \overline{y} \), and
\[
s, h \models \exists \overline{x} (\Pi \land x \mapsto (\overline{x}) \ast *_{i} (P^{(m-1)}(t_{i}, \overline{t}_{i}) \land \overline{y}_{i}\downarrow)).
\]

By IH,
\[
s, h \models \exists \overline{x} (\Pi \land x \mapsto (\overline{x}) \ast *_{i} \hat{P}^{(m-1)}(t_{i}, \overline{t}_{i}, \overline{y}_{i})).
\]

Hence \( s, h \models \hat{P}^{(m)}(x, \overline{x}, \overline{y}). \)

We have shown the claim 1.
\( \leftarrow \):
It is sufficient to show the following claim.
Claim 2: \( \hat{P}^{(m)}(x, \overline{x}, \overline{y}) \rightarrow P^{(m)}(x, \overline{x}) \land \overline{y}\downarrow \).

By induction on \( m \) we show this claim.
If \( m = 0 \), the immediately holds. Assume \( m > 0 \).
Assume \( s, h \) satisfies the antecedent.
We have cases according to the definition clause (1) or (2) of \( \hat{P}(x, \overline{x}, \overline{y}) \).
Case (1). Let the definition clause be
\[
\exists \overline{x} (\Pi \land x = y \land x \mapsto (\overline{x}) \ast *_{!} \hat{P}(t_{i}, \overline{t}_{i}, \overline{y}_{i}))
\]
and
\[
s, h \models \exists \overline{x} (\Pi \land x = y \land x \mapsto (\overline{x}) \ast *_{!} \hat{P}^{(m-1)}(t_{i}, \overline{t}_{i}, \overline{y}_{i})).
\]

By IH,
\[
s, h \models \exists \overline{x} (\Pi \land x = y \land x \mapsto (\overline{x}) \ast *_{!} (P^{(m-1)}(t_{i}, \overline{t}_{i}) \land \overline{y}_{i}\downarrow)).
\]

Hence \( s, h \models P^{(m)}(x, \overline{x}) \land \overline{y}\downarrow \).

Case (2) is similar to the case (1). We have shown the claim 2. \( \square \)

We check the satisfiability of \( \psi \) by extending the decision procedure for the satisfiability of a symbolic heap given in [10].

**Theorem 9.2 ([10])** For a given symbolic heap \( \phi \), we can effectively compute the set \( \llbracket \phi \rrbracket \) of its base pairs such that \( \phi \) is satisfiable iff for some \((V, \Pi) \in \llbracket \phi \rrbracket \), \( \Pi \) is satisfiable.
Definition 9.3 We give a decision algorithm to check the satisfiability of given
\[ Y \uparrow \land \Pi \land \forall i \in I \, (X_i \downarrow \land P_i(\overrightarrow{t}_i)) \]

Step 1. By using the Lemma 9.1 to eliminate \( \downarrow \) keeping equivalence, we transform the goal into
\[ Y \uparrow \land \Pi \land \forall i \in I \, \hat{P}_i(\overrightarrow{t}_i, \overrightarrow{\varphi}_i) \]
where \( \overrightarrow{\varphi}_i \) is a sequence of elements in \( X_i \).

Step 2. For each \( i \in I \), we compute \( [\hat{P}_i(\overrightarrow{t}_i, \overrightarrow{\varphi}_i)] \) by using the algorithm by [10], and let it be \( B_i \).

Step 3. Try to find some \( (V_i, \Pi_i) \) in \( B_i \) for each \( i \in I \) such that \( V_i \) and \( Y \) are disjoint under the equality of \( \Pi \), and \( \otimes(\bigcup_{i \in I} V_i) \land \Pi \land \bigwedge_{i \in I} \Pi_i \) is satisfiable.

Step 4. If there are such \( (V_i, \Pi_i) \) \( (i \in I) \), answer with ”satisfiable”. Otherwise, answer with ”unsatisfiable”.

10 Decision Algorithm

This section gives the algorithm to decide the validity of a given entailment. First define normal forms and groups, then define the algorithm, finally we will show the partial correctness, loop invariants, and the termination of the algorithm.

10.1 Normal Form and Group

This section defines a normal form and a group. In our proof search algorithm, a normal form is used as a bud in cyclic proofs and the termination will be proved by counting normal forms. Groups are used for the (*) rules to keep validity.

Definition 10.1 (Groups) A variable group of \( \psi \) is defined to be the set \( \text{(Roots+Cells)}(\Delta) \) of variables for some \( \Delta \) in \( \psi \). A variable group of an entailment \( J \) is defined to be a variable group of the antecedent in \( J \). A formula \( F \) in an entailment \( J \) is called a group when \( \text{(Roots+Cells)}(F) \) is a variable group of \( J \). In particular, the \( * \)-conject \( \land y_1 \downarrow \land \ldots \land y_n \downarrow \) in the antecedent is a group. A formula \( *F_i \) in an entailment \( J \) is called grouping when each \( F_i \) is a group of \( J \).

\( (\Gamma_1, \Gamma_2) \) is called group-disjoint if \( \text{(Roots+Cells)}(\Gamma_1) \cap \text{(Roots+Cells)}(\Gamma_2) = \emptyset \).

\( (\Phi_1, \Phi_2) \) is called a group split by \( (\Gamma_1, \Gamma_2) \) if \( \text{(Roots+Cells)}(\Gamma_i) = \text{(Roots+Cells)}(\Phi_i) \) for \( i = 1, 2 \).

A variable group is the set of address variables that belong to some single \( \Delta \) in the antecedent. A group is a \( * \)-conjunction of formulas whose address variables are in a single group. \( (\Gamma_1, \Gamma_2) \) is group-disjoint when their address variables are disjoint. \( (\Phi_1, \Phi_2) \) is a group split by \( (\Gamma_1, \Gamma_2) \) when they are split by their address variables according to variable groups of \( \Gamma_1, \Gamma_2 \).

For a set \( V \) of variables, we define \( |P_1(t_1, \overrightarrow{t}_1), \ldots, P_n(t_n, \overrightarrow{t}_n)|_V = |\{t_1, \ldots, t_n\} - V| \).

Definition 10.2 (Normal Form) For a given set \( V_0 \) of variables and a given number \( d \), an entailment \( J \) is called normal with \( (V_0, d) \) if \( J \) is of the form \( Y \uparrow \land \Pi \land \bigwedge_i \{ \Phi_i \mid i \in I \} \) and \( \Phi_i \) is of the form \( \exists \overrightarrow{\varphi}_i \exists \overrightarrow{\psi}_i \upharpoonright (\Pi_i \land \Gamma_i) \) and by letting \( V \) be \( \text{BV}(J) \),
1. \( \Gamma \) is a single group (single group condition),
2. \( Y + \text{(Roots+Cells)}(\Gamma) = V \) (variable condition),
3. \( \text{Roots}(\Phi_i) \) is defined (disjunct root condition),
4. \( \text{(Roots+Cells)}(\Gamma) = \text{(Roots+Cells)}(\Phi_i) \) for every \( i \in I \) (group condition),
5. if \( P(x, \overrightarrow{t}) \land y \downarrow \) in \( \Phi_i \), then \( y \in \overrightarrow{t} \) (disjunct definedness condition),
6. \( \overrightarrow{\varphi}_i \subseteq \text{Cells}(\Gamma_i) \) (disjunct existential condition),
7. \( \overrightarrow{\psi}_i \subseteq \text{BV}(\Gamma_i) \) (unrelated existential condition),
8. \( \Pi \) is \( \neq \text{(V)} \) (equality condition),
9. \( \Pi_i \) is \( \neq (\overrightarrow{\varphi}_i, V + \{\overrightarrow{\psi}_i, \text{nil}\}) \) (disjunct equality condition),
10. if \( i \neq j \), then \( \Phi_i \neq \Phi_j \theta \) for all variable renaming \( \theta \) such that \( \text{Dom}(\theta) \cap \text{FV}(\Gamma) = \emptyset \) (disjunct renaming condition),
11. \( \text{FV}(Y, \Pi) \subseteq \text{FV}(\Gamma, \Phi_i) \) (antecedent variable condition),
12. \( |\overrightarrow{Q}|_{V_0} \leq d \) for every predicate symbol \( \overrightarrow{Q} \land *P \) in \( J \) (wand condition).
10.2 Definition of Algorithm

For a given entailment \( A \vdash \overline{B} \), our algorithm calls the function \( \text{MainLoop}(A \vdash \overline{B}) \) to decide whether \( A \vdash \overline{B} \) is valid or not. If it returns Yes, then \( A \vdash \overline{B} \) is valid. If it returns No, then \( A \vdash \overline{B} \) is not valid. The function \( \text{Main} \) calls \( \text{MainLoop} \), which may fork, namely, it copies itself and produces new processes. The function \( \text{Main} \) waits all these processes to terminate or fail.

Let \( k_{\text{max}} \) be the maximum arity for predicate symbols in the original language.

For sets \( T, T' \) of terms, we define \( (= \neq (T, T')) \) as the set of conjunctions of all combinations of \( t = u \) or \( t \neq u \) for \( t \in T, u \in T' \). We define \( (= \neq (T)) \) as \( (= \neq (T \cup \{\text{nil}\}, T \cup \{\text{nil}\})) \). We write \( (\text{Dom} \cup \text{Range})(f) \) for \( \text{Dom}(f) \cup \text{Range}(f) \).

We say we apply a rule to a computation step when the input to the step is the conclusion of the rule and the output of the step are the assumptions of the rule.

All the functions are defined below. We describe functions in codes and then describe the same functions again in English for explanation. The algorithm of satisfiability check is given in Section 9.

```
function MainLoop(A \vdash \overline{B})
  let \( J \) be \( A \vdash \overline{B} \).
  let \( V \) be \( \text{FV}(J) \).
  let \( \Pi \) be \( \text{Dom}(J) \).
  \( \Sigma_x \) for each \( x \in \Pi \).
  let \( K := \{ (Y, \{x_i\}) : Y \cup \Sigma_x \} \).
  for each \( (Y, \{x_i\}) \) in \( K \)
    add \( G := (Y \uparrow \Pi \land \Pi U \leadsto \overline{B}) \).
  let \( (B_j) \) be \( \overline{B} \).
  let \( X \) be \( \text{Roots}(A) \).
  for each \( (X_j, x_i) \) in \( X \)
    let \( \psi \) be \( \psi \in \psi \).
    \( J := \psi \).
    \( \psi \vdash \psi \).
  end for
end function
```

```
function MainLoop(J, V, d)
  /* J an entailment to prove, d the maximum depth of wands. V the set of cell variables. */
  \( S := \{ (J, \emptyset) \} \).
  while \( S \neq \emptyset \)
    /* \( S \) a set of pairs of a subgoal \( J \) and a history \( H \). \( J \) and elements in \( H \) are normal forms except the current entailment. */
    for each \( (J, H) \) in \( S \)
      copy the current process and continue the computation
      the current process becomes \( |S| \) processes by fork.
      \( S := S \) the fork \( \emptyset \) \( J \).
      if there are some \( J' \in H \) and \( \theta \) such that \( J' \theta \equiv J \) then continue
    end for
end for
return Yes
end function
```
\[ H := H + \{ J \} \]
\[
\text{let } \psi \vdash G \text{ be } J. \\
\text{if } \text{Roots}(\psi) \cap \text{Roots}(\Phi_i) = \emptyset \text{ then} \\
\quad \text{Choose } y \in \text{Roots}(\psi). \quad \text{/* There may be choices only for the first loop */} \\
\text{let } \psi \vdash G, \Phi[P(\overline{t})] \land y \downarrow \text{ be } J. \\
J := \text{Factor}(V, d, (\psi \vdash G, \Phi[ ], P(\overline{t})), y).
\]
end if
\[ S' := \text{Unfold}(J). \]
\text{for each } J \text{ in } S' \text{ do} \\
\quad /* \text{Case analysis } */ \text{ let } \psi \vdash G \text{ be } J. \quad S'' := \{ \Pi \land \psi \vdash \overline{t} \mid \Pi \in (\neq \text{FV}(J)) \}.
\text{for each } J \text{ in } S'' \text{ do} \\
\quad /* \text{Unsat check } */ \text{ let } \psi \vdash G \text{ be } J. \quad \text{if } \psi \text{ unsat then continue} \\
\quad J := \text{Match}(J) \\
\quad /* \text{In emp case. Terminiation Check } */ \\
\quad \text{if } \psi \equiv Y \uparrow \land \Pi \land \text{emp then} \\
\quad \text{if some } \exists \overline{v} \uparrow (\Pi' \land \text{emp}) \in \overline{t} \text{ then continue } /* \Pi' \text{ has only } w \neq t */ \\
\quad \text{fail} \\
\text{end if} \\
G := \text{Split}(J, V, d) \\
\text{For each } S'' \in G, \text{ copy the current process and continue the computation (namely, the current process becomes } |G| \text{ processes by fork).} \\
\text{for each } J \text{ in } S'' \text{ do} \\
\quad J := \text{Normalize}(J) \\
\quad S := S \cup \{(J, H)\}. \\
\text{end for} \\
\text{end for}
\]
end while

function \text{Factor}(V, d, J[ ], P'(\overline{t})), y)
\text{/* } J[P'(\overline{t}) \land y \downarrow], P' \text{ is in the extended language. */} \\
\text{/* (1) Factor */} \\
\text{let } F \vdash G, G[ ] \text{ be } J[ ]. \\
\text{let } Q(\overline{t}) \to P(\overline{t}) \text{ be } P'(\overline{t}) \text{ where the predicate symbols } Q, P \text{ are in the original language.} \\
\text{Apply the following rule (BoundedFactor) to } J. \\
\frac{x \neq y \land F \vdash G, \{G \exists \overline{w} ((Q_1(\overline{t}_1), Q(y,\overline{w}) \to P(\overline{t})) \to (Q_2(\overline{t}_2) \to P(\overline{t}))) \mid \{Q_1(\overline{t}_1), Q_2(\overline{t}_2)\} = \{Q(\overline{t})\}, Q \in \text{Dep}(P), Q_2 \subseteq \text{Dep}(Q), |Q_1(\overline{t}_1), Q(y,\overline{w})|_V \leq d, |Q_2(\overline{t}_2)|_V \leq d, \overline{w} \text{ fresh}\}}{x \neq y \land F \vdash G, \{Q(\overline{t}) \to P(\overline{t})\} \land y \downarrow} \text{ (BoundedFactor)}
\]
\text{/* (2) Name Case Analysis */} \\
\text{let } \psi \vdash F'[\exists w \overline{w}((Q_1(\overline{t}_1), Q(y,\overline{w}_1, w\overline{w}_2) \to P(\overline{t})) \to (Q_2(\overline{t}_2) \to P(\overline{t})))], \overline{t} \text{ be } J. \\
\text{let } V \text{ be } \text{FV}(J). \\
\text{let } F[ ] \text{ be } (Q_1(\overline{t}_1), Q(y,\overline{w}_1), \mid \overline{w}_2) \to P(\overline{t})). \\
\text{let } G[ ] \text{ be } (Q_2(\overline{t}_2) \to \exists w \overline{w}((F[\overline{t}] \to G[\overline{t}])) \mid t \in V \cup \{\overline{w}\}). \\
\text{Do this for each } \exists \overline{w} \text{ produced by the previous step.} \\
\text{return } J
\]
end function

function \text{Unfold}(J)
\text{/* (1) Unfold L and R */} \\
\text{let } \psi \vdash G \text{ be } J. \\
\text{Choose } x \in \text{Roots}(\psi) \cap \bigcap \text{Roots}(F_i). \\
\text{S'} := \{ \psi' \ast (A(x, \overline{t}, \overline{t})) \land X \downarrow \} \ast \{F_i[\phi_i(x, \overline{t}_i)] \mid F_i[\overline{P}(x, \overline{t}_i)] \in \overline{t}, \phi_i(x, \overline{t}_i) \text{ is a definition clause of } P_i(x, \overline{t}_i) \text{ if } P \text{ is } P_i, \phi_i(x, \overline{t}_i) \text{ is } P_i(x, \overline{t}_i) \text{ if } P_i \text{ is } \rightarrow \} \\
\quad \text{where } \exists \overline{s} A(x, \overline{t}, \overline{t}) \text{ is a definition clause of } P(x, \overline{t}) \text{ if } P \text{ is } P, A(x, \overline{t}, \overline{t}) \text{ is } P(x, \overline{t}) \text{ if } P \text{ is } \rightarrow, \overline{s} \text{ fresh}. \\
\text{/* (2) Left Definedness Distribution */}
while \( \psi \ast ((\Pi \land \ast \in X \Gamma(x_i)) \land X \downarrow) \vdash \bar{F} \) is in \( S' \) and \(|I| > 1 \) and \( X \neq \emptyset \) do
\[
S' := S' - \{ \psi \ast ((\Pi \land \ast \in X \Gamma(x_i)) \land X \downarrow) \vdash \bar{F} \} + \{ \psi \land \Pi \ast \in (\Pi \land \ast \in X \Gamma(x_i) \land X \downarrow) \vdash \bar{F} \mid \Sigma_i(X_i + \{x_i\}) = X \}.
\]
end while
return \( S' \)
end function

function Match(\( J \))
/* (1) Equality Elimination */
let \( x = t \land X \uparrow \land \Pi \land \Gamma \vdash \bar{F} \) be \( J \).
\( J := ((X[x := t] - \{nil\}) \uparrow \land \Pi[x := t] \land \Gamma[x := t]) \vdash \bar{F}[x := t] \)
Repeat this until the antecedent does not contain \( \ast \).
/* (2) Match */
let \( \psi \ast x \mapsto (\bar{\varphi}) \vdash \bar{F}, \exists \bar{\varphi}(F \ast x \mapsto (\bar{\varphi})) \) be \( J \).
\( J := (\psi \ast x \mapsto (\bar{\varphi}) \vdash \bar{F}, \exists \bar{\varphi}(F \ast x \mapsto (\bar{\varphi}) \land \varphi = \bar{\varphi})) \).
Do this for each \( x \mapsto (\bar{\varphi}) \) in the succedent.
/* (3) Existential Instantiation */
let \( \psi \vdash \bar{F}, \exists \bar{\varphi} z = t \land F \) be \( J \).
\( J := (\psi \vdash \bar{F}, \exists \bar{\varphi} F[z := t]) \).
Repeat this until this cannot apply anymore.
/* (4) Unmatch Disjunct Elimination */
let \( \psi \vdash \bar{F}, t = u \land F \) be \( J \).
if \( t \equiv u \) then \( J := (\psi \vdash \bar{F}, F) \)
else \( J := (\psi \vdash \bar{F}) \)
Do this for each \( = \) in the succedent.
/* (5) Disequality Elimination */
let \( \psi \vdash \bar{F}, t \neq u \land F \) be \( J \).
if \( t \equiv u \) then \( J := (\psi \vdash \bar{F}) \)
else \( J := (\psi \vdash \bar{F}, F) \)
/* (6) Emp Disjunct Elimination */
let \( \psi \vdash \bar{F}, F \) be \( J \).
if \( F \) is \( \exists \bar{\varphi} \exists \bar{\varphi} \uparrow (\Pi \land \emp \land X \downarrow) \) and \( X \neq \emptyset \) then \( J := (\psi \vdash \bar{F}) \)
Do this for each disjunct.
/* (7) Unleaf Elimination */
let \( G \vdash \bar{F}, F \) be \( J \).
if \( F \) contains \( x \mapsto (\bar{\varphi}) \land w \uparrow \) and \( w \in \bar{\varphi} \) then \( J := (G \land \Gamma \vdash \bar{F}) \)
Do this for each disjunct.
/* (8) \( \mapsto \) Removal */
let \( \psi \ast x \mapsto (\bar{\varphi}) \vdash \exists \bar{\varphi}(F_i \ast x \mapsto (\bar{\varphi})) \), be \( J \).
\( J := (x \uparrow \land \psi \vdash \exists \bar{\varphi}(F_i)) \).
return \( J \)
end function

function Split(\( J, V, d \))
/* (1) Extra Definedness */
let \( \psi \vdash \bar{F}, F \) be \( J \).
if \( z \in \text{Roots}(\psi) - (\text{Roots} + \text{Cells})(F) \) then \( J := (\psi \vdash \bar{F}, F \land z \downarrow) \).
Repeat this until there is not \( z \in \text{Roots}(\psi) - (\text{Roots} + \text{Cells})(F) \).
while true do
/* (2) Right Definedness Distribution */
let \( \psi \vdash \bar{F}, F[\exists \bar{\varphi} \exists \bar{\varphi} \uparrow (\Pi \land F \ast (\ast \in X \Gamma(x_i)) \land X \downarrow)] \) be \( J \).
\( J := (\psi \vdash \bar{F}, \{ F[\exists \bar{\varphi} \exists \bar{\varphi} \uparrow (\Pi \land F \ast (\ast \in X \Gamma(x_i)) \land X \downarrow)] \mid X = \Sigma_i(x_i) \}) \).
Repeat this until the succedent becomes \( \bar{G} \).
/* (3) Disjunct Grouping */
if \( J \) is \( \psi \vdash \bar{F}, \Phi[P(x, \bar{\varphi}) \land \varphi \downarrow] \) and \( x \) and \( y \) are in different groups then
\( J := \text{Factor}(V, d, (\psi \vdash \bar{F}, \Phi[ ]), P(x, \bar{\varphi}), y) \).
else break
end if
end while
/* (4) Existential Disequality */
/* We have \( \neq \) for every variable except \( \bar{\varphi} \). \( \bar{\varphi} \neq \bar{\varphi} \) from the antecedent. \( \bar{\varphi} \neq \bar{\varphi} \) from \( \bar{\varphi} \uparrow \) and \( \bar{\varphi} \downarrow \). */
let \( \psi \vdash \bar{F}, \exists \bar{\varphi} \exists \bar{\varphi} \uparrow (\Pi \land \ast \in \Gamma_i) \) be \( J \) where \( \ast \in \Gamma_i \) is grouping.
\(J := (\psi \vdash \overline{\Phi}, \exists \overline{x} \exists \overline{y} \uparrow ((\not= (\overline{x} \overline{y}), \text{FV}(\overline{x}) \cup \{\text{nil}\})) \land \Pi \land \star_i \Gamma_i)).\)

/* (5) Unrelatedness Introduction */
let \(\psi \vdash \overline{\Phi}, \exists \overline{x} \exists \overline{y} \uparrow ((\Delta \land x \downarrow) \land \star_i \Gamma_i)\) be \(J\) where \((\Delta \land x \downarrow) \land \star_i \Gamma_i\) is grouping.
\(J := (\psi \vdash \overline{\Phi}, \exists \overline{x} \exists \overline{y} \uparrow (\Pi \land (\Delta \land x \downarrow) \land \star_i (\Gamma_i \land x \uparrow))).\)

Do this for each \(x\) ↓.

/* (6) Existential Split */
let \(\psi \vdash \overline{\Phi}, \exists \overline{x} \exists \overline{y} \uparrow (\Pi \land \star_i \Gamma_i)\) be \(J\) where \(\star_i \Gamma_i\) is grouping.
\(J := (\psi \vdash \overline{\Phi}, \exists \overline{x} \exists \overline{y} \uparrow (\Pi \land \Gamma_i)).\)

/* (7) (∗)-Split */
Repeatedly apply the following rule (Split) to \(J\) until the antecedent becomes a single group in order to generate a set \(G\) of sets of subgoals.

\[
\begin{align*}
(Y \cup Y_i) \uparrow \land \Pi \land \Pi_2 \vdash \{\Phi_2[i \in I - I']\} \lor (Y \cup Y_2) \uparrow \land \Pi \land \Pi_1 \vdash \{\Phi_1[i \in I']\} & \quad (\forall I' \subseteq I) \\
Y \uparrow \land \Pi \land \Pi_1 \land \Pi_2 & \vdash \{\Phi_1 \land \Phi_2[i \in I]\}
\end{align*}
\]

where

\(V = \text{FV}(Y \uparrow \land \Pi \land \Pi_1 \land \Pi_2 \vdash \{\Phi_1 \land \Phi_2[i \in I]\}),\)
\(Y_i = (\text{Roots} + \text{Cells})(\Gamma_i)\) \((i = 1, 2),\)
\(\Pi \equiv (\not= (V)),\)
\((\text{Roots} + \text{Cells})(\Gamma_1 \land \Pi_2) + Y = V,\)

\((\Gamma_1, \Gamma_2)\) is group-disjoint, and \((\Phi_1, \Phi_2)\) \((i \in I)\) is group split by \((\Gamma_1, \Gamma_2).\)
return \(G\)

end function

function Normalize(\(J\))
/* (1) Fresh Variable Disjunct Elimination */
let \(X \uparrow \land \Pi \land \Gamma \vdash \overline{\Phi}, \Phi', \Phi'\) be \(J\).
if \(\theta\) is a variable renaming, \(\Phi' \equiv \Phi\theta\), \((\text{Dom} \cup \text{Range})(\theta) \subseteq X, (\text{Dom} \cup \text{Range})(\theta) \cap \text{FV}(\Gamma) = \emptyset\) then
\(J := (X \uparrow \land \Pi \land \Gamma \vdash \overline{\Phi}, \Phi)\)
end if
Repeat this until it cannot apply.

/* (2) Unnecessary Disequality Elimination */
let \(X \uparrow \land \Pi \land \Gamma \vdash \overline{\Phi}\) be \(J\).
if \(x \not\in \text{FV}(\Gamma, \overline{\Phi})\) then
\(J := (X \uparrow \land \Pi \land \Gamma \vdash \overline{\Phi})\)
Repeat this until it cannot apply.

/* (3) Unnecessary Variable Elimination */
let \(x \uparrow \land \Pi \land \Gamma \vdash \overline{\Phi}\) be \(J\).
if \(x \not\in \text{FV}(\Gamma, \overline{\Phi})\) then
\(J := (X \uparrow \land \Pi \land \Gamma \vdash \overline{\Phi})\)
Repeat this until it cannot apply.

/* (4) Unnecessary Unrelated Existential Elimination */
let \(\psi \vdash \overline{\Phi}, \exists \overline{x} \exists \overline{y} \uparrow (\Pi \land \Gamma)\) be \(J\).
if \(y \not\in \text{FV}(\Gamma)\) then
Remove \(y \not\in \Pi\) to obtain \(\Pi'\).
\(J := (\psi \vdash \overline{\Phi}, \exists \overline{x} \exists \overline{y} \uparrow (\Pi' \land \Gamma))\)
end if
Repeat this until it cannot apply.
return \(J\)
end function

For explanations of these functions we give their English descriptions except Mainloop below.

function Main(\(A \vdash \overline{B}\))
1. In the antecedent \(A\), consider cases by classifying all the free variables except the roots into \(Y\uparrow\) and \(X_i\downarrow\) for each atomic spatial formula \(P_i(x_i)\), and in each case, add \(Y \uparrow \land\) and \(X_i \downarrow \land\) to the \(\star\)-conjunct \(P_i(x_i)\), and do the following.
2. In each conjunct of the succedent \(\overline{B}\), consider cases by classifying the address variables of the antecedent into \(X_i\) and \(x_i\) for each atomic spatial formula \(Q_i(x_i)\), and in each case, generate a new conjunct by adding \(X_i \downarrow \land\) to the \(\star\)-conjunct \(Q_i(x_i)\), and do the following.
3. Consider cases by = and \(\not=\) for all the variables and nil, and in each case, add \(\Pi_1 \land\) to the antecedent where \(\Pi_1\) is the conjunction of these = and \(\not=\), and do the following.
4. If the antecedent is unsatisfiable, finish this case.
5. If the spatial part of the antecedent is empty, do the following. If there is some disjunction whose spatial part is empty and pure part is included in the pure part of the antecedent, finish this case. Otherwise stop with fail.

6. Let $J$ be the entailment. Let $V$ be the union of Cells for each disjunct in the succedent. Let $d_{\text{wand}}$ be $k_{\text{max}}$. Call $\text{MainLoop}(\{J, V, d_{\text{wand}}\})$. $\text{MainLoop}$ may fork and process several processes. Wait either of the following cases happens: If some process terminates without fail, finish this case; If every process fails, stop with No.

7. If all the cases are finished, stop with Yes.

\textbf{end function}

\textbf{function} Factor($V, d, J, )$, $P'((\overline{t}), y)$

1. Factor. Let $J$ be $F \vdash \overline{G}, G$. Let $P'((\overline{t}))$ be $\overline{Q((\overline{t}))} \leftarrow s*P((\overline{t}))$ where the predicate symbols $\overline{Q}, P$ are in the original language. Apply the following rule (BoundedFactor) to $J$.

\[
x \neq y \land F \vdash \overline{G}, G[\exists \overline{w}((\overline{Q_1(\overline{t}_1)}, Q(y, \overline{w}) \leftarrow s*P(x, \overline{t})) * (\overline{Q_2(\overline{t}_2)}) \leftarrow s*Q(y, \overline{w}))]) \mid \\
\{Q_1(\overline{t}_1), Q_2(\overline{t}_2)\} = \{\overline{Q(\overline{t})}\}, Q \in \text{Dep}(P), \overline{Q_2(\overline{t})} \subseteq \text{Dep}(Q), \\
|Q_1(\overline{t}_1), Q(y, \overline{w})|_V \leq d, |Q_2(\overline{t}_2)|_V \leq d, \overline{w} \text{ fresh}\]

\[
x \neq y \land F \vdash \overline{G}, G[(\overline{Q(\overline{t})} \leftarrow s*P(x, \overline{t})) \land y \downarrow]
\] (BoundedFactor)

2. Name Case Analysis. For each $\exists w$ generated in the step 1, do the following. Let $V_1$ be the union of the set of free variables in $J$ and the set of bound variables whose scopes include this $\exists w$. Generate disjuncts by classifying cases by $=$ and $\neq$ between each element in $V_1 \cup \{\text{nil}\}$ and $w$. If a disjunct has $w = t$, eliminate $\exists w$ by substituting $w := t$ in the disjunct. If a disjunct has $w \neq V_1 \cup \{\text{nil}\}$, add $w \downarrow \land$ to $\overline{Q_1(\overline{t}_1)}, Q(y, \overline{w}) \leftarrow s*P(\overline{t})$ in the disjunct.

3. Return the entailment.

\textbf{end function}

\textbf{function} Unfold($J$)

1. Unfold L and R. In the entailment $J$, choose a common root $x$ among the antecedent and all disjuncts of the succedent.

Generate a set of subgoals by replacing $P(x)$ in the antecedent by formulas obtained from the definition clauses of $P(x)$ by removing $\exists \overline{w}$ and replacing $\overline{w}$ by fresh variables, if $P(x)$ is in the antecedent. In each entailment in the subgoal set, replace $Q(x)$ by each definition clause of $Q(x)$ to generate new disjuncts, if $Q(x)$ in the disjunct.

2. Left Definedness Distribution. In the antecedent of each subgoal entailment $J$ in the subgoal set, consider cases of distributing $\land x \downarrow$ to spatial atomic formulas, and replace $J$ by new subgoals generated from these cases.

3. Return the subgoal set.

\textbf{end function}

\textbf{function} Match($J$)

1. Equality Elimination. In the entailment $J$, if the antecedent has $x = t$, eliminate it by substitution $x := t$.

2. Match. If the antecedent has $x \mapsto (\overline{w})$ and a disjunct in the succedent has $x \mapsto (\overline{v})$, add $\overline{v} = \overline{w} \land$ to $x \mapsto (\overline{v})$.

3. Existential Instantiation. If the succedent has $\exists z(z = t \land \ldots)$, eliminate $\exists z$ by substitution $z := t$.

4. Unmatch Disjunct Elimination. If the succedent has $t = t$, remove it. If a disjunct of the succedent has $t = u$ and $t \neq u$, remove the disjunct.

5. Disequality Elimination. If a disjunct of the succedent has $t \neq t$, remove the disjunct. If the succedent has $t \neq u$ and $t \neq u$, remove $t \neq u$.

6. Emp Disjunct Elimination. If a disjunct of the succedent has $\text{emp} \land t \downarrow$, remove this disjunct.

7. Unleaf Elimination. If a disjunct of the succedent has $x \mapsto (\overline{u}) \land u_i \uparrow$ and $u_i \in \overline{v}$, remove the disjunct.

8. $\mapsto$ Removal. If the antecedent and all the disjuncts of the succedent have $x \mapsto (\downarrow)$, remove these $x \mapsto (\downarrow)$ and add $x \uparrow \land$ to the antecedent.

9. Return the entailment.

\textbf{end function}
function Split(J, V, d)
    1. Extra Definedness. For each root z of the antecedent and each disjunct F of the succedent, if
       z ∉ (Roots + Cells)(F), add z ↓ ∧ to F. Repeat this until there are not such z and F.
    2. Right Definedness Distribution. In each disjunct in the succedent of the entailment J, consider
       cases of distributing ∧ x ↓ to spatial atomic formulas, and replace the original disjunct by all the disjuncts
       generated from these cases.
    3. Disjunct Grouping. If some disjunct has P(x, t) ∧ y ↓ and x and y are in different groups, by
       letting the entailment be ψ ⊢征服, Φ[P(x, t)], call Factor(V, d, (ψ ⊢征服, Φ[]), P(x, t), y). Then go to the
       step 2.
    4. Existential Disequality. In each disjunct, add w ≠ z ∧ to the Φ-body for each w, z such that the
       disjunct has ∃w and z is a fresh variable introduced by the function Unfold.
    5. Unrelatedness Introduction. Replace each disjunct ∃(Π ∧ ∃x ↓)∗,φ(Φ ↓) by ∃(Π ∧ (φ ∧ ∆) ↓)∗,φ(Φ ↓).
    6. Existential Split. Replace each disjunct ∃(Π ∧ ∃1 ∧ Y ↓) by ∃(Π ∧ ∃1 ∧ Y ↓).
    7. (*)-Split. Apply the following rule (Split) repeatedly to generate the subgoal set G until the spatial
       part of the antecedent in every entailment becomes atomic.

       (Y ∪ Y1) ↑ ∧ ∆ ∧ Γ2 ⊢ {F2i| i ∈ I − I’} or (Y ∪ Y2) ↑ ∧ ∆ ∧ Γ1 ⊢ {F1i| i ∈ I’} (∀I’ ⊆ I)

       (Split)

       where

       V = FV(Y ↑ ∧ ∆ ∧ Γ1 + Γ2 ⊢ {F1i * F2i| i ∈ I}),
       Y1 = (Roots + Cells)(Γ1) (i = 1, 2),
       Y2 = (Roots + Cells)(Γ1 + Γ2) + Y = V,
       (Γ1, Γ2) is group-disjoint, and (F1i, F2i) (i ∈ I) is group split by (Γ1, Γ2).
    8. Return G.
end function

function Normalize(J)
    1. Fresh Variable Disjunct Elimination. If the entailment J has the form X ↑ ∧ ∆ ∧ Γ ⊢ ψ, Φ, Φ’ and
       there is a variable renaming θ such that Φ’ ≡ Φθ, (Dom ∪ Range)(θ) ⊆ X, and (Dom ∪ Range)(θ) ∩ FV(Γ) =
       ∅, then remove Φ’.
    2. Unnecessary Disequality Elimination. If x does not occur in neither the spatial part and ↓ of the
       antecedent or the succedent, then remove x ≠ t in the antecedent.
    3. Unnecessary Variable Elimination. If x does not occur in neither the spatial part and ↓ of the
       antecedent or the succedent, then remove x ↑ in the antecedent.
    4. Unnecessary Unrelated Existential Elimination. If y does not occur in the spatial part in a disjunct
       with ∃y ↑, remove ∃y ↑ and y ≠ t in the disjunct.
    5. Return the entailment.
end function

When an entailment A ⊢征服 is given to Main, we generate subgoals by putting ↓ and ↑ of all the free
variables to the antecedent with all the cases, putting ↓ of all the address variables in the antecedent to
the succedent with all the cases, putting = and ≠ of all the variables with nil to the antecedent with all cases,
check satisfiability, and send the subgoal set to the function MainLoop. MainLoop may fork and
produce several processes. If for some subgoal every process of MainLoop fails, the function Main returns
No. If for all subgoals some process of MainLoop terminates without fail, the function Main returns Yes.

MainLoop first checks whether the current subgoal appeared already. If so, we finish this subgoal and
go to the next subgoal. Secondly MainLoop calls Factor to create common roots on both sides when there
is no common root. In order to find a common root such that the bounded factor rule for it is locally
complete, the function MainLoop forks and produces new processes with each common root. When there
is a common root x, MainLoop first unfolds the predicate of form P(x) on both sides (the function Unfold).
Next it forces match of them on both sides by putting equalities on the succedent and removes
x ↔ (x) on both sides (the function Match). Then MainLoop checks termination condition when the
antecedent is emp. Then MainLoop splits 3-scopes. Then we split separating conjunctions until every
entailment become a single group (the function Split). For each rule instance of (Split), MainLoop forks
and produces new processes to compute this case, and executes these processes nondeterministically. Then MainLoop transforms subgoals into normal forms and goes to the first step of the function MainLoop. The function MainLoop does this loop until the subgoal set becomes empty.

### 10.3 Partial Correctness of Algorithm

We can show the correctness of the algorithm when it terminates with Yes. We will show the case with No in the completeness proof later.

**Lemma 10.3 (Partial Correctness)** If the algorithm terminates with Yes, then the input entailment is provable.

**Proof.** When the algorithm terminates with Yes, for each subgoal there is some process that terminates without fail.

For the input goal, the function Main generates subgoals for it. For each subgoal, the function MainLoop constructs its proof when it terminates without fail. In the while loop of the function MainLoop, for each subgoal \((J, H) \in S\), the function MainLoop proves \(J\) by using \(H\) as companions.

We can show that each step of the algorithm consists of applications of inference rules as follows. We consider each step of each function.

**Function Main.**

Case analysis by \((Y, (X_i)_i))\). By the rules (Case \(L\)) and (\(↓\) Out \(L\)).

Case analysis by \(K''\). By (\(∧\)Elim) and (\(↓\) Out).

Case analysis by \(K'\): By rule (Case \(L\)).

Unsatisfiability check. By (Unsat).

Termination check: By (emp), (\(∧\)L), (\(∧\)R), (\(∨\)R).

By MainLoop, each subgoal \(J\) is provable.

**Function MainLoop.**

If MainLoop terminates without fail for the input \(\{(J, H)\}\), \(J\) is provable by using \(H\) as companions.

We use rule (Subst) from \(J'\) to \(J\) and discharge \(J\) as a bud with the companion \(J' \in H\). Since \(H\) is the set of judgments from the root to the previous subgoal, the step for \(J' \in H\) and \(J'\theta = J\) in MainLoop gives a bud-companion relation.

Case Analysis: By rules (Case \(L\)).

Unsat Check: By rule (Unsat).

Termination Check: By (emp), (\(↑\) R), (\(∧\)L), (\(∨\)R).

**Function Factor.** We show the input formula is derived from the output formula.

(1) Factor. By Lemma 5.2.

(2) Name Case Analysis. By the rules (\(∧\)Elim) and (\(∃\)R).

**Function Unfold.** We show that \(J\) is derivable from \(S'\).

(1) Unfold L and R. By the rules (Pred \(L\)) and (Pred \(R\)).

(2) Left Definedness Distribution. By (\(↓\) Out \(L\)).

**Function Match.** We show the input formula is derived from the output formula.

(1) Equality Elimination. By the rule (\(=\) L).

(2) Match. By the rule (\(∧\)Elim).

(3) Existential Instantiation. By the rules (\(=\) R) and (\(∃\)R).

(4) Unmatch Disjunct Elimination. By the rules (\(=\) R) and (\(∨\)R).

(5) Disequality Elimination. By (\(∧\)R), (\(∨\)R).

(6) Emp Disjunct Elimination. By the rule (\(∨\)R).

(7) Unleaf Elimination. By the rule (\(∨\)R).

(8) \(\mapsto\) Removal. By the rule (* \(\mapsto\)).

**Function Split.** We show that \(J\) is derived from any subgoal set in \(S\).

(1) Extra Definedness. By (\(↓\) R).

(2) Right Definedness Distribution. By the rule (\(↓\) Out \(R\)).

(3) Disjunct Grouping. By Factor.

(4) Existential Disequality. By the rule (\(∧\)Elim).

(5) Unrelatedness Introduction. By the rule (\(∧\)Elim).

(6) Existential Split. Use (\(∃\)Amalg2) for \(\overline{\not{\exists}}\) and (\(∃\)Amalg1) for \(\overline{\not{∃}}\) \(↑\).
(7) (*)-Split. The rule (Split) is derivable from the rule (*) and († Elim).

Function Normalization. We show that the input formula is derived from the output formula.

1. Fresh Variable Disjunct Elimination. By the rule (∨R).
2. Unnecessary Disequality Elimination. By the rule (∧L).
3. Unnecessary Variable Elimination. By the rule (∨L).
4. Unnecessary Unrelated Existential Elimination. By (⇑ R).

We have shown that each step of the algorithm consists of applications of inference rules. Hence the process produces a proof of the input entailment, when it terminates without fail. Hence the function Main produces the proof when it returns Yes. □

10.4 Loop Invariant

We will use the following loop invariants of the function MainLoop.

Lemma 10.4 Let \( V, d_{\text{wand}} \) be those in the call of the function MainLoop. At the beginning of the while loop in the function MainLoop, every entailment \((J, H) \in S\) satisfies the following.

1. \( J \) is a normal form with \((V, d_{\text{wand}})\) except the single group condition, the disjunct equality condition, and the disjunct renaming condition.
2. \( J \) is a normal form with \((V, d_{\text{wand}})\) after the first loop.
3. The antecedent of \( J \) is satisfiable.
4. \( \text{Cells}(\Gamma) \subseteq \text{Cells}(\Gamma') \) where \( \Gamma \) is the spatial part of antecedent in \( J \) and \( \Gamma' \) is that in the initial goal given to the function MainLoop.

Proof.
1 and 2.

The single group condition. By the function Split.
The variable condition. This holds for the initial entailment. New variables are \( \nabla \) in the function Unfold, and they are in the roots.
The disjunct root condition is trivial.
The group condition. \( J \) sent to MainLoop satisfies this condition. New variables \( \exists \) introduced by Unfold is added to the succedent by the extra definedness step in Split.
The disjunct definedness condition. By the name case analysis step of Factor.
The disjunct existential condition. By the name case analysis step of Factor.
The unrelated existential condition. By the unrelatedness introduction step of Split.
The equality condition. By case analysis in the function Main.
The disjunct equality condition. By the existential disequality step of Split.
The disjunct renaming condition and the antecedent variable condition. By the function Normalize.
The wand condition. By the function Factor.
3. By the unsatisfiability check in the functions Main and MainLoop.
4. The algorithm does not increase \( \text{Cells}(\Gamma) \). □

10.5 Termination

This subsection shows the termination of the algorithm. First we show the finiteness of the set of normal forms possibly used in the algorithm. Then we show the termination by using the finiteness.

Since a normal form during the loop consists of a single group, the number of normal forms up to variable renaming is proved to be finite.

Lemma 10.5 The set of normal forms with \((V_0, d)\) up to variable renaming is finite.

Proof.
\( d \) is the maximum depth of wands. Let \( J \) be the entailment sent to MainLoop from Main, \( V \) be \( \text{FV}(J) \), \( Y \uparrow \land \Pi \land \Gamma \) be the antecedent of \( J \), \( (\Phi_i)_{i \in I} \) be the succedent of \( J \), \( c_1 \) be \( |\text{Cells}(\Gamma)| \), \( k_{\text{max}} \) be the maximum arity of original inductive predicates and the \( \mapsto \) predicate, \( c_4 \) be the number of original inductive predicates and the \( \mapsto \) predicate.
Proposition 10.7 The algorithm is nondeterministic double exponential time.

Proof. By the upper bound given by the proof of Lemma 10.5. □

11 Constant Store Validity

Definition 11.1 For a bijection β : Locs \rightarrow Locs, we define

\[ \beta(s) = \beta \circ s, \]
\[ \beta(h) = \beta \circ h \circ \beta^{-1}. \]

Lemma 11.2 For a formula F in the extended language,

\[ s, h \models F \]

and β : Locs \rightarrow Locs is a bijection, then

\[ \beta(s), \beta(h) \models F. \]

Proof. First we show:

Claim 1: the claim holds when F does not contain inductive predicates.

We can show the claim 1 by induction on F. Every case is straightforward. We show only the case ¬G.

Case ¬G.
Assume \( s, h \models \neg G \) in order to show \( \beta(s), \beta(h) \models \neg G \). Then \( s, h \nmodels G \). Hence \( \beta^{-1}(\beta(s)), \beta^{-1}(\beta(h)) \nmodels G \). By IH, \( \beta(s), \beta(h) \nmodels G \). Hence \( \beta(s), \beta(h) \models \neg G \).

We have proved the claim 1.

Next we show the claim of the lemma by induction on \( F \). We show only the case of an inductive predicate. The other cases are proved straightforwardly.

Case \( P(\overrightarrow{T}) \).

Assume \( s, h \models P(\overrightarrow{T}) \). We have \( m \) such that \( s, h \models P^{(m)}(\overrightarrow{T}) \). By the claim (1), \( \beta(s), \beta(h) \models P^{(m)}(\overrightarrow{T}) \). Hence \( \beta(s), \beta(h) \models P(\overrightarrow{T}) \). □

**Definition 11.3** \( A \models_s \{ B_i \mid i \in I \} \) is defined by

\[
\forall h(s, h \models A \rightarrow \bigvee_{i \in I} s, h \models B_i).
\]

**Lemma 11.4** Let \( F_i, G_i^j \) be formulas. If

\[
\Pi \supseteq \neg (\neg \text{FV}(\Pi, (F_j, G_j^i)_{i \in I, j = 1, 2}))
\]

\[
\forall s((\Pi \land F_1 \models_s \{ G_1^i \mid i \in I_1 \}) \lor (\Pi \land F_2 \models_s \{ G_2^i \mid i \in I_2 \}))
\]

then

\[
(\Pi \land F_1 \models \{ G_1^i \mid i \in I_1 \}) \lor (\Pi \land F_2 \models \{ G_2^i \mid i \in I_2 \})
\]

**Proof.** Let \( \overrightarrow{y} = \text{FV}((F_j, G_j^i)_{i \in I, j = 1, 2}) \).

Assume

\[
(\Pi \land F_1 \models \{ G_1^i \mid i \in I_1 \}) \lor (\Pi \land F_2 \models \{ G_2^i \mid i \in I_2 \})
\]

does not hold. Then we have \( s_1, h_1, s_2, h_2 \) such that

\[
s_1, h_1 \models \Pi \land F_1,
\]

\[
s_2, h_2 \not\models \Pi \land F_2,
\]

\[
s_1, h_1 \not\models \{ G_1^i \mid i \in I_1 \},
\]

\[
s_2, h_2 \not\models \{ G_2^i \mid i \in I_2 \}.
\]

Let \( s_1(\overrightarrow{y}) = \overrightarrow{a} \) and \( s_2(\overrightarrow{y}) = \overrightarrow{b} \). Since \( s_1 \models \Pi \) and \( s_2 \models \Pi \), we have \( \overrightarrow{a} \neq \text{nil} \) and \( \overrightarrow{b} \neq \text{nil} \). \( \overrightarrow{a} \) and \( \overrightarrow{b} \) are in Locs and distinct to each other, and \( \overrightarrow{y} \) are in Locs and distinct to each other. Take some bijection \( \beta : \text{Locs} \rightarrow \text{Locs} \) such that \( \beta(a_i) = b_i \). Then \( s_2 = \overrightarrow{y} \beta(s_1) \). By Lemma 11.2

\[
\beta^{-1}(s_2), \beta^{-1}(h_2) \models \Pi \land F_2,
\]

\[
\beta^{-1}(s_2), \beta^{-1}(h_2) \not\models \{ G_2^i \mid i \in I_2 \}.
\]

Hence

\[
s_1, \beta^{-1}(h_2) \models \Pi \land F_2,
\]

\[
s_1, \beta^{-1}(h_2) \not\models \{ G_2^i \mid i \in I_2 \}.
\]

By taking \( s \) to be \( s_1 \) in the assumption,

\[
(\Pi \land F_1 \models_{s_1} \{ G_1^i \mid i \in I_1 \}) \lor (\Pi \land F_2 \models_{s_1} \{ G_2^i \mid i \in I_2 \})
\]

**Case 1.** \( \Pi \land F_1 \models_{s_1} \{ G_1^i \mid i \in I_1 \} \). Since

\[
s_1, h_1 \models \Pi \land F_1,
\]

\[
s_1, h_1 \not\models \{ G_1^i \mid i \in I_1 \}
\]

we have contradiction.

**Case 2.** \( \Pi \land F_2 \models_{s_1} \{ G_2^i \mid i \in I_2 \} \). Since

\[
s_1, \beta^{-1}(h_2) \models \Pi \land F_2,
\]

\[
s_1, \beta^{-1}(h_2) \not\models \{ G_2^i \mid i \in I_2 \}.
\]

we have contradiction.

Since every case leads to contradiction, we have

\[
(\Pi \land F_1 \models \{ G_1^i \mid i \in I_1 \}) \lor (\Pi \land F_2 \models \{ G_2^i \mid i \in I_2 \})
\]

□
12 Cone

Definition 12.1 For a heap $h$ and $a \in \text{Val}$, the heap $h \downarrow a$ is defined as $h|_X$ where $X$ is the least fixed point of

$$F(X) = \{a \cap \text{Dom}(h) \cup \{c \in \text{Dom}(h) | b \in X, b \not\sim c\}.$$  

$b \sim c$ denotes $(h(b))_k = c$ for some $k$. We call $h \downarrow a$ a cone of root $a$ in the heap $h$. For $b \in h \downarrow a$, the depth of $b$ in $h \downarrow a$ is defined as the least number $d$ such that $F^d(\emptyset) \ni b$. We write $b \rightarrow c$ when $b \sim c$ and $c \in \text{Dom}(h)$. We write $\rightarrow$ to the reflexive and transitive closure of $\rightarrow$. We write $b \rightarrow_{ut} c \in h \downarrow a$ when $b \rightarrow c$, $F^d(\emptyset) \ni b$ and $F^d(\emptyset) \not\ni c$ for some $d$. We write $b \rightarrow_{uk} c \in h \downarrow a$ when $b \rightarrow c$ holds and $b \not\rightarrow_{ut} c \in h \downarrow a$ does not hold.

For a heap $h$, $a \in \text{Val}$ and $S \subseteq \text{Val}$, the heap $h \downarrow S a$ is defined as $(h|_{\text{Dom}(h)-(S-\{a\})}) \downarrow a$, and we call it a cone of root $a$ with guard $S$ in the heap $h$.

We write $a \in h$ for $a \in \text{Dom}(h)$, and $h \subseteq h'$ for $h = h'|_{\text{Dom}(h)}$.

Lemma 12.2 (1) If

$s, h \models \Gamma,$

then

$$h = \bigcup_{x \in \text{Roots}(\Gamma)} h \downarrow s(x).$$

(2) If $h = h_1 + h_2$ and $\Gamma$ is $\Gamma_1 \cdot \Gamma_2$ and for $i = 1, 2$

$s, h_i \models \Gamma_i \wedge (\text{FV}(\Gamma) - (\text{Roots} + \text{Cells})(\Gamma_i)) \uparrow$,

then

$$h_i = \bigcup_{x \in (\text{Roots} + \text{Cells})(\Gamma_i)} h \downarrow_{s((\text{Roots} + \text{Cells})(\Gamma_i))} s(x).$$

Proof. (1) Let $\Gamma$ be $s_i(\mathcal{P}_i(x_i, \overrightarrow{\mathcal{T}}_i) \wedge X_i \downarrow)$. Then we have $h_i$ such that $h = \Sigma_i h_i$ and $s', h_i \models \mathcal{P}_i(x_i, \overrightarrow{\mathcal{T}}_i) \wedge X_i \downarrow$.

Hence

$$h_i \subseteq h \downarrow s(x_i).$$

Hence

$$h \subseteq \bigcup h_i(h \downarrow s(x_i)).$$

Hence

$$h = \bigcup_{x \in \text{Roots}(\Gamma)} h \downarrow s(x).$$

(2) We write $\text{RC}$ for $(\text{Roots} + \text{Cells})$ for simplicity.

By (1),

$$h = \bigcup_{x \in \text{Roots}(\Gamma)} h \downarrow s(x).$$

Hence

$$h = \bigcup_{x \in \text{RC}(\Gamma)} h \downarrow_{s(\text{RC}(\Gamma))} s(x).$$

We can show

$$h_1 \supseteq \bigcup_{x \in \text{RC}(\Gamma_1)} h \downarrow_{s(\text{RC}(\Gamma))} s(x)$$

as follows. Assume $\not\supseteq$ in order to show contradiction. Then there is $a \in h_1$ and $a$ in the right-hand side. Hence $a \in h_2$. Hence there is $x \in \text{RC}(\Gamma_1)$ such that $a \in h \downarrow_{\text{RC}(\Gamma)} s(x)$. Hence there is the path $s(x) \rightarrow a$. By going from $s(x)$ to $a$, take the first $h_2$-element to be $c$. Let the previous element be $b.$
Then \( s(x) \rightarrow b \rightarrow c \rightarrow a \) and \( s(x) \rightarrow b \) is in \( h_1 \). From \( s, h_1 \models \Gamma_1 \) and \( s, h_2 \models \Gamma_2 \), since \( b \rightarrow c \) is from one cone to another cone, we have \( y \in \text{FV}(\Gamma) \) such that \( c = s(y) \). Since \( c \in h_2 \), we have \( y \in \text{RC}(\Gamma_2) \), which contradicts with the path \( s(x) \rightarrow b \rightarrow s(y) \rightarrow a \) in \( h \downharpoonright_{\text{RC}(\Gamma)} s(x) \) since \( y \in \text{RC}(\Gamma) \).

Similarly we have
\[
h_2 \supseteq \bigcup_{x \in \text{RC}(\Gamma_2)} h \downharpoonright_{\text{RC}(\Gamma)} s(x).
\]
Since
\[
h_1 + h_2 = \bigcup_{x \in \text{RC}(\Gamma_1) \cup \text{RC}(\Gamma_2)} h \downharpoonright_{\text{RC}(\Gamma)} s(x),
\]
we have
\[
h_i = \bigcup_{x \in \text{RC}(\Gamma_i)} h \downharpoonright_{\text{RC}(\Gamma)} s(x).
\]
\( \square \)

**Lemma 12.3** (1) If \( \Gamma \) is \( \Gamma_1 * \Gamma_2 \), and \( \Gamma' \) is \( \Gamma'_1 * \Gamma'_2 \), and
\[
h_1 + h_2 = h'_1 + h'_2,
\]
and for \( i = 1, 2, \)
\[
s, h_i \models \Gamma_i \wedge (\text{FV}(\Gamma) - (\text{Roots} + \text{Cells})(\Gamma_i)) \uparrow,
\]
\[
s', h'_i \models \Gamma'_i \wedge (\text{FV}(\Gamma') - (\text{Roots} + \text{Cells})(\Gamma'_i)) \uparrow,
\]
\[
\text{Roots}(\Gamma'_i) \subseteq (\text{Roots} + \text{Cells})(\Gamma_i) \subseteq (\text{Roots} + \text{Cells})(\Gamma'_i),
\]
then \( h_i = h'_i \) for \( i = 1, 2 \).

(2) If
\[
s, h_i \models \Gamma_i \ (i = 1, 2),
\]
\[
s, h_1 + h_2 \models \Phi_1 * \Phi_2,
\]
and \( (\Gamma_1, \Gamma_2) \) is group-disjoint, \( (\Phi_1, \Phi_2) \) is a group split by \( (\Gamma_1, \Gamma_2) \), then
\[
s, h_i \models \Phi_i \ (i = 1, 2).
\]

**Proof.** (1) Let \( V_i \) be \( (\text{Roots} + \text{Cells})(\Gamma_i) \) and \( V'_i \) be \( (\text{Roots} + \text{Cells})(\Gamma'_i) \) for \( i = 1, 2 \). Let \( V \) be \( V_1 \cup V_2 \) and \( V' \) be \( V'_1 \cup V'_2 \). Let \( h \) be \( h_1 + h_2 \).

By Lemma [12.12] (2),
\[
h'_i = \bigcup_{x \in V'_i} h \downharpoonright_{\text{RC}(V')} s'(x).
\]
We can show
Claim 1: \( x \in V'_1 \setminus V_1 \) implies \( s(x) \in h_1 \)

as follows. Assume \( x \in V'_1 \setminus V_1 \). Then \( x \in \text{Cells}(\Gamma'_1) \). Then \( s(x) \notin s(V) \) since \( s(x) \in s(V_2) \) implies \( s(x) \in s(V'_2) \), which contradicts with \( x \in V'_1 \). Moreover there is \( y \in \text{Roots}(\Gamma'_1) \) such that \( P(y) \wedge x \downarrow \) is in \( \Gamma'_1 \). Hence there is a path \( s(y) \rightarrow s(x) \) in \( h'_1 \). By going from \( s(y) \) to \( s(x) \), take \( s(z) \) to be the last \( s(V) \)-element. Then \( z \in V \) and \( s(y) \rightarrow s(z) \rightarrow a \rightarrow s(x) \) and \( a \rightarrow s(x) \) are not in \( s(V) \). Then \( z \in V_1 \) since \( z \notin V_1 \) implies \( z \in V_2 \) and \( z \in V'_2 \) from \( V_2 \subseteq V' \), so \( s(z) \in h'_2 \), which contradicts with \( s(z) \in h'_1 \).

Hence \( s(x) \in h \downharpoonright_{\text{RC}(V)} s(z) \). By Lemma [12.12] (3),
\[
h_1 = \bigcup_{x \in V_1} h \downharpoonright_{\text{RC}(V)} s(x).
\]
Hence \( h_1 \supseteq s(x) \). We have shown the claim 1.

We will show
Claim 2: \( h \downharpoonright_{\text{RC}(V')} s(x) \subseteq h_1 \) for any \( x \in V'_1 \)

as follows. If \( x \in V'_1 \), then \( h \downharpoonright_{\text{RC}(V')} s(x) \subseteq h \downharpoonright_{\text{RC}(V)} s(x) \subseteq h_1 \). Assume \( x \in V'_1 \setminus V_1 \). By the claim 1, \( s(x) \in h_1 \). Hence there is \( y \in V_1 \) such that \( s(x) \in h \downharpoonright_{\text{RC}(V)} s(y) \). Hence \( h \downharpoonright_{\text{RC}(V)} s(x) \subseteq h \downharpoonright_{\text{RC}(V)} s(y) \subseteq h_1 \). We have shown the claim 2.

By the claim 2, \( h_1 \supseteq h'_1 \). Similarly we have \( h_2 \supseteq h'_2 \). Hence \( h_i = h'_i \) for \( i = 1, 2 \).

(2) Note that the roots are not bounded in \( \Phi_i \) for \( i = 1, 2 \) by the group split.
We have \( h' + h'' = h_1 + h_2 \) such that
\[ s, h'_i \models \Phi_i. \]

Let \( \Phi_i \) be \( \exists v \exists \overrightarrow{y} \uparrow (\Pi_i \wedge \Gamma'_i) \). Then we have \( \overrightarrow{a}_i, \overrightarrow{b}_i \) such that by letting \( s' \) be \( s[\overrightarrow{a}_i := \overrightarrow{a}_i', \overrightarrow{b}_i := \overrightarrow{b}_i (i = 1, 2)] \), we have
\[ s', h'_i \models \Gamma'_i. \]

By (1), \( h_i = h'_i \).

### 13 Local Completeness of Rule (BoundedFactor) in Algorithm

A rule is defined to be \textit{locally complete} if all its assumptions are valid when its conclusion is valid.

**Lemma 13.1** Let \( V, d \) be the arguments sent to MainLoop from Main. Then, among processes produced by MainLoop, there is some process in which every use of the rule
\[
\begin{align*}
x \neq y \wedge F \vdash \overrightarrow{G}, \{G[\exists \overrightarrow{w}((Q_2(\overrightarrow{t}_1), Q(y, \overrightarrow{w}) \rightarrow \star P(x, \overrightarrow{t}))) | \{Q_2(\overrightarrow{t}_1), Q_2(\overrightarrow{t}_2)\} = \{Q(\overrightarrow{t}), Q \in \text{Dep}(P), \overrightarrow{Q_2} \subseteq \text{Dep}(Q), \\
|Q_1(\overrightarrow{t}_1), Q(y, \overrightarrow{w})|_V \leq d, |Q_2(\overrightarrow{t}_2)|_V \leq d, \overrightarrow{w} \text{ fresh}\}] \\
x \neq y \wedge F \vdash \overrightarrow{G}, G[Q(\overrightarrow{t}) \rightarrow \star P(x, \overrightarrow{t})) \wedge y \downarrow]
\end{align*}
\]

is locally complete.

**Proof.** Assume all the processes use some locally incomplete application of (BoundedFactor), in order for contradiction.

Consider the tree of processes where each path represents a process, the nodes are factor nodes and fork nodes, which represent the application of the bounded factor rule and fork with new processes respectively, and any process is represented by some path. We cut each path by the first application of locally incomplete application of the bounded factor rule. By the assumption, each path is cut. By König’s Lemma, there is the maximum depth of the cut tree. Take a path of the maximum depth and consider the process represented by this path. Then for every choice for the application of the bounded factor rule by the process at the cut node is locally incomplete. (Otherwise, by choosing some application that is locally complete, the path can extend more than the maximum depth, which contradicts.)

For simplicity, we write \( v \rightarrow \star v' \rightarrow \star P'(v) \rightarrow \star P''(v') \) for some \( P', P'' \). We also write \( \{v_1, \ldots, v_n\} \rightarrow \star v \) for \( P_1(v_1) \rightarrow \star \ldots \rightarrow \star P_n(v_n) \rightarrow \star P(v) \) for some \( P_1, \ldots, P_n \). Note that any order of \( \{v_1, \ldots, v_n\} \) gives the equivalent inductive predicate by Lemma 5.1.

Consider this process at the cut node. We write \( p \) for this process at the cut node.

Fix an application of the bounded factor rule in \( p \).

Since the rule (Factor) without the restriction on depth of wands is locally complete by Lemma 5.3, we have some \( \overrightarrow{F}, \Phi, \overrightarrow{F'}, \Phi' \), such that the entailment of the cut note is the conclusion \( \psi \models \overrightarrow{F}, \Phi \) of the rule (Factor) and valid, \( \psi \models \overrightarrow{F}, \Phi' \) is the assumption of the rule with the restriction \( d \) and invalid, \( \psi \models \overrightarrow{F}, \Phi', \Phi'' \) is the assumption of the rule without the restriction and valid and contains some wand of depth \( d + 1 \). Then there are \( s', h' \) such that
\[
\begin{align*}
s', h' &\models \psi, \\
s', h' &\models \overrightarrow{F}, \Phi'.
\end{align*}
\]

Then there is \( \Phi'' \in \overrightarrow{F''} \) such that
\[ s', h' \models \Phi''. \]

Let
\[ \Phi'' = F[\exists \overrightarrow{x}(S \rightarrow \star x) \star y] \]

where \( S \) is a set of variables and \( y \in S \) and \( |S - V| = d + 1 \).
Define $s$ from $s'$ by adding assignments for existential variables in $\Phi''$. Then we have some $h'' \subseteq h'$ such that

$$s, h'' \models S \rightarrow \ast^*x.$$

We obtain some initial heap $h$ by going back along the computation from $h'$ to the beginning in reverse order.

We call a variable $w$ an extra in an entailment $J$ when $w \in \text{Roots}(w) - \bigcap_i (\text{Roots}(\Phi_i))$ where $J$ be $\psi \vdash (\Phi_i)_{i \in I}$. Assume $w \not\in V$. The wand $P(w) \rightarrow \ast^* \ldots$ appears in the computation only when $w$ is an extra at some step. $w$ is an extra at the step only when at the previous steps we unfold some predicate with some root in the antecedent and unfold another predicate with the same root in the succedent, and we match some $\exists z$ in the antecedent and the corresponding $\exists w$ in the succedent.

By this observation, we make a sequence as follows: We start with some process $w$ such that $w \not\in V$, we match some $\exists y, z$'s may be the same. $z_i \neq z'_i$ for all $i$, $z'_i$ may be in $V$.

Since an application of the bounded factor rule in $p$ is arbitrary, we have these for each application of the bounded factor rule in $p$. We index these applications by $I$. Then in $h$ for the succedent, for all $i \in I$, we have sequences $y_i, z_{i1}, \ldots, z_{i(d+1)}$ such that

$$y_i \in V,$$
$$z_i \not\in V,$$
$$y_i \leftarrow bk z_{i1},$$
$$z_i \leftarrow tr \rightarrow bk z_{i+1} \quad (1 \leq i \leq d),$$
$$z'_i \leftarrow tr z_i \quad (1 \leq j \leq d + 1),$$
$$z'_{i(d+1)} \leftarrow tr \ldots \rightarrow tr z_{i1} \leftarrow tr y_i.$$  

$z'_{i1}$'s may be the same. $z_{ij} \neq z'_{ij}$ for all $i, j$. $z_{ij}$ may be in $V$.

**Claim 1.** $z_{ij}$'s are different.

It is because: It is clear that $z_{ij}$'s are different for a fixed $i$. For $i \neq k$, $z_{ij}$ and $z_{kl}$ are different since they belong to different heaps after some split step.

**Claim 2.** In $h$ for the antecedent, $z_{ij} \quad (1 \leq j \leq d)$ is not below $z_{k(d+1)}$.

It is because $z_{ij}$ is unfolded before $z_{k(d+1)}$ is unfolded.

**Claim 3.** The path $z'_{i(d+1)} \leftarrow tr \ldots \rightarrow tr z_{i1} \leftarrow tr y_i$ does not contain any $z_{kj}$.

We can show it as follows. Assume the path contains some $z_{kj}$ in order for contradiction. Consider some process $q$ such that $q$ unfolds the same variables as $p$, and $q$ unfolds $z_{kj}$ before $q$ does the application of bounded factor indexed by $i$. Then this application by $q$ is locally complete with depth $d$, so the path of $q$ is longer than that of $p$, which contradicts with the maximum length for $p$. Hence we have shown the claim 3.

**Claim 4.** In $h$ for the antecedent, every $z'_{ij}$ is below some $z_{k(d+1)}$.

We can show it as follows. Assume some $z'_{ij}$ is not below $z_{k(d+1)}$ for all $k \in I$, in order for contradiction. Then $z'_{ij}$ is above or equals every $z$ such that the bounded factor for $z$ is locally incomplete. Consider some process $p$ such that $p$ unfolds the same variables as $p$, and $p$ unfolds $z'_{ij}$ before $q$ does the application of bounded factor indexed by $i$. Then this application by $q$ is locally complete with depth $d$, so the path of $q$ is longer than that of $p$, which contradicts with the maximum length for $p$. Hence we have shown the claim 4.

By the claim 1, the number of $z_{ij}$'s is $|I|(d+1)$. Since the number of $z_{i(d+1)}$'s is $|I|$, by the claims 2 and 4, there is some $k \in I$ such that $|S| \geq d + 1$ where $S$ is the set of $z_{ij}$ such that $z'_{ij}$ below $z_{k(d+1)}$ in $h$ for the antecedent.
By the claims 2 and 4, the paths $z'_i \rightarrow z_i$ ($1 \leq i \leq k+1$) are back edges in $h$ for the antecedent. Hence we have every element of $S$ in the arguments of the predicate for $z_{k(d+1)}$. But $|S| \geq d + 1 = k_{\text{max}} + 1$ and the number of the arguments at $z_{k(d+1)}$ is $\leq k_{\text{max}}$, which contradicts. $\blacksquare$

14 Local Completeness of Step Unrelatedness Introduction

Lemma 14.1 If

$$Y \uparrow \land \Gamma \ast \Delta_1 \ast \Delta_2 \models \Phi \uparrow \exists \overline{w}(Y') \uparrow \land \Pi' \land \Gamma' \ast (\Gamma_1 \ast (\Delta \land w) \downarrow) \ast \Gamma_2),$$

$$w \in \overline{w},$$

$$(\text{Roots + Cells})(\Gamma) = (\text{Roots + Cells})(\Gamma') - \overline{w},$$

$$(\text{Roots + Cells})(\Delta_1) = (\text{Roots + Cells})(\Gamma_1 \ast (\Delta \land w) \downarrow) - \overline{w},$$

$$w \in \land \overline{w},$$

$$(\text{Roots + Cells})(\Delta_2) = (\text{Roots + Cells})(\Gamma_2) - \overline{w},$$

and $\exists \overline{w}(Y') \uparrow \land \Pi' \land \Gamma' \ast (\Gamma_1 \ast (\Delta \land w) \downarrow) \ast \Gamma_2)$ is equality-full, then

$$Y \uparrow \land \Gamma \ast \Delta_1 \ast \Delta_2 \models \Phi \uparrow \exists \overline{w}(Y') \uparrow \land \Pi' \land \Gamma' \ast (\Gamma_1 \ast (\Delta \land w) \downarrow) \ast (\Gamma_2 \land w \uparrow)).$$

Proof. Assume $s, h$ satisfies the antecedent. Then we have $h_0 + h_1 + h_2 = h$ such that

$s, h_0 \models \Gamma,$

$s, h_1 \models \Delta_1,$

$s, h_2 \models \Delta_2.$

If $s, h \models \Phi$, the claim immediately holds. Assume

$s, h \models \exists \overline{w}(Y') \uparrow \land \Pi' \land \Gamma' \ast (\Gamma_1 \ast (\Delta \land w) \downarrow) \ast \Gamma_2).$

Then we have $\overline{w}, h'_0 + h'_1 + h'_2 = h$ such that by letting $s'$ be $s[\overline{w} := \overline{w}],$

$s', h'_0 \models \Gamma',$

$s', h'_1 \models \Gamma_1 \ast (\Delta \land w) \downarrow,$

$s', h'_2 \models \Gamma_2.$

By Lemma 12.3 (1),

$h_0 = h'_0,$

$h_1 = h'_1,$

$h_2 = h'_2.$

Hence we have $w \uparrow$ for $h'_2$. By Lemma 4.5 and the equality-fullness, $s'(w) \notin \text{Leaves}(h_2)$. Hence

$s', h'_2 \models w \uparrow.$

$\blacksquare$

Lemma 14.2 At the Unrelatedness Introduction step, the rule

$$\psi \models \Phi, \exists \overline{w}(Y') \uparrow (\Pi \land \Gamma \ast (\Delta \land x \downarrow) \ast *_i(\Gamma_i \land x \uparrow))$$

$$\psi \models \Phi, \exists \overline{w}(Y') \uparrow (\Pi \land \Gamma \ast (\Delta \land x \downarrow) \ast *_i \Gamma_i)$$

is locally complete.

Proof. Since $\Pi \land \Gamma \ast (\Delta \land x \downarrow) \ast *_i \Gamma_i)$ is grouping and equality-full at the step, there are $\Delta_1, \Delta_2$ such that the conditions of Lemma 14.1 hold. By Lemma 14.1 the claim holds. $\blacksquare$
15 Selective Local Completeness of Rule (⋆)

A set of rules of the same conclusion is defined to be selectively locally complete if for every valid conclusion of the rules there is a locally complete rule in the set.

Proposition 15.1 Let $S$ be the set of

$$(Y \cup Y_1) \uparrow \land \Pi \land \Gamma_2 \vdash \{ \Phi_2[i] | i \in I - I' \} \text{ or } (Y \cup Y_2) \uparrow \land \Pi \land \Gamma_1 \vdash \{ \Phi_1[i] | i \in I' \} \quad (\forall I' \subseteq I) \tag{Split}$$

where

$V = \text{FV}(Y \uparrow \land \Pi \land \Gamma_1 \ast \Gamma_2 \vdash \{ \Phi_1[i] * \Phi_2[i] | i \in I \})$,

$Y_i = (\text{Roots} + \text{Cells})(\Gamma_i)$ ($i = 1, 2$),

$\Pi \equiv (\neq (V))$,

$(\text{Roots} + \text{Cells})(\Gamma_1 \ast \Gamma_2) + Y = V$,

$(\Gamma_1, \Gamma_2)$ is group-disjoint, and $(\Phi_1[i], \Phi_2[i])$ $(i \in I)$ is group split by $(\Gamma_1, \Gamma_2)$. Then $S$ is selectively locally complete, namely, if $Y \uparrow \land \Pi \land \Gamma_1 \ast \Gamma_2 \vdash \{ \Phi_1[i] * \Phi_2[i] | i \in I \}$ is valid, then there is some rule in $S$ such that all assumptions of the rule are valid.

Proof. Assume

$$Y \uparrow \land \Pi \land \Gamma_1 \ast \Gamma_2 \vdash \{ \Phi_1[i] * \Phi_2[i] | i \in I \}.$$

Let

$\psi_1 = (Y \cup Y_2) \uparrow \land \Pi \land \Gamma_1$,

$\psi_2 = (Y \cup Y_1) \uparrow \land \Pi \land \Gamma_2$.

Fix $s, h_1, h_2$ and assume

$s, h_1 \models \psi_1$,

$s, h_2 \models \psi_2$.

Let $h = h_1 + h_2$. Then

$s, h \models Y \uparrow \land \Pi \land \Gamma_1 \ast \Gamma_2$.

Then we have

$$\bigvee_{i \in I} s, h_1 + h_2 \models \Phi_1[i] * \Phi_2[i]$$

By Lemma 12.3 (2),

$$\bigvee_{i \in I} (s, h_1 \models \Phi_1[i] \land s, h_2 \models \Phi_2[i]).$$

By Lemma 7.1

$$\bigwedge_{l = l_1 + l_2} \left( (\bigvee_{i \in I_1} s, h_1 \models \Phi_1[i]) \lor (\bigvee_{i \in I_2} s, h_2 \models \Phi_2[i]) \right).$$

Since $s, h_1, h_2$ are arbitrary,

$$\forall s \forall h_2 (s, h_1 \models \psi_1 \land s, h_2 \models \psi_2 \rightarrow$$

$$\bigwedge_{l = l_1 + l_2} \left( (\bigvee_{i \in I_1} s, h_1 \models \Phi_1[i]) \lor (\bigvee_{i \in I_2} s, h_2 \models \Phi_2[i]) \right)).$$

Fix $I_1 + I_2 = I$. Then

$$\forall s \forall h_2 (s, h_1 \models \psi_1 \land s, h_2 \models \psi_2 \rightarrow$$

$$((\bigvee_{i \in I_1} s, h_1 \models \Phi_1[i]) \lor (\bigvee_{i \in I_2} s, h_2 \models \Phi_2[i]) \)).$$
Hence
\[ \forall s,h_1 h_2 ((s,h_1 \models \psi_1 \land s,h_2 \models \psi_2 \rightarrow (\bigvee_{i \in I_1} s,h_1 \models \Phi_{1i})) \lor (s,h_1 \models \psi_1 \land s,h_2 \models \psi_2 \rightarrow (\bigvee_{i \in I_2} s,h_2 \models \Phi_{2i}))). \]

If \( s,h_1 \models \psi_1 \land s,h_2 \models \psi_2 \), then \( \bigvee_{i \in I_1} (s,h_1 \models \Phi_{1i}) \lor \bigvee_{i \in I_2} (s,h_2 \models \Phi_{2i}) \). If \( s,h_1 \not\models \psi_1 \), then \( s,h_1 \models \psi_1 \rightarrow \bigvee_{i \in I_1} (s,h_1 \models \Phi_{1i}) \). Hence
\[ \forall s,h_1 h_2 ((s,h_1 \models \psi_1 \rightarrow \bigvee_{i \in I_1} s,h_1 \models \Phi_{1i}) \lor (s,h_2 \models \psi_2 \rightarrow \bigvee_{i \in I_2} s,h_2 \models \Phi_{2i})). \]

Hence
\[ \forall s (\psi_1 \models s \{ \Phi_{1i} | i \in I_1 \} \lor \psi_2 \models s \{ \Phi_{2i} | i \in I_2 \}). \]

By Lemma \ref{lemma11.3}
\[ \psi_1 \models s \{ \Phi_{1i} | i \in I_1 \} \lor \psi_2 \models s \{ \Phi_{2i} | i \in I_2 \}. \]

Since \( I_1 + I_2 = I \) are arbitrary, we have
\[ \psi_1 \models s \{ \Phi_{1i} | i \in I_1 \} \lor \psi_2 \models s \{ \Phi_{2i} | i \in I_2 \} \]
for all \( I_1 + I_2 = I \).

Consider the rule in \( S \) such that for each \( I' \), by taking \( I_1 \) to be \( I' \) and \( I_2 \) to be \( I - I' \), the assumption for \( I' \) is taken to be the first disjunct when
\[ \psi_2 \models s \{ \Phi_{2i} | i \in I_2 \} \]
and is taken to be the second disjunct when
\[ \psi_1 \models s \{ \Phi_{1i} | i \in I_1 \}. \]

Then all the assumptions of this rule are valid. \( \square \)

16 Fresh Variable in Succeedent

**Lemma 16.1** If \( \theta \) is a variable renaming, \( (\text{Dom} \cup \text{Range})(\theta) \subseteq X \), and \( (\text{Dom} \cup \text{Range})(\theta) \cap \text{FV}(\Gamma) = \emptyset \), and \( \Pi \supseteq (\neg (\text{FV}(X \uparrow \land \Pi \land \Gamma \models \overline{\Phi}, \Phi, \Phi \theta))) \) and \( \Phi \) is equality-full, then the rule
\[
\frac{X \uparrow \land \Pi \land \Gamma \models \overline{\Phi}, \Phi \theta}{X \uparrow \land \Pi \land \Gamma \models \overline{\Phi}, \Phi \theta} \quad \text{(FreshVariableDisjunctElim)}
\]
is locally complete.

**Proof.** Let \( \overline{x}_1 \) be \( \text{FV}(\Phi) \cap \text{Dom}(\theta) \) and \( \overline{x}_2 \) be \( \theta(\overline{x}_1) \).

Assume the conclusion of the rule is valid in order to show the assumption of the rule is valid. Assume \( s,h \models X \uparrow \land \Pi \land \Gamma \) in order to show \( \overline{\Phi}, \Phi \). Then \( s,h \models \overline{\Phi}, \Phi, \Phi \theta \). Assume \( s,h \models \Phi \). We will show \( s,h \models \Phi \).

By Lemma \ref{lemma11.3} \text{Leaves}(h) \subseteq s(\text{FV}(\Gamma) \cup \{\text{nil}\}). \) Hence \( s(\overline{x}_1 \overline{x}_2) \notin \text{Leaves}(h), \) since \( \overline{x}_1 \overline{x}_2 \notin \text{FV}(\Gamma) \). Hence \( s(\overline{x}_1 \overline{x}_2) \notin (\text{Dom} \cup \text{Range})(h), \) since \( \overline{x}_1 \overline{x}_2 \subseteq X \).

Let \( T \) be \( \text{FV}(\Phi_1) \cup \{\text{nil}\} - \overline{x}_1 \). Then
\[ s,h \models \exists \overline{x}_2 \uparrow (\Phi \land (\neg (\overline{x}_2, T \cup \overline{x}_2))). \]
By variable renaming,

\[ s, h \models \exists \overline{x} \uparrow (\Phi \land (\neq (\overline{x}, T \cup \overline{x}))). \]

Here \( \exists \overline{x} \uparrow (\Phi \land (\neq (\overline{x}, T \cup \overline{x}))) \) is equality-full. By Lemma 6.1 (2),

\[ s[\overline{x} := s(\overline{x})], h \models \Phi \land (\neq (\overline{x}, T \cup \overline{x})). \]

Hence

\[ s, h \models \Phi. \]

\[ \square \]

### 17 Completeness of CSLID\(^\omega\)

This section shows the completeness of CSLID\(^\omega\) by using the algorithm, the local completeness, and the termination.

By the lemmas in previous sections, we can show some process does not fail for a valid input.

**Lemma 17.1**

1. In the algorithm, each step except the rules (BoundedFactor) and (Split) in each function except Main and MainLoop transforms a valid entailment into subgoals consisting of valid entailments.

2. If a valid entailment is given to the algorithm, for each call of the function MainLoop, there is some process such that all applications of the rules (BoundedFactor) and (Split) are locally complete.

3. If a valid entailment is given to the algorithm, for each call of the function MainLoop, some process does not fail.

**Proof.** (1) We only discuss interesting cases.

- The function Split.
  - Step 4. Existential Disequality. We have \( \neq \) for every variable except \( \overline{y} \). \( \overline{y} \neq \overline{z} \) from the antecedent. \( \overline{y} \neq \overline{z} \) from \( \overline{y} \uparrow \) and \( \overline{z} \downarrow \).
  - Step 5. Unrelatedness Introduction. By Lemma 14.2
  - Normalize
  - Step 1. Fresh Variable Disjunct Elimination. By Lemma 16.1
  - Step 2. Unnecessary Disequality Elimination. Assume the antecedent of the assumption is true at \((s, h)\) in order to show the succedent of the assumption is true at \((s, h)\). Choose \( a \in \text{Val} - \text{Dom}(h) \) such that \( a \notin s(\text{FV}(\Pi, \Gamma, \overline{y})) \cup \{\text{nil}\} \). Let \( s' \) be \( s[x := a] \). Then the antecedent of the conclusion is true at \((s', h)\). Then the succedent of the conclusion is true at \((s', h)\). Then the succedent of the assumption is true at \((s', h)\). Since \( y \) does not appear in the succedent, the succedent of the assumption is true at \((s, h)\).
  - Step 3. Unnecessary Variable Elimination. It is similar to Unnecessary Disequality Elimination by choosing \( a \notin \text{Dom}(h) \).
  - Step 4. Unnecessary Unrelated Existential Elimination. By Lemma 4.3

2. By Proposition 15.1 and Lemma 13.1

3. From (2), we have some process such that all applications of the rules (BoundedFactor) and (Split) are locally complete. Since Main does case analysis and MainLoop does case analysis and discharges by bud-companion relation. By (1), in this process, applications of each rule are locally complete. Hence all the entailments handled by this process are valid. We show this process does not fail. Assume it fails in order to show contradiction. When the process fails, there is either some Termination Check step in the function MainLoop or the step for checking emp in the function Main. In both cases we have contradiction since the entailment is shown to be invalid as follows: For the function Main. Since \( \Pi \rightarrow \bigvee \Pi' \) is not true, we have \( s \) that satisfies \( \Pi \) and \( \bigwedge \neg \Pi' \). Take \( h \) to be the empty heap. \((s, h)\) is a counterexample since for every disjunct either emp is not in it or \( \Pi' \) in it is false. For MainLoop. Since \( \psi \) is satisfiable by the Unsat check step, we have \( s \) that satisfies \( \Pi \). Take \( h \) to be the empty heap. Then \((s, h)\) is a counterexample since emp is not in the succedent. \( \square \)

Finally we can prove the completeness of CSLID\(^\omega\).
**Theorem 17.2 (Completeness)** (1) The system CSLID$^\omega$ is complete. Namely, if a given entailment $J$ is valid, then it is provable in CSLID$^\omega$.

(2) The algorithm decides the validity of a given entailment. Namely, For a given input $J$, the algorithm returns Yes when the input is valid, and it returns No when the input is invalid.

**Proof.** (1) Assume $J$ is valid in order to show $J$ is provable in CSLID$^\omega$. When we input $J$ to the algorithm, by Lemma 17.1 (3), for each call of MainLoop, some process does not fail. By Lemma 10.6 (2), for each call of MainLoop, the process terminates without fail. Hence the algorithm terminates with Yes. By Lemma 10.3, $J$ is provable.

(2) Assume $J$ is valid, in order to show the algorithm with input $J$ terminates with Yes. By Lemma 17.1 (3), for each call of MainLoop, some process does not fail. By Lemma 10.6 (2), the algorithm terminates with Yes.

Assume $J$ is invalid, in order to show the algorithm with input $J$ terminates with No. By Lemma 10.6 (2), the algorithm terminates. Assume the algorithm with input $J$ terminates with Yes, in order to show contradiction. By Lemma 10.3, $J$ is provable. By Theorem 8.3, $|J| \neq J$, which contradicts. Hence the algorithm with input $J$ terminates with No. □

18 Conclusion

We have proposed the cyclic proof system CSLID$^\omega$ for symbolic heaps with inductive definitions, and have proved its soundness theorem and its completeness theorem, and have given the decision procedure for the validity of a given entailment.

Future work would be to apply ideas in this paper to other systems, in particular, the strong wand and the selective local completeness of the rule ($\ast$).

References

[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] S. Berardi and M. Tatsuta, Classical System of Martin-Lof’s Inductive Definitions is not Equivalent to Cyclic Proof System, In: Proceeding of FoSSaCS 2017, LNCS 10203 (2017) 301–317.

[3] S. Berardi and M. Tatsuta, Equivalence of Inductive Definitions and Cyclic Proofs under Arithmetic, In: Proceedings of LICS2017 (2017) 1–12.

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

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

[6] J. Brotherston, Formalised Inductive Reasoning in the Logic of Bunched Implications, In: Proceedings of SAS 2007, LNCS 4634 (2007), 87–103.

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

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

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

[10] J. Brotherston, C. Fuhs, N. Gorogiannis, and J. Navarro Pérez, A Decision Procedure for Satisfiability in Separation Logic with Inductive Predicates, In: Proceedings of CSL-LICS’14 (2014) Article 25.

[11] J. Brotherston, N. Gorogiannis, M. Kanovich, and R. Rowe, Model checking for symbolic-heap separation logic with inductive predicates, In: Proceedings of POPL 2016 (2016) 84-96.

[12] W. Chin, C. David, H. Nguyen, and S. Qin, Automated Verification of Shape, Size and Bag Properties via User-Defined Predicates in Separation Logic, In Science of Computer Programming 77 (9) (2012) 1006–1036.
[13] D. Chu, J. Jaffar, and M. Trinh, Automatic Induction Proofs of Data-Structures in Imperative Programs, In: Proceedings of PLDI 2015 (2015) 457–466.
[14] B. Cook, C. Haase, J. Ouaknine, M. J. Parkinson, J. Worrell, Tractable reasoning in a fragment of Separation Logic, In: Proceedings of CONCUR’11, LNCS 6901 (2011) 235–249.
[15] C. Enea, V. Saveluc, M. Sighireanu, Compositional invariant checking for overlaid and nested linked lists, In: Proceedings of ESOP 2013, LNCS 7792 (2013) 129–148.
[16] C. Enea, O. Lengál, M. Sighireanu, T. Vojnar, Compositional Entailment Checking for a Fragment of Separation Logic, In: Proceedings of APLAS 2014, LNCS 8858 (2014) 314–333.
[17] 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.
[18] R. Iosif, A. Rogalewicz, and T. Vojnar, Deciding Entailments in Inductive Separation Logic with Tree Automata, In: Proceedings of ATVA 2014, LNCS 8837 (2014) 201–218.
[19] J. Navarro Pérez and A. Rybalchenko, Separation logic modulo theories, In: Proceedings of APLAS 2013, LNCS 8301 (2013) 90-106.
[20] R. Piskac, T. Wies, and D. Zufferey, Automating Separation Logic Using SMT, In: Proceedings of CAV 2013, LNCS 8044 (2013) 773-789.
[21] R. Piskac, T. Wies, and D. Zufferey, Automating separation logic with trees and data, In: Proceedings of CAV 2014, LNCS 8559 (2014) 711–728.
[22] J.C. Reynolds, Separation Logic: A Logic for Shared Mutable Data Structures, In: Proceedings of Seventeenth Annual IEEE Symposium on Logic in Computer Science (LICS2002) (2002) 55–74.
[23] A. Simpson, Cyclic Arithmetic is Equivalent to Peano Arithmetic, In: Proceedings of FoSSaCS 2017 (2017) 283–300.
[24] M. Tatsuta and D. Kimura, Separation Logic with Monadic Inductive Definitions and Implicit Existentials, In: Proceedings of APLAS 2015, LNCS 9458 (2015) 69–89.
[25] M. Tatsuta, K. Nakazawa and D. Kimura, Completeness of Cyclic Proofs for Symbolic Heaps, arXiv:1804.03938, 2018.