Decision Problems in a Logic for Reasoning About Reconfigurable Distributed Systems

Marius Bozga(✉), Lucas Bueri, and Radu Iosif

Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000 Saint-Martin-d'Hères, France
marius.bozga@univ-grenoble-alpes.fr

Abstract. We consider a logic used to describe sets of configurations of distributed systems, whose network topologies can be changed at runtime, by reconfiguration programs. The logic uses inductive definitions to describe networks with an unbounded number of components and interactions, written using a multiplicative conjunction, reminiscent of Bunched Implications [37] and Separation Logic [39]. We study the complexity of the satisfiability and entailment problems for the configuration logic under consideration. Additionally, we consider the robustness property of degree boundedness (is every component involved in a bounded number of interactions?), an ingredient for decidability of entailments.

1 Introduction

Distributed systems are increasingly used as critical parts of the infrastructure of our digital society, as in e.g., datacenters, e-banking and social networking. In order to address maintenance (e.g., replacement of faulty and obsolete network nodes by new ones) and data traffic issues (e.g., managing the traffic inside a datacenter [35]), the distributed systems community has recently put massive effort in designing algorithms for *reconfigurable systems*, whose network topologies change at runtime [23]. However, dynamic reconfiguration in the form of software or network upgrades has been recognized as one of the most important sources of cloud service outage [25].

This paper contributes to a logical framework that addresses the timely problems of formal *modeling* and *verification* of reconfigurable distributed systems. The basic building blocks of this framework are (i) a Hoare-style program proof calculus [1] used to write formal proofs of correctness of reconfiguration programs, and (ii) an invariant synthesis method [6] that proves the safety (i.e., absence of reachable error configurations) of the configurations defined by the assertions that annotate a reconfiguration program. These methods are combined to prove that an initially correct distributed system cannot reach an error state, following the execution of a given reconfiguration sequence.

The assertions of the proof calculus are written in a logic that defines infinite sets of configurations, consisting of *components* (i.e., processes running on different nodes of the network) connected by *interactions* (i.e., multi-party channels alongside which messages between components are transfered). Systems that share the same architectural style (e.g., pipeline, ring, star, tree, etc.) and differ by the number of components and interactions are described using inductively defined predicates. Such configurations can be modified either by (a) adding or removing components and interactions (reconfiguration), or (b) changing the local states of components, by firing interactions.

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 691–711, 2022.
https://doi.org/10.1007/978-3-031-10769-6_40

The assertion logic views components and interactions as *resources*, that can be created or deleted, in the spirit of resource logics *à la* Bunched Implications [37], or Separation Logic [39]. The main advantage of using resource logics is their support for *local reasoning* [12]: reconfiguration actions are specified by pre- and postconditions mentioning only the resources involved, while framing out the rest of the configuration.

The price to pay for this expressive power is the difficulty of automating the reasoning in these logics. This paper makes several contributions in the direction of proof automation, by studying the complexity of the *satisfiability* and *entailment* problems, for the configuration logic under consideration. Additionally, we study the complexity of a robustness property [27], namely *degree boundedness* (is every component involved in a bounded number of interactions?). In particular, the latter problem is used as a prerequisite for defining a fragment with a decidable entailment problem. For space reasons, the proofs of the technical results are given in [5].

1.1 Motivating Example

The logic studied in this paper is motivated by the need for an assertion language that supports reasoning about dynamic reconfigurations in a distributed system. For instance, consider a distributed system consisting of a finite (but unknown) number of *components* (processes) placed in a ring, executing the same finite-state program and communicating via *interactions* that connect the *out* port of a component to the *in* port of its right neighbour, in a round-robin fashion, as in Fig. 1(a). The behavior of a component is a machine with two states, T and H, denoting whether the component has a token (T) or not (H). A component c_i without a token may receive one, by executing a transition $\mathsf{H} \xrightarrow{in} \mathsf{T}$, simultaneously with its left neighbour c_j, that executes the transition $\mathsf{T} \xrightarrow{out} \mathsf{H}$. Then, we say that the interaction (c_j, out, c_i, in) has fired, moving a token one position to the right in the ring. Note that there can be more than one token, moving independently in the system, as long as no token overtakes another token.

The token ring system is formally specified by the following inductive rules:

$$\mathsf{ring}_{h,t}(x) \leftarrow \exists y \exists z \,.\, [x]@q * \langle x.out, z.in \rangle * \mathsf{chain}_{h',t'}(z,y) * \langle y.out, x.in \rangle$$

$$\mathsf{chain}_{h,t}(x,y) \leftarrow \exists z.\, [x]@q * \langle x.out, z.in \rangle * \mathsf{chain}_{h',t'}(z,y)$$

$$\mathsf{chain}_{0,1}(x,x) \leftarrow [x]@\mathsf{T} \qquad \mathsf{chain}_{1,0}(x,x) \leftarrow [x]@\mathsf{H} \qquad \mathsf{chain}_{0,0}(x,x) \leftarrow [x]$$

$$\text{where } h' \stackrel{\text{def}}{=} \begin{cases} \max(h-1,0) & \text{, if } q = \mathsf{H} \\ h & \text{, if } q = \mathsf{T} \end{cases} \text{ and } t' \stackrel{\text{def}}{=} \begin{cases} \max(t-1,0) & \text{, if } q = \mathsf{T} \\ t & \text{, if } q = \mathsf{H} \end{cases}$$

The predicate $\mathsf{ring}_{h,t}(x)$ describes a ring with at least two components, such that at least h (resp. t) components are in state H (resp. T). The ring consists of a component x in state q, described by the formula $[x]@q$, an interaction from the *out* port of x to the *in* port of another component z, described as $\langle x.out, z.in \rangle$, a separate chain of components stretching from z to y ($\mathsf{chain}_{h',t'}(z,y)$), and an interaction connecting the *out* port of component y to the *in* port of component x ($\langle y.out, x.in \rangle$). Inductively, a chain consists of a component $[x]@q$, an interaction $\langle x.out, z.in \rangle$ and a separate $\mathsf{chain}_{h',t'}(z,y)$. Figure 1(b) depicts the unfolding of the inductive definition of the token ring, with the

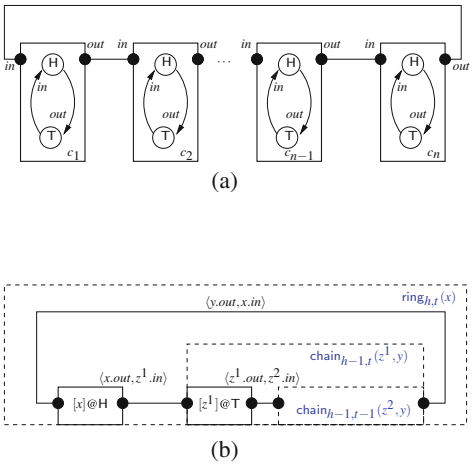

Fig. 1. Inductive Specification and Reconfiguration of a Token Ring

existentially quantified variables z from the above rules α-renamed to $z^1, z^2, \ldots$ to avoid confusion.

A *reconfiguration program* takes as input a mapping of program variables to components and executes a sequence of *basic operations* i.e., component/interaction creation/deletion, involving the components and interactions denoted by these variables. For instance, the reconfiguration program in Fig. 1(c) takes as input three adjacent components, mapped to the variables x, y and z, respectively, removes the component y together with its left and right interactions and reconnects x directly with z. Programming reconfigurations is error-prone, because the interleaving between reconfiguration actions and interactions in a distributed system may lead to bugs that are hard to trace. For instance, if a reconfiguration program removes the last component in state T (resp. H) from the system, no token transfer interaction may fire and the system deadlocks.

We prove absence of such errors using a Hoare-style proof system [1], based on the logic introduced above as assertion language. For instance, the proof from Fig. 1(c) shows that the reconfiguration sequence applied to a component y in state H (i.e., $[y]@\mathsf{H}$) in a ring with at least $h \geq 2$ components in state H and at least $t \geq 1$ components in state T leads to a ring with at least $h-1$ components in state H and at least t in state T; note that the states of the components may change during the execution of the reconfiguration program, as tokens are moved by interactions.

The proof in Fig. 1(c) uses *local axioms* specifying, for each basic operation, only those components and interactions required to avoid faulting, with a *frame rule* $\{\phi\}\ \mathsf{P}\ \{\psi\} \Rightarrow \{\phi * \boxed{F}\}\ \mathsf{P}\ \{\psi * F\}$; for readability, the frame formulæ (from the preconditions of the conclusion of the frame rule applications) are enclosed in boxes.

The proof also uses the *consequence rule* $\{\phi\}\ \mathsf{P}\ \{\psi\} \Rightarrow \{\phi'\}\ \mathsf{P}\ \{\psi'\}$ that applies if ϕ' is stronger than ϕ and ψ' is weaker than ψ. The side conditions of the consequence rule require checking the validity of the entailments $\mathsf{ring}_{h,t}(y) \models \exists x \exists z\ .\ \langle x.out, y.in\rangle * [y]@\mathsf{H} * \langle y.out, z.in\rangle * \mathsf{chain}_{h-1,t}(z,x)$ and $\mathsf{chain}_{h-1,t}(z,x) * \langle x.out, z.in\rangle \models \mathsf{ring}_{h-1,t}(z)$,

for all $h \geq 2$ and $t \geq 1$. These side conditions can be automatically discharged using the results on the decidability of entailments given in this paper. Additionally, checking the satisfiability of a precondition is used to detect trivially valid Hoare triples.

1.2 Related Work

Formal modeling coordinating architectures of component-based systems has received lots of attention, with the development of architecture description languages (ADL), such as BIP [3] or REO [2]. Many such ADLs have extensions that describe programmed reconfiguration, e.g., [19,30], classified according to the underlying formalism used to define their operational semantics: *process algebras* [13,33], *graph rewriting* [32,41,44], *chemical reactions* [43] (see the surveys [7,11]). Unfortunately, only few ADLs support formal verification, mainly in the flavour of runtime verification [10,17,20,31] or finite-state model checking [14].

Parameterized verification of unbounded networks of distributed processes uses mostly hard-coded coordinating architectures (see [4] for a survey). A first attempt at specifying architectures by logic is the *interaction logic* of Konnov et al. [29], a combination of Presburger arithmetic with monadic uninterpreted function symbols, that can describe cliques, stars and rings. More structured architectures (pipelines and trees) can be described using a second-order extension [34]. However, these interaction logics are undecidable and lack support for automated reasoning.

Specifying parameterized component-based systems by inductive definitions is not new. *Network grammars* [26,32,40] use context-free grammar rules to describe systems with linear (pipeline, token-ring) architectures obtained by composition of an unbounded number of processes. In contrast, we use predicates of unrestricted arities to describe architectural styles that are, in general, more complex than trees. Moreover, we write inductive definitions using a resource logic, suitable also for writing Hoare logic proofs of reconfiguration programs, based on local reasoning [12].

Local reasoning about concurrent programs has been traditionally the focus of Concurrent Separation Logic (CSL), based on a parallel composition rule [36], initially with a non-interfering (race-free) semantics [8] and later combining ideas of assume-and rely-guarantee [28,38] with local reasoning [22,42] and abstract notions of framing [15,16,21]. However, the body of work on CSL deals almost entirely with shared-memory multithreading programs, instead of distributed systems, which is the aim of our work. In contrast, we develop a resource logic in which the processes do not just share and own resources, but become mutable resources themselves.

The techniques developed in this paper are inspired by existing techniques for similar problems in the context of Separation Logic (SL) [39]. For instance, we use an abstract domain similar to the one defined by Brotherston et al. [9] for checking satisfiability of symbolic heaps in SL and reduce a fragment of the entailment problem in our logic to SL entailment [18]. In particular, the use of existing automated reasoning techniques for SL has pointed out several differences between the expressiveness of our logic and that of SL. First, the configuration logic describes hypergraph structures, in which edges are ℓ-tuples for $\ell \geq 2$, instead of directed graphs as in SL, where ℓ is a parameter of the problem: considering ℓ to be a constant strictly decreases the complexity of the problem. Second, the degree (number of hyperedges containing a given

vertex) is unbounded, unlike in SL, where the degree of heaps is constant. Therefore, we dedicate an entire section (Sect. 4) to the problem of deciding the existence of a bound (and computing a cut-off) on the degree of the models of a formula, used as a prerequisite for the encoding of the entailment problems from the configuration logic as SL entailments.

2 Definitions

We denote by $\mathbb{N}$ the set of positive integers including zero. For a set A, we define $A^1 \stackrel{\text{def}}{=} A$, $A^{i+1} \stackrel{\text{def}}{=} A^i \times A$, for all $i \geq 1$, and $A^+ = \bigcup_{i \geq 1} A^i$, where $\times$ denotes the Cartesian product. We denote by $\text{pow}(A)$ the powerset of A and by $\text{mpow}(A)$ the power-multiset (set of multisets) of A. The cardinality of a finite set A is denoted as $\|A\|$. By writing $A \subseteq_{fin} B$ we mean that A is a finite subset of B. Given integers i and j, we write $[i, j]$ for the set $\{i, i+1, \ldots, j\}$, assumed to be empty if $i > j$. For a tuple $\mathbf{t} = \langle t_1, \ldots, t_n \rangle$, we define $|\mathbf{t}| \stackrel{\text{def}}{=} n$, $\langle \mathbf{t} \rangle_i \stackrel{\text{def}}{=} t_i$ and $\langle \mathbf{t} \rangle_{[i,j]} \stackrel{\text{def}}{=} \langle t_i, \ldots, t_j \rangle$. By writing $x = poly(y)$, for given $x, y \in \mathbb{N}$, we mean that there exists a polynomial function $f : \mathbb{N} \to \mathbb{N}$, such that $x \leq f(y)$.

2.1 Configurations

We model distributed systems as hypergraphs, whose vertices are *components* (i.e., the nodes of the network) and hyperedges are *interactions* (i.e., describing the way the components communicate with each other). The components are taken from a countably infinite set $\mathbb{C}$, called the *universe*. We consider that each component executes its own copy of the same *behavior*, represented as a finite-state machine $\mathbb{B} = (\mathcal{P}, Q, \rightarrow)$, where $\mathcal{P}$ is a finite set of *ports*, Q is a finite set of *states* and $\rightarrow \subseteq Q \times \mathcal{P} \times Q$ is a transition relation. Intuitively, each transition $q \xrightarrow{p} q'$ of the behavior is triggerred by a visible event, represented by the port p. For instance, the behavior of the components of the token ring system from Fig. 1(a) is $\mathbb{B} = (\{in, out\}, \{\mathsf{H}, \mathsf{T}\}, \{\mathsf{H} \xrightarrow{in} \mathsf{T}, \mathsf{T} \xrightarrow{out} \mathsf{H}\})$. *The universe $\mathbb{C}$ and the behavior $\mathbb{B} = (\mathcal{P}, Q, \rightarrow)$ are fixed in the rest of this paper.*

We introduce a logic for describing infinite sets of *configurations* of distributed systems with unboundedly many components and interactions. A configuration is a snapshot of the system, describing the topology of the network (i.e., the set of present components and interactions) together with the local state of each component:

Definition 1. *A* configuration *is a tuple $\gamma = (C, I, \rho)$, where:*

- *$C \subseteq_{fin} \mathbb{C}$ is a finite set of* components*, that are present in the configuration,*
- *$I \subseteq_{fin} (\mathbb{C} \times \mathcal{P})^+$ is a finite set of* interactions*, where each interaction is a sequence $(c_1, p_1, \ldots, c_n, p_n) \in (\mathbb{C} \times \mathcal{P})^n$ that binds together the ports $p_1, \ldots, p_n$ of the pairwise distinct components $c_1, \ldots, c_n$, respectively.*
- *$\rho : \mathbb{C} \to Q$ is a* state map *associating each (possibly absent) component, a state of the behavior $\mathbb{B}$, such that the set $\{c \in \mathbb{C} \mid \rho(c) = q\}$ is infinite, for each $q \in Q$.*

The last condition requires that there is an infinite pool of components in each state $q \in Q$; since $\mathbb{C}$ is infinite and Q is finite, this condition is feasible. For example, the configurations of the token ring from Fig. 1(a) are $(\{c_1, \ldots, c_n\}, \{(c_i, out, c_{(i \bmod n)+1}, in) \mid$

$i \in [1,n]\}, \rho)$, where $\rho : \mathbb{C} \to \{\mathsf{H}, \mathsf{T}\}$ is a state map. The ring topology is described by the set of components $\{c_1, \ldots, c_n\}$ and interactions $\{(c_i, out, c_{(i \bmod n)+1}, in) \mid i \in [1,n]\}$.

Intuitively, an interaction $(c_1, p_1, \ldots, c_n, p_n)$ synchronizes transitions labeled by the ports $p_1, \ldots, p_n$ from the behaviors (i.e., replicas of the state machine $\mathbb{B}$) of $c_1, \ldots, c_n$, respectively. Note that the components c_i are not necessary part of the configuration. The interactions are classified according to their sequence of ports, called the *interaction type* and let $\mathsf{Inter} \stackrel{\text{def}}{=} \mathcal{P}^+$ be the set of interaction types; an interaction type models, for instance, the passing of a certain kind of message (e.g., request, acknowledgement, etc.). From an operational point of view, two interactions that differ by a permutation of indices e.g., $(c_1, p_1, \ldots, c_n, p_n)$ and $(c_{i_1}, p_{i_1}, \ldots, c_{i_n}, p_{i_n})$ such that $\{i_1, \ldots, i_n\} = [1,n]$, are equivalent, since the set of transitions is the same; nevertheless, we chose to distinguish them in the following, exclusively for reasons of simplicity.

Below we define the composition of configurations, as the union of disjoint sets of components and interactions:

Definition 2. *The composition of two configurations $\gamma_i = (\mathcal{C}_i, I_i, \rho)$, for $i = 1,2$, such that $\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$ and $I_1 \cap I_2 = \emptyset$, is defined as $\gamma_1 \bullet \gamma_2 \stackrel{\text{def}}{=} (\mathcal{C}_1 \cup \mathcal{C}_2, I_1 \cup I_2, \rho)$. The composition $\gamma_1 \bullet \gamma_2$ is undefined if $\mathcal{C}_1 \cap \mathcal{C}_2 \neq \emptyset$ or $I_1 \cap I_2 \neq \emptyset$.*

In analogy with graphs, the *degree* of a configuration is the maximum number of interactions from the configuration that involve a (possibly absent) component:

Definition 3. *The* degree *of a configuration $\gamma = (\mathcal{C}, I, \rho)$ is defined as $\delta(\gamma) \stackrel{\text{def}}{=} \max_{c \in \mathbb{C}} \delta_c(\gamma)$, where $\delta_c(\gamma) \stackrel{\text{def}}{=} \|\{(c_1, p_1, \ldots, c_n, p_n) \in I \mid c = c_i,\ i \in [1,n]\}\|$.*

For instance, the configuration of the system from Fig. 1(a) has degree two.

2.2 Configuration Logic

Let $\mathbb{V}$ and $\mathbb{A}$ be countably infinite sets of *variables* and *predicates*, respectively. For each predicate $\mathsf{A} \in \mathbb{A}$, we denote its arity by $\#\mathsf{A}$. The formulæ of the *Configuration Logic* (CL) are described inductively by the following syntax:

$$\phi := \mathsf{emp} \mid [x] \mid \langle x_1.p_1, \ldots, x_n.p_n \rangle \mid x@q \mid x = y \mid x \neq y \mid \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \mid \phi * \phi \mid \exists x \,.\, \phi$$

where $x, y, x_1, \ldots \in \mathbb{V}$, $q \in Q$ and $\mathsf{A} \in \mathbb{A}$. A formula $[x]$, $\langle x_1.p_1, \ldots, x_n.p_n \rangle$, $x@q$ and $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$ is called a *component*, *interaction*, *state* and *predicate* atom, respectively. These formulæ are also referred to as *atoms*. The connective $*$ is called the *separating conjunction*. We use the shorthand $[x]@q \stackrel{\text{def}}{=} [x] * x@q$. For instance, the formula $[x]@q * [y]@q' * \langle x.out, y.in \rangle * \langle x.in, y.out \rangle$ describes a configuration consisting of two distinct components, denoted by the values of x and y, in states q and q', respectively, and two interactions binding the *out* port of one to the *in* port of the other component.

A formula is said to be *pure* if and only if it is a separating conjunction of state atoms, equalities and disequalities. A formula with no occurrences of predicate atoms (resp. existential quantifiers) is called *predicate-free* (resp. *quantifier-free*). A variable is *free* if it does not occur within the scope of an existential quantifier ; we note $\mathrm{fv}(\phi)$ the set of free variables of ϕ. A *sentence* is a formula with no free variables. A *substitution*

$\phi[x_1/y_1 \ldots x_n/y_n]$ replaces simultaneously every free occurrence of x_i by y_i in ϕ, for all $i \in [1,n]$. Before defining the semantics of CL formulæ, we introduce the set of inductive definitions that assigns meaning to predicates:

Definition 4. *A* set of inductive definitions (SID) Δ *consists of* rules $\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \phi$, *where* $x_1,\ldots,x_{\#\mathsf{A}}$ *are pairwise distinct variables, called* parameters, *such that* $\mathrm{fv}(\phi) \subseteq \{x_1,\ldots,x_{\#\mathsf{A}}\}$. *The rule* $\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \phi$ defines A *and we denote by* $\mathrm{def}_\Delta(\mathsf{A})$ *the set of rules from* Δ *that define* A.

Note that having distinct parameters in a rule is without loss of generality, as e.g., a rule $\mathsf{A}(x_1,x_1) \leftarrow \phi$ can be equivalently written as $\mathsf{A}(x_1,x_2) \leftarrow x_1 = x_2 * \phi$. As a convention, we shall always use the names $x_1,\ldots,x_{\#\mathsf{A}}$ for the parameters of a rule that defines A.

The semantics of CL formulæ is defined by a satisfaction relation $\gamma \models^{\nu}_{\Delta} \phi$ between configurations and formulæ. This relation is parameterized by a *store* $\nu : \mathbb{V} \to \mathbb{C}$ mapping the free variables of a formula into components from the universe (possibly absent from γ) and an SID Δ. We write $\nu[x \leftarrow c]$ for the store that maps x into c and agrees with ν on all variables other than x. The definition of the satisfaction relation is by induction on the structure of formulæ, where $\gamma = (C, I, \rho)$ is a configuration (Definition 1):

$$
\begin{array}{ll}
\gamma \models^{\nu}_{\Delta} \mathsf{emp} & \iff C = \emptyset \text{ and } I = \emptyset \\
\gamma \models^{\nu}_{\Delta} [x] & \iff C = \{\nu(x)\} \text{ and } I = \emptyset \\
\gamma \models^{\nu}_{\Delta} \langle x_1.p_1,\ldots,x_n.p_n\rangle & \iff C = \emptyset \text{ and } I = \{(\nu(x_1),p_1,\ldots,\nu(x_n),p_n)\} \\
\gamma \models^{\nu}_{\Delta} x@q & \iff \gamma \models^{\nu}_{\Delta} \mathsf{emp} \text{ and } \rho(\nu(x)) = q \\
\gamma \models^{\nu}_{\Delta} x \sim y & \iff \gamma \models^{\nu}_{\Delta} \mathsf{emp} \text{ and } \nu(x) \sim \nu(y), \text{ for all } \sim \in \{=,\neq\} \\
\gamma \models^{\nu}_{\Delta} \mathsf{A}(y_1,\ldots,y_{\#\mathsf{A}}) & \iff \gamma \models^{\nu}_{\Delta} \phi[x_1/y_1,\ldots,x_{\#\mathsf{A}}/y_{\#\mathsf{A}}], \text{ for some rule} \\
 & \quad \mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \phi \text{ from } \Delta \\
\gamma \models^{\nu}_{\Delta} \phi_1 * \phi_2 & \iff \text{exist } \gamma_1,\gamma_2, \text{ such that } \gamma = \gamma_1 \bullet \gamma_2 \text{ and } \gamma_i \models^{\nu}_{\Delta} \phi_i, \text{ for } i = 1,2 \\
\gamma \models^{\nu}_{\Delta} \exists x \,.\, \phi & \iff \gamma \models^{\nu[x \leftarrow c]}_{\Delta} \phi, \text{ for some } c \in \mathbb{C}
\end{array}
$$

If ϕ is a sentence, the satisfaction relation $\gamma \models^{\nu}_{\Delta} \phi$ does not depend on the store, written $\gamma \models_{\Delta} \phi$, in which case we say that γ is a *model* of ϕ. If ϕ is a predicate-free formula, the satisfaction relation does not depend on the SID, written $\gamma \models^{\nu} \phi$. A formula ϕ is *satisfiable* if and only if the sentence $\exists x_1 \ldots \exists x_n \,.\, \phi$ has a model, where $\mathrm{fv}(\phi) = \{x_1,\ldots,x_n\}$. A formula ϕ *entails* a formula ψ, written $\phi \models_{\Delta} \psi$ if and only if, for any configuration γ and store ν, we have $\gamma \models^{\nu}_{\Delta} \phi$ only if $\gamma \models^{\nu}_{\Delta} \psi$.

2.3 Separation Logic

Separation Logic (SL) [39] will be used in the following to prove several technical results concerning the decidability and complexity of certain decision problems for CL. For self-containment reasons, we define SL below. The syntax of SL formulæ is described by the following grammar:

$$\phi := \mathsf{emp} \mid x_0 \mapsto (x_1,\ldots,x_{\mathfrak{K}}) \mid x = y \mid x \neq y \mid \mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \mid \phi * \phi \mid \exists x \,.\, \phi$$

where $x,y,x_0,x_1,\ldots \in \mathbb{V}$, $\mathsf{A} \in \mathbb{A}$ and $\mathfrak{K} \geq 1$ is an integer constant. Formulæ of SL are interpreted over finite partial functions $\mathsf{h} : \mathbb{C} \rightharpoonup_{fin} \mathbb{C}^{\mathfrak{K}}$, called *heaps*[1], by a satisfaction relation $\mathsf{h} \Vdash^{\nu} \phi$, defined inductively as follows:

[1] We use the universe $\mathbb{C}$ here for simplicity, the definition works with any countably infinite set.

$$
\begin{array}{ll}
\mathsf{h} \Vdash^{\nu}_{\Delta} \mathsf{emp} & \iff \mathsf{h} = \emptyset \\
\mathsf{h} \Vdash^{\nu}_{\Delta} x_0 \mapsto (x_1, \ldots, x_{\mathfrak{K}}) & \iff \mathrm{dom}(\mathsf{h}) = \{\nu(x_0)\} \text{ and } \mathsf{h}(\nu(x_0)) = \langle \nu(x_1), \ldots, \nu(x_{\mathfrak{K}}) \rangle \\
\mathsf{h} \Vdash^{\nu} \phi_1 * \phi_2 & \iff \text{there exist } \mathsf{h}_1, \mathsf{h}_2 \text{ such that } \mathrm{dom}(\mathsf{h}_1) \cap \mathrm{dom}(\mathsf{h}_2) = \emptyset, \\
 & \qquad \mathsf{h} = \mathsf{h}_1 \cup \mathsf{h}_2 \text{ and } \mathsf{h}_i \Vdash^{\nu}_{\Delta} \phi_i, \text{ for both } i = 1, 2
\end{array}
$$

where $\mathrm{dom}(\mathsf{h}) \stackrel{\text{def}}{=} \{c \in \mathbb{C} \mid \mathsf{h}(c) \text{ is defined}\}$ is the domain of the heap and (dis-) equalities, predicate atoms and existential quantifiers are defined same as for SL.

2.4 Decision Problems

We define the decision problems that are the focus of the upcoming sections. As usual, a decision problem is a class of yes/no queries that differ only in their input. In our case, the input consists of an SID and one or two predicates, written between square brackets.

Definition 5. *We consider the following problems, for a SID* Δ *and predicates* $\mathsf{A}, \mathsf{B} \in \mathbb{A}$*:*

1. $\mathsf{Sat}[\Delta, \mathsf{A}]$*: is the sentence* $\exists x_1 \ldots \exists x_{\#\mathsf{A}} \,.\, \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$ *satisfiable for* Δ*?*
2. $\mathsf{Bnd}[\Delta, \mathsf{A}]$*: is the set* $\{\delta(\gamma) \mid \gamma \models_{\Delta} \exists x_1 \ldots \exists x_{\#\mathsf{A}} \,.\, \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})\}$ *finite?*
3. $\mathsf{Entl}[\Delta, \mathsf{A}, \mathsf{B}]$*: does* $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \models_{\Delta} \exists x_{\#\mathsf{B}+1} \ldots \exists x_{\#\mathsf{A}} \,.\, \mathsf{B}(x_1, \ldots, x_{\#\mathsf{B}})$ *hold?*

The size of a formula ϕ is the total number of occurrences of symbols needed to write it down, denoted by $\mathrm{size}(\phi)$. The size of a SID Δ is $\mathrm{size}(\Delta) \stackrel{\text{def}}{=} \sum_{\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \phi \in \Delta} \mathrm{size}(\phi) + \#\mathsf{A} + 1$. Other parameters of a SID Δ are:

- $\mathrm{arity}(\Delta) \stackrel{\text{def}}{=} \max\{\#\mathsf{A} \mid \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \phi \in \Delta\}$,
- $\mathrm{width}(\Delta) \stackrel{\text{def}}{=} \max\{\mathrm{size}(\phi) \mid \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \phi \in \Delta\}$,
- $\mathrm{intersize}(\Delta) \stackrel{\text{def}}{=} \max\{n \mid \langle x_1.p_1, \ldots, x_n.p_n \rangle \text{ occurs in } \phi, \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \phi \in \Delta\}$.

For a decision problem $\mathsf{P}[\Delta, \mathsf{A}, \mathsf{B}]$, we consider its (k, ℓ)-bounded versions $\mathsf{P}^{(k,\ell)}[\Delta, \mathsf{A}, \mathsf{B}]$, obtained by restricting the predicates and interaction atoms occurring Δ to $\mathrm{arity}(\Delta) \leq k$ and $\mathrm{intersize}(\Delta) \leq \ell$, respectively, where k and ℓ are either positive integers or infinity. We consider, for each $\mathsf{P}[\Delta, \mathsf{A}, \mathsf{B}]$, the subproblems $\mathsf{P}^{(k,\ell)}[\Delta, \mathsf{A}, \mathsf{B}]$ corresponding to the three cases (1) $k < \infty$ and $\ell = \infty$, (2) $k = \infty$ and $\ell < \infty$, and (3) $k = \infty$ and $\ell = \infty$. As we explain next, this is because, for the decision problems considered (Definition 5), the complexity for the case $k < \infty, \ell < \infty$ matches the one for the case $k < \infty, \ell = \infty$.

Satisfiability (1) and entailment (3) arise naturally during verification of reconfiguration programs. For instance, $\mathsf{Sat}[\Delta, \phi]$ asks whether a specification ϕ of a set configurations (e.g., a pre-, post-condition, or a loop invariant) is empty or not (e.g., an empty precondition typically denotes a vacuous verification condition), whereas $\mathsf{Entl}[\Delta, \phi, \psi]$ is used as a side condition for the Hoare rule of consequence, as in e.g., the proof from Fig. 1(c). Moreover, entailments must be proved when checking inductiveness of a user-provided loop invariant.

The $\mathsf{Bnd}[\Delta, \phi]$ problem is used to check a necessary condition for the decidability of entailments i.e., $\mathsf{Entl}[\Delta, \phi, \psi]$. If $\mathsf{Bnd}[\Delta, \phi]$ has a positive answer, we can reduce the problem $\mathsf{Entl}[\Delta, \phi, \psi]$ to an entailment problem for SL, which is always interpreted over heaps of bounded degree [18]. Otherwise, the decidability status of the entailment problem is open, for configurations of unbounded degree, such as the one described by the example below.

Example 1. The following SID describes star topologies with a central controller connected to an unbounded number of workers stations:

$$\mathit{Controller}(x) \leftarrow [x] * \mathit{Worker}(x)$$
$$\mathit{Worker}(x) \leftarrow \exists y \,.\, \langle x.\mathit{out}, y.\mathit{in}\rangle * [y] * \mathit{Worker}(x) \qquad \mathit{Worker}(x) \leftarrow \quad \mathsf{emp} \quad \blacksquare$$

3 Satisfiability

We show that the satisfiability problem (Definition 5, point 1) is decidable, using a method similar to the one pioneered by Brotherston et al. [9], for checking satisfiability of inductively defined symbolic heaps in SL. We recall that a formula π is *pure* if and only if it is a separating conjunction of equalities, disequalities and state atoms. In the following, the order of terms in (dis-)equalities is not important i.e., we consider $x = y$ (resp. $x \neq y$) and $y = x$ (resp. $y \neq x$) to be the same formula.

Definition 6. *The* closure $\mathrm{cl}(\pi)$ *of a pure formula* π *is the limit of the sequence* $\pi^0, \pi^1, \pi^2, \ldots$ *such that* $\pi^0 = \pi$ *and, for each* $i \geq 0$, π^{i+1} *is obtained by joining (with* $*$*) all of the following formulæ to* π^i*:*

- $x = z$*, where* x *and* z *are the same variable, or* $x = y$ *and* $y = z$ *both occur in* π^i*,*
- $x \neq z$*, where* $x = y$ *and* $y \neq z$ *both occur in* π^i*, or*
- $y@q$*, where* $x@q$ *and* $x = y$ *both occur in* π^i*.*

Because only finitely many such formulæ can be added, the sequence of pure formulæ from Definition 6 is bound to stabilize after polynomially many steps. A pure formula is satisfiable if and only if its closure does not contain contradictory literals i.e., $x = y$ and $x \neq y$, or $x@q$ and $x@q'$, for $q \neq q' \in Q$. We write $x \approx_\pi y$ (resp. $x \not\approx_\pi y$) if and only if $x = y$ (resp. $x \neq y$) occurs in $\mathrm{cl}(\pi)$ and $\mathrm{not}(x \approx_\pi y)$ (resp. $\mathrm{not}(x \not\approx_\pi y)$) whenever $x \approx_\pi y$ (resp. $x \not\approx_\pi y$) does not hold. Note that e.g., $\mathrm{not}(x \approx_\pi y)$ is not the same as $x \not\approx_\pi y$.

Base tuples constitute the abstract domain used by the algorithms for checking satisfiability (point 1 of Definition 5) and boundedness (point 2 of Definition 5), defined as follows:

Definition 7. *A* base tuple *is a triple* $\mathsf{t} = (\mathcal{C}^\sharp, I^\sharp, \pi)$*, where:*

- $\mathcal{C}^\sharp \in \mathit{mpow}\mathbb{V}$ *is a multiset of variables denoting present components,*
- $I^\sharp : \mathsf{Inter} \to \mathit{mpow}\mathbb{V}^+$ *maps each interaction type* $\tau \in \mathsf{Inter}$ *into a multiset of tuples of variables of length* $|\tau|$ *each, and*
- π *is a pure formula.*

A base tuple is called satisfiable *if and only if* π *is satisfiable and the following hold:*

1. *for all* $x, y \in \mathcal{C}^\sharp$, $\mathrm{not}(x \approx_\pi y)$,
2. *for all* $\tau \in \mathsf{Inter}$, $\langle x_1, \ldots, x_{|\tau|}\rangle, \langle y_1, \ldots, y_{|\tau|}\rangle \in I^\sharp(\tau)$*, there exists* $i \in [1, |\tau|]$ *such that* $\mathrm{not}(x_i \approx_\pi y_i)$,
3. *for all* $\tau \in \mathsf{Inter}$, $\langle x_1, \ldots, x_{\#\tau}\rangle \in I^\sharp(\tau)$ *and* $1 \leq i < j \leq |\tau|$*, we have* $\mathrm{not}(x_i \approx_\pi x_j)$.

We denote by SatBase *the set of satisfiable base tuples.*

Intuitively, a base tuple is an abstract representation of a configuration, where components (resp. interactions) are represented by variables (resp. tuples of variables). Note that a base tuple $(\mathcal{C}^\sharp, I^\sharp, \pi)$ is unsatisfiable if $\mathcal{C}^\sharp$ ($I^\sharp$) contains the same variable (tuple of variables) twice (for the same interaction type), hence the use of multisets in the definition of base tuples. It is easy to see that checking the satisfiability of a given base tuple $(\mathcal{C}^\sharp, I^\sharp, \pi)$ can be done in time $poly(\|\mathcal{C}^\sharp\| + \sum_{\tau \in \mathsf{Inter}} \|I^\sharp(\tau)\| + \mathrm{size}(\pi))$.

We define a partial *composition* operation on satisfiable base tuples, as follows:

$$(\mathcal{C}_1^\sharp, I_1^\sharp, \pi_1) \otimes (\mathcal{C}_2^\sharp, I_2^\sharp, \pi_2) \stackrel{\mathsf{def}}{=} (\mathcal{C}_1^\sharp \cup \mathcal{C}_2^\sharp, I_1^\sharp \cup I_2^\sharp, \pi_1 * \pi_2)$$

where the union of multisets is lifted to functions $\mathsf{Inter} \to \mathrm{mpow}(\mathbb{V}^+)$ in the usual way. The composition operation $\otimes$ is undefined if $(\mathcal{C}_1^\sharp, I_1^\sharp, \pi_1) \otimes (\mathcal{C}_2^\sharp, I_2^\sharp, \pi_2)$ is not satisfiable e.g., if $\mathcal{C}_1^\sharp \cap \mathcal{C}_2^\sharp \neq \emptyset$, $I_1^\sharp(\tau) \cap I_2^\sharp(\tau) \neq \emptyset$, for some $\tau \in \mathsf{Inter}$, or $\pi_1 * \pi_2$ is not satisfiable.

Given a pure formula π and a set of variables X, the projection $\pi{\downarrow_X}$ removes from π all atoms α, such that $\mathrm{fv}(\alpha) \not\subseteq X$. The *projection* of a base tuple $(\mathcal{C}^\sharp, I^\sharp, \pi)$ on a variable set X is formally defined below:

$$(\mathcal{C}^\sharp, I^\sharp, \pi){\downarrow_X} \stackrel{\mathsf{def}}{=} \Big(\mathcal{C}^\sharp \cap X, \lambda\tau \,.\, \{\langle x_1, \ldots, x_{|\tau|}\rangle \in I^\sharp(\tau) \mid x_1, \ldots, x_{|\tau|} \in X\}, \mathrm{cl}(\mathrm{dist}(I^\sharp) * \pi){\downarrow_X}\Big)$$
$$\text{where } \mathrm{dist}(I^\sharp) \stackrel{\mathsf{def}}{=} \ast_{\tau \in \mathsf{Inter}} \ast_{\langle x_1, \ldots, x_{|\tau|}\rangle \in I^\sharp(\tau)} \ast_{1 \leq i < j \leq |\tau|} x_i \neq x_j$$

The *substitution* operation $(\mathcal{C}^\sharp, I^\sharp, \pi)[x_1/y_1, \ldots, x_n/y_n]$ replaces simultaneously each x_i with y_i in $\mathcal{C}^\sharp$, $I^\sharp$ and π, respectively. We lift the composition, projection and substitution operations to sets of satisfiable base tuples, as usual.

Next, we define the base tuple corresponding to a quantifier- and predicate-free formula $\phi = \psi * \pi$, where ψ consists of component and interaction atoms and π is pure. Since, moreover, we are interested in those components and interactions that are visible through a given indexed set of parameters $X = \{x_1, \ldots, x_n\}$, for a variable y, we denote by $\{\{y\}\}_\pi^X$ the parameter x_i with the least index, such that $y \approx_\pi x_i$, or y itself, if no such parameter exists. We define the following sets of formulæ:

$$\mathsf{Base}(\phi, X) \stackrel{\mathsf{def}}{=} \begin{cases} \{(\mathcal{C}^\sharp, I^\sharp, \pi)\} & \text{, if } (\mathcal{C}^\sharp, I^\sharp, \pi) \text{ is satisfiable} \\ \emptyset & \text{, otherwise} \end{cases}$$
$$\text{where } \mathcal{C}^\sharp \stackrel{\mathsf{def}}{=} \{\{\{x\}\}_\pi^X \mid [x] \text{ occurs in } \psi\}$$
$$I^\sharp \stackrel{\mathsf{def}}{=} \lambda\langle p_1, \ldots, p_s\rangle . \left\{\left\langle \{\{y_1\}\}_\pi^X, \ldots, \{\{y_s\}\}_\pi^X \right\rangle \mid \langle y_1.p_1, \ldots, y_s.p_s\rangle \text{ occurs in } \psi\right\}$$

We consider a tuple of variables $\overrightarrow{\mathcal{X}}$, having a variable $\mathcal{X}(\mathsf{A})$ ranging over $\mathrm{pow}(\mathsf{SatBase})$, for each predicate A that occurs in Δ. With these definitions, each rule of Δ:

$$\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * \mathsf{B}_1(z_1^1, \ldots, z_{\#\mathsf{B}_1}^1) * \ldots * \mathsf{B}_h(z_1^h, \ldots, z_{\#\mathsf{B}_h}^h)$$

where ϕ is a quantifier- and predicate-free formula, induces the constraint:

$$\mathcal{X}(\mathsf{A}) \supseteq \Big(\mathsf{Base}(\phi, \{x_1, \ldots, x_{\#\mathsf{A}}\}) \otimes \bigotimes_{\ell=1}^{h} \mathcal{X}(\mathsf{B}_\ell)[x_1/z_1^\ell, \ldots, x_{\#\mathsf{B}_\ell}/z_{\#\mathsf{B}_\ell}^\ell]\Big){\downarrow_{x_1, \ldots, x_{\#\mathsf{A}}}} \qquad (1)$$

input: a SID Δ **output**: $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp$

1: initially $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp := \lambda \mathsf{A} \,.\, \emptyset$
2: **for** $\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi \in \Delta$, with ϕ quantifier- and predicate-free **do**
3: $\quad \mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{A}) := \mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{A}) \cup \mathsf{Base}(\phi, \{x_1,\ldots,x_{\#\mathsf{A}}\})\downarrow_{x_1,\ldots,x_{\#\mathsf{A}}}$
4: **while** $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp$ still change **do**
5: $\quad$ **for** $\mathsf{r} : \mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * \circledast_{\ell=1}^{h} \mathsf{B}_\ell(z_1^\ell,\ldots,z_{\#\mathsf{B}_\ell}^\ell) \in \Delta$ **do**
6: $\quad\quad$ **if there exist** $\mathsf{t}_1 \in \mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{B}_1),\ldots,\mathsf{t}_h \in \mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{B}_h)$ **then**
7: $\quad\quad\quad \mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{A}) := \mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{A}) \cup \left(\mathsf{Base}(\phi, \{x_1,\ldots,x_{\#\mathsf{A}}\}) \otimes \bigotimes_{\ell=1}^{h} \mathsf{t}_\ell[x_1/z_1^\ell,\ldots,x_{\#\mathsf{B}_\ell}/z_{\#\mathsf{B}_\ell}^\ell]\right)\downarrow_{x_1,\ldots,x_{\#\mathsf{A}}}$

Fig. 2. Algorithm for the Computation of the Least Solution

Let $\Delta^\sharp$ be the set of such constraints, corresponding to the rules in Δ and let $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp$ be the tuple of least solutions of the constraint system generated from Δ, indexed by the tuple of predicates that occur in Δ, such that $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{A})$ denotes the entry of $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp$ correponding to A. Since the composition and projection are monotonic operations, such a least solution exists and is unique. Since $\mathsf{SatBase}$ is finite, the least solution can be attained in a finite number of steps, using a Kleene iteration (see Fig. 2).

We state below the main result leading to an elementary recursive algorithm for the satisfiability problem (Theorem 1). The intuition is that, if $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{A})$ is not empty, then it contains only satisfiable base tuples, from which a model of $\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}})$ can be built.

Lemma 1. $\mathsf{Sat}[\Delta,\mathsf{A}]$ *has a positive answer if and only if* $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{A}) \neq \emptyset$.

If the maximal arity of the predicates occurring in Δ is bound by a constant k, no satisfiable base tuple $(\mathcal{C}^\sharp, I^\sharp, \pi)$ can have a tuple $\langle y_1,\ldots,y_{|\tau|}\rangle \in I^\sharp(\tau)$, for some $\tau \in \mathsf{Inter}$, such that $|\tau| > k$, since all variables $y_1,\ldots,y_{|\tau|}$ are parameters denoting distinct components (point 3 of Definition 7). Hence, the upper bound on the size of a satisfiable base tuple is constant, in both the $k < \infty, \ell < \infty$ and $k < \infty, \ell = \infty$ cases, which are, moreover indistinguishable complexity-wise (i.e., both are NP-complete). In contrast, in the cases $k = \infty, \ell < \infty$ and $k = \infty, \ell = \infty$, the upper bound on the size of satisfiable base tuples is polynomial and simply exponential in size(Δ), incurring a complexity gap of one and two exponentials, respectively. The theorem below states the main result of this section:

Theorem 1. $\mathsf{Sat}^{(k,\infty)}[\Delta,\mathsf{A}]$ *is* NP*-complete for* $k \geq 4$, $\mathsf{Sat}^{(\infty,\ell)}[\Delta,\mathsf{A}]$ *is* EXP*-complete and* $\mathsf{Sat}[\Delta,\mathsf{A}]$ *is in* 2EXP.

The upper bounds are consequences of the fact that the size of a satisfiable base tuple is bounded by a simple exponential in the $\min(\text{arity}(\Delta), \text{intersize}(\Delta))$, hence the number of such tuples is doubly exponential in $\min(\text{arity}(\Delta), \text{intersize}(\Delta))$. The lower bounds are by a polynomial reduction from the satisfiability problem for SL [9].

Example 2. The doubly-exponential upper bound for the algorithm computing the least solution of a system of constraints of the form (1) is necessary, in general, as illustrated by the following worst-case example. Let n be a fixed parameter and consider the n-arity predicates $A_1,\ldots,A_n$ defined by the following SID:

$$A_i(x_1,\ldots,x_n) \leftarrow \circledast_{j=0}^{n-i} A_{i+1}(x_1,\ldots,x_{i-1},[x_i,\ldots,x_n]^j), \quad \text{for all } i \in [1,n-1]$$
$$A_n(x_1,\ldots,x_n) \leftarrow \langle x_1.p,\ldots,x_n.p\rangle \quad A_n(x_1,\ldots,x_n) \leftarrow \mathsf{emp}$$

where, for a list of variables $x_i,\ldots,x_n$ and an integer $j \geq 0$, we write $[x_i,\ldots,x_n]^j$ for the list rotated to the left j times (e.g., $[x_1,x_2,x_3,x_4,x_5]^2 = x_3,x_4,x_5,x_1,x_2$). In this example, when starting with $A_1(x_1,\ldots,x_n)$ one eventually obtains predicate atoms $A_n(x_{i_1},\ldots,x_{i_n})$, for any permutation $x_{i_1},\ldots,x_{i_n}$ of $x_1,\ldots,x_n$. Since A_n may choose to create or not an interaction with that permutation of variables, the total number of base tuples generated for A_1 is $2^{n!}$. That is, the fixpoint iteration generates $2^{2^{\mathcal{O}(n\log n)}}$ base tuples, whereas the size of the input of $\mathsf{Sat}[\Delta,\mathsf{A}]$ is $poly(n)$. ■

4 Degree Boundedness

The boundedness problem (Definition 5, point 2) asks for the existence of a bound on the degree (Definition 3) of the models of a sentence $\exists x_1 \ldots \exists x_{\#\mathsf{A}} \,.\, \mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}})$. Intuitively, the $\mathsf{Bnd}[\Delta,\mathsf{A}]$ problem has a negative answer if and only if there are increasingly large unfoldings (i.e., expansions of a formula by replacement of a predicate atom with one of its definitions) of $\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}})$ repeating a rule that contains an interaction atom involving a parameter of the rule, which is always bound to the same component. We formalize the notion of unfolding below:

Definition 8. *Given a predicate* A *and a sequence* $(\mathsf{r}_1,i_1),\ldots,(\mathsf{r}_n,i_n) \in (\Delta\times\mathbb{N})^+$, *where* $\mathsf{r}_1 : \mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \phi \in \Delta$, *the* unfolding $\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \xrightarrow{(\mathsf{r}_1,i_1)\ldots(\mathsf{r}_n,i_n)}_{\Delta} \psi$ *is inductively defined as (1)* $\psi = \phi$ *if* $n = 1$, *and (2)* ψ *is obtained from* ϕ *by replacing its* i_1*-th predicate atom* $\mathsf{B}(y_1,\ldots,y_{\#\mathsf{B}})$ *with* $\psi_1[x_1/y_1,\ldots,x_{\#\mathsf{B}}/y_{\#\mathsf{B}}]$, *where* $\mathsf{B}(x_1,\ldots,x_{\#\mathsf{B}}) \xrightarrow{(\mathsf{r}_2,i_2)\ldots(\mathsf{r}_n,i_n)}_{\Delta} \psi_1$ *is an unfolding, if* $n > 1$.

We show that the $\mathsf{Bnd}[\Delta,\mathsf{A}]$ problem can be reduced to the existence of increasingly large unfoldings or, equivalently, a cycle in a finite directed graph, built by a variant of the least fixpoint iteration algorithm used to solve the satisfiability problem (Fig. 3).

Definition 9. *Given satisfiable base pairs* $\mathfrak{t},\mathfrak{u} \in \mathsf{SatBase}$ *and a rule from* Δ:

$$\mathsf{r} \;:\; \mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * \mathsf{B}_1(z_1^1,\ldots,z_{\#\mathsf{B}_1}^1) * \ldots * \mathsf{B}_h(z_1^h,\ldots,z_{\#\mathsf{B}_h}^h)$$

where ϕ *is a quantifier- and predicate-free formula, we write* $(\mathsf{A},\mathfrak{t}) \overset{(\mathsf{r},i)}{\rightsquigarrow} (\mathsf{B},\mathfrak{u})$ *if and only if* $\mathsf{B} = \mathsf{B}_i$ *and there exist satisfiable base tuples* $\mathfrak{t}_1,\ldots,\mathfrak{u} = \mathfrak{t}_i,\ldots,\mathfrak{t}_h \in \mathsf{SatBase}$, *such that* $\mathfrak{t} \in \big(\mathsf{Base}(\phi,\{x_1,\ldots,x_{\#\mathsf{A}}\}) \otimes \bigotimes_{\ell=1}^{h} \mathfrak{t}_\ell[x_1/z_1^\ell,\ldots,x_{\#\mathsf{B}_\ell}/z_{\#\mathsf{B}_\ell}^\ell]\big)\downarrow_{x_1,\ldots,x_{\#\mathsf{A}}}$. *We define the directed graph with edges labeled by pairs* $(\mathsf{r},i) \in \Delta\times\mathbb{N}$:

$$\mathcal{G}(\Delta) \overset{\mathsf{def}}{=} \big(\{\mathsf{def}(\Delta)\times\mathsf{SatBase}\}, \{\langle(\mathsf{A},\mathfrak{t}),(\mathsf{r},i),(\mathsf{B},\mathfrak{u})\rangle \mid (\mathsf{A},\mathfrak{t}) \overset{(\mathsf{r},i)}{\rightsquigarrow} (\mathsf{B},\mathfrak{u})\}\big)$$

The graph $\mathcal{G}(\Delta)$ is built by the algorithm in Fig. 3, a slight variation of the classical Kleene iteration algorithm for the computation of the least solution of the constraints of the form (1). A path $(\mathsf{A}_1,\mathfrak{t}_1) \overset{(\mathsf{r}_1,i_1)}{\rightsquigarrow} (\mathsf{A}_2,\mathfrak{t}_2) \overset{(\mathsf{r}_2,i_2)}{\rightsquigarrow} \ldots \overset{(\mathsf{r}_n,i_n)}{\rightsquigarrow} (\mathsf{A}_n,\mathfrak{t}_n)$ in $\mathcal{G}(\Delta)$ induces a unique

input: a SID Δ **output**: $\mathcal{G}(\Delta) = (V, E)$

initially $V := \emptyset$, $E := \emptyset$
for $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi \in \Delta$, with ϕ quantifier- and predicate-free **do**
  $V := V \cup \left(\{\mathsf{A}\} \times \mathsf{Base}(\phi, \{x_1, \ldots, x_{\#\mathsf{A}}\})\downarrow_{x_1, \ldots, x_{\#\mathsf{A}}}\right)$
while V or E still change **do**
  for $\mathsf{r} : \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * \divideontimes_{\ell=1}^{h} \mathsf{B}_\ell(z_1^\ell, \ldots, z_{\#\mathsf{B}_\ell}^\ell) \in \Delta$ **do**
    if there exist $(\mathsf{B}_1, \mathsf{t}_1), \ldots, (\mathsf{B}_h, \mathsf{t}_h) \in V$ **then**
      $\mathcal{X} := \left(\mathsf{Base}(\phi, \{x_1, \ldots, x_{\#\mathsf{A}}\}) \otimes \bigotimes_{\ell=1}^{h} \mathsf{t}_\ell[x_1/z_1^\ell, \ldots, x_{\#\mathsf{B}_\ell}/z_{\#\mathsf{B}_\ell}^\ell]\right)\downarrow_{x_1, \ldots, x_{\#\mathsf{A}}}$
      $V := V \cup (\{\mathsf{A}\} \times \mathcal{X})$
      $E := E \cup \{\langle(\mathsf{A}, \mathsf{t}), (\mathsf{r}, \ell), (\mathsf{B}_\ell, \mathsf{t}_\ell)\rangle \mid \mathsf{t} \in \mathcal{X}, \ell \in [1, h]\}$

Fig. 3. Algorithm for the Construction of $\mathcal{G}(\Delta)$

unfolding $\mathsf{A}_1(x_1, \ldots, x_{\#\mathsf{A}_1}) \xrightarrow{(\mathsf{r}_1, i_1)\ldots(\mathsf{r}_n, i_n)}_{\Delta} \phi$ (Definition 8). Since the vertices of $\mathcal{G}(\Delta)$ are pairs (A, t), where t is a satisfiable base tuple and the edges of $\mathcal{G}(\Delta)$ reflect the construction of the base tuples from the least solution of the constraints (1), the outcome ϕ of this unfolding is always a satisfiable formula.

An *elementary cycle* of $\mathcal{G}(\Delta)$ is a path from some vertex (B, u) back to itself, such that (B, u) does not occur on the path, except at its endpoints. The cycle is, moreover, *reachable* from (A, t) if and only if there exists a path $(\mathsf{A}, \mathsf{t}) \overset{(\mathsf{r}_1, i_1)}{\rightsquigarrow} \ldots \overset{(\mathsf{r}_n, i_n)}{\rightsquigarrow} (\mathsf{B}, \mathsf{u})$ in $\mathcal{G}(\Delta)$. We reduce the complement of the $\mathsf{Bnd}[\Delta, \mathsf{A}]$ problem, namely the existence of an infinite set of models of $\exists x_1 \ldots \exists x_{\#\mathsf{A}} \,.\, \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$ of unbounded degree, to the existence of a reachable elementary cycle in $\mathcal{G}(\Delta')$, where Δ' is obtained from Δ, as described in the following.

First, we consider, for each predicate $\mathsf{B} \in \mathrm{def}(\Delta)$, a predicate B', of arity $\#\mathsf{B} + 1$, not in $\mathrm{def}(\Delta)$ i.e., the set of predicates for which there exists a rule in Δ. Second, for each rule $\mathsf{B}_0(x_1, \ldots, x_{\#\mathsf{B}_0}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * \divideontimes_{\ell=2}^{h} \mathsf{B}_\ell(z_1^\ell, \ldots, z_{\#\mathsf{B}_\ell}^\ell) \in \Delta$, where ϕ is a quantifier- and predicate-free formula and $\mathrm{iv}(\phi) \subseteq \mathrm{fv}(\phi)$ denotes the subset of variables occurring in interaction atoms in ϕ, the SID Δ' has the following rules:

$$\mathsf{B}'_0(x_1, \ldots, x_{\#\mathsf{B}_0}, x_{\#\mathsf{B}_0+1}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * \divideontimes_{\xi \in \mathrm{iv}(\phi)} x_{\#\mathsf{B}_0+1} \neq \xi * \divideontimes_{\ell=2}^{h} \mathsf{B}'_\ell(z_1^\ell, \ldots, z_{\#\mathsf{B}_\ell}^\ell, x_{\#\mathsf{B}_0+1}) \tag{2}$$

$$\mathsf{B}'_0(x_1, \ldots, x_{\#\mathsf{B}_0}, x_{\#\mathsf{B}_0+1}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * x_{\#\mathsf{B}_0+1} = \xi * \divideontimes_{\ell=2}^{h} \mathsf{B}'_\ell(z_1^\ell, \ldots, z_{\#\mathsf{B}_\ell}^\ell, x_{\#\mathsf{B}_0+1}) \tag{3}$$

for each variable $\xi \in \mathrm{iv}(\phi)$, that occurs in an interaction atom in ϕ.

There exists a family of models (with respect to Δ) of $\exists x_1 \ldots \exists x_{\#\mathsf{A}} \,.\, \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$ of unbounded degree if and only if these are models of $\exists x_1 \ldots \exists x_{\#\mathsf{A}+1} \,.\, \mathsf{A}'(x_1, \ldots, x_{\#\mathsf{A}+1})$ (with respect to Δ') and the last parameter of each predicate $\mathsf{B}' \in \mathrm{def}(\Delta')$ can be mapped, in each of the these models, to a component that occurs in unboundedly many interactions. The latter condition is equivalent to the existence of an elementary cycle, containing a rule of the form (3), that it, moreover, reachable from some vertex $(\mathsf{A}', \mathsf{t})$ of $\mathcal{G}(\Delta')$, for some $\mathsf{t} \in \mathsf{SatBase}$. This reduction is formalized below:

Lemma 2. *There exists an infinite sequence of configurations* $\gamma_1, \gamma_2, \ldots$ *such that* $\gamma_i \models_\Delta \exists x_1 \ldots \exists x_{\#\mathsf{A}} \,.\, \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$ *and* $\delta(\gamma_i) < \delta(\gamma_{i+1})$*, for all* $i \geq 1$ *if and only if* $\mathcal{G}(\Delta')$ *has an elementary cycle containing a rule (3), reachable from a node* $(\mathsf{A}', \mathsf{t})$*, for* $\mathsf{t} \in \mathsf{SatBase}$.

The complexity result below uses a similar argument on the maximal size of (hence the number of) base tuples as in Theorem 1, leading to similar complexity gaps:

Theorem 2. $\mathsf{Bnd}^{(k,\infty)}[\Delta, \mathsf{A}]$ *is in* co-NP, $\mathsf{Bnd}^{(\infty,\ell)}[\Delta, \mathsf{A}]$ *is in* EXP, $\mathsf{Bnd}[\Delta, \mathsf{A}]$ *is in* 2EXP.

Moreover, the construction of $\mathcal{G}(\Delta')$ allows to prove the following cut-off result:

Proposition 1. *Let* γ *be a configuration and* ν *be a store, such that* $\gamma \models^{\nu}_{\Delta} \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$. *If* $\mathsf{Bnd}^{(k,\ell)}[\Delta, \mathsf{A}]$ *then (1)* $\delta(\gamma) = \mathit{poly}(\mathrm{size}(\Delta))$ *if* $k < \infty$, $\ell = \infty$, *(2)* $\delta(\gamma) = 2^{\mathit{poly}(\mathrm{size}(\Delta))}$ *if* $k = \infty$, $\ell < \infty$ *and (3)* $\delta(\gamma) = 2^{2^{\mathit{poly}(\mathrm{size}(\Delta))}}$ *if* $k = \infty$, $\ell = \infty$.

5 Entailment

This section is concerned with the entailment problem $\mathsf{Entl}[\Delta, \mathsf{A}, \mathsf{B}]$, that asks whether $\gamma \models^{\nu}_{\Delta} \exists x_{\#\mathsf{A}+1} \ldots \exists x_{\#\mathsf{B}} \,.\, \mathsf{B}(x_1, \ldots, x_{\#\mathsf{B}})$, for every configuration γ and store ν, such that $\gamma \models^{\nu}_{\Delta} \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$. For instance, the proof from Fig. 1(c) relies on the following entailments, that occur as the side conditions of the Hoare logic rule of consequence:

$$\mathsf{ring}_{h,t}(y) \models_\Delta \exists x \exists z.[y]@\mathsf{H} * \langle y.out, z.in \rangle * \mathsf{chain}_{h-1,t}(z,x) * \langle x.out, y.in \rangle$$
$$[z]@\mathsf{H} * \langle z.out, x.in \rangle * \mathsf{chain}_{h-1,t}(x,y) * \langle y.out, z.in \rangle \models_\Delta \mathsf{ring}_{h,t}(z)$$

By introducing two fresh predicates A_1 and A_2, defined by the rules:

$$\mathsf{A}_1(x_1) \leftarrow \exists y \exists z.[x_1]@\mathsf{H} * \langle x_1.out, z.in \rangle * \mathsf{chain}_{h-1,t}(z,y) * \langle y.out, x_1.in \rangle \quad (4)$$
$$\mathsf{A}_2(x_1, x_2) \leftarrow \exists z.[x_1]@\mathsf{H} * \langle x_1.out, z.in \rangle * \mathsf{chain}_{h-1,t}(z,x_2) * \langle x_2.out, x_1.in \rangle \quad (5)$$

the above entailments are equivalent to $\mathsf{Entl}[\Delta, \mathsf{ring}_{h,t}, \mathsf{A}_1]$ and $\mathsf{Entl}[\Delta, \mathsf{A}_2, \mathsf{ring}_{h,t}]$, respectively, where Δ consists of the rules (4) and (5), together with the rules that define the $\mathsf{ring}_{h,t}$ and $\mathsf{chain}_{h,t}$ predicates (Sect. 1.1).

We show that the entailment problem is undecidable, in general (Thm. 3), and recover a decidable fragment, by means of three syntactic conditions, typically met in our examples. These conditions use the following notion of *profile*:

Definition 10. *The* profile *of a SID* Δ *is the pointwise greatest function* $\lambda_\Delta : \mathbb{A} \to \mathit{pow}(\mathbb{N})$*, mapping each predicate* A *into a subset of* $[1, \#\mathsf{A}]$*, such that, for each rule* $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \phi$ *from* Δ*, each atom* $\mathsf{B}(y_1, \ldots, y_{\#\mathsf{B}})$ *from* ϕ *and each* $i \in \lambda_\Delta(\mathsf{B})$*, there exists* $j \in \lambda_\Delta(\mathsf{A})$*, such that* x_j *and* y_i *are the same variable.*

The profile identifies the parameters of a predicate that are always replaced by a variable $x_1, \ldots, x_{\#\mathsf{A}}$ in each unfolding of $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$, according to the rules in Δ; it is computed by a greatest fixpoint iteration, in time $\mathit{poly}(\mathrm{size}(\Delta))$.

Definition 11. *A rule* $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * \mathbin{\ast\!\!\ast}_{\ell=1}^{h} \mathsf{B}_\ell(z_1^\ell, \ldots, z_{\#\mathsf{B}_\ell}^\ell)$*, where* ϕ *is a quantifier- and predicate-free formula, is said to be:*

1. progressing *if and only if* $\phi = [x_1] * \psi$, *where* ψ *consists of interaction atoms involving* x_1 *and (dis-)equalities, such that* $\bigcup_{\ell=1}^{h} \{z_1^\ell, \ldots, z_{\#\mathsf{B}_\ell}^\ell\} = \{x_2, \ldots, x_{\#\mathsf{A}}\} \cup \{y_1, \ldots, y_m\}$,
2. connected *if and only if, for each* $\ell \in [1, h]$ *there exists an interaction atom in* ψ *that contains both* z_1^ℓ *and a variable from* $\{x_1\} \cup \{x_i \mid i \in \lambda_\Delta(\mathsf{A})\}$,
3. equationally-restricted (e-restricted) *if and only if, for every disequation* $x \neq y$ *from* ϕ, *we have* $\{x, y\} \cap \{x_i \mid i \in \lambda_\Delta(\mathsf{A})\} \neq \emptyset$.

A SID Δ *is* progressing, connected *and* e-restricted *if and only if each rule in* Δ *is* progressing, connected *and* e-restricted, *respectively.*

For example, the SID consisting of the rules from Sect. 1.1, together with rules (4) and (5) is progressing, connected and e-restricted.

We recall that $\mathrm{def}_\Delta(\mathsf{A})$ is the set of rules from Δ that define A and denote by $\mathrm{def}_\Delta^*(\mathsf{A})$ the least superset of $\mathrm{def}_\Delta(\mathsf{A})$ containing the rules that define a predicate from a rule in $\mathrm{def}_\Delta^*(\mathsf{A})$. The following result shows that the entailment problem becomes undecidable as soon as the connectivity condition is even slightly lifted:

Theorem 3. $\mathsf{Entl}[\Delta, \mathsf{A}, \mathsf{B}]$ *is undecidable, even when* Δ *is progressing and e-restricted, and only the rules in* $\mathrm{def}_\Delta^*(\mathsf{A})$ *are connected (the rules in* $\mathrm{def}_\Delta^*(\mathsf{B})$ *may be disconnected).*

On the positive side, we prove that $\mathsf{Entl}[\Delta, \mathsf{A}, \mathsf{B}]$ is decidable, if Δ is progressing, connected and e-restricted, assuming further that $\mathsf{Bnd}[\Delta, \mathsf{A}]$ has a positive answer. In this case, the bound on the degree of the models of $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$ is effectively computable, using the algorithm from Fig. 3 (see Proposition 1 for a cut-off result) and denote by $\mathfrak{B}$ this bound, throughout this section.

The proof uses a reduction of $\mathsf{Entl}[\Delta, \mathsf{A}, \mathsf{B}]$ to a similar problem for SL, showed to be decidable [18]. We recall the definition of SL, interpreted over heaps $\mathsf{h} : \mathbb{C} \rightharpoonup_{fin} \mathbb{C}^{\mathfrak{K}}$, introduced in Sect. 2.3. SL rules are denoted as $\overline{\mathsf{A}}(x_1, \ldots, x_{\#(\overline{\mathsf{A}})}) \leftarrow \phi$, where ϕ is a SL formula, such that $\mathrm{fv}(\phi) \subseteq \{x_1, \ldots, x_{\#(\overline{\mathsf{A}})}\}$ and SL SIDs are denoted as $\overline{\Delta}$. The profile $\lambda_{\overline{\Delta}}$ is defined for SL same as for CL (Definition 10).

Definition 12. *A* SL *rule* $\overline{\mathsf{A}}(x_1, \ldots, x_{\#(\overline{\mathsf{A}})}) \leftarrow \phi$ *from a SID* $\overline{\Delta}$ *is said to be:*

1. progressing *if and only if* $\phi = \exists t_1 \ldots \exists t_m \,.\, x_1 \mapsto (y_1, \ldots, y_{\mathfrak{K}}) * \psi$, *where* ψ *contains only predicate and equality atoms,*
2. connected *if and only if* $z_1 \in \{x_i \mid i \in \lambda_{\overline{\Delta}}(\overline{\mathsf{A}})\} \cup \{y_1, \ldots, y_{\mathfrak{K}}\}$, *for every predicate atom* $\overline{\mathsf{B}}(z_1, \ldots, z_{\#(\overline{\mathsf{B}})})$ *from* ϕ.

Note that the definitions of progressing and connected rules are different for SL, compared to CL (Definition 11); in the rest of this section, we rely on the context to distinguish progressing (connected) SL rules from progressing (connected) CL rules. Moreover, e-restricted rules are defined in the same way for CL and SL (point 3 of Definition 11). A tight upper bound on the complexity of the entailment problem between SL formulæ, interpreted by progressing, connected and e-restricted SIDs, is given below:

Theorem 4 ([18])**.** *The* SL *entailment problem is in* $2^{2^{poly(\mathrm{width}(\overline{\Delta}) \cdot \log \mathrm{size}(\overline{\Delta}))}}$, *for progressing, connected and e-restricted SIDs.*

The reduction of $\mathsf{Entl}[\Delta, \mathsf{A}, \mathsf{B}]$ to SL entailments is based on the idea of viewing a configuration as a logical structure (hypergraph), represented by a undirected *Gaifman graph*, in which every tuple from a relation (hyperedge) becomes a clique [24]. In a similar vein, we encode a configuration, of degree at most $\mathfrak{B}$, by a heap of degree $\mathfrak{K}$ (Definition 13), such that $\mathfrak{K}$ is defined using the following integer function:

$$\mathrm{pos}(i,j,k) \stackrel{\mathsf{def}}{=} 1 + \mathfrak{B} \cdot \sum_{\ell=1}^{j-1} |\tau_\ell| + i \cdot |\tau_j| + k$$

where $\mathsf{Inter} \stackrel{\mathsf{def}}{=} \{\tau_1, \ldots, \tau_M\}$ is the set of interaction types and $Q \stackrel{\mathsf{def}}{=} \{q_1, \ldots, q_N\}$ is the set of states of the behavior $\mathbb{B} = (\mathcal{P}, Q, \rightarrow)$ (Sect. 2). Here $i \in [0, \mathfrak{B}-1]$ denotes an interaction of type $j \in [1, M]$ and $k \in [0, N-1]$ denotes a state. We use M and N throughout the rest of this section, to denote the number of interaction types and states, respectively.

For a set I of interactions, let $\mathsf{Tuples}_I^j(c) \stackrel{\mathsf{def}}{=} \{\langle c_1, \ldots, c_n \rangle \mid (c_1, p_1, \ldots, c_n, p_n) \in I,\ \tau_j = \langle p_1, \ldots, p_n \rangle,\ c \in \{c_1, \ldots, c_n\}\}$ be the tuples of components from an interaction of type τ_j from I, that contain a given component c.

Definition 13. *Given a configuration $\gamma = (\mathcal{C}, I, \rho)$, such that $\delta(\gamma) \leq \mathfrak{B}$, a* Gaifman heap *for γ is a heap $\mathsf{h} : \mathbb{C} \rightharpoonup_{fin} \mathbb{C}^{\mathfrak{K}}$, where $\mathfrak{K} \stackrel{\mathsf{def}}{=} \mathrm{pos}(0, M+1, N)$, $\mathrm{dom}(\mathsf{h}) = \mathrm{nodes}(\gamma)$ and, for all $c_0 \in \mathrm{dom}(\mathsf{h})$, such that $\mathsf{h}(c_0) = \langle c_1, \ldots, c_{\mathfrak{K}} \rangle$, the following hold:*

1. *$c_1 = c_0$ if and only if $c_0 \in \mathcal{C}$,*
2. *for all $j \in [1, M]$, $\mathsf{Tuples}_I^j(c) = \{\mathbf{c}_1, \ldots, \mathbf{c}_s\}$ if and only if there exist integers $0 \leq k_1 < \ldots < k_s < \mathfrak{B}$, such that $\langle \mathsf{h}(c_0) \rangle_{\mathrm{inter}(k_i, j)} = \mathbf{c}_i$, for all $i \in [1, s]$, where $\mathrm{inter}(i, j) \stackrel{\mathsf{def}}{=} [\mathrm{pos}(i-1, j, 0), \mathrm{pos}(i, j, 0)]$ are the entries of the i-th interaction of type τ_j in $\mathsf{h}(c_0)$,*
3. *for all $k \in [1, N]$, we have $\langle \mathsf{h}(c_0) \rangle_{\mathrm{state}(k)} = c_0$ if and only if $\rho(c_0) = q_k$, where the entry $\mathrm{state}(k) \stackrel{\mathsf{def}}{=} \mathrm{pos}(0, M+1, k-1)$ in $\mathsf{h}(c_0)$ corresponds to the state $q_k \in Q$.*

We denote by $\mathbb{G}(\gamma)$ the set of Gaifman heaps for γ.

Intuitively, if h is a Gaifman heap for γ and $c_0 \in \mathrm{dom}(\mathsf{h})$, then the first entry of $\mathsf{h}(c_0)$ indicates whether c_0 is present (condition 1 of Definition 13), the next $\mathfrak{B} \cdot \sum_{j=1}^{M} |\tau_j|$ entries are used to encode the interactions of each type τ_j (condition 2 of Definition 13), whereas the last N entries are used to represent the state of the component (condition 3 of Definition 13). Note that the encoding of configurations by Gaifman heaps is not unique: two Gaifman heaps for the same configuration may differ in the order of the tuples from the encoding of an interaction type and the choice of the unconstrained entries from $\mathsf{h}(c_0)$, for each $c_0 \in \mathrm{dom}(\mathsf{h})$. On the other hand, if two configurations have the same Gaifman heap encoding, they must be the same configuration.

Example 3. Figure 4(b) shows a Gaifman heap for the configuration in Fig. 4(a), where each component belongs to at most 2 interactions of type $\langle out, in \rangle$. ■

We build a SL SID $\overline{\Delta}$ that generates the Gaifman heaps of the models of the predicate atoms occurring in a progressing CL SID Δ. The construction associates to each variable x, that occurs free or bound in a rule from Δ, a unique $\mathfrak{K}$-tuple of variables $\eta(x) \in \mathbb{V}^{\mathfrak{K}}$,

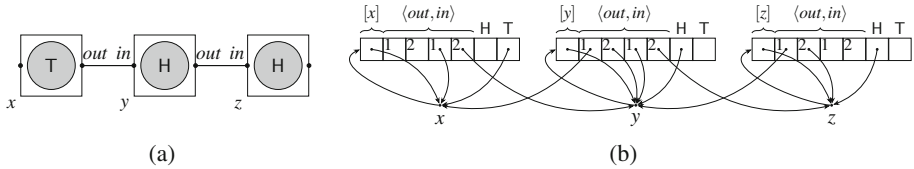

Fig. 4. Gaifman Heap for a Chain Configuration

that represents the image of the store value $\nu(x)$ in a Gaifman heap h i.e., $\mathsf{h}(\nu(x)) = \nu(\eta(x))$. Moreover, we consider, for each predicate symbol $\mathsf{A} \in \mathrm{def}(\Delta)$, an annotated predicate symbol $\overline{\mathsf{A}}_\iota$ of arity $\#\overline{\mathsf{A}}_\iota = (\Re+1)\cdot\#\mathsf{A}$, where $\iota : [1,\#\mathsf{A}] \times [1,M] \to 2^{[0,\mathfrak{B}-1]}$ is a map associating each parameter $i \in [1,\#\mathsf{A}]$ and each interaction type τ_j, for $j \in [1,M]$, a set of integers $\iota(i,j)$ denoting the positions of the encodings of the interactions of type τ_j, involving the value of x_i, in the models of $\overline{\mathsf{A}}_\iota(x_1,\ldots,x_{\#\mathsf{A}},\eta(x_1),\ldots,\eta(x_{\#\mathsf{A}}))$ (point 2 of Definition 13). Then $\overline{\Delta}$ contains rules of the form:

$$\overline{\mathsf{A}}_\iota(x_1,\ldots,x_{\#(\mathsf{A})},\eta(x_1),\ldots,\eta(x_{\#(\mathsf{A})})) \leftarrow \tag{6}$$
$$\exists y_1\ldots\exists y_m\exists\eta(y_1)\ldots\exists\eta(y_m)\ .\ \overline{\psi}*\pi\ *\circledast_{\ell=1}^{h}\ \overline{\mathsf{B}}^{\ell}_{\iota^\ell}(z^\ell_1,\ldots,z^\ell_{\#(\mathsf{B}^\ell)},\eta(z^\ell_1),\ldots,\eta(z^\ell_{\#(\mathsf{B}^\ell)}))$$

for which Δ has a *stem rule* $\mathsf{A}(x_1,\ldots,x_{\#(\mathsf{A})}) \leftarrow \exists y_1\ldots\exists y_m\ .\ \psi*\pi*\circledast_{\ell=1}^{h}\mathsf{B}^\ell(z^\ell_1,\ldots,z^\ell_{\#\mathsf{B}^\ell})$, where $\psi*\pi$ is a quantifier- and predicate-free formula and π is the conjunction of equalities and disequalities from $\psi*\pi$. However, not all rules (6) are considered in $\overline{\Delta}$, but only the ones meeting the following condition:

Definition 14. *A rule of the form (6) is* well-formed *if and only if, for each $i \in [1,\#\mathsf{A}]$ and each $j \in [1,M]$, there exists a set of integers $Y_{i,j} \subseteq [0,\mathfrak{B}-1]$, such that:*

- $\|Y_{i,j}\| = \|I^j_{\psi,\pi}(x_i)\|$, *where $I^j_{\psi,\pi}(x)$ is the set of interaction atoms $\langle z_1.p_1,\ldots,z_n.p_n\rangle$ from ψ of type $\tau_j = \langle p_1,\ldots,p_n\rangle$, such that $z_s \approx_\pi x$, for some $s \in [1,n]$,*
- $Y_{i,j} \subseteq \iota(i,j)$ *and* $\iota(i,j)\backslash Y_{i,j} = \mathcal{Z}_j(x_i)$, *where* $\mathcal{Z}_j(x) \stackrel{\mathrm{def}}{=} \bigcup_{\ell=1}^{h}\bigcup_{k=1}^{\#\mathsf{B}^\ell}\{\iota^\ell(k,j) \mid x \approx_\pi z^\ell_k\}$ *is the set of positions used to encode the interactions of type τ_j involving the store value of the parameter x, in the sub-configuration corresponding to an atom $\mathsf{B}_\ell(z^\ell_1,\ldots,z^\ell_{\#(\mathsf{B}^\ell)})$, for some $\ell \in [1,h]$.*

We denote by $\overline{\Delta}$ the set of well-formed rules (6), such that, moreover:

$$\overline{\psi} \stackrel{\mathrm{def}}{=} x_1 \mapsto \eta(x_1)\ *\ \circledast_{x\in\mathrm{fv}(\psi)}\ \mathsf{CompStates}_\psi(x)\ *\ \circledast_{i=1}^{\#\mathsf{A}}\ \mathsf{InterAtoms}_\psi(x_i), \text{ where:}$$
$$\mathsf{CompStates}_\psi(x) \stackrel{\mathrm{def}}{=} \circledast_{[x]\text{ occurs in }\psi}\ \langle\eta(x)\rangle_1 = x\ *\ \circledast_{x@q_k\text{ occurs in }\psi}\ \langle\eta(x)\rangle_{\mathrm{state}(k)} = x$$
$$\mathsf{InterAtoms}_\psi(x_i) \stackrel{\mathrm{def}}{=} \circledast_{j=1}^{M}\circledast_{p=1}^{r_j}\ \langle\eta(x_i)\rangle_{\mathrm{inter}(j,k^j_p)} = \mathbf{x}^j_p \text{ and } \{k^j_1,\ldots,k^j_{r_j}\} \stackrel{\mathrm{def}}{=} \iota(i,j)\backslash\mathcal{Z}_j(x_i)$$

Here for two tuples of variables $\mathbf{x} = \langle x_1,\ldots,x_k\rangle$ and $\mathbf{y} = \langle y_1,\ldots,y_k\rangle$, we denote by $\mathbf{x} = \mathbf{y}$ the formula $\circledast_{i=1}^{k} x_i = y_i$. Intuitively, the SL formula $\mathsf{CompStates}_\psi(x)$ realizes the encoding of the component and state atoms from ψ, in the sense of points (1) and (3) from Definition 13, whereas the formula $\mathsf{InterAtoms}_\psi(x_i)$ realizes the encodings of

the interactions involving a parameter x_i in the stem rule (point 2 of Definition 13). In particular, the definition of $\mathsf{InterAtoms}_\psi(x_i)$ uses the fact that the rule is well-formed.

We state below the main result of this section on the complexity of the entailment problem. The upper bounds follow from a many-one reduction of $\mathsf{Entl}[\Delta, \mathsf{A}, \mathsf{B}]$ to the SL entailment $\overline{\mathsf{A}}_\iota(x_1, \ldots, x_{\#\mathsf{A}}, \eta(x_1), \ldots, \eta(x_{\#\mathsf{A}})) \Vdash_{\overline{\Delta}} \exists x_{\#\mathsf{B}+1} \ldots \exists x_{\#\mathsf{B}}$ $\exists \eta(x_{\#\mathsf{B}+1}) \ldots \exists \eta(x_{\#\mathsf{B}})$. $\overline{\mathsf{B}}_{\iota'}(x_1, \ldots, x_{\#\mathsf{B}}, \eta(x_1), \ldots, \eta(x_{\#\mathsf{B}}))$, in combination with the upper bound provided by Theorem 4, for SL entailments. If $k < \infty$, the complexity is tight for CL, whereas gaps occur for $k = \infty, \ell < \infty$ and $k = \infty, \ell = \infty$, due to the cut-off on the degree bound (Proposition 1), which impacts the size of $\overline{\Delta}$ and time needed to generate it from Δ.

Theorem 5. *If Δ is progressing, connected and e-restricted and, moreover,* $\mathsf{Bnd}[\Delta, \mathsf{A}]$ *has a positive answer,* $\mathsf{Entl}^{k,\ell}[\Delta, \mathsf{A}, \mathsf{B}]$ *is in* 2EXP, $\mathsf{Entl}^{\infty,\ell}[\Delta, \mathsf{A}, \mathsf{B}]$ *is in* 3EXP $\cap$ 2EXP-*hard, and* $\mathsf{Entl}[\Delta, \mathsf{A}, \mathsf{B}]$ *is in* 4EXP $\cap$ 2EXP-*hard.*

6 Conclusions and Future Work

We study the satisfiability and entailment problems in a logic used to write proofs of correctness for dynamically reconfigurable distributed systems. The logic views the components and interactions from the network as resources and reasons also about the local states of the components. We reuse existing techniques for Separation Logic [39], showing that our configuration logic is more expressive than SL, fact which is confirmed by a number of complexity gaps. Closing up these gaps and finding tight complexity classes in the more general cases is considered for future work. In particular, we aim at lifting the boundedness assumption on the degree of the configurations that must be considered to check the validity of entailments.

References

1. Ahrens, E., Bozga, M., Iosif, R., Katoen, J.: Local reasoning about parameterized reconfigurable distributed systems. CoRR, abs/2107.05253 (2021)
2. Arbab, F.: Reo: a channel-based coordination model for component composition. Math. Struct. Comput. Sci. **14**(3), 329–366 (2004)
3. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM 2006), pp. 3–12. IEEE Computer Society (2006)
4. Bloem, R., et al.: Decidability of Parameterized Verification. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers (2015)
5. Bozga, M., Bueri, L., Iosif, R.: Decision problems in a logic for reasoning about reconfigurable distributed systems. CoRR, abs/2202.09637 (2022)
6. Bozga, M., Iosif, R., Sifakis, J.: Verification of component-based systems with recursive architectures. CoRR, abs/2112.08292 (2021)
7. Bradbury, J., Cordy, J., Dingel, J., Wermelinger, M.: A survey of self-management in dynamic software architecture specifications. In: Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems, pp. 28–33. ACM (2004)
8. Brookes, S., O'Hearn, P.W.: Concurrent separation logic. ACM SIGLOG News **3**(3), 47–65 (2016)

9. Brotherston, J., Fuhs, C., Pérez, J.A.N., Gorogiannis, N.: A decision procedure for satisfiability in separation logic with inductive predicates. In: CSL-LICS, pp. 25:1–25:10. ACM (2014)
10. Bucchiarone, A., Galeotti, J.P.: Dynamic software architectures verification using dynalloy. Electron. Commun. Eur. Assoc. Softw. Sci. Technol. **10** (2008). https://doi.org/10.14279/tuj.eceasst.10.145
11. Butting, A., Heim, R., Kautz, O., Ringert, J.O., Rumpe, B., Wortmann, A.: A classification of dynamic reconfiguration in component and connector architecture description. In: Proceedings of MODELS 2017 Satellite Event: Workshops (ModComp). CEUR Workshop Proceedings, vol. 2019, pp. 10–16. CEUR-WS.org (2017)
12. Calcagno, C., O'Hearn, P.W., Yang, H.: Local action and abstract separation logic. In: 22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10–12 July 2007, Wroclaw, Poland, Proceedings, pp. 366–378. IEEE Computer Society (2007)
13. Cavalcante, E., Batista, T.V., Oquendo, F.: Supporting dynamic software architectures: from architectural description to implementation. In: Bass, L., Lago, P., Kruchten, P. (eds.) 12th Working IEEE/IFIP Conference on Software Architecture, WICSA 2015, pp. 31–40. IEEE Computer Society (2015)
14. Clarke, D.: A basic logic for reasoning about connector reconfiguration. Fundam. Inf. **82**(4), 361–390 (2008)
15. Dinsdale-Young, T., Birkedal, L., Gardner, P., Parkinson, M., Yang, H.: Views: compositional reasoning for concurrent programs. SIGPLAN Not. **48**(1), 287–300 (2013)
16. Dinsdale-Young, T., Dodds, M., Gardner, P., Parkinson, M.J., Vafeiadis, V.: Concurrent abstract predicates. In: D'Hondt, T. (ed.) ECOOP 2010. LNCS, vol. 6183, pp. 504–528. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14107-2_24
17. Dormoy, J., Kouchnarenko, O., Lanoix, A.: Using temporal logic for dynamic reconfigurations of components. In: Barbosa, L.S., Lumpe, M. (eds.) FACS 2010. LNCS, vol. 6921, pp. 200–217. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27269-1_12
18. Echenim, M., Iosif, R., Peltier, N.: Unifying decidable entailments in separation logic with inductive definitions. In: Platzer, A., Sutcliffe, G. (eds.) CADE 2021. LNCS (LNAI), vol. 12699, pp. 183–199. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79876-5_11
19. El-Ballouli, R., Bensalem, S., Bozga, M., Sifakis, J.: Programming dynamic reconfigurable systems. Int. J. Softw. Tools Technol. Transf. **23**, 701–719 (2021)
20. El-Hokayem, A., Bozga, M., Sifakis, J.: A temporal configuration logic for dynamic reconfigurable systems. In: Hung, C., Hong, J., Bechini, A., Song, E. (eds.) SAC 2021: The 36th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, Republic of Korea, 22–26 March 2021, pp. 1419–1428. ACM (2021)
21. Farka, F., Nanevski, A., Banerjee, A., Delbianco, G.A., Fábregas, I.: On algebraic abstractions for concurrent separation logics. Proc. ACM Program. Lang. **5**(POPL), 1–32 (2021)
22. Feng, X., Ferreira, R., Shao, Z.: On the relationship between concurrent separation logic and assume-guarantee reasoning. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 173–188. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71316-6_13
23. Foerster, K., Schmid, S.: Survey of reconfigurable data center networks: enablers, algorithms, complexity. SIGACT News **50**(2), 62–79 (2019)
24. Gaifman, H.: On local and non-local properties. Stud. Log. Found. Math. **107**, 105–135 (1982)
25. Gunawi, H.S., et al.: Why does the cloud stop computing? Lessons from hundreds of service outages. In: Proceedings of the Seventh ACM Symposium on Cloud Computing, SoCC 2016, pp. 1–16. Association for Computing Machinery, New York (2016)
26. Hirsch, D., Inverardi, P., Montanari, U.: Graph grammars and constraint solving for software architecture styles. In: Proceedings of the Third International Workshop on Software Architecture, ISAW 1998, pp. 69–72. Association for Computing Machinery, New York (1998)

27. Jansen, C., Katelaan, J., Matheja, C., Noll, T., Zuleger, F.: Unified reasoning about robustness properties of symbolic-heap separation logic. In: Yang, H. (ed.) ESOP 2017. LNCS, vol. 10201, pp. 611–638. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54434-1_23
28. Jones, C.B.: Developing methods for computer programs including a notion of interference. Ph.D. thesis, University of Oxford, UK (1981)
29. Konnov, I.V., Kotek, T., Wang, Q., Veith, H., Bliudze, S., Sifakis, J.: Parameterized systems in BIP: design and model checking. In: 27th International Conference on Concurrency Theory, CONCUR 2016, volume 59 of LIPIcs, pp. 30:1–30:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016)
30. Krause, C., Maraikar, Z., Lazovik, A., Arbab, F.: Modeling dynamic reconfigurations in Reo using high-level replacement systems. Sci. Comput. Program. **76**, 23–36 (2011)
31. Lanoix, A., Dormoy, J., Kouchnarenko, O.: Combining proof and model-checking to validate reconfigurable architectures. Electron. Notes Theor. Comput. Sci. **279**(2), 43–57 (2011)
32. Le Metayer, D.: Describing software architecture styles using graph grammars. IEEE Trans. Softw. Eng. **24**(7), 521–533 (1998)
33. Magee, J., Kramer, J.: Dynamic structure in software architectures. In: ACM SIGSOFT Software Engineering Notes, vol. 21, no. 6, pp. 3–14. ACM (1996)
34. Mavridou, A., Baranov, E., Bliudze, S., Sifakis, J.: Configuration logics: modeling architecture styles. J. Log. Algebr. Meth. Program. **86**(1), 2–29 (2017)
35. Noormohammadpour, M., Raghavendra, C.S.: Datacenter traffic control: understanding techniques and tradeoffs. IEEE Commun. Surv. Tutor. **20**(2), 1492–1525 (2018)
36. O'Hearn, P.W.: Resources, concurrency, and local reasoning. Theor. Comput. Sci. **375**(1–3), 271–307 (2007)
37. O'Hearn, P.W., Pym, D.J.: The logic of bunched implications. Bull. Symb. Log. **5**(2), 215–244 (1999)
38. Owicki, S., Gries, D.: An axiomatic proof technique for parallel programs. In: Gries, D. (ed.) Programming Methodology. Texts and Monographs in Computer Science, pp. 130–152. Springer, New York (1978). https://doi.org/10.1007/978-1-4612-6315-9_12
39. Reynolds, J.C.: Separation logic: a logic for shared mutable data structures. In: Proceedings of 17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22–25 July 2002, Copenhagen, Denmark, pp. 55–74. IEEE Computer Society (2002)
40. Shtadler, Z., Grumberg, O.: Network grammars, communication behaviors and automatic verification. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 151–165. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-52148-8_13
41. Taentzer, G., Goedicke, M., Meyer, T.: Dynamic change management by distributed graph transformation: towards configurable distributed systems. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) TAGT 1998. LNCS, vol. 1764, pp. 179–193. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-540-46464-8_13
42. Vafeiadis, V., Parkinson, M.: A marriage of rely/guarantee and separation logic. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 256–271. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74407-8_18
43. Wermelinger, M.: Towards a chemical model for software architecture reconfiguration. IEE Proc.-Softw. **145**(5), 130–136 (1998)
44. Wermelinger, M., Fiadeiro, J.L.: A graph transformation approach to software architecture reconfiguration. Sci. Comput. Program. **44**(2), 133–155 (2002)