Compositional Entailment Checking for a Fragment of Separation Logic

FIT BUT Technical Report Series

Constantin Enea, Ondřej Lengál, Mihaela Sighireanu, and Tomáš Vojnar

Technical Report No. FIT-TR-2014-01
Faculty of Information Technology, Brno University of Technology

Last modified: April 4, 2014

Compositional Entailment Checking for a Fragment of Separation Logic

Constantin Enea[1], Ondřej Lengál[2], Mihaela Sighireanu[1], and Tomáš Vojnar[2]

[1] Univ. Paris Diderot, LIAFA CNRS UMR 7089
[2] FIT, Brno University of Technology, IT4Innovations Centre of Excellence, Czech Republic

Abstract. We present a (semi-)decision procedure for checking entailment between separation logic formulas with inductive predicates specifying complex data structures corresponding to finite nesting of various kinds of linked lists: acyclic or cyclic, singly or doubly linked, skip lists, etc. The decision procedure is compositional in the sense that it reduces the problem of checking entailment between two arbitrary formulas to the problem of checking entailment between a formula and an atom. Subsequently, in case the atom is a predicate, we reduce the entailment to testing membership of a tree derived from the formula in the language of a tree automaton derived from the predicate. We implemented this decision procedure and tested it successfully on verification conditions obtained from programs using singly and doubly linked nested lists as well as skip lists.

1 Introduction

Automatic verification of programs manipulating dynamic linked data structures is highly challenging since it requires one to reason about complex program configurations having the form of graphs of an unbounded size. In general, one needs highly expressive formalisms capable of expressing the specification of the desired program and the effect of program statements. Moreover, in order to scale to large programs, the use of such a formalism within program analysis should be highly efficient. In this context, *separation logic* (SL) [11,16] has emerged as one of the most promising formalisms, offering both high expressiveness and scalability. The latter is due to its support of *compositional reasoning* based on the separating conjunction $*$ and the frame rule, which states that if a Hoare triple $\{\phi\}P\{\psi\}$ holds and P does not alter free variables in σ, then $\{\phi * \sigma\}P\{\psi * \sigma\}$ holds too. Therefore, when reasoning about P, one has to manipulate only specifications for the heap region altered by P.

Usually, SL is used together with higher-order *inductive definitions* that describe the data structures manipulated by the program. If we consider general inductive definitions, then SL is undecidable [3]. Various decidable fragments of SL have been introduced in the literature [1,10,14] by restricting the syntax of the inductive definitions and the boolean structure of the formulas.

In this work, we focus on a fragment of SL with inductive definitions that allows one to specify program configurations (heaps) containing finite nestings of various kinds of linked lists (acyclic or cyclic, singly or doubly linked, skip lists, etc.), which are very

frequent in practice. This fragment contains formulas of the form $\exists \overrightarrow{X} . \Pi \wedge \Sigma$ where X is a set of variables, Π is a conjunction of (dis)equalities, and Σ is a set of *spatial atoms* connected by the separating conjunction. Spatial atoms can be *points-to atoms*, which describe values of pointer fields of a given heap location, or *inductively defined predicates*, which describe data structures of an unbounded size. We propose a novel (semi-)decision procedure for checking the validity of an entailment of the form $\varphi \Rightarrow \psi$ where φ may contain existential quantifiers and ψ is a quantifier-free formula. Such a decision procedure can be used in Hoare-style reasoning to check inductive invariants but also in program analysis frameworks to decide termination of fixpoint computations. As usual, checking entailments of the form $\bigvee_i \varphi_i \Rightarrow \bigvee_j \psi_j$ can be soundly reduced to checking that for each i there exists j such that $\varphi_i \Rightarrow \psi_j$.

The insight of our decision procedure is an idea to use the semantics of the separating conjunction in order to reduce the problem of checking $\varphi \Rightarrow \psi$ to the problem of checking a set of simpler entailments where the right-hand side is an inductively-defined predicate $P(\ldots)$. This reduction shows that the compositionality principle holds not only for deciding the validity of a Hoare triple but also for deciding the validity of an entailment between two formulas.

Further, to check entailments $\varphi \Rightarrow P(\ldots)$ resulting from the above reduction, we define a semi-decision procedure (safely approximating the entailment) based on the membership problem for tree automata (TA). In particular, we reduce the entailment to testing membership of a tree derived from φ in the language of a TA $\mathcal{A}[P]$ derived from $P(\ldots)$. The tree encoding preserves some edges of the graph, called *backbone edges*, while others are re-directed to new nodes, related to the original destination by special symbols. Roughly, such a symbol may be a variable represented by the original destination, or it may show how to reach the original destination using backbone edges only.

To infer (dis)equalities implied by spatial atoms, the reduction to checking simpler entailments is based on boolean unsatisfiability checking, which is in co-NP but can usually be checked efficiently by current SAT solvers. The rest of the procedure is polynomial as the size of the TA $\mathcal{A}[P]$ is polynomial in the size of P, and the number of generated simpler entailment queries is also polynomial. Moreover, the approach can be easily extended into a full decision procedure, but then it becomes exponential for some of the considered structures (more complex than singly and doubly linked lists).

We implemented our decision procedure and tested it successfully on verification conditions obtained from programs using singly and doubly linked nested lists as well as skip lists. The results show that our procedure does not only have a theoretically favourable complexity (for the given context), but it also behaves nicely in practice (at the same time offering the additional benefit of compositionality that can be exploited within larger verification frameworks that can cache the simpler entailment queries).

Related Work. Several decision procedures for fragments of SL have been introduced in the literature [1,3,4,6,10,9,13,14,15].

Some of these works [1,3,4,13] consider a fragment of SL that uses only one predicate describing singly linked lists, which is a much more restricted setting than what is considered in this paper. In particular, Cook et al. [4] prove that the satisfiability/entailment problem can be solved in polynomial time. Piskac et al. [14] show that the boolean closure of this fragment can be translated to a decidable fragment of first-

order logic, and this way, they prove that the satisfiability/entailment problem can be decided in NP/co-NP. Furthermore, they consider the problem of combining SL formulas with constraints on data using the Nelson-Oppen theory combination framework. Adding constraints on data to SL formulas is considered also in Qiu *et al* [15].

Compared with our previous work [6], we consider a larger fragment of SL in this work that includes inductively-defined predicates for describing nestings of cyclic lists and doubly linked lists (DLLs).

Iosif et al. [10] introduce a decidable fragment of SL that can describe data structures even more complex than those considered here, including, e.g., trees with parent pointers or trees with linked leaves. However, [10] reduces the entailment problem to MSO on graphs with a bounded tree width, resulting in a multiply-exponential complexity. The recent work [9] considers a more restricted fragment (incomparable with ours) and proposes a more practical, purely TA-based decision procedure, which reduces the entailment problem to *language inclusion* on TA, establishing EXPTIME-completeness of the considered fragment. Our decision procedure deals with the boolean structure of the formulas using SAT solvers, thus reducing the entailment problem to the problem of checking the entailment between a formula and an atom. Such simpler entailments are then checked using a polynomial semi-decision procedure based on the *membership problem* for TA. The approach of [9] can deal with various forms of trees as well as with entailment of structures with skeletons based on different selectors (e.g., DLLs viewed from the beginning and DLLs viewed from the end). On the other hand, it currently cannot deal with structures of zero length and with some forms of structure concatenation (such as concatenation of two DLL segments), which we can handle.

2 Separation Logic Fragment

Let $Vars$ be a set of *program variables*, ranged over using x, y, z, and $LVars$ a set of *logical variables*, disjoint from $Vars$, ranged over using X, Y, Z. We assume that $Vars$ contains a variable `NULL`. Also, let $\mathbb{F}$ be a set of *fields*. We consider a fragment of Separation Logic whose syntax is given by the following grammar:

$$x, y \in Vars \text{ program variables} \quad X, Y \in LVars \text{ logical variables} \quad E, F ::= x \mid X$$
$$f \in \mathbb{F} \text{ fields} \qquad \rho ::= (f, E) \mid \rho, \rho \qquad P \in \mathbb{P} \text{ predicates}$$
$$\overrightarrow{B} \in (Vars \cup LVars)^* \text{ vectors of variables}$$
$$\Pi ::= E = F \mid E \neq F \mid \Pi \wedge \Pi \qquad \text{pure formulas}$$
$$\Sigma ::= emp \mid E \mapsto \{\rho\} \mid P(E, F, \overrightarrow{B}) \mid \Sigma * \Sigma \qquad \text{spatial formulas}$$
$$\varphi \triangleq \exists \overrightarrow{X}. \Pi \wedge \Sigma \qquad \text{formulas}$$

The set of program variables used in a formula φ is denoted by $pv(\varphi)$. By $\varphi(\overrightarrow{E})$, we denote a formula where the set of free variables is $\overrightarrow{E}$. Given a formula φ, $pure(\varphi)$ denotes its pure part Π. We allow set operations to be applied over vectors. Moreover, $E \neq \overrightarrow{B}$ is a shorthand for $\bigwedge_{F \in \overrightarrow{B}} E \neq F$.

The *points-to atom* $E \mapsto \{(f_i, F_i)\}_{i \in \mathcal{I}}$ specifies that the heap contains a location E whose f_i field points to F_i, for all i. W.l.o.g. we assume that each field f_i appears at

singly linked lists:

$$\mathtt{ls}(E,F) \triangleq lemp(E,F) \vee (E \neq F \wedge \exists X_{\mathtt{tl}}.\, E \mapsto \{(f, X_{\mathtt{tl}})\} * \mathtt{ls}(X_{\mathtt{tl}}, F))$$

lists of acyclic lists:

$$\mathtt{nll}(E,F,B) \triangleq lemp(E,F) \vee (E \neq \{F,B\} \wedge \exists X_{\mathtt{tl}}, Z.\, E \mapsto \{(s, X_{\mathtt{tl}}), (h, Z)\} * \mathtt{ls}(Z,B) * \mathtt{nll}(X_{\mathtt{tl}}, F, B))$$

lists of cyclic lists:

$$\mathtt{nlcl}(E,F) \triangleq lemp(E,F) \vee (E \neq F \wedge \exists X_{\mathtt{tl}}, Z.\, E \mapsto \{(s, X_{\mathtt{tl}}), (h, Z)\} * \circlearrowleft^{1+} \mathtt{ls}(Z) * \mathtt{nlcl}(X_{\mathtt{tl}}, F))$$

skip lists with three levels:

$$\mathtt{skl}_3(E,F) \triangleq lemp(E,F) \vee (E \neq F \wedge \exists X_{\mathtt{tl}}, Z_1, Z_2.\, E \mapsto \{(f_3, X_{\mathtt{tl}}), (f_2, Z_2), (f_1, Z_1)\} * \mathtt{skl}_1(Z_1, Z_2) * \mathtt{skl}_2(Z_2, X_{\mathtt{tl}}) * \mathtt{skl}_3(X_{\mathtt{tl}}, F))$$

$$\mathtt{skl}_2(E,F) \triangleq lemp(E,F) \vee (E \neq F \wedge \exists X_{\mathtt{tl}}, Z_1.\, E \mapsto \{(f_3, \mathtt{NULL}), (f_2, X_{\mathtt{tl}}), (f_1, Z_1)\} * \mathtt{skl}_1(Z_1, X_{\mathtt{tl}}) * \mathtt{skl}_2(X_{\mathtt{tl}}, F))$$

$$\mathtt{skl}_1(E,F) \triangleq lemp(E,F) \vee (E \neq F \wedge \exists X_{\mathtt{tl}}.\, E \mapsto \{(f_3, \mathtt{NULL}), (f_2, \mathtt{NULL}), (f_1, X_{\mathtt{tl}})\} * \mathtt{skl}_1(X_{\mathtt{tl}}, F))$$

Fig. 1. Examples of inductive definitions ($lemp(E,F) \triangleq E = F \wedge emp$).

most once in a set of pairs ρ. The fragment is parameterized by a set $\mathbb{P}$ of *inductively defined predicates*; intuitively, $P(E, F, \overrightarrow{B})$ describes a possibly empty *nested list segment* delimited by its arguments, i.e., all the locations it represents are reachable from E and co-reachable either from one of its arguments or from a non-empty loop.

Inductively defined predicates. We consider predicates defined as

$$P(E,F,\overrightarrow{B}) \triangleq (E = F \wedge emp) \vee \left(E \neq \{F\} \cup \overrightarrow{B} \ \wedge \exists X_{\mathtt{tl}}.\, \Sigma(E, X_{\mathtt{tl}}, \overrightarrow{B}) \ * \ P(X_{\mathtt{tl}}, F, \overrightarrow{B})\right)$$

where Σ is an existentially quantified formula, called *the matrix of* P, of the form:

$$\begin{array}{l} \Sigma(E, X_{\mathtt{tl}}, \overrightarrow{B}) \triangleq \exists \overrightarrow{Z}.\, E \mapsto \rho[X_{\mathtt{tl}}, \overrightarrow{Z}, \overrightarrow{B}] * \Sigma' \\ \qquad \Sigma' ::= Q(Z, U, \overrightarrow{Y}) \mid \circlearrowleft^{1+} Q[Z, \overrightarrow{Y}] \mid \Sigma' * \Sigma' \mid emp \\ \qquad\qquad \text{for } Z \in \overrightarrow{Z}, U \in \overrightarrow{Z} \cup \overrightarrow{B} \cup \{E, X_{\mathtt{tl}}\}, \overrightarrow{Y} \subseteq \overrightarrow{B} \cup \{E, X_{\mathtt{tl}}\}, \text{ and} \\ \circlearrowleft^{1+} Q[Z, \overrightarrow{Y}] \triangleq \exists Z'.\, \Sigma_Q(Z, Z', \overrightarrow{Y}) * Q(Z', Z, \overrightarrow{Y}) \text{ where } \Sigma_Q \text{ is the matrix of } Q. \end{array} \quad (1)$$

The formula Σ specifies the values of the fields defined in E and the (possibly cyclic) nested list segments starting at the locations $\overrightarrow{Z}$ referenced by fields of E. We assume that Σ contains a single points-to atom in order to simplify the presentation.

The macro $\rho[X_{\mathtt{tl}}, \overrightarrow{Z}, \overrightarrow{B}]$ denotes the (non-empty) set $\{(f_1, X_1), \ldots, (f_n, X_n)\}$ where for all $1 \leq i \leq n$, $X_i \in \{X_{\mathtt{tl}}\} \cup \overrightarrow{Z} \cup \overrightarrow{B}$, $f_i \in \mathbb{F}$, and at least one $X_i = X_{\mathtt{tl}}$. Further, the macro $\circlearrowleft^{1+} Q[Z, \overrightarrow{Y}]$ is used to represent a *non-empty* cyclic (nested) list segment in Z whose shape is described by the predicate Q.

We consider several restrictions on Σ which are defined using its *Gaifman graph* $Gf[\Sigma]$. The nodes of $Gf[\Sigma]$ represent variables of Σ and the edges of $Gf[\Sigma]$ represent spatial atoms: for every (f, X) in ρ, $Gf[\Sigma]$ contains an edge from E to X labeled by f; for every predicate $Q(Z, U, \overrightarrow{Y})$, $Gf[\Sigma]$ contains an edge from Z to U labeled by Q; and for every macro $\circlearrowleft^{1+} Q[Z, \overrightarrow{Y}]$, $Gf[\Sigma]$ contains a self-loop in Z labeled by Q.

The first restriction is that $Gf[\Sigma]$ contains no cycles (other than self-loops) built solely of edges labeled by predicates. This ensures that the predicate is *precise*, i.e.,

$(S,H) \models E = F$ iff $S(E) = S(F)$

$(S,H) \models E \neq F$ iff $S(E) \neq S(F)$

$(S,H) \models \varphi \wedge \psi$ iff $(S,H) \models \varphi$ and $(S,H) \models \psi$

$(S,H) \models emp$ iff $dom(H) = \emptyset$

$(S,H) \models E \mapsto \{\rho\}$ iff $dom(H) = \{(S(E), f_i) \mid (f_i, E_i) \in \{\rho\}\}$ and for every $(f_i, E_i) \in \{\rho\}$, $H(S(E), f_i) = S(E_i)$

$(S,H) \models \Sigma_1 * \Sigma_2$ iff $\exists H_1, H_2$ s.t. $ldom(H) = ldom(H_1) \uplus ldom(H_2)$, $(S,H_1) \models \Sigma_1$, and $(S,H_2) \models \Sigma_2$

$(S,H) \models P(E,F,\overrightarrow{B})$ iff there exists $k \in \mathbb{N}$ s.t. $(S,H) \models P^k(E,F,\overrightarrow{B})$ and $ldom(H) \cap (\{S(F)\} \cup \{S(B) \mid B \in \overrightarrow{B}\}) = \emptyset$

$(S,H) \models P^0(E,F,\overrightarrow{B})$ iff $(S,H) \models E = F \wedge emp$

$(S,H) \models P^{k+1}(E,F,\overrightarrow{B})$ iff $(S,H) \models E \neq \{F\} \cup \overrightarrow{B} \wedge \exists X_{\mathtt{t1}}.\, \Sigma(E, X_{\mathtt{t1}}, \overrightarrow{B}) \ * \ P^k(X_{\mathtt{t1}}, F, \overrightarrow{B})$

$(S,H) \models \exists X.\, \varphi$ iff there exists $\ell \in Locs$ s.t. $(S[X \leftarrow \ell], H) \models \varphi$

Fig. 2. The semantics of inductive predicates ($\uplus$ denotes the disjoint union of sets and $S[X \leftarrow \ell]$ denotes the function S' s.t. $S'(X) = \ell$ and $S'(Y) = S(Y)$ for any $Y \neq X$).

for any heap, there exists at most one sub-heap on which the predicate holds. Precise assertions are very important for concurrent separation logic [7]. Then, we require that all the maximal paths of $Gf[\Sigma]$ start in E and end either in a self-loop or in a node from $\overrightarrow{B} \cup \{E, X_{\mathtt{t1}}\}$. This ensures that the predicates describe heaps where only the locations represented by variables in $F \cup \overrightarrow{B}$ are dangling. Moreover, for simplicity, we require that every node n of $Gf[\Sigma]$ has at most one outgoing edge labeled by a predicate.

For example, the predicates given in Fig. 1 describe singly linked lists, lists of acyclic lists, lists of cyclic lists, and skip lists with three levels.

We define the relation $\prec_{\mathbb{P}}$ on $\mathbb{P}$ by $P_1 \ \prec_{\mathbb{P}} \ P_2$ iff P_2 appears in the matrix of P_1. The reflexive and transitive closure of $\prec_{\mathbb{P}}$ is denoted by $\prec^*_{\mathbb{P}}$. For example, if $\mathbb{P} = \{\mathtt{skl}_1, \mathtt{skl}_2, \mathtt{skl}_3\}$, then $\mathtt{skl}_3 \prec_{\mathbb{P}} \mathtt{skl}_2$ and $\mathtt{skl}_3 \prec^*_{\mathbb{P}} \mathtt{skl}_1$.

Given a predicate P of the matrix Σ as in (1), let $\mathbb{F}_{\mapsto}(P)$ denote the set of fields f occurring in a pair (f, X) of ρ. For example, $\mathbb{F}_{\mapsto}(\mathtt{nll}) = \{s, h\}$ and $\mathbb{F}_{\mapsto}(\mathtt{skl}_3) = \mathbb{F}_{\mapsto}(\mathtt{skl}_1) = \{n_3, n_2, n_1\}$. Also, let $\mathbb{F}^*_{\mapsto}(P)$ denote the union of $\mathbb{F}_{\mapsto}(P')$ for all $P \prec^*_{\mathbb{P}} P'$. For example, $\mathbb{F}^*_{\mapsto}(\mathtt{nll}) = \{s, h, n\}$.

We assume that $\prec^*_{\mathbb{P}}$ is a partial order, i.e., there are no mutually recursive definitions in $\mathbb{P}$. Moreover, for simplicity, we assume that for any two predicates P_1 and P_2 which are incomparable w.r.t. $\prec^*_{\mathbb{P}}$ it holds that $\mathbb{F}_{\mapsto}(P_1) \cap \mathbb{F}_{\mapsto}(P_2) = \emptyset$. This is to avoid predicates named differently but which have exactly the same set of models.

Semantics. Let $Locs$ be a set of *locations*. A *heap* is a pair (S, H) where $S : Vars \cup LVars \rightarrow Locs$ maps variables to locations and $H : Locs \times \mathbb{F} \rightharpoonup Locs$ is a partial function that defines values of fields for some of the locations in $Locs$. The domain of H is denoted by $dom(H)$ and the set of locations in the domain of H is denoted by $ldom(H)$. We say that a location ℓ (resp., a variable E) is *allocated* in the heap (S, H) or that (S, H) allocates ℓ (resp., E) iff ℓ (resp., $S(E)$) belongs to $ldom(H)$.

```
// normalization
φ1 ← norm(φ1); φ2 ← norm(φ2);
if φ1 = false then return true;
if φ2 = false then return false;
// entailment of pure parts
if pure(φ1) ⇏ pure(φ2) then return false;
// entailment of shape parts
foreach a2 : points-to atom in φ2 do
    φ1[a2] ← select(φ1, a2);
    if φ1[a2] ⇏ a2 then return false;
for P2 ← max≺(ℙ) down to min≺(ℙ) do
    forall the a2 = P2(E, F, B⃗) : predicate atom in φ2 s.t. pure(φ1) ⇏ E = F do
        φ1[a2] ← select(φ1, a2);
        if φ1[a2] ⇏sh a2 then return false;
return isMarked(φ1);
```

Fig. 3. Compositional entailment checking ($\prec$ is any total order compatible with $\prec^*_{\mathbb{P}}$).

The set of heaps satisfying a formula φ is defined by the relation $(S, H) \models \varphi$ given in Fig. 2. Note that a heap satisfying a predicate $P(E, F, \overrightarrow{B})$ should not allocate any variable in $F \cup \overrightarrow{B}$ since these variables are considered not to be a part of its domain. A heap satisfying this property is called *well-formed w.r.t. the atom* $P(E, F, \overrightarrow{B})$. The set of models of a formula φ is denoted by $[\![\varphi]\!]$. Given two formulas φ_1 and φ_2, we say that φ_1 entails φ_2, denoted by $\varphi_1 \Rightarrow \varphi_2$, iff $[\![\varphi_1]\!] \subseteq [\![\varphi_2]\!]$. By an abuse of notation, $\varphi_1 \Rightarrow E = F$ (resp., $\varphi_1 \Rightarrow E \neq F$) denotes the fact that E and F are interpreted to the same location (resp., different locations) in all models of φ_1.

3 Compositional Entailment Checking

We define a procedure for reducing the problem of checking the validity of an entailment between two formulas to the problem of checking the validity of an entailment between a formula and an atom. We assume that the right-hand side of the entailment is a quantifier-free formula (which usually suffices for checking verification conditions in practice). The reduction can be extended to the general case, but it becomes incomplete.

Hence, we consider the problem of deciding validity of entailments $\varphi_1 \Rightarrow \varphi_2$ with φ_2 quantifier-free. We assume $pv(\varphi_2) \subseteq pv(\varphi_1)$; otherwise, the entailment is not valid.

The main steps of the reduction are given in Fig. 3. The reduction starts by a normalization phase (described in Sec. 3.1), which adds to each of the two formulas all the (dis-)equalities implied by spatial sub-formulas and removes all the atoms $P(E, F, \overrightarrow{B})$ representing *empty* list segments, i.e., those where $E = F$ occurs in the pure part. The normalization of a formula outputs $false$ iff the input formula is unsatisfiable.

In the second phase, the procedure tests the entailment between the pure parts of the normalized formulas. This can be done using any decision procedure for quantifier-free formulas in the first-order theory of pure equality.

For the spatial parts, the procedure builds a mapping from spatial atoms of φ_2 to sub-formulas of φ_1. Intuitively, the sub-formula $\varphi_1[a_2]$ associated to an atom a_2 of φ_2, computed by `select`, describes the region of a heap modeled by φ_1 that should

satisfy a_2. For predicate atoms, procedure `select` is called (in the second loop of the algorithm) only if there must exist a non-empty heap region that satisfies a_2, i.e., $E = F$ does not occur in φ_1, and its output is guaranteed to describe heap regions which are well-formed w.r.t. a_2. The fact that the heap region represented by $\varphi_1[a_2]$ satisfies a_2 corresponds to the usual entailment when a_2 is a points-to atom and to the entailment operator $\Rightarrow_{sh}$ when a_2 is a predicate atom. In the latter case, we cannot use the usual entailment because $\varphi_1[a_2]$ alone (i.e., without the other constraints in φ_1) may have models which are not well-formed. Therefore, $\varphi_1[a_2] \Rightarrow_{sh} a_2$ holds iff all the models of $\varphi_1[a_2]$, which are well-formed w.r.t. a_2, are also models of a_2.

If there exists an atom a_2 of φ_2, which is not entailed by the associated sub-formula, then $\varphi_1 \Rightarrow \varphi_2$ is not valid. By the semantics of the separating conjunction, the sub-formulas of φ_1 associated with two different atoms of φ_2 must not share spatial atoms. To this, the spatial atoms obtained from each application of `select` are marked and cannot be reused in the future. Note that the mapping is built by enumerating the atoms of φ_2 in a particular order: first, the points-to atoms and then the inductive predicates, in a decreasing order wrt $\prec_{\mathbb{P}}$. This is important for completeness (see Section 3.2).

The procedure `select` is detailed in Section 3.2. It returns emp if the construction of the sub-formula of φ_1 associated with the input atom fails (this implies that also the entailment $\varphi_1 \Rightarrow \varphi_2$ is not valid). If all the entailments between formulas and atoms are valid, then $\varphi_1 \Rightarrow \varphi_2$ holds provided that all the spatial atoms of φ_1 are marked (tested by `isMarked`). In Section 3.4, we introduce a procedure for checking entailments between a formula and a spatial atom.

Graph representations. Some of the sub-procedures mentioned above work on a graph representation of the input formulas, called *SL graphs*. Therefore, a formula φ is represented by a directed graph $G[\varphi]$, where each node represents a maximal set of variables that are all equal according to the pure part of φ, and each edge represents a disequality $E \neq F$ or a spatial atom. Every node n is labeled by the set of variables $\texttt{Var}(n)$ it represents; for every variable E, $\texttt{Node}(E)$ denotes the node n s.t. $E \in \texttt{Var}(n)$. Also, (1) a disequality $E \neq F$ is represented by an undirected edge between $\texttt{Node}(E)$ and $\texttt{Node}(F)$, (2) a spatial atom $E \mapsto \{(f_1, E_1), \ldots, (f_n, E_n)\}$ is represented by n directed edges, for each $1 \leq i \leq n$, an edge from $\texttt{Node}(E)$ to $\texttt{Node}(E_i)$ labeled by f_i, and (3) a spatial atom $P(E, F, \overrightarrow{B})$ is represented by a directed edge from $\texttt{Node}(E)$ to $\texttt{Node}(F)$ labeled by $P(\overrightarrow{B})$. Edges may be referred to as disequality, points-to, or predicate edges, depending on the atom they represent. Also, for simplicity, we may say that the graph representation of a formula is simply a formula.

For example, the graph representation of the formula $\psi_2 \triangleq y \neq t \wedge \texttt{nll}(x, y, z) * \texttt{skl}_2(y, t) * t \mapsto \{(s, y)\}$ is given in the top right-hand part of Fig. 4.

Running example. In the following, we use as a running example the entailment $\psi_1 \Rightarrow \psi_2$ between the formulas ψ_1, ψ_2 whose graphs are shown in the top part of Fig. 4 (with the formula ψ_2 given above and the formula ψ_1 left out for brevity).

3.1 Normalization

To infer the implicit (dis-)equalities in a formula, we adapt the boolean abstraction proposed in [6] for our logic. Therefore, given a formula φ, we define an equisatisfiable

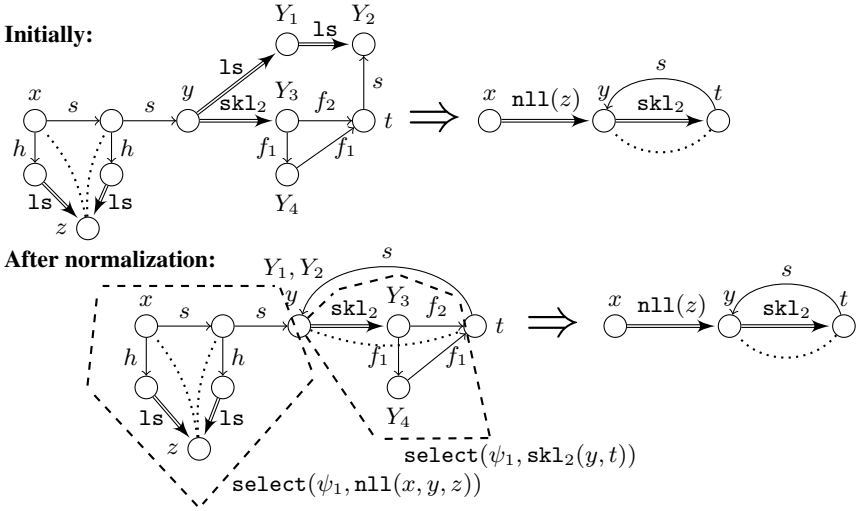


Fig. 4. An example of applying compositional entailment checking. Points-to edges are represented by simple lines, predicate edges by double lines, and disequality edges by dashed lines. For readability, we omit some of the labeling with existentially-quantified variables and some of the disequality edges in the normalized graphs.

boolean formula $\mathsf{BoolAbs}[\varphi]$ in CNF over a set of boolean variables containing the boolean variable $[E = F]$ for every two variables E and F occuring in φ and the boolean variable $[E, a]$ for every variable E and spatial atom a of the form $E \mapsto \{\rho\}$ or $P(E, F, \overrightarrow{B})$ in φ. The variable $[E = F]$ denotes the equality between E and F while $[E, a]$ denotes the fact that the atom a describes a heap where E is allocated.

Given $\varphi \triangleq \exists \overrightarrow{X}.\, \Pi \wedge \Sigma$, $\mathsf{BoolAbs}[\varphi] \triangleq F(\Pi) \wedge F(\Sigma) \wedge F_= \wedge F_*$ where $F(\Pi)$ and $F(\Sigma)$ encode the atoms of φ (using $\oplus$ to denote xor), $F_=$ encodes reflexivity, symmetry, and transitivity of equality, and F_* encodes the semantics of the separating conjunction:

$$F_\Pi \triangleq \bigwedge_{E=F \in \Pi} [E = F] \;\wedge \bigwedge_{E \neq F \in \Pi} \neg[E = F] \qquad F_\Sigma \triangleq \bigwedge_{a = E \mapsto \{\rho\} \in \Sigma} [E, a] \;\wedge \bigwedge_{a = P(E,F,\overrightarrow{B}) \in \Sigma} [E, a] \oplus [E = F]$$

$$F_= \triangleq \bigwedge_{E_1, E_2, E_3 \text{ variables in } \varphi} [E_1 = E_1] \wedge ([E_1 = E_2] \Leftrightarrow [E_2 = E_1]) \wedge ([E_1 = E_2] \wedge [E_2 = E_3] \Rightarrow [E_1 = E_3])$$

$$F_* \triangleq \bigwedge_{\substack{E, F \text{ variables in } \varphi \\ a, a' \text{different atoms in } \Sigma}} ([E = F] \wedge [E, a]) \Rightarrow \neg[F, a']$$

For example, $\mathsf{BoolAbs}[\psi_1]$ is a conjunction of several formulas including:

1. $[y, \mathtt{skl}_2(y, Y_3)] \oplus [y = Y_3]$, which encodes the atom $\mathtt{skl}_2(y, Y_3)$,
2. $[Y_3, Y_3 \mapsto \{(f_1, Y_4), (f_2, t)\}]$ and $[t, t \mapsto \{(s, Y_2)\}]$, encoding points-to atoms,
3. $([y = t] \wedge [t, t \mapsto \{(s, Y_2)\}]) \Rightarrow \neg[y, \mathtt{skl}_2(y, Y_3)]$, which encodes the separating conjunction between $t \mapsto \{(s, Y_2)\}$ and $\mathtt{skl}_2(y, Y_3)$,
4. $([Y_3 = t] \wedge [t, t \mapsto \{(s, Y_2)\}]) \Rightarrow \neg[Y_3, Y_3 \mapsto \{(f_1, Y_4), (f_2, t)\}]$, which encodes the separating conjunction between $t \mapsto \{(s, Y_2)\}$ and $Y_3 \mapsto \{(f_1, Y_4), (f_2, t)\}$.

Proposition 1. *Let φ be a formula. Then,* $\mathsf{BoolAbs}[\varphi]$ *is equisatisfiable to φ and for any two variables E, F of φ,* $\mathsf{BoolAbs}[\varphi] \Rightarrow [E = F]$ *(resp.,* $\mathsf{BoolAbs}[\varphi] \Rightarrow \neg[E = F]$*) iff $\varphi \Rightarrow E = F$ (resp. $\varphi \Rightarrow E \neq F$).*

For example, $\mathsf{BoolAbs}[\psi_1] \Rightarrow \neg[y = t]$, which is a consequence of the sub-formulas we have given above together with $F_{=}$.

If $\mathsf{BoolAbs}[\varphi]$ is unsatisfiable, then the output of $\texttt{norm}(\varphi)$ is false. Otherwise, the output of $\texttt{norm}(\varphi)$ is the formula φ' obtained from φ by (1) adding all the (dis-)equalities implied by $\mathsf{BoolAbs}[\varphi]$ and (2) removing all predicates $P(E, F, \overrightarrow{B})$ s.t. $E = F$ occurs in the pure part. For example, the normalizations of ψ_1 and ψ_2 are given in the bottom part of Fig. 4. Note that the `ls` atoms reachable from y are removed because $\mathsf{BoolAbs}[\psi_1] \Rightarrow [y = Y_1]$ and $\mathsf{BoolAbs}[\psi_1] \Rightarrow [Y_1 = Y_2]$.

The following result is important for the completeness of the `select` procedure.

Proposition 2. *Let* $\texttt{norm}(\varphi)$ *be the normal form of a formula φ. For any two* distinct *nodes n and n' in the SL graph of* $\texttt{norm}(\varphi)$*, there cannot exist two* disjoint *sets of atoms A and A' in* $\texttt{norm}(\varphi)$ *s.t. both A and A' represent paths between n and n'.*

If we assume by contradiction that $\texttt{norm}(\varphi)$ contains two such sets of atoms, then, by the semantics of the separating conjunction, $\varphi \Rightarrow E = F$ where E and F label n and n', respectively. Therefore, $\texttt{norm}(\varphi)$ does not include all the equalities implied by φ, which contradicts its definition.

3.2 Selection of Spatial Atoms

Points-to atoms. Let $\varphi_1 \triangleq \exists \overrightarrow{X}. \Pi_1 \wedge \Sigma_1$ be a normalized formula. The procedure $\texttt{select}(\varphi_1, E_2 \mapsto \{\rho_2\})$ outputs the sub-formula $\exists \overrightarrow{X}. \Pi_1 \wedge E_1 \mapsto \{\rho_1\}$ s.t. $E_1 = E_2$ occurs in Π_1 if it exists, or emp otherwise. The procedure `select` is called only if φ_1 is satisfiable and consequently, φ_1 cannot contain two different atoms $E_1 \mapsto \{\rho_1\}$ and $E_1' \mapsto \{\rho_1'\}$ such that $E_1 = E_1' = E_2$. Also, if there exists no such points-to atom, then $\varphi_1 \Rightarrow \varphi_2$ is not valid. In the running example, $\texttt{select}(\psi_1, t \mapsto \{(s, y)\}) = \exists Y_2. y = Y_2 \wedge \ldots \wedge t \mapsto \{(s, Y_2)\}$ (we have omitted some existential variables and pure atoms).

Predicate atoms. Given an atom $a_2 = P_2(E_2, F_2, \overrightarrow{B_2})$, the procedure $\texttt{select}(\varphi_1, a_2)$ outputs the formula $\exists \overrightarrow{X}. \Pi_1 \wedge \Sigma'$ where Σ' consists of all the atoms represented by edges of the sub-graph G' of $G[\varphi_1]$ described hereafter. If G' is empty, then $\Sigma' = \mathit{emp}$. Let $\texttt{Dangling}[a_2] = \texttt{Node}(F_2) \cup \{\texttt{Node}(B) \mid B \in \overrightarrow{B_2}\}$.

The graph G' is built in two steps. In the first step, G' is defined as the union of all the paths of $G[\varphi_1]$ that (1) consist of edges labeled by fields in $\mathbb{F}^*_{\mapsto}(P_2)$ or predicates Q with $P_2 \prec^*_{\mathbb{P}} Q$, (2) start in the node labeled by E_2, and (3) end either in a node from the set $\texttt{Dangling}[a_2]$ or in a cycle, in which case they must not traverse nodes in $\texttt{Dangling}[a_2]$. The paths in G' that end in a node from $\texttt{Dangling}[a_2]$ are not allowed to traverse other nodes from $\texttt{Dangling}[a_2]$. Therefore, G' does not contain edges that start in a node from $\texttt{Dangling}[a_2]$. The instances of G' for $\texttt{select}(\psi_1, \texttt{nll}(x, y, z))$ and $\texttt{select}(\psi_1, \texttt{skl}_2(y, t))$ are emphasized in Fig. 4.

In the second step, the procedure `select` checks that in every model of φ_1, the heap region described by G' is well-formed w.r.t. a_2, i.e., it does not allocate variables

in $F_2 \cup \overrightarrow{B_2}$. This is equivalent to the fact that for every variable $V \in F_2 \cup \overrightarrow{B_2}$ and every model of φ_1, the interpretation of V is different from all the allocated locations in the heap region described by G'. This is in turn equivalent to the fact that for every variable $V \in F_2 \cup \overrightarrow{B_2}$ the two following conditions hold:

1. For every predicate edge e included in G' that does not end in $\texttt{Node}(V)$, V is allocated in all the models of $E \neq F \wedge (\varphi_1 \setminus G')$ where E and F are variables labeling the source and the destination of e, respectively, and $\varphi_1 \setminus G'$ is obtained from φ_1 by deleting all the *spatial* atoms represented by edges of G'.
2. For every variable V' labeling the source of a points-to edge of G', $\varphi_1 \Rightarrow V \neq V'$.

The first condition guarantees that V is not interpreted as an allocated location in a list segment described by a predicate edge of G' (this trivially holds for predicate edges ending in $\texttt{Node}(V)$). If V was not allocated in some model (S, H_1) of $E \neq F \wedge (\varphi_1 \setminus G')$, then one could construct a model (S, H_2) of G' where e would be interpreted to a non-empty list and $S(V)$ would equal an allocated location inside this list. Therefore, there would exist a model of φ_1, defined as the union of (S, H_1) and (S, H_2), in which the heap region described by G' would not be well-formed w.r.t. a_2. For the graph $\texttt{select}(\psi_1, \texttt{skl}_2(y, t))$ in Fig. 4, t is not interpreted as an allocated location in the list segment $\texttt{skl}_2(y, Y_3)$ iff t is allocated in all the models of $y \neq Y_3 \wedge (\psi_1 \setminus \texttt{select}(\psi_1, \texttt{skl}_2(y, t)))$. The latter holds because of the atom $t \mapsto \{(s, Y_2)\}$.

To check that variables are allocated we use the following property: given a formula $\varphi \triangleq \exists \overrightarrow{X} . \Pi \wedge \Sigma$, a variable V is allocated in every model of φ iff $\exists \overrightarrow{X} . \Pi \wedge \Sigma * V \mapsto \{(f, V_1)\}$ is unsatisfiable. Here, we assume that f and V_1 are not used in φ. Note that, by Prop. 1, unsatisfiability can be decided using the boolean abstraction $\mathsf{BoolAbs}$.

The second condition guarantees that V is different from all the allocated locations represented by sources of points-to edges in G'. For the graph $\texttt{select}(\psi_1, \texttt{nll}(x, y, z))$ in Fig. 4, the variable z must be different from all the existential variables labeling a node which is the source of a points-to edge. These disequalities appear explicitly in the formula. By Prop. 1, $\varphi_1 \Rightarrow V \neq V'$ can be decided using the boolean abstraction.

Finally, at the end of the second step, if G' is not well-formed, $\texttt{select}$ returns emp, otherwise it returns the outcome of the first step.

3.3 Soundness and Completeness

The following theorem states that the procedure given in Fig. 3 is sound and complete. The soundness is a direct consequence of the semantics. The completeness is a consequence of Prop. 1 and 2. In particular, Prop. 2 implies that the sub-formula returned by $\texttt{select}(\varphi_1, a_2)$ is the only one that can describe a heap region satisfying a_2.

Theorem 1. *Let φ_1 and φ_2 be two formulas s.t. φ_2 is quantifier-free. Then, $\varphi_1 \Rightarrow \varphi_2$ iff the procedure in Fig. 3 returns $true$.*

3.4 Checking Entailments between a Formula and an Atom

Checking the validity of an entailment between a formula and a points-to atom is straightforward and we omit it for brevity. Given a formula φ and an atom $P(E, F, \overrightarrow{B})$,

we define a procedure for checking that $\varphi \Rightarrow_{sh} P(E, F, \overrightarrow{B})$, which works as follows: (1) $G[\varphi]$ is transformed into a tree $\mathcal{T}[\varphi]$ by splitting nodes that have multiple incoming edges, (2) the inductive definition of $P(E, F, \overrightarrow{B})$ is used to define a tree automaton $\mathcal{A}[P]$ s.t. $\mathcal{T}[\varphi]$ belongs to the language of $\mathcal{A}[P]$ only if $\varphi \Rightarrow_{sh} P(E, F, \overrightarrow{B})$. To keep the size of $\mathcal{A}[P]$ polynomial in the size of the inductive definition of P, this automaton does not recognize the tree representations of all the formulas φ s.t. $\varphi \Rightarrow_{sh} P(E, F, \overrightarrow{B})$ (cf. Sec. 5). The transformation of graphs into trees is presented in Sec. 4 while the definition of the tree automata is introduced in Sec. 5.

4 Representing SL Graphs as Trees

We define a canonical representation of SL graphs (ignoring their disequality edges that are treated by the entailment of pure parts) in the form of trees. This representation is at the core of checking whether a formula entails a spatial atom w.r.t. $\Rightarrow_{sh}$. Essentially, an SL graph G is represented by a tree T that contains a spanning tree of G and that is obtained by splitting every node of G with at least 2 incoming edges, called a *join node*, into several copies, one for each incoming edge. The fact that these copies do in fact represent a single node is encoded in their labelling that can either be based on a variable that points to the original node and that will appear in the labelling of all the copies, or the label may describe the path from the copy to the original node by one of two allowed ways: one intended for breaking loops and the other for breaking parallel paths between nodes. The spanning tree of G contained in T is formed of paths labeled by sequences of fields which are minimum according to the order $\prec_{\mathbb{F}^*}$ defined hereafter.

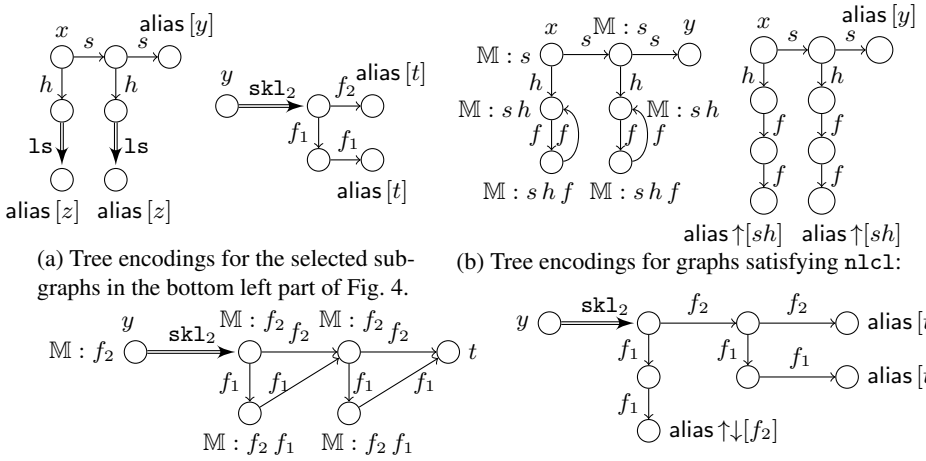
Given a predicate P with the matrix Σ as in (1), let $\mathbb{F}_{\mapsto X_{\mathtt{t1}}}(P)$ be the set of fields f occurring in a pair $(f, X_{\mathtt{t1}})$ of ρ, $\mathbb{F}_{\mapsto \overrightarrow{Z}}(P)$ the set of fields f occurring in a pair (f, Z) of ρ with $Z \in \overrightarrow{Z}$, and $\mathbb{F}_{\mapsto \overrightarrow{B}}(P)$ the set of fields f occurring in a pair (f, X) of ρ with $X \in \overrightarrow{B}$. We assume that there exists a total order $\prec_{\mathbb{F}}$ on the set of fields such that for all predicates P, P_1, P_2:

$$\forall f_1 \in \mathbb{F}_{\mapsto X_{\mathtt{t1}}}(P)\ \forall f_2 \in \mathbb{F}_{\mapsto \overrightarrow{Z}}(P)\ \forall f_3 \in \mathbb{F}_{\mapsto \overrightarrow{B}}(P).\ f_1 \prec_{\mathbb{F}} f_2 \prec_{\mathbb{F}} f_3 \text{ and}$$
$$(f_1 \in \mathbb{F}_{\mapsto}(P_1) \wedge f_2 \in \mathbb{F}_{\mapsto}(P_2) \wedge f_1 \neq f_2 \wedge P_1 \prec_{\mathbb{P}} P_2) \Rightarrow f_1 \prec_{\mathbb{F}} f_2.$$

For example, if $\mathbb{P} = \{\mathtt{nll}, \mathtt{ls}\}$ or $\mathbb{P} = \{\mathtt{nlcl}, \mathtt{ls}\}$, then $s \prec_{\mathbb{F}} h \prec_{\mathbb{F}} f$; and if $\mathbb{P} = \{\mathtt{skl}_2, \mathtt{skl}_1\}$, then $f_2 \prec_{\mathbb{F}} f_1$. The order $\prec_{\mathbb{F}}$ is extended to a lexicographic order $\prec_{\mathbb{F}^*}$ over sequences in $\mathbb{F}^*$.

Let G be an SL graph and $P(E, F, \overrightarrow{B})$ an atom for which we want to prove that $G \Rightarrow_{sh} P(E, F, \overrightarrow{B})$. The tree encoding of G is computed by the procedure $\mathtt{toTrees}(G, P(E, F, \overrightarrow{B}))$ described hereafter. We assume that all the nodes of G are reachable from the node $Root$ labeled by E. Otherwise, `toTrees` returns an error value $\bot$ because $G \Rightarrow P(E, F, \overrightarrow{B})$ does not hold. An edge of G is called an *f-edge* if it is a points-to edge labeled by f or a predicate edge labeled by $P(\overrightarrow{N})$ s.t. the minimum field in $\mathbb{F}_{\mapsto}(P)$ w.r.t. $\prec_{\mathbb{F}}$ is f. The tree encoding $\mathcal{T}[G]$ of G is built by `toTrees` as follows.

Node marking. The procedure `toTrees` starts by computing the so-called node markings that reflect the $\prec_{\mathbb{F}^*}$ ordering and that are subsequently used to identify which



(a) Tree encodings for the selected subgraphs in the bottom left part of Fig. 4.

(b) Tree encodings for graphs satisfying `nlcl`:

(c) Tree encodings for graphs satisfying $\texttt{skl}_2$:

Fig. 5. Tree encodings.

edges should be redirected and how. For every path π formed of a sequence of edges $e_1\, e_2\, \ldots e_n$ starting in Root, the *labeling of* π, denoted by $\mathbb{L}(\pi)$, is the sequence of fields $f_1\, f_2 \ldots\, f_n$ s.t. e_i is an f_i-edge, for all i. The *marking* of a node n, denoted by $\mathbb{M}(n)$, is defined as $\mathbb{M}(n) = \mathit{Reduce}(\mathbb{L}_{\min}(n))$, where $\mathbb{L}_{\min}(n)$ is the minimum $\mathbb{L}(\pi)$ (w.r.t. $\prec_{\mathbb{F}^*}$) for all paths π from Root to n, and Reduce reduces consecutive appearances of the same field to a single appearance. For technical reasons, we add the minimum field in $\mathbb{F}_{\mapsto}(P)$ at the beginning of all the markings that do not contain it. Fig. 5(b)–(c) depict two graphs and the markings of their nodes. Fig. 5(c) contains a node with the marking f_2 (the node left of the node labelled with t), which is reachable by two paths, the first with the labeling $f_2\, f_2$ and the other with $f_2\, f_1\, f_1$. The labeling of the first path is the minimum one and the marking of this node is obtained by reducing $f_2\, f_2$ to f_2.

Removing join nodes labeled by $F \cup \overrightarrow{B}$. Every edge (m, n) of G leading to a join node n labeled by a variable $V \in F \cup \overrightarrow{B}$ is replaced by an edge (m, n') where n' is a fresh copy of n. The node n' is labeled by a special symbol $\mathsf{alias}\,[V]$ to identify the node of which it is a copy. In addition, to allow all nodes labeled by variables from $F \cup \overrightarrow{B}$ to be treated uniformly, even non-join nodes labeled by a variable $V \in F \cup \overrightarrow{B}$ are re-labelled by the special symbol $\mathsf{alias}\,[V]$. This transformation is applied to obtain the tree encodings of $\texttt{select}(\psi_1, \texttt{nll}(x, y, z))$ and $\texttt{select}(\psi_1, \texttt{skl}_2(y, t))$ in Fig. 5(a).

Removing join nodes not labeled by $F \cup \overrightarrow{B}$. Let n be a join node that is not labeled by $F \cup \overrightarrow{B}$. Further, for any field f, let $\mathbb{M}(n) \odot f = \mathbb{M}(n)$ provided $\mathbb{M}(n)$ ends by f, and $\mathbb{M}(n) \odot f = \mathbb{M}(n)\, f$ otherwise. Every edge (m, n) of G labeled by a field f s.t. $\mathbb{M}(n) \neq \mathbb{M}(m) \odot f$ (meaning that (m, n) is not at the end of the minimum path leading to n) is replaced by an edge (m, n') where n' is a fresh copy of n labeled by:

– $\mathsf{alias}\!\uparrow\![\mathbb{M}(n)]$ if m is reachable from n, and all the predecessors of m (by a simple path) marked by $\mathbb{M}(n)$ are also predecessors of n. Intuitively, this label is used to

break loops, and it refers to the closest predecessor of n having the given marking.[3] The use of this labelling is illustrated in Fig. 5(b).
- $\mathsf{alias}{\uparrow}{\downarrow}[\mathbb{M}(n)]$ if there exists a node p which is a predecessor of m such that all the predecessors of m that have a unique successor marked by $\mathbb{M}(n)$ are also predecessors of p, and n is the unique successor of p marked by $\mathbb{M}(n)$. Intuitively, this transformation is used to break multiple paths between p and n.[4] The transformation is illustrated in Fig. 5(c).
- $\bot$ otherwise.

Updating the labeling of predicate edges. The labelling of predicate edges is changed in order to remove arguments which are not in $F \cup \overrightarrow{B}$. By the restrictions on the syntax of the predicate matrices, each argument X which is different from $F \cup \overrightarrow{B}$ can be replaced by $\mathsf{alias}{\uparrow}[\mathbb{M}(n)]$ or $\mathsf{alias}{\uparrow}{\downarrow}[\mathbb{M}(n)]$, which describe the position of the node n labeled by X w.r.t. the source of the predicate edge. For example, X is replaced by $\mathsf{alias}{\uparrow}[\mathbb{M}(n)]$ when n is the first predecessor of the source of the predicate edge of the marking $\mathbb{M}(n)$.

Transforming edge labels to node labels. Since the generated trees will be tested for membership in the language of a tree automaton, which accepts node-labelled trees only, `toTrees` moves labels of edges to the labels of their source nodes and concatenates them according to the ordering of the fields (predicates in the labels are ordered according to the minimum field in their matrix). The output of $\texttt{toTrees}(G, P(E, F, \overrightarrow{B}))$ is then the resulting tree or $\bot$ should the tree contain nodes labeled by $\bot$.

Proposition 3. *Let $P(E, F, \overrightarrow{B})$ be an atom and G an SL graph. If $\texttt{toTrees}(G, P(E, F, \overrightarrow{B})) = \bot$, then $G \not\Rightarrow P(E, F, \overrightarrow{B})$.*

5 Tree Automata Recognizing Tree Encodings of SL Graphs

Next, we proceed to the construction of tree automata $\mathcal{A}[P(E, F, \overrightarrow{B})]$ that recognize tree encodings of SL graphs that imply atoms of the form $P(E, F, \overrightarrow{B})$.

Tree automata. A *ranked alphabet* $\mathcal{F}$ is a finite set of symbols, each symbol having attached a unique *arity* in $\mathbb{N}$. Let $\mathcal{F}_n$ be the set of symbols of $\mathcal{F}$ of arity n. A (non-deterministic top-down) *tree automaton* over the ranked alphabet $\mathcal{F}$ is a tuple $\mathcal{A} = (Q, \mathcal{F}, q_0, \Delta)$ where Q is a set of states, $q_0 \in Q$ is the initial state, and Δ is a set of transition rules of the form $q \hookrightarrow a(q_1, \ldots, q_n)$, where $n \geq 0$, $a \in \mathcal{F}_n$, and $q, q_1, \ldots, q_n \in Q$. When a is a symbol of arity 0, a transition rule of $\mathcal{A}$ is of the form $q \hookrightarrow a$. The set of trees $L(\mathcal{A})$ recognized by $\mathcal{A}$ is defined as usual.

Definition of $\mathcal{A}[P(E, F, \overrightarrow{B})]$. The tree automaton $\mathcal{A}[P(E, F, \overrightarrow{B})]$ is defined starting from the inductive definition of P. If P does not call other predicates, the automa-

[3] The reference to the closest predecessor is needed to make the reference deterministic, e.g., when dealing with lists of cyclic lists where the nested cyclic lists close through the nodes of the top level list, not through their successors as in our running example.

[4] The combination of up and down arrows in the label corresponds to the need of going up and then down in the resulting tree—whereas in the previous case, it suffices to go up only.

ton simply recognizes the tree encodings of the SL graphs that are obtained by "concatenating" a sequence of Gaifman graphs representing the matrix $\Sigma(E, X_{\mathtt{t1}}, \overrightarrow{B})$ and predicate edges $P(E, X_{\mathtt{t1}}, \overrightarrow{B})$. In these sequences, occurrences of the Gaifman graphs representing the matrix $\Sigma(E, X_{\mathtt{t1}}, \overrightarrow{B})$ and the predicate edges $P(E, X_{\mathtt{t1}}, \overrightarrow{B})$ can be mixed in an arbitrary order and in an arbitrary number. Intuitively, this corresponds to a partial unfolding of the predicate P in which there appear concrete segments described by points-to edges as well as (possibly multiple) segments described by predicate edges. Concatenating two Gaifman graphs representing the matrix $\Sigma(E, X_{\mathtt{t1}}, \overrightarrow{B})$ means that the node labeled by $X_{\mathtt{t1}}$ in the first graph is merged with the node labeled by E in the other graph. Note that the tree encoding of such an SL graph can be computed locally on each of the Gaifman graphs. When P calls other predicates, the automaton recognizes tree encodings of concatenations of more general SL graphs, obtained from $Gf[\Sigma]$ by replacing predicate edges with unfoldings of these predicates.

The automaton $\mathcal{A}[P(E, F, \overrightarrow{B})]$ is defined over an alphabet that contains symbols of the form $\lambda : \overrightarrow{\mu}$ where λ is empty or a node label from the tree encoding, i.e., $\mathsf{alias}\,[V]$, $\mathsf{alias}{\uparrow}[\alpha]$ and $\mathsf{alias}{\uparrow\downarrow}[\alpha]$, and $\overrightarrow{\mu}$ is a vector of SL graph edge labels. The arity of a symbol is the size of μ. A transition rule of the form $q \hookrightarrow \lambda : \overrightarrow{\mu}(q_1, \ldots, q_n)$ denotes the fact that the current node is labeled by λ and it is the source of n edges, the i-th edge being labeled by $\overrightarrow{\mu}[i]$, for all $1 \leq i \leq n$. By an abuse of notation, we will write the transition rules as follows: $q \hookrightarrow \lambda : \overrightarrow{\mu}[1](q_1), \ldots, \overrightarrow{\mu}[n](q_n)$. We assume $\overrightarrow{\mu}$ to be ordered according to the ordering of fields (resp. the minimum fields in the matrix in the case of predicates) in the same way as in the case of trees encoding SL graphs.

The automaton $\mathcal{A}[P(E, F, \overrightarrow{B})]$ is defined recursively based on its matrix. Due to space constraints, we describe the construction on two typical examples only—a full description can be found in App. A. To illustrate as much as possible, we leave our running example this time (TAs for the predicates used in this example are given in App. B).

Instead, we first consider a predicate $P_1(E, F, B)$ that does not call other predicates and that has the matrix

$$\Sigma_1 \triangleq E \mapsto \{(f_1, X_{\mathtt{t1}}), (f_2, X_{\mathtt{t1}}), (f_3, B)\}.$$

The automaton $\mathcal{A}_1$ corresponding to $P_1(E, F, B)$ has the following transition rules:

$$\begin{array}{ll}
(1)\ q_0 \hookrightarrow f_1(q_0), f_2(q_1), f_3(q_2) & (5)\ q_3 \hookrightarrow \mathsf{alias}\,[F] \\
(2)\ q_1 \hookrightarrow \mathsf{alias}{\uparrow\downarrow}[f_1] & (6)\ q_0 \hookrightarrow P_1(B)(q_0) \\
(3)\ q_2 \hookrightarrow \mathsf{alias}\,[B] & (7)\ q_0 \hookrightarrow P_1(B)(q_3) \\
(4)\ q_0 \hookrightarrow f_1(q_3), f_2(q_1), f_3(q_2) &
\end{array}$$

Rules 1–3 recognize the tree encoding of the Gaifman graph of Σ_1, assuming the following total order on the fields: $f_1 \prec_{\mathbb{F}} f_2 \prec_{\mathbb{F}} f_3$. Rule 4 is used to distinguish the "last" instance of this tree encoding, which ends in the node labeled by $\mathsf{alias}\,[F]$ accepted by Rule 5. Finally, Rules 6 and 7 recognize predicate edges labeled by $P_1(B)$. As in the previous case, we distinguish the predicate edge that ends in the node labeled by $\mathsf{alias}\,[F]$. q_0 is the initial state of $\mathcal{A}_1$.

Now, let us consider a predicate $P_2(E, F)$ that calls P_1 and that has the matrix

$$\Sigma_2 \triangleq \exists Z.\, E \mapsto \{(g_1, X_{\mathtt{t1}}), (g_2, Z)\} * \circlearrowleft^{1+} P_1[Z, E].$$

The tree automaton $\mathcal{A}_2$ for $P_2(E, F)$ contains the following set of transition rules:

(1) $qq_0 \hookrightarrow g_1(qq_0), g_2(q_0)$
transition rules of $\mathcal{A}_1$, where
alias $[F]$ is substituted by alias$\uparrow[g_1\ g_2]$,
alias $[B]$ by alias$\uparrow[g_1]$, and
alias$\uparrow\downarrow[f_1]$ is substituted by alias$\uparrow\downarrow[g_1\ g_2\ f_1]$

(2) $qq_0 \hookrightarrow g_1(qq_1), g_2(q_0)$
(3) $qq_1 \hookrightarrow$ alias $[F]$
(4) $qq_0 \hookrightarrow P_2(qq_0)$
(5) $qq_0 \hookrightarrow P_2(qq_1)$

The first new rule and the ones imported from $\mathcal{A}_1$ describe tree encodings of SL graphs that imply Σ_2, obtained from $Gf[\Sigma_2]$ by replacing the self-loop on Z with its unfolding. More precisely, they describe trees obtained from the tree encoding of $Gf[\Sigma_2]$ by replacing the edge starting in Z with a tree recognized by $\mathcal{A}_1$. According to the definition of the tree encoding, the predicate edge starting in Z ends in a node labeled by alias$\uparrow[g_1\ g_2]$. Transition rules imported from $\mathcal{A}_1$ are modified in order to reflect (a) the actual arguments of the recursive call to P_1, i.e., alias $[F]$ is substituted by alias$\uparrow[g_1\ g_2]$ and alias $[B]$ by alias$\uparrow[g_1]$, and (b) the fact that trees recognized by $\mathcal{A}_1$ are now subtrees of the trees recognized by $\mathcal{A}_2$, hence the node markings change from α to $g_1\ g_2\ \alpha$. qq_0 is the initial state of $\mathcal{A}_2$.

The following result states the correctness of the tree automata construction.

Theorem 2. *For any atom $P(E, F, \overrightarrow{B})$ and any SL graph G, if the tree generated by* `toTrees`$(G, P(E, F, \overrightarrow{B}))$ *is recognized by $\mathcal{A}[P(E, F, \overrightarrow{B})]$, then $G \Rightarrow P(E, F, \overrightarrow{B})$.*

Precision. In general, there exist SL graphs that entail $P(E, F, \overrightarrow{B})$ whose tree encodings are *not* recognized by $\mathcal{A}[P(E, F, \overrightarrow{B})]$. The models of these SL graphs are nested list segments where inner pointer fields specified by the matrix of P are aliased. For example, the TA for $\mathtt{skl}_2$ does not recognize tree encodings of SL graphs modeled by heaps where $X_{\mathtt{t1}}$ and Z_1 are interpreted to the same location. The construction can be extended to cover such SL graphs (cf. [**?**]), but the size of the obtained automata may become exponential in the size of P (defined as the number of symbols in the matrices of all Q with $P \prec^*_{\mathbb{P}} Q$) as it considers all the possible aliasings of targets of inner pointer fields permitted by the predicate. For the verification conditions that we have encountered in our experiments, the TA defined above are precise enough in the vast majority of the cases. In particular, note that the TA generated for the predicates for `ls` and `dll` (defined below) are precise. We have, however, implemented even the above mentioned extension and realized that it also provides acceptable performance.

6 Extensions

The procedures presented above can be extended to a larger fragment of SL that uses more general inductively defined predicates. In particular, they can be extended to cover finite nestings of singly or doubly linked lists. To describe DLL segments between two

locations E and F where P is the predecessor of E and S is the successor of F, one can use the predicate

$$\begin{aligned}\mathtt{dll}(E,F,P,S) \triangleq\ & (E = S \wedge F = P \wedge \mathit{emp}) \vee \\ & \big(E \neq S \wedge F \neq P \wedge \exists X_{\mathtt{t1}}.\, E \mapsto \{(\mathit{next}, X_{\mathtt{t1}}), (\mathit{prev}, P)\} * \mathtt{dll}(X_{\mathtt{t1}}, F, E, S)\big)\end{aligned}$$

Finite nestings of such list segments can be defined by replacing the matrix $E \mapsto \{(next, X_{\mathtt{t1}}), (prev, P)\}$ with more general formulas that include other predicates.

The key point in this extension is the definition of the tree encoding. Basically, one needs to consider two more types of labels for the tree nodes: $\mathsf{alias}\uparrow^2[\alpha]$ with $\alpha \in \mathbb{F}^*$, which denotes the fact that the node is a copy of its second predecessor of marking α, and $\mathsf{alias}\uparrow\downarrow_{\mathsf{last}}[\alpha]$ with $\alpha \in \mathbb{F}^*$, which denotes the fact that the node is a copy of the last successor of marking α of its first predecessor that has a successor of marking α. The first label is needed to handle inner nodes of doubly linked lists, which have two incoming edges, one from their successor and one from their predecessor, while the second label is needed to "break" cyclic doubly linked lists. In the latter case, the label is used for the copy of the predecessor of the header of the list (cf. App. C for more details).

7 Implementation and Experimental Results

We implemented the decision procedure in a solver called SPEN (SeParation logic ENtailment). The tool takes as the input an entailment problem $\varphi_1 \Rightarrow \varphi_2$ (including the definition of the predicates used) encoded in the SMTLIB2 format. For non-valid entailments, SPEN prints the atom of φ_2 which is not entailed by a sub-formula of φ_1. The tool is based on the MINISAT solver for deciding unsatisfiablity of boolean formulas and the VATA library [12] as the tree automata backend.

We applied SPEN to entailment problems that use various recursive predicates[5]. First, we considered the benchmark provided in [13], which uses only the `ls` predicate. It consists of three classes of entailment problems called *spaguetti*, *bolognesa*, and *clones*. The first two classes contain 110 problems each (split into 11 groups) generated randomly according to the rules specified in [13], whereas the last class contains 100 problems (split into 10 groups) obtained from the verification conditions generated by the tool SMALLFOOT [2]. The results are listed in Table 1. We give the average time for running SPEN on the 10 problems of each group. For the first two benchmark suites, we observe a deviation of the running times of ± 100 ms w.r.t. the ones reported for SELOGER [8][6], the most efficient tool for deciding entailments of SL formulas with singly linked lists we are aware of. The TA for `ls` is quite small so these experiments evaluate the performance of the procedure in Fig. 3.

To evaluate our procedure for checking entailments between formulas and atoms, we considered experiments listed in Table 2 (among which, $\mathtt{skl}_3$ required the extension of our approach to a full decision procedure as discussed at the end of Sec. 5). The full

[5] Our experiments were performed on an Intel Core 2 Duo 2.53 GHz processor with 4 GiB DDR3 1067 MHz running a virtual machine with Fedora 20 (64-bit).

[6] The times reported for SELOGER in [8] have been obtained on an Intel Core TM i5-2467M 1.60 GHz processor with 4 GiB DDR3 1066 MHz under Windows 7 Home Premium (64-bit).

Table 1. Running SPEN on the benchmarks from [13].

Bolognesa	bo-10	bo-11	bo-12	bo-13	bo-14	bo-15	bo-16	bo-17	bo-18	bo-19	bo-20
Average time [ms]	352	386	385	394	483	562	424	510	503	516	522
Spaguetti	sp-10	sp-11	sp-12	sp-13	sp-14	sp-15	sp-16	sp-17	sp-18	sp-19	sp-20
Average time [ms]	146	156	145	153	189	258	198	254	249	252	282
Clones	cl-01	cl-02	cl-03	cl-04	cl-05	cl-06	cl-07	cl-08	cl-09	cl-10	
Average time [ms]	316	314	335	336	321	334	351	374	407	436	

Table 2. Running SPEN on entailments between formulas and atoms.

φ_2	`nll`			`nlcl`			$\texttt{skl}_3$			`dll`		
φ_1	`tc1`	`tc2`	`tc3`	`tc1`	`tc2`	`tc3`	`tc1`	`tc2`	`tc3`	`tc1`	`tc2`	`tc3`
Time [ms]	344	335	319	318	316	317	334	349	326	358	324	322
Status	vld	vld	inv	vld	vld	inv	vld	vld	inv	vld	vld	inv
States/Trans. of $\mathcal{A}[\varphi_2]$	6/17			6/15			80/193			9/16		
Nodes/Edges of $T(Gf[\varphi_1])$	7/7	7/7	6/7	10/9	7/7	6/6	7/7	8/8	6/6	7/7	7/7	5/5

benchmark is available with our tool [5]. The entailment problems are extracted from verification conditions of operations like adding or deleting an element at the start, in the middle, or at the end of various kinds of list segments (see App. D). Table 2 gives for each example the running time, the valid/invalid status, and the size of the tree encoding and TA for φ_1 and φ_2, respectively. We find the resulting times quite encouraging.

8 Conclusion

We proposed a novel (semi-)decision procedure for a fragment of SL with inductive predicates describing various forms of lists (singly or doubly linked, nested, circular, with skip links, etc.). The procedure is compositional in that it reduces the given entailment query to a set of simpler queries between a formula and an atom. For solving them, we proposed a novel reduction to testing membership of a tree derived from the formula in the language of a TA derived from a predicate. We implemented the procedure, and our experiments show that it has not only a favourable theoretical complexity, but it also efficiently handles practical verification conditions.

In the future, we plan to investigate extensions of our approach to formulas with a more general boolean structure or using more general inductive definitions. Concerning the latter, we plan to investigate whether some ideas from [9] could be used to extend our decision procedure for entailments between formulas and atoms. From a practical point of view, apart from improving the implementation of our procedure, we plan to integrate it into a complete program analysis framework.

References

1. Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. A decidable fragment of separation logic. In *Proc. of FSTTCS'04*, volume 3328 of *LNCS*, pages 97–109. Springer, 2005.
2. Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *Proc. of FMCO'05*, volume 4111 of *LNCS*, pages 115–137. Springer, 2006.

3. Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *Proc. of FSTTCS'01*, volume 2245 of *LNCS*, pages 108–119, 2001.
4. Byron Cook, Christoph Haase, Joël Ouaknine, Matthew J. Parkinson, and James Worrell. Tractable reasoning in a fragment of separation logic. In *Proc. of CONCUR'11*, volume 6901 of *LNCS*, pages 235–249. Springer, 2011.
5. Constantin Enea, Ondřej Lengál, Mihaela Sighireanu, and Tomáš Vojnar. SPEN, 2014. `http://www.liafa.univ-paris-diderot.fr/spen`.
6. Constantin Enea, Vlad Saveluc, and Mihaela Sighireanu. Compositional invariant checking for overlaid and nested linked lists. In *Proc. of ESOP'13*, volume 7792 of *LNCS*, pages 129–148. Springer, 2013.
7. Alexey Gotsman, Josh Berdine, and Byron Cook. Precision and the conjunction rule in concurrent separation logic. *Electronic Notes in Theoretical Computer Science*, 276:171–190, 2011.
8. Christoph Haase, Samin Ishtiaq, Joël Ouaknine, and Matthew J. Parkinson. SeLoger: A tool for graph-based reasoning in separation logic. In *Proc. of CAV'13*, volume 8044 of *LNCS*, pages 790–795, 2013.
9. Radu Iosif, Adam Rogalewicz, and Tomáš Vojnar. Deciding entailments in inductive separation logic with tree automata. Technical Report arXiv:1402.2127, 2014. `http://arxiv.org/pdf/1402.2127v2.pdf`.
10. Radu Iosif, Adam Rogalewicz, and Jiří Šimáček. The tree width of separation logic with recursive definitions. In *Proc. of CADE-24*, volume 7898 of *LNCS*, pages 21–38. Springer, 2013.
11. S. Ishtiaq and P.W. O'Hearn. BI as an assertion language for mutable data structures. In *POPL*, pages 14–26. ACM, 2001.
12. Ondřej Lengál, Jiří Šimáček, and Tomáš Vojnar. VATA: A library for efficient manipulation of non-deterministic tree automata. In *Proc. of TACAS'12*, volume 7214 of *LNCS*, pages 79–94. Springer, 2012.
13. J.A. Navarro Pérez and A. Rybalchenko. Separation logic + superposition calculus = heap theorem prover. In *Proc. of PLDI'11*, pages 556–566. ACM, 2011.
14. Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating separation logic using SMT. In *Proc. of CAV'13*, volume 8044 of *LNCS*, pages 773–789. Springer, 2013.
15. Xiaokang Qiu, Pranav Garg, Andrei Stefanescu, and Parthasarathy Madhusudan. Natural proofs for structure, data, and separation. In *Proc. of PLDI'13*, pages 231–242. ACM, 2013.
16. John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. of LICS'02*, pages 55–74. IEEE, 2002.

A Construction of Tree Automata for Predicates

Consider the definition of the matrix of the predicate $P(E, F, \overrightarrow{B})$ as given in Equation (1) repeated for the sake of convenience here:

$$P(E, F, \overrightarrow{B}) \triangleq (E = F \wedge emp) \vee \big(E \neq \{F\} \cup \overrightarrow{B} \wedge \exists X_{\mathtt{t1}}.\, \Sigma(E, X_{\mathtt{t1}}, \overrightarrow{B}) * P(X_{\mathtt{t1}}, F, \overrightarrow{B})\big)$$

where Σ is of the form

$$\Sigma(E, X_{\mathtt{t1}}, \overrightarrow{B}) \triangleq \exists \overrightarrow{Z}.\, E \mapsto \rho[X_{\mathtt{t1}}, \overrightarrow{Z}, \overrightarrow{B}] * \Sigma'$$

$$\Sigma' ::= Q(Z, U, \overrightarrow{Y}) \mid \circlearrowleft^{1+} Q[Z, \overrightarrow{Y}] \mid \Sigma' * \Sigma' \mid emp$$

$$\text{for } Z \in \overrightarrow{Z},\ U \in \overrightarrow{Z} \cup \overrightarrow{B} \cup \{E, X_{\mathtt{t1}}\},\ \overrightarrow{Y} \subseteq \overrightarrow{B} \cup \{E, X_{\mathtt{t1}}\}, \text{ and}$$

$$\circlearrowleft^{1+} Q[Z, \overrightarrow{Y}] \triangleq \exists Z'.\, \Sigma_Q(Z, Z', \overrightarrow{Y}) * Q(Z', Z, \overrightarrow{Y}) \text{ where } \Sigma_Q \text{ is the matrix of } Q.$$

Construction of the automaton $\mathcal{A}[P]$ is described in the following. Suppose the matrix of P is of the form $\Sigma(E, X_{\mathtt{t1}}, \overrightarrow{B}) \triangleq \exists \overrightarrow{Z}.\, E \mapsto \{(f_1, X_1), \ldots, (f_n, X_n)\} * \Sigma'$. W.l.o.g. we further assume that $f_1 \prec_{\mathbb{F}} \ldots \prec_{\mathbb{F}} f_n$ (i.e. f_1 is the minimum field in $\mathbb{F}_{\mapsto}(P)$). Before we start with the construction, we obtain the SL graph G of the matrix $\Sigma(E, X_{\mathtt{t1}}, \overrightarrow{B})$ in such a way that during its construction, we do not expand the macro $\circlearrowleft^{1+} Q[Z, \overrightarrow{Y}]$ and transform it into a predicate edge from $\mathtt{Node}(Z)$ to $\mathtt{Node}(Z)$ labelled with $Q(\overrightarrow{Y})$. Then we get the modified tree encoding[7] $\mathcal{T}[G]$ of G and check that it is not equal to $\bot$, otherwise we abort the procedure.

1. First, we create the *skeleton* of $\mathcal{A}[P]$ by taking $\mathcal{T}[G]$ and transforming it in the following way:
   (a) We start with an empty automaton $\mathcal{A}[P]$.
   (b) For each node u of $\mathcal{T}[G]$, we create a unique state $q(u)$ in $\mathcal{A}[P]$ and set $q(\mathtt{Node}(E))$ as the initial node of $\mathcal{A}[P]$.
   (c) If the node u is labelled in $\mathcal{T}[G]$ with an aliasing relation $r \in \{\mathsf{alias}\,[B], \mathsf{alias}\!\uparrow[m_1], \mathsf{alias}\!\uparrow\!\downarrow[m_2]\}$ for some border variable $B \in \overrightarrow{B}$ and markings m_1 and m_2, we add the transition
   $$q(u) \hookrightarrow r. \tag{2}$$
   (d) If there is a predicate edge from u to v labelled with $Q(\overrightarrow{Y})$, we add the transition
   $$q(u) \hookrightarrow Q(\overrightarrow{Y})(q(v)). \tag{3}$$
   Note that after transforming G to the tree $\mathcal{T}[G]$, the tuple $\overrightarrow{Y}$ can contain both variables and aliasing relations.

[7] The considered modification is the following: during the construction of $\mathcal{T}[G]$, we do not consider $X_{\mathtt{t1}}$ to be a border variable and therefore avoid aliasing relations of the form $\mathsf{alias}\,[X_{\mathtt{t1}}]$. This is because $X_{\mathtt{t1}}$ is in fact existentially quantified in the definition of P and $\mathsf{alias}\,[X_{\mathtt{t1}}]$ would therefore be ambiguous. Therefore, it needs to be substituted by the relations $\mathsf{alias}\!\uparrow[m]$ or $\mathsf{alias}\!\uparrow\!\downarrow[m]$ for some marking m.

(e) If u is the source of points-to edges $e_1, \ldots, e_k$ labelled with the fields $h_1, \ldots, h_k$ respectively, assuming that $h_1 \prec_{\mathbb{F}} \ldots \prec_{\mathbb{F}} h_k$, and entering nodes $v_1, \ldots, v_k$, in this order, we add the transition

$$q(u) \hookrightarrow h_1(q(v_1)), \ldots, h_k(q(v_k)). \tag{4}$$

Note that this rule also creates the initial transition

$$q(\mathtt{Node}(E)) \hookrightarrow h_1(q(\mathtt{Node}(X_{\mathtt{t1}}))), h_2(q(v_2)), \ldots, h_k(q(v_k)). \tag{5}$$

(f) We add the transition

$$q(\mathtt{Node}(X_{\mathtt{t1}})) \hookrightarrow \mathsf{alias}\,[F]. \tag{6}$$

Note that this skeleton is able to accept precisely a single unfolding of the predicate P between E and F such that nested predicates are not unfolded.

2. Next, we make $\mathcal{A}[P]$ accept an arbitrary number of these unfoldings along the backbone of the predicate. To do this, we take the initial transition (5) and insert into $\mathcal{A}[P]$ a new transition

$$q(\mathtt{Node}(E)) \hookrightarrow h_1(q(\mathtt{Node}(E))), h_2(q(v_2)), \ldots, h_k(q(v_k)). \tag{7}$$

3. In the following step, for each transition over a predicate symbol

$$q(\mathtt{Node}(R)) \hookrightarrow Q(\overrightarrow{Y})(q(\mathtt{Node}(S))), \tag{8}$$

in $\mathcal{A}[P]$ we instantiate the automaton for the predicate $Q(R, S, \overrightarrow{Y})$ with unique names of states and make the following substitutions. First, the initial node of $\mathcal{A}[Q(R, S, \overrightarrow{Y})]$ (including all its occurences in transitions) is renamed to $q(\mathtt{Node}(R))$. Second, every occurrence of $\mathsf{alias}\uparrow[m_1]$ and $\mathsf{alias}\uparrow\downarrow[m_2]$ is changed to $\mathsf{alias}\uparrow[m_R \odot m_1]$ and $\mathsf{alias}\uparrow\downarrow[m_R \odot m_2]$ respectively, where m_R is the marking of $\mathtt{Node}(R)$ in $\mathcal{T}[G]$. Third, every occurrence of $\mathsf{alias}\,[U]$ in a transition of $\mathcal{A}[Q(R, S, \overrightarrow{Y})]$ is changed according to the following rules:

(a) if $U \in \overrightarrow{B}$ then we keep the occurrence unchanged,

(b) if $U \in \overrightarrow{Z} \setminus \{S\}$ then $\mathsf{alias}\,[U]$ is changed to $\mathsf{alias}\uparrow\downarrow[m_U]$ where m_U is the marking of $\mathtt{Node}(U)$ in $\mathcal{T}[G]$,

(c) if $U = E$ then $\mathsf{alias}\,[U]$ is changed to $\mathsf{alias}\uparrow[f_1]$.

Fourth, let $q_{Q,\mathtt{t1}}$ be the state of $\mathcal{A}[Q(R, S, \overrightarrow{Y})]$ to which $X_{\mathtt{t1}}$ maps in $\mathcal{A}[Q(R, S, \overrightarrow{Y})]$. We remove the transition $q_{Q,\mathtt{t1}} \hookrightarrow \mathsf{alias}\,[S]$ and rename the state $q_{Q,\mathtt{t1}}$ (again, including all its occurrences in transitions) to $q(\mathtt{Node}(S))$.

4. Finally, we add transitions allowing an arbitrary interleaving of folded and unfolded occurences of the predicate P:

$$q(\mathtt{Node}(E)) \hookrightarrow P(\overrightarrow{B})(q(\mathtt{Node}(E))) \tag{9}$$

$$q(\mathtt{Node}(E)) \hookrightarrow P(\overrightarrow{B})(q(\mathtt{Node}(X_{\mathtt{t1}}))). \tag{10}$$

B Tree Automata for the Running Example

The automaton $\mathcal{A}[\texttt{ls}(E, F)]$ contains the following set of transition rules (with q_0 being the initial state):

$$q_0 \hookrightarrow f(q_0) \qquad q_0 \hookrightarrow \texttt{ls}(q_0)$$
$$q_0 \hookrightarrow f(q_1) \qquad q_0 \hookrightarrow \texttt{ls}(q_1)$$
$$q_1 \hookrightarrow \mathsf{alias}\,[F]$$

The automaton $\mathcal{A}[\texttt{nll}(G, H, B)]$ contains the following set of transition rules (with qq_0 being the initial state):

$$qq_0 \hookrightarrow s(qq_0), h(q_0) \qquad qq_0 \hookrightarrow s(qq_1), h(q_0)$$
$$qq_1 \hookrightarrow \mathsf{alias}\,[H] \qquad qq_0 \hookrightarrow \texttt{nll}(B)(qq_0)$$
$$q_0 \hookrightarrow \mathsf{alias}\,[B] \qquad qq_0 \hookrightarrow \texttt{nll}(B)(qq_1)$$
transition rules of $\mathcal{A}[\texttt{ls}(E, B)]$

The automaton $\mathcal{A}[\texttt{skl}_1(K, L)]$ for skip lists of level three contains the following set of transition rules (p_0 is the initial state):

$$p_0 \hookrightarrow f_3(p_\perp), f_2(p_\perp), f_1(p_0) \qquad p_0 \hookrightarrow \texttt{skl}_1(p_0)$$
$$p_0 \hookrightarrow f_3(p_\perp), f_2(p_\perp), f_1(p_1) \qquad p_0 \hookrightarrow \texttt{skl}_1(p_1)$$
$$p_1 \hookrightarrow \mathsf{alias}\,[L] \qquad p_\perp \hookrightarrow \mathsf{alias}\,[\texttt{NULL}]$$

The automaton $\mathcal{A}[\texttt{skl}_2(M, N)]$ contains the following set of transition rules (pp_0 is the initial state):

$$pp_0 \hookrightarrow f_3(p_\perp), f_2(pp_0), f_1(p_0) \qquad pp_0 \hookrightarrow \texttt{skl}_2(pp_0)$$
$$pp_0 \hookrightarrow f_3(p_\perp), f_2(pp_1), f_1(p_0) \qquad pp_0 \hookrightarrow \texttt{skl}_2(pp_1)$$
$$p_0 \hookrightarrow \mathsf{alias}\uparrow\downarrow[f2] \qquad pp_1 \hookrightarrow \mathsf{alias}\,[N]$$
transition rules of $\mathcal{A}[\texttt{skl}_1(K, L)]$, where
$\mathsf{alias}\,[L]$ is substituted by $\mathsf{alias}\uparrow\downarrow[f_2]$

The automaton $\mathcal{A}[\texttt{skl}_3(P, R)]$ contains the following set of transition rules (ppp_0 is the initial state):

$ppp_0 \hookrightarrow f_3(ppp_0), f_2(pp_0), f_1(p_0)$
$ppp_0 \hookrightarrow f_3(ppp_1), f_2(pp_0), f_1(p_0)$
$pp_0 \hookrightarrow \mathsf{alias}\uparrow\downarrow[f_3]$
transition rules of $\mathcal{A}[\texttt{skl}_2(M, N)]$, where
$\mathsf{alias}\,[N]$ is substituted by $\mathsf{alias}\uparrow\downarrow[f_3]$
$\mathsf{alias}\uparrow\downarrow[f_2]$ is substituted by $\mathsf{alias}\uparrow\downarrow[f_3\ f_2]$

$ppp_0 \hookrightarrow \texttt{skl}_3(ppp_0)$
$ppp_0 \hookrightarrow \texttt{skl}_3(ppp_1)$
$ppp_1 \hookrightarrow \mathsf{alias}\,[R]$
$p_0 \hookrightarrow \mathsf{alias}\uparrow\downarrow[f_3\ f_2]$
transition rules of $\mathcal{A}[\texttt{skl}_1(K, L)]$, where
$\mathsf{alias}\,[N]$ is substituted by $\mathsf{alias}\uparrow\downarrow[f_3\ f_2]$

The automaton $\mathcal{A}[\texttt{nlcl}(S, T)]$ contains the following set of transition rules (with qq_0 being the initial state):

$$qq_0 \hookrightarrow s(qq_0), h(q_0) \qquad qq_0 \hookrightarrow s(qq_1), h(q_0)$$
$$qq_1 \hookrightarrow \mathsf{alias}\,[T] \qquad qq_0 \hookrightarrow \texttt{nlcl}(qq_0)$$
transition rules of $\mathcal{A}[\texttt{ls}(E, F)]$, where $\quad qq_0 \hookrightarrow \texttt{nlcl}(qq_1)$
$\mathsf{alias}\,[F]$ is substituted by $\mathsf{alias}\uparrow[s\ h]$

C Extending the tree encoding to deal with doubly linked lists

Let us consider the following predicates for describing doubly linked list segments and lists of cyclic doubly linked lists:

$$\texttt{dll}(E, F, P, S) \triangleq (E = S \wedge F = P \wedge emp) \vee \big(E \neq S \wedge F \neq P \wedge \exists X_{\texttt{tl}}.\, E \mapsto \{(n, X_{\texttt{tl}}), (p, P)\} * \texttt{dll}(X_{\texttt{tl}}, F, E, S)\big)$$

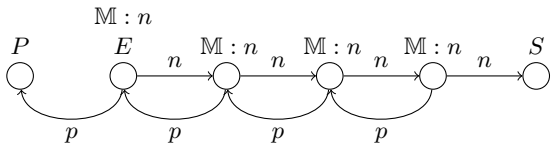
$$\texttt{nlcdl}(E, F) \triangleq (E = F \wedge emp) \vee \big(E \neq F \wedge \exists X_{\texttt{tl}}, Z.\, E \mapsto \{(s, X_{\texttt{tl}}), (h, Z)\} * \circlearrowleft^{1+}\, \texttt{dll}(Z) * \texttt{nlcdl}(X_{\texttt{tl}}, F)\big)$$

where $\circlearrowleft^{1+}\ \texttt{dll}(Z)$ is a macro for describing non-empty cyclic doubly linked lists defined by

$$\circlearrowleft^{1+}\ \texttt{dll}[Z] \triangleq \exists Z_1, Z_2.\, Z \mapsto \{(n, Z_1), (p, Z_2)\} * \texttt{dll}(Z_1, Z_2, Z, Z).$$

The SL graphs of two formulas that entail $\texttt{dll}(E, F, P, S)$ and $\texttt{nlcdl}(E, F)$ and their tree encodings are given in Fig. 6 and Fig. 7 respectively. To deal with doubly linked lists, one has to modify the step of the procedure `toTrees` that removes join nodes which are not labeled by P or S in the case of `dll` and by F in the case of `nlcdl`. These are the arguments that are not supposed to be allocated in any model of the predicate. Basically, we need to consider the labels $\mathsf{alias}{\uparrow^2}[\alpha]$ and $\mathsf{alias}{\uparrow}{\downarrow_{\mathsf{last}}}[\alpha]$ introduced in Sec. 6.

An SL graph which entails $\texttt{dll}(E, F, P, S)$:



and its tree encoding:

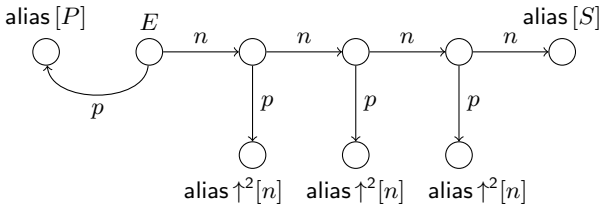
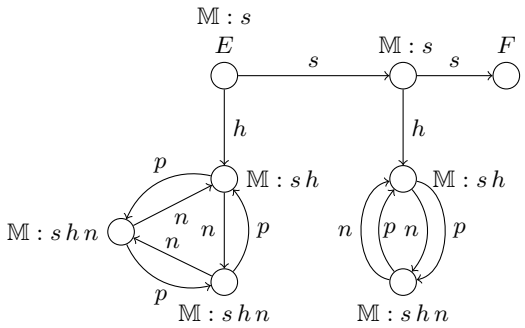


Fig. 6. Tree encodings for doubly linked lists.

An SL graph which entails $\texttt{nlcdl}(E, F)$:



and its tree encoding:

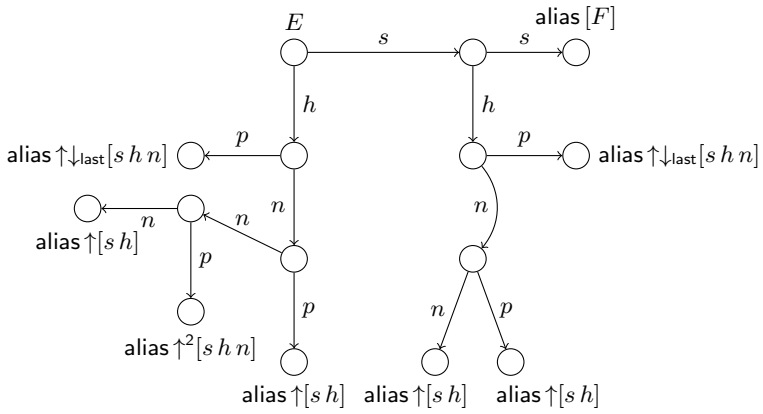


Fig. 7. Tree encodings for lists of cyclic doubly linked lists.

D Details of the Experiments

In the following, we give the formulas for φ_1 used in the experiments for checking entailments between formulas and atoms. For all cases, entailments are valid for `tc1` and `tc2`, and invalid for `tc3`.

- $\varphi_2 = \mathtt{nll}(x, y, z)$

$$\begin{aligned}
\mathtt{tc1} &\triangleq x \mapsto \{(s,u),(h,a)\} * u \mapsto \{(s,y),(h,b)\} * \mathtt{ls}(a,z) * \mathtt{ls}(b,z) \\
\mathtt{tc2} &\triangleq \mathtt{nll}(x,u,z) * u \mapsto \{(s,w),(h,a)\} * a \mapsto \{(f,b)\} * \mathtt{ls}(b,z) * \\
&\quad\ \mathtt{nll}(w,y,z) \\
\mathtt{tc3} &\triangleq \mathtt{nll}(x,u,z) * u \mapsto \{(s,w),(h,a)\} * a \mapsto \{(f,b)\} * b \mapsto \{(f,a)\} * \\
&\quad\ \mathtt{nll}(w,y,z)
\end{aligned}$$

- $\varphi_2 = \mathtt{nlcl}(x, y)$

$$\begin{aligned}
\mathtt{tc1} &\triangleq x \mapsto \{(s,u),(h,a)\} * a \mapsto \{(f,b)\} * b \mapsto \{(f,a)\} * \\
&\quad\ u \mapsto \{(s,y),(h,c)\} * c \mapsto \{(f,d)\} * \mathtt{ls}(d,c) \\
\mathtt{tc2} &\triangleq \mathtt{nlcl}(x,u) * u \mapsto \{(s,v),(h,a)\} * a \mapsto \{(f,b)\} * \mathtt{ls}(b,a) * \\
&\quad\ \mathtt{nlcl}(v,y) \\
\mathtt{tc3} &\triangleq \mathtt{nlcl}(x,u) * u \mapsto \{(s,v),(h,a)\} * a \mapsto \{(f,y)\} * \mathtt{nlcl}(v,y)
\end{aligned}$$

- $\varphi_2 = \mathtt{skl}_3(x, y)$

$$\begin{aligned}
\mathtt{tc1} &\triangleq x \mapsto \{(f_1,z),(f_2,z),(f_3,z)\} * z \mapsto \{(f_1,y),(f_2,y),(f_3,y)\} \\
\mathtt{tc2} &\triangleq \mathtt{skl}_3(x,z) * z \mapsto \{(f_3,w),(f_2,z_2)(f_1,z_1)\} * \mathtt{skl}_1(z_1,z_2) * \\
&\quad\ \mathtt{skl}_2(z_2,w) * \mathtt{skl}_3(w,y) \\
\mathtt{tc3} &\triangleq x \mapsto \{(f_1,w),(f_2,w),(f_3,w)\} * w \mapsto \{(f_1,z),(f_2,w_2),(f_3,z)\} * \\
&\quad\ \mathtt{skl}_2(w_2,z) * \mathtt{skl}_3(z,y)
\end{aligned}$$

- $\varphi_2 = \mathtt{dll}(x, y, z, v)$

$$\begin{aligned}
\mathtt{tc1} &\triangleq x \mapsto \{(n,u),(p,z)\} * u \mapsto \{(n,y),(p,x)\} * y \mapsto \{(n,v),(p,u)\} \\
\mathtt{tc2} &\triangleq x \mapsto \{(n,u),(p,z)\} * \mathtt{dll}(u,w,x,y) * y \mapsto \{(n,v),(p,w)\} \\
\mathtt{tc3} &\triangleq x \mapsto \{(n,u),(p,z)\} * \mathtt{dll}(u,w,x,y) * y \mapsto \{(n,v)\}
\end{aligned}$$