Tractability of Separation Logic with Inductive Definitions: Beyond Lists *

Taolue Chen[1,4], Fu Song[2], and Zhilin Wu[3]

1 **Department of Computer Science and Information Systems, Birkbeck, University of London, UK**
2 **School of Information Science and Technology, ShanghaiTech University, China**
3 **State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China**
4 **State Key Laboratory of Novel Software Technology, Nanjing University, China**

Abstract

In 2011, Cook et al. showed that the satisfiability and entailment can be checked in polynomial time for a fragment of separation logic that allows for reasoning about programs with pointers and linked lists. In this paper, we investigate whether the tractability results can be extended to more expressive fragments of separation logic that allow defining data structures beyond linked lists. To this end, we introduce separation logic with a *simply-nonlinear compositional* inductive predicate where source, destination, and static parameters are identified explicitly ($\mathsf{SLID}_{\mathsf{SNC}}$). We show that if the inductive predicate has more than one source (destination) parameter, the satisfiability problem for $\mathsf{SLID}_{\mathsf{SNC}}$ becomes intractable in general. This is exemplified by an inductive predicate for doubly linked list segments. By contrast, if the inductive predicate has only one source (destination) parameter, the satisfiability and entailment problems for $\mathsf{SLID}_{\mathsf{SNC}}$ are tractable. In particular, the tractability results hold for inductive predicates that define list segments with tail pointers and trees with one hole.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Separation logic, Inductive definitions, Satisfiability, Entailment

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2017.33

1 Introduction

Separation Logic (SL, [23, 25]), an extension of Hoare logic, is a well-established formalism for the verification of heap manipulating programs. SL features a *separating conjunction operator* and *inductive predicates*, which allow to express how data structures are laid out in memory in a succinct way. Since its introduction, various verification tools based on separation logic have been developed. A notable one among them is the INFER tool [9], which was acquired by Facebook in 2013 and has been actively used in its development process [10]. To enable program verification with SL assertions in a Hoare-logic style, it is vital to be able to check satisfiability and entailment of SL formulae. Unfortunately, entailment checking in full SL is undecidable [11]. Thus one has to either resort to heuristics or consider restricted, decidable fragments. We focus on the latter approach in this paper. Earlier efforts, including the theoretical work [2] which leads to the tool SmallFoot [3], considered SL with primitives of pointers and the hardwired singly linked list segments. A remarkable result on this fragment was given by Cook et al. [14], who developed a polynomial-time algorithm for

* This work was partially supported by the UK EPSRC grant (EP/P00430X/1), the European CHIST-ERA project SUCCESS, the NSFC grants (61472474, 61402179, 61532019, 61662035, 61572478).

the satisfiability and entailment problems based on graph reasoning. This has also led to the tool SeLoger [19].

Hardwired inductive predicates have obvious limitations. More recent research has been focusing on automated reasoning about generic *user-defined* inductive predicates inside the framework of SL. This class of logic is usually referred to as *Separation Logic with Inductive Definitions* (SLID). Because of their generality, satisfiability and entailment checking are usually much more difficult. Nevertheless, notable progress has been made in this direction. For instance, Iosif et al. showed decidability of satisfiability and entailment for an expressive fragment of SLID by a reduction to Monadic Second Order Logic on graphs with bounded tree width [20]. Later on, they reduced the entailment problem to the language inclusion problem of tree automata, and implemented the tool SLIDE [21]. Brotherston et al. showed that the satisfiability problem (but not entailment) for a very expressive fragment of SLID is ExpTime-complete [6]. In the same paper, they also proved that the satisfiability problem becomes Np-complete if the arities of all predicates are bounded by a constant.

Contributions. We are mostly interested in tractable (polynomial-time) satisfiability and entailment checking for SLID. Our strategy is to generalise the graph reasoning initiated in [14], which currently is limited to singly linked lists only. To this end, we concentrate on a class of *simply-nonlinear compositional* inductive predicates, where the source, destination, and static parameters are identified explicitly. This class of inductive predicates enjoys some nice compositional properties introduced in [17], which makes, for example, the entailment problem easier to solve. The main question is *whether the satisfiability and entailment checking for the resulting SLID fragment, denoted by* $\mathsf{SLID}_{\mathsf{SNC}}$*, can be done in polynomial time.* We obtain both positive and negative results in this regard.

(1) We show that, with an inductive predicate *dllseg* for *doubly linked list segments*, the satisfiability problem for $\mathsf{SLID}_{\mathsf{SNC}}[dllseg]$ is Np-complete. From the perspective of data structures, doubly linked list segments are perhaps the minimal extension of singly linked list segments, and thus one would have expected the tractability result for doubly linked list segments by a straightforward adaptation of the method for singly linked list segments by Cook et al. in [14]. Our result is among the strongest intractability results for the satisfiability problem for SLID so far. Note that the Np-hardness reduction in [6] requires four predicates, while our reduction uses only one predicate *dllseg*. However, it should be noted that there are at most three parameters for predicates in [6], but *dllseg* has four parameters (i.e., two source and destination parameters respectively). The result suggests that $\mathsf{SLID}_{\mathsf{SNC}}$ becomes intractable in general if inductive predicates with more than one source (destination) parameter are allowed.

(2) We show that both the satisfiability and the entailment checking can be done in polynomial time if the inductive predicate has only one source (destination) parameter. Remarkably, the fragment covers linear data structures (e.g., list segments with tail pointers), as well as non-linear data structures (e.g., trees with one hole). To our best knowledge, this is one of the first tractability results of SLID for a *class* of user-defined inductive predicates. (Independently, [8] studied a different class mostly on the model checking problem; cf. *related work* for comparison.)

To achieve this, we generalise the graph-theoretic techniques developed in [14], in particular, the concept of graph homomorphisms, to non-linear structures. Compared to [14], the graph representations of the $\mathsf{SLID}_{\mathsf{SNC}}$ formulae have a considerably more involved structure, e.g., the unique simple path property between a pair of nodes in [14] is lost here. An additional intricacy is that we consider the classical semantics of $\mathsf{SLID}_{\mathsf{SNC}}$ formulae rather than the (less general)*intuitionistic* semantics in [14]. The authors of [14] did briefly mention

that, to tackle the classical semantics, extra conditions—although not specified—should be added to ensure that all arcs are covered in the graph homomorphism. They, however, suggested that "the details are messy and deferred to a full version." We concur with these insights. As singly linked list segments are a (very) special case of our inductive predicates, the decision procedure in this paper provides a complete account for the entailment checking under the classical semantics.

Related work. There is a vast literature on separation logic, and we will have to focus on work which is the most relevant to us. In particular, we only cover the work on SLID, and skip many results on, for instance, first-order separation logic (without inductive definitions) [4, 15]. Antonopoulos et al. established some fundamental decidability and complexity results for the entailment problem of SLID [1]. In particular, they showed that deciding the entailment $\varphi \models \psi$ for SLID is intractable where ψ may contain existentially quantified variables, even for a single predicate of list segments. This result is largely incomparable to us, since existential quantified variables are disallowed in our setting. The work [5, 13] took the cyclic-proof approach which is based on induction on the paths of proof trees, but the decision procedures therein are incomplete in general. Brotherston et al. investigated array separation logic and obtained some complexity results [7]: They showed that the satisfiability is NP-complete, entailment is decidable with high complexity, and bi-abduction is in NP. Recently, [8] gave another tractable fragment of SLID, but mostly focused on the model checking problem. The tractable fragment of [8], defined by three constraints MEM ("Memory-consuming"), CV ("Constructively valued") and DET ("Deterministic"), is incomparable to ours: "trees with one hole" is in our tractable fragment, while it does not belong to MEM+CV+DET, as the DET constraint fails; however, an MEM+CV+DET formula may contain several different inductive predicates, while an $\mathsf{SLID}_{\mathsf{SNC}}$ formula allows at most one inductive predicate. We also mention recent work considering decision procedures for SLID extended with data and size constraints [12, 18, 26, 22, 24, 27].

In the sequel, we discuss [18] and [16] which are technically more related to this paper. Strictly speaking, [18] considered SLID extended with data constraints, so here we limit the comparison to the data-free part. First of all, [18] tackled *linear* structures, while here we focus on more general non-linear structures (e.g., trees). More importantly, the decision procedures in [18] are *intractable*, even when specialised to the data-free setting. Indeed, for both satisfiability and entailment, only NP upper-bounds can be obtained. The current work improves [18] by giving polynomial-time algorithms for even non-linear structures. In particular, for satisfiability, we extend the graph-theoretic approach in [14]; for entailment, we give a new definition of graph homomorphism, which is much more involved than that in [18], but yields an efficient checking algorithm.

[16] considered a fragment of SLID where nested lists and skip lists are expressible, and provided an *incomplete* decision procedure for the entailment problem by generalising the graph homomorphism in [14]. A more technical comparison is deferred to Section 4.2.

2 Preliminaries

For $n \in \mathbb{N}$, $[n] := \{1, \ldots, n\}$. We assume a set of variables Vars ranged over by $E, F, X, Y, \cdots$, and a set of *locations* $\mathbb{L}$ typically ranged over by $l, l', \cdots$. We assume a designated variable $\mathsf{nil} \in \mathsf{Vars}$ (for the "null" value of pointers in programs) and $\mathbb{L}$ contains a special element null. Moreover, we assume a set of fields $\mathcal{F}$ ranged over by $f, f', \cdots$. We focus on separation logic with a *simply-nonlinear compositional* inductive predicate (denoted by $\mathsf{SLID}_{\mathsf{SNC}}[P]$):

$$\begin{array}{lcl} \Pi & ::= & E = F \mid E \neq F \mid \Pi \wedge \Pi, \\ \Sigma & ::= & \mathtt{emp} \mid E \mapsto \rho \mid P(E, \boldsymbol{E'}; F, \boldsymbol{F'}; \boldsymbol{B}) \mid \Sigma * \Sigma, \end{array}$$

where ρ is a tuple of elements from $\mathcal{F} \times \mathsf{Vars}$, and $P(E, \boldsymbol{E}'; F, \boldsymbol{F}'; \boldsymbol{B})$ is a simply-nonlinear compositional predicate whose definition will be specified shortly.

The formulae Π and Σ are called the *pure* and *spatial* formulae respectively. Atomic formulae of the form $E \mapsto \rho$ and $P(E, \boldsymbol{E}'; F, \boldsymbol{F}'; \boldsymbol{B})$ are called the *points-to* and *predicate* atoms respectively. For an $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formula φ, we use $\mathsf{Vars}(\varphi)$ to denote the set of variables occurring in φ, in addition, we use $\mathsf{Atoms}(\varphi)$ to denote the set of points-to or predicate atoms occurring in φ. We require that the predicate P in $\mathsf{SLID}_{\mathsf{SNC}}[P]$ is *simply-nonlinear compositional*, that is, of the following form:

1. The parameters of P are divided into three categories: the *source* parameters $E, \boldsymbol{E}'$, the *destination* parameters $F, \boldsymbol{F}'$, and the *static* parameters $\boldsymbol{B}$. (Note that $\boldsymbol{E}', \boldsymbol{F}', \boldsymbol{B}$ denote a vector of variables.) We require that the source and the destination parameters are *symmetric*, in particular, they must be of the same length. A predicate P is usually denoted by $P(E, \boldsymbol{E}'; F, \boldsymbol{F}'; \boldsymbol{B})$ to highlight the parameters.
2. $P(E, \boldsymbol{E}'; F, \boldsymbol{F}'; \boldsymbol{B})$ is defined by a set of rules including:
   - base rule: $P(E, \boldsymbol{E}'; F, \boldsymbol{F}'; \boldsymbol{B}) ::= E = F \wedge \boldsymbol{E}' = \boldsymbol{F}' \wedge \mathtt{emp}$,
   - inductive rule:
     $$P(E, \boldsymbol{E}'; F, \boldsymbol{F}'; \boldsymbol{B}) ::= \exists \boldsymbol{X}.\ E \mapsto [(f_1, Y_1), \ldots, (f_k, Y_k)]* P(X_0, \boldsymbol{X_0'}; F, \boldsymbol{F}'; \boldsymbol{B}) * P(X_1, \boldsymbol{X_1'}; \mathsf{nil}, \mathbf{nil}; \boldsymbol{B}) * \cdots * P(X_l, \boldsymbol{X_l'}; \mathsf{nil}, \mathbf{nil}; \boldsymbol{B}),$$
     such that
     - $X_0, \ldots, X_l$ are mutually distinct variables and $\boldsymbol{X} = \{X_0, \ldots, X_l\}$,
     - $Y_1, \ldots, Y_k \in \boldsymbol{E}' \cup \boldsymbol{B} \cup \boldsymbol{X} \cup \{\mathsf{nil}\}$ and each variable from $\boldsymbol{E}' \cup \boldsymbol{B} \cup \boldsymbol{X}$ occurs exactly once in $E \mapsto [(f_1, Y_1), \ldots, (f_k, Y_k)]$,
     - for each $i : 0 \leqslant i \leqslant l$, $\boldsymbol{X_i'} \subseteq \{E\} \cup \boldsymbol{X}$.

Note that above, in $Y_1, \ldots, Y_k \in \boldsymbol{E}' \cup \boldsymbol{B} \cup \boldsymbol{X} \cup \{\mathsf{nil}\}$ or $\boldsymbol{X_i'} \subseteq \{E\} \cup \boldsymbol{X}$, $\boldsymbol{E}'$ should be interpreted as the *set* of variables occurring in $\boldsymbol{E}'$, similarly for $\boldsymbol{B}, \boldsymbol{X}, \boldsymbol{X_i'}$. We assume that, if there are multiple inductive rules, the same set of fields $\mathsf{Flds}(P)$ is used. In addition, we define the set $\mathsf{PFlds}(P)$ of *principal fields*, as the set of fields $f \in \mathsf{Flds}(P)$ such that there is an inductive rule of P where (f, Z) occurs for some $Z \in \boldsymbol{X}$, and the set $\mathsf{AFlds}(P)$ of *auxiliary fields*, as the set of fields $f \in \mathsf{Flds}(P)$ such that there is an inductive rule of P where (f, Z') occurs for some $Z' \in \boldsymbol{E}' \cup \boldsymbol{B} \cup \{\mathsf{nil}\}$. We assume that $\mathsf{PFlds}(P) \cap \mathsf{AFlds}(P) = \varnothing$, which implies that $\mathsf{Flds}(P) = \mathsf{PFlds}(P) \uplus \mathsf{AFlds}(P)$.

In the base or inductive rule, the formula on the left (resp. right) of $::=$ is called the *head* (resp. *body*) of the rule. For each predicate $P(E, \boldsymbol{E}'; F, \boldsymbol{F}'; \boldsymbol{B})$, there is exactly one base rule, and in each inductive rule, each destination parameter from $F, \boldsymbol{F}'$ occurs exactly once in the body of the rule, namely, in $P(X_0, \boldsymbol{X_0'}; F, \boldsymbol{F}'; \boldsymbol{B})$. We use $\mathcal{P}$ to denote the set of simply-nonlinear compositional inductive predicates. Moreover, we use $\mathcal{P}_1$ to denote the set of inductive predicates $P(E; F; \boldsymbol{B}) \in \mathcal{P}$, i.e., the set of inductive predicates with a single source resp. destination parameter.

Semantics. Each $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formula is interpreted on *states*. Formally, given a finite set of fields $\mathbb{F} \subseteq \mathcal{F}$, a *state* is a pair (s, h), where

- s is a *stack* which is a partial function $\mathsf{Vars} \rightharpoonup \mathbb{L}$ such that $\mathsf{dom}(s)$ is finite, $\mathsf{nil} \in \mathsf{dom}(s)$, and $s(\mathsf{nil}) = \mathsf{null}$,
- h is a *heap* which is a partial function $\mathbb{L} \times \mathbb{F} \rightharpoonup \mathbb{L}$ such that $\mathsf{dom}(h)$ is finite, $h(\mathsf{null}, f)$ is undefined for each $f \in \mathbb{F}$, and for each $l \in \mathbb{L}$, if $h(l, f)$ is defined for some $f \in \mathbb{F}$, then $h(l, f')$ is defined for each $f' \in \mathbb{F}$.

We use States to denote the set of states. For a heap h, we use $\mathsf{ldom}(h)$ to denote the set of locations $l \in \mathbb{L}$ such that $h(l, f)$ is defined for some $f \in \mathbb{F}$. Note that $\mathsf{null} \notin \mathsf{ldom}(h)$. We

write $h_1 \# h_2$ if $\mathsf{ldom}(h_1) \cap \mathsf{ldom}(h_2) = \varnothing$, in which case we write $h_1 \uplus h_2$ for the union of h_1 and h_2. A heap h_1 is called a *subheap* of h_2 if $\mathsf{ldom}(h_1) \subseteq \mathsf{ldom}(h_2)$ and for each $l \in \mathsf{ldom}(h_1)$ and $f \in \mathbb{F}$, $h_1(l, f) = h_2(l, f)$.

Let $(s, h) \in \mathsf{States}$ and φ be an $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formula. Then the semantics of φ is defined as follows,

- $(s, h) \models E = F$ (resp. $(s, h) \models E \neq F$) if $s(E) = s(F)$ (resp. $s(E) \neq s(F)$),
- $(s, h) \models \Pi_1 \wedge \Pi_2$ if $(s, h) \models \Pi_1$ and $(s, h) \models \Pi_2$,
- $(s, h) \models \mathtt{emp}$ if $\mathsf{ldom}(h) = \varnothing$,
- $(s, h) \models E \mapsto [(f_1, X_1), \cdots, (f_k, X_k)]$ if $\mathsf{ldom}(h) = s(E)$, and for each $i : 1 \leqslant i \leqslant k$, $h(s(E), f_i) = s(X_i)$,
- $(s, h) \models P(E, \boldsymbol{E'}; F, \boldsymbol{F'}; \boldsymbol{B})$ if $(s, h) \in [\![P(E, \boldsymbol{E'}; F, \boldsymbol{F'}; \boldsymbol{B})]\!]$,
- $(s, h) \models \Sigma_1 * \Sigma_2$ if there are h_1, h_2 such that $h = h_1 \uplus h_2$, $(s, h_1) \models \Sigma_1$ and $(s, h_2) \models \Sigma_2$,

where the semantics of predicate $[\![P(E, \boldsymbol{E'}; F, \boldsymbol{F'}; \boldsymbol{B})]\!]$ is given by the least fixepoint of a monotone operator constructed from the body of rules for P. More concretely, assume that P contains m inductive rules, each of which is denoted by R_i and is of the form $R_i : P(E, \boldsymbol{E'}, F, \boldsymbol{F'}, \boldsymbol{B}) ::= \Theta_i$ (i.e., Θ_i is the body of R_i). For each R_i (where $i \in [m]$), we define a monotone operator $\tau_i : 2^{\mathsf{States}} \to 2^{\mathsf{States}}$ as follows: $\tau_i(S) := \{(s, h) \mid (s, h) \models_{P,S} \Theta_i\}$, where $\models_{P,S}$ is the satisfaction relation defined above, except that P is interpreted by S, that is, $(s, h) \models P(E, \boldsymbol{E'}; F, \boldsymbol{F'}; \boldsymbol{B})$ if $(s, h) \in S$. We finally define $[\![P]\!] := \mu S. \bigcup_i \tau_i(S)$.

▸ **Example 1.** We use predicates *lseg*, *dllseg*, *tlseg*, and *th* to exemplify our definitions, which represent list segments, doubly linked list segments, list segments with tail pointers, and binary trees with one hole, respectively.

$$
\begin{array}{lcl}
\mathit{lseg}(E; F) & ::= & E = F \wedge \mathtt{emp}, \\
\mathit{lseg}(E; F) & ::= & \exists X.\ E \mapsto (\mathtt{next}, X) * \mathit{lseg}(X, F). \\
\mathit{dllseg}(E, P; F, L) & ::= & E = F \wedge P = L \wedge \mathtt{emp}, \\
\mathit{dllseg}(E, P; F, L) & ::= & \exists X.\ E \mapsto [(\mathtt{next}, X), (\mathtt{prev}, P)] * \mathit{dllseg}(X, E; F, L). \\
\mathit{tlseg}(E; F; B) & ::= & E = F \wedge \mathtt{emp}, \\
\mathit{tlseg}(E; F; B) & ::= & \exists X.\ E \mapsto [(\mathtt{next}, X), (\mathtt{tail}, B)] * \mathit{tlseg}(X; F; B). \\
\mathit{th}(E; F) & ::= & E = F \wedge \mathtt{emp}, \\
\mathit{th}(E; F) & ::= & \exists X, Y.\ E \mapsto [(\mathtt{left}, X), (\mathtt{right}, Y)] * \mathit{th}(X; \mathsf{nil}) * \mathit{th}(Y; F), \\
\mathit{th}(E; F) & ::= & \exists X, Y.\ E \mapsto [(\mathtt{left}, X), (\mathtt{right}, Y)] * \mathit{th}(X; F) * \mathit{th}(Y; \mathsf{nil}).
\end{array}
$$

For $P \in \mathcal{P}$, we investigate the following two decision problems:

Satisfiability. Given an $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formula φ, decide whether there is a state $(s, h) \models \varphi$.

Entailment. Given two $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formulae φ and ψ with $\mathsf{Vars}(\psi) \subseteq \mathsf{Vars}(\varphi)$, decide whether $\varphi \models \psi$, i.e., for every state (s, h), $(s, h) \models \varphi$ implies $(s, h) \models \psi$.

The inductive predicates defined above fall into the categy of *compositional inductive predicates* [17] and admit the composition lemma, i.e., $P(E_1, \boldsymbol{E'_1}; E_2, \boldsymbol{E'_2}; \boldsymbol{B}) * P(E_2, \boldsymbol{E'_2}; E_3, \boldsymbol{E'_3}; \boldsymbol{B}) \models P(E_1, \boldsymbol{E'_1}; E_3, \boldsymbol{E'_3}; \boldsymbol{B})$ holds, which is crucial for the decision procedure of entailment.

In literature, there is usually a distinction of the *classical* semantics and the *intuitionistic* semantics. The aforementioned semantics for $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formulae are called the classical semantics. In the intuitionistic semantics, pure and spatial formulae are interpreted in the same way as the classical one. The only difference is, in the intuitionistic semantics, we have, for an $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formula $\varphi = \Pi \wedge \Sigma$, $(s, h) \models \varphi$ iff *there is a subheap* h_1 *of* h, $(s, h_1) \models \Pi$ and $(s, h_1) \models \Sigma$. Henceforth, when necessary, we write $\models_c$ to denote the satisfiability relation under the classical semantics, and accordingly, $\models_i$ to denote the satisfiability relation under the intuitionistic semantics. The two semantics are equivalent for the satisfiability problem, but differ in the entailment problem. For instance, $\mathit{lseg}(E_1; E_2) * \mathit{lseg}(E_2; E_3) \models_c \mathit{lseg}(E_1; E_2)$ does *not* hold, while $\mathit{lseg}(E_1; E_2) * \mathit{lseg}(E_2; E_3) \models_i \mathit{lseg}(E_1; E_2)$ *does* hold.

Graph-theoretic notions. We shall work extensively upon an arc-labeled directed graph equipped with a symmetry relation. Here we collect some notations on this matter. For simplicity, we ignore the arc-labels and symmetric relations here, since they are irrelevant right now. In general, a directed graph in $\mathcal{G}$ is a pair $(\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is a finite set of nodes, $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ is a set of directed arcs. For an arc $e = (v, w) \in \mathcal{E}$, v and w are called the *source* and *destination* node of e and sometimes w is called a *successor* of v and v is called a *predecessor* of w.

A *path* in $\mathcal{G}$ is a sequence of consecutive directed arcs in $\mathcal{G}$. A node w is *reachable* from v if there is a path from v to w (note that v is reachable from v itself). In this case, we also say that v is an *ancestor* of w. A *cycle* is a path where the starting node and the ending node are equal. A path is *simple* if no nodes occur twice on the path. A cycle is *simple* if all nodes are distinct, except the ending node. For a node v and a directed arc $e = (v', w')$, e is said to be *reachable* from v if v' is reachable from v, and e is said to be *co-reachable* from v if v is reachable from w'. We use $\mathcal{R}_{\mathcal{G}}(v)$ to denote the set of *arcs* that are reachable from v.

For a graph $\mathcal{G}$, let $\mathcal{N}(\mathcal{G})$ denote the set of nodes in $\mathcal{G}$. For a set of arcs $\mathcal{E}' \subseteq \mathcal{E}$, let $\mathcal{N}(\mathcal{E}')$ denote the set of all source or destination nodes of arcs in $\mathcal{E}'$. For $\mathcal{N}' \subseteq \mathcal{N}$, *the subgraph of* $\mathcal{G}$ *induced by* $\mathcal{N}'$, denoted by $\mathcal{G}[\mathcal{N}']$, is $(\mathcal{N}', \mathcal{E} \cap (\mathcal{N}' \times \mathcal{N}'))$. On the other hand, for $\mathcal{E}' \subseteq \mathcal{E}$, *the subgraph of* $\mathcal{G}$ *induced by* $\mathcal{E}'$, denoted by $\mathcal{G}[\mathcal{E}']$, is $(\mathcal{N}(\mathcal{E}'), \mathcal{E}')$.

A connected component (CC) $\mathcal{C}$ (resp. strongly connected component, SCC, $\mathcal{S}$) of $\mathcal{G}$ is said to be *nontrivial* if $\mathcal{C}$ (resp. $\mathcal{S}$) contains at least one arc.

3 Intractability: Doubly linked list segments

Let $\varphi = \Pi \wedge \Sigma$ be an $\mathsf{SLID}_{\mathsf{SNC}}[\mathit{dllseg}]$ formula, where *dllseg* is given in Example 1. We shall show the intractability of deciding the satisfiability of $\mathsf{SLID}_{\mathsf{SNC}}[\mathit{dllseg}]$ formulae. The intractability is proved by a reduction from 3SAT. It turns out that the reduction works even for a *restricted* class of $\mathsf{SLID}_{\mathsf{SNC}}[\mathit{dllseg}]$ formulae $\Pi \wedge \Sigma$ where Π only contains inequalities and Σ only contains predicate atoms. To ease the presentation of the reduction, we will introduce a graphical representation for this restricted class of $\mathsf{SLID}_{\mathsf{SNC}}[\mathit{dllseg}]$ formulae. For an $\mathsf{SLID}_{\mathsf{SNC}}[\mathit{dllseg}]$ formula φ, recall that $\mathsf{Vars}(\varphi)$ and $\mathsf{Atoms}(\varphi)$ denote the set of variables and atoms in φ. We construct a graph $\mathcal{G}_\varphi = (\mathcal{N}, \mathcal{A}, \mathcal{E}, \mathcal{L}, R_{\neq})$ as follows.

- $\mathcal{N} = \mathsf{Vars}(\varphi)$ and $\mathcal{A} = \mathsf{Atoms}(\varphi)$.
- $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{A} \times \mathcal{N}$ is the set of arcs and $\mathcal{L}$ is the arc-labeling function, defined as: for each predicate atom $\mathfrak{a} = \mathit{dllseg}(E, P; F, L)$ in $\mathsf{Atoms}(\varphi)$, there are two arcs $e = (E, \mathfrak{a}, F) \in \mathcal{E}$ and $e' = (L, \mathfrak{a}, P)$ with $\mathcal{L}(e) = (\mathfrak{a}, +)$ and $\mathcal{L}(e') = (\mathfrak{a}, -)$ respectively.
- $R_{\neq} = \{(E, F), (F, E) \mid \Pi \models E \neq F\}$.

For convenience, we usually write $E \xrightarrow{\ell} F$ for an arc (E, F) labeled by ℓ. Clearly, $\mathcal{E}$ satisfies that, for each predicate atom $\mathfrak{a}$, there are exactly two arcs in $\mathcal{G}$, labeled by $(\mathfrak{a}, +)$ and $(\mathfrak{a}, -)$ respectively. We usually say that these two arcs are dual and have the opposite poles. All graphs for restricted $\mathsf{SLID}_{\mathsf{SNC}}[\mathit{dllseg}]$ formulae defined as above form a class of (restricted) $\mathsf{SLID}_{\mathsf{SNC}}[\mathit{dllseg}]$ graphs. From each restricted $\mathsf{SLID}_{\mathsf{SNC}}[\mathit{dllseg}]$ graph $\mathcal{G}_\varphi = (\mathcal{N}, \mathcal{A}, \mathcal{E}, \mathcal{L}, R_{\neq})$, we can recover a (restricted) $\mathsf{SLID}_{\mathsf{SNC}}[\mathit{dllseg}]$ formula $\varphi_{\mathcal{G}} = \Pi_{\mathcal{G}} \wedge \Sigma_{\mathcal{G}}$, where $\Pi_{\mathcal{G}} = \bigwedge_{(E,F) \in R_{\neq}} E \neq F$ and $\Sigma_{\mathcal{G}} = \mathop{*}_{\substack{\mathcal{L}((E,\mathfrak{a},F)) = (\mathfrak{a},+) \\ \mathfrak{a} \in \mathcal{A}, \mathcal{L}((L,\mathfrak{a},P)) = (\mathfrak{a},-)}} \mathit{dllseg}(E, P; F, L)$. Since $\mathsf{SLID}_{\mathsf{SNC}}[\mathit{dllseg}]$ formulae and $\mathsf{SLID}_{\mathsf{SNC}}[\mathit{dllseg}]$ graphs can be easily transformed to each other, we will use them interchangeably.

We are ready to present the reduction from the 3SAT Problem. Let Ψ be a CNF formula with clauses $C_1, \cdots, C_m$ over a set of variables $\{x_1, \cdots, x_n\}$. For each $j \in [m]$, C_j is a disjunction of at most three literals, that is, $C_j = l_{j,1}$, or $C_j = l_{j,1} \vee l_{j,2}$, or

$C_j = l_{j,1} \vee l_{j,2} \vee l_{j,3}$, where for each $k = 1, 2, 3$, $l_{j,k} = x_i$ or $\neg x_i$ for some $i \in [n]$. Without loss of generality, we assume that for each $i \in [n]$ and $j \in [m]$, x_i occurs at most once in C_j.

We introduce a set of atoms $\mathcal{A}_\Psi = \{\mathfrak{x}_{i,j}, \overline{\mathfrak{x}_{i,j}} \mid x_i \text{ occurs in } C_j\}$. Furthermore, for each $j \in [m]$, each literal $l_{j,k}$ in C_j ($k \in \{1, 2, 3\}$) is associated with an atom $\mathfrak{l}_{j,k}$ defined as follows: if $l_{j,k} = x_i$, then $\mathfrak{l}_{j,k} = \mathfrak{x}_{i,j}$, otherwise, $\mathfrak{l}_{j,k} = \overline{\mathfrak{x}_{i,j}}$.

We then construct $\mathsf{SLID}_{\mathsf{SNC}}[dllseg]$-graphs for variables and clauses respectively.

- For each variable x_i, assume that x_i appears in $C_{j_1}, C_{j_2}, \cdots, C_{j_{k_i}}$ (where $1 \leqslant j_1 < \cdots < j_{k_i} \leqslant m$), we have the following arcs: (1) $E_i \xrightarrow{(\mathfrak{x}_{i,j_1},+)} X_i^1$, $E_i \xrightarrow{(\overline{\mathfrak{x}_{i,j_1}},+)} Y_i^1$, and (2) $X_i^s \xrightarrow{(\mathfrak{x}_{i,j_{s+1}},+)} X_i^{s+1}$, $Y_i^s \xrightarrow{(\overline{\mathfrak{x}_{i,j_{s+1}}},+)} Y_i^{s+1}$ for $s : 1 \leqslant s < k_i$. Moreover, for convenience, we write F_i for $X_i^{k_i}$ and F_i' for $Y_i^{k_i}$.Then we set $(F_i, F_i'), (F_i', F_i) \in R_{\neq}$.
- For each clause C_j, if $C_j = l_{j,1} \vee l_{j,2} \vee l_{j,3}$, then we have arcs $G_j \xrightarrow{(\mathfrak{l}_{j,1},-)} G_j' \xrightarrow{(\mathfrak{l}_{j,2},-)} G_j'' \xrightarrow{(\mathfrak{l}_{j,3},-)} H_j$. In addition, we set $(G_j, H_j), (H_j, G_j) \in R_{\neq}$. The constructions for the cases $C_j = l_{j,1}$ and $C_j = l_{j,2} \vee l_{j,3}$ are similar.

Finally, $\mathcal{G}_\Psi$ comprises the collection of $\mathsf{SLID}_{\mathsf{SNC}}[dllseg]$-graphs for all variables x_i ($i \in [n]$) and all clauses C_j ($j \in [m]$). Roughly speaking, in $\mathcal{G}_\Psi$, the semantics of separating conjunction $*$ and the variable inequalities ensure that each Boolean variable is assigned with exactly one value from $\{\texttt{true}, \texttt{false}\}$. Moreover, the synchronisation of the two arcs with opposite poles, together with variable inequalities, guarantees that the assignment of Boolean variables satisfies all clauses in Ψ.

It is easy to see that in $\mathcal{G}_\Psi$, for each pair of atoms $\mathfrak{x}_{i,j}$ or $\overline{\mathfrak{x}_{i,j}}$, one of them appears twice, but the other occurs exactly once. To make $\mathcal{G}_\Psi$ a (restricted) $\mathsf{SLID}_{\mathsf{SNC}}[dllseg]$-graph defined above, we can simply add some additional isolated arcs. For instance, if $(\mathfrak{x}_{i,j}, +)$ occurs, but no $(\mathfrak{x}_{i,j}, -)$ in $\mathcal{G}_\Psi$ (because $\neg x_i$, but not x_i, occurs in C_j), we add an arc $G_j^\dagger \xrightarrow{(\mathfrak{x}_{i,j},-)} H_j^\dagger$, where $G_j^\dagger, H_j^\dagger$ are fresh variables.

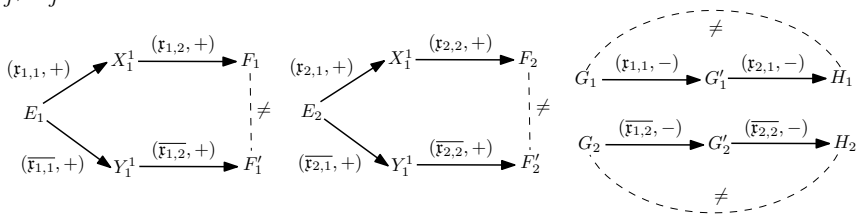

Figure 1 G_Ψ for $\Psi = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$

▸ **Example 2.** The following example illustrates the intuition of the reduction. Suppose $\Psi = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ and the graph $\mathcal{G}_\Psi$ is illustrated in Fig. 1, where dashed lines represent $R_{\neq}$. In a model (s, h) of $\mathcal{G}_\Psi$, since $(G_1, H_1) \in R_{\neq}$, at least one of $\mathfrak{x}_{1,1}, \mathfrak{x}_{2,1}$, say, $\mathfrak{x}_{1,1}$, must be assigned with a nonempty subheap. By the semantics of $*$, at least one of the two paths from E_1 to F_1 and from E_1 to F_1' must be empty. Since $(\mathfrak{x}_{1,1}, +)$ appears in the path from E_1 to F_1 and $\mathfrak{x}_{1,1}$ is nonempty, we deduce that the path from E_1 to F_1' has to be assigned with an empty subheap. Accordingly, since $(\overline{\mathfrak{x}_{1,2}}, +)$ occurs in the path from E_1 to F_1', the atom $\overline{\mathfrak{x}_{1,2}}$ is assigned with an empty subheap. In addition, from $G_2 \xrightarrow{(\overline{\mathfrak{x}_{1,2}},-)} G_2' \xrightarrow{(\overline{\mathfrak{x}_{2,2}},-)} H_2$ and $(G_2, H_2) \in R_{\neq}$, we have that $\overline{\mathfrak{x}_{2,2}}$ has to be assigned with a nonempty subheap, which, in turn, implies that the path from E_2 to F_2 is assigned with an empty subheap. Therefore, $\mathfrak{x}_{1,1}$ and $\overline{\mathfrak{x}_{2,2}}$ are assigned with a nonempty heap and they induce a satisfiable assignment θ of Ψ with $\theta(x_1) = \texttt{true}$ and $\theta(x_2) = \texttt{false}$.

The satisfiability of $\mathsf{SLID}_{\mathsf{SNC}}[dllseg]$ can be checked easily in NP. Hence, we have:

▸ **Theorem 3.** *The satisfiability problem for* $\mathsf{SLID}_{\mathsf{SNC}}[dllseg]$ *is* NP*-complete.*

4 Tractability: Beyond list segments

In this section, we show that if $P \in \mathcal{P}_1$, then $\mathsf{SLID}_{\mathsf{SNC}}[P]$ is tractable for both satisfiability and entailment. We fix a predicate $P(E; F; \boldsymbol{B}) \in \mathcal{P}_1$.

4.1 Satisfiability

For an $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formula φ, we construct a graph $\mathcal{G}_\varphi$ to represent φ. Specifically, $\mathcal{G}_\varphi$ is a tuple $(\mathsf{Vars}(\varphi), \mathsf{Atoms}(\varphi), \mathcal{N}_\varphi, \mathcal{E}_\varphi, \mathcal{L}_\varphi, R_{\neq})$ defined as follows.

- $\mathcal{N}_\varphi = \{[E] \mid E \in \mathsf{Vars}(\varphi)\}$ such that $[E]$ is the equivalence class of E under (the equivalence relation) $\sim_\Pi$ where $E' \sim_\Pi F'$ iff $\Pi \models E' = F'$.
- $\mathcal{E}_\varphi$ is the set of arcs, and $\mathcal{L}_\varphi$ is the arc-labeling function, defined as follows:
  - for each points-to atom $\mathfrak{a} = E \mapsto [(f_1, F_1), \ldots, (f_k, F_k)]$ in $\mathsf{Atoms}(\varphi)$ and each $i \in [k]$ with $f_i \in \mathsf{PFlds}(P)$, $\mathcal{E}_\varphi$ contains a *field-labeled* arc $e = ([E], [F_i])$ with $\mathcal{L}_\varphi(e) = (f_i, \mathfrak{a})$;
  - for each predicate atom $\mathfrak{a} = P(E; F; \boldsymbol{B})$ in $\mathsf{Atoms}(\varphi)$, $\mathcal{E}_\varphi$ contains a *predicate-labeled* arc $e = ([E], [F])$ with $\mathcal{L}_\varphi(e) = (P, \mathfrak{a})$;

  In both cases, $\mathfrak{a}$ is the atom associated with e, denoted by $\mathsf{atom}(e)$. In some places later on, in order to emphasise that e is from $\mathcal{G}_\varphi$, we also use $\mathsf{atom}_\varphi(e)$, instead of $\mathsf{atom}(e)$.
- $R_{\neq} = \{([E], [F]), ([F], [E]) \mid \Pi \models E \neq F\}$.

As a convention, $\bot$ represents the graph for an unsatisfiable $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formula.

All graphs for $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formulae defined as above form a class of $\mathsf{SLID}_{\mathsf{SNC}}[P]$ graphs, whose formal definition is omitted here due to the page limit. It is easy to verify that $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formulae and $\mathsf{SLID}_{\mathsf{SNC}}[P]$ graphs are equivalent, namely, they can be transformed into each other. Hence we will use them interchangeably.

▸ **Example 4.** An $\mathsf{SLID}_{\mathsf{SNC}}[th]$-graph $\mathcal{G}$ is given in Fig. 2, where each node is just one variable. Labels of the arcs are abbreviated, e.g., the label $(\texttt{left}, E_1 \mapsto [(\texttt{left}, E_2), (\texttt{right}, E_3)])$ of the arc (E_1, E_2) is abbreviated as $\texttt{left}$.

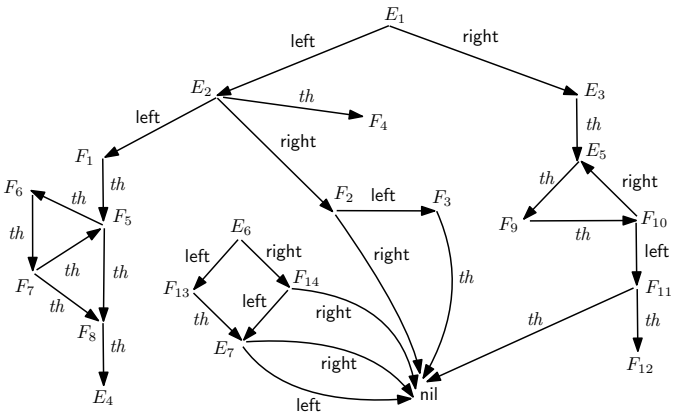

Figure 2 An example of $\mathsf{SLID}_{\mathsf{SNC}}[th]$ graphs

Our decision procedure for satisfiability follows [14] closely. The underpinning idea is, on a high level, to reduce the $\mathsf{SLID}_{\mathsf{SNC}}[P]$ graph by exploiting that the subheaps represented by different spatial atoms must be domain disjoint. The main difference to [14] will be discussed when they appear. We start with some notion of persistent set of arcs, largely from [14]. Intuitively, a set of arcs is persistent if in every model, a nonempty subheap has to be assigned to it.

▸ **Definition 5** (Persistent set of arcs [14]). Suppose $\mathcal{G} = (\mathcal{V}, \mathcal{A}, \mathcal{N}, \mathcal{E}, \mathcal{L}, R_{\neq})$ is an $\mathsf{SLID}_{\mathsf{SNC}}[P]$-graph and $\mathcal{E}' \subseteq \mathcal{E}$. Then $\mathcal{E}'$ is said to be *persistent* in $\mathcal{G}$ if either $\mathcal{E}'$ contains a field-labeled

arc, or there are two nodes $[E], [F]$ in a *connected component* of $\mathcal{G}[\mathcal{E}']$ (the subgraph of $\mathcal{G}$ induced by $\mathcal{E}'$) such that $([E], [F]) \in R_{\neq}$.

For an $\mathsf{SLID}_{\mathsf{SNC}}[P]$-graph $\mathcal{G} = (\mathcal{V}, \mathcal{A}, \mathcal{N}, \mathcal{E}, \mathcal{L}, R_{\neq})$ and $e \in \mathcal{E}$, we use $\mathcal{G} - \mathsf{atom}(e)$ to denote the graph obtained from $\mathcal{G}$ by removing all the arcs $e' \in \mathcal{E}$ such that $\mathsf{atom}(e') = \mathsf{atom}(e)$.

▸ **Definition 6** (Reduced graphs)**.** An $\mathsf{SLID}_{\mathsf{SNC}}[P]$-graph $\mathcal{G} = (\mathcal{V}, \mathcal{A}, \mathcal{N}, \mathcal{E}, \mathcal{L}, R_{\neq})$ is *reduced* if $\mathcal{G} = \bot$, or otherwise if it satisfies the following conditions:

(C1) For each $[E] \in \mathcal{N}$, $([E], [E]) \notin R_{\neq}$.

(C2) For each arc e out of $[E]$, $\mathcal{R}_{\mathcal{G}-\mathsf{atom}(e)}([E])$ is *not* persistent.

(C3) For each arc e out of $[E]$, $[\mathsf{nil}]$ is *not* reachable from $[E]$ in $\mathcal{R}_{\mathcal{G}-\mathsf{atom}(e)}([E])$.

(C4) For each pair of distinct nodes $[E], [F]$, there do not exist two predicate-labeled arc-disjoint simple paths from $[E]$ to $[F]$.

▸ Remark. We would like to remark that the arcs e in **(C2)** and **(C3)** may be field-labeled arcs. Moreover, we would like to mention the connection of the aforementioned conditions with those in [14, Table 2]: **(C2)** is adapted from conditions (i)—(iii) therein to deal with nonlinear structures, and **(C4)** is adapted from condition (iv) therein. On the other hand, **(C3)** is new to deal with the special variable nil ([14] did not consider nil).

▸ **Example 7.** The graph $\mathcal{G}$ in Fig. 2 is not reduced, because it violates **(C2)**: For the arc $e_1 = (E_2, F_4)$, $\mathcal{R}_{\mathcal{G}-\mathsf{atom}(e_1)}(E_2)$ is persistent; **(C3)**: For the arc $e_2 = (F_{11}, F_{12})$, nil is reachable from F_{11} in $\mathcal{R}_{\mathcal{G}-\mathsf{atom}(e_2)}(F_{11})$; and **(C4)**: There are two arc-disjoint predicate-labeled simple paths from F_5 to F_8. $\mathcal{G}$ can be transformed into a reduced graph, given as $\mathcal{G}_\varphi$ in Fig. 3.

As in [14], deciding whether a non-$\bot$ $\mathsf{SLID}_{\mathsf{SNC}}[P]$ graph $\mathcal{G}$ is reduced can be done in polynomial time in the size of $\mathcal{G}$. To transform an arbitrary non-$\bot$ $\mathsf{SLID}_{\mathsf{SNC}}[P]$ graph in to a reduced one, we provide a procedure which checks, for a given input non-$\bot$ $\mathsf{SLID}_{\mathsf{SNC}}[P]$ graph $\mathcal{G}$ if any condition in Def. 6 is violated. If this is the case, the algorithm performs certain operations (e.g., contract arcs, or merge nodes, or mark the graph to be $\bot$ directly) depending on which condition is violated, until $\mathcal{G}$ is reduced. Most of these are similar to [14]. However, regarding the new condition **(C3)**, if there is an arc e out of $[E]$ and $[\mathsf{nil}]$ is reachable from $[E]$ in $\mathcal{R}_{\mathcal{G}-\mathsf{atom}(e)}([E])$, then we contract the arc e in $\mathcal{G}$ if e is predicate-labeled, and mark $\mathcal{G}$ to be $\bot$ directly otherwise. The details of the procedure is omitted due to the page limit.

By Def. 6, when the graph $\mathcal{G}$ is reduced, there are only two possibilities: either $\mathcal{G} = \bot$ implying that the corresponding formula $\mathsf{SLID}_{\mathsf{SNC}}[P]$ is unsatisfiable, or $\mathcal{G} \neq \bot$ in which case one can construct a model out of $\mathcal{G}$. The similar result for linked lists is rather straightforward [14]. However, it is a bit more involved in our setting, since we need to construct a nonlinear structure to witness the satisfiability of $\mathcal{G}$.

▸ **Proposition 8.** *Given an* $\mathsf{SLID}_{\mathsf{SNC}}[P]$ *graph* $\mathcal{G}$*, one can construct, in polynomial time, a reduced* $\mathsf{SLID}_{\mathsf{SNC}}[P]$ *graph* $\mathcal{G}'$ *such that* $\mathcal{G}$ *is satisfiable iff* $\mathcal{G}' \neq \bot$*. In addition, if* $\mathcal{G}' \neq \bot$*,* $(s, h) \models \mathcal{G}$ *iff* $(s, h) \models \mathcal{G}'$ *for each state* (s, h)*.*

▸ **Theorem 9.** *Suppose* $P \in \mathcal{P}_1$*. Then the satisfiability for* $\mathsf{SLID}_{\mathsf{SNC}}[P]$ *is in polynomial time.*

4.2 Entailment

In this section, we consider the entailment problem, i.e., to decide whether $\varphi \models \psi$ for two *satisfiable* $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formulae φ, ψ such that $\mathsf{Vars}(\psi) \subseteq \mathsf{Vars}(\varphi)$. Our goal is to provide a polynomial time decision procedure for the entailment problem.

By Prop. 8, we assume that $\mathcal{G}_\varphi = (\mathsf{Vars}(\varphi), \mathsf{Atoms}(\varphi), \mathcal{N}_\varphi, \mathcal{E}_\varphi, \mathcal{L}_\varphi, R_{\neq,\varphi})$ and $\mathcal{G}_\psi = (\mathsf{Vars}(\psi), \mathsf{Atoms}(\psi), \mathcal{N}_\psi, \mathcal{E}_\psi, \mathcal{L}_\psi, R_{\neq,\psi})$ are *reduced*. For $E \in \mathsf{Vars}(\varphi)$ (resp. $E \in \mathsf{Vars}(\psi)$),

we use $[E]_\varphi$ (resp. $[E]_\psi$) to denote the node in $\mathcal{G}_\varphi$ (resp. $\mathcal{G}_\psi$) that contains E. Moreover, we assume that for each $[E]_\varphi, [F]_\varphi \in \mathcal{N}_\varphi$ (resp. $[E]_\psi, [F]_\psi \in \mathcal{N}_\psi$), if $E = F \wedge \varphi$ (resp. $E = F \wedge \psi$) is *unsatisfiable*, then $([E], [F]), ([F], [E]) \in R_{\neq,\varphi}$ (resp. $([E], [F]), ([F], [E]) \in R_{\neq,\psi}$). According to Prop. 8, these (possibly implicit) inequalities can be added to $\mathcal{G}_\varphi$ and $\mathcal{G}_\psi$ in polynomial time. Fig. 3 depicts two graphs $\mathcal{G}_\varphi$ and $\mathcal{G}_\psi$, which will be used as the running example. ($\mathcal{G}_\varphi$ is obtained from the graph $\mathcal{G}$ in Example 4.)

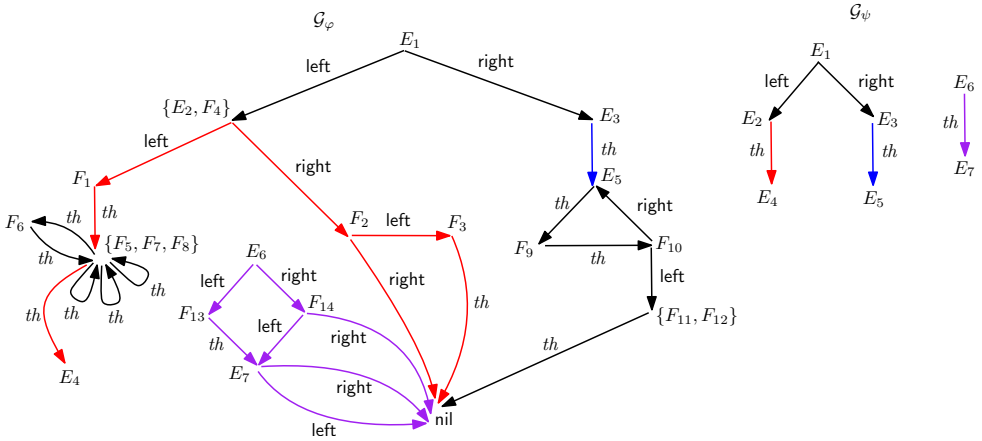

Figure 3 $\mathcal{G}_\varphi$ and $\mathcal{G}_\psi$: Running example for the entailment problem

The main idea of our decision procedure is to utilise graph representations $\mathcal{G}_\varphi$ and $\mathcal{G}_\psi$, and to extend the concept of graph homomorphisms used in [14] to nonlinear structures and the classical semantics. Let us first briefly recall the decision procedure for the entailment problem in [14]: As mentioned in the introduction, [14] focused on singly linked list segments and the intuitionistic semantics. In [14], the entailment $\varphi \models_i \psi$ is reduced to the existence of a graph homomorphism from $\mathcal{G}_\psi$ to some subgraph of $\mathcal{G}_\varphi$, that is, a mapping that assigns to each arc e from $[E]_\psi$ to $[F]_\psi$ of $\mathcal{G}_\psi$ a simple path from $[E]_\varphi$ to $[F]_\varphi$ of $\mathcal{G}_\varphi$, in addition, these simple paths are mutually arc-disjoint. The arcs that belong to these simple paths are called *covered*, and those that do not are called *uncovered*. As a result of the intuitionistic semantics, the uncovered arcs of $\mathcal{G}_\varphi$ can be ignored. The fact that the existence of a graph homomorphism from $\mathcal{G}_\psi$ to some subgraph of $\mathcal{G}_\varphi$ can be decided in polynomial time is attributed to the *unique simple path* property enjoyed by $\mathcal{G}_\varphi$, that is, for each pair of distinct nodes, say $[E]_\varphi$ and $[F]_\varphi$, there is *at most one simple path* from $[E]_\varphi$ to $[F]_\varphi$. The unique simple path property of $\mathcal{G}_\varphi$ implies that the graph homomorphism from $\mathcal{G}_\psi$ to some subgraph of $\mathcal{G}_\varphi$ is *unique* if it exists.

To deal with the entailment problem $\varphi \models_c \psi$ for $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formulae φ and ψ in this paper, we need to generalise the concept of graph homomorphisms in [14] to nonlinear structures and deal with the classical semantics as follows:

- Since the structures defined by the predicate atoms in $\mathsf{SLID}_{\mathsf{SNC}}[P]$ are nonlinear in general, a graph homomorphism should map each predicate-labeled arc e of $\mathcal{G}_\psi$ to a potentially nonlinear subgraph $\mathsf{Subgraph}_e(\mathcal{G}_\varphi)$ of $\mathcal{G}_\varphi$, instead of just a simple path. In addition, for each predicate-labeled arc e of $\mathcal{G}_\psi$, $\mathsf{Subgraph}_e(\mathcal{G}_\varphi)$ should *match* $\mathsf{atom}_\psi(e)$, that is, $\mathsf{Subgraph}_e(\mathcal{G}_\varphi) \models_c \mathsf{atom}_\psi(e)$. (Recall that we assume that an $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formula and its graph representation are interchangeable.) Actually, we will choose $\mathsf{Subgraph}_e(\mathcal{G}_\varphi)$ to be the *minimum* subgraph of $\mathcal{G}_\varphi$ that matches $\mathsf{atom}_\psi(e)$, in order to guarantee the uniqueness of the graph homomorphism.
- Because we are considering the classical semantics in this paper, those uncovered arcs of $\mathcal{G}_\varphi$ *cannot* be simply ignored. We need to guarantee that there is a way to *dispatch* the

uncovered arcs of $\mathcal{G}_\varphi$ to the predicate-labeled arcs of $\mathcal{G}_\psi$ so that for each predicate-labeled arc e of $\mathcal{G}_\psi$, $\mathsf{atom}_\psi(e)$ is matched by the graph consisting of $\mathsf{Subgraph}_e(\mathcal{G}_\varphi)$ and these uncovered arcs of $\mathcal{G}_\varphi$ which are dispatched to e.
- Finally, we need to show that checking whether a subgraph of $\mathcal{G}_\varphi$ matches $\mathsf{atom}_\psi(e)$ for a predicate-labeled arc e of $\mathcal{G}_\psi$ can be done in polynomial time.

We shall use examples to illustrate the main ideas of our decision procedure and delegate the technical details to the appendix. Let us first demonstrate, for each predicate-labeled arc e of $\mathcal{G}_\psi$, how we choose the graph $\mathsf{Subgraph}_e(\mathcal{G}_\varphi)$ to be the *minimum* subgraph of $\mathcal{G}_\varphi$ that matches $\mathsf{atom}_\psi(e)$. The following example shows that $\mathcal{G}_\varphi$ for an $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formula φ does not enjoy the unique simple path property in general.

▸ **Example 10.** Let us consider the $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formulae φ, ψ whose graph representations are illustrated in Fig. 3. The graph $\mathcal{G}_\varphi$ does not satisfy the unique simple path property: there are *two* arc-disjoint simple paths from E_6 to E_7 in $\mathcal{G}_\varphi$. Note that the subgraph of $\mathcal{G}_\varphi$ comprising all the arcs reachable from E_6 *does* match $th(E_6; E_7)$. This follows from the fact $E_6 \mapsto [(\mathtt{left}, F_{13}), (\mathtt{right}, F_{14})] * th(F_{13}; E_7) * th(F_{14}; \mathsf{nil}) \models th(E_6; E_7)$ and $F_{14} \mapsto ((\mathtt{left}, E_7), (\mathtt{right}, \mathsf{nil})) * E_7 \mapsto ((\mathtt{left}, \mathsf{nil}), (\mathtt{right}, \mathsf{nil})) \models th(F_{14}, \mathsf{nil})$.

Although the graph representations of $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formulae do not satisfy the unique simple path property in general, we can still find a way to define $\mathsf{Subgraph}_e(\mathcal{G}_\varphi)$ as the minimum subgraph of $\mathcal{G}_\varphi$ that matches $\mathsf{atom}_\psi(e)$.

▸ **Definition 11** ($\mathsf{Subgraph}_e(\mathcal{G}_\varphi)$)**.** Given a predicate-labeled arc $e = ([E]_\psi, [F]_\psi)$ in $\mathcal{G}_\psi$, $\mathsf{Subgraph}_e(\mathcal{G}_\varphi)$ is defined as follows:

- If $[E]_\varphi = [F]_\varphi$, then $\mathsf{Subgraph}_e(\mathcal{G}_\varphi)$ is the empty graph.
- Otherwise,
  - if $[F]_\varphi = [\mathsf{nil}]_\varphi$, then $\mathsf{Subgraph}_e(\mathcal{G}_\varphi) = \mathcal{G}_\varphi[\mathcal{R}_{\mathcal{G}_\varphi}([E]_\varphi)]$, that is, the subgraph of $\mathcal{G}_\varphi$ comprising all the arcs reachable from $[E]_\varphi$,
  - if $[F]_\varphi \neq [\mathsf{nil}]_\varphi$,
    - if there is a *unique* simple path from $[E]_\varphi$ to $[F]_\varphi$, denoted by π_e, then $\mathsf{Subgraph}_e(\mathcal{G}_\varphi)$ is the subgraph of $\mathcal{G}_\varphi$ comprising: (1) all the arcs in π_e, (2) all field-labeled arcs $e' = ([E']_\varphi, [F']_\varphi)$ such that $[E']_\varphi \in \mathcal{N}(\pi_e)$ and $[F']_\varphi \notin \mathcal{N}(\pi_e)$, and (3) all arcs of $\mathcal{G}_\varphi$ that are reachable from $[F']_\varphi$,
    - otherwise, $\mathsf{Subgraph}_e(\mathcal{G}_\varphi) = \mathcal{G}_\varphi[\mathcal{R}_{\mathcal{G}_\varphi}([E]_\varphi)]$.

▸ **Example 12.** Let us consider the formulae φ, ψ illustrated in Fig. 3. Let e_1, e_2, e_3 be the predicate-labeled arcs in $\mathcal{G}_\psi$, from E_2 to E_4, E_3 to E_5, and E_6 to E_7 respectively. Since $[E_2]_\varphi \neq [E_4]_\varphi$, $[E_4]_\varphi \neq [\mathsf{nil}]_\varphi$, and there is a unique path from $[E_2]_\varphi$ to $[E_4]_\varphi$, according to Definition 11, $\mathsf{Subgraph}_{e_1}(\mathcal{G}_\varphi)$ comprises the unique simple path from $[E_2]_\varphi$ to $[E_4]_\varphi$, the arc from $[E_2]_\varphi$ to $[F_2]_\varphi$, and all the arcs reachable from $[F_2]_\varphi$. Similarly, $\mathsf{Subgraph}_{e_2}(\mathcal{G}_\varphi)$ comprises the unique simple path from $[E_3]_\varphi$ to $[E_5]_\varphi$ in $\mathcal{G}_\varphi$, which is just the predicate-labeled arc from $[E_3]_\varphi$ to $[E_5]_\varphi$. Since $[E_6]_\varphi \neq [E_7]_\varphi$, $[E_7]_\varphi \neq [\mathsf{nil}]_\varphi$, and there are two distinct simple paths from $[E_6]_\varphi$ to $[E_7]_\varphi$, we have that $\mathsf{Subgraph}_{e_3}(\mathcal{G}_\varphi) = \mathcal{G}_\varphi[\mathcal{R}_{\mathcal{G}_\varphi}([E_6]_\varphi)]$, that is, the subgraph of $\mathcal{G}_\varphi$ comprising all the arcs reachable from $[E_6]_\varphi$.

Example 13 illustrates the difference between the intuitionistic and classical semantics.

▸ **Example 13.** Let $\varphi = lseg(E_1; E_2) * lseg(E_2; E_3) * lseg(E_3; E_2) * lseg(E_3; E_4)$, $\psi = lseg(E_1; E_4)$, and $\mathcal{G}_\varphi$ and $\mathcal{G}_\psi$ are depicted in Fig. 4. (For simplicity, arc label $(lseg, lseg(E_1; E_2))$ is abbreviated as $lseg$, *sic passim.*) We claim $\varphi \not\models \psi$. To see this, let (s, h) be the state such that $s(E_1) = 1$, $s(E_2) = 2$, $s(E_3) = s(E_4) = 3$, $h(1, \mathtt{next}) = 2$, $h(2, \mathtt{next}) = 3$, $h(3, \mathtt{next}) = 2$.

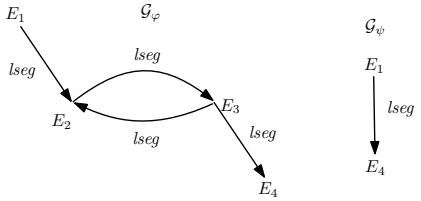

Figure 4 Graph $\mathcal{G}_\varphi$ and $\mathcal{G}_\psi$

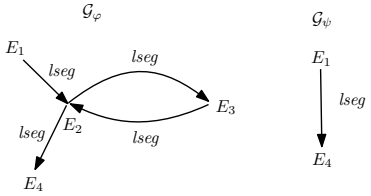

Figure 5 Graph $\mathcal{G}_\varphi$ and $\mathcal{G}_\psi$

Then $(s,h) \models \varphi$, but $(s,h) \not\models \psi$ under the classical semantics, although the subheap h' of h with $\mathsf{ldom}(h') = \{1,2\}$ satisfies ψ. This is in contrast to the intuitionistic semantics, under which $h \models \psi$ follows from $h' \models \psi$.

Regarding the homomorphism [14], although the unique simple path from E_1 to E_4 in $\mathcal{G}_\varphi$ matches $lseg(E_1; E_4)$, the arc (E_3, E_2) in $\mathcal{G}_\varphi$ is uncovered. This is the issue of using graph homomorphism to entailment under the classical semantics, as also pointed out in [14]. On the other hand, let us assume another pair of graphs in Fig. 5. It is not hard to see that the entailment $\varphi \models \psi$ holds. Interestingly, in this case, *all* the arcs in the cycle $E_2 \leftrightarrows E_3$ in $\mathcal{G}_\varphi$ are uncovered. ◂

As suggested in Example 13, to deal with the classical semantics, it is necessary to put some proper constraints on the uncovered arcs of nontrivial SCCs in $\mathcal{G}_\varphi$. Let us introduce some additional notations for $\mathcal{G}_\varphi$. An arc e' in $\mathcal{G}_\varphi$ is *covered* if

- either e' is an arc labeled by $(f, -)$ from $[E]_\varphi$ to $[F]_\varphi$ and there is an arc e labeled by $(f, -)$ from $[E]_\psi$ to $[F]_\psi$ in $\mathcal{G}_\psi$, where $E, F \in \mathsf{Vars}(\psi)$,
- or e' belongs to $\mathsf{Subgraph}_e(\mathcal{G}_\varphi)$ for some predicate-labeled arc e in $\mathcal{G}_\psi$.

A simple cycle C of $\mathcal{G}_\varphi$ is *covered* if *all* arcs in C are covered; C is *uncovered* if *none of the arcs* in C are covered. A nontrivial SCC $\mathcal{S}$ is said to be *final w.r.t. covered arcs* if there are *no* covered arcs that are reachable from the nodes in $\mathcal{S}$, moreover, one of the following holds: 1) there is a covered arc whose destination node belongs to $\mathcal{S}$, 2) there is a predicate-labeled arc e from $[E]_\psi$ to $[F]_\psi$ in $\mathcal{G}_\psi$ such that $[E]_\varphi = [F]_\varphi$ is in $\mathcal{S}$. A collection of simple cycles $\mathcal{S} = \{C_1, \ldots, C_n\}$ is said to be *unentangled* if C_i and C_j $(i \neq j)$ do *not* share any arc, and share at most one node.

In the sequel, we first analyse the structure of nontrivial SCCs in $\mathcal{G}_\varphi$.

▸ **Proposition 14.** *Suppose φ, ψ are two $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formulae such that $\varphi \models_c \psi$. Then the nontrivial SCCs of $\mathcal{G}_\varphi$ satisfy the following structural constraints.*

- *If a nontrivial SCC $\mathcal{S}$ of $\mathcal{G}_\varphi$ contains only* predicate-labeled *arcs, then $\mathcal{S}$ comprises a collection of* unentangled *simple cycles $\{C_1, \ldots, C_n\}$.*
- *If a nontrivial SCC $\mathcal{S}$ of $\mathcal{G}_\varphi$ contains both an uncovered arc and a field-labeled arc, then $\mathcal{S}$ is final w.r.t. covered arcs and is exactly a simple cycle.*

A graph homomorphism from $\mathcal{G}_\psi$ to $\mathcal{G}_\varphi$ is required to satisfy that *each simple cycle C of $\mathcal{G}_\varphi$ is either covered or uncovered.* In addition, the uncovered simple cycles of $\mathcal{G}_\varphi$ should satisfy some additional constraint given below.

▸ **Definition 15.** Let $\mathcal{S} = \{C_1, \cdots, C_n\}$ be a nontrivial SCC of $\mathcal{G}_\varphi$ with unentangled simple cycles, where all C_i's are either covered or uncovered, with at least one uncovered C_i.

- An *uncovered connected component (UCC)* $\mathcal{C}$ of $\mathcal{S}$ is a connected component of the graph obtained from $\mathcal{S}$ by removing all covered simple cycles.
- Suppose $\mathcal{S}$ is not final w.r.t. covered arcs. Then $\mathcal{S}$ is *indirectly covered* if for each UCC $\mathcal{C}$ of $\mathcal{S}$ and each simple cycle C in $\mathcal{C}$, and there is an arc $e = ([E]_\psi, [F]_\psi)$ in $\mathcal{G}_\psi$ with $\mathsf{atom}_\psi(e) = P(E; F; \boldsymbol{B})$, satisfying one of the following constraints:

(1) $[E]_\varphi = [F]_\varphi \in \mathcal{C}$, and for each arc e' in C with $\mathsf{atom}_\varphi(e') = P(E'; F'; \boldsymbol{B}')$, it holds that $\boldsymbol{B} \sim_\varphi \boldsymbol{B}'$ (where $\sim_\varphi$ is extended to vectors of variables in an obvious way),

(2) $[E]_\varphi \neq [F]_\varphi$, there is a node belonging to both $\mathsf{Subgraph}_e(\mathcal{G}_\varphi)$ and $\mathcal{C}$, and for each arc e' in C with $\mathsf{atom}_\varphi(e') = P(E'; F'; \boldsymbol{B}')$, it holds that $\boldsymbol{B} \sim_\varphi \boldsymbol{B}'$.

Note that if the predicate P contains no static parameters, then the condition is simplified as follows: for each UCC $\mathcal{C}$ of $\mathcal{S}$, there is an arc $e = ([E]_\psi, [F]_\psi)$ in $\mathcal{G}_\psi$ such that either $[E]_\varphi = [F]_\varphi \in \mathcal{C}$ or there is a node belonging to both $\mathsf{Subgraph}_e(\mathcal{G}_\varphi)$ and $\mathcal{C}$.

- Suppose $\mathcal{S}$ is final w.r.t. covered arcs. Then $\mathcal{S}$ is *indirectly covered* if one of the following holds: 1) there is a covered arc e' in $\mathcal{G}_\varphi$ with $\mathsf{atom}_\varphi(e') = P(E; F; \boldsymbol{B})$ such that $[F]_\varphi$ is in $\mathcal{S}$ and $\mathcal{G}_\varphi[\mathcal{R}_{\mathcal{G}_\varphi}([F]_\varphi)]$ matches $P(F; F; \boldsymbol{B})$, 2) there is an arc e in $\mathcal{G}_\psi$ with $\mathsf{atom}_\psi(e) = P(E; F; \boldsymbol{B})$ such that $[E]_\varphi = [F]_\varphi$ is in $\mathcal{S}$ and $\mathcal{G}_\varphi[\mathcal{R}_{\mathcal{G}_\varphi}([F]_\varphi)]$ matches $P(F; F; \boldsymbol{B})$.

▸ **Definition 16** (Criteria of coverage of arcs)**.** A graph homomorphism from $\mathcal{G}_\psi$ to $\mathcal{G}_\varphi$ is *fully covered*, if

(I) for each nontrivial SCC $\mathcal{S}$ that is *not* final w.r.t. covered arcs and contains an uncovered arc (which implies that $\mathcal{S}$ contains predicate-labeled arcs only by Prop. 14), each simple cycle C in $\mathcal{S}$ is either covered or uncovered, and each UCC of $\mathcal{S}$ is indirectly covered,

(II) for each nontrivial SCC $\mathcal{S}$ that is final w.r.t. covered arcs, $\mathcal{S}$ is indirectly covered,

(III) each uncovered arc of $\mathcal{G}_\varphi$ either belongs to some nontrivial SCC or is reachable from a node in some nontrivial SCC that is final w.r.t. covered arcs.

We now collate all the ingredients together and specify the formal definition of the graph homomorphism from $\mathcal{G}_\psi$ to $\mathcal{G}_\varphi$.

▸ **Definition 17** (Graph homomorphism from $\mathcal{G}_\psi$ to $\mathcal{G}_\varphi$)**.** A *homomorphism* from $\mathcal{G}_\psi$ to $\mathcal{G}_\varphi$ is a function $\theta : \mathcal{N}_\psi \to \mathcal{N}_\varphi$ satisfying the following constraints.

- **Variable subsumption**: For each $E \in \mathsf{Vars}(\psi)$, $[E]_\psi \subseteq \theta([E]_\psi)$. In particular, we have $E \in \theta([E]_\psi)$. This implies that $\theta([E]_\psi)$ has to be $[E]_\varphi$, and thus, for convenience, we sometime use $[E]_\varphi$ to denote $\theta([E]_\psi)$.
- **Field-labeled arc**: For each field-labeled arc $e = ([E]_\psi, [F]_\psi)$ in $\mathcal{G}_\psi$ with $\mathcal{L}_\psi(e) = (f, E \mapsto \rho)$, there is a field-labeled arc $e' = ([E]_\varphi, [F]_\varphi)$ in $\mathcal{G}_\varphi$ with $\mathcal{L}_\varphi(e') = (f, E' \mapsto \rho')$, and for each (f', F_1) in ρ and (f', F_1') in ρ', $\theta([F_1]_\psi) = [F_1']_\varphi$.
- **Predicate-labeled arc**: For each predicate-labeled arc $e = ([E]_\psi, [F]_\psi)$ in $\mathcal{G}_\psi$ with $\mathsf{atom}_\psi(e) = P(E; F; \boldsymbol{B})$ and $[E]_\varphi \neq [F]_\varphi$, $\mathsf{Subgraph}_e(\mathcal{G}_\varphi)$ matches $P(E; F; \boldsymbol{B})$.
- **Inequality**: For each $([E]_\psi, [F]_\psi) \in R_{\neq,\psi}$, $([E]_\varphi, [F]_\varphi) \in R_{\neq,\varphi}$.
- **Coverage**: θ is fully covered (cf. Definition 16).
- **Separation constraint**:
  - For each pair of distinct arcs e_1, e_2 in $\mathcal{G}_\psi$, the two subgraphs $\mathsf{Subgraph}_{e_1}(\mathcal{G}_\varphi)$ and $\mathsf{Subgraph}_{e_2}(\mathcal{G}_\varphi)$ are arc-disjoint.
  - For every two distinct nontrivial SCCs $\mathcal{S}_1, \mathcal{S}_2$ of $\mathcal{G}_\varphi$ that are final w.r.t. covered arcs, the sets of arcs that are reachable from (nodes in) $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively are *disjoint*.

▸ **Proposition 18.** *$\varphi \models_c \psi$ iff there is a graph homomorphism from $\mathcal{G}_\psi$ to $\mathcal{G}_\varphi$.*

▸ **Example 19.** Consider the graphs $\mathcal{G}_\varphi, \mathcal{G}_\psi$ in Fig. 3. The function $\theta : \mathcal{N}_\psi \to \mathcal{N}_\varphi$ can be defined obviously, for instance $\theta(E_2) = \{E_2, F_4\}$. For the predicate-labeled arc $e = (E_2, E_4)$ in $\mathcal{G}_\psi$, it is a routine to check that $\mathsf{Subgraph}_e(\mathcal{G}_\varphi)$ matches e. Similarly for the other predicate-labeled arcs in $\mathcal{G}_\psi$. In addition, the homomorphism θ is *fully covered*. Let $\mathcal{S}_1, \mathcal{S}_2$ be the uncovered nontrivial SCCs in $\mathcal{G}_\varphi$, which contain the node $\{F_5, F_7, F_8\}$ and E_5 respectively.

$\mathcal{S}_1$ is not final w.r.t. covered arcs and contains only one uncovered connected component (i.e., itself). Since th contains no static parameters and the arc from F_1 to $\{F_5, F_7, F_8\}$ is covered, according to Definition 15, we know that $\mathcal{S}_1$ is indirectly covered. Moreover, $\mathcal{S}_2$ is final w.r.t. covered arcs. Because the arc from E_3 to E_5 is covered and $\mathcal{G}_\varphi[\mathcal{R}_{\mathcal{G}_\varphi}(E_5)]$, which comprises all the arcs reachable from E_5 in $\mathcal{G}_\varphi$, matches $th(E_5; E_5)$, we deduce that $\mathcal{S}_2$ is indirectly covered as well. Finally, it is easy to see that the separation constraint is satisfied. Therefore, θ is a graph homomorphism, and we conclude that $\varphi \models_c \psi$.

Finally, in order to show that the existence of a graph homomorphism from $\mathcal{G}_\varphi$ to $\mathcal{G}_\psi$ can be decided in polynomial time, it remains to show that checking that a subgraph of $\mathcal{G}_\varphi$ matches a predicate atom $P(E; F; \boldsymbol{B})$ can be decided in polynomial time.

▸ **Proposition 20.** *Given a subgraph $\mathcal{G}$ of $\mathcal{G}_\varphi$ and a predicate atom $P(E; F; \boldsymbol{B})$, one can decide whether $\mathcal{G}$ matches $P(E; F; \boldsymbol{B})$ in polynomial time.*

The proof of Prop. 20 relies on the fact that the problem of whether $\mathcal{G}$ matches $P(E; F; \boldsymbol{B})$ can be reduced to checking some structural properties of $\mathcal{G}$, which can be done in polynomial time.

Combining Prop. 18 and Prop. 20, we conclude:

▸ **Theorem 21.** *Let $P \in \mathcal{P}_1$. The entailment problem for $\mathsf{SLID}_{\mathsf{SNC}}[P]$ is in polynomial time.*

A detailed comparison with [16].

Our polynomial-time decision procedure for $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formulae with $P \in \mathcal{P}_1$ is related to that in [16], since the parameters of an inductive predicate in [16] are also divided as source, destination, and static parameters. Nevertheless, our decision procedure differs from that in [16] in the following three aspects. (1) $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formulae with $P \in \mathcal{P}_1$ and the fragment of SL in [16] are incomparable: The latter allows nesting inductive predicates so that nested lists and skip lists can be defined, while our fragment disallows this. On the other hand, $\mathsf{SLID}_{\mathsf{SNC}}[P]$ formulae with $P \in \mathcal{P}_1$ allows defining tree structures, which are not supported in [16]. (2) The inequality $E \neq F$ may appear in the inductive rules of predicates in [16], but is excluded here. As a result, the fragment in [16] is *precise* in the sense that when checking $\varphi \models \psi$, for each arc e in $\mathcal{G}_\psi$, there is a *unique* subgraph of $\mathcal{G}_\varphi$ that matches $\mathsf{atom}_\psi(e)$. On the other hand, $\mathsf{SLID}_{\mathsf{SNC}}[P]$ is *not* precise because of the uncovered simple cycles of $\mathcal{G}_\varphi$. Some proper constraints on these uncovered simple cycles are necessary in the graph homomorphism to achieve a *complete* decision procedure for the entailment problem. (3) In [16], checking whether a subgraph of $\mathcal{G}_\varphi$ matches a predicate atom $P(E; F; \boldsymbol{B})$ is reduced to the membership problem of a tree automaton constructed from the inductive definitions of P, which is of exponential size in the worst case, due to the nested inductive predicates. However, in our decision procedure, we simply check whether the subgraph satisfies some structural properties in polynomial time.

5 Conclusion

We have provided polynomial-time satisfiability and entailment checking algorithms for $\mathsf{SLID}_{\mathsf{SNC}}$ with one source (destination) parameter. We have also shown that the satisfiability problem is generally NP-complete if more than one source (destination) parameter is allowed, particularly for an inductive predicate of doubly linked list segments.

For the future work, we strongly believe that our polynomial-time decision procedures can be further generalised to handle inductive predicates with multiple source (destination) parameters in a constrained form. Moreover, adding data constraints certainly is one of the promising extensions of the current work.

References

1 Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max I. Kanovich, and Joël Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In *FoSSaCS 2014*, pages 411–425, 2014.

2 Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. A decidable fragment of separation logic. In *FSTTCS 2004*, pages 97–109, 2004.

3 Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Symbolic execution with separation logic. In *APLAS 2005*, pages 52–68, 2005.

4 Rémi Brochenin, Stéphane Demri, and Étienne Lozes. On the almighty wand. *Inf. Comput.*, 211:106–137, 2012.

5 James Brotherston, Dino Distefano, and Rasmus Lerchedahl Petersen. Automated cyclic entailment proofs in separation logic. In *CADE 2011*, pages 131–146, 2011.

6 James Brotherston, Carsten Fuhs, Juan A. Navarro Perez, and Nikos Gorogiannis. A decision procedure for satisfiability in separation logic with inductive predicates. In *LICS 2014*, pages 25:1–25:10, 2014.

7 James Brotherston, Nikos Gorogiannis, and Max I. Kanovich. Biabduction (and related problems) in array separation logic. *CoRR*, abs/1607.01993, 2016. URL: `http://arxiv.org/abs/1607.01993`.

8 James Brotherston, Nikos Gorogiannis, Max I. Kanovich, and Reuben Rowe. Model checking for symbolic-heap separation logic with inductive predicates. In *POPL 2016*, pages 84–96, 2016.

9 Cristiano Calcagno and Dino Distefano. Infer: An automatic program verifier for memory safety of C programs. In *NFM 2011*, pages 459–465, 2011.

10 Cristiano Calcagno, Dino Distefano, Jérémy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter W. O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. Moving fast with software verification. In *NFM 2015*, pages 3–11, 2015.

11 Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *FSTTCS 2001*, pages 108–119, 2001.

12 Wei-Ngan Chin, Cristina David, Huu Hai Nguyen, and Shengchao Qin. Automated verification of shape, size and bag properties via user-defined predicates in separation logic. *Sci. Comput. Program.*, 77(9):1006–1036, 2012. URL: `http://dx.doi.org/10.1016/j.scico.2010.07.004`, `doi:10.1016/j.scico.2010.07.004`.

13 Duc-Hiep Chu, Joxan Jaffar, and Minh-Thai Trinh. Automatic induction proofs of data-structures in imperative programs. In *PLDI 2015*, pages 457–466, 2015.

14 Byron Cook, Christoph Haase, Joël Ouaknine, Matthew Parkinson, and James Worrell. Tractable reasoning in a fragment of separation logic. In *CONCUR 2011*, pages 235–249, 2011.

15 Stéphane Demri and Morgan Deters. Expressive completeness of separation logic with two variables and no separating conjunction. *ACM Trans. Comput. Log.*, 17(2):12, 2016.

16 Constantin Enea, Ondřej Lengál, Mihaela Sighireanu, and Tomáš Vojnar. Compositional entailment checking for a fragment of separation logic. In *APLAS 2014*, pages 314–333, 2014.

17 Constantin Enea, Mihaela Sighireanu, and Zhilin Wu. On automated lemma generation for separation logic with inductive definitions. In *ATVA 2015*, pages 80–96, 2015.

18 Xincai Gu, Taolue Chen, and Zhilin Wu. A complete decision procedure for linearly compositional separation logic with data constraints. In *IJCAR 2016*, pages 532–549, 2016.

19 Christoph Haase, Samin Ishtiaq, Joël Ouaknine, and Matthew J. Parkinson. Seloger: A tool for graph-based reasoning in separation logic. In *CAV 2013*, pages 790–795, 2013.

20 Radu Iosif, Adam Rogalewicz, and Jiri Simacek. The tree width of separation logic with recursive definitions. In *CADE 2013*, pages 21–38, 2013.

21 Radu Iosif, Adam Rogalewicz, and Tomás Vojnar. Deciding entailments in inductive separation logic with tree automata. In *ATVA 2014*, pages 201–218, 2014.

22 Quang Loc Le, Jun Sun, and Wei-Ngan Chin. Satisfiability modulo heap-based programs. In *CAV 2016*, pages 382–404, 2016.

23 Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *CSL 2001*, pages 1–19, 2001.

24 Andrew Reynolds, Radu Iosif, Cristina Serban, and Tim King. A decision procedure for separation logic in SMT. In *ATVA 2016*, pages 244–261, 2016.

25 John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS 2002*, pages 55–74, 2002.

26 Makoto Tatsuta, Quang Loc Le, and Wei-Ngan Chin. Decision procedure for separation logic with inductive definitions and presburger arithmetic. In *APLAS 2016*, pages 423–443, 2016.

27 Zhaowei Xu, Taolue Chen, and Zhilin Wu. Satisfiability of compositional separation logic with tree predicates and data constraints. Technical report, State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, 2017.