Cyclic Theorem Prover for Separation Logic by Magic Wand

Koji Nakazawa[1], Makoto Tatsuta[2], Daisuke Kimura[3], and 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 the strong wand and the Factor rule in a cyclic-proof system for the separation-logic entailment checking problem with general inductive predicates. The strong wand is similar to but stronger than the magic wand and is defined syntactically. The Factor rule, which uses the strong wand, is an inference rule for spatial factorization to expose an implicitly allocated cell in inductive predicates. By this rule, we can avoid getting stuck in the Unfold-Match-Remove strategy. We show a semi-decision algorithm of proof search in the cyclic-proof system with the Factor rule and the experimental results of its prototype implementation.

1 Introduction

Separation logic [10, 2, 3] is a logical system for the verification of programs operating heap memories, and it is used as an assertion language of the Hoare logic. One of the keys of automated verification systems based on the separation logic is to decide the validity for entailments of symbolic heaps.

Inductive definitions are important for symbolic heaps, since recursive structures such as lists and trees are described by inductive definitions. The inductive definitions without any restriction cause undecidability [1]. The paper [8] 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 by [8] the *bounded treewidth condition* for restricting the class of inductive definitions. The bounded treewidth condition is one of the most flexible conditions for a decidable system.

We propose *cone inductive definitions*. This condition is stricter than the bounded treewidth condition, namely, this class is included in the class of definitions having the bounded tree-width condition. However we can expect a more efficient proof-search procedure for cone inductive definitions. Moreover our class of inductive definitions is still quite large, since our class contains doubly-linked lists, skip lists, nested lists, and so on. The cone inductive definition requires that every definition clause of the inductive predicate $P(x, x_1, x_2)$ contains $x \mapsto (u)$. We call the first argument x a *root* of $P(x, x_1, x_2)$, since it is the root of a tree-like structure described by this predicate.

Cyclic proofs [4, 5] give efficient semi-decision procedure for the entailment checking problem. In a cyclic-proof system, the induction is represented as a cyclic structure, where some open assumptions are allowed as buds, which play a role of the induction hypothesis. This mechanism gives us more efficient proof search, since we need not fix an induction formula in advance.

Combining Unfold-and-Match proof strategy [7] to cone inductive definitions and cyclic proofs, we obtain the following strategy. First, unfold the atomic formulas 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 $x \mapsto (u)$ 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. However we will have to add more ideas to make this procedure a real algorithm.

In this paper we will propose a procedure as precise as possible. We believe the procedure that we will propose is complete under the condition that every definition clause is linear (namely it has at most one atomic spatial formula except the atomic formula for the root). For this purpose, we extend the succedent to disjunctions. By this, we can unfold $\vdash P(x)$ to $\vdash A, B$ where A and B are the definition clauses of $P(x)$ keeping validity. We also introduce predicates $t\downarrow$ and $t\uparrow$ which mean that t is in the heap and t is not in the heap respectively. By this, we can analyze cases by $\downarrow$ or not.

Then the U-M-R procedure will have a problem, since it may get stuck when any common root does not exist in antecedent and the disjuncts of succedent. To solve this problem, in this paper, we give a new idea, the *strong wand* and the *Factor rule*.

Let us consider the following inductive predicate as a motivating example.

$$\begin{aligned}\mathrm{lsa}(x,y,z) =_{\mathrm{def}}\ & x = z \wedge x \mapsto (y)\\ & \vee\ \exists w.x = z \wedge x \mapsto (w) * \mathrm{lsa}(w,y,w)\\ & \vee\ \exists w.x \mapsto (w) * \mathrm{lsa}(w,y,z).\end{aligned}$$

$\mathrm{lsa}(x,y,z)$ means a (possibly cyclic) list from x to y such that z is allocated in it. Hence, the entailment

$$\mathrm{lsa}(x,x,y) \vdash \mathrm{lsa}(y,y,x)$$

is valid, since both sides represent a circular list that contains x and y. However, in a naive version of U-M-R, we cannot find common roots on both sides which we start to unfold. In general it is hard to find such a circular list during proof search. In fact, the existing system based on the cyclic-proof system, Cyclist [5], fails to find the proof of this entailment.

In order to solve this problem, we propose a strong variant of the magic wand, called the *strong wand*. As the magic wands, the strong wands represent the difference of two heaps. The strong wand $Q —\!\!*^s P$ represents a subheap of P which is obtained by removing a heap of Q. It is done syntactically in

an unfolding tree of P. The point is that the strong wand is not a new logical connective, but $Q \text{—}\!*^s P$ is introduced as a new inductive predicate. For example, the definition clauses of $\mathrm{lsa}(x', y', z') \text{—}\!*^s \mathrm{lsa}(x, y, z)$ are given as follows.

$$\begin{aligned}\mathrm{lsa}(x', y', z') \text{—}\!*^s \mathrm{lsa}(x, y, z) =_{\mathrm{def}}\ & y' = y \wedge z' = x' \wedge x = z \wedge x \mapsto (x') \\ \vee\ & y' = y \wedge z' = z \wedge x \mapsto (x') \\ \vee\ & \exists w. x = z \wedge x \mapsto (w) * (\mathrm{lsa}(x', y', z') \text{—}\!*^s \mathrm{lsa}(w, y, w)) \\ \vee\ & \exists w. x \mapsto (w) * (\mathrm{lsa}(x', y', z') \text{—}\!*^s \mathrm{lsa}(w, y, z)).\end{aligned}$$

By using the strong wand, we introduce an inference rule, called the *Factor rule*, by which we can factorize the spatial predicate and expose a predicate representing the corresponding subheap.

The Factor rule enables us to find the same root on both sides as follows. Suppose that a heap model (s, h) satisfies $P(x)$ and contains a cell named with y. Furthermore, suppose that the definition of the predicate P recursively depends on P itself and another predicate Q. Then, by the shape of definition clauses, some subheap of h satisfies either $P(y)$ or $Q(y)$. Hence $P(x)$ can be factored into either $P(y)$ (or $Q(y)$) and the rest R_1 (or R_2) of it as $P(x) \Leftrightarrow R_1 * P(y) \vee R_2 * Q(y)$. The strong wands enable us to represent these R_1 and R_2 as $P(y) \text{—}\!*^s P(x)$ and $Q(y) \text{—}\!*^s P(x)$ respectively, so we can consider the following instance of the Factor rule

$$\frac{A \vdash (P(y) \text{—}\!*^s P(x)) * P(y), (Q(y) \text{—}\!*^s P(x)) * Q(y)}{A \vdash P(x)}\ (\text{Factor}),$$

by which we can expose the cell named with y in the succedent of the premise. For the above example of lsa, the rule instance of Factor is

$$\frac{\mathrm{lsa}(x, x, y) \vdash \exists y' z'. (\mathrm{lsa}(x, y', z') \text{—}\!*^s \mathrm{lsa}(y, y, x)) * \mathrm{lsa}(x, y', z')}{\mathrm{lsa}(x, x, y) \vdash \mathrm{lsa}(y, y, x)}\ (\text{Factor}),$$

and we can find the corresponding cell named with x in $\mathrm{lsa}(x, y', z')$.

In this paper, we propose the strong wand and the Factor rule, and show the soundness of the Factor rule. We also propose a semi-decision procedure for cyclic-proof search, and give a prototype prover based on it to confirm the usefulness of the spatial factorization.

We give the language for our system in Section 2. The strong wand and the Factor rule are introduced in Section 3, the proof-search procedure with the Factor rule is introduced in Section 4, and some experimental results are given in Section 5. We conclude in Section 6.

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 $x_1, \ldots, x_k$ 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 $x, y, z, \ldots$ and inductive predicate symbols $P, Q, R, \ldots$. We use $\mathcal{P}$ to denote either an inductive predicate or the point-to predicate $\mapsto$. If $\mathcal{P}$ is $\mapsto$, we write $\mathcal{P}(t, \overrightarrow{t})$ for $t \mapsto \overrightarrow{t}$.

Definition 1 (Base language). *The base language is given as follows.*

The terms are defined as $t, u, \cdots ::= \mathrm{nil} \mid x$*, where* nil *is a first-order constant.*

The pure formulas, the spatial formulas, the symbolic heaps, and the entailments 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 fix the length of $\overrightarrow{t}$ *in* $t \mapsto \overrightarrow{t}$ *to* N_c*. The term* t *in* $\mathcal{P}(t, \overrightarrow{t})$ *is called a* root*. We will sometimes write* $\mathcal{P}(t)$ *or* $\mathcal{P}$ *for* $\mathcal{P}(t, \overrightarrow{t})$*. For sets* S *and* S' *of terms, we define*

$$(\neq (S, S')) = \{t \neq t' \mid t \in S,\, t' \in S',\, t \not\equiv t'\}, \qquad (\neq S) = (\neq (S, S)).$$

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 the 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 must be of the form

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

and satisfy the following conditions.

$$\begin{aligned} &\overrightarrow{z} = \{z_j \mid j \in J\} && \text{(strong connectivity)},\\ &\overrightarrow{z} \subseteq \overrightarrow{u} && \text{(decisiveness)}. \end{aligned}$$

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*. The strong connectivity implies the bounded treewidth condition. The decisiveness guarantees that the cell at address x decides the content of every existential variable. It is similar to the constructively valued condition in [6].

An example that is covered by the bounded treewidth condition but is not covered by cone inductive definitions is the tree with parent pointers and linked leaves predicate tll given in [8].

2.2 Extended Language

For case analysis in the proof search, 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 (Extended language). *We extend the spatial formulas of the base language as follows.*

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

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 define the following.

Definition 3. *We define $\Downarrow(\Gamma, X)$ and $\Downarrow(\psi, X)$ as follows.*

$$\begin{aligned} \Downarrow(*_{i\in I}\Delta_i, X) &= \{*_{i\in I}(\Delta_i \wedge X_i \downarrow) \mid \ X = \textstyle\sum_{i\in I} X_i\}, \\ \Downarrow(Y \uparrow \wedge \Pi \wedge \Gamma, X) &= \{Y \uparrow \wedge \Pi \wedge \Gamma' \mid \Gamma' \in \Downarrow(\Gamma, X)\}, \end{aligned}$$

where $\sum_{i\in I} X_i$ means the disjoint union of $\{X_i \mid i \in I\}$.

Definition 4 (Roots). *We define* Roots *as follows.*

$$\begin{aligned} \mathrm{Roots}(X \downarrow \wedge \mathcal{P}(t, \overrightarrow{u})) &= \{t\}, & \mathrm{Roots}(*_i \Delta_i) &= \cup_i \mathrm{Roots}(\Delta_i), \\ \mathrm{Roots}(Y \uparrow \wedge \Pi \wedge \Gamma) &= \mathrm{Roots}(\Gamma), \\ \mathrm{Roots}(\exists x.\phi) &= \mathrm{Roots}(\phi) - \{x\}. \end{aligned}$$

2.3 Semantics

We define the following structure: $\mathrm{Val} = N$, $\mathrm{Locs} = \{x \in N | x > 0\}$, $\mathrm{Heaps} = \mathrm{Locs} \rightarrow_{fin} \mathrm{Val}^{N_c}$, $\mathrm{Stores} = \mathrm{Vars} \rightarrow \mathrm{Val}$. 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 extend 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.

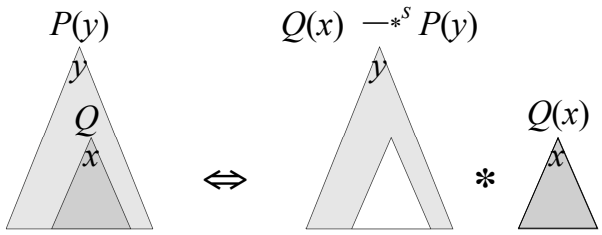


Fig. 1. Strong wand

For a formula F we define the interpretation $s, h \models F$ as follows.
$s, h \models t_1 = t_2$ if $s(t_1) = s(t_2)$,
$s, h \models t_1 \neq t_2$ if $s(t_1) \neq s(t_2)$,
$s, h \models F_1 \wedge F_2$ if $s, h \models F_1$ and $s, h \models F_2$,
$s, h \models \mathrm{emp}$ 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 $b \in N$,
$s, h \models x \uparrow$ if $s(x) \notin \mathrm{Dom}(h)$,
$s, h \models x \downarrow$ if $s(x) \in \mathrm{Dom}(h)$,
$\psi \models \overrightarrow{\phi}$ if $\forall sh(s, h \models \psi$ implies $s, h \models \phi_i$ for some $\phi_i \in \overrightarrow{\phi})$.
The entailment $\psi \vdash \overrightarrow{\phi}$ is said to be valid if $\psi \models \overrightarrow{\phi}$ holds.

3 Strong wand and spatial factorization

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

Under the conditions on the inductive definition, a heap model satisfying a predicate and an unfolding tree of the predicate are naturally corresponding, since any cell in a heap of 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 an inductive predicate R, whose inductive definition is obtained removing the occurrence of Q from the definition clauses of P. We call this R the strong wand $Q —*^s P$. Then, $s, h \models P \wedge x \downarrow$ iff $s, h \models (Q(x) —*^s P) * Q(x)$ 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 Strong wand

We define the strong wand Q—$*^s P$ for inductive predicates P and Q where the definition of P recursively depends on Q. The strong wand is not introduced as a logical connective, but as an additional inductive predicate. $Q(y)$—$*^s P(x)$ plays a similar role to the ordinary magic wand $Q(y)$—$*P(x)$, but it is stronger than the ordinary one and it is defined syntactically.

We write Pr for the set of the original predicate symbols, and Pr is extended with strong wands Q—$*^s P$, where $Q \in \mathrm{Pr}$ and P may be another strong wand. In the following, $P, Q, \cdots$ denote predicate symbols extended by the strong wand.

When the arities of P and Q are n_P and n_Q, respectively, the arity of Q—$*^s P$ is $n_P + n_Q$. We write $Q(v, \overrightarrow{w})$—$*^s P(x, \overrightarrow{y})$ for $(Q$—$*^s P)(x, \overrightarrow{y}, v, \overrightarrow{w})$, so the root of $Q(v, \overrightarrow{w})$—$*^s P(x, \overrightarrow{y})$ is x.

Definition 5. *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 strong wand by* $\mathrm{Dep}(Q$—$*^s P) = \mathrm{Dep}(P)$*.*

Definition 6. *For* $Q \in \mathrm{Dep}(P)$*, the definition clauses of* $Q(v, \overrightarrow{w})$—$*^s P(x, \overrightarrow{y})$ *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 $Q(v, \overrightarrow{w})$—$*^s P(x, \overrightarrow{y})$*.*

(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* $l \in L$ *such that* $Q \in \mathrm{Dep}(P_l)$*, we add*

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

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

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

Example 1. For the list segment

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

the strong wand ls—$*^s$ls is defined by the following clauses.

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

Example 2. For the doubly-linked list

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

the strong wand dll—$*^s$dll is defined by the following clauses.

$$\mathrm{dll}(h', p', n', t')—*^s\mathrm{dll}(h, p, n, t) =_{\mathrm{def}} p' = h \wedge n' = n \wedge t' = t \wedge h \mapsto (p, h') \\ \vee \exists z. h \mapsto (p, z) * (\mathrm{dll}(h', p', n', t')—*^s\mathrm{dll}(z, h, n, t)).$$

The strong wand is a stronger variant of the magic wand, that is, $P—*^sQ \models P—*Q$ always holds. However, the converse is not necessarily true. For example, the empty heap satisfies $\mathrm{ls}(x, y)—*\mathrm{ls}(x, y)$, but does not satisfy $\mathrm{ls}(x, y)—*^s\mathrm{ls}(x, y)$, since any strong wand is defined by the cone inductive definition, and hence a heap satisfying $\mathrm{ls}(x, y)—*^s\mathrm{ls}(x, y)$ contains at least one cell.

Note that we have no definition clause for $P—*^sQ$ when $P \notin \mathrm{Dep}(Q)$, and then $P—*^sQ$ means false.

We write $Q_1, \ldots, Q_m—*^sP$ for $Q_1—*^s \ldots —*^sQ_m—*^sP$. We also use the notation $\overrightarrow{Q}—*^sP$. If $\overrightarrow{Q}$ is empty, $\overrightarrow{Q}—*^sP$ denotes P. Intuitively, $Q_1, \ldots, Q_m—*^sP$ represents a heap obtained by removing subheaps satisfying $Q_1, \ldots, Q_m$ from a heap satisfying P.

By the following lemma, the order of predicates in $\overrightarrow{Q}$ does not change the definition of $\overrightarrow{Q}—*^sP$. For simplicity of notation, the arity of every predicate is suppose to be two.

Lemma 1. *The definition clauses of* $\overrightarrow{R(v,u)}—*^sP(x,z)$ *are*

$$\exists (w_{1i})_{i\in I}.(\Pi \wedge \bigwedge \overrightarrow{u_2 = s_2} \wedge x \mapsto (\overrightarrow{t}) **_{i\in I} (\overrightarrow{R_{1i}(v_{1i}, u_{1i})}—*^sQ_{1i}(w_{1i}, s_{1i})))[\overrightarrow{w}_2 := \overrightarrow{v}_2]$$

for each clause $\exists \overrightarrow{w}.\Pi \wedge x \mapsto (\overrightarrow{t}) * \overrightarrow{Q(w,s)} \Rightarrow P(x,z)$*, and divisions (with permutations)*

$$\overrightarrow{R(v,u)} = (\overrightarrow{R_{1i}(v_{1i}, u_{1i})})_{i\in I}, \overrightarrow{R_2(v_2, u_2)}, \\ \overrightarrow{Q(w,s)} = (Q_{1i}(w_{1i}, s_{1i}))_{i\in I}, \overrightarrow{Q_2(w_2, s_2)}$$

such that $\overrightarrow{R}_2 = \overrightarrow{Q}_2$*, where each* $\overrightarrow{R}_{1i}$ *can be empty.*

Hence, if $\overrightarrow{R}'$ *is a permutation of* $\overrightarrow{R}$*, then* $\overrightarrow{R}'—*^sP$ *is equivalent to* $\overrightarrow{R}—*^sP$*.*

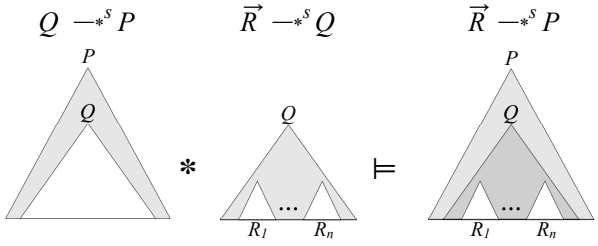
Proof. By induction on the length of $\overrightarrow{R}$. The detailed proof is in [13]. □

The strong wand is stronger than the magic wand, that is, $P—*^sQ \models P—*Q$ holds.

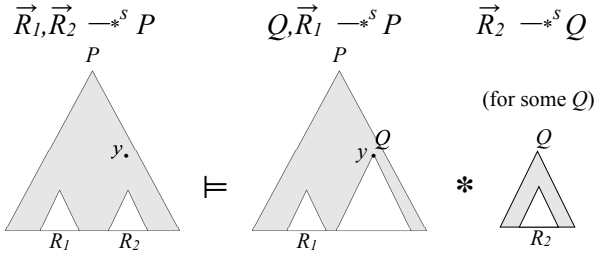
The properties of the strong wand are given as the following lemmas, which are depicted as Figure 2. These are generalization of the following.

$$(Q(y,w)—*^sP(x,z)) * Q(y,w) \models P(x,z), \\ x \neq y \wedge y \downarrow \wedge P(x,z) \models \{\exists w.(Q(y,w)—*^sP(x,z)) * Q(y,w) \mid Q \in \mathrm{Dep}(P)\}.$$

The first property, called the *strong-wand elimination*, means that, by combining two heaps satisfying $Q(y)—*^sP(x)$ and $Q(y)$ we obtain the heap satisfying $P(x)$



Strong-wand elimination (Lemma 2)



Strong-wand introduction (Lemma 3)

Fig. 2. Strong-wand elimination and introduction

in the same way as the ordinary magic wand. The second property, called the *strong-wand introduction*, means that, if a heap satisfies $P(x)$ and contains the cell named with y, the heap can be factorized into two parts $Q(y)—\!\!*^s P(x)$ and $Q(y)$. Since we want as a precise proof-search procedure as possible, we consider the disjunction of all possibilities of $Q \in \mathrm{Dep}(P)$. Note that, for $Q \notin \mathrm{Dep}(P)$, the strong wand $Q—\!\!*^s P$ always means false, so we do not have to consider this case.

In the following, we do not consider the order in sequences for divisions such as $\overrightarrow{R} = \overrightarrow{R}_1, \overrightarrow{R}_2$.

Lemma 2 (Strong-wand elimination). *We have*

$$(Q(y,w)—\!\!*^s P(x,z)) * (\overrightarrow{R(v,u)}—\!\!*^s Q(y,w)) \models \overrightarrow{R(v,u)}—\!\!*^s P(x,z).$$

Proof. We prove $s,h \models LHS$ implies $s,h \models RHS$ by induction on the size of h. The detailed proof is in [13]. □

Lemma 3 (Strong-wand introduction). *We have*

$$\begin{aligned} x \neq y \wedge y \downarrow \wedge (\overrightarrow{R(v,u)}—\!\!*^s P(x,z)) \models \\ \{\exists w.(Q(y,w), \overrightarrow{R_1(v_1,u_1)}—\!\!*^s P(x,z)) * (\overrightarrow{R_2(v_2,u_2)}—\!\!*^s Q(y,w)) \mid \\ \overrightarrow{R} = \overrightarrow{R}_1, \overrightarrow{R}_2,\ Q \in \mathrm{Dep}(P),\ \overrightarrow{R}_2 \subseteq \mathrm{Dep}(Q)\}. \end{aligned}$$

Proof. We prove $s, h \models LHS$ implies $s, h \models RHS$ by induction on the size of h. The detailed proof is in [13]. □

Let P be $\overrightarrow{R'}\text{—}*^s P$ in Lemma 2, we have

$$(Q, \overrightarrow{R'}\text{—}*^s P) * (\overrightarrow{R}\text{—}*^s Q) \models \overrightarrow{R}, \overrightarrow{R'}\text{—}*^s P(x, z).$$

3.2 Spatial factorization

The factor rule is the following.

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

where the auxiliary functions are defined as

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

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

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

Proposition 1 (Soundness of Factor). *If* $\psi \models \overrightarrow{\phi}, \mathrm{Fact}(\phi, y)$*, then* $\psi \models \overrightarrow{\phi}, \phi$*.*

Proof. Suppose $s, h \models \psi$, and, by the assumption, we have $s, h \models \overrightarrow{\phi}, \mathrm{Fact}(\phi, y)$. In the case of $s, h \models \overrightarrow{\phi}$, it is trivial that $s, h \models \overrightarrow{\phi}, \phi$. Otherwise, we have $s, h \models \mathrm{Fact}(\phi, y)$. Then, $\phi \equiv \exists \overrightarrow{v}.\Pi \wedge \Sigma * (\overrightarrow{R}\text{—}*^s P)$ and $s, h \models \exists \overrightarrow{v}\,\overrightarrow{w}.\Pi \wedge \Sigma * \Sigma'$ for some $\exists \overrightarrow{w}.\Sigma' \equiv \exists \overrightarrow{w}.(\overrightarrow{R}_1, Q(y, \overrightarrow{w})\text{—}*^s P) * (\overrightarrow{R}_2\text{—}*^s Q(y, \overrightarrow{w})) \in \mathrm{Fact}_{\mathrm{aux}}(\overrightarrow{R}\text{—}*^s P, y)$, where $\overrightarrow{w} \notin FV(\Pi \wedge \Sigma * (\overrightarrow{R}\text{—}*^s P))$. By the definition, there exist $\overrightarrow{a}$ and $\overrightarrow{b} \in N$ such that $s[\overrightarrow{w} := \overrightarrow{a}, \overrightarrow{v} := \overrightarrow{b}], h \models \Pi \wedge \Sigma * (\overrightarrow{R}_1, Q(y, \overrightarrow{w})\text{—}*^s P) * (\overrightarrow{R}_2\text{—}*^s Q(y, \overrightarrow{w}))$. Let s' be $s[\overrightarrow{w} := \overrightarrow{a}, \overrightarrow{v} := \overrightarrow{b}]$. We have $h_1 + h_2 = h$ such that $s', h_1 \models \Sigma$ and $s', h_2 \models (\overrightarrow{R}_1, Q(y, \overrightarrow{w})\text{—}*^s P) * (\overrightarrow{R}_2\text{—}*^s Q(y, \overrightarrow{w}))$. By Lemma 2, we have $s', h_2 \models \overrightarrow{R}_1, \overrightarrow{R}_2\text{—}*^s P$, where $\overrightarrow{R}_1, \overrightarrow{R}_2 \equiv \overrightarrow{R}$. Hence, we have $s', h \models \Pi \wedge \Sigma * (\overrightarrow{R}\text{—}*^s P)$. Therefore, $s, h \models \phi$ holds. □

4 Inference rules and proof search

In this section, we introduce the proof-search procedure with the Factor rule.

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

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

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

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

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

$$\frac{\Pi' \wedge \psi * x \mapsto (\overrightarrow{t}) * \Gamma' \vdash \overrightarrow{\phi} \quad (\forall \Gamma' \in \cup\{\Downarrow(\Sigma', X) \mid \exists \overrightarrow{z}.\Pi' \wedge x \mapsto (\overrightarrow{t}) * \Sigma' \Rightarrow P(x),\ \overrightarrow{z} \text{ is fresh}\})}{\psi * (P(x) \wedge X\downarrow) \vdash \overrightarrow{\phi}}\ (\mathrm{Pred}\ L)$$

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

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

$$\frac{x\uparrow \wedge \psi \vdash (\exists z_i.\Pi_i \wedge \Sigma_i)_i}{\psi * x \mapsto (\overrightarrow{u}) \vdash (\exists z_i.\Pi_i \wedge \Sigma_i * x \mapsto (\overrightarrow{u}))_i}\ (* \mapsto)$$

The definition of $\Downarrow$ in (Case$\uparrow\downarrow$) and (Pred$L$) is given in Definition 3. The definition of Fact in (Factor) is given in Section 3.2.

Fig. 3. Inference Rules

4.1 Inference rules

The inference rules are summarized in Figure 3. Note that the SAT check for the extended language is decidable [13].

Proposition 2. *Every inference rule in Figure 3 is sound.*

Proof. The soundness of (emp) is proved as follows. Let J be $X\uparrow \wedge \Pi \wedge \mathrm{emp} \vdash \exists \overrightarrow{x}.\Pi' \wedge \mathrm{emp}$. Suppose (s, h) satisfies the antecedent. Since N is an infinite set, there exist $\overrightarrow{a} \notin s(FV(J))$ such that the elements in $\overrightarrow{a}$ are mutually distinct. Then, we have $s[\overrightarrow{x} := \overrightarrow{a}], h \models \Pi'$, since Π' contains only $x \neq t$ such that $x \in \overrightarrow{x}$.

The soundness of (Case$\uparrow\downarrow$) follows from that $x\uparrow \vee x\downarrow$ always holds. The soundness of (Factor) is proved in Proposition 1. The other rules are proved in a straightforward way. □

A proof in our proof system is a proof structure constructed with the inference rules with cycles formed by a bud-companion relation satisfying the global trace

condition [4], such that the proof system is sound. The global trace condition imposes that for every infinite path in a proof structure (called pre-proof in [4]), there is a trace of predicate occurrences in the antecedents following the path such that infinitely many predicates in the trace are unfolded by the rule (Pred$L$). Note that every proof structure obtained by the following proof-search procedure satisfies the global trace condition, since any bud has a companion as an ancestor and the path from the companion to the bud contains (Pred$L$).

Theorem 1. *The proof system is sound, that is, if* $\psi \vdash \overrightarrow{\phi}$ *holds, then we have* $\psi \models \overrightarrow{\phi}$.

Proof. It is proved by the soundness of the inference rules and the condition on the cyclic structure. □

4.2 Initialization for proof search

In this and the next sections, we describe our proof-search procedure, which is an Unfold-Match-Remove procedure. Note that we have no split rule such as

$$\frac{\Pi \wedge \Gamma_1 \vdash \Pi' \wedge \Sigma_1 \quad \Pi \wedge \Gamma_2 \vdash \Pi' \wedge \Sigma_2}{\Pi \wedge \Gamma_1 * \Gamma_2 \vdash \Pi' \wedge \Sigma_1 * \Sigma_2} \ (*)$$

similarly to the proof systems in [7, 11, 12]. If we impose the condition that every definition clause has at most one atomic spatial formula except $x \mapsto (\overrightarrow{t})$ for the root x, we believe that our procedure will be complete. On the other hand, if we do not have such an additional condition, we have to consider a split rule for completeness and decidability. Such a split rule for entailments with multiple conclusions which is locally complete is challenging, and it is proposed in [13].

For the input entailment, apply the following steps 1–6, and then repeat the main loop given in the next section. If all subgoals finish in the termination check (step 5) or the cycle check (step 12), the procedure stops and answers VALID. If one of subgoals stops with FAIL in the termination check (step 5) or the succedent of one of the subgoals becomes empty by the rule ($\vee R$) at the step 4.4 or 6.1, the procedure stops and answers FAIL.

1. **[Case analysis** $\uparrow$ / $\downarrow$**]** Repeat (Case$\uparrow\downarrow$) to make cases with respect to $x\uparrow$ and $x\downarrow$ for each free variable x of the entailment. Then, we obtain a set of subgoal entailments.

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

3. **[Unsat check]** For each subgoal, if the antecedent is unsatisfiable, finish this case by applying (Unsat).

4. **[Pure check]** For each subgoal J, do the following 4.1–4.4.

4.1. Repeat ($= L$) until the antecedent contains no $=$, and then the pure part in the antecedent becomes the set of $t \neq t'$ for every pair of $FV(J) \cup \{\mathrm{nil}\}$.

4.2. For every $\exists x.x = t \wedge \Pi \wedge \Sigma$ in the succedent, replace it to $(\Pi \wedge \Sigma)[x := t]$ by ($\exists R$).

4.3. Repeat ($\wedge R$), (Triv$R$) until these rules cannot be applied.

4.4. Remove all disjuncts 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. [**Termination check**] If the spatial part of the antecedent is emp, do the following 5.1. Otherwise, go to the step 6.

5.1. If there is a disjunct whose spatial part is emp, finish this case by applying ($\wedge L_{\downarrow}$), ($\wedge L_p$), and (emp). Otherwise, stop with FAIL.

6. [**Normalization**] For each subgoal J, do the following 6.1–6.2.

6.1. Remove all disjuncts containing two or more spatial atoms with the same root by repeating ($\vee R$).

6.2. Remove all $x \neq t$ and $x \uparrow$ in the antecedent such that x does not occur in the spatial part of the antecedent or the succedent by repeating ($\wedge L_p$) and ($\wedge L_{\uparrow}$). Remove all $x \downarrow$ in $\Gamma \wedge x \downarrow$ such that $\mathrm{Roots}(\Gamma) = \{x\}$ by ($\wedge L_{\downarrow}$).

4.3 Main loop for proof search

In this section, we define the main loop of our procedure.

We keep the set of entailments that we have already processed as a subgoal. We call it the history and we use it for companions in the cyclic proof.

After the initialization, we obtain a set of subgoal entailments $\overrightarrow{J}$. Repeat to apply the following steps 7–12 to the subgoals $\overrightarrow{J}$.

7. [**Common root**] For each subgoal $\psi \vdash (\phi_i)_i$, check the set $\mathrm{Roots}(\psi) \cap (\cap_i \mathrm{Roots}(\phi_i))$. If this set is not empty, choose x in this set, and go to the step 9. Otherwise, choose $x \in \mathrm{Roots}(\psi)$, and do the step 8.

8. [**Spatial factorization**] For all disjuncts ϕ such that $x \notin \mathrm{Roots}(\phi)$, apply (Factor). Then, x is in Roots of all disjuncts.

9. [**Unfolding**] Apply (Pred$L$) to the predicate with the root x in the antecedent, and then we obtain several subgoals. For each subgoal, apply (Pred$R$) to all of the predicates with the root x in the disjuncts in the succedent to generate new disjuncts.

10. [**Matching**] For each subgoal, do the following 10.1–10.2.

10.1. Repeat ($= L$), ($\mapsto =$), ($\exists =$), and (Triv$R$) until they cannot be applied. Then, all of $x \mapsto (\overrightarrow{u})$ in the disjuncts are identical to the $x \mapsto (\overrightarrow{u}')$ in the antecedent.

10.2. Apply ($\mapsto *$).

11. Do the steps 2–6 of the initialization.

12. [**Cycle check**] For each subgoal J do the following. If J is obtained by substitution from an entailment in the history, finish this case. Otherwise, add J to the history and continue the main loop with the subgoal J.

4.4 Examples of proof search

We give two examples to see how the procedure works.

Example 3. The first example is on the following simple predicates.

$$\begin{aligned} \mathrm{two}(x,y) &=_{\mathrm{def}} \exists z.y = z \wedge x \mapsto (z) * \mathrm{to}(z,x), \\ \mathrm{to}(x,y) &=_{\mathrm{def}} x \mapsto (y), \end{aligned}$$

where the predicate $\mathrm{to}(x,y)$ has the definition clause $x \mapsto (y)$ and introduced to make the inductive definition of two satisfy the conditions for definition clauses. The predicate $\mathrm{two}(x,y)$ just means $x \mapsto (y) * y \mapsto (x)$, and hence the entailment $\mathrm{two}(x,y) \vdash \mathrm{two}(y,x)$ is valid. The proof of this entailment is constructed as follows.

First, we do the initialization (steps 1–6), and then the following subgoal remains

$$(\neq \{x, y, \mathrm{nil}\}) \wedge \mathrm{two}(x,y) \wedge y \downarrow \vdash \mathrm{two}(y,x).$$

Then, we start the main loop with this subgoal. The succedent has no root x, so we apply (Factor) at the step 8, and then we obtain

$$(\neq \{x, y, \mathrm{nil}\}) \wedge \mathrm{two}(x,y) \wedge y \downarrow \vdash \exists y'.(\mathrm{to}(x,y') \mathrel{-\!\!*^s} \mathrm{two}(y,x)) * \mathrm{to}(x,y').$$

By unfolding (step 9) we obtain

$$(\neq \{x, y, \mathrm{nil}\}) \wedge y = z \wedge x \mapsto (z) * (\mathrm{to}(z,x) \wedge y \downarrow) \vdash \exists y'.(\mathrm{to}(x,y') \mathrel{-\!\!*^s} \mathrm{two}(y,x)) * x \mapsto (y'),$$

and by matching (step 10) we obtain

$$x \uparrow \wedge (\neq \{x, y, \mathrm{nil}\}) \wedge (\mathrm{to}(y,x) \wedge y \downarrow) \vdash \mathrm{to}(x,y) \mathrel{-\!\!*^s} \mathrm{two}(y,x).$$

The steps 2–5 do not change it, and $y \downarrow$ is removed at the step 6.3, so the subgoal

$$x \uparrow \wedge (\neq \{x, y, \mathrm{nil}\}) \wedge \mathrm{to}(y,x) \vdash \mathrm{to}(x,y) \mathrel{-\!\!*^s} \mathrm{two}(y,x) \qquad (*)$$

is the input of the next main loop.

Here, by the definition, the strong wand $\mathrm{to} \mathrel{-\!\!*^s} \mathrm{two}$ is defined by the following clause.

$$\mathrm{to}(x',y') \mathrel{-\!\!*^s} \mathrm{two}(x,y) =_{\mathrm{def}} y = x' \wedge x = y' \wedge x \mapsto (x').$$

Note that there is no clause with $\mathrm{to} \mathrel{-\!\!*^s} \mathrm{to}$, since $\mathrm{to} \notin \mathrm{Dep}(\mathrm{to})$. $\mathrm{to}(x',y') \mathrel{-\!\!*^s} \mathrm{two}(x,y)$ implies that $\mathrm{to}(x',y')$ is a proper subheap of $\mathrm{two}(x,y)$ which does not contain x, and the strong wand represents the singleton heap $x \mapsto (y)$.

For $(*)$, we have common root y, so skip the step 8. By unfolding we obtain

$$x \uparrow \wedge (\neq \{x, y, \mathrm{nil}\}) \wedge y \mapsto (x) \vdash x = x \wedge y = y \wedge y \mapsto (x),$$

and by matching we obtain

$$\{x, y\} \uparrow \wedge (\neq \{x, y, \mathrm{nil}\}) \wedge \mathrm{emp} \vdash x = x \wedge y = y \wedge \mathrm{emp}.$$

By the pure check (step 4) the equalities in the succedent are removed, and in the termination check (step 5) the proof successfully finishes.

Example 4. The second example is the motivating example given in the introduction. The inductive definition of lsa is

$$\begin{aligned}\mathrm{lsa}(x,y,z) =_{\mathrm{def}}\ & x = z \wedge x \mapsto (y)\\ \vee\ & \exists w.x = z \wedge x \mapsto (w) * \mathrm{lsa}(w,y,w)\\ \vee\ & \exists w.x \mapsto (w) * \mathrm{lsa}(w,y,z),\end{aligned}$$

and the entailment to be checked is $\mathrm{lsa}(x,x,y) \vdash \mathrm{lsa}(y,y,x)$.

By initialization (the steps 1–6), we have the subgoals:

$$\begin{aligned}\mathrm{lsa}(x,x,x) &\vdash \mathrm{lsa}(x,x,x),\\ (\neq \{x,y,\mathrm{nil}\}) \wedge \mathrm{lsa}(x,x,y) \wedge y\downarrow &\vdash \mathrm{lsa}(y,y,x),\end{aligned}$$

The case of $y\uparrow$ is removed by the unsat check.

The nontrivial subgoal is the second one. In order to choose predicates to be unfolded, we want to find the common root on both sides, but there is not such a root. Hence, we apply (Factor) at the step 8 to expose the root x in the succedent.

$$(\neq \{x,y,\mathrm{nil}\}) \wedge \mathrm{lsa}(x,x,y) \wedge y\downarrow \vdash \exists y'z'.(\mathrm{lsa}(x,y',z') \mathrel{-\!\!*^s} \mathrm{lsa}(y,y,x)) * \mathrm{lsa}(x,y',z').$$

By unfolding lsa in the antecedent (step 9), we obtain the three cases

$$\begin{aligned}(\neq \{x,y,\mathrm{nil}\}) \wedge x = y \wedge x \mapsto x &\vdash \cdots,\\ (\neq \{x,y,\mathrm{nil}\}) \wedge x = y \wedge x \mapsto (w) * (\mathrm{lsa}(w,x,w) \wedge y\downarrow) &\vdash \cdots,\\ (\neq \{x,y,\mathrm{nil}\}) \wedge x \mapsto (w) * (\mathrm{lsa}(w,x,y) \wedge y\downarrow) &\vdash \cdots,\end{aligned}$$

but the first two cases will be removed by the unsat check, so we ignore these cases here. By unfolding lsa in the succedent, we obtain one subgoal with three disjuncts in the succedent.

$$\begin{aligned}&(\neq \{x,y,\mathrm{nil}\}) \wedge x \mapsto (w) * (\mathrm{lsa}(w,x,y) \wedge y\downarrow) \vdash\\ &\quad \exists y'z'.x = z' \wedge (\mathrm{lsa}(x,y',z') \mathrel{-\!\!*^s} \mathrm{lsa}(y,y,x)) * x \mapsto (y'),\\ &\quad \exists y'z'w'.x = z' \wedge (\mathrm{lsa}(x,y',z') \mathrel{-\!\!*^s} \mathrm{lsa}(y,y,x)) * x \mapsto (w') * \mathrm{lsa}(w',y',w'),\\ &\quad \exists y'z'w'.(\mathrm{lsa}(x,y',z') \mathrel{-\!\!*^s} \mathrm{lsa}(y,y,x)) * x \mapsto (w') * \mathrm{lsa}(w',y',z').\end{aligned}$$

By the matching step (step 10), we have

$$\begin{aligned}x\uparrow \wedge (\neq \{x,y,\mathrm{nil}\}) \wedge \mathrm{lsa}(w,x,y) \wedge y\downarrow \vdash\\ \mathrm{lsa}(x,w,x) \mathrel{-\!\!*^s} \mathrm{lsa}(y,y,x),\\ \exists y'.(\mathrm{lsa}(x,y',x) \mathrel{-\!\!*^s} \mathrm{lsa}(y,y,x)) * \mathrm{lsa}(w,y',w),\\ \exists y'z'.(\mathrm{lsa}(x,y',z') \mathrel{-\!\!*^s} \mathrm{lsa}(y,y,x)) * \mathrm{lsa}(w,y',z').\end{aligned}$$

Then, at the step 2, we have already known $(\neq \{x,y,\mathrm{nil}\})$, so the case analysis for w is sufficient. Since $x\uparrow$ and w is a root, so the cases $w = x$ and $w = \mathrm{nil}$ are unsat, and hence we have two cases, $w = y$ and $w \neq y$. Then, the first

disjunct containing $x = w$ is removed in the pure check (step 4), and we have the following two subgoals.

$$\begin{aligned}
&x \uparrow \wedge (\neq \{x, y, \mathrm{nil}\}) \wedge \mathrm{lsa}(y, x, y) \wedge y \downarrow \vdash \\
&\qquad \mathrm{lsa}(x, y, x) \text{—}\!*^{s} \mathrm{lsa}(y, y, x), \\
&\qquad \exists y'.(\mathrm{lsa}(x, y', x) \text{—}\!*^{s} \mathrm{lsa}(y, y, x)) * \mathrm{lsa}(y, y', y), \\
&\qquad \exists y' z'.(\mathrm{lsa}(x, y', z') \text{—}\!*^{s} \mathrm{lsa}(y, y, x)) * \mathrm{lsa}(y, y', z'). \\
&x \uparrow \wedge (\neq \{x, y, w, \mathrm{nil}\}) \wedge \mathrm{lsa}(w, x, y) \wedge y \downarrow \vdash \\
&\qquad \mathrm{lsa}(x, w, x) \text{—}\!*^{s} \mathrm{lsa}(y, y, x), \\
&\qquad \exists y'.(\mathrm{lsa}(x, y', x) \text{—}\!*^{s} \mathrm{lsa}(y, y, x)) * \mathrm{lsa}(w, y', w), \\
&\qquad \exists y' z'.(\mathrm{lsa}(x, y', z') \text{—}\!*^{s} \mathrm{lsa}(y, y, x)) * \mathrm{lsa}(w, y', z').
\end{aligned}$$

At the normalization step (step 6), $y \downarrow$ and disjuncts containing two predicates with the same root y in the first subgoal are removed.

$$\begin{aligned}
&x \uparrow \wedge (\neq \{x, y, \mathrm{nil}\}) \wedge \mathrm{lsa}(y, x, y) \downarrow \vdash \\
&\qquad \mathrm{lsa}(x, y, x) \text{—}\!*^{s} \mathrm{lsa}(y, y, x). \\
&x \uparrow \wedge (\neq \{x, y, w, \mathrm{nil}\}) \wedge \mathrm{lsa}(w, x, y) \wedge y \downarrow \vdash \\
&\qquad \mathrm{lsa}(x, w, x) \text{—}\!*^{s} \mathrm{lsa}(y, y, x), \\
&\qquad \exists y'.(\mathrm{lsa}(x, y', x) \text{—}\!*^{s} \mathrm{lsa}(y, y, x)) * \mathrm{lsa}(w, y', w), \\
&\qquad \exists y' z'.(\mathrm{lsa}(x, y', z') \text{—}\!*^{s} \mathrm{lsa}(y, y, x)) * \mathrm{lsa}(w, y', z').
\end{aligned}$$

These are the output of the normalization phase. We cannot find any cycle, so we repeat the main loop for this set of subgoals. At last, the procedure successfully stop for all of subgoals, and answer VALID.

5 Experiments

In order to confirm the usefulness of our procedure with the spatial factorization, we give a prototype prover, which is implemented in OCaml with about 6k lines of code. The entailments which are proved in our system are listed in Figure 4, where the inductive definitions of the predicates are in Figure 5. These experiments were conducted on Mac OS 10.12 machine with Intel® Core™ i5 (1.3GHz) processor and 16GB RAM.

The current implementation is limited on the following points.

- The depth of the strong wands is limited to one, that is, the set $\mathrm{Fact}_{\mathrm{aux}}(\overrightarrow{R} \text{—}\!*^{s} P)$ is considered only for $\overrightarrow{R} = \emptyset$.
- We have not had complete SAT checker for the extended language, so we use only a sufficient condition for unsatisfiability for (Unsat) rule.

Hence, if the prover fails to prove an entailment, it just means unknown.

For #1, #2, #4, and #7 the prover actually uses the Factor rule. #7 is slightly different from the motivating example, in which the assumption $y \downarrow$ is

#	entailments	time (sec.)
1*	$\mathrm{two}(x,y) \vdash \mathrm{two}(y,x)$	0.01
2*	$\mathrm{tri}(x,y,z) \vdash \mathrm{tri}(y,z,x)$	1.03
3	$\mathrm{ls}(x,y) * \mathrm{ls}(y,z) \vdash \mathrm{ls}(x,z)$	0.10
4*	$\mathrm{ls}(x,x) \wedge y \downarrow\vdash \mathrm{ls}(y,y)$	0.02
5	$\mathrm{lsa}(x,y,z) \vdash \mathrm{ls}(x,y)$	0.61
6	$\mathrm{ls}(x,y) \vdash \mathrm{lsa}(x,y,x)$	0.51
7*	$\mathrm{lsa}(x,x,y) \wedge y \downarrow\vdash \mathrm{lsa}(y,y,x)$	3.90
8	$z \uparrow \wedge \mathrm{lsn}(x,y) * \mathrm{lsn}(y,z) \vdash \mathrm{lsn}(x,z)$	0.02
9	$\mathrm{lsn}(x,y) * \mathrm{lsn}(y,z) * \mathrm{to}(z,\mathrm{nil}) \vdash \mathrm{lsn}(x,z) * \mathrm{to}(z,\mathrm{nil})$	0.33
10	$\mathrm{dll}(h,p,n,t) * \mathrm{dll}(n,t,h,p) \vdash \mathrm{dll}(h,p,h,p)$	27.74

(* proved with the Factor rule)

Fig. 4. Proved entailments

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

Fig. 5. Definitions of inductive predicates

not needed. Since our implementation uses a weaker satisfiability checker, we need $y \downarrow$ for experiments. It takes much time to prove #10. We guess that one of the reasons is the cost of case analysis, since this entailment contains four variables, and they make a lot of cases on $= / \neq$ and $\uparrow / \downarrow$. If we have efficient SAT checker, we can exclude the case of $t \uparrow \wedge \mathrm{dll}(h, p, n, t)$, and expect faster answer.

6 Conclusion

We introduced the strong wand, which is a variant of the so-called magic wand, to define the Factor rule for entailment checking in symbolic heaps with cone inductive definition. The Factor rule is the inference rule for the spatial factorization, which enables us to find a common root on both sides of entailments we should unfold in the Unfold-Match-Remove proof-search procedure.

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, including the motivating example, are correctly proved. However, in order to check more complicated examples, we have to remove the limitation given in Section 5, and refine the prover. It is one of the pressing issues to implement a complete SAT checker for the extended language. Optimization to avoid the large case analysis is also necessary.

Another future work is a theoretical aspect of cyclic-proof system. In particular, complete cyclic-proof system with decidable and efficient proof search is expected. We believe the procedure that we have proposed is complete under the condition that every definition clause is linear. In [9], a complete cyclic-proof system is introduced with some semantic conditions for inductive predicates. The authors currently propose another complete cyclic-proof system CSLIDω [13] with the syntactic conditions for inductive predicates, which are the same as this paper. In that system, the strong wand and the Factor rule play important roles.

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. J. Berdine, C. Calcagno, P. W. O'Hearn, A Decidable Fragment of Separation Logic, In: Proceedings of FSTTCS 2004, *LNCS* 3328 (2004) 97–109.
3. J. Berdine, C. Calcagno, and P. W. O'Hearn, Symbolic Execution with Separation Logic, In: Proceedings of APLAS 2005, *LNCS* 3780 (2005) 52–68.
4. J. Brotherston, D. Distefano, and R. L. Petersen, Automated cyclic entailment proofs in separation logic, In: *Proceedings of CADE-23* (2011) 131–146.
5. J. Brotherston, N. Gorogiannis, and R. L. Petersen, A Generic Cyclic Theorem Prover, In: Proceedings of APLAS 2012, *LNCS* 7705 (2012) 350–367.
6. 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.

7. D. Chu, J. Jaffar, and M. Trinh, Automatic Induction Proofs of Data-Structures in Imperative Programs, In: *Proceedings of PLDI 2015* (2015) 457–466.
8. 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.
9. R. Iosif and C. Serban, Complete Cyclic Proof Systems for Inductive Entailments, Available at `https://arxiv.org/abs/1707.02415`, (2017).
10. 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.
11. 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.
12. Q. Ta, T. Le, S.. Khoo, and W. Chin, Automated Lemma Synthesis in Symbolic-Heap Separation Logic, In: Proceedings of POPL 2018, (2018)
13. M. Tatsuta, K. Nakazawa, and D. Kimura, Completeness of Cyclic Proofs for Symbolic Heaps, Available at `https://arxiv.org/abs/1804.03938`, (2018).