Spatial Factorization in Cyclic-Proof System for Separation Logic

Koji Nakazawa[1], Makoto Tatsuta[2], Daisuke Kimura[3], Mitsuru Yamamura[1]

[1] Nagoya University, Japan
{knak, yamamura326}@sqlab.jp

[2] National Institute of Informatics / Sokendai, Japan
tatsuta@nii.ac.jp

[3] Toho University, Japan
kmr@is.sci.toho-u.ac.jp

Abstract We propose a new proof system for entailment checking in the separation logic with general inductive predicates. The proposed system is based on a cyclic-proof system and using the Unfold-Match-Remove proof strategy. One of the difficulties in this strategy is to find the predicates which should be unfolded. In order to solve this problem, we introduce a new inference rule, called the factor rule, which enables us to factorize the inductive predicates in spatial formulas and to find predicates to be unfolded in the Unfold-Match-Remove strategy. Our proof system is complete and decidable when we restrict inductive predicates to linear ones. We also give some experimental results by a prototype implementation of our proof-search procedure. Our system proves some challenging examples without the help of any heuristic mechanisms such as finding cut formulas or lemmas.

1 Introduction

Separation logic [24, 4, 5] is an extension of Hoare logic for the verification of programs manipulating heaps. One of the keys for automated verification systems based on separation logic is to decide the validity for entailments of formulas in a special class, called symbolic heaps.

In order to deal with programs manipulating recursive data structures such as lists and trees in separation logic, it is important to consider inductively defined predicates in symbolic heaps. Our purpose is to obtain a decision procedure for entailments of symbolic heaps with general inductive definitions. However inductive definitions without any restriction cause undecidability [3].

Iosif et al. [16] proposed the system SLRD$_{btw}$, which is the first decidable system for entailments of symbolic heaps with general inductive definitions. The condition on inductive definitions imposed in [16] is called the *bounded-treewidth condition.* The bounded-treewidth condition is one of the most flexible conditions for decidability.

In contrast to model theoretic approaches such as [16], we choose another approach based on a proof system, which has several advantages: it is extendable with new rules; it shows reasons for the output; it may be fast when the input entailment is valid; we easily change it into a semi-decision procedure with heuristics for proof search. There are some complete and decidable proof system for separation logic [4, 5], but all of them are not for general inductive definitions, but for fixed (or hard-coded) predicates. Therefore, it is an important issue to find a class of inductive predicates which is sufficiently expressive and for which we can achieve complete and decidable proof system.

Another issue on a proof system which deals with inductive predicates is how induction is represented. Cyclic-proof systems [8, 9] are suitable for our purpose. In cyclic proofs, the induction is represented as a cyclic structure, where some open assumptions are allowed as induction hypotheses. This mechanism gives us more efficient proof search, since we need not fix an induction formula in advance. In fact, some systems (explicitly or implicitly) based on cyclic-proof systems have been proposed for the entailment checking problem for separation logic [8, 9, 12, 25, 26]. However it is not well studied on complete or decidable cyclic-proof systems for separation logic.

This paper proposes a cyclic-proof system for entailment checking for symbolic heaps and a proof-search procedure for the proof system. Our proof system is complete and decidable when we restrict the inductive definitions to linear ones. We adopt the Unfold-Match proof-search strategy in [12] and introduce some new notions and mechanisms: (1) a new class of inductive definitions, called the *cone inductive definitions*, which covers sufficiently a large class of inductive definitions and is suitable for the Unfold-Match strategy, (2) a new rule for *spatial factorization*, called the *factor rule*, to prevent the proof search from getting stuck, and (3) an auxiliary extension of the language for explicit case analysis. Furthermore, in order to confirm the usefulness of the new mechanisms, we give a prototype prover based on our proof-search procedure with the factor rule and show some experimental results.

The class of the *cone inductive definitions* is introduced as a subclass of the class of the bounded-treewidth, by imposing an additional condition that the definition body itself guarantees existentials to be allocated. The cone inductive definition requires that every definition clause of the inductive predicate $P(x, \overrightarrow{y})$ contains the singleton heap $x \mapsto (\_)$, that is, the heap contains a memory cell named with x. We call the first argument x a *root* of $P(x, \overrightarrow{y})$, since it is the root of a tree-like structure described by the predicate. One of the simplest examples of the cone inductive definitions is the following singly-linked list:

$$\mathrm{ls}(x, y) =_{\mathrm{def}} x \mapsto y \vee \exists z.x \mapsto (z) * \mathrm{ls}(z, y).$$

The cone inductive definitions are still quite expressive, since this class contains doubly-linked lists, skip lists, nested lists, and so on.

Combining the Unfold-Match strategy to the cone inductive definitions and the cyclic proofs, we obtain the following strategy. First, unfold predicates of the same root x in both antecedent and succedent. Next, the existentials of the antecedent are replaced by fresh variables and instantiate the existentials of the succedent by matching. Finally, remove the same $x \mapsto (\_)$ in both antecedent and succedent. We repeat these steps for each subgoal. We call this strategy *Unfold-Match-Remove* and write U-M-R for it. This idea is natural and seems to work well. For example, a proof of the entailment $\mathrm{ls}(x, y) * \mathrm{ls}(y, z) \vdash \mathrm{ls}(x, z)$ can be found by U-M-R as

$$\dfrac{\begin{array}{c}\vdots\\ x \mapsto y * \mathrm{ls}(y,z) \vdash \mathrm{ls}(x,z)\end{array} \quad \dfrac{\dfrac{\dfrac{\mathrm{ls}(w,y) * \mathrm{ls}(y,z) \vdash \mathrm{ls}(w,z)}{x \mapsto w * \mathrm{ls}(w,y) * \mathrm{ls}(y,z) \vdash x \mapsto w * \mathrm{ls}(w,z)}\,(4)}{\exists w.x \mapsto w * \mathrm{ls}(w,y) * \mathrm{ls}(y,z) \vdash \exists w.x \mapsto w * \mathrm{ls}(x,z)}\,(3)}{\exists w.x \mapsto w * \mathrm{ls}(w,y) * \mathrm{ls}(y,z) \vdash \mathrm{ls}(x,z)}\,(2)}{\mathrm{ls}(x,y) * \mathrm{ls}(y,z) \vdash \mathrm{ls}(x,z)}\,(1) \quad ,$$

where we omit the left branch. At (1) and (2), we unfold the predicates of the same root x. At (3), the existentials are instantiated by matching. At (4), we remove the common $x \mapsto w$ from the both sides. Then, we have the entailment $\mathrm{ls}(w, y) * \mathrm{ls}(y, z) \vdash \mathrm{ls}(w, z)$ which is the same as the bottom entailment except renaming, so it is allowed as an induction hypothesis. Hence, the proof is completed in the cyclic-proof system.

However the U-M-R strategy may get stuck when we cannot find predicates on both sides which have a common root. For example, the entailment $\mathrm{dll}(h, t, p, n) \vdash \mathrm{dllrev}(t, h, n, p)$ is valid for the following two variants of the doubly-linked lists:

$$\mathrm{dll}(h, t, p, n) =_{\mathrm{def}} h = t \wedge h \mapsto (p, n) \vee \exists z.h \mapsto (p, z) * \mathrm{dll}(z, t, h, n),$$
$$\mathrm{dllrev}(h, t, p, n) =_{\mathrm{def}} h = t \wedge h \mapsto (n, p) \vee \exists z.h \mapsto (z, p) * \mathrm{dllrev}(z, t, h, n).$$

However, in a naive version of U-M-R, we cannot find common roots on both sides.

In order to solve this problem, we propose a new inference rule of the *spatial factorization*, called the factor rule. The factor rule is given by means of the *inductive wands*. The inductive wands represent some segments of heaps represented by inductive predicates, that is, the inductive wand $P \text{—}*^{\mathrm{i}} Q$ represents a subheap of Q which is obtained by removing a heap which satisfies P. The inductive wands are also predicates defined by cone inductive definitions. By the inductive wand, we can divide a predicate $P(x)$ in succedent into two predicates $Q(y)\text{—}*^{\mathrm{i}} P(x)$ and $Q(y)$, where we can find the root y which occurs in antecedent as a root. For the above example of dll and dllrev, the spatial factorization is applied as the following factor rule:

$$\frac{\mathrm{dll}(h,t,p,n) \vdash \exists t'p'n'.(\mathrm{dllrev}(h,t',p',n')\text{—}*^{\mathrm{i}}\mathrm{dllrev}(t,h,n,p)) * \mathrm{dllrev}(h,t',p',n')}{\mathrm{dll}(h,t,p,n) \vdash \mathrm{dllrev}(t,h,n,p)} \text{ (Factor)}.$$

Here, the inductive wand $\mathrm{dllrev}(h',t',p',n')\text{—}*^{\mathrm{i}}\mathrm{dllrev}(h,t,p,n)$ is defined by the definition clauses

$$\begin{aligned} & t' = t \wedge p' = h \wedge n' = n \wedge h \mapsto (h', p) \\ & \vee \exists z.h \mapsto (z,p) * (\mathrm{dllrev}(h',t',p',n')\text{—}*^{\mathrm{i}}\mathrm{dllrev}(z,t,h,n)). \end{aligned}$$

A similar idea to the inductive wand has been proposed as the inductive segment in [2], where they used it for software verification by abstract interpretation. By the inductive segment, we can unfold inductive predicates in the reverse direction to find some fixed points in shape analysis. On the other hand, we will use inductive wands for a different purpose, namely, deciding entailments of symbolic heaps. We will show the class of cone inductive predicates is closed under inductive wands. It is not easy to obtain a useful class of inductive definition that is closed under inductive wands. For example, the class of inductive definitions with bounded treewidth is not closed under inductive wands. Our results also show that the use of inductive wands keep decidability for cone inductive definitions. It is not easy either to obtain a decidable class that is closed under inductive wands, since magic wands often causes undecidability [1]. By this closure property, it is not necessary to add any special clauses to the definition of semantics of inductive wands.

Furthermore, for completeness and decidability, we extend the succedent to disjunctions and introduce predicates $t\downarrow$ and $t\uparrow$, which mean that t is in the heap and t is not in the heap respectively.

Our proof system is complete and decidable when we restrict cone inductive definitions to linear ones, that is, every definition clause of $P(x,\overrightarrow{y})$ has at most one atomic spatial formula except $x \mapsto (\_)$. The singly-linked and the doubly-linked lists are linear. For those structures, the factor rule give a complete proof system in a uniform way, whereas every existing proof system such as [4, 19] contains only one hard-coded inductive predicate and needs nontrivial rules, called the unroll-collapse.

The authors have a paper [28] deeply related to this paper. The proof system in [28] is the full system including the spatial factorization. Note that the inductive wands are called the strong wands in [28]. The completeness and decidability of the system are proved for the general cone inductive definitions, but this system needs many other mechanisms and is very complicated. This paper focuses on the details of the spatial factorization, which is one of the essential ideas in [28]. Moreover, this paper shows that the spatial factorization is sufficient to prove completeness and decidability if we restrict the predicate to linear ones.

Many entailment checkers for symbolic heaps with inductive definitions have been discussed. Several of them do not have general inductive definitions and have only hard-coded inductive predicates [4, 5, 13, 21, 22]. The entailment checkers for general inductive definitions are studied in [8, 9, 11, 12, 14, 15, 16, 17, 23, 27]. The systems in [14, 15, 16, 17, 23, 27] are all model theoretic. The systems in [8, 9, 12] use cyclic proofs, but neither of them is complete. The system in [11] is based on ordinary sequent calculus and is not complete.

We give the language and the semantics of our proof system in Section 2. The inductive wand and the factor rule are introduced in Section 3, and the proof-search procedure with the factor rule is introduced in Section 4. In Section 5, we prove that our proof system is complete and decidable for linear cone inductive predicates. The experimental results are given in Section 6. We conclude in Section 7.

2 Symbolic Heaps with Inductive Definitions

This section defines the symbolic heaps which we consider in our proof system.

We will use vector notation $\overrightarrow{x}$ to denote a sequence $x_1, \dots, x_k$ for simplicity. Sometimes we will also use this notation to denote a set for simplicity. We write $\equiv$ for the syntactical equivalence.

2.1 Language

The language of our system is based on that for the ordinary symbolic heaps in separation logic with inductively defined predicates. We consider only address values and no arithmetic on them. We have countably many first-order variables $x, y, z, \dots$ and inductive predicate symbols $P, Q, R, \dots$. We use $\mathcal{P}$ to denote either an inductive predicate or the point-to predicate $\mapsto$. When $\mathcal{P}$ is $\mapsto$, $\mathcal{P}(t, \overrightarrow{t})$ denotes $t \mapsto \overrightarrow{t}$. We sometimes write $\mathcal{P}(t)$ or $\mathcal{P}$ for $\mathcal{P}(t, \overrightarrow{t})$.

Definition 2.1 (Base language) *The base language is given as follows. The* terms *are defined as* $t, u, \dots ::= \mathrm{nil} \mid x$*, where* nil *is a first-order constant. The* pure formulas*, the* spatial formulas*, and the* symbolic heaps *are defined as follows.*

$$\begin{aligned} \Pi, \Pi' &::= t = u \mid t \neq u \mid \Pi \wedge \Pi' && \text{(pure formula)},\\ \Sigma &::= \mathrm{emp} \mid \mathcal{P}(t, \overrightarrow{t}) * \Sigma && \text{(spatial formula)},\\ \phi &::= \exists \overrightarrow{z}.\Pi \wedge \Sigma && \text{(symbolic heap)}. \end{aligned}$$

Here we assume that the length of $\overrightarrow{t}$ in $t \mapsto \overrightarrow{t}$ is fixed to some N_c. The term t in $\mathcal{P}(t, \overrightarrow{t})$ is called a root*. For sets S and S' of terms, we define the pure formula $(\neq (S, S')) = \bigwedge\{t \neq t' \mid t \in S,\ t' \in S',\ t \not\equiv t'\}$, and $(\neq S) = (\neq (S, S))$. For $I = \{1, \cdots, n\}$, we write $*_{i\in I}\mathcal{P}_i(t_i, \overrightarrow{t}_i)$ for $\mathcal{P}_1(t_1, \overrightarrow{t}_1) * \cdots * \mathcal{P}_n(t_n, \overrightarrow{t}_n)$. We sometimes omit the index set I as $*_i\mathcal{P}_i(t_i, \overrightarrow{t}_i)$. An* inductive definition system *is a finite set of* inductive definitions *of the form*

$$P(x, \overrightarrow{y}) =_{\mathrm{def}} \bigvee_{i\in I} \phi_i(x, \overrightarrow{y}) \qquad \text{(inductive definition)},$$

where the definition clauses $\phi_i(x, \overrightarrow{y})$ *must be of the form*

$$\phi_i(x, \overrightarrow{y}) \equiv \exists \overrightarrow{z}.\Pi \wedge x \mapsto (\overrightarrow{u}) * *_{j\in J} P_j(z_j, \overrightarrow{t}_j) \qquad \text{(definition clause)},$$

where $\mathrm{FV}(\mathrm{RHS}) \subseteq \{x, \overrightarrow{y}\}$*, and satisfy the conditions*

$$\overrightarrow{z} = \{z_j \mid j \in J\} \text{ (strong connectivity)} \qquad \textit{and} \qquad \overrightarrow{z} \subseteq \overrightarrow{u} \text{ (decisiveness)}.$$

We write $\phi \Rightarrow P(x, \overrightarrow{y})$ if ϕ is a definition clause of $P(x, \overrightarrow{y})$.

We call the inductive definition under these conditions *cone inductive definition*. These conditions imply the bounded-treewidth condition in [16]. The decisiveness guarantees that the cell at address x decides the content of every existential variable, and is similar to the constructively valued condition in [10].

An example that is covered by the bounded-treewidth condition but is not covered by the cone inductive definitions is the tree with parent pointers and linked leaves predicate tll given in [16].

2.2 Extended Language

We extend the language with the auxiliary predicates $x\uparrow$ and $x\downarrow$, which intuitively mean "x is not allocated" and "x is allocated", respectively. For a finite set X of variables, we write $X\uparrow$ and $X\downarrow$ for $\bigwedge\{x\uparrow \mid x \in X\}$ and $\bigwedge\{x\downarrow \mid x \in X\}$, respectively.

Definition 2.2 (Extended language) *We extend the spatial formulas of the base language as follows.*

$$\Delta ::= \mathcal{P}(t, \overrightarrow{t}) \wedge X\downarrow \qquad \text{(extended spatial atom)},$$
$$\Gamma ::= \mathrm{emp} \mid \Delta * \Gamma \qquad \text{(extended spatial formula)}.$$

The entailments *are of the form $Y\uparrow \wedge \Pi \wedge \Gamma \vdash \overrightarrow{\phi}$. We use ψ to denote an antecedent of an entailment.*

For saving space, we identify some syntactical objects that have the same meaning, namely, we use implicit transformation of formulas by using the following properties: $\wedge$ is commutative, associative, and idempotent; $*$ is commutative, associative, and has the unit emp; $=$ and $\neq$ are symmetric.

For the case analysis on $\downarrow$, we prepare the following operator to distribute $\downarrow$.

Definition 2.3 *We define $\Downarrow(\Gamma, X)$ and $\Downarrow(\psi, X)$ as follows.*

$$\Downarrow(*_{i\in I}\Delta_i, X) = \{*_{i\in I}(\Delta_i \wedge X_i\downarrow) \mid \ X = \uplus_{i\in I} X_i\},$$
$$\Downarrow(Y\uparrow \wedge \Pi \wedge \Gamma, X) = \{Y\uparrow \wedge \Pi \wedge \Gamma' \mid \Gamma' \in \Downarrow(\Gamma, X)\},$$

where $\uplus_{i\in I} X_i$ is the disjoint union of $\{X_i \mid i \in I\}$.

Note that $\psi \wedge X\downarrow$ is equivalent to the disjunction of $\Downarrow(\psi, X)$.

Definition 2.4 (Roots and cells) *We define* $\mathrm{Roots}(\Gamma)$ *as the set of variables which occur at the first argument of some $\mathcal{P}$ in Γ.* $\mathrm{Roots}(\Sigma)$ *is similarly defined. We define* $\mathrm{Cells}(\Gamma)$ *as the set of variables which occur accompanied by $\downarrow$ in Γ.*

2.3 Semantics

We define the following structure: $\mathrm{Val} = N$, $\mathrm{Locs} = \{x \in N | x > 0\}$, $\mathrm{Heaps} = \mathrm{Locs} \to_{fin} \mathrm{Val}^{N_c}$, $\mathrm{Stores} = \mathrm{Vars} \to \mathrm{Val}$. Each $s \in \mathrm{Stores}$ is called a *store*. Each $h \in \mathrm{Heaps}$ is called a *heap*, and $\mathrm{Dom}(h)$ is the domain of h. We write $h = h_1 + h_2$ when $\mathrm{Dom}(h_1)$ and $\mathrm{Dom}(h_2)$ 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 the current memory state. The value $s(x)$ means the value of the variable x in the model (s, h), and it is naturally extended to the domain $\mathrm{Vars} \cup \{\mathrm{nil}\}$ as $s(\mathrm{nil}) = 0$. Each value $a \in \mathrm{Dom}(h)$ means an address, and the value of $h(a)$ is in the memory cell of address a in the model (s, h). We suppose each memory cell has N_c elements as its content. For a formula F we define the interpretation $s, h \models F$ as follows.

$s, h \models t_1 = t_2$ if $s(t_1) = s(t_2)$, $\qquad s, h \models t_1 \neq t_2$ if $s(t_1) \neq s(t_2)$,
$s, h \models x\uparrow$ if $s(x) \notin \mathrm{Dom}(h)$, $\qquad s, h \models x\downarrow$ if $s(x) \in \mathrm{Dom}(h)$,
$s, h \models F_1 \wedge F_2$ if $s, h \models F_1$ and $s, h \models F_2$, $\qquad s, h \models \mathrm{emp}$ if $\mathrm{Dom}(h) = \emptyset$,
$s, h \models t \mapsto (t_1, \ldots, t_{N_c})$ if $\mathrm{Dom}(h) = \{s(t)\}$ and $h(s(t)) = (s(t_1), \ldots, s(t_{N_c}))$,
$s, h \models F_1 * F_2$ if $s, h_1 \models F_1$ and $s, h_2 \models F_2$ for some $h_1 + h_2 = h$,
$s, h \models P^0(\overrightarrow{t})$ does not hold,
$s, h \models P^{k+1}(\overrightarrow{t})$ if $s, h \models \phi[P := P^k]$ for some definition clause ϕ of $P(\overrightarrow{t})$,
$s, h \models P(\overrightarrow{t})$ if $s, h \models P^m(\overrightarrow{t})$ for some m,
$s, h \models \exists z.F$ if $s[z := b], h \models F$ for some $b \in \mathrm{Val}$.

We define $\psi \models \overrightarrow{\phi}$ as $\forall(s, h)(s, h \models \psi$ implies $\exists \phi_i \in \overrightarrow{\phi}(s, h \models \phi_i))$. The entailment $\psi \vdash \overrightarrow{\phi}$ is said to be valid if $\psi \models \overrightarrow{\phi}$ holds.

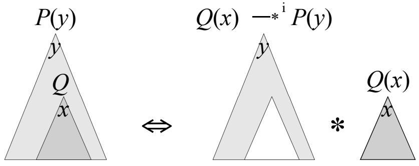


Figure 1. Inductive wand

3 Inductive wand and spatial factorization

In this section, we will define the *inductive wands*, by which we give an inference rule called the *factor rule* for the spatial factorization.

Under the conditions of the cone inductive definitions, a heap model satisfying a predicate and an unfolding tree of the predicate correspond naturally, since any memory cell in a heap satisfying P corresponds to the root of some predicate Q occurring in an unfolding tree of P. Therefore, if a heap model (s, h) satisfies P and a cell with the name x is allocated in h, that is, if $s, h \models P \wedge x\downarrow$, then $s(x)$ is a root of a subheap $h' \subseteq h$ satisfying some Q in the unfolding tree of P. The point is that the difference $h - h'$ can be also represented by some inductive predicate R, whose inductive definition is obtained by removing the occurrence of Q from the definition clauses of P. We call this predicate R the inductive wand $Q \text{—}\!\!*^{\mathrm{i}} P$. Then, $s, h \models x \neq y \wedge P(y) \wedge x\downarrow$ iff $s, h \models (Q(x) \text{—}\!\!*^{\mathrm{i}} P(y)) * Q(x)$ for some predicate Q which can occur in an unfolding tree of P (as Figure 1). By this property, we can factorize a predicate, and expose the cell named with x.

3.1 Inductive wand

We define the inductive wand $Q \text{—}\!\!*^{\mathrm{i}} P$ for inductive predicates P and Q where the definition of P recursively depends on Q. The inductive wand is not introduced as a logical connective, but as an additional inductive predicate. $Q(y) \text{—}\!\!*^{\mathrm{i}} P(x)$ plays a similar role to the ordinary magic wand $Q(y) \text{—}\!\!* P(x)$, but it is stronger than the ordinary one. Note that the class of cone inductive definitions is closed under the inductive wands.

We write Pr for the set of the original predicate symbols, and Pr is extended with inductive wands $Q \text{—}\!\!*^{\mathrm{i}} P$, where $Q \in \mathrm{Pr}$ and P may be another inductive wand. In the following, $P, Q, \cdots$ denote predicate symbols extended by the inductive wand. When the arities of P and Q are n_P and n_Q, respectively, the arity of $Q \text{—}\!\!*^{\mathrm{i}} P$ is $n_P + n_Q$. We write $Q(v, \overrightarrow{w}) \text{—}\!\!*^{\mathrm{i}} P(x, \overrightarrow{y})$ for $(Q \text{—}\!\!*^{\mathrm{i}} P)(x, \overrightarrow{y}, v, \overrightarrow{w})$, and hence the root of $Q(v, \overrightarrow{w}) \text{—}\!\!*^{\mathrm{i}} P(x, \overrightarrow{y})$ is x.

Definition 3.1 *The set* $\mathrm{Dep}(P)$ *of predicate symbols in* Pr *for* $P \in \mathrm{Pr}$ *is the least set satisfying the following: (i) If* $Q \in \mathrm{Pr}$ *occurs in some definition clause of* P*, then* $Q \in \mathrm{Dep}(P)$*. (ii) If* $P_1 \in \mathrm{Dep}(P_2)$ *and* $P_2 \in \mathrm{Dep}(P_3)$*, then* $P_1 \in \mathrm{Dep}(P_3)$*. We extend the domain of* Dep *with the inductive wand by* $\mathrm{Dep}(Q \text{—}\!\!*^{\mathrm{i}} P) = \mathrm{Dep}(P)$.

Definition 3.2 *For* $Q \in \mathrm{Dep}(P)$*, the definition clauses of* $Q(v, \overrightarrow{w}) \text{—}\!\!*^{\mathrm{i}} P(x, \overrightarrow{y})$ *are as follows.*

1. For each $\exists \overrightarrow{z}.\Pi \wedge x \mapsto (\overrightarrow{u}) * Q(z_Q, \overrightarrow{t}_Q) * \Sigma \Rightarrow P(x, \overrightarrow{y})$*, we add*

$$\exists(\overrightarrow{z} - z_Q).(\overrightarrow{w} = \overrightarrow{t}_Q \wedge \Pi \wedge x \mapsto (\overrightarrow{u}) * \Sigma)[z_Q := v].$$

to the definition clauses of $Q(v, \overrightarrow{w}) \text{—}\!\!*^{\mathrm{i}} P(x, \overrightarrow{y})$.

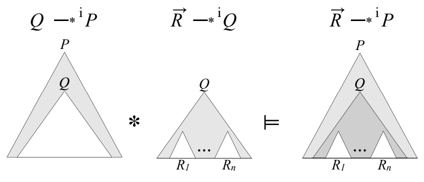
2. For each $\exists \overrightarrow{z}.\Pi \wedge x \mapsto (\overrightarrow{u}) * *_{l \in L} P_l(z_l, \overrightarrow{t}_l) \Rightarrow P(x, \overrightarrow{y})$ *and* $i \in L$ *such that* $Q \in \mathrm{Dep}(P_i)$*, we add*

$$\exists \overrightarrow{z}.\Pi \wedge x \mapsto (\overrightarrow{u}) * (Q(v, \overrightarrow{w}) \text{—}\!\!*^{\mathrm{i}} P_i(z_i, \overrightarrow{t}_i)) * *_{l \neq i, l \in L} P_l(z_l, \overrightarrow{t}_l).$$

to the definition clauses of $Q(v, \overrightarrow{w}) \text{—}\!\!*^{\mathrm{i}} P(x, \overrightarrow{y})$.

If the definition clause contains two or more occurrences of Q*, then we have one definition clause of the form in 1 for each occurrence.*

Inductive-wand elimination (Lemma 3.4):


$Q \mathrel{—\!\!*^{\mathrm{i}}} P$
$\vec{R} \mathrel{—\!\!*^{\mathrm{i}}} Q$
$\vec{R} \mathrel{—\!\!*^{\mathrm{i}}} P$
P
Q
$*$
Q
...
R_1
R_n
$\models$
P
Q
...
R_1
R_n

Inductive-wand introduction (Lemma 3.5):

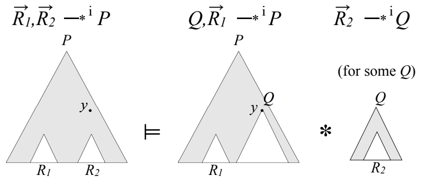

$\vec{R_1},\vec{R_2} \mathrel{—\!\!*^{\mathrm{i}}} P$
$Q,\vec{R_1} \mathrel{—\!\!*^{\mathrm{i}}} P$
$\vec{R_2} \mathrel{—\!\!*^{\mathrm{i}}} Q$
P
y.
R_1
R_2
$\models$
P
y.
Q
R_1
$*$
(for some Q)
Q
R_2

Figure 2. Inductive-wand elimination and introduction

Example 3.3 *For the singly-linked list*

$$\mathrm{ls}(x,y) =_{\mathrm{def}} x \mapsto y \vee \exists z.x \mapsto (z) * \mathrm{ls}(z,y),$$

the inductive wand $\mathrm{ls} \mathrel{—\!\!*^{\mathrm{i}}} \mathrm{ls}$ *is defined by the following clauses.*

$$\begin{aligned}\mathrm{ls}(v,w) \mathrel{—\!\!*^{\mathrm{i}}} \mathrm{ls}(x,y) =_{\mathrm{def}}\ & w = y \wedge x \mapsto (v) \\ & \vee \exists z.x \mapsto (z) * (\mathrm{ls}(v,w) \mathrel{—\!\!*^{\mathrm{i}}} \mathrm{ls}(z,y)).\end{aligned}$$

Here, both clauses are generated from the second clause of ls*: the first clause is added by 1 of Definition 3.2, and the second clause is added by 2.*

The inductive wand is a stronger variant of the magic wand, that is, $P \mathrel{—\!\!*^{\mathrm{i}}} Q \models P \mathrel{—\!\!*} Q$ always holds. However, the converse is not necessarily true. For example, the empty heap satisfies $\mathrm{ls}(x,y) \mathrel{—\!\!*} \mathrm{ls}(x,y)$, but does not satisfy $\mathrm{ls}(x,y) \mathrel{—\!\!*^{\mathrm{i}}} \mathrm{ls}(x,y)$, since any inductive wand is defined by the cone inductive definition, and hence a heap satisfying $\mathrm{ls}(x,y) \mathrel{—\!\!*^{\mathrm{i}}} \mathrm{ls}(x,y)$ contains at least one memory cell. Furthermore, we have no definition clause for $P \mathrel{—\!\!*^{\mathrm{i}}} Q$ when $P \notin \mathrm{Dep}(Q)$, and then $P \mathrel{—\!\!*^{\mathrm{i}}} Q$ means false.

We write $Q_1, \ldots, Q_m \mathrel{—\!\!*^{\mathrm{i}}} P$ for $Q_1 \mathrel{—\!\!*^{\mathrm{i}}} (\cdots \mathrel{—\!\!*^{\mathrm{i}}} (Q_m \mathrel{—\!\!*^{\mathrm{i}}} P) \cdots)$. Intuitively, $Q_1, \ldots, Q_m \mathrel{—\!\!*^{\mathrm{i}}} P$ represents a heap obtained by removing subheaps satisfying $Q_1, \ldots, Q_m$ from a heap satisfying P. We also use the notation $\overrightarrow{Q} \mathrel{—\!\!*^{\mathrm{i}}} P$. If $\overrightarrow{Q}$ is empty, $\overrightarrow{Q} \mathrel{—\!\!*^{\mathrm{i}}} P$ denotes P. We can show that difference of the order of predicates in $\overrightarrow{Q}$ does not change the meaning of $\overrightarrow{Q} \mathrel{—\!\!*^{\mathrm{i}}} P$. Hence, in the following, we consider $\overrightarrow{Q}$ in $\overrightarrow{Q} \mathrel{—\!\!*^{\mathrm{i}}} P$ as a set, and $\overrightarrow{Q}_1 \uplus \overrightarrow{Q}_2$ denotes the disjoint union of $\overrightarrow{Q}_1$ and $\overrightarrow{Q}_2$.

In the following, we show some fundamental property of the inductive wands.

For the inductive wands, the following hold: (i) $(Q(y,\overrightarrow{w}) \mathrel{—\!\!*^{\mathrm{i}}} P(x,\overrightarrow{z})) * Q(y,\overrightarrow{w}) \models P(x,\overrightarrow{z})$, and (ii) $x \neq y \wedge y \downarrow \wedge P(x,\overrightarrow{z}) \models \{\exists \overrightarrow{w}.(Q(y,\overrightarrow{w}) \mathrel{—\!\!*^{\mathrm{i}}} P(x,\overrightarrow{z})) * Q(y,\overrightarrow{w}) \mid Q \in \mathrm{Dep}(P)\}$. Since we want as a precise proof-search procedure as possible, we consider the disjunction of all possible $Q \in \mathrm{Dep}(P)$. These properties are generalized as the following lemmas, which are depicted as Figure 2.

Lemma 3.4 (Inductive-wand elimination) *We have*
$(Q(y,\overrightarrow{w}) \mathrel{—\!\!*^{\mathrm{i}}} P(x,\overrightarrow{z})) * (\overrightarrow{R(v,\overrightarrow{u})} \mathrel{—\!\!*^{\mathrm{i}}} Q(y,\overrightarrow{w})) \models \overrightarrow{R(v,\overrightarrow{u})} \mathrel{—\!\!*^{\mathrm{i}}} P(x,\overrightarrow{z})$.

Lemma 3.5 (Inductive-wand introduction) *We have*
$x \neq y \wedge y \downarrow \wedge (\overrightarrow{R(v,\overrightarrow{u})} \mathrel{—\!\!*^{\mathrm{i}}} P(x,\overrightarrow{z})) \models \{\exists \overrightarrow{w}.(Q(y,\overrightarrow{w}), \overrightarrow{R_1(v_1,\overrightarrow{u_1})} \mathrel{—\!\!*^{\mathrm{i}}} P(x,\overrightarrow{z})) * (\overrightarrow{R_2(v_2,\overrightarrow{u_2})} \mathrel{—\!\!*^{\mathrm{i}}} Q(y,\overrightarrow{w})) \mid$ $\overrightarrow{R} = \overrightarrow{R}_1 \uplus \overrightarrow{R}_2,\ Q \in \mathrm{Dep}(P),\ \overrightarrow{R}_2 \subseteq \mathrm{Dep}(Q)\}$.

3.2 Spatial factorization

The rule (Factor) is the following.

$$\frac{\psi \vdash \overrightarrow{\phi}, \mathrm{Fact}(\phi, y)}{\psi \vdash \overrightarrow{\phi}, \phi} \text{ (Factor)} ,$$

where the auxiliary functions are defined as

$$\begin{aligned}
&\mathrm{Fact}_{\mathrm{aux}}(\overrightarrow{R(u, \overrightarrow{u})} \text{—}*^{\mathrm{i}} P(t, \overrightarrow{t}), y) = \\
&\qquad \{\exists \overrightarrow{w}.(\overrightarrow{R_1(u_1, \overrightarrow{u}_1)}, Q(y, \overrightarrow{w}) \text{—}*^{\mathrm{i}} P(t, \overrightarrow{t})) * (\overrightarrow{R_2(u_2, \overrightarrow{u}_2)} \text{—}*^{\mathrm{i}} Q(y, \overrightarrow{w})) \mid \\
&\qquad \overrightarrow{R(u, \overrightarrow{u})} = \overrightarrow{R_1(u_1, \overrightarrow{u}_1)} \uplus \overrightarrow{R_2(u_2, \overrightarrow{u}_2)},\ Q \in \mathrm{Dep}(P),\ \overrightarrow{R}_2 \subseteq \mathrm{Dep}(Q),\ \overrightarrow{w} \text{ fresh}\}, \\
&\mathrm{Fact}(\exists \overrightarrow{v}.\Pi \wedge *_{j \in J} \mathcal{P}_j, y) = \{\exists \overrightarrow{v} \overrightarrow{w}.\Pi \wedge (*_{j \in J - \{i\}} \mathcal{P}_j) * \Sigma \mid \\
&\qquad\qquad i \in J,\ \mathcal{P}_i \text{ is not } \mapsto,\ \exists \overrightarrow{w}.\Sigma \in \mathrm{Fact}_{\mathrm{aux}}(\mathcal{P}_i, y).\}
\end{aligned}$$

When $\overrightarrow{R}$ is empty, the definition of $\mathrm{Fact}_{\mathrm{aux}}$ means

$$\mathrm{Fact}_{\mathrm{aux}}(P(t, \overrightarrow{t}), y) = \{\exists \overrightarrow{w}.(Q(y, \overrightarrow{w}) \text{—}*^{\mathrm{i}} P(t, \overrightarrow{t})) * Q(y, \overrightarrow{w}) \mid Q \in \mathrm{Dep}(P),\ \overrightarrow{w} \text{ fresh}\},$$

that is, we can factorize P by (Factor) into two parts Q—$*^{\mathrm{i}}P$ and Q and we can find the hidden root y.

The soundness of (Factor) is proved by Lemma 3.4.

Proposition 3.6 (Soundness of factor) *If* $\psi \models \overrightarrow{\phi}, \mathrm{Fact}(\phi, y)$*, then* $\psi \models \overrightarrow{\phi}, \phi$.

4 Inference rules and proof search

4.1 Proof system

The inference rules are summarized in Figure 3. The rule (emp) is an axiom, where the spatial parts on both sides are emp. The conclusion is always valid since the set Val is infinite. For the rule (Unsat), note that the satisfiability problem for the extended language is decidable [28]. The rule (Case$\uparrow\downarrow$) is for case analysis on $x\uparrow$ and $x\downarrow$. For $x\downarrow$, we also distribute $x\downarrow$ over the $*$-conjunction. This rule is sound since $\models x\uparrow \vee x\downarrow$ always holds. The rule (CaseEq) is for case analysis on $=$ and $\neq$. The rule ($= L$) removes substitution by equality. The rules ($\wedge L_{\uparrow}$), ($\wedge L_{\downarrow}$), and ($\wedge L_p$) are usual $\wedge$-introduction rules for antecedent. The rule ($\exists R$) is the usual $\exists$-introduction rule for succedent. The rule (Case$R$) is for case analysis in the succedent. (Factor) is for the spatial factorization, and the soundness is proved in the previous section. The rule (Pred$L$) is for folding and case analysis for a predicate P in the antecedent. Here, we also consider the case analysis on distribution of $X\downarrow$ to the spatial atoms in Σ. In the rule (Pred$R$), we fold subformulas in the succedent matched to definition clauses of the predicate P. Note that we consider all the definition clauses in the succedent as the disjuncts. The rule ($\mapsto=$) is a usual $\wedge$-elimination rule. The rule ($*\mapsto$) adds $\mapsto$-predicates to both sides. For any (s, h), $s, h \models \psi * x \mapsto (\overrightarrow{u})$ implies that the subheap of h satisfying ψ does not contain the location x, and hence it satisfies $x\uparrow$. Therefore, this rule is sound.

We use the same bud-companion relation in ordinary cyclic proofs [7], but for the global trace condition, we count the number of the rule ($*\mapsto$) instead of the case rule.

Definition 4.1 (Proofs) *1. A* pre-proof *is a finite proof tree constructed by the rules in Fig. 3 associated with a mapping assigning each non-axiom leaf, called a bud, to an inner node with the same entailment, called a companion.*

2. A proof *is defined as a pre-proof such that, for any bud, its companion lies on the path from the bud to the root of the pre-proof, and every path segment between a bud and its companion contains at least one rule instance of ($*\mapsto$).*

Theorem 4.2 (Soundness) *If* $\psi \vdash \overrightarrow{\phi}$ *holds, then we have* $\psi \models \overrightarrow{\phi}$.

$$\frac{(\Pi' \text{ contains only } x \neq t \text{ for } x \in \overrightarrow{x} \text{ and } x \not\equiv t)}{\mathrm{emp} \vdash \exists \overrightarrow{x}.\Pi' \wedge \mathrm{emp}, \overrightarrow{\phi}}\ (\mathrm{emp}) \qquad \frac{(\psi \quad \text{unsat})}{\psi \vdash \overrightarrow{\phi}}\ (\mathrm{Unsat})$$

$$\frac{x\uparrow \wedge \psi \vdash \overrightarrow{\phi} \quad \psi' \vdash \overrightarrow{\phi} \quad (\forall \psi' \in \Downarrow(\psi, \{x\}))}{\psi \vdash \overrightarrow{\phi}}\ (\mathrm{Case}\uparrow\downarrow) \qquad \frac{t = t' \wedge \psi \vdash \overrightarrow{\phi} \quad t \neq t' \wedge \psi \vdash \overrightarrow{\phi}}{\psi \vdash \overrightarrow{\phi}}\ (\mathrm{CaseEq})$$

$$\frac{\psi[x := t] \vdash \overrightarrow{\phi}[x := t]}{x = t \wedge \psi \vdash \overrightarrow{\phi}}\ (= L) \quad \frac{\psi \vdash \overrightarrow{\phi}}{x\uparrow \wedge \psi \vdash \overrightarrow{\phi}}\ (\wedge L_{\uparrow}) \quad \frac{\psi * \mathcal{P} \vdash \overrightarrow{\phi}}{\psi * (\mathcal{P} \wedge x\downarrow) \vdash \overrightarrow{\phi}}\ (\wedge L_{\downarrow}) \quad \frac{\psi \vdash \overrightarrow{\phi}}{\Pi \wedge \psi \vdash \overrightarrow{\phi}}\ (\wedge L_p)$$

$$\frac{\psi \vdash \overrightarrow{\phi}}{\psi \vdash \overrightarrow{\phi}, \phi}\ (\vee R) \quad \frac{\psi \vdash \overrightarrow{\phi}, \phi[w := t]}{\psi \vdash \overrightarrow{\phi}, \exists w.\phi}\ (\exists R) \quad \frac{t \neq t' \wedge \psi \vdash \overrightarrow{\phi}, \phi}{t \neq t' \wedge \psi \vdash \overrightarrow{\phi}, t \neq t' \wedge \phi}\ (\wedge R) \quad \frac{\psi \vdash \overrightarrow{\phi}, \phi}{\psi \vdash \overrightarrow{\phi}, t = t \wedge \phi}\ (\mathrm{Triv}R)$$

$$\frac{\psi \vdash \exists \overrightarrow{w}.t \neq t' \wedge \Pi \wedge \Sigma, \overrightarrow{\phi} \quad \psi \vdash \exists \overrightarrow{w}.t = t' \wedge \Pi \wedge \Sigma, \overrightarrow{\phi}}{\psi \vdash \exists \overrightarrow{w}.\Pi \wedge \Sigma, \overrightarrow{\phi}}\ (\mathrm{Case}R)$$

$$\frac{\psi \vdash \overrightarrow{\phi} \quad (\theta \text{ is a substitution})}{\psi\theta \vdash \overrightarrow{\phi}\theta}\ (\mathrm{Subst}) \qquad \frac{\psi \vdash \overrightarrow{\phi}, \mathrm{Fact}(\phi, y)}{\psi \vdash \overrightarrow{\phi}, \phi}\ (\mathrm{Factor})$$

$$\frac{\Pi \wedge \psi * \Gamma \vdash \overrightarrow{\phi} \quad (\forall \Gamma \in \cup\{\Downarrow(\Sigma, X) \mid \exists \overrightarrow{z}.\Pi \wedge \Sigma \Rightarrow P(x),\ \overrightarrow{z} \text{ is fresh}\})}{\psi * (P(x) \wedge X\downarrow) \vdash \overrightarrow{\phi}}\ (\mathrm{Pred}\ L)$$

$$\frac{\psi \vdash \overrightarrow{\phi}, \{\exists \overrightarrow{v}\overrightarrow{z}.\Pi \wedge \Pi' \wedge \Sigma * \Sigma' \mid \exists \overrightarrow{z}.\Pi' \wedge \Sigma' \Rightarrow P(x, \overrightarrow{t})\}}{\psi \vdash \overrightarrow{\phi}, \exists \overrightarrow{v}.\Pi \wedge \Sigma * P(x, \overrightarrow{t})}\ (\mathrm{Pred}\ R)$$

$$\frac{\psi * x \mapsto (\overrightarrow{u}) \vdash \overrightarrow{\phi}, \exists \overrightarrow{z}.\Pi \wedge \overrightarrow{u} = \overrightarrow{v} \wedge \Sigma * x \mapsto (\overrightarrow{v})}{\psi * x \mapsto (\overrightarrow{u}) \vdash \overrightarrow{\phi}, \exists \overrightarrow{z}.\Pi \wedge \Sigma * x \mapsto (\overrightarrow{v})}\ (\mapsto=) \qquad \frac{x\uparrow \wedge \psi \vdash (\exists z_i.\Pi_i \wedge \Sigma_i)_i}{\psi * x \mapsto (\overrightarrow{u}) \vdash (\exists z_i.\Pi_i \wedge \Sigma_i * x \mapsto (\overrightarrow{u}))_i}\ (* \mapsto)$$

The operation $\Downarrow$ in (Case$\uparrow\downarrow$) and (Pred$L$) is introduced in Definition 2.3. The definition of Fact in (Factor) is given in Section 3.2.

Figure 3. Inference Rules

4.2 Proof-search procedure

In the following subsections, we describe our proof-search procedure, which follows the Unfold-Match-Remove strategy. Our proof-search procedure consists of two parts: first, we apply the initialization phase, given in Section 4.3, and then repeat the main phase, given in Section 4.4.

In the initialization phase, we do some case analyses (steps 1 and 2), some trivial termination checks (steps 3 and 6), and normalization (steps 4, 5 and 7). After the initialization, we obtain some entailments $\overrightarrow{J}$ for a given input.

The inputs and the outputs of the main phase are sets of pairs (J, H) of an entailment J, called a subgoal, and a set H of entailments, called a history. For the output $\overrightarrow{J}$ of the initialization, we run the main phase with the input $\{(J, \{J\}) \mid J \in \overrightarrow{J}\}$. In the main phase, we unfold predicates on both sides with a common root. If we cannot find any common root, we apply the factor rule (step 9), and then we obtain a common root. At the end of the main phase (step 15), we check a cycle by comparing the subgoal and the entailments in the history. If we find a companion, we finish this subgoal. For each remaining subgoal, we add the subgoal to the history, and repeat the main phase. Every subgoal in outputs of the initialization and the main phases is in some normal form, which is defined in the next section.

If all subgoals successfully finish in the unsat check (step 3), the termination check (step 6), or the cycle check (step 15), the procedure stops and answers VALID. If one of subgoals stops with FAIL in the termination check (step 6) or the succedent of one of the subgoals becomes empty by the rule $(\vee R)$ at the disjunct root check (step 5), the procedure stops and answers FAIL.

4.3 Initialization

The initialization phase consists of the following steps 1–7. Let J_0 be the input entailment.

1. **[Case analysis** $\uparrow$ / $\downarrow$**]** For each $x \in \mathrm{FV}(J_0)$, repeat (Case$\uparrow\downarrow$) to make cases with respect

to $x \uparrow$ and $x \downarrow$. Then, we obtain subgoals $\overrightarrow{J}_1$.

2. [**Case analysis** $= / \neq$] For each $J \in \overrightarrow{J}_1$, repeat (CaseEq) to make cases with respect to $t = t'$ and $t \neq t'$ for all pairs of elements in $\mathrm{FV}(J) \cup \{\mathrm{nil}\}$ except for $t \equiv t'$. Then, we obtain subgoals $\overrightarrow{J}_2$.

3. [**Unsat check**] For each $\psi \vdash \overrightarrow{\phi} \in \overrightarrow{J}_2$, if ψ is unsatisfiable, finish this case by applying (Unsat). Let $\overrightarrow{J}_3$ be the remaining subgoals.

4. [**Pure check**] For each subgoal $\psi \vdash \overrightarrow{\phi} \in \overrightarrow{J}_3$, do the following 4.1–4.3. Then, we obtain subgoals $\overrightarrow{J}_4$.

4.1. Repeat $(= L)$ for each $x = y$ in ψ until the antecedent contains no $=$. Let $J_{4.1}$ be the result. Then, the pure part of the antecedent of $J_{4.1}$ is $(\neq (\mathrm{FV}(J_{4.1}) \cup \{\mathrm{nil}\}))$.

4.2. For every $\exists x.x = t \wedge \Pi \wedge \Sigma$ in the succedent of $J_{4.1}$, replace it to $(\Pi \wedge \Sigma)[x := t]$ by $(\exists R)$. Then, repeat $(\wedge R)$, (Triv$R$) until these rules cannot be applied. Let $J_{4.2}$ be the result.

4.3. Let $\psi \vdash \overrightarrow{\phi} \equiv J_{4.2}$. Remove all $\phi \in \overrightarrow{\phi}$ containing $t = t'$ such that $t \not\equiv t'$ by repeating $(\vee R)$. Then, the pure parts in the succedent contain only $x \neq t'$ such that x is bound by $\exists$.

5. [**Disjunct root check**] For each $\psi \vdash \overrightarrow{\phi} \in \overrightarrow{J}_4$, repeat $(\vee R)$ to remove all $\phi \in \overrightarrow{\phi}$ such that either (1) ϕ contains two or more spatial atoms with the same root, (2) ϕ contains a root variable which is not in $\mathrm{Roots}(\Gamma) \cup \mathrm{Cells}(\Gamma)$, or (3) ϕ contains a predicate whose root is nil. Let $\overrightarrow{J}_5$ be the resulting subgoals.

6. [**Termination check**] For each $\psi \vdash \overrightarrow{\phi} \in \overrightarrow{J}_5$ such that the spatial part of ψ is emp, check the following. If there is $\phi \in \overrightarrow{\phi}$ whose spatial part is emp, finish this case successfully by applying $(\wedge L_{\downarrow})$, $(\wedge L_p)$, and (emp). Otherwise, stop the procedure with FAIL. Let $\overrightarrow{J}_6$ be the remaining subgoals.

7. [**Normalization**] For each subgoal $\psi \vdash \overrightarrow{\phi} \in \overrightarrow{J}_6$, do the following 7.1–7.4. Then we obtain subgoals $\overrightarrow{J}_7$.

7.1. Let Γ be the spatial part of ψ. For each $\phi = \exists \overrightarrow{w}.\Pi \wedge \Sigma \in \overrightarrow{\phi}$, $w \in \overrightarrow{w}$, and $t \in \mathrm{FV}(\Gamma) \cup \{\mathrm{nil}\}$ such that $w \neq t \notin \Pi$, replace ϕ to two disjuncts $\exists \overrightarrow{w}.w \neq t \wedge \Pi \wedge \Sigma$ and $\exists \overrightarrow{w} - w.(\Pi \wedge \Sigma)[w := t]$ by (Case$R$), $(\exists R)$, and (Triv$R$). Repeat it until $\Pi = (\neq (\mathrm{FV}(\Gamma) \cup \overrightarrow{w} \cup \{\mathrm{nil}\}, \overrightarrow{w}))$ holds for every disjunct $\exists \overrightarrow{w}.\Pi \wedge \Sigma$. Then, we obtain a subgoal $J_{7.1}$.

7.2. Let $Y \uparrow \wedge \Pi \wedge \Gamma \vdash \{\exists \overrightarrow{w}_i.(\Pi_i \wedge \Sigma_i)\}_i \equiv J_{7.1}$. Repeat $(\wedge L_p)$ and $(\wedge L_{\uparrow})$ to remove all $x \uparrow \in Y \uparrow$ and $x \neq t \in \Pi$ such that $x \notin \mathrm{FV}(\Gamma, \{\Sigma_i\}_i)$. Then, for each $\mathcal{P}(x, \overrightarrow{t}) \wedge x \downarrow$ in Γ, remove $x \downarrow$ by $(\wedge L_{\downarrow})$. Then, we obtain a subgoal $J_{7.2}$.

7.3. Repeat $(\exists R)$ to remove all $\exists x$ in the succedent of $J_{7.2}$ such that x does not occur in its body. Then, we obtain a subgoal $J_{7.3}$.

7.4. Let $Y \uparrow \wedge \Pi \wedge \Gamma \vdash \overrightarrow{\phi} \equiv J_{7.3}$. For each pair of disjuncts $\phi, \phi' \in \overrightarrow{\phi}$, if it satisfies the following condition (*), remove ϕ' by applying $(\vee R)$: (*) there is a variable renaming θ such that $\phi' \equiv \phi\theta$, $\mathrm{Dom}(\theta) \subseteq Y$, and $\mathrm{Dom}(\theta) \cap \mathrm{FV}(\Gamma) = \emptyset$. Repeat this step until there is no such pair.

4.4 Main loop

After the initialization, we repeat the main phase, which consists of the following steps 8–15. For the output $\overrightarrow{J}$, we run the main phase with the input $\{(J, \{J\}) \mid J \in \overrightarrow{J}\}$. During the main phase, when two or more subgoals are generated from one subgoal, the history is copied to each subgoal. In the following, we omit the histories since they are changed only at the last step (step 15) of the main phase.

For every path of a pre-proof constructed in the main phase, if the procedure does not fail during processing the path, either it reaches a leaf or it contains a rule instance of $(* \mapsto)$ (step 13). Hence, if we obtain a pre-proof by the procedure, it is a proof.

Let $\overrightarrow{J}_0$ be the subgoals of the input of the main phase.

8. [**Common root**] For each $\psi \vdash (\phi_i)_i \in \overrightarrow{J}_0$, if the set $\mathrm{Roots}(\psi) \cap (\cap_i \mathrm{Roots}(\phi_i))$ not empty, choose x in this set and skip the step 9 for this subgoal. Otherwise, choose $x \in \mathrm{Roots}(\psi)$ and do

the step 9 for this subgoal.

9. [**Spatial factorization**] For each $\psi \vdash (\phi_i)_i \in \overrightarrow{J}_0$ such that we chose $x \in \mathrm{Roots}(\psi)$ in the latter case of step 8, apply (Factor) for each ϕ_i such that $x \notin \mathrm{Roots}(\phi_i)$. Then, we obtain the subgoals $\overrightarrow{J}_9$. Note that, for each $\psi \vdash (\phi_i)_i \in \overrightarrow{J}_9$, we have $x \in \mathrm{Roots}(\psi) \cap (\cap_i \mathrm{Roots}(\phi_i))$.

10. [**Unfolding**] For each $J \in \overrightarrow{J}_9$, apply (Pred$L$) to the predicate with the root x in the antecedent, and then we obtain subgoals $\overrightarrow{J}'_{10}$. For each $J \in \overrightarrow{J}'_{10}$, apply (Pred$R$) to all of the predicates with the root x in the disjuncts of the succedent to generate new disjuncts. Then, we obtain the subgoals $\overrightarrow{J}_{10}$.

11. [**Matching**] For each $J \in \overrightarrow{J}_{10}$, apply ($\mapsto$=) for all disjuncts. Then, we obtain the subgoals $\overrightarrow{J}_{11}$.

12. Let $\overrightarrow{J}_1$ be $\overrightarrow{J}_{11}$, and do the steps 2–5 of the initialization. Then, we obtain the subgoals $\overrightarrow{J}_{12}$.

13. [**Removing**] Note that, for any $\psi \vdash (\phi_i)_i \in \overrightarrow{J}_{12}$, every $x \mapsto \overrightarrow{t}$ in ϕ is identical to the $x \mapsto \overrightarrow{u}$ in ψ. Remove these $\mapsto$ predicates by ($\mapsto *$), and add $x \uparrow$ in the antecedent. Then, we obtain the subgoals $\overrightarrow{J}_{13}$

14. Let $\overrightarrow{J}_5$ be $\overrightarrow{J}_{13}$, and do the steps 6–7 of the initialization phase. Then, we obtain the subgoals $\overrightarrow{J}_{14}$.

15. [**Cycle check**] For each $J \in \overrightarrow{J}_{14}$, if J is obtained by substitution from an entailment in the history, finish this case successfully. Otherwise, add J to the history.

Then, repeat the main phase with the remaining subgoals.

4.5 An example of proof search

We give an example to see how our procedure works.

Example 4.3 *We show how the entailment* $\mathrm{dll}(h,t,p,n) \vdash \mathrm{dllrev}(t,h,n,p)$ *is proved.*

The initialization produces many cases, and we consider one of the most typical cases as the input of the main loop, which is the following.

$$(\neq \{h,t,\mathrm{nil}\}) \wedge \mathrm{dll}(h,t,\mathrm{nil},\mathrm{nil}) \wedge t \downarrow \vdash \mathrm{dllrev}(t,h,\mathrm{nil},\mathrm{nil}).$$

There is no common root, so we apply (Factor), and then we have

$$\begin{aligned}&(\neq \{h,t,\mathrm{nil}\}) \wedge \mathrm{dll}(h,t,\mathrm{nil},\mathrm{nil}) \wedge t \downarrow \vdash \\ &\qquad \exists t'p'n'.(\mathrm{dllrev}(h,t',p',n') \text{—}*^{\mathrm{i}} \mathrm{dllrev}(t,h,\mathrm{nil},\mathrm{nil})) * \mathrm{dllrev}(h,t',p',n').\end{aligned}$$

By (PredL), we obtain two cases, but the case for the first definition clause of dll *is unsat, so we omit this case. By (PredR), we obtain*

$$\begin{aligned}&(\neq \{h,t,\mathrm{nil}\}) \wedge h \mapsto (\mathrm{nil},z) * \mathrm{dll}(z,t,h,\mathrm{nil}) \wedge t \downarrow \vdash \\ &\qquad \exists t'p'n'.h = t' \wedge (\mathrm{dllrev}(h,t',p',n') \text{—}*^{\mathrm{i}} \mathrm{dllrev}(t,h,\mathrm{nil},\mathrm{nil})) * h \mapsto (n',p'), \\ &\qquad \exists t'p'n'z'_1.(\mathrm{dllrev}(h,t',p',n') \text{—}*^{\mathrm{i}} \mathrm{dllrev}(t,h,\mathrm{nil},\mathrm{nil})) * h \mapsto (z'_1,p') * \mathrm{dllrev}(z'_1,t',h,n').\end{aligned}$$

By matching, removing, and normalization (steps 11–14), we have the following.

$$\begin{aligned}&h \uparrow \wedge (\neq \{h,t,\mathrm{nil}\}) \wedge \mathrm{dll}(t,t,h,\mathrm{nil}) \vdash \mathrm{dllrev}(h,h,t,\mathrm{nil}) \text{—}*^{\mathrm{i}} \mathrm{dllrev}(t,h,\mathrm{nil},\mathrm{nil}), \\ &h \uparrow \wedge (\neq \{h,t,z,\mathrm{nil}\}) \wedge \mathrm{dll}(z,t,h,\mathrm{nil}) \wedge t \downarrow \vdash \mathrm{dllrev}(h,h,z,\mathrm{nil}) \text{—}*^{\mathrm{i}} \mathrm{dllrev}(t,h,\mathrm{nil},\mathrm{nil}).\end{aligned}$$

The procedure will halt with the answer VALID as follows. The first subgoal is proved without induction, since $\mathrm{dll}(t,t,h,\mathrm{nil})$ *is equivalent to* $t \mapsto (h,\mathrm{nil})$*, which implies the succedent. For the* *second subgoal, (Factor) is applied again during the next execution of the main loop. Then, we obtain*

$$\exists t'p'n'.(\mathrm{dllrev}(z,t',p',n') \text{—}*^{\mathrm{i}} \mathrm{dllrev}(t,h,\mathrm{nil},\mathrm{nil})) * (\mathrm{dllrev}(h,h,z,\mathrm{nil}) \text{—}*^{\mathrm{i}} \mathrm{dllrev}(z,t',p,n')),$$

as a disjunct in the succedent, and unfold the second inductive wand. After some repetitions of the main loop (the repeat count is 5 in the experiment shown in Section 6), we will find a cycle with the bud entailment

$$\{z',h\}\uparrow \wedge(\neq \{h,t,z',v,\mathrm{nil}\}) \wedge \mathrm{dll}(v,t,z',\mathrm{nil}) \wedge t \downarrow \vdash \mathrm{dllrev}(z',h,v,\mathrm{nil}) \text{—}*^{\mathrm{i}} \mathrm{dllrev}(t,h,\mathrm{nil},\mathrm{nil}),$$

where $\mathrm{dll}(v,t,z',\mathrm{nil})$ *represents heaps obtained by removing some "first" cells (from h to z′) from* $\mathrm{dll}(h,t,\mathrm{nil},\mathrm{nil})$*, and it corresponds to* $\mathrm{dllrev}(z',h,v,\mathrm{nil}) \text{—}*^{\mathrm{i}} \mathrm{dllrev}(t,h,\mathrm{nil},\mathrm{nil})$*, which represents heaps obtained by removing some "last" cells (from z′ to h) from* $\mathrm{dllrev}(t,h,\mathrm{nil},\mathrm{nil})$*.*

5 Completeness and decidability for linear cone inductive predicates

In this section, we show that our proof system is complete and decidable when we restrict inductive predicates to linear ones. A cone inductive definition is said to be *linear* if its definition clauses are of the form either $\Pi \wedge x \mapsto (\overrightarrow{t})$ or $\exists z.(\Pi \wedge x \mapsto (\overrightarrow{t}) * Q(z,\overrightarrow{u}))$ with $z \in \overrightarrow{t}$. All of the inductive predicates mentioned in this paper, including the singly-linked and the doubly-linked lists, are linear. It is easy to see that the class of the linear cone inductive definitions is closed under construction of the inductive wands.

In the following, completeness and decidability of the proof system for the linear cone inductive predicates are proved.

We prove the completeness and the decidability by the following: (1) each rule instance in the proof search is locally complete, that is, if the conclusion of the rule instance is valid, then every assumption is valid (Lemma 5.3), and (2) the number of entailments which can occur as inputs of the main phase during the proof search is finite. By these properties, the proof search does not get stuck and eventually halts (Lemma 5.6). The possible entailments which can be inputs of the main phase are normal forms defined in Definition 5.4.

The restriction of the linearity plays important roles in both of them. For (1), we can limit the depth of inductive wands, which is shown in the following Lemma 5.1. Note that this is not a special case of Lemma 3.5 since here we need not consider the case of $(R, Q \text{—}*^{\mathrm{i}} P) * Q$ in the succedent. For (2), the unfolding steps do not increase the size of spatial parts in entailments. In order to achieve a complete and decidable proof system for general cone inductive predicates which are not restricted to linear ones, we need many other mechanisms such as an elaborately generalized split rule [28].

Lemma 5.1 (Inductive-wand introduction for linear cone inductive predicates) *$x \neq y \wedge y \downarrow \wedge (R(v,u) \text{—}*^{\mathrm{i}} P(x,z)) \models \{\exists w((Q(y,w) \text{—}*^{\mathrm{i}} P(x,z)) * (R(v,u) \text{—}*^{\mathrm{i}} Q(y,w)) \mid Q \in \mathrm{Dep}(P), R \in \mathrm{Dep}(Q)\}$.*

This lemma leads us to a restricted variant of (Factor) rule:

$$\frac{\psi \vdash \overrightarrow{\phi}, \text{1-Fact}(\phi, y)}{\psi \vdash \overrightarrow{\phi}, \phi}\ \text{(1-Factor)},$$

where 1-Fact(ϕ, y) is the same as Fact(ϕ, y) defined in the section 3.2 except the length of $\overrightarrow{R}$ in every $\overrightarrow{R} \text{—}*^{\mathrm{i}} Q$ is at most 1.

Definition 5.2 *The* proof system for the linear cone inductive predicates *is defined as the same as the proof system defined in Section 4.1 except the cone inductive definitions are restricted to linear cone inductive definitions and (*Factor*) is replaced to (*1-Factor*).*

The rule (1-Factor) is a combination of (Factor) and ($\vee R$) since 1-Fact(ϕ, y) is a subset of Fact(ϕ, y). Hence, the proof system for the linear cone inductive predicates is sound.

Lemma 5.3 *For every application of inference rules in the proof-search procedure, if the input of each step is valid, then every output of the step is valid.*

Proof. The nontrivial cases are the steps 7.4 and 9. The proof for the step 7.4 is omitted. For 9, the local completeness of (1-Factor) follows from Lemmas 3.5 and 5.1. □

We define the normal forms as follows. The outputs of the initialization and the main phase are the normal forms, and the number of normal forms which can occur in the proof search is finite up to renaming.

Definition 5.4 (Normal forms) *An entailment* $J \equiv Y \uparrow \wedge\Pi \wedge \Gamma \vdash \{\exists\overrightarrow{w}_i.(\Pi_i \wedge \Sigma_i)\}_{i\in I}$ *is a* normal form *iff the following hold.*

1. $\mathrm{FV}(J) = Y + \mathrm{Roots}(\Gamma) + \mathrm{Cells}(\Gamma)$.
2. $\mathrm{Roots}(\Sigma_i) \subseteq \mathrm{Roots}(\Gamma) + \mathrm{Cells}(\Gamma)$.
3. $Y \subseteq \mathrm{FV}(\Gamma, \{\exists\overrightarrow{w}_i.\Sigma_i\}_{i\in I})$.
4. $\overrightarrow{w}_i \subseteq \mathrm{FV}(\Pi_i \wedge \Sigma_i)$.
5. $\Pi = (\neq (\mathrm{FV}(J) \cup \{\mathrm{nil}\}))$.
6. $\Pi_i = (\neq (\mathrm{FV}(\Gamma) \cup \{\overrightarrow{w}_i, \mathrm{nil}\}, \overrightarrow{w}_i))$.
7. For $i \neq j \in I$, $(\exists\overrightarrow{w}_i(\Pi_i \wedge \Sigma_i))\theta \not\equiv \exists\overrightarrow{w}_j(\Pi_j \wedge \Sigma_j)$ *for any renaming* θ *such that* $\mathrm{Dom}(\theta) \cap \mathrm{FV}(\Gamma) = \emptyset$.

Lemma 5.5 *Let* $J \equiv Y \uparrow \wedge\Pi \wedge \Gamma \vdash \overrightarrow{\phi}$ *be an initial input of the proof-search procedure. For any input entailment* $J' \equiv Y' \uparrow \wedge\Pi' \wedge \Gamma' \vdash \overrightarrow{\phi}'$ *of each main phase in the proof search, we have the following.*

1. J' *is a normal form.*
2. $|\mathrm{Roots}(\Gamma)| \geq |\mathrm{Roots}(\Gamma')|$ *and* $\mathrm{Cells}(\Gamma) \supseteq \mathrm{Cells}(\Gamma')$.
3. $|\overrightarrow{R}| \leq 1$ *for every inductive wand* $\overrightarrow{R} \mathrel{-\!\!*}^{\mathrm{i}} P$ *in* $\overrightarrow{\phi}'$.

Lemma 5.6 *The number of normal forms which can appear for the cycle check of the main phase in the proof search is finite up to renaming.*

By these Lemmas, we conclude completeness of the proof system for the linear cone inductive predicates and that our proof-search procedure decides validity of entailments.

Theorem 5.7 (Completeness and decidability) *1. For the proof system for the linear cone inductive predicates, the procedure decides the validity of a given entailment. Namely, For a given input* J*, the procedure terminates without failure when the input is valid, and it fails when the input is invalid.*

2. The proof system for the linear cone inductive predicates is complete. Namely, every valid entailment is provable.

Proof. First, we show that the procedure terminates for any input. Assume the procedure does not terminate for some input. By Lemma 5.6, we have some upper bound n for the number of normal forms. The cycle-check step in the main loop increases the size of the history. After the $(n + 1)$-th execution of the main loop, the size of the history becomes more than $n + 1$. Since

#	input entailment	result	time (sec.)	depth
1*	$\mathrm{two}(x,y) \vdash \mathrm{two}(y,x)$	VALID	0.02	2
2*	$\mathrm{tri}(x,y,z) \vdash \mathrm{tri}(y,z,x)$	VALID	0.54	3
3*	$\mathrm{tri}(x,y,z) \vdash \mathrm{tri}(z,x,y)$	VALID	0.57	3
4	$\mathrm{ls}(x,y) * \mathrm{ls}(y,\mathrm{nil}) \vdash \mathrm{ls}(x,\mathrm{nil})$	VALID	0.01	1
5	$\mathrm{ls}(x,y) * \mathrm{ls}(y,z) \vdash \mathrm{ls}(x,z)$	VALID	0.12	4
6*	$\mathrm{ls}(x,x) \wedge y \downarrow \vdash \mathrm{ls}(y,y)$	VALID	0.09	5
7	$\mathrm{lsa}(x,y,z) \vdash \mathrm{ls}(x,y)$	VALID	3.05	5
8	$\mathrm{ls}(x,y) \vdash \mathrm{lsa}(x,y,x)$	VALID	0.44	8
9	$\mathrm{ls}(x,y) \vdash \mathrm{lsa}(x,y,y)$	INVALID	0.08	-
10	$x \neq y \wedge \mathrm{lsa}(x,y,y) \vdash \mathrm{ls}(x,y) * \mathrm{ls}(y,y)$	VALID	0.11	3
11	$\mathrm{lsa}(x,y,y) \vdash \mathrm{ls}(x,y) * \mathrm{ls}(y,y)$	INVALID	0.09	-
12*	$\mathrm{lsa}(x,x,y) \vdash \mathrm{lsa}(y,y,x)$	VALID	25.27	9
13	$z \uparrow \wedge \mathrm{lsn}(x,y) * \mathrm{lsn}(y,z) \vdash \mathrm{lsn}(x,z)$	VALID	0.03	2
14	$\mathrm{lsn}(x,y) * \mathrm{lsn}(y,z) * \mathrm{to}(z,\mathrm{nil}) \vdash \mathrm{lsn}(x,z) * \mathrm{to}(z,\mathrm{nil})$	VALID	0.03	2
15	$\mathrm{lsn}(x,y) * \mathrm{lsn}(y,z) \vdash \mathrm{lsn}(x,z)$	INVALID	0.16	-
16	$x \neq z \wedge \mathrm{lsn}(x,y) * \mathrm{lsn}(y,z) \vdash \mathrm{lsn}(x,z)$	INVALID	0.09	-
17	$\mathrm{dll}(h,t,p,n) * \mathrm{dll}(n,p,t,h) \vdash \mathrm{dll}(h,p,p,h)$	VALID	23.05	5
18	$\mathrm{dll}(h,t,p,n) * \mathrm{dll}(n,p,t,h) \vdash \mathrm{dll}(h,p,h,p)$	INVALID	21.17	-
19*	$\mathrm{dll}(h,t,\mathrm{nil},\mathrm{nil}) \vdash \mathrm{dllrev}(t,h,\mathrm{nil},\mathrm{nil})$	VALID	0.73	5
20*	$p \uparrow \wedge \mathrm{dll}(h,t,p,n) \vdash \mathrm{dllrev}(t,h,n,p)$	VALID	47.06	5

(* proved with the factor rule)

Table 1. Experimental result

descriptions	predicates	definition clauses
point to	$\mathrm{to}(x,y)$	$x \mapsto (y)$
two cells linked to each other	$\mathrm{two}(x,y)$	$\exists z.y = z \wedge x \mapsto (z) * \mathrm{to}(z,x)$
cyclic three cells	$\mathrm{tri}(x,y,z)$	$\exists w.y = w \wedge x \mapsto (w) * \mathrm{tri1}(w,z,x)$
	$\mathrm{tri1}(y,z,x)$	$\exists w.z = w \wedge y \mapsto (w) * \mathrm{to}(w,x)$
list	$\mathrm{ls}(x,y)$	$x \mapsto (y)$
		$\vee \exists z.x \mapsto (z) * \mathrm{ls}(z,y)$
list with an allocated cell	$\mathrm{lsa}(x,y,z)$	$x = z \wedge x \mapsto (y)$
		$\vee \exists w.x = z \wedge x \mapsto (w) * \mathrm{lsa}(w,y,w)$
		$\vee \exists w.x \mapsto (w) * \mathrm{lsa}(w,y,z)$
acyclic list	$\mathrm{lsn}(x,y)$	$x \neq y \wedge x \mapsto (y)$
		$\vee \exists z.x \neq y \wedge x \mapsto (z) * \mathrm{lsn}(z,y)$
doubly-linked list	$\mathrm{dll}(h,t,p,n)$	$h = t \wedge h \mapsto (p,n)$
		$\vee \exists z.h \mapsto (p,z) * \mathrm{dll}(z,t,h,n)$
reverse doubly-linked list	$\mathrm{dllrev}(h,t,p,n)$	$h = t \wedge h \mapsto (n,p)$
		$\vee \exists z.h \mapsto (z,p) * \mathrm{dllrev}(z,t,h,n)$

Table 2. Definitions of inductive predicates

every element in the history is a normal form and they are not equal by variable renaming, it contradicts.

1. Assume that the procedure terminates without failure for a given input J. Since each step of the procedure consists of an application of an inference rule, J is provable. By the soundness theorem, J is valid. Assume that the procedure fails for a given input J. By Lemma 5.3, J is invalid. Hence the procedure decides the validity of an entailment.

2. Assume that J is valid in order to show J is provable. When we input J to the procedure, by 1, the procedure terminates without failure. Since each step of the procedure consists of applications of inference rules, J is provable. □

6 Experiments

In order to confirm the usefulness of our procedure with the spatial factorization, we have built a prototype prover, which was implemented in OCaml. The experiments were conducted on Mac OS 10.12 machine with Intel® Core™ i5 (1.3GHz) processor and 16GB RAM. The entailments which were checked in our system are listed in Table 1, where the inductive definitions of the predicates are in Table 2. The depth for each valid entailment is the repeat count of the main loop, namely the depth of the proof. The first three examples may seem artificial, but they are just to check how our proof search with the spatial factorization works. The current implementation is for (1-Factor), not for the full (Factor) rule. Hence, the answer FAIL correctly means INVALID for linear cone inductive predicates, whereas it means UNKNOWN for nonlinear cone inductive predicates. All of the predicates in Fig. 2 are linear. The SAT checker for the extended language is based on the idea in [28].

The examples #19 and #20 are (special cases of) the motivating example on the doubly-linked list and its reverse variant. As we have seen in Section 4.5, it can be proved by the spatial factorization in our system. However, the general form $\mathrm{dll}(h,t,p,n) \vdash \mathrm{dllrev}(t,h,n,p)$ cannot be proved in 3 minutes. We guess that one of the reasons is the cost of case analysis, since this entailment contains 4 variables, and they make many cases on $=$ / $\neq$ and $\uparrow$ / $\downarrow$. In fact, if we add a little more assumptions ($p = n = \mathrm{nil}$ for #19, and $p \uparrow$ for #20), we can prove it much faster. The examples #17 and #18 append two doubly-linked lists (we put wrong arguments in #18 on purpose), and the reason why it requires time also seems the cost of case analysis.

Another example where the spatial factorization plays an important role is #12. The predicate $\mathrm{lsa}(x,y,z)$ represents a list segment from x to y such that the cell z lies on it. It allows cyclic lists as $\mathrm{ls}(x,x,y)$, and hence $\mathrm{lsa}(x,x,y) \vdash \mathrm{lsa}(y,y,x)$ is valid. By the spatial factorization, we can find the proof of this entailment. This example also takes much time although it contains only 2 variables. In the proof search, the inductive wand $\mathrm{lsa} {-\!\!*}^{\mathrm{i}} \mathrm{lsa}$ is introduced in succedents. The arity of this inductive wand is 6, and that makes many possibilities of disjuncts in succedents. We guess that makes it hard to find cycles since the depth of the found proof is 9, which is relatively large.

7 Conclusion

We have proposed the cyclic-proof system for entailments of symbolic heaps with cone inductive definitions, and implemented the procedure to decide the validity of entailments by using proof search of this logical system in order to show the usefulness of the logical system by experiments. We have also shown that if the logical system is restricted to linear cone inductive definitions, the procedure becomes a decision procedure for the logical system, and the logical system becomes complete.

For entailment checking in symbolic heaps with cone inductive definition, we have introduced the new inference rule, the factor rule, by means of the inductive wand, which is a variant of the magic wand. The Factor rule enables us to find a common root on both sides of entailments, and by which the Unfold-Match-Remove strategy can be adopted to more widely.

The prover introduced in this paper is a first-step prototype to make sure that the spatial factorization works well for entailment prover, and some examples are correctly checked by it. It is one of the pressing issues to avoid the large case analysis, since our system currently cannot answer $\mathrm{dll} \vdash \mathrm{dllrev}$ without any other assumption.

Another future work is a theoretical aspect of cyclic-proof systems for more relaxed class of the inductive definitions. In particular, a complete cyclic-proof system with decidable and efficient proof search is expected.

Acknowledgements The authors thank anonymous referees for helpful comments. This work is partially supported by JSPS KAKENHI (18K11161) and JSPS Core-to-Core Program (A. Advanced Re- search Networks).

References

[1] R. Brochenin, S. Demri, and E. Lozes, On the Almighty Wand, In: *Proceedings of CSL 2008* (2008) 323–338.

[2] B.-Y. .E. Chang and X. Rival, Relational Inductive Shape Analysis, In: *Proceedings of POPL 2008* (2008) 247–260.

[3] 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.

[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. Berdine, C. Calcagno, B. Cook, D. Distefano, P. W. O'Hearn, T. Wies, H. Yang, Shape Analysis for Composite Data Structures, In: Proceedings of CAV 2007, *LNCS* 4590 (2007) 178–192

[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, 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.

[11] 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.

[12] D. Chu, J. Jaffar, and M. Trinh, Automatic Induction Proofs of Data-Structures in Imperative Programs, In: *Proceedings of PLDI 2015* (2015) 457–466.

[13] 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.

[14] C. Enea, V. Saveluc, M. Sighireanu, Compositional invariant checking for overlaid and nested linked lists, In: Proceeding of ESOP 2013, *LNCS* 7792 (2013) 129–148.

[15] 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.

[16] 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.

[17] 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.

[18] R. Iosif and C. Serban, Complete Cyclic Proof Systems for Inductive Entailments, Available at `https://arxiv.org/abs/1707.02415`, (2017).

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

[20] D. Kimura, M. Tatsuta, and K .Nakazawa, Entailment Checker for Symbolic Heap using Complete Proof System, submitted.

[21] J. Navarro Pérez and A. Rybalchenko, Separation logic modulo theories, In: Proceedings of APLAS 2013, *LNCS 8301* (2013) 90-106.

[22] R. Piskac, T. Wies, and D. Zufferey, Automating Separation Logic Using SMT, In: Proceedings of CAV 2013, *LNCS 8044* (2013) 773-789.

[23] R. Piskac, T. Wies, and D. Zufferey, Automating separation logic with trees and data, In: Proceedings of CAV 2014, *LNCS 8559* (2014) 711–728.

[24] 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.

[25] Q. Ta, T. Le, S. Khoo, and W. Chin, Automated Mutual Induction in Separation Logic, In: Proceedings of FM 2016, *LNCS* 9995 (2016) 659–676.

[26] Q. Ta, T. Le, S.. Khoo, and W. Chin, Automated Lemma Synthesis in Symbolic-Heap Separation Logic, In: Proceedings of POPL 2018, (2018)

[27] M. Tatsuta and D. Kimura, Separation Logic with Monadic Inductive Definitions and Implicit Existentials, In: Proceedings of APLAS 2015, *LNCS* 9458 (2015) 69–89.

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