Electronic Communications of the EASST
Volume 076 (2019)

Automated Verification of Critical Systems 2018 (AVoCS 2018)

An Entailment Checker for Separation Logic with Inductive Definitions

Radu Iosif and Cristina Serban

21 pages

Guest Editors: David Pichardie, Mihaela Sighireanu
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

An Entailment Checker for Separation Logic with Inductive Definitions

Radu Iosif and Cristina Serban

CNRS/VERIMAG/Université Grenoble Alpes
Radu.Iosif@univ-grenoble-alpes.fr, Cristina.Serban@univ-grenoble-alpes.fr

Abstract: In this paper, we present Inductor, a checker for entailments between mutually recursive predicates, whose inductive definitions contain ground constraints belonging to the quantifier-free fragment of Separation Logic. Our tool implements a proof-search method for a cyclic proof system that we have shown to be sound and complete, under certain semantic restrictions involving the set of constraints in a given inductive system. Dedicated decision procedures from the DPLL(T)-based SMT solver CVC4 are used to establish the satisfiability of Separation Logic formulae. Given inductive predicate definitions, an entailment query, and a proof-search strategy, Inductor uses a compact tree structure to explore all derivations enabled by the strategy. A successful result is accompanied by a proof, while an unsuccessful one is supported by a counterexample.

Keywords: cyclic proofs, inductive definitions, infinite descent, separation logic

1 Introduction

Inductive definitions play an important role in computing, being an essential component of programming languages, databases, automated reasoning and program verification systems. The main advantage of using inductive definitions is the ability of recursively reasoning about sets of logical objects. The semantics of these definitions is defined in terms of least fixed points of higher-order functions on assignments mapping predicates to sets of models. A natural problem is the *entailment*, that asks whether the least solution of one predicate is included in the least solution of another. Examples of entailments are language inclusion between finite-state (tree) automata, context-free grammars or verification conditions generated by shape analysis tools using specifications of recursive data structures as contracts of program correctness.

The interest for automatic proof generation is two-fold. On one hand, machine-checkable proofs are certificates for the correctness of the answer given by an automated checker, that increase our trust in the reliability of a particular implementation [SOR+13]. On the other hand, the existence of a sound and complete proof system provides a (theoretical) decision procedure for the entailment problem. Assuming that the sets of models and derivations are both recursively enumerable, one can interleave the enumeration of counter-models with the enumeration of derivations; if the entailment holds one finds a finite proof (provided that the proof system is complete), or a finite counterexample, otherwise. Moreover, proof generation can be made effective by providing suitable strategies that limit the possibilities of applying the inference rules and guide the search towards finding a proof or a counterexample.

In this paper we consider inductive systems with constraints written in Separation Logic [Rey02] with the classical (strict) interpretation of spatial atoms. We introduce a set of inference rules tailored for proving inductive entailments in Separation Logic, which have been shown to be sound [IS17] under a ranking assumption for the constraints of the inductive definitions. Completeness is assured under three additional restrictions and only for a particular interpretation of the least solution of the inductive system, taking into account the coverage trees of the heap generated by the inductive definitions. We then describe Inductor [Ser17], a prototype implementation for the more general proof search algorithm given in [IS17] and reprised in §2.4, and discuss several case studies involving both valid and invalid entailments.

Related Work The problem of entailment in Separation Logic with inductive definitions has been approached by other solvers. The generic cyclic proof framework CYCLIST [BGP12] has an instantiation for this fragment and allows for the discovery of inductive arguments during proof construction. CYCLIST builds proofs in which infinite traces can be cut by induction when they satisfy a global trace condition requiring them to visit infinitely many progress points. SLEEK [CDNQ12] and SPEN [ELS17, ESW15] both provide methods of proving entailments by relying on lemmas that relate the inductive definitions. However, SLEEK utilizes a database of user-provided lemmas, while SPEN is able to automatically discover and synthesize concatenation lemmas.

Chu et al. [CJT15] propose a proof system that extends the basic cyclic proof method with a cut rule type that uses previously encountered sequents as inductive hypotheses and applies them by matching and replacing the left with the right-hand side of such a hypothesis. This method can prove entailments between predicates whose coverage trees differ and only soundness is guaranteed. An automata-based decision procedure that also tackles such entailments is given in [IRV14]. This method translates the entailment problem to a language inclusion between tree automata and uses a closure operation on automata to match divergent predicates. Unlike proof search, this method uses existing tree automata inclusion algorithms, which do not produce proof witnesses.

2 Cyclic Proofs for Inductive Entailments in Separation Logic

2.1 Preliminaries

For two integers $0 \leq i \leq j$, we denote by $[i, j]$ the set $\{i, i+1, \ldots, j\}$ and by $[i]$ the set $[1, i]$, where $[0] = \emptyset$. $\|S\|$ denotes the cardinality of the finite set S.

We consider a *signature* $\Sigma = (\Sigma^{\mathrm{s}}, \Sigma^{\mathrm{f}})$, where Σ^{s} is a set of *sort symbols* and Σ^{f} is a set of *function symbols*. For the purpose of this paper, we restrict the signature such that $\Sigma^{\mathrm{s}} = \{\mathsf{Loc}, \mathsf{Bool}\}$ and Σ^{f} contains only equality, the constant symbol nil of sort Loc and the boolean constants $\top$ and $\bot$. Let Var be a countable set of *first order variables*, each $x^{\sigma} \in \mathsf{Var}$ having a sort σ. We write $\mathbf{x}, \mathbf{y}, \ldots$ for both sets and ordered tuples of variables and, for brevity, we use $\in, \cup, \cap, \subseteq$ on tuples such that $x \in \mathbf{x}$ iff x occurs in $\mathbf{x}$, $\mathbf{x} \cup \mathbf{y} = \{x \mid x \in \mathbf{x} \text{ or } x \in \mathbf{y}\}$, $\mathbf{x} \cap \mathbf{y} = \{x \mid x \in \mathbf{x} \text{ and } x \in \mathbf{y}\}$, $\mathbf{x} \subseteq \mathbf{y}$ iff $x \in \mathbf{y}$ for any $x \in \mathbf{x}$.

A term t^{σ} is either a constant or a variable of sort $\sigma \in \Sigma^{\mathrm{s}}$. Separation Logic ($\mathsf{SL}$) formulae are generated by the following syntax:

$\phi ::= \top \mid \bot \mid t_1^\sigma \approx t_2^\sigma \mid \mathsf{emp} \mid x^{\mathsf{Loc}} \mapsto (t_1^{\mathsf{Loc}}, \ldots, t_k^{\mathsf{Loc}}) \mid \phi_1 * \phi_2 \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \exists x.\phi_1 \mid \forall x.\phi_1$ where $k > 0$ is a fixed constant. Given a set of formulae $F = \{\phi_1, \ldots, \phi_n\}$, we write $\circledast F$ for $\phi_1 * \ldots * \phi_n$, which is equivalent to emp if $F = \emptyset$. For a formula ϕ (set of formulae F), $\mathrm{FV}(\phi)$ ($\bigcup_{\phi \in F} \mathrm{FV}(\phi)$) is the set of variables not occurring under a quantifier scope, and $\phi(\mathbf{x})$ ($F(\mathbf{x})$) means that, for every $x \in \mathbf{x}$, we have $x \in \mathrm{FV}(\phi)$ ($x \in \bigcup_{\phi \in F} \mathrm{FV}(\phi)$).

Given sets of variables $\mathbf{x}$ and $\mathbf{y}$, a *flat substitution* $\theta : \mathbf{x} \to \mathbf{y}$ is a mapping of the variables in $\mathbf{x}$ to variables in $\mathbf{y}$. We denote by $\mathbf{x}\theta = \{\theta(x) \mid x \in \mathbf{x}\}$ its image under the substitution θ. For a formula $\phi(\mathbf{x})$, $\phi\theta$ is the formula obtained by replacing each occurrence of $x \in \mathbf{x}$ with the term $\theta(x)$. Observe that θ is always a surjective mapping between $\mathrm{FV}(\phi)$ and $\mathrm{FV}(\phi\theta)$. We lift this notation to sets as $F\theta = \{\phi\theta \mid \phi \in F\}$.

We fix an *interpretation* I such that $I(\top) = \mathit{true}$, $I(\bot) = \mathit{false}$, $I(\mathsf{Loc})$ is a countably infinite set L, and $I(\mathsf{nil}) = \ell^{\mathsf{nil}}$ is a fixed element of L. A *valuation* ν maps each variable x^{Bool} to *true* or *false* and each variable y^{Loc} to an element of L. Given a term t^σ, by writing t_ν^I we mean either $I(t)$ (if t is a constant symbol) or $\nu(t)$ (if t is a variable).

A *heap* is a finite partial mapping $h : \mathsf{L} \rightharpoonup_{\mathsf{fin}} \mathsf{L}^k$ associating locations with k-tuples of locations. We denote by $\mathrm{dom}(h)$ the set of locations on which h is defined, by $\mathrm{img}(h)$ the set of locations occurring in the range of h, and by Heaps the set of heaps. Two heaps h_1 and h_2 are disjoint if $\mathrm{dom}(h_1) \cap \mathrm{dom}(h_2) = \emptyset$. In this case, $h_1 \uplus h_2$ denotes their union, which is undefined if h_1 and h_2 are not disjoint. Given a valuation ν and a heap h, the semantics of SL formulae is defined as:

$$
\begin{array}{lcl}
\nu, h \models^{\mathsf{sl}} t_1 \approx t_2 & \Leftrightarrow & (t_1)_\nu^I = (t_2)_\nu^I \\
\nu, h \models^{\mathsf{sl}} \mathsf{emp} & \Leftrightarrow & h = \emptyset \\
\nu, h \models^{\mathsf{sl}} x \mapsto (t_1, \ldots, t_k) & \Leftrightarrow & h = \{\langle \nu(x), ((t_1)_\nu^I, \ldots, (t_k)_\nu^I)\rangle\} \\
\nu, h \models^{\mathsf{sl}} \phi_1 * \phi_2 & \Leftrightarrow & \exists h_1, h_2 \in \mathsf{Heaps}.\, h = h_1 \uplus h_2 \text{ and } \nu, h_i \models^{\mathsf{sl}} \phi_i, i \in [2]
\end{array}
$$

The semantics of boolean and first order connectives is the usual one, omitted for brevity. Given SL formulae ϕ and ψ, we say that ϕ *entails* ψ (i.e. $\phi \models^{\mathsf{sl}} \psi$) iff $\nu, h \models^{\mathsf{sl}} \phi$ implies $\nu, h \models^{\mathsf{sl}} \psi$, for any valuation ν and heap h.

2.2 Inductive Systems of Predicates in Separation Logic

Let Pred be a countable set of *predicates*, each $p^{\sigma_1 \ldots \sigma_n} \in \mathsf{Pred}$ having an associated tuple of argument sorts. Given a tuple of terms $(t_1^{\sigma_1}, \ldots, t_n^{\sigma_n})$, we call $p(t_1, \ldots, t_n)$ a *predicate atom*. A *predicate rule* is a pair $\langle \{\phi(\mathbf{x}, \mathbf{x}_1, \ldots, \mathbf{x}_n), q_1(\mathbf{x}_1), \ldots, q_n(\mathbf{x}_n)\}, p(\mathbf{x})\rangle$, where $\mathbf{x}, \mathbf{x}_1, \ldots, \mathbf{x}_n$ are tuples whose corresponding sets of variables are pairwise disjoint, ϕ is a formula, called the *constraint*, $p(\mathbf{x})$ is a predicate atom called the *goal* and $q_1(\mathbf{x}_1), \ldots, q_n(\mathbf{x}_n)$ are predicate atoms called *subgoals*. The variables $\mathbf{x}$ are the *goal variables*, whereas $\bigcup_{i=1}^n \mathbf{x}_i$ are the *subgoal variables*.

An *inductive system* $\mathcal{S}$ (system, for short) is a finite set of predicate rules. In this paper, we consider inductive systems whose constraints belong to the SL fragment described in §2.1. We assume w.l.o.g. that each predicate $p \in \mathsf{Pred}$ is the goal of at least one rule of $\mathcal{S}$ and that there are no goals with the same predicate and different goal variables. We write $p(\mathbf{x}) :=_{\mathcal{S}} R_1 \mid \ldots \mid R_m$ when $\{\langle R_1, p(\mathbf{x})\rangle, \ldots, \langle R_m, p(\mathbf{x})\rangle\}$ is the set of all predicate rules in $\mathcal{S}$ with goal $p(\mathbf{x})$. We consider only quantifier-free constraints, in which no disjunction occurs positively and no conjunction occurs negatively, and assume that the set of constraints of a system has a decidable satisfiability

problem. Disjunctions can be eliminated by splitting $\langle\{\phi_1 \vee \ldots \vee \phi_m, q_1(\mathbf{x}_1), \ldots, q_n(\mathbf{x}_n)\}, p(\mathbf{x})\rangle$ into m rules $\langle\{\phi_i, q_1(\mathbf{x}_1), \ldots, q_n(\mathbf{x}_n)\}, p(\mathbf{x})\rangle$, one for each $i \in [m]$.

Example 1 *Consider the* SL *inductive system* $\mathcal{S}_{AB}$ *consisting of the predicate rules:*

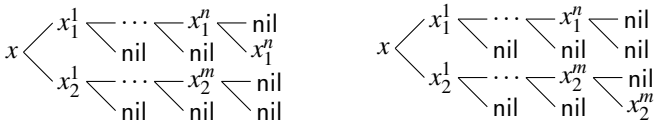
$$
\begin{array}{llll}
p(x) & :=_{\mathcal{S}_{AB}} & x \mapsto (x_1, x_2), p_1(x_1), p_2(x_2) & \\
p_1(x) & :=_{\mathcal{S}_{AB}} & x \mapsto (x_1, \mathsf{nil}), p_1(x_1) \mid x \mapsto (\mathsf{nil}, x) & \\
p_2(x) & :=_{\mathcal{S}_{AB}} & x \mapsto (x_1, \mathsf{nil}), p_2(x_1) \mid x \mapsto (\mathsf{nil}, \mathsf{nil}) & \\
q(x) & :=_{\mathcal{S}_{AB}} & x \mapsto (x_1, x_2), q_1(x_1), q_2(x_2) & \\
 & \mid & x \mapsto (x_1, x_2), q_2(x_1), q_1(x_2) & \\
q_1(x) & :=_{\mathcal{S}_{AB}} & x \mapsto (x_1, \mathsf{nil}), q_1(x_1) \mid x \mapsto (\mathsf{nil}, x) & \\
q_2(x) & :=_{\mathcal{S}_{AB}} & x \mapsto (x_1, \mathsf{nil}), q_2(x_1) \mid x \mapsto (\mathsf{nil}, \mathsf{nil}) &
\end{array}
$$

Broadly speaking, the predicates define binary trees in which the root node points to two lists and one leaf is not nil*, but its position differs in each definition.* □

Given an SL inductive system $\mathcal{S}$, an *assignment* $\mathcal{X}$ maps each predicate $p^{\sigma_1 \ldots \sigma_n} \in \mathsf{Pred}$ to a set $\mathcal{X}(p) \subseteq \mathsf{L}^n \times \mathsf{Heaps}$. For a set $F = \{\phi, q_1(\mathbf{x}_1), \ldots, q_n(\mathbf{x}_n)\}$, where ϕ is an SL formula and $q_1(\mathbf{x}_1), \ldots, q_n(\mathbf{x}_n)$ are predicate atoms, we define $\mathcal{X}(*F) = \{(\nu, h_0 \uplus \biguplus_{i=1}^m h_i) \mid \nu, h_0 \models^{\mathsf{sl}} \phi, (\nu(\mathbf{x}_i), h_i) \in \mathcal{X}(q_i), \forall i \in [m]\}$.

Then $\mathcal{S}$ induces a function $\mathbb{F}^{\mathsf{sl}}_{\mathcal{S}}(\mathcal{X})$ on assignments, which maps each predicate $p \in \mathsf{Pred}$ into the set $\bigcup_{i=1}^m \{(\nu(\mathbf{x}), h) \mid (\nu, h) \in \mathcal{X}(*R_i)\}$, where $p(\mathbf{x}) :=_{\mathcal{S}} R_1 \mid \ldots \mid R_m$. A *solution* of $\mathcal{S}$ is an assignment $\mathcal{X}$ such that $\mathbb{F}^{\mathsf{sl}}_{\mathcal{S}}(\mathcal{X}) \subseteq \mathcal{X}$, where inclusion between assignments is defined pointwise. It can be shown that the set of all assignments, together with the $\subseteq$ relation, is a complete lattice and that $\mathbb{F}^I_{\mathcal{S}}$ is monotone. Therefore, by the Knaster-Tarski theorem, $\mu\mathcal{S}^{\mathsf{sl}} = \bigcap\{\mathcal{X} \mid \mathbb{F}^I_{\mathcal{S}}(\mathcal{X}) \subseteq \mathcal{X}\}$ is the least fixed point of $\mathbb{F}^{\mathsf{sl}}_{\mathcal{S}}$ and the least solution of $\mathcal{S}$.

Example 2 *Considering the* $\mathcal{S}_{AB}$ *inductive system from Example 1, the set* $\mu\mathcal{S}^{\mathsf{sl}}_{AB}(q)$ *consists of trees of both following forms, where* $n \geq 1$ *and* $m \geq 1$*:*



while the set $\mu\mathcal{S}^{\mathsf{sl}}_{AB}(p)$ *contains trees of the first above form, but not the second.* □

We are concerned with the following *entailment problem*: given an inductive system $\mathcal{S}$ and predicates $p^{\sigma_1 \ldots \sigma_m}, q_1^{\sigma_1 \ldots \sigma_m}, \ldots, q_n^{\sigma_1 \ldots \sigma_m}$, is it true that $\mu\mathcal{S}^{\mathsf{sl}}(p) \subseteq \bigcup_{i=1}^n \mu\mathcal{S}^{\mathsf{sl}}(q_i)$? We denote entailment problems as $p \models^{\mathsf{sl}}_{\mathcal{S}} q_1, \ldots, q_n$.

Example 3 *Given the least solution of the* $\mathcal{S}_{AB}$ *inductive system from Example 2, observe that the entailment* $p \models^{\mathsf{sl}}_{\mathcal{S}_{AB}} q$ *holds, while* $q \models^{\mathsf{sl}}_{\mathcal{S}_{AB}} p$ *does not.* □

2.3 Tree Automata Inclusion as Cyclic Proof Search

We consider top-down nondeterministic finite tree automata (NFTA), where a tree over a ranked alphabet $\mathfrak{F}$ is either a symbol $a \in \mathfrak{F}$ of rank (or arity) 0, or $f(t_1, \ldots, t_n)$ such that $f \in \mathfrak{F}$ is of rank n and $t_1, \ldots, t_n$ are, in turn, trees. The actions of an NFTA are described by transition rules

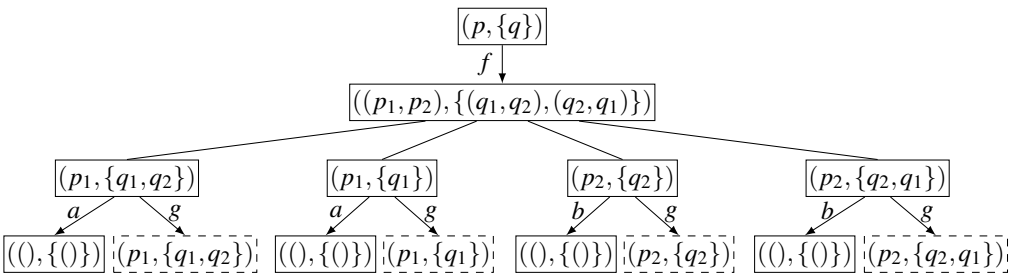
$q \xrightarrow{f} (q_1,\ldots,q_n)$, with the following meaning: if the automaton is in state q and the input is a tree $f(t_1,\ldots,t_n)$, then it moves simultaneously on each t_i changing its state to q_i, for all $i \in [n]$. A tree is *accepted* by an automaton A if each leaf can be eventually read by a transition of the form $q \xrightarrow{a} ()$. The language of a state q in A, denoted $\mathcal{L}(A,q)$, is the set of trees accepted by A starting with state q.

An NFTA can be naturally viewed as an inductive system, where predicates represent states and predicate rules are obtained directly from transition rules. For instance, $q \xrightarrow{f} (q_1,\ldots,q_n)$ can be written as $\langle \{x \approx f(x_1,\ldots,x_n), q_1(x_1),\ldots,q_n(x_n)\}, q(x)\rangle$, where variables range over trees and the function symbols are interpreted in the canonical (Herbrand) sense. To further obtain an SL inductive system, an encoding of the constraints in each predicate rule using SL connectives is required.

Example 4 *The* SL *inductive system* $\mathcal{S}_{AB}$ *from Example 1 encodes two NFTA* A *and* B *with states* $\{p,p_1,p_2\}$ *and* $\{q,q_1,q_2\}$, *respectively, where* p *and* q *are initial states, using the alphabet* $\{f(,),g(),a,b\}$ *and having the transition sets* $\{p \xrightarrow{f} (p_1,p_2), p_1 \xrightarrow{g} p_1,\ p_1 \xrightarrow{a} (), p_2 \xrightarrow{g} p_2, p_2 \xrightarrow{b} ()\}$ *and* $\{q \xrightarrow{f} (q_1,q_2), q \xrightarrow{f} (q_2,q_1), q_1 \xrightarrow{g} q_1, q_1 \xrightarrow{a} (), q_2 \xrightarrow{g} q_2,\ q_2 \xrightarrow{b} ()\}$, *respectively. We encoded the binary symbol* f *as* $x \mapsto (x_1,x_2)$, *the unary symbol* g *as* $x \mapsto (x_1,\mathsf{nil})$, *and the constant symbols* a *and* b *as* $x \mapsto (\mathsf{nil},x)$ *and* $x \mapsto (\mathsf{nil},\mathsf{nil})$, *respectively, where* x, x_1 *and* x_2 *are always allocated, thus different from* nil. □

Since language inclusion is decidable for NFTA [CDG+05, Corollary 1.7.9], we leverage an existing algorithm for this problem by Holík et al. [HLSV11] to build a set of inference rules and derive a proof search technique. This algorithm searches for counterexamples of the inclusion problem $\mathcal{L}(A,p) \subseteq \bigcup_{i=1}^{n} \mathcal{L}(B,q_i)$ by enumerating pairs $(r,\{s_1,\ldots,s_m\})$, where r is a state that can be reached via a series of transitions from p, and $\{s_1,\ldots,s_m\}$ are all the states that can be reached via the same series of transitions from $q_1,\ldots,q_k$. A counterexample is found when reaching a pair $(r,\{s_1,\ldots,s_m\})$ such that there exists a transition $r \xrightarrow{f} (r_1,\ldots,r_k)$, but there is no transition $s_i \xrightarrow{f} (s_i^1,\ldots,s_i^k)$ for any $i \in [m]$.

Example 5 *Consider the NFTA* A *and* B *from Example 4. To check* $\mathcal{L}(A,p) \subseteq \mathcal{L}(B,q)$, *we start with* $(p,\{q\})$. *A possible run is:*



The algorithm performs two types of moves: transitions and split actions. The arrows labeled by symbols f,g,a *and* b *are transitions, for instance the arrow labeled by* f *takes* p *into the tuple* (p_1,p_2) *by the transition* $p \xrightarrow{f} (p_1,p_2)$ *and* $\{q\}$ *into the set of tuples* $\{(q_1,q_2),(q_2,q_1)\}$,

by the transitions $q \xrightarrow{f} (q_1, q_2)$ *and* $q \xrightarrow{f} (q_2, q_1)$. *However, the pair* $((p_1, p_2), \{(q_1, q_2), (q_2, q_1)\})$ *is problematic because it asserts that* $\mathcal{L}(A, p_1) \times \mathcal{L}(A, p_2) \subseteq \mathcal{L}(B, q_1) \times \mathcal{L}(B, q_2) \cup \mathcal{L}(B, q_2) \times \mathcal{L}(B, q_1)$. *Using several properties of the Cartesian product [HLSV11, Theorem 1] there are multiple ways to split this proof obligation into several simpler conjunctive subgoals. If at least one split move leads to a proof, then the inital proof obligation holds. The split move used above simultaneously considers* $(p_1, \{q_1, q_2\})$, $(p_1, \{q_1\})$, $(p_2, \{q_2\})$ *and* $(p_2, \{q_2, q_1\})$, *together asserting that* $\mathcal{L}(A, p_1) \subseteq \mathcal{L}(B, q_1)$ *and* $\mathcal{L}(A, p_2) \subseteq \mathcal{L}(B, q_2)$. *The other options are:*

(1) $(p_1, \{q_1, q_2\})$ $(p_1, \{q_1\})$, $(p_1, \{q_2\})$, *and* $(p_2, \{q_2, q_1\})$;

(2) $(p_1, \{q_1, q_2\})$, $(p_2, \{q_1\})$, $(p_1, \{q_2\})$, *and* $(p_2, \{q_2, q_1\})$;

(3) $(p_1, \{q_1, q_2\})$, $(p_2, \{q_1\})$, $(p_2, \{q_2\})$, *and* $(p_2, \{q_2, q_1\})$.

The algorithm does not expand nodes (p, S) *with* $p \in S$, *for which inclusion holds trivially, or having a predecessor* (p, S') *with* $S' \subseteq S$ *(enclosed in dashed boxes), since any counterexample that can be found from* (p, S) *could have been discovered from* (p, S'). □

2.4 A Proof Search Semi-algorithm

$$(IR)\ \frac{\Gamma_1 \vdash \Delta_1 \ \ldots \ \Gamma_n \vdash \Delta_n}{\begin{array}{c}\Gamma \vdash \Delta \\ \vdots\ \mathrm{c} \\ \Gamma_p \vdash \Delta_p\end{array}}\ \text{side conditions}$$

We denote *sequents* as $\Gamma \vdash \Delta$, where Γ and Δ are sets of formulae and commas are read as set union, thus contraction rules are not necessary. We omit braces and a sequent of the form $p(\mathbf{x}) \vdash q_1(\mathbf{x}), \ldots, q_n(\mathbf{x})$ is called *basic*. An *inference rule schema IR* is a possibly infinite set of inference rules, called *instances* of the schema, sharing the same pattern. An inference rule has *antecedents* $\Gamma_1 \vdash \Delta_1, \ldots, \Gamma_n \vdash \Delta_n$, and a *consequent* $\Gamma \vdash \Delta$. We write $\top$ for an empty antecedent list and an inference rule without antecedents may have a *pivot* $\Gamma_p \vdash \Delta_p$, which is an ancestor of the consequent or, in other words, a sequent preceding the consequent in the transitive closure of the consequent-antecedent relation. The sequence of inference rules applied along the path between the pivot and the consequent is subject to a *pivot condition* **C**.

A *proof system* is a set $\mathcal{R}$ of inference rule schemata. A *derivation built with* $\mathcal{R}$ is a possibly infinite tree $\mathcal{D} = (V, v_0, S, R, P, B)$, where V is a set of vertices (or nodes) and $v_0 \in V$ is the root. Every $v \in V$ is labeled with sequent $S(v)$ and an inference rule schema $R(v) \in \mathcal{R}$ such that $S(v)$ is the consequent of the instance of $R(v)$ applied at v. Moreover, $B(v) \in V$ is a node such that $S(B(v))$ is the pivot for $R(v)$, if it has one. We call $(v, B(v))$ a *backlink*. If $v \neq v_0$, $P(v)$ is the parent of v in the derivation and $S(v)$ is an antecedent for the instance of $R(P(v))$ with consequent $S(P(v))$. A *proof* is a finite derivation in which $S(v) = \top$ for all leaves $v \in V$ – i.e. on every branch of the derivation, the last inference rule application generates an empty list of antecedents.

Given an inductive system $\mathcal{S}$ and predicates $p^{\sigma_1,\ldots,\sigma_n}, q_1^{\sigma_1,\ldots,\sigma_n}, \ldots, q_k^{\sigma_1,\ldots,\sigma_n} \in \mathsf{Pred}$, a proof system $\mathcal{R}$ is: (i) *sound* if, for any proof $\mathcal{D} = (V, v_0, S, R, P, B)$ with $S(v_0) = p(\mathbf{x}) \vdash q_1(\mathbf{x}), \ldots, q_n(\mathbf{x})$, we have $p \models^{\mathsf{sl}}_{\mathcal{S}} q_1, \ldots, q_n$, (ii) *complete* if $p \models^{\mathsf{sl}}_{\mathcal{S}} q_1, \ldots, q_n$ implies the existence of a proof $\mathcal{D} = (V, v_0, S, R, P, B)$ with $S(v_0) = p(\mathbf{x}) \vdash q_1(\mathbf{x}), \ldots, q_n(\mathbf{x})$.

A sequence $\pi = v_1, \ldots, v_n$ of vertices is a *trace* if, for any $i \in [n-1]$, either $v_i = P(v_{i+1})$ or $v_{i+1} = B(v_i)$. π is *path* if only the former condition holds and, moreover, a *direct path* if $v_1 = B(v_n)$. We denote by $\Lambda(\pi) = R(v_1), \ldots, R(v_n - 1)$ the sequence of inference rule schemata

applied between v_1 and v_n. An inference rule schema IR is *applicable on* v_n and π if there exists an instance ir of IR whose consequent matches $S(v_n)$ and whose pivot (if it exists) matches $S(v_i)$ for some $i < n$, such that both the side conditions of ir are satisfied and and $\Lambda(v_i, \ldots, v_n)$ abides by its pivot condition. A *strategy* is a set $\mathbf{S}$ of inference rule schemata sequences. A sequence s is a *valid prefix* for $\mathbf{S}$ if there exists another sequence s' such that their concatenation $s \cdot s' \in \mathbf{S}$. A derivation (proof) $\mathcal{D}$ is an $\mathbf{S}$-derivation ($\mathbf{S}$-proof) if, for each maximal path π in $\mathcal{D}$, $\Lambda(\pi) \in \mathbf{S}$.

Algorithm 1 Proof search semi-algorithm.

data structure: $\texttt{Node}(\mathit{sequent}, \mathit{rule}, \mathit{parent}, \mathit{pivot}, \mathit{children})$, where:

- *sequent* is the sequent that labels the node,
- *rule* is the inference rule with consequent *sequent*.
- *parent* is the link to the parent of the node,
- *pivot* is the pivot for the instance of *rule* applied on *sequent*
- *children* is the list of children nodes

input: inductive system $\mathcal{S}$, sequent $p(\mathbf{x}) \vdash q_1(\mathbf{x}), \ldots, q_n(\mathbf{x})$, proof system $\mathcal{R}$, strategy $\mathbf{S}$

output: an $\mathbf{S}$-proof built with $\mathcal{R}$, whose root is labeled with sequent $p(\mathbf{x}) \vdash q_1(\mathbf{x}), \ldots, q_n(\mathbf{x})$

1: $\mathsf{Root} \leftarrow \texttt{Node}(p(\mathbf{x}) \vdash q_1(\mathbf{x}), \ldots, q_n(\mathbf{x}), \mathit{null}, \mathit{null}, \mathit{null}, [])$
2: $\mathsf{WorkList} \leftarrow \{\mathsf{Root}\}$
3: **while** $\mathsf{WorkList} \neq \emptyset$ **do**
4: remove N from $\mathsf{WorkList}$
5: let π be the path between Root and N
6: let $\mathcal{R}_{\mathsf{N}} \subseteq \mathcal{R}$ be the inference rule schemata applicable on N and π
7: let $\mathcal{R}^0_{\mathsf{N}} \subseteq \mathcal{R}_{\mathsf{N}}$ be the subset of $\mathcal{R}_{\mathsf{N}}$ with empty antecedent lists
8: **if** $\Lambda(\pi) \cdot IR$ is a valid prefix of $\mathbf{S}$ for some $IR \in \mathcal{R}^0_{\mathsf{N}}$ **then**
9: let ir be an instance of IR such that $\mathsf{N}.\mathit{sequent}$ is the consequent of ir
10: $\mathsf{N}.\mathit{rule} \leftarrow IR$
11: **if** ir has pivot $\mathsf{N}'.\mathit{sequent}$ for some $\mathsf{N}' \in \pi$ **then** $\mathsf{N}.\mathit{pivot} \leftarrow \mathsf{N}'$
12: mark N as *Closed*
13: **if** N not *Closed* and $\Lambda(\pi) \cdot IR$ is a valid prefix of $\mathbf{S}$ for some $IR \in \mathcal{R}_{\mathsf{N}}$ **then**
14: let ir be an instance of IR such that $\mathsf{N}.\mathit{sequent}$ is the consequent of ir
15: **for** each antecedent $\Gamma' \vdash \Delta'$ of ir **do**
16: $\mathsf{N}' \leftarrow \texttt{Node}(\Gamma' \vdash \Delta', \mathit{null}, \mathsf{N}, \mathit{null}, [])$
17: add N' to $\mathsf{N}.\mathit{children}$ and to $\mathsf{WorkList}$
18: **if** $\mathsf{N}.\mathit{children}$ is empty **then** mark N as *Closed*

Given an input sequent $p(\mathbf{x}) \vdash q_1(\mathbf{x}), \ldots, q_n(\mathbf{x})$, a set $\mathcal{R}$ of inference rules and a strategy $\mathbf{S}$, the proof search semi-algorithm 1 uses a worklist iteration to build a derivation of $p(\mathbf{x}) \vdash q_1(\mathbf{x}), \ldots, q_n(\mathbf{x})$. When a node is removed from the worklist, it chooses (non-deterministically) an inference rule and an instance whose consequent matches the sequent of the node, if one exists. To speed up termination, inference rule schemata without antecedents are considered eagerly (line 8). If a proof of the input sequent exists, then there exists a finite execution of the semi-algorithm 1 leading to it.

2.5 The Set $\mathcal{R}^{\mathsf{sl}}_{\not\mathsf{ind}}$ of Inference Rules for Separation Logic Entailments

Figure 1 gives a set $\mathcal{R}^{\mathsf{sl}}_{\not\mathsf{ind}}$ of inference rule schemata for the entailment problem in SL, which generalize the transitions and split actions performed by the NFTA language inclusion algorithm described in §2.3. To shorten the presentation, we write $\langle \Gamma_i \vdash \Delta_i \rangle_{i=1}^{n}$ for $\Gamma_1 \vdash \Delta_1, \ldots, \Gamma_n \vdash \Delta_n$.

(LU) and (RU) unfold a predicate atom $p(\mathbf{x})$ by replacing it with the set of predicate rules $p(\mathbf{x}) :=_{\mathcal{S}} R_1(\mathbf{x}) \mid \ldots \mid R_n(\mathbf{x})$, with goal variables $\mathbf{x}$ and fresh subgoal variables. The left unfolding yields a set of sequents, one for each $R_i(\mathbf{x})$ with $i \in [n]$, that must be all proved, whereas the right unfolding replaces $p(\mathbf{x})$ with the set of formulae obtained from $R_1(\mathbf{x}) \mid \ldots \mid R_n(\mathbf{x})$ in which the subgoal variables are existentially quantified.

$$\text{(LU)}\ \frac{\langle R_i(\mathbf{x},\mathbf{y}_i), \Gamma \setminus p(\mathbf{x}) \vdash \Delta \rangle_{i=1}^{n}}{\Gamma \vdash \Delta}\ \begin{array}{l} p(\mathbf{x}) \in \Gamma,\ p(\mathbf{x}) :=_{\mathcal{S}} R_1(\mathbf{x},\mathbf{y}_1) \mid \ldots \mid R_n(\mathbf{x},\mathbf{y}_n) \\ \mathbf{y}_1,\ldots,\mathbf{y}_n \text{ fresh variables} \end{array}$$

$$\text{(RU)}\ \frac{\Gamma \vdash \{\exists \mathbf{y}_i \,.\, * R_i(\mathbf{x},\mathbf{y}_i)\}_{i=1}^{n}, \Delta \setminus p(\mathbf{x})}{\Gamma \vdash \Delta}\ \begin{array}{l} p(\mathbf{x}) \in \Delta,\ \mathbf{y}_1,\ldots,\mathbf{y}_n \text{ fresh} \\ p(\mathbf{x}) :=_{\mathcal{S}} R_1(\mathbf{x},\mathbf{y}_1) \mid \ldots \mid R_n(\mathbf{x},\mathbf{y}_n) \end{array}$$

$$\text{(RD)}\ \frac{p_1(\mathbf{x}_1), \ldots, p_n(\mathbf{x}_n) \vdash \{Q_j\theta \mid \theta \in S_j\}_{j=1}^{i}}{\phi(\mathbf{x},\mathbf{x}_1,\ldots,\mathbf{x}_n), p_1(\mathbf{x}_1), \ldots, p_n(\mathbf{x}_n) \vdash \{\exists \mathbf{y}_j \,.\, \psi_j(\mathbf{x},\mathbf{y}_j) * Q_j(\mathbf{y}_j)\}_{j=1}^{k}}\ \begin{array}{l} \phi \models^{\mathsf{sl}} \bigwedge_{j=1}^{i} \exists \mathbf{y}_j . \psi_j \\ \phi \not\models^{\mathsf{sl}} \bigvee_{j=i+1}^{k} \exists \mathbf{y}_j . \psi_j \\ S_j \subseteq \mathrm{Sk}(\phi,\psi_j), j \in [i] \end{array}$$

$$\text{(AX)}\ \frac{\top}{\Gamma \vdash \Delta}\ * \Gamma \models^{\mathsf{sl}} \bigvee \Delta \qquad \text{(ID)}\ \frac{\top}{\Gamma\theta \vdash \Delta'\theta}\ \theta \text{ flat injective substitution},\ \Delta \subseteq \Delta'$$

$$\vdots\ (\mathcal{R}^{\mathsf{sl}}_{\not\mathsf{ind}})^* \cdot \mathrm{LU} \cdot (\mathcal{R}^{\mathsf{sl}}_{\not\mathsf{ind}})^*$$

$$\Gamma \vdash \Delta$$

$$\text{(SP)}\ \frac{\langle p_{\bar{\imath}_j}(\mathbf{x}) \vdash \{q^{\ell}_{\bar{\imath}_j}(\mathbf{x}) \mid \ell \in [k],\ f_j(\overline{Q}_\ell) = \bar{\imath}_j\}\rangle_{j=1}^{n^k}}{p_1(\mathbf{x}_1), \ldots, p_n(\mathbf{x}_n) \vdash Q_1(\mathbf{x}_1,\ldots,\mathbf{x}_n), \ldots, Q_k(\mathbf{x}_1,\ldots,\mathbf{x}_n)}\ \begin{array}{l} \forall i,j \in [n] \,.\, \mathbf{x}_i \cap \mathbf{x}_j = \emptyset,\ \bar{\imath} \in [n]^{n^k} \\ Q_i = *_{j=1}^{n} q^i_j(\mathbf{x}_j), \overline{Q}_i = \langle q^i_1, \ldots, q^i_n \rangle \\ \mathcal{F}(\overline{Q}_1,\ldots,\overline{Q}_k) = \{f_1,\ldots,f_{n^k}\} \end{array}$$

Figure 1: The set $\mathcal{R}^{\mathsf{sl}}_{\not\mathsf{ind}}$ of inference rule schemata for inductive entailments in SL.

(RD) eliminates constraints from both sides of a sequent. The existentially quantified variables on the right-hand side are replaced using a (subset of) the finite set $\mathrm{Sk}(\phi,\psi_j) = \{\theta : \bigcup_{i=1}^{k} \mathbf{y}_i \to \{\mathsf{nil}\} \cup \mathbf{x} \cup \bigcup_{i=1}^{n} \mathbf{x}_i) \mid \phi \models^{\mathsf{sl}} \psi_j\theta\}$ of substitutions that witness the entailments $\phi(\mathbf{x},\mathbf{x}_1,\ldots,\mathbf{x}_n) \models^{\mathsf{sl}} \exists \mathbf{y}_j \,.\, \psi_j$ between the left and right constraints.

A transition move in the language inclusion algorithm of [HLSV11] (Example 5) performs (LU), (RU) and (RD) all at once. This is natural because the transition rules of tree automata are controlled uniquely by the function symbols labeling the root of the current input tree, which can be matched unambiguously. When considering more general constraints, matching amounts to discovering non-trivial substitutions that prove an entailment between existentially quantified constraints.

(SP) generalizes the split moves performed by the language inclusion algorithm of [HLSV11] and breaks a sequent $p_1(\mathbf{x}_1), \ldots, p_n(\mathbf{x}_n) \vdash Q_1(\mathbf{x}_1,\ldots,\mathbf{x}_n), \ldots, Q_n(\mathbf{x}_1,\ldots,\mathbf{x}_n)$ into basic sequents. Given tuples $\{\overline{Q}_1,\ldots,\overline{Q}_k\} \subseteq \mathsf{Pred}^n$ with $n \geq 1$, a *choice function* f maps each tuple $\overline{Q}_i$ into an index $f(\overline{Q}_i) \in [n]$ corresponding to a position in the tuple. Let $\mathcal{F}(\overline{Q}_1,\ldots,\overline{Q}_k)$ be the set of such choice functions, having cardinality $n^k \leq n^{\|\mathsf{Pred}\|^n}$. Given a tuple $\bar{\imath} \in [n]^{n^k}$, associating a value in $[n]$ to each choice function $f \in \mathcal{F}(\overline{Q}_1,\ldots,\overline{Q}_k)$, there exists an application of (SP) generating n^k antecedents with left hand-side $p_{\bar{\imath}_j}(\mathbf{x}_{\bar{\imath}_j})$, $j \in [n^k]$ and right hand-side consisting of all predicate

atoms $q^{\ell}_{\bar{\iota}_j}(\mathbf{x}_{\bar{\iota}_j}), \ell \in [k]$ obtained from predicates at position $\bar{\iota}_j$ in the tuples $\overline{Q_\ell}$ which are mapped to $\bar{\iota}_j$ by the choice function f_j. In order to obtain a proof, there must exist some application of (SP) – and, therefore, some $\bar{\iota} \in [n]^{n^k}$ – for which all the generated antecedents can be proven. As shown in [HLSV11, Section 3], the tuples $\bar{\iota} \in [n]^{n^k}$ encode the transformation of a formula from CNF to DNF and, as such, not all are relevant. More precisely, any $\bar{\iota}$ for which there exists $j \in [n^k]$ such that $\bar{\iota}_j \notin \mathrm{img}(f_j)$ can be discarded.

(AX) closes the current branch of the proof if the sequent can be proved using a decision procedure for the underlying constraint logic, by treating predicate symbols as uninterpreted boolean functions. This is a generalization of encountering a pair (p,S) with $p \in S$ in the NFTA language inclusion algorithm of [HLSV11].

(ID) introduces backlinks in a derivation, from the consequent $\Gamma\theta \vdash \Delta'\theta$ to a pivot $\Gamma \vdash \Delta$. The pivot condition $(\mathcal{R}^{\mathsf{sl}}_{\mathsf{ind}})^* \cdot \mathrm{LU} \cdot (\mathcal{R}^{\mathsf{sl}}_{\mathsf{ind}})^*$ requires that (LU) must be applied on the direct path between the pivot and the consequent. Observe that, if $\Gamma\theta \vdash \Delta'\theta$ denotes a non-valid entailment, there exists $(\nu,h) \in \mu\mathcal{S}^{\mathsf{sl}}(*\Gamma\theta) \setminus \mu\mathcal{S}^{\mathsf{sl}}(\bigvee \Delta'\theta)$. Since θ is surjective by construction and injective by the side condition, the restriction of θ to $\mathrm{FV}(\Gamma \cup \Delta')$ has an inverse and, because $\Delta' \subseteq \Delta$, we obtain that $(\nu \circ \theta^{-1}, h) \in \mu\mathcal{S}^{\mathsf{sl}}(*\Gamma) \setminus \mu\mathcal{S}^{\mathsf{sl}}(\bigvee \Delta)$ is a counterexample for the pivot.

The local soundness of $\mathcal{R}^{\mathsf{sl}}_{\mathsf{ind}} \setminus \{\mathrm{ID}\}$ is given by [IS17, Lemma 15], whereas the soundness of proofs containing (ID) is established by [IS17, Theorem 6] through the following argument. If the root sequent of a proof denotes an entailment that admits a counterexample, then, by the local soundness, there exists a path in the proof on which this counterexample can be propagated. This path may not end with an application of (AX), as it would violate its side condition, and, thus, must end with an application of (ID), which allows it to be extended to an infinite trace. Then, using the reasoning above to additionally propagate counterexamples through a backlinks, we obtain that the counterexample for the root sequent can also be propagated along a trace with an infinite number of direct paths [IS17, Proposition 1]. We use an additional *ranking* assumption given by a pre-established well-founded ordering of the counterexamples. In SL, we consider the subheap ordering, where $h_1 \trianglelefteq h_2$ iff there exists $h \in \mathsf{Heaps}$ such that $h_2 = h_1 \uplus h$ and $h_1 \triangleleft h_2$ if, moreover, $h \neq \emptyset$. An SL inductive system is ranked if the constraints of every predicate rule with at least one subgoal do not admit an empty heap model. Since (LU) is required on each direct path, this leads to an infinite sequence of counterexamples $(\nu_1,h_1),(\nu_2,h_2),\ldots$ with strictly decreasing heaps $h_1 \rhd h_2 \rhd \ldots$ (see [IS17, Lemma 15] and the proof of [IS17, Theorem 6]). However, since the subheap ordering is well-founded, the existence of such sequences is prohibited. We have reached a contradiction and may conclude that there was no counterexample to begin with.

Analogously, the language inclusion algorithm of [HLSV11] stops expanding a branch in the search tree whenever it has discovered a pair (p,S) that has a predecessor (p,S'), with $S' \subseteq S$. Just as for the (ID) inference rules, backtracking relies on the Infinite Descent principle [Bus18], that forbids infinitely descending sequences of counterexamples.

Example 6 *Given the system $\mathcal{S}_{AB}$ from Example 1, we can use $\mathcal{R}^{\mathsf{sl}}_{\mathsf{ind}}$ to build a proof for $p(x) \vdash q(x)$, partially shown below – we only include the subproof for the first sequent obtained after split. Note the similarities with the proof tree in Example 5.*

$$\dfrac{\dfrac{\dfrac{\top}{x \mapsto (\mathsf{nil},x) \vdash x \mapsto (\mathsf{nil},x), q_2(x), \exists y_1 \,.\, x \mapsto (y_1,\mathsf{nil}) * q_1(y_1)}\,\mathrm{AX}}{x \mapsto (\mathsf{nil},x) \vdash q_1(x), q_2(x)}\,\mathrm{RU} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\top}{p_1(x) \vdash q_1(x), q_2(x)}\,\mathrm{ID}}{x \mapsto (x_1,\mathsf{nil}), p_1(x_1) \vdash x \mapsto (\mathsf{nil},x), \exists y_1 \,.\, x \mapsto (y_1,\mathsf{nil}) * q_1(y_1), x \mapsto (\mathsf{nil},\mathsf{nil}), \exists y_1 \,.\, x \mapsto (y_1,\mathsf{nil}) * q_2(y_1)}\,\mathrm{RD}}{x \mapsto (x_1,\mathsf{nil}), p_1(x_1) \vdash x \mapsto (\mathsf{nil},x), \exists y_1 \,.\, x \mapsto (y_1,\mathsf{nil}) * q_1(y_1), q_2(x)}\,\mathrm{RU}}{x \mapsto (x_1,\mathsf{nil}), p_1(x_1) \vdash q_1(x), q_2(x)}\,\mathrm{RU}}{p_1(x) \vdash q_1(x), q_2(x)}\,\mathrm{LU}$$

$$\dfrac{\dfrac{\dfrac{p_1(x) \vdash q_1(x), q_2(x) \quad p_1(x) \vdash q_1(x) \quad p_2(x) \vdash q_2(x) \quad p_2(x) \vdash q_2(x), q_1(x)}{p_1(x_1), p_2(x_2) \vdash q_1(x_1) * q_2(x_2), q_2(x_1) * q_1(x_2)}\,\mathrm{SP}}{x \mapsto (x_1,x_2), p_1(x_1), p_2(x_2) \vdash \exists y_1,y_2 \,.\, x \mapsto (y_1,y_2) * q_1(y_1) * q_2(y_2), \exists y_1,y_2 \,.\, x \mapsto (y_1,y_2) * q_2(y_1) * q_1(y_2)}\,\mathrm{RD}}{x \mapsto (x_1,x_2), p_1(x_1), p_2(x_2) \vdash q(x)}\,\mathrm{RU}}{p(x) \vdash q(x)}\,\mathrm{LU}$$

For (SP)*, let* $\overline{Q}_1 = (q_1,q_2)$ *and* $\overline{Q}_2 = (q_2,q_1)$ *be the tuples of predicates on the right-hand side. The set of choice functions is* $\mathcal{F}(\overline{Q}_1,\overline{Q}_2) = \{f_1 = \{(\overline{Q}_1,1),(\overline{Q}_2,1)\}, f_2 = \{(\overline{Q}_1,1),(\overline{Q}_2,2)\}, f_3 = \{(\overline{Q}_1,2),(\overline{Q}_2,1)\}, f_4 = \{(\overline{Q}_1,2),(\overline{Q}_2,2)\}\}$*. Out of the 16 index choice tuples for* $\mathcal{F}(\overline{Q}_1,\overline{Q}_2)$*, only* $(1,1,1,2)$*,* $(1,1,2,2)$*,* $(1,2,1,2)$ *and* $(1,2,2,2)$ *are relevant. To obtain the above proof, we chose* $\bar{\iota} = (1,1,2,2)$*.* □

Only the ranking assumption is necessary to ensure soundness of $\mathcal{R}^{\mathsf{sl}}_{\mathsf{ind}}$. Three additional restrictions required for completeness are given in [IS17, Section 4.2]. Effectively checking whether a given inductive system satisfies these restrictions requires the existence of a decision procedure for the $\exists^*\forall^*$-quantified fragment of the underlying logic. In general, this problem is undecidable for SL [EIP18, Theorem 1], but the fragment decribed in §2.1 omits the $-\!\!*$ operator (primarily responsible for loss of decidability), while the $\mapsto$ operator only maps elements in L to tuples in L^k. As such, the satisfiability of $\exists^*\forall^*$-quantified formulae is PSPACE-complete in the fragment we consider [EIP18, Theorems 2 and 3]. A suitable decision procedure for this fragment of SL is given in [RIS17]. Completeness is then assured for entailments involving inductive definitions which generate matching coverage trees of the heap (see [IS17, Section 4.3]).

Example 7 *Consider the following definitions for doubly-linked lists:*

$$\begin{array}{rl} dlls(hd,p,tl,n) :=_{\mathcal{S}} & hd \approx tl \wedge hd \mapsto (p,n) \\ \mid & hd \mapsto (p,x), dlls(x,hd,tl,n) \\ dlls^r(hd,p,tl,n) :=_{\mathcal{S}} & hd \approx tl \wedge hd \mapsto (p,n) \\ \mid & tl \mapsto (n,tl'), dlls^r(hd,p,x,tl) \end{array} \qquad \begin{array}{rl} t_1 = & \{(1,\langle 0,2\rangle)\} \\ & | \\ & \{(2,\langle 1,3\rangle)\} \\ & | \\ & \{(3,\langle 2,4\rangle)\} \end{array} \quad \begin{array}{rl} t_2 = & \{(3,\langle 2,4\rangle)\} \\ & | \\ & \{(2,\langle 1,3\rangle)\} \\ & | \\ & \{(1,\langle 0,2\rangle)\} \end{array}$$

The predicate dlls unfolds the list starting at the head, while $dlls^r$ *unfolds it starting at the tail. Both* $dlls \models^{\mathsf{sl}}_{\mathcal{S}} dlls^r$ *and* $dlls^r \models^{\mathsf{sl}}_{\mathcal{S}} dlls$ *hold, but they cannot be proven using our inference rules. Take, for instance, the tuple* $\bar{\ell} = \langle 0,1,3,4\rangle$ *and the heap* $h = \{(1,\langle 0,2\rangle),(2,\langle 1,3\rangle),(3,\langle 2,4\rangle)\}$*. The pair* $(\bar{\ell},h)$ *belongs to both* $\mu\mathcal{S}^{\mathsf{sl}}(dlls)$ *and* $\mu\mathcal{S}^{\mathsf{sl}}(dlls^r)$*, but dlls generates the coverage tree* t_1 *for h, while* $dlls^r$ *generates the coverage tree* t_2*. Since the trees do not match,* $\mathcal{R}^{\mathsf{sl}}_{\mathsf{ind}}$ *cannot built proofs for either entailment.*

3 An Inductive Entailment Checker for Separation Logic

In this section we describe Inductor, an entailment checker tool that implements the proof-search semi-algorithm 1 from §2.4, using the set $\mathcal{R}^{\mathsf{sl}}_{\mathsf{ind}}$ of inference rules for inductive entailments in SL. Inductor is written in C++ and uses the DPLL(T)-based SMT solver CVC4 [BCD+11] as a back-end that it queries in order to establish the satisfiability of SL formulae, in which the occurrences of inductive predicates are treated as uninterpreted functions. More specifically, these queries are handled by the decision procedures provided in [RISK16, RIS17] and integrated into CVC4.

The inputs mainly handled by Inductor are SMT-LIB scripts, abiding by the SMT-LIB Standard: Version 2.6 [BFT17]. Theory and logic files are loaded automatically, based on the logic set in the input script being handled. Additionally, proof strategies are specified as nondeterministic finite word automata (NFA), in a language similar to that accepted by libVATA [LSV+] (for more details, see Appendix A.1). The front-end interprets these input files using custom parsers constructed with Flex[1] and Bison[2].

3.1 A Breadth-First Proof Search Implementation

The proof search method sketched by algorithm 1 is reliant on the choice of *IR* made at lines 8 and 13. Whenever there are more than one applicable inference rules, only one is selected and the rest are discarded. Furthermore, as is the case for SP, some inference rules can have multiple possible instances for the same sequent, where only one is required to succeed in obtaining a proof. Algorithm 1 again only chooses one of them. In our implementation, we wanted to explore all the potential derivations resulting from the inference rule instances available at any point. Moreover, since we use a queue for the nodes still needing to be explored, we generate derivations in a breadth-first fashion. Thus, proofs or counterexample are obtained from the shortest possible paths.

We use a different tree-like structure to compactly store all the derivations explored. This structure accepts two types of nodes, depicted in Figure 2, which represent sequents (`SNode`) and inference rule instances (`RNode`), respectively. The node types alternate in the tree, thus an `SNode` only has `RNode` children, and vice-versa.

`SNode` { *sequent* : A sequent $\Gamma \vdash \Delta$,
states : A list of states in the strategy
parent : `RNode` parent of the current node
children : A list of `RNode` children }

`RNode` { *rule* : An inference rule schema
pivot : `SNode` pivot for this *rule* instance,
parent : `SNode` parent of the current node,
children : A list of `SNode` children }

Figure 2: The data structures representing sequents and inference rule instances

With these new data structures, we say that an inference rule $IR \in \mathcal{R}^{\mathsf{sl}}_{\mathsf{ind}}$ is applicable on a given `SNode` N whenever there exists an instance *ir* of *IR* for which: (i) the consequent of *ir* matches N.*sequent* and the pivot of *ir* matches N′.*sequent*, for some `SNode` ancestor N′ of N, such that the side conditions of *ir* are satisfied, and (ii) if $\mathsf{R}_1, \ldots, \mathsf{R}_n$ is the `RNode` sequence extracted from

[1] Flex – The Fast Lexical Analyzer, `github.com/westes/flex`
[2] GNU Bison – The Yacc-compatible Parser Generator, `www.gnu.org/software/bison`

the path starting at N' and ending at N, then the sequence $\mathsf{R}_1.rule, \ldots, \mathsf{R}_n.rule$ satisfies the pivot condition of ir.

Algorithm 2 Sketch of our exhaustive proof search implementation

input: an SL inductive system $\mathcal{S}$, a basic sequent $p(\mathbf{x}) \vdash q_1(\mathbf{x}), \ldots, q_n(\mathbf{x})$,
and a proof strategy NFA $\mathbf{S} = (Q_\mathbf{S}, \mathcal{R}^{\mathsf{sl}}_{\mathsf{ind}}, T_\mathbf{S}, q_0, F_\mathbf{S})$
output: `VALID` and an $\mathbf{S}$-proof starting with $p(\mathbf{x}) \vdash q_1(\mathbf{x}), \ldots, q_n(\mathbf{x})$, built with $\mathcal{R}^{\mathsf{sl}}_{\mathsf{ind}}$;
`INVALID` and a counterexample for $p(\mathbf{x}) \vdash q_1(\mathbf{x}), \ldots, q_n(\mathbf{x})$;
`UNKNOWN` and the proof search tree constructed by the algorithm

1: $\mathsf{Root} \leftarrow \texttt{SNode}(p(\mathbf{x}) \vdash q_1(\mathbf{x}), \ldots, q_n(\mathbf{x}), [q_0], \mathit{null}, [])$
2: $\mathsf{Queue} \leftarrow \{\mathsf{Root}\}$
3: **while** Queue $\neq []$ and Root is *Unknown* **do**
4: dequeue N from Queue
5: let $Q^{IR}_\mathsf{N} = \{q' \mid (q, IR) \rightarrow q' \in T_\mathbf{S} \text{ and } q \in \mathsf{N}.\mathit{states}\}$ for any $IR \in \mathcal{R}^{\mathsf{sl}}_{\mathsf{ind}}$
6: let $\mathcal{R}_\mathsf{N} = \{IR \mid Q^{IR}_\mathsf{N} \neq \emptyset \text{ and } IR \text{ applicable on } \mathsf{N}\}$
7: **if** $\mathrm{AX} \in \mathcal{R}_\mathsf{N}$ and $Q^{\mathrm{AX}}_\mathsf{N} \cap F_\mathbf{S} \neq \emptyset$ **then**
8: $\mathsf{R} \leftarrow \texttt{RNode}(\mathrm{AX}, \mathit{null}, \mathsf{N}, [])$ and add R to N.*children*
9: $\mathsf{N}' \leftarrow \texttt{SNode}(\top, Q^{\mathrm{AX}}_\mathsf{N}, \mathsf{R}, [])$, add N' to R.*children* and mark it as *Closed*
10: **else if** $\mathrm{ID} \in \mathcal{R}_\mathsf{N}$ and $Q^{\mathrm{ID}}_\mathsf{N} \cap F_\mathbf{S} \neq \emptyset$ **then**
11: $\mathsf{R} \leftarrow \texttt{RNode}(\mathrm{ID}, \mathsf{N}', \mathsf{N}, [])$ for some pivot N' of ID and add R to N.*children*
12: $\mathsf{N}' \leftarrow \texttt{SNode}(\top, Q^{\mathrm{ID}}_\mathsf{N}, \mathsf{R}, [])$, add N' to R.*children* and mark it as *Closed*
13: **else**
14: **for** each instance ir of each $IR \in \mathcal{R}_\mathsf{N}$ **do**
15: $\mathsf{R} \leftarrow \texttt{RNode}(IR, \mathit{null}, \mathsf{N}, [])$ and add R to N.*children*
16: let k be the number of antecedents generated by ir
17: **if** $k = 0$ and $Q^{IR}_\mathsf{N} \cap F_\mathbf{S} \neq \emptyset$ **then**
18: $\mathsf{N}' \leftarrow \texttt{SNode}(\top, Q^{IR}_\mathsf{N}, \mathsf{R}, [])$, add N' to R.*children* and mark it as *Closed*
19: **for** each antecedent $\Gamma_i \vdash \Delta_i$ of ir with $i \in [k]$ **do**
20: $\mathsf{N}_i \leftarrow \texttt{SNode}(\Gamma_i \vdash \Delta_i, Q^{IR}_\mathsf{N}, \mathsf{R}, [])$ and add N_i to R.*children*
21: **if** $\Delta_i = \emptyset$ **then** mark N_i as *Failed*
22: **if** R is not *Failed* **then** enqueue $\mathsf{N}_1, \ldots, \mathsf{N}_k$ in Queue
23: **if** Root is *Closed* **then**
24: **return** `VALID` and `ExtractProof`(Root)
25: **else if** Root is *Failed* **then**
26: **return** `INVALID` and `ExtractCounterexamples`(Root)
27: **else return** `UNKNOWN` and Root

Both types of nodes are marked with either a *Closed*, *Failed* or *Unknown* status. All nodes are initially *Unknown*. The status of an `SNode` can be changed to *Closed* whenever: (i) its sequent is $\top$, or (ii) at least one of its `RNode` children is *Closed*. An `RNode` becomes *Closed* when all of its `SNode` children are *Closed*. Conversely, an `SNode` is marked as *Failed* whenever: (i) its sequent is of the form $\Gamma \vdash \emptyset$, or (ii) all of its `RNode` children are *Failed*. An `RNode` is marked *Failed* when at least one of its `SNode` children is *Failed*. Changing the status of a node prompts

a status update for all of its ancestors.

Algorithm 2 sketches our proof search implementation for $\mathcal{R}^{\mathsf{sl}}_{\mathsf{ind}}$, which, given an input sequent $p(\mathbf{x}) \vdash q_1(\mathbf{x}), \ldots, q_n(\mathbf{x})$ and an NFA $\mathbf{S} = (Q_\mathbf{S}, \mathcal{R}^{\mathsf{sl}}_{\mathsf{ind}}, T_\mathbf{S}, q_0, F_\mathbf{S})$ describing the proof strategy, explores all derivations rooted at the input sequent. The default strategy is $(\mathrm{LU} \cdot \mathrm{RU}^* \cdot \mathrm{RD} \cdot \mathrm{SP}?)^* \cdot \mathrm{LU}? \cdot \mathrm{RU}^* \cdot (\mathrm{AX} \mid \mathrm{ID})$ from [IS17, Theorem 7]. We construct a node Root and add it to the work queue. While the work queue is not empty and the status of Root is *Unknown*, we dequeue an `SNode` N. We denote by Q^{IR}_{N} the set of states in $\mathbf{S}$ towards which we transition from $\mathsf{N}.\mathit{states}$ by applying *IR*, and build a set $\mathcal{R}_{\mathsf{N}}$ of applicable inference rule schemata that are also accepted by the strategy.

If AX or ID are in $\mathcal{R}_{\mathsf{N}}$ and, moreover, their application leads $\mathbf{S}$ to transition to some final states, then this branch of the derivation has been successful. We add a $\top$ leaf, which is marked as *Closed*. Otherwise, for each $\mathit{IR} \in \mathcal{R}_{\mathsf{N}}$ we consider each instance *ir* of *IR* with antecedents $\Gamma_1 \vdash \Delta_1, \ldots, \Gamma_k \vdash \Delta_k$. If $k = 0$ and we reach some final state in $\mathbf{S}$ by transition with *IR*, then this branch is successful and we add a $\top$ leaf that we mark as *Closed*. Otherwise, if $k > 1$, we create an `RNode` R for *ir* and an `SNode` N_i for each of its antecedents. If $\Delta_i = \emptyset$ for some $i \in [k]$, then N_i is marked as *Failed*. If this is not the case for any $i \in [k]$, then we add $\mathsf{N}_1, \ldots, \mathsf{N}_k$ to the work queue and continue.

When the status of Root changes to *Closed*, then a proof has been obtained. The proof is extracted from the proof search tree and offered as a certificate. Otherwise, if it changes to *Failed*, then at least one counterexample has been discovered. We extract the counterexamples from the proof search tree and give them as witnesses. If the work queue becomes empty, but the status of Root is still *Unknown*, then the proof search was inconclusive and our entire proof search tree is returned as justification.

3.2 Case Study: Binary Trees

Consider the following ranked definitions for binary trees. The predicate *tree* accepts any tree model, tree^+_1 accepts trees with at least one node, and tree^+_2 accepts trees with at least one node in which the children of a node are either both allocated or both nil.

$$
\begin{aligned}
\mathit{tree}(x) &:=_{\mathcal{S}_t} x \approx \mathsf{nil} \wedge \mathsf{emp} \mid x \mapsto (l,r), \mathit{tree}(l), \mathit{tree}(r) \\
\mathit{tree}^+_1(x) &:=_{\mathcal{S}_t} x \mapsto (\mathsf{nil}, \mathsf{nil}) \mid x \mapsto (l,r), \mathit{tree}^+_1(l), \mathit{tree}(r) \mid x \mapsto (l,r), \mathit{tree}(l), \mathit{tree}^+_1(r) \\
\mathit{tree}^+_2(x) &:=_{\mathcal{S}_t} x \mapsto (\mathsf{nil}, \mathsf{nil}) \mid x \mapsto (l,r), \mathit{tree}^+_2(l), \mathit{tree}^+_2(r)
\end{aligned}
$$

The entailments $\mathit{tree}^+_1 \models^{\mathsf{sl}}_{\mathcal{S}_t} \mathit{tree}$, $\mathit{tree}^+_2 \models^{\mathsf{sl}}_{\mathcal{S}_t} \mathit{tree}$ and $\mathit{tree}^+_2 \models^{\mathsf{sl}}_{\mathcal{S}_t} \mathit{tree}^+_1$ hold, facts corroborated by Inductor. A branch of the proof for $\mathit{tree}^+_2(x) \vdash \mathit{tree}^+_1(x)$ is depicted below. However, the reversed entailments do not hold and the counterexamples provided are:

- $x \approx \mathsf{nil} \wedge \mathsf{emp}$ for $\mathit{tree}(x) \vdash \mathit{tree}^+_1(x)$ and $\mathit{tree}(x) \vdash \mathit{tree}^+_2(x)$;
- $x \mapsto (l_0, r_0) * \mathit{tree}^+_1(l_0) * (r_0 \approx \mathsf{nil} \wedge \mathsf{emp})$ for $\mathit{tree}^+_1(x) \vdash \mathit{tree}^+_2(x)$. Note that predicate atoms can occur within counterexamples and indicate that they can be substituted by any model to obtain a more concrete one. In this case, an immediate substitution with the base case of $\mathit{tree}^+_1(l_0)$ gives us $x \mapsto (l_0, r_0) * l_0 \mapsto (\mathsf{nil}, \mathsf{nil}) * (r_0 \approx \mathsf{nil} \wedge \mathsf{emp})$, which can be further simplified to $x \mapsto (l_0, \mathsf{nil}) * l_0 \mapsto (\mathsf{nil}, \mathsf{nil})$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\top}{tree_2^+(l_{00}) \vdash tree(l_{00})}\,\mathrm{ID}}{tree_2^+(l_{00}), tree_2^+(r_{00}) \vdash tree(l_{00}) * tree(r_{00})}\,\mathrm{SP}}{l_0 \mapsto (l_{00}, r_{00}), tree_2^+(l_{00}), tree_2^+(r_{00}) \vdash l_0 \mapsto (\mathsf{nil}, \mathsf{nil}), \exists l_{11} \exists r_{11} \,.\, l_0 \mapsto (l_{11}, r_{11}) * tree(l_{11}) * tree(r_{11})}\,\mathrm{RD}}{l_0 \mapsto (l_{00}, r_{00}), tree_2^+(l_{00}), tree_2^+(r_{00}) \vdash tree(l_0)}\,\mathrm{RU}}{tree_2^+(l_0) \vdash tree(l_0)}\,\mathrm{LU}}{tree_2^+(l_0), tree_2^+(r_0) \vdash tree_1^+(l_0) * tree(r_0), tree(l_0) * tree_1^+(r_0)}\,\mathrm{SP}}{x \mapsto (l_0, r_0), tree_2^+(l_0), tree_2^+(r_0) \vdash x \mapsto (\mathsf{nil}, \mathsf{nil}), \exists l_1 \exists r_1 \,.\, x \mapsto (l_1, r_1) * tree_1^+(l_1) * tree(r_1), \exists l_1 \exists r_1 \,.\, x \mapsto (l_1, r_1) * tree(l_1) * tree_1^+(r_1)}\,\mathrm{RD}}{x \mapsto (l_0, r_0), tree_2^+(l_0), tree_2^+(r_0) \vdash tree_1^+(x)}\,\mathrm{RU}}{tree_2^+(x) \vdash tree_1^+(x)}\,\mathrm{LU}$$

3.3 Case Study: Possibly Cyclic and Acyclic List Segments

Consider the following ranked definitions for possibly cyclic and acyclic list segments.

$$ls(x,y) :=_{\mathcal{S}_l} x \approx y \wedge \mathsf{emp} \mid x \mapsto z, ls(z,y) \qquad ls^a(x,y) :=_{\mathcal{S}_l} x \approx y \wedge \mathsf{emp} \mid \neg(x \approx y) \wedge x \mapsto z, ls^a(z,y)$$

Naturally, the entailment $ls^a \models^{\mathsf{sl}}_{\mathcal{S}_l} ls$ holds, while $ls \models^{\mathsf{sl}}_{\mathcal{S}_l} ls^a$ does not. The proof for the former case is shown below. In the latter case, the counterexample provided by Inductor for $ls(x,y) \vdash ls^a(x,y)$ is $x \approx y \wedge x \mapsto z_0 * ls(z_0,y)$, from which we can obtain the more concrete one $x \approx y \wedge x \mapsto z_0 * (z_0 \approx y \wedge \mathsf{emp})$, further simplified to $x \mapsto x$.

$$\dfrac{\dfrac{\dfrac{\top}{x \approx y \wedge \mathsf{emp} \vdash x \approx y \wedge \mathsf{emp}, ls(x,y)}\,\mathrm{AX}}{x \approx y \wedge \mathsf{emp} \vdash ls(x,y)}\,\mathrm{RU} \qquad \dfrac{\dfrac{\dfrac{\top}{ls^a(z_0,y) \vdash ls(z_1,y)}\,\mathrm{ID}}{\neg(x \approx y) \wedge x \mapsto z_0, ls^a(z_0,y) \vdash x \approx y \wedge \mathsf{emp}, \exists z_1 . x \mapsto z_1 * ls(z_1,y)}\,\mathrm{RD}}{\neg(x \approx y) \wedge x \mapsto z_0, ls^a(z_0,y) \vdash ls(x,y)}\,\mathrm{RU}}{ls^a(x,y) \vdash ls(x,y)}\,\mathrm{LU}$$

3.4 Case Study: List Segments of Even and Odd Length

Consider the following ranked definitions for list segments of even and odd length, together with two alternate definitions of list segments with at least one element.

$$ls^e(x,y) \leftarrow_{\mathcal{S}_{eo}} x \approx y \wedge \mathsf{emp} \mid x \mapsto z, ls^o(z,y) \qquad ls^+(x,y) \leftarrow_{\mathcal{S}_{eo}} x \mapsto y \mid x \mapsto z, ls^+(z,y)$$
$$ls^o(x,y) \leftarrow_{\mathcal{S}_{eo}} x \mapsto y \mid x \mapsto z, ls^e(z,y) \qquad \widehat{ls}^+(x,y) \leftarrow_{\mathcal{S}_{eo}} x \mapsto z, ls^e(z,y) \mid x \mapsto z, ls^o(z,y)$$

The entailments $ls^o \models^{\mathsf{sl}}_{\mathcal{S}_{eo}} \widehat{ls}^+$, $ls^+ \models^{\mathsf{sl}}_{\mathcal{S}_{eo}} \widehat{ls}^+$, $ls^+ \models^{\mathsf{sl}}_{\mathcal{S}_{eo}} ls^e, ls^o$ and $\widehat{ls}^+ \models^{\mathsf{sl}}_{\mathcal{S}_{eo}} ls^e, ls^o$ hold, while entailments such as $ls^e \models^{\mathsf{sl}}_{\mathcal{S}_{eo}} \widehat{ls}^+$, $ls^e \models^{\mathsf{sl}}_{\mathcal{S}_{eo}} ls^o$, $ls^+ \models^{\mathsf{sl}}_{\mathcal{S}_{eo}} ls^e$ or $\widehat{ls}^+ \models^{\mathsf{sl}}_{\mathcal{S}_{eo}} ls^o$ do not. A branch of the proof for $ls^+(x,y) \vdash \widehat{ls}^+(x,y)$ is shown below. For the invalid entailments, Inductor gives the counterexamples: $x \approx y \wedge \mathsf{emp}$ for both $ls^e(x,y) \vdash \widehat{ls}^+(x,y)$ and $ls^e(x,y) \vdash ls^o(x,y)$; $x \mapsto y$ for $ls^+(x,y) \vdash ls^e(x,y)$; $x \mapsto z_0 * z_0 \mapsto y$ for $\widehat{ls}^+(x,y) \vdash ls^o(x,y)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\top}{ls^+(z_{00},y)\vdash ls^o(z_{00},y),ls^e(z_{00},y)}\,\mathrm{ID}}{z_0\mapsto z_{00},ls^+(z_{00},y)\vdash z_0\approx y\wedge\mathsf{emp},\exists z_{01}\,.\,z_0\mapsto z_{01}*ls^o(z_{01},y),z_0\mapsto y,\exists z_{11}\,.\,z_0\mapsto z_{11}*ls^e(z_{11},y)}\,\mathrm{RD}}{z_0\mapsto z_{00},ls^+(z_{00},y)\vdash z_0\approx y\wedge\mathsf{emp},\exists z_{01}\,.\,z_0\mapsto z_{01}*ls^o(z_{01},y),ls^o(z_0,y)}\,\mathrm{RU}}{z_0\mapsto z_{00},ls^+(z_{00},y)\vdash ls^e(z_0,y),ls^o(z_0,y)}\,\mathrm{RU}}{ls^+(z_0,y)\vdash ls^e(z_0,y),ls^o(z_0,y)}\,\mathrm{LU}}{x\mapsto z_0,ls^+(z_0,y)\vdash\exists z_1\,.\,x\mapsto z_1*ls^e(z_1,y),\exists z_1\,.\,x\mapsto z_1*ls^o(z_1,y)}\,\mathrm{RD}}{x\mapsto z_0,ls^+(z_0,y)\vdash\widehat{ls}^+(x,y)}\,\mathrm{RU}}{ls^+(x,y)\vdash\widehat{ls}^+(x,y)}\,\mathrm{LU}$$

3.5 Case Study: Lists of Possibly Cyclic and Acyclic List Segments

We adapt our definitions from §3.3 to a fragment in which each memory location points to a pair of locations, and use them to define lists whose elements point at possibly cyclic or acyclic list segments. The last elements of these list segments are, in turn, linked backwards and the last element of the primary list points to the last element of the last secondary list segment.

$$\begin{aligned}
ls(x,y) &:=_{\mathcal{S}_{ll}} x\approx y\wedge\mathsf{emp}\ \mid x\mapsto(z,\mathsf{nil}),ls(z,y)\\
ls^a(x,y) &:=_{\mathcal{S}_{ll}} x\approx y\wedge\mathsf{emp}\ \mid\neg(x\approx y)\wedge x\mapsto(z,\mathsf{nil}),ls^a(z,y)\\
lls(x,v) &:=_{\mathcal{S}_{ll}} x\approx v\wedge\mathsf{emp}\mid x\mapsto(z,u)*w\mapsto(v,\mathsf{nil}),ls(u,v),lls(z,w)\\
lls^a(x,v) &:=_{\mathcal{S}_{ll}} x\approx v\wedge\mathsf{emp}\mid x\mapsto(z,u)*w\mapsto(v,\mathsf{nil}),ls^a(u,v),lls^a(z,w)
\end{aligned}$$

The entailment $lls^a \models^{\mathsf{sl}}_{\mathcal{S}_{ll}} lls$ holds, while its reverse $lls \models^{\mathsf{sl}}_{\mathcal{S}_{ll}} lls^a$ does not. Part of the proof for $lls^a(x,v)\vdash lls(x,v)$ is shown below. The subproof for $ls^a(u_0,v)\vdash ls(u_0,v)$ is mostly skipped as it is identical to the one from §3.3 modulo a variable renaming.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\top}{ls^a(u_{00},v)\vdash ls(u_{00},v)}\,\mathrm{ID}}{\vdots}\,\mathrm{LU}}{ls^a(u_0,v)\vdash ls(u_0,v)}\quad\dfrac{\dfrac{\top}{lls^a(z_0,w_0)\vdash lls(u_0,w_0)}\,\mathrm{ID}}{}}{ls^a(u_0,v),lls^a(z_0,w_0)\vdash ls(u_0,v)*lls(z_0,w_0)}\,\mathrm{SP}}{x\mapsto(z_0,u_0)*w_0\mapsto(v,\mathsf{nil}),ls^a(u_0,v),lls^a(z_0,w_0)\vdash x\approx\mathsf{nil}\wedge\mathsf{emp},\ \exists z_1\exists u_1\exists w_1\,.\,x\mapsto(z_1,u_1)*w_1\mapsto(v,\mathsf{nil})*ls(u_1,v)*lls(z_1,w_1)}\,\mathrm{RD}}{x\mapsto(z_0,u_0)*w_0\mapsto(v,\mathsf{nil}),ls^a(u_0,v),lls^a(z_0,w_0)\vdash lls(x,v)}\,\mathrm{RU}}{lls^a(x,v)\vdash lls(x,v)}\,\mathrm{LU}$$

The counterexample provided by Inductor for $lls(x,v)\vdash lls^a(x,v)$ is $x\mapsto(z_0,u_0)*w_0\mapsto(v,\mathsf{nil})*(u_0\approx v\wedge u_0\mapsto(u_{00},\mathsf{nil})*ls(u_{00},v))*lls(z_0,w_0)$, from which we obtain the more concrete one $x\mapsto(z_0,u_0)*w_0\mapsto(v,\mathsf{nil})*(u_0\approx v\wedge u_0\mapsto(u_{00},\mathsf{nil})*(u_{00}\approx v\wedge\mathsf{emp}))*(z_0\approx w_0\wedge\mathsf{emp})$, further simplified to $x\mapsto(z_0,v)*z_0\mapsto(v,\mathsf{nil})*v\mapsto(v,\mathsf{nil})$.

3.6 Experimental results

Table 1 summarizes the experimental results obtained for the entailments discussed in §3.2-3.5. All experiments were run on a 2.10GHz Intel® Core™ i7-4600U CPU machine with 4MB of cache. For each case, we indicate: (i) the result (column **R**), which can be `V` for `VALID` or `I` for `INVALID`, (ii) the total number of sequent nodes (column **Seq**), the maximum number of sequent nodes along a branch (column **H**) and the maximum number of (LU) and (SP) applications along a branch (columns $\mathbf{H}_{\mathrm{LU}}$ and $\mathbf{H}_{\mathrm{SP}}$) of the tree structure defined in Figure 2, which encodes the derivation, (iii) the run time for the proof search algorithm (column **T**), and (iv) the total number of calls to CVC4 (column **CVC4**).

LHS	RHS	R	Seq	$\mathbf{H}_{\mathrm{Seq}}$	$\mathbf{H}_{\mathrm{LU}}$	$\mathbf{H}_{\mathrm{SP}}$	T	CVC4	LHS	RHS	R	Seq	$\mathbf{H}_{\mathrm{Seq}}$	$\mathbf{H}_{\mathrm{LU}}$	$\mathbf{H}_{\mathrm{SP}}$	T	CVC4
$tree_1^+$	$tree$	V	34	7	2	1	0.096s	9	$tree$	$tree_1^+$	I	7	4	1	0	0.033s	7
$tree_2^+$	$tree$	V	21	7	2	1	0.053s	7	$tree$	$tree_2^+$	I	7	4	1	0	0.028s	5
$tree_2^+$	$tree_1^+$	V	1477	11	3	2	5.515s	37	$tree_1^+$	$tree_2^+$	I	38	8	2	1	0.096s	14
ls^a	ls	V	8	5	1	0	0.014s	2	ls	ls^a	I	7	4	1	0	0.015s	2
lls^a	lls	V	21	9	2	1	0.048s	4	lls	lls^a	I	20	8	2	1	0.043s	4
ls^o	$\widehat{ls}^+$	V	10	6	1	0	0.032s	5	ls^e	$\widehat{ls}^+$	I	7	4	1	0	0.024s	4
ls^+	$\widehat{ls}^+$	V	16	8	2	0	0.049s	9	ls^e	ls^o	I	7	4	1	0	0.030s	5
ls^+	ls^e, ls^o	V	8	5	1	0	0.020s	4	ls^+	ls^e	I	13	6	2	0	0.075s	8
$\widehat{ls}^+$	ls^e, ls^o	V	9	5	1	0	0.028s	8	$\widehat{ls}^+$	ls^o	I	20	9	3	0	0.143s	12

Table 1: Experimental results

As shown by the **T** column in both halves of the table, the execution times are fairly low. The size of the derivations is influenced by how elaborate the inductive definitions are. For instance, $tree_1^+$ is defined by three predicate rules, thus when encountered on the left-hand side of an entailment will generate a larger number of nodes due to left-unfolding. On the right-hand side, the number of predicate rules in a definition and the number of subgoals in each predicate rule both influence the complexity of (SP), which can lead to higher execution times than expected, given the size of the derivation, since all instances of (SP) need to be generated and then checked.

4 Conclusions

We describe an entailment checker tool called Inductor, which implements a cyclic proof system for Separation Logic with inductive definitions and utilizes dedicated decision procedures in the SMT solver CVC4 to establish the satisfiability of quantifier-free or $\exists^*\forall^*$-quantified Separation Logic formulae. The tool outputs a proof whenever an entailment is found to be valid, or counterexamples when it is not. Soundness is warranted by imposing a ranking restriction on the inductive system given as input. It is possible, although the results may be inconclusive, to use Inductor outside of these boundaries. We discuss several case studies and provide experimental results showing fairly low execution times and moderate sizes for the derivations built by the tool in order to obtain proofs or counterexamples.

Our inference rules build cyclic proofs with backlinks in similar fashion as CYCLIST [BGP12],

closing recurring branches of a proof with (ID). We restrict backlinks to ancestral nodes, which allows us to embed the condition necessary to ensure progress along an infinite trace directly into (ID). Because we allow disjunctions on the right-hand side of sequents and (RU) introduces all the cases of an inductive definition – as opposed to CYCLIST and [CJT15], which always choose only one – Inductor can tackle entailments such as $ls^+(x,y) \models^{\mathsf{sl}}_{\mathcal{S}_{eo}} \widehat{ls}^+(x,y)$ in §3.4, which CYCLIST and [CJT15] cannot prove. However, the cut rule in [CJT15] and the canonical rotation relation between trees in SLIDE [IRV14] enable these systems to show entailments such as the ones in Example 7, for which Inductor cannot build proofs. On a different note, the fragment of inductive definitions that SLIDE can translate to tree automata does not allow disequalities between variables, thus it cannot handle predicates and entailments such as $ls^a(x,y) \models^{\mathsf{sl}}_{\mathcal{S}_l} ls(x,y)$ in §3.3.

SLEEK [CDNQ12] and SPEN [ELS17, ESW15] go further than Inductor and are able to check much more complex verification conditions, involving, for instance, concatenations of predicates, formulae equivalent to several unfoldings of a predicate and various combinations of allocated heap cells and predicate calls. Although most of the inductive definitions for data structures used in these entailments fall into the fragment accepted by Inductor, the entailments themselves are out of the scope of our current implementation. Multiple extensions of the inference rules are possible in order to allow the building of proofs for such entailments (e.g. right unfolding inside the same formula multiple times, reducing any subset of constraints) and are considered for future work.

Bibliography

[BCD+11] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, C. Tinelli. CVC4. In Gopalakrishnan and Qadeer (eds.), *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*. Pp. 171–177. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[BFT17] C. Barrett, P. Fontaine, C. Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017. Available at `www.SMT-LIB.org`.

[BGP12] J. Brotherston, N. Gorogiannis, R. L. Petersen. A Generic Cyclic Theorem Prover. In *Programming Languages and Systems: 10th Asian Symposium (APLAS'12)*. Pp. 350–367. Springer, 2012.

[Bus18] W. H. Bussey. Fermat's Method of Infinite Descent. *The American Mathematical Monthly* 25(8):333–337, 1918.

[CDG+05] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, M. Tommasi. Tree Automata Techniques and Applications. 2005.
URL: `http://www.grappa.univ-lille3.fr/tata`.

[CDNQ12] W.-N. Chin, C. David, H. H. Nguyen, S. Qin. Automated Verification of Shape, Size and Bag Properties via User-defined Predicates in Separation Logic. *Sci. Comput. Program.* 77(9):1006–1036, Aug. 2012.
doi:10.1016/j.scico.2010.07.004
http://dx.doi.org/10.1016/j.scico.2010.07.004

[CJT15] D. Chu, J. Jaffar, M. Trinh. Automatic Induction Proofs of Data-Structures in Imperative Programs. In *Proc. of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*. Pp. 457–466. ACM, New York, NY, USA, 2015.

[EIP18] M. Echenim, R. Iosif, N. Peltier. On the Expressive Completeness of Bernays-Schönfinkel-Ramsey Separation Logic. *ArXiv e-prints*, feb 2018.
https://arxiv.org/abs/1802.00195v2

[ELS17] C. Enea, O. Lengál, M. Sighireanu, T. V. . SPEN: A Solver for Separation Logic. In Barrett et al. (eds.), *NASA Formal Methods*. Pp. 302–309. Springer International Publishing, Cham, 2017.

[ESW15] C. Enea, M. Sighireanu, Z. Wu. On Automated Lemma Generation for Separation Logic with Inductive Definitions. In *Automated Technology for Verification and Analysis: 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proc.* Pp. 80–96. Springer International Publishing, Cham, Switzerland, 2015.

[HLSV11] L. Holík, O. Lengál, J. Simácek, T. Vojnar. Efficient Inclusion Checking on Explicit and Semi-symbolic Tree Automata. In *ATVA 2011, Proc.* Pp. 243–258. 2011.

[IRV14] R. Iosif, A. Rogalewicz, T. Vojnar. Deciding Entailments in Inductive Separation Logic with Tree Automata. In *Automated Technology for Verification and Analysis: 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proc.* Pp. 201–218. Springer International Publishing, Cham, Switzerland, 2014.

[IS17] R. Iosif, C. Serban. Complete Cyclic Proof Systems for Inductive Entailments. *CoRR* abs/1707.02415, 2017.
http://arxiv.org/abs/1707.02415

[LSV+] O. Lengál, J. Simácek, T. Vojnar, M. Hruska, L. Holík. libVATA - A C++ library for efficient manipulation with non-deterministic finite (tree) automata.
URL: `https://github.com/ondrik/libvata`.

[Rey02] J. C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proc. of LICS*. Pp. 55–74. 2002.

[RIS17] A. Reynolds, R. Iosif, C. Serban. Reasoning in the Bernays-Schönfinkel-Ramsey Fragment of Separation Logic. In *Verification, Model Checking, and Abstract Interpretation: 18th International Conference, VMCAI 2017, Paris, France, January*

15–17, 2017, Proc. Pp. 462–482. Springer International Publishing, Cham, Switzerland, 2017.

[RISK16] A. Reynolds, R. Iosif, C. Serban, T. King. A Decision Procedure for Separation Logic in SMT. In Artho et al. (eds.), *Automated Technology for Verification and Analysis: 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings*. Pp. 244–261. Springer International Publishing, 2016.

[Ser17] C. Serban. Inductor: an entailment checker for inductive systems. URL: `https://github.com/cristina-serban/inductor`, 2017.

[SOR+13] A. Stump, D. Oe, A. Reynolds, L. Hadarean, C. Tinelli. SMT Proof Checking Using a Logical Framework. *Formal Methods in System Design* 42(1):91–118, 2013. doi:10.1007/s10703-012-0163-3

A Additional Material

A.1 Specifying Proof Strategies as Automata

Inductor can also accept a proof strategy as input – if no proof strategy is given, then $\mathbf{S} = (\mathrm{LU} \cdot \mathrm{RU}^* \cdot \mathrm{RD} \cdot \mathrm{SP}?)^* \cdot \mathrm{LU}? \cdot \mathrm{RU}^* \cdot (\mathrm{AX} \mid \mathrm{ID})$ from [IS17, Theorem 7] will be used as default. By Kleene's Theorem, it is known that, given a regular expression, there exists an equivalent nondeterministic finite word automaton (NFA), possibly with ε-transitions (NFA-ε). Figure 3 depicts a straightforward NFA-ε that is equivalent to **S**.

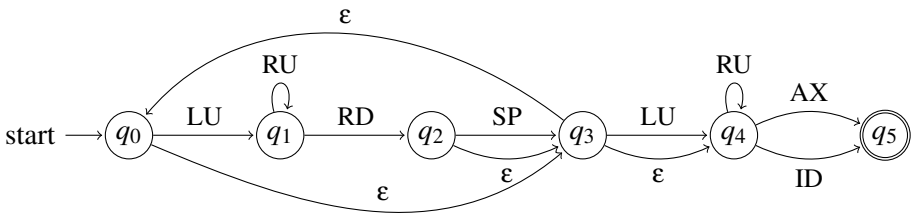


Figure 3: An NFA-ε equivalent to our default proof strategy **S**

We are more interested in such a representation because, after applying a certain inference rule, we want to easily check which inference rules that comply with the strategy could be applied next. However, given an NFA-ε, the ε-transitions are cumbersome and, thus, we prefer an equivalent NFA – which is guaranteed to exist, since the two classes of automata are known to be equivalent. As such, the proof strategies that Inductor accepts as input are given as NFA, rather than regular expressions. The definition of such an NFA is specified in a language inspired by the simplicity of the one used by libVATA [LSV+] and whose grammar is depicted in Listing 1.

```
<file>        :  'Rules' <rule_list> <automaton>
<rule_list>   :  <rule> <rule> ...
<automaton>   :  'Automaton' string 'States' <state_list>
'Initial State' <state> 'Final States' <state_list>
'Transitions' <trans_list>
<state_list>  :  <state> <state> ...
<state>       :  string
<trans_list>  :  <trans> <trans> ...
<trans>       :  '(' <state> ',' <rule> ')' '->' <state>
<rule>        :  string
```

Listing 1: Grammar for files specifying proof strategies as NFA

Using this language, Listing 2 defines an NFA that is equivalent with the NFA-ε from Figure 3, and consequently, is also equivalent with our default proof search strategy.

```
Rules LU RU RD SP ID AX
Automaton Default    States q0 q1 q2 q3 q4 q5
Initial state q0     Final states q5
Transitions
(q0, LU) -> q1   (q1, RD) -> q3   (q2, RI) -> q3   (q3, LU) -> q4
(q0, LU) -> q4   (q1, RD) -> q4   (q2, RI) -> q4   (q3, RU) -> q4
```

```
(q0, RU) -> q4   (q2, LU) -> q1   (q2, SP) -> q0   (q3, AX) -> q5
(q0, AX) -> q5   (q2, LU) -> q4   (q2, SP) -> q3   (q3, ID) -> q5
(q0, ID) -> q5   (q2, RU) -> q4   (q2, SP) -> q4   (q4, RU) -> q4
(q1, RU) -> q1   (q2, RI) -> q0   (q2, AX) -> q5   (q4, AX) -> q5
(q1, RD) -> q0   (q2, RI) -> q2   (q2, ID) -> q5   (q4, ID) -> q5
(q1, RD) -> q2
```

Listing 2: The definition of the NFA corresponding to the default proof search strategy