Local Reasoning for Reconfigurable Distributed Systems

Bachelor Thesis Computer Science

Emma Ahrens

March 5, 2021

Supervised by Dr. Radu Iosif*
and Prof. Dr. Joost-Pieter Katoen

Submitted to Lehrstuhl i2: Software Modeling and Verification,
RWTH Aachen University

*With support from Dr. Marius Bozga.

Abstract

This thesis investigates the use of separation logic for reasoning about dynamically reconfigurable behavior, interaction, priority (DR-BIP) systems and allows the verification of reconfiguration programs on certain distributed systems. Separation logic extends Hoare logic to enable the verification of programs on resources. It makes use of local reasoning, which is primarily enabled by a frame rule. Static BIP systems represent component-based systems, where a fixed number of components encapsulate behavior specified as transition systems and interact via a fixed set of multi-party interactions (synchronizations) that are executed atomically and non-deterministically. The extension to DR-BIP systems additionally allows dynamic reconfiguration through addition and/or removal of interactions and components during runtime.

This work defines BIP configurations that represent a current state in a BIP system and then gives a separation logic that is evaluated on the BIP configurations. Furthermore, a reconfiguration language is specified that allows the definition of reconfiguration programs on those BIP configurations. We give axioms and inference rules that enable us to check the partial correctness of reconfiguration programs in a Hoare calculus-style. Because these programs are not usually local, the main challenge is the definition of a frame rule. For that purpose, we generalize the notion of locality and reason about the possible state changes in a system.

As an application example of the theory, we give reconfiguration rules on parametric token rings and prove their correctness.

Contents

List of Figures vii

Notations ix

1. Introduction 1

2. Separation Algebra of BIP Configurations 5
2.1. Theory of Abstract Separation Logic 5
2.2. BIP Configurations 6
2.3. State Transitions for BIP Configurations 16

3. Separation Logic on BIP 21

4. Reconfiguration Language for BIP 29

5. Rules for the Verification of Reconfigurations 37
5.1. Reconfiguration Rules 37
5.2. Relaxing Locality 42
5.3. Havoc Rules 46
5.4. Frontier and Composition Rule 53

6. Reconfigurations on Token Rings 59
6.1. BIP Configurations, Predicates, and Programs 59
6.2. Proofs of the Reconfiguration Programs 62

7. Conclusion 69

Bibliography 71

A. X-Locality 73

B. Proofs for Token Ring Example 77
B.1. Implications 77
B.2. Invariance Under Havoc 81

List of Figures

1.1. Token Ring as BIP Configuration 2

2.1. Table of Dining Philosophers 8
2.2. Component Type PHILO 9
2.3. Component Type FORK 10
2.4. Table of Dining Philosophers Represented As BIP System 12
2.5. Table of Dining Philosophers Represented as BIP Configuration 14
2.6. Composition of BIP Configurations 16
2.7. Philosopher and Forks with States 17
2.8. Philosopher and Forks with Changed States 18

3.1. Seat of a Philosopher 27
3.2. Chain as a Recursive Predicate 27

4.1. Reconfigurations on BIP Configurations 34
4.2. Two Components and an Enabled Interaction 35

5.1. Open Semantics 47
5.2. Frontier of a BIP Configuration 54

6.1. Component Type in Token Ring 60
6.2. Valid Token Ring 61

Notations

Symbols	Description
	Abstract Separation Algebra
Σ	arbitrary set
$(\Sigma, \bullet, E)$	separation algebra
$\sigma, \sigma_0, \sigma_1, \sigma' \in \Sigma$	elements in Σ
$p, q, \mathtt{emp} \in \mathcal{P}(\Sigma)$	predicates
$\mathcal{P}(\Sigma)^\top$	set of predicates extended by new greatest predicate $\top$
$f : \Sigma \to \mathcal{P}(\Sigma)^\top$	function
	BIP Configurations
$\mathcal{U}$	countably infinite set of indices
$u_1, \ldots, u_{\alpha(j)}, u_k \in \mathcal{U}$	indices of components
$\mathcal{V}$	countably infinite set of variables
$x_1, \ldots, x_{\alpha(j)}, y_1, \ldots, y_{\alpha(j)} \in \mathcal{V}$	variables
$\mathcal{P}$	countably infinite set of predicate symbols
C	countably finite set of constant symbols
$C = \{C_1, \ldots, C_n\}$	component symbols
$I = \{I_n, \ldots, I_m\}$	interaction symbols
$\alpha(I_j)$	arity of interaction symbol for $I_j \in I$
$\langle C, I \rangle$	signature with component and interaction symbols
$\mathbb{S}_i$	set of states of component type $C_i \in C$
$s_i^0 \in \mathbb{S}_i$	initial state of component type $C_i \in C$
$s_i, s_i' \in \mathbb{S}_i$	states of component type $C_i \in C$
$\mathbb{P}$	set of ports of component type $C_i \in C$
$I_j^i \in \mathbb{P}_i$	port of component type $C_i \in C$
$\rightsquigarrow_i$	transition function of component type $C_i \in C$
$\mathfrak{S}$	BIP system
$C_i^{\mathfrak{S}} \subseteq \mathcal{U}$	interpretation of component symbol as relation
$I_j^{\mathfrak{S}} \subseteq \mathcal{U}^{\alpha(I_j)}$	interpretation of interaction symbol as relation
$\varsigma, \varsigma' : \mathcal{U} \times C \to \mathbb{S}$	state snapshot

$\nu : \mathcal{V} \to \mathcal{U}$	variable mapping
$(\mathfrak{S}, \varsigma, \nu)$	BIP configuration
$\Sigma_{\langle C,I\rangle}$	set of BIP configurations for a fixed signature
$\Sigma^{\emptyset}_{\langle C,I\rangle}$	set of units
$(\Sigma_{\langle C,I\rangle}, \bullet, \Sigma^{\emptyset}_{\langle C,I\rangle})$	multi-unit separation algebra
Σ	sets of interaction atoms
$I_j(x_1, \ldots, x_{\alpha(j)})$	interaction atom for $I_j \in I$ and $x_1, \ldots, x_{\alpha(j)} \in \mathcal{V}$
$\rightsquigarrow_c : \Sigma_{\langle C,I\rangle} \times \Sigma \to \Sigma_{\langle C,I\rangle}$	transitions in concrete semantics

Separation Logic on BIP

$\mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$	separation logic on BIP
$\phi, \psi \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$	formulae in separation logic
$\mathrm{fv}(\psi) \subset \mathcal{V}$	free variables of a formula
$A \in \mathcal{P}$	predicate symbol with arity $\alpha(A)$
$t_1, \ldots, t_{\alpha(A)} \in \mathcal{V} \cup \mathcal{C}$	terms that are either variables or constants
$A \in \mathcal{P}$	predicate symbol
$\mathcal{R}$	set of rules for system of inductive definitions
$\leftarrow_{\mathcal{R}}$	unfolding of predicate symbol via rule in $\mathcal{R}$
$\theta : X \to \mathcal{V}$	injective substitution of variables $X \subset \mathcal{V}$
$\Gamma(\psi)$	set of component atoms in ψ
$C_i(x_i)$	component atom for $C_i \in C$ and $x_i \in \mathcal{V}$
$\Sigma(\psi)$	set of interaction atoms in ψ
$I_j(x_1, \ldots, x_{\alpha(j)})$	interaction atom for $I_j \in I$ and $x_1, \ldots, x_{\alpha(j)} \in \mathcal{V}$
$\mathcal{A}, \mathcal{B} \in \Sigma(\psi)$	other interaction atoms

Reconfiguration Language on BIP

$\mathcal{L}\langle C, I\rangle$	reconfiguration language on BIP
$\ell, \ell' \in \mathcal{L}\langle C, I\rangle$	reconfiguration programs
$\phi \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$	downward closed $\langle C, I\rangle$-formula
$\psi \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$	arbitrary $\langle C, I\rangle$-formula
$\mathrm{Modifies}(\ell)$	modified variables for a program $\ell \in \mathcal{L}\langle C, I\rangle$
`error`	represents an error state
$\top = p \cup \{\texttt{error}\}$	predicate that contains error state and $p \in \mathcal{P}(\Sigma_{\langle C,I\rangle})$
$\mathcal{P}(\Sigma_{\langle C,I\rangle})^{\top}$	set of predicates together with additional predicate $\top$
$[\![\ell]\!] : \Sigma_{\langle C,I\rangle} \to \mathcal{P}(\Sigma_{\langle C,I\rangle})^{\top}$	valuation of program $\ell \in \mathcal{L}\langle C, I\rangle$

Reconfiguration Rules

$P, P', P_i, Q, Q', Q_i, R, F \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$	pre- and postconditions
$\phi \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$	downward closed $\langle C, I\rangle$-formula
$\psi \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$	$\langle C, I\rangle$-formula
$\ell, \ell_0, \ell_1 \in \mathcal{L}\langle C, I\rangle$	program
Ax	set of axioms
$\mathcal{L}_X\langle C, I\rangle$	subset of all programs without `with` ψ `do`, sequential composition and the Kleene star
$P{\uparrow_X}$	lift of $P \in \mathcal{P}(\Sigma_{\langle C,I\rangle})$ on $X \subset \mathcal{V} \times C$

Havoc Rules

$\rightsquigarrow_o \colon \Sigma_{\langle C,I\rangle} \times \Sigma \to \Sigma_{\langle C,I\rangle}$	transitions in open semantics
L, L_1, L_2	language expression
$\mathrm{supp}(L)$	support alphabet of a language expression L
$[\![L]\!]$	valuation of a language expression L
$w \in [\![L]\!]$	word in a language, trace
$P \dagger \mathcal{A}$	interaction atom $\mathcal{A}$ is disabled for $P \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$
$\equiv_\Gamma$	equivalence relation on interaction atoms
$[\mathcal{A}]_\Gamma$	representative of equivalence class for an interaction atom $\mathcal{A}$
$\mathcal{F}_{P_2}(\theta, P_1)$	frontier of P_1 for P_2
$\tilde{L}$	language expression, where interaction atoms are substituted by its equivalence class representatives

1. Introduction

Software surrounds us in almost all aspects of our lives and its importance increases every day. At the same time, the costs of software errors are also increasing [ZC09]. Meanwhile, software projects become more and more complex, which makes it harder to oversee them. To find errors, big software companies, like facebook, have recently successfully applied automated proof tools to verify code using *separation logic* [O'H19].

Separation Logic The idea is to specify pre- and postconditions P and Q and prove that the execution of a program C on a system that fulfills P leads to a system in state Q. This can be expressed by a *Hoare triple*

$$\{ P \}\ C\ \{ Q \}.$$

This notation, together with a set of proof rules, was first proposed by C. A. R. Hoare in [Hoa69] and is called *Hoare logic*. It enables the verification of programs without pointers, where pre- and postconditions are formulae in first-order logic.

In order to prove the correctness of programs with pointers, John C. Reynolds proposed *separation logic* [Rey02] as an extension of Hoare logic. He assumes that pointers map to cells in a heap and introduces a spatial conjunction $*$, where $P * Q$ states that the property P holds on one part of the heap and Q on the other. Programs with pointers (which are also called programs on heaps) only change certain cells of the heap and the rest remains unaltered. This enables *local reasoning*, which was formalized by Peter O'Hearn, John C. Reynolds and Hongseok Yang in [ORY01] in the *frame rule*, which is an inference rule that states

$$\frac{\{ P \}\ C\ \{ Q \}}{\{ P * F \}\ C\ \{ Q * F \}.}$$

Hence, if a program C is executed on a model of P and the resulting heap models Q, then the execution of the same program C on a model of $P * F$ (hence a greater heap, where a part models P) results in a model of $Q * F$ for a formula F. This states that we can enlarge a heap that models P arbitrarily and the program C still changes the same specific cells.

Separation logic was adapted for the reasoning and verification of programs on other resources. In [COY07], this was formalized as *abstract separation logic* and abstract inference rules were given for any programs and resources that allow local reasoning, where the programs are *local actions* and the resources are *separation algebras*.

BIP This work aims to provide the tools to verify programs that alter a certain type of distributed systems. As resources, we specify *BIP configurations*, which are inspired by

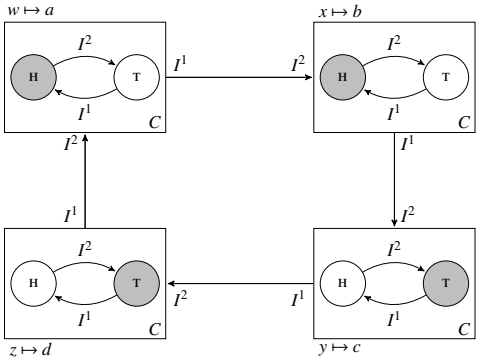

Figure 1.1.: Token Ring as BIP Configuration

the architecture description language BIP (behavior, interaction, priority framework) that can be used to describe component-based distributed systems [BBB+11]. A BIP configuration represents *components* that are connected via *interactions*. Each component contains a finite-state transition system and a set of ports (both of which are specified via the type of the component). An interaction connects a finite number of ports of components, where the ports are specified by the interaction type. Interactions may fire non-deterministically. If they trigger, then all the connected components change their state via a transition in their transition system.

A token ring is a typical example of a system that can be modeled via BIP. It contains a finite number of components that are connected according to a ring topology. Each component has two possible states; either it has a token (T) or it does not (H). A token may be passed from a component with a token to a component without a token and thereby both components change their state accordingly. See Figure 1.1 for an illustration as a BIP configuration.

An extension of BIP is DR-BIP [BBBS18], which adds *reconfiguration programs* that additionally specify how and if a distributed system can be reconfigured. In the token ring example, such a reconfiguration program might specify that a component in the ring is only deleted if the resulting ring remains deadlock-free.

Contribution We propose a *reconfiguration language on BIP* that allows us to define programs on BIP configurations. Furthermore, we define a *separation logic on BIP (SL on BIP)* that enables us to specify properties of BIP configurations. We want to prove the correctness of the reconfiguration programs on BIP configurations and specify a set of inference rules for that purpose. Those rules are used to prove the correctness of a Hoare triple $\{ P \}\ C\ \{ Q \}$ for pre- and postconditions P and Q written as formulae in SL on BIP and a program C that is given via the reconfiguration language on BIP.

For the token ring example, we may specify, both as pre- and postcondition, that the ring consists of at least one component in state T and one component in state H. Furthermore, we might define a program for the deletion of a component in the ring that deletes a component

only if there exists at least one other component in the same state.

The logic and the reconfiguration language are both inspired by abstract separation logic [COY07], but we cannot apply the theory directly, because the atomic reconfigurations of the reconfiguration language on BIP are not local actions. By relaxing the notion of *locality*, we circumvent this problem. Furthermore, abstract separation logic is intended for static resources, whereas we analyze concurrent BIP configurations with interactions that trigger arbitrarily. We assume that composed programs are not atomic and that the states of components may change in between the execution of two commands. Hence, the inference rule for sequential composition of two programs differs from the associated abstract rule, because the state changes need to be taken into account. For that purpose, we define an additional set of rules that reasons only about the firing of interactions and the associated state changes. Finally, we use two different sets of inference rules; the *reconfiguration rules* resemble the inference rules of abstract separation logic and are used for reasoning about static BIP configurations and the *havoc rules* for reasoning about the possible state changes.

The reconfiguration rules contain a frame rule that enables local reasoning and we want to reuse this concept for the havoc rules. Suppose that a token ring is split into two non-empty chains of components and one chain has no tokens and the other chain contains at least one. There are no possible state changes in the first chain, but the tokens in the whole ring could move an arbitrary number of times. Thus, we cannot reason about the state changes of a whole system simply by looking at its parts separately. We solve this problem by adding the incoming and outgoing interactions for the first chain and reason about the chain together with those interactions. Then, we can assume that an arbitrary number of tokens get into the chain and obtain a superset of the possible state changes. This is done on both parts and finally, the possible state changes of both parts are interwoven. In general, we define a *frontier* that contains the interactions of a part of the BIP configuration that connect components of the current part. Then we reason about each part together with its frontier and interweave the traces of state changes. Using this, we obtain a way to reason locally.

Organization This thesis starts by defining BIP configurations, which we use as models for our logic, in Chapter 2. Chapter 3 specifies the separation logic on BIP and Chapter 4 defines the reconfiguration language on BIP. The *reconfiguration rules* on static BIP configurations and the *havoc rules* on concurrent BIP configurations are given in Chapter 5. Last but not least we illustrate our verification method on the *token ring* example in Chapter 6. We define reconfiguration programs on the token ring and prove their correctness using the reconfiguration and havoc rules.

2. Separation Algebra of BIP Configurations

Calcagno, O'Hearn and Yang [COY07] defined *separation algebras*, which are cancellative, partial commutative monoids. They showed that the elements in an arbitrary separation algebra can be used as models for a generic separation logic. They defined the syntax and semantics of such a logic and additionally defined a programming language that alters the elements in a separation algebra using special relations, so-called *local actions*. This notion was refined by Dockins, Hobor and Appel in [COY07], where they defined *multi-unit separation algebras* that allow multiple units.

In the first section of this chapter, we summarize the theory of *separation algebras* and in the second section, we define *BIP configurations* and show that they form a separation algebra for any fixed set of component and interaction types.

2.1. Theory of Abstract Separation Logic

This section summarizes the theory of *(multi-unit) separation algebras* given in [COY07] and [DHA09]. The definitions are extractions of those two papers.

Definition 1 (Multi-unit Separation Algebra, [COY07], [DHA09])**.** *A* multi-unit separation algebra *is a cancellative, partial commutative monoid* $(\Sigma, \bullet, E)$*, where the elements in* $E \subseteq \Sigma$ *are* units *satisfying that*

$$\textit{for each } \sigma \in \Sigma \textit{ there exists exactly one } \sigma_E \in E \textit{ such that } \sigma \bullet \sigma_E = \sigma.$$

A partial commutative monoid is given by a partial binary operation where the unity, commutativity and associativity laws hold for the equality, that means both sides are defined and equal, or both are undefined. The cancellative property says that for each $\sigma \in \Sigma$*, the partial function* $\sigma \bullet (\cdot) : \Sigma \to \Sigma$ *is injective. The induced* separateness *(#) and* substate *(*$\preceq$*) relations are given by*

$$\begin{aligned} \sigma_0 \# \sigma_1 \quad &\textit{iff} \quad \sigma_0 \bullet \sigma_1 \textit{ is defined}, \\ \sigma_0 \preceq \sigma_2 \quad &\textit{iff} \quad \exists \sigma_1 . \sigma_2 = \sigma_0 \bullet \sigma_1 . \end{aligned}$$

We define our multi-unit separation algebra to be disjoint*, thus*

$$\sigma_0 \bullet \sigma_0 = \sigma_1 \quad \textit{implies} \quad \sigma_0 = \sigma_1 .$$

We want to define a programming language on elements in a separation algebra. For a

program P we assume pre- and postconditions and they can be expressed by subsets of the separation algebra, which we call *predicates*.

Definition 2 (Predicate, [COY07])**.** *Let* $(\Sigma, \bullet, E)$ *be a separation algebra.* Predicates *over* Σ *are elements of the powerset* $\mathcal{P}(\Sigma)$*. The set of predicates* $\mathcal{P}(\Sigma)$ *has an ordered total commutative monoid structure* $(*, \texttt{emp})$ *given by*

$$p * q = \{\sigma_0 \bullet \sigma_1 \mid \sigma_0 \# \sigma_1 \wedge \sigma_0 \in p \wedge \sigma_1 \in q\} \text{ and } \texttt{emp} := E$$

for $p, q \in \mathcal{P}(\Sigma)$ *and* `emp` *is the predicate of units. The elements in the power set* $\mathcal{P}(\Sigma)$ *are ordered using the subset relation.*

The empty predicate contains only the unit elements of the separation algebra. For the separation algebra on heaps, `emp` would contain only the empty heap. We want to add a predicate that we use to denote errors later.

Definition 3 ([COY07])**.** *The set* $\mathcal{P}(\Sigma)^\top$ *is obtained by adding a new greatest element* $\top$ *to* $\mathcal{P}(\Sigma)$*. It has a total commutative monoid structure, keeping the unit* `emp` *the same as in* $\mathcal{P}(\Sigma)$*, and extending* $*$ *so that* $p * \top = \top * p = \top$ *for all* $p \in \mathcal{P}(\Sigma)^\top$*. The subset relation is extended such that* $p \subset \top$ *holds for every predicate* $p \in \mathcal{P}(\Sigma)$*.*

A key property of separation logic is local reasoning. O'Hearn, Reynolds and Yang summarized this in the following way in [ORY01] for separation logic on heaps:

> "To understand how a program works, it should be possible for reasoning and specification to be confined to cells that the program actually accesses. The value of any other cell will automatically remain unchanged."

Assume a program that alters a heap by writing the number 5 into the first cell (and assume that the cells are numbered). If we execute this program on any heap, then only the first cell is changed and the rest of the cells stay the same. If the rest of the heap satisfies a certain property before the execution, then it still satisfies this property afterwards.

Programs that behave like this are called *local*, since they alter only certain parts (in the case of heaps they alter only certain cells). A counterexample would be a program that sets the content of every cell in the given heap to zero. This program does not act local, but rather global.

Definition 4 (Local Action, [COY07])**.** *Suppose* $(\Sigma, \bullet, E)$ *is a separation algebra. A* local action $f : \Sigma \to \mathcal{P}(\Sigma)^\top$ *is a function satisfying the* locality condition*:*

$$\sigma_0 \# \sigma_1 \quad \textit{implies} \quad f(\sigma_0 \bullet \sigma_1) \subseteq (f(\sigma_0)) * \{\sigma_1\},$$

where $\sigma_0, \sigma_1 \in \Sigma$*. We call* $\mathrm{LocAct}(\Sigma, \bullet, E)$ *the set of local actions over the separation algebra* $(\Sigma, \bullet, E)$ *and note that it can be ordered pointwise.*

2.2. BIP Configurations

The BIP framework [BBB+11] is used to design component-based systems and models systems built of *components*, and *interactions* that connect those components. Components

and interactions are always instances of some component respectively interaction type.

Components have an internal *behavior*, which is described by a finite-state transition system. The type of each component determines its behavior. Furthermore, the component type specifies a number of *ports*. Those ports are connected to ports of other components by interactions. Each interaction may connect only designated ports, and their types and the number of ports is determined by the interaction type.

Throughout this section, we define *BIP systems* and *BIP configurations* that represent simplified versions of settings in BIP frameworks. The BIP configurations represent states of BIP systems and will be used as models of the separation logic on BIP in further chapters.

The *dining philosophers problem*, given by Dijkstra, is used throughout this section to illustrate the theory and make its application more clear.

Example 1. The *dining philosophers problem* is about five philosophers sitting at a round table where each one has her own plate and there lie five forks between the plates. The philosophers are alternately thinking (about philosophical problems) and eating. Furthermore [Dij71] states:

> "Their only problem - besides those of philosophy - is that the dish served is a very difficult kind of spaghetti, that has to be eaten with two forks. There are two forks next to each plate, so that presents no difficulty: as a consequence, however, no two neighbors may be eating simultaneously."

Dijkstra gave this example to reason about the sequential composition of processes and possibly occurring deadlocks. Assume that a philosopher picks up the right fork first and the left fork afterwards, if she is hungry. After eating, she puts both forks down and resumes thinking. This leads to problems if all the philosophers get hungry at exactly the same time because then each one holds their right fork but lacks the left one. They wait until they can pick up the left one, which will never happen since the philosopher to their left is holding this fork and is also waiting.

We represent the problem via BIP configurations, describe settings via formulae in separation logic, and reconfigure BIP configurations via a programming language.

Remark 1. Throughout this thesis, we assume that $\mathcal{U}$ is a fixed, countably infinite set. We use it to differentiate between different components by mapping them to elements in the universe $\mathcal{U}$. Furthermore, the set of variables $\mathcal{V}$ is another fixed, infinite set.

We start by defining a *signature* that determines the names of the component and interaction types.

Definition 5 (Signature)**.** *We define a finite set of* component symbols $C = \{C_1, \ldots, C_n\}$ *and a finite set of* interaction symbols $I = \{I_1, \ldots, I_m\}$*, where each interaction symbol $I_j \in I$ has an arity greater or equal 2, which is denoted by a mapping $\alpha : \{1, \ldots, m\} \to \mathbb{N}_0$. A* signature *is*

$$\langle C, I \rangle = \langle\, C_1, \ldots, C_n, I_1, \ldots, I_m \,\rangle.$$

Example 2 (Dining Philosophers). For the *dining philosophers problem*, we define the component symbols PHILO and FORK and the interaction symbols TAKE_RIGHT, TAKE_LEFT, and

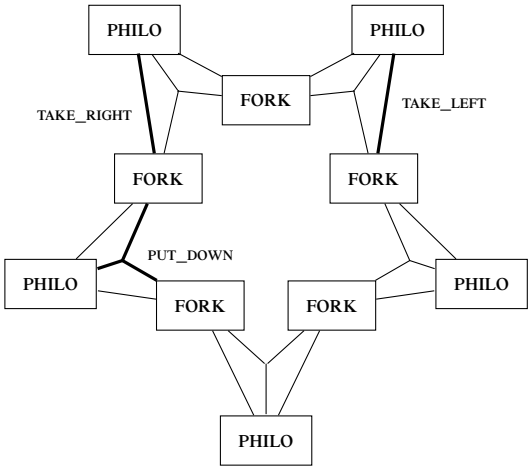

Figure 2.1.: Setting in the *dining philosophers problem* visualized by an undirected hypergraph.

PUT_DOWN. The arity of TAKE_RIGHT and TAKE_LEFT is 2, whereas the arity of PUT_DOWN is 3 (compare also with Figure 2.1). Hence the signature is

$$\langle C, I\rangle = \langle \text{PHILO}, \text{FORK}, \text{TAKE\_RIGHT}, \text{TAKE\_LEFT}, \text{PUT\_DOWN}\rangle.$$

For each component symbol $C_i \in C$, we define a *component type* that specifies the behavior of each instance of the type; it determines a set of ports, a set of states, and a deterministic finite-state transition system. This transition system changes its state when a connector (thus an interaction that is connected at some port) fires.

Definition 6 (Component Types)**.** *The* component type $C_i \in C$ *is defined by the deterministic finite-state transition system*

$$(\mathbb{S}_i, \mathbb{P}_i, s_i^0, \rightsquigarrow_i),$$

where $\mathbb{P}_i \subseteq \{I_j^\ell \mid I_j \in I,\ 1 \leq \ell \leq \alpha(j)\}$ *is a finite set of ports,* $\mathbb{S}_i$ *is a finite set of states,* $s_i^0 \in \mathbb{S}_i$ *is the initial state, and* $\rightsquigarrow_i \colon \mathbb{S}_i \times \mathbb{P}_i \to \mathbb{S}_i$ *is a partial transition function. For two component types* $C_i, C_j \in C$*,* $i \neq j$*, the sets of ports* $\mathbb{P}_i$ *and* $\mathbb{P}_j$ *and the sets of states* $\mathbb{S}_i$ *and* $\mathbb{S}_j$ *are disjoint.*

We say that a component type sets the *behavior* of the component. Note that the ports of a component type are also the labels of the transition system. Furthermore, they are interaction symbols with an additional number. We illustrate the previous definition by specifying component types for the *dining philosophers problem*.

Each philosopher can be thinking, hungry or eating. She changes her state if she picks up forks or lies them down; she thinks if she does not have a fork, she is hungry if she has one fork and she is eating if she has both forks. A fork is either busy, if a philosopher to its right or left uses it, or free.

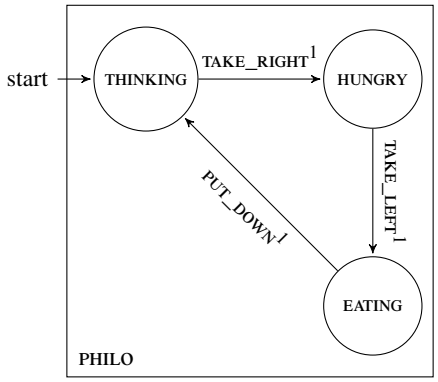

Figure 2.2.: The state diagram describes the behavior of instances of the component type PHILO.

Example 3 (Dining Philosophers). First, we define the component type PHILO and set

$$\mathbb{S}_{\text{PHILO}} = \{\text{THINKING}, \text{HUNGRY}, \text{EATING}\},$$
$$\mathbb{P}_{\text{PHILO}} = \{\text{TAKE\_RIGHT}^1, \text{TAKE\_LEFT}^1, \text{PUT\_DOWN}^1\} \text{ and}$$
$$s^0_{\text{PHILO}} = \text{THINKING}.$$

The transitions in the finite-state transition system are

$$\text{THINKING} \overset{\text{TAKE\_RIGHT}^1}{\rightsquigarrow_{\text{PHILO}}} \text{HUNGRY}, \ \text{HUNGRY} \overset{\text{TAKE\_LEFT}^1}{\rightsquigarrow_{\text{PHILO}}} \text{EATING},$$
$$\text{and } \text{EATING} \overset{\text{PUT\_DOWN}^1}{\rightsquigarrow_{\text{PHILO}}} \text{THINKING}.$$

The transition rules reflect the behavior of the philosophers and are depicted in Figure 2.2. Then the component type is $\text{PHILO} = (\mathbb{S}_{\text{PHILO}}, \mathbb{P}_{\text{PHILO}}, s^0_{\text{PHILO}}, \rightsquigarrow_{\text{PHILO}})$.
The component type FORK is specified as $\text{FORK} = (\mathbb{S}_{\text{FORK}}, \mathbb{P}_{\text{FORK}}, s^0_{\text{FORK}}, \rightsquigarrow_{\text{FORK}})$ with

$$\mathbb{S}_{\text{FORK}} = \{\text{FREE}, \text{BUSY}\},$$
$$\mathbb{P}_{\text{FORK}} = \{\text{TAKE\_LEFT}^2, \text{TAKE\_RIGHT}^2, \text{PUT\_DOWN}^2, \text{PUT\_DOWN}^3\},$$
$$s^0_{\text{FORK}} = \text{FREE}$$

and the transition rules are

$$\text{FREE} \overset{\text{TAKE\_LEFT}^2}{\rightsquigarrow_{\text{FORK}}} \text{BUSY}, \ \text{FREE} \overset{\text{TAKE\_RIGHT}^2}{\rightsquigarrow_{\text{FORK}}} \text{BUSY},$$
$$\text{BUSY} \overset{\text{PUT\_DOWN}^2}{\rightsquigarrow_{\text{FORK}}} \text{FREE and BUSY} \overset{\text{PUT\_DOWN}^3}{\rightsquigarrow_{\text{FORK}}} \text{FREE}.$$

The associated state diagram can be found in Figure 2.3.

The sets of states $\mathbb{S}_{\text{PHILO}}$ and $\mathbb{S}_{\text{FORK}}$ and the sets of ports $\mathbb{P}_{\text{PHILO}}$ and $\mathbb{P}_{\text{FORK}}$ are disjoint. The

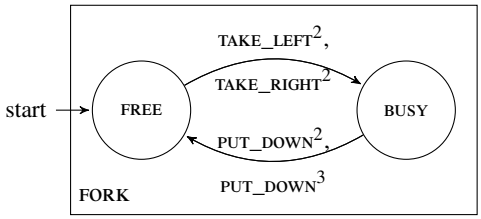

Figure 2.3.: The state diagram describes the behavior of instances of the component type FORK.

set of all ports is

$$\mathbb{P}_{\text{PHILO}} \cup \mathbb{P}_{\text{FORK}} = \{\text{TAKE\_LEFT}^1, \text{TAKE\_LEFT}^2, \text{TAKE\_RIGHT}^1, \text{TAKE\_RIGHT}^2, \\ \text{PUT\_DOWN}^1, \text{PUT\_DOWN}^2, \text{PUT\_DOWN}^3\}.$$

Interaction symbols are already interaction types and determine implicitly which components interactions of this type can connect. Let $I_j \in I$ be an interaction type. Then instances of this type connect $\alpha(j)$ components via the ports $I_j^1, \ldots, I_j^{\alpha(j)}$.

Example 4 (Dining Philosophers). The interaction types and their arity were already specified in Example 2 and the ports follow implicitly via the definition of the ports from the component types: There are two ports TAKE_LEFT^1 and TAKE_LEFT^2 with name TAKE_LEFT. The same holds for the interaction type TAKE_RIGHT. Only for the type PUT_DOWN there exist three ports PUT_DOWN^1, PUT_DOWN^2, and PUT_DOWN^3.

The definitions of the component types specify which ports belong to which component type. Here we can derive that TAKE_LEFT and TAKE_RIGHT connect a philosopher with a fork and PUT_DOWN connects a philosopher with two forks.

A *BIP system* is a collection of instances of the different component and interaction types, where each instance of a type is specified via an element in the universe $\mathcal{U}$. The set of instances for each type is given as a relation.

Definition 7 (BIP System)**.** *Let $\langle C, I\rangle$ be a signature, then*

$$\mathfrak{S} := \langle\, C_1^{\mathfrak{S}}, \ldots, C_n^{\mathfrak{S}}, I_1^{\mathfrak{S}}, \ldots, I_m^{\mathfrak{S}} \,\rangle$$

is a BIP system over $\langle C, I\rangle$*, where $C_i^{\mathfrak{S}} \subseteq \mathcal{U}$, $1 \le i \le n$, are relations over the universe $\mathcal{U}$ with arity 1, and $I_j^{\mathfrak{S}} \subseteq \mathcal{U}^{\alpha(j)}$, $1 \le j \le m$, are relations over the universe $\mathcal{U}$ with arity $\alpha(j)$.*

Example 5 (Dining Philosophers). Dijkstra specified the problem for five philosophers and five forks and even though this fixes the number of components to ten, we could represent this problem by various BIP systems over the signature $\langle C, I\rangle$: Let $u_1, u_2, \ldots, u_5 \in \mathcal{U}$ be five

arbitrary, but pairwise distinct elements in the universe. Then

$$\begin{aligned}\mathfrak{S} = \langle\ &\textsc{philo}^{\mathfrak{S}} = \{u_1, \ldots, u_5\},\ \textsc{fork}^{\mathfrak{S}} = \{u_1, \ldots, u_5\},\\ &\textsc{take\_left}^{\mathfrak{S}} = \{(u_1, u_1), \ldots, (u_5, u_5)\},\\ &\textsc{take\_right}^{\mathfrak{S}} = \{(u_1, u_5), (u_2, u_1), \ldots, (u_5, u_1)\},\\ &\textsc{put\_down}^{\mathfrak{S}} = \{(u_1, u_1, u_5), (u_2, u_2, u_1), \ldots, (u_5, u_5, u_4)\}\ \rangle\end{aligned}$$

represents the *dining philosophers problem* via a BIP system. The components of type PHILO and those of type FORK may be designated by the same elements in the universe $\mathcal{U}$. For example, there is an interaction (u_1, u_1, u_5) for type PUT_DOWN. Since the types of the connected components are specified via the ports, this interaction connects the component u_1 of type PHILO with the components u_1 and u_5 of type FORK.

Figure 2.4 gives a visualization of a part of the setting specified by Dijkstra and represented via a BIP system.

Whereas a BIP system describes a current setup of a distributed system, we also want to specify the current state of a system, thus the current states of all its components.

Definition 8 (State Snapshot)**.** *Let $\mathbb{S} = \bigcup_{i=1}^{n} \mathbb{S}_i$ be the set of all states of all component types. A* state snapshot over the signature $\langle C, I\rangle$ *is a function*

$$\varsigma : \mathcal{U} \times C \to \mathbb{S},$$

where $\varsigma(u, C_i) \in \mathbb{S}_i$ for every $1 \leq i \leq n$, hence the resulting state always belongs to the specified component type.

Furthermore, we want to describe components by using variables and add a map from variables into the universe $\mathcal{U}$. The combination of a BIP system, a state snapshot and a mapping of variables into the universe suffices to describe a setting for our purposes.

Definition 9 (BIP Configuration)**.** *A* BIP configuration over the signature $\langle C, I\rangle$ *is a triple $(\mathfrak{S}, \varsigma, \nu)$, where*

- *$\mathfrak{S}$ is a BIP system over $\langle C, I\rangle$,*
- *ς is a state snapshot over $\langle C, I\rangle$, and*
- *$\nu : \mathcal{V} \to \mathcal{U}$ maps each variable to an element in the universe.*

The set of all BIP configurations over the signature $\langle C, I\rangle$ is $\Sigma_{\langle C,I\rangle}$. The set of units *is*

$$\Sigma^{\emptyset}_{\langle C,I\rangle} = \{(\mathfrak{S}, \varsigma, \nu) \mid (\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle} \text{ and the relations in } \mathfrak{S} \text{ are empty}\}.$$

The definition of *BIP configurations* enables us to expand the BIP system specified in Example 5 by a *state snapshot* and a variable mapping.

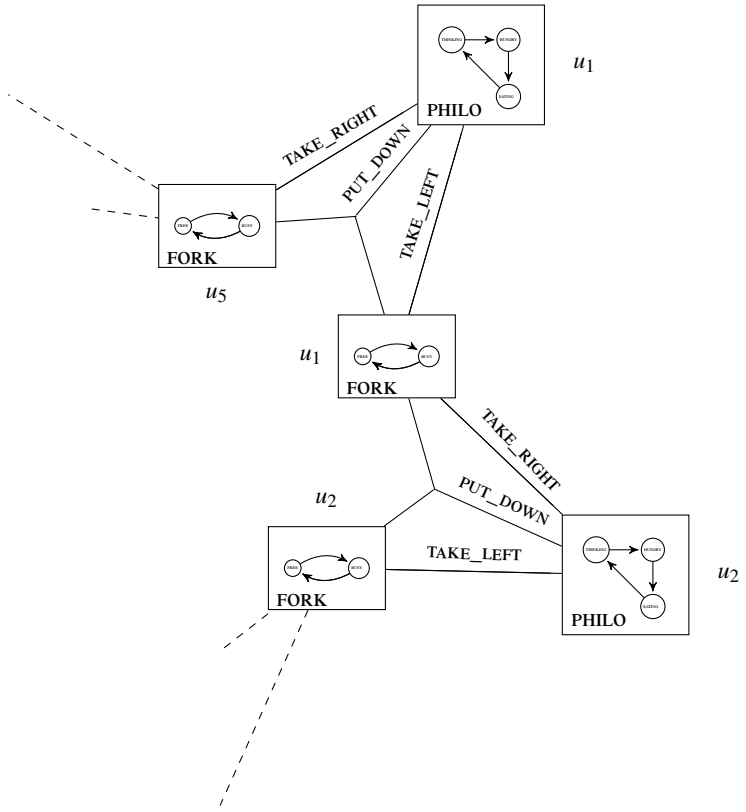

Figure 2.4.: A part of the setting in the *dining philosophers problem* visualized as a BIP system.

Example 6. Let $\mathfrak{S}$ be the BIP system defined in Example 5. We set $\mathbb{S} := \mathbb{S}_{\text{PHILO}} \cup \mathbb{S}_{\text{FORK}}$ and define an arbitrary total function $\varsigma : \mathcal{U} \times C \to \mathbb{S}$ such that

$$\varsigma(u_1, \text{PHILO}) = \text{THINKING},\ \varsigma(u_2, \text{PHILO}) = \text{EATING},$$
$$\varsigma(u_1, \text{FORK}) = \text{BUSY},\ \varsigma(u_2, \text{FORK}) = \text{BUSY},\ \varsigma(u_5, \text{FORK}) = \text{FREE}$$

and another arbitrary total function $\nu : \mathcal{V} \to \mathcal{U}$ such that

$$\nu(x) = u_1,\ \nu(y) = u_2,\ \nu(a) = u_5,\ \nu(b) = u_1,\ \nu(c) = u_2.$$

Then $(\mathfrak{S}, \varsigma, \nu)$ is a BIP configuration, which is partly visualized in Figure 2.5.

We want to show that the set of BIP configurations $\Sigma_{\langle C,I\rangle}$ for a signature $\langle C, I\rangle$ is a *separation algebra* for an operation $\bullet$ and with $\Sigma^{\emptyset}_{\langle C,I\rangle}$ as the set of units. We define the operation such that the composition of two BIP configurations is the disjoint union of each of the matching relations if the state snapshot ς and the variable mapping ν are equal. And, since the pointwise disjoint union is commutative, associative and cancellative, this carries over to the operation $\bullet$.

Theorem 1 (Multi-unit Separation Algebra)**.** *The triple $(\Sigma_{\langle C,I\rangle}, \bullet, \Sigma^{\emptyset}_{\langle C,I\rangle})$ is a multi-unit separation algebra on the set of BIP configurations over the signature $\langle C, I\rangle$, where $\Sigma^{\emptyset}_{\langle C,I\rangle}$ is the set of units and $\bullet$ satisfies:*

$$\begin{aligned}(\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1) = (\mathfrak{S}, \varsigma, \nu) \iff \quad & C_i^{\mathfrak{S}_0} \uplus C_i^{\mathfrak{S}_1} = C_i^{\mathfrak{S}} \quad \textit{for all } C_i \in C, \\ & I_i^{\mathfrak{S}_0} \uplus I_i^{\mathfrak{S}_1} = I_i^{\mathfrak{S}} \quad \textit{for all } I_i \in I, \\ & \varsigma_0 = \varsigma_1 = \varsigma \quad \textit{and} \\ & \nu_0 = \nu_1 = \nu\end{aligned}$$

if the relations are disjoint, hence $C_i^{\mathfrak{S}_0} \cap C_i^{\mathfrak{S}_1} = \emptyset$ and $I_i^{\mathfrak{S}_0} \cap I_i^{\mathfrak{S}_1} = \emptyset$ for all $C_i \in C$, and $I_i \in I$, and $(\mathfrak{S}_0, \varsigma_0, \nu_0), (\mathfrak{S}_1, \varsigma_1, \nu_1), (\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$.

Proof. First, we show that the operation $\bullet$ is commutative and associative, then we establish the existence of a unit $(\mathfrak{S}^{\emptyset}, \varsigma, \nu) \in \Sigma^{\emptyset}_{\langle C,I\rangle}$ for every element $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$. And lastly, we show the cancellativity of the operation.

1. Assume $(\mathfrak{S}_0, \varsigma_0, \nu_0), (\mathfrak{S}_1, \varsigma_1, \nu_1), (\mathfrak{S}_2, \varsigma_2, \nu_2) \in \Sigma_{\langle C,I\rangle}$, then we have

$$\begin{aligned}&[(\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1)] \bullet (\mathfrak{S}_2, \varsigma_2, \nu_2) = (\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet [(\mathfrak{S}_1, \varsigma_1, \nu_1) \bullet (\mathfrak{S}_2, \varsigma_2, \nu_2)] \\ \iff\ & (C_i^{\mathfrak{S}_0} \uplus C_i^{\mathfrak{S}_1}) \uplus C_i^{\mathfrak{S}_2} = C_i^{\mathfrak{S}_0} \uplus (C_i^{\mathfrak{S}_1} \uplus C_i^{\mathfrak{S}_2}) \quad \text{for all } C_i \in C, \\ & (I^{\mathfrak{S}_0} \uplus I^{\mathfrak{S}_1}) \uplus I^{\mathfrak{S}_2} = I^{\mathfrak{S}_0} \uplus (I^{\mathfrak{S}_1} \uplus I^{\mathfrak{S}_2}) \quad \text{for all } I_i \in I, \\ & \varsigma_0 = \varsigma_1 = \varsigma_2 \quad \text{and} \quad \nu_0 = \nu_1 = \nu_2.\end{aligned}$$

   If the operation $\bullet$ is defined on the given BIP configurations, then it is associative, since the disjoint union of sets and the equality of functions is associative. Similarly, commutativity follows from the commutativity of the disjoint union and equality of functions.

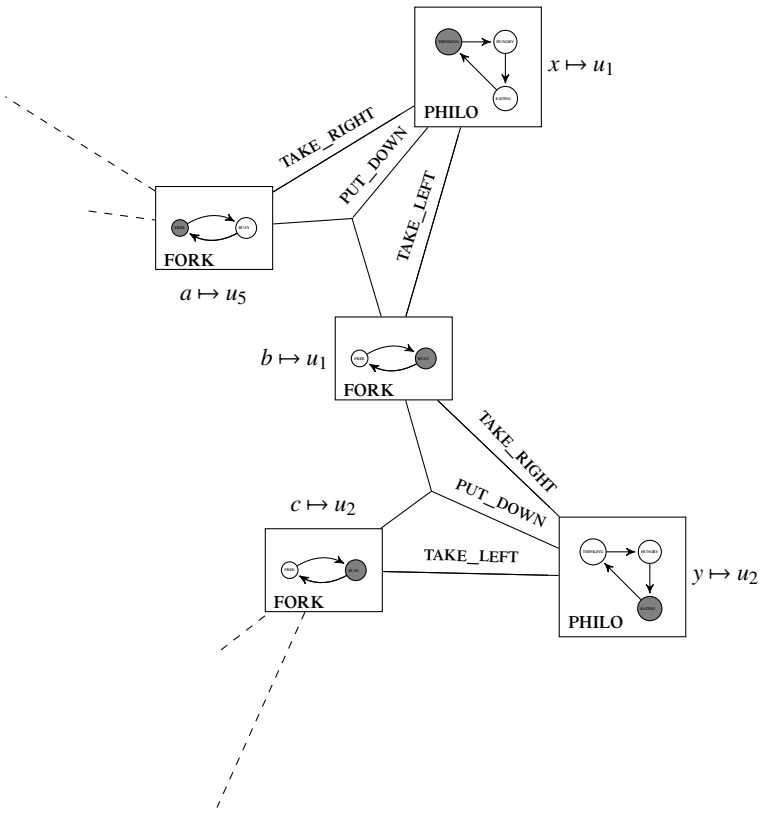

Figure 2.5.: A part of the setting in the *dining philosophers problem* visualized as a BIP configuration. BIP configurations expand BIP systems by specifying the state of each component and adding variables to address single components.

2. Let $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I \rangle}$ be an arbitrary BIP configuration. Then there exists an element $(\mathfrak{S}^{\emptyset}, \varsigma, \nu) \in \Sigma^{\emptyset}_{\langle C,I \rangle}$, where the relations in $\mathfrak{S}^{\emptyset}$ are empty, but the current states (ς) and the mapping of the variables (ν) are equal. Thus

$$(\mathfrak{S}, \varsigma, \nu) \bullet (\mathfrak{S}^{\emptyset}, \varsigma, \nu) = (\mathfrak{S}, \varsigma, \nu),$$

since the union of any set with the empty set does not alter the original set.

3. Let $(\mathfrak{S}_0, \varsigma_0, \nu_0), (\mathfrak{S}_1, \varsigma_1, \nu_1), (\mathfrak{S}_2, \varsigma_2, \nu_2) \in \Sigma_{\langle C,I \rangle}$ be three BIP configurations such that

$$\begin{aligned}
&(\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1) = (\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_2, \varsigma_2, \nu_2) \\
\iff\ & C_i^{\mathfrak{S}_0} \uplus C_i^{\mathfrak{S}_1} = C_i^{\mathfrak{S}_0} \uplus C_i^{\mathfrak{S}_2} \quad \text{for all } C_i \in C, \\
& I^{\mathfrak{S}_0} \uplus I^{\mathfrak{S}_1} = I^{\mathfrak{S}_0} \uplus I^{\mathfrak{S}_2} \quad \text{for all } I_i \in I, \\
& \varsigma_0 = \varsigma_1,\ \varsigma_0 = \varsigma_2 \quad \text{and} \quad \nu_0 = \nu_1,\ \nu_0 = \nu_2 \\
\iff\ & C_i^{\mathfrak{S}_1} = C_i^{\mathfrak{S}_2} \quad \text{for all } C_i \in C, \\
& I^{\mathfrak{S}_1} = I^{\mathfrak{S}_2} \quad \text{for all } I_i \in I, \\
& \varsigma_1 = \varsigma_2 \quad \text{and} \quad \nu_1 = \nu_2.
\end{aligned}$$

The first equivalence holds because we assume that the operation $\bullet$ is defined on the respective BIP configurations. Then the equality of $(\mathfrak{S}_1, \varsigma_1, \nu_1)$ and $(\mathfrak{S}_2, \varsigma_2, \nu_2)$ follows and hence the operation $\bullet$ is cancellative.

Hence the triple $(\Sigma_{\langle C,I \rangle}, \bullet, \Sigma^{\emptyset}_{\langle C,I \rangle})$ is a multi-unit separation algebra. □

Two BIP configurations are matching if their state snapshot and variable mapping are equal, and for each component and interaction type, the relations in both systems are disjoint. A BIP configuration $(\mathfrak{S}_0, \varsigma_0, \nu_0)$ is *larger* than another configuration $(\mathfrak{S}_1, \varsigma_1, \nu_1)$ if the relations in $\mathfrak{S}_0$ are supersets of the corresponding relations in $\mathfrak{S}_1$. The operation $\bullet$ enables us to split certain BIP configurations into several *smaller* configurations and analyze those. We will see that the smaller configurations are sometimes easier to study (see e.g *frame rule* in Chapter 5). An example can be found in Figure 2.6.

In this section, we have defined component and interaction types, which are instantiated in BIP systems. Then we extended the theory and defined BIP configurations as triples of a BIP system, a state snapshot (representing the state of components), and a variable mapping. Finally, we have proven that the set of BIP configurations $\Sigma_{\langle C,I \rangle}$ with the operation $\bullet$ and the set of units $\Sigma^{\emptyset}_{\langle C,I \rangle}$ is a separation algebra. The next step is to define a logic to specify properties of BIP configurations. The configurations will be models of the logical formulae. But before, we make a short excursion to the impact of the state changes within BIP configurations.

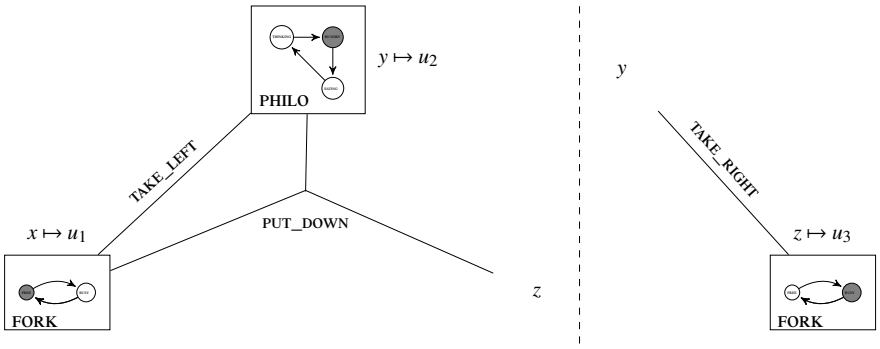

Figure 2.6.: Two BIP configurations that can be composed via the operation $\bullet$ if the state snapshot and the variable mapping match.

2.3. State Transitions for BIP Configurations

Remember that the component type specifies the behavior of each component by a finite-state transition system. The labels on the transitions equal the ports of the component and a component changes its state if and only if a port fires. A port fires when a connected interaction fires. And interactions can only fire if they are *closed* and *enabled*. A closed interaction is an interaction, where all the connected components are included in the BIP configuration.

Definition 10 (Closed Interaction)**.** *Let $(\Im, \varsigma, \nu)$ be a BIP configuration over the signature $\langle C, I\rangle$ and $I_j \in I$ an interaction type such that $a = (u_1, \ldots, u_{\alpha(j)}) \in I_j^{\Im}$ is an interaction. Then the interaction a is* closed *if, for each $1 \leq k \leq \alpha(j)$, there exists a component type $C_k \in C$ that contains the port $I_j^k \in \mathbb{P}_k$ and $u_k \in C_k^{\Im}$ is a component in the configuration.*

Intuitively, we can see the interactions as edges and an interaction is *closed* if no edges are dangling. An interaction is enabled if there exist matching transition rules in all of the connected components.

Definition 11 (Enabled Interaction)**.** *Let $(\Im, \varsigma, \nu)$ be a BIP configuration over the signature $\langle C, I\rangle$ and $I_j \in I$ an interaction type. Then, an interaction $(u_1, \ldots, u_{\alpha(j)}) \in I_j^{\Im}$ is* enabled *if, for each $1 \leq k \leq \alpha(j)$ and each component $C_k \in C$ with port $I_j^k \in \mathbb{P}_k$ and $u_k \in C_k^{\Im}$, there exists a transition*

$$s_k \overset{I_j^k}{\rightsquigarrow}_k s_k'$$

such that s_k is the current state of the component $u_k \in C_k^{\Im}$, hence $\varsigma(u_k, C_k) = s_k$.

Now, a *closed* and *enabled* interaction may fire, and if it fires then all the connected components change their state accordingly. We say that it "may" fire because it fires non-deterministically. Look at the following example for clarification.

Example 7 (Dining Philosophers). Figure 2.7 represents a BIP configuration that contains one component of type PHILO, two components of type FORK, and interactions of the types TAKE_LEFT, TAKE_RIGHT, and PUT_DOWN. Each of the interactions is *closed*, because

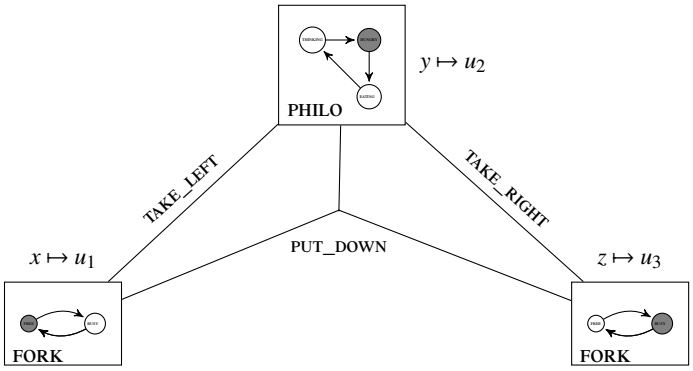

Figure 2.7.: An illustration of the BIP configuration $(\mathfrak{S}_2, \varsigma_2, \nu_2)$ specified in Example 7.

- TAKE_RIGHT connects two components, a component y of type PHILO that has TAKE_RIGHT^1 as a port, and a component z of type FORK that has TAKE_RIGHT^2 as a port,
- the same holds for the interaction TAKE_LEFT with the components y and x, and
- PUT_DOWN connects three components, the component y of type PHILO with port PUT_DOWN^1 and the components x and z of type FORK with ports PUT_DOWN^2 and PUT_DOWN^3. Since x and z each have both ports, it does not matter whether we connect x or z as the third component in the interaction.

Compare this with the definitions of the component and interaction types in Example 3 and Example 4.

Fork x is in state FREE, fork z is in state BUSY and philosopher y is in state HUNGRY. Hence only the interaction TAKE_LEFT is *enabled* since there exists a transition from the current state HUNGRY to EATING with label TAKE_LEFT^1 for the philosopher y, and a transition from the current state FREE to the state BUSY with label TAKE_LEFT^2 for the fork z. The other interactions are not open, because both of them connect the philosopher y and the philosopher is in a state, where the only possible transition has label TAKE_LEFT^1.

Thus, the interaction TAKE_LEFT may fire, but we do not know when and if it does happen (see Figure 2.8). If it fires, then the current BIP configuration changes into another configuration with a different state snapshot. The BIP structure and the variable mapping are not affected by state changes and stay the same. In the new BIP configuration, the component PHILO is in state EATING and fork x is in state BUSY. Then the only *enabled* interaction would be PUT_DOWN.

We use the transitions in single components to define a transition function for BIP configurations. For each enabled interaction, there should be a transition from a current BIP configuration to a configuration where exactly the states of the components change that are connected by the interaction.

Definition 12 (Concrete Semantics)**.** *Let* $\Sigma = \{I_j(x_1, \ldots, x_{\alpha(j)}) \mid I_j \in I, x_1, \ldots, x_{\alpha(j)} \in \mathcal{V}\}$ *be the set of all interaction atoms. Then we define a transition function* $\rightsquigarrow_c \colon \Sigma_{\langle C,I\rangle} \times \Sigma \to \Sigma_{\langle C,I\rangle}$,

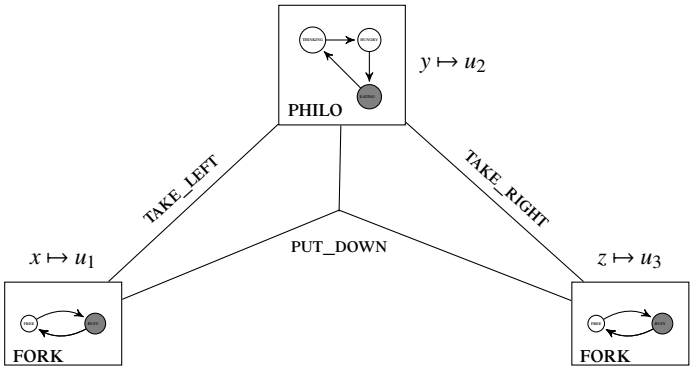

Figure 2.8.: This BIP configuration is obtained if the interaction TAKE_RIGHT in Figure 2.7 fires.

where it is

$$(\mathfrak{S}, \varsigma, \nu) \overset{I_j(x_1,\ldots,x_{\alpha(j)})}{\rightsquigarrow_c} (\mathfrak{S}, \varsigma', \nu), \text{ if and only if}$$

1. $a := (\nu(x_1), \ldots, \nu(x_\alpha)) \in I_j^{\mathfrak{S}}$ *is an interaction,*
2. *a is closed and enabled in* $(\mathfrak{S}, \varsigma, \nu)$*,*
3. $\varsigma' = \varsigma[(u_k, C_k) \leftarrow s'_k \mid u_k \in a, u_k \in C_k^{\mathfrak{S}}, I_j^k \in \mathbb{P}_k]$ *is the state snapshot, where* s'_k *is defined in Definition 11.*

We extend the notation to $(\mathfrak{S}, \varsigma, \nu) \overset{w}{\rightsquigarrow_c} (\mathfrak{S}, \varsigma', \nu)$ *for a word* $w \in \Sigma^*$ *and write* $(\mathfrak{S}, \varsigma, \nu) \rightsquigarrow_c{}^* (\mathfrak{S}, \varsigma', \nu)$ *if there exists any word* $w \in \Sigma^*$ *such that* $(\mathfrak{S}, \varsigma, \nu) \overset{w}{\rightsquigarrow_c} (\mathfrak{S}, \varsigma', \nu)$ *is a transition.*

Suppose that $(\mathfrak{S}_2, \varsigma_2, \nu_2)$ is the BIP configuration in Figure 2.7 and $(\mathfrak{S}_2, \varsigma_2', \nu_2)$ is the configuration in Figure 2.8. Then there exists a transition

$$(\mathfrak{S}_2, \varsigma_2, \nu_2) \overset{\text{TAKE\_LEFT}(y,x)}{\rightsquigarrow_c} (\mathfrak{S}_2, \varsigma_2', \nu_2).$$

In this chapter, we have combined the notion of separation algebras with BIP frameworks. We defined BIP systems that represent a current manifestation of a BIP framework for given component and interaction types $\langle C, I \rangle$. Furthermore, we specified BIP configurations as extensions of BIP systems that also model the current state of the components. We defined an operation $\bullet$ and a set of units such that the set of BIP configurations for a fixed signature $\langle C, I \rangle$ is a separation algebra. This allows us to split BIP configurations into *smaller* BIP configurations and put them back together.

Furthermore, we considered the types of components and how they lead to state changes within a BIP system. A BIP configuration transforms into another BIP configuration if interactions fire.

There are thus two different views of BIP configurations: On one hand, we regard a static BIP configuration that is part of a separation algebra and that can e.g. be split. And on the other hand, we consider dynamic BIP configurations, where interactions may fire and states of components may change. In the next chapters, this difference will be crucial because it will result in two different approaches for the proofs of the correctness of programs.

3. Separation Logic on BIP

Since we want to analyze reconfiguration programs on BIP systems, we need to specify preconditions that hold before the execution of such a reconfiguration program and prove postconditions that hold afterwards. Pre- and postconditions are usually predicates that contain all elements having a certain property (predicates were defined in Definition 2). In our case, the predicates are subsets of the set of all BIP configurations over a fixed signature and we want to describe them by their distinct property. For that purpose, we define the *separation logic on BIP* that is an adaption of abstract separation logic (see [COY07]).

In this chapter, we give the syntax of the separation logic on BIP $\mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ and define its semantics on BIP configurations. As in the previous chapter, the universe $\mathcal{U}$ and the set of variables $\mathcal{V}$ remain fixed. Additionally, we assume that C and $\mathcal{P}$ are fixed, countably infinite sets, where C is a set of constants (e.g. the set of natural numbers) and $\mathcal{P}$ is a set of predicate symbols and each predicate symbol $A \in \mathcal{P}$ has an arity $\alpha(A)$. Furthermore, we assume that $\langle C, I\rangle$ is a signature as in Definition 7.

In order to describe properties of BIP configurations, we want to be able to use the usual conjunctions and implications (hence we need to define $\wedge$ and $\neg$). Furthermore, we want to say that a property holds on one part of the configuration and another property holds on the other part (this is the spatial conjunction $*$), we want to retrieve the types of components and interactions, the current state of a component and specify a BIP configuration that is empty. The general construction of $\mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ differs from the construction of separation logic on heaps mainly in the atomic formulae, furthermore, we do not consider the spatial implication $-\!\!*$ (compare with [IO01]).

Moreover, we define inductive predicates that allow an inductive description of properties of BIP configurations. For the inductive predicates, we assume that a set of rules $\mathcal{R}$ for the unfolding of the predicate symbols is given. This will be specified formally in Definition 16.

The syntax is defined via the Backus-Naur normal form.

Definition 13 (Syntax)**.** *The* separation logic on BIP configurations *over the signature* $\langle C, I\rangle$ *for a set of rules* $\mathcal{R}$ *is the set* $\mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ *defined by*

$$\begin{aligned}\phi \ ::= \ & \texttt{emp} \mid C_i(x) \mid I_j(x_1, \ldots, x_{\alpha(j)}) \mid \mathrm{state}(x, s) \mid A(t_1, \ldots, t_{\alpha(A)}) \mid \\ & \texttt{true} \mid \neg\phi \mid \phi * \psi \mid \phi \wedge \psi \mid \exists x.\phi\end{aligned}$$

with $\phi, \psi \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$, $C_i \in C$ *for* $1 \leq i \leq n$, $I_j \in I$ *for* $1 \leq j \leq m$, $A \in \mathcal{P}$, $x, x_1, \ldots, x_{\alpha(j)} \in \mathcal{V}$ *and* $s \in \mathbb{S}_i$ *for* $1 \leq i \leq n$. *Furthermore,* $t_1, \ldots, t_{\alpha(A)} \in \mathcal{V} \cup C$ *are so-called* terms. *The elements* $\phi, \psi \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ *are called* $\langle C, I\rangle$-formulae.

Here, `emp` describes an empty system and the atomic fomula $C_i(x)$ specifies the existence of exactly one instance of the component type C_i which is mapped to by x. Similarly,

$I_j(x_1, \ldots, x_{\alpha(j)})$ states that there exists exactly one instance of the interaction type I_j which connects the components pointed at by x_1 to $x_{\alpha(j)}$. The atomic formula state(x, s) specifies that the component x is in state s. Even though there may exist multiple components described by x, this is well-defined since each of the components has a different type and only one of the component types has state s in its set of states. Furthermore, the formula $A(t_1, \ldots, t_{\alpha(A)})$ is a predicate, which can be unfolded using the rules $\mathcal{R}$ (see Definition 16). It represents an arbitrary unfolding of the predicate symbol and can be used to describe inductive structures.

The star operator $*$ represents the spatial conjunction that is used to "divide" BIP configurations into several parts (see e.g. Figure 2.6). The other symbols have their usual meanings; `true` represents a valid formula, $\neg\phi$ negates the boolean valuation of the formula ϕ, $\wedge$ corresponds to conjunction and $\exists$ is the existential quantifier. Following the convention, we define that $*$ has higher precedence than $\wedge$, hence $A \wedge B * C = A \wedge (B * C)$ for formulae $A, B, C \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$. Furthermore, we write $A \vee B$ as a shorthand for the formula $\neg(\neg A \wedge \neg B)$. Last but not least, we call a formula in separation logic on BIP that is constructed without predicate symbols *predicateless*.

Before we specify the interpretation of a predicate symbol via the set of rules $\mathcal{R}$, we give examples for predicateless formulae in $\mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$, define symbolic configurations and specify the standard function that maps a $\langle C, I\rangle$-formula to the set of its free variables.

Example 8 (Dining Philosophers). Let $\langle C, I\rangle$ be the signature of the *dining philosophers problem* as defined in Example 5. Exemplary formulae in $\mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ are

$$\texttt{emp} \qquad \textsc{philo}(y) \qquad \textsc{fork}(z) \wedge \mathrm{state}(z, \textsc{busy})$$
$$\textsc{philo}(y) * \textsc{take\_left}(y, z) * \textsc{fork}(z) \quad \text{and}$$
$$\exists a.\textsc{fork}(a) * \textsc{take\_right}(y, a) * \textsc{philo}(y) * \textsc{take\_left}(y, z) * \textsc{fork}(z) * \textsc{put\_down}(y, a, z).$$

Remember that PHILO and FORK are component types and TAKE_LEFT, TAKE_RIGHT, and PUT_DOWN are interaction types. Furthermore, BUSY is a possible state of an instance of the component type FORK and x, y and z are variables in $\mathcal{V}$.

A *symbolic configuration* is a formula in $\mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ in a certain form. It allows a direct construction of a model by creating a BIP system with the specified components and interactions, specifying matching state snapshots and variable mappings, and composing the resulting BIP configuration with models of the predicates.

Definition 14 (Symbolic Configuration)**.** *A* symbolic configuration *is a formula of the form*

$$\exists \mathbf{x} \,.\, \circledast_{p=1}^{k} C_{i_p}(x_p) * \circledast_{q=1}^{\ell} I_{j_q}(\mathbf{y}_q) * \circledast_{r=1}^{h} A_r(\mathbf{t}_r) \wedge \bigwedge_{s=1}^{g} \mathrm{state}(x_s, s_s),$$

over a signature $\langle C, I\rangle$, where $\mathbf{x} \in \mathcal{V}^{k'}$, $k' \leq k$, is a tuple of variables and $C_{i_p} \in C$ are component types for all $1 \leq p \leq k$, $\mathbf{y}_q \in \mathcal{V}^{\alpha(j_q)}$ are tuples of variables and $I_{j_q} \in I$ are interaction types for all $1 \leq q \leq \ell$. Furthermore, $\mathbf{t}_r \in (\mathcal{V} \cup C)^{\alpha(A_r)}$ are tuples of terms and $A_r \in \mathcal{P}$ are predicate symbols for all $1 \leq r \leq h$, and $x_s \in \mathcal{V}$ and $s_s \in \mathbb{S}$ for all $1 \leq s \leq g$.

A symbolic configuration with no occurrences of predicate atoms (i.e. $h = 0$) is said to be predicateless.

Definition 15 (Free Variables)**.** *The set of free variables* $\mathrm{fv}(\psi) \subset \mathcal{V}$ *of a* $\langle C, I\rangle$*-formula* $\psi \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ *is defined inductively as*

- $\mathrm{fv}(\mathtt{emp}) = \mathrm{fv}(\mathtt{true}) = \emptyset$,
- $\mathrm{fv}(C_i(x)) = \{x\}$, $\mathrm{fv}(I_j(x_1, \ldots, x_{\alpha(j)})) = \{x_1, \ldots, x_{\alpha(j)}\}$, $\mathrm{fv}(\mathrm{state}(x, s)) = \{x\}$ *and* $\mathrm{fv}(A(t_1, \ldots, t_{\alpha(A)})) = \{t_1, \ldots, t_{\alpha(A)}\} \cap \mathcal{V}$,
- $\mathrm{fv}(\neg\psi) = \mathrm{fv}(\psi)$, $\mathrm{fv}(\phi \star \psi) = \mathrm{fv}(\psi) \cup \mathrm{fv}(\psi)$ *for* $\star \in \{*, \wedge, \vee\}$ *and* $\mathrm{fv}(\exists x.\psi) = \mathrm{fv}(\psi) \setminus \{x\}$,

for $C_i \in C$, $I_j \in I$, $x, x_1, \ldots, x_{\alpha(j)} \in \mathcal{V}$, $s \in \mathbb{S}$, *and* $\phi, \psi \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$.

Finally, we can give the interpretation of a predicate symbol that is defined by a set of rules $\mathcal{R}$, which is called a *system of inductive definitions*. Predicate symbols are interpreted by unfolding the symbols following the rules $\mathcal{R}$ to obtain new formulae; here, an unfolding is roughly a substitution of a predicate symbol by the right-hand side of a rule. This can be done iteratively until a predicateless formula is obtained, if the rules are well-defined.

Definition 16 (System of Inductive Definitions)**.** *A* system of inductive definitions *(SID) is a set* $\mathcal{R}$ *of rules of the form* $A(t_1, \ldots, t_{\alpha(A)}) \leftarrow \rho$*, where* ρ *is a symbolic configuration, such that* $\mathrm{fv}(\rho) \subseteq \{t_1, \ldots, t_{\alpha(A)}\} \cap \mathcal{V}$*,* $A \in \mathcal{P}$ *is a predicate symbol and* $t_1, \ldots, t_{\alpha(A)}$ *are terms.*

Given symbolic configurations ψ *and* φ*, the* rewriting step $\psi \leftarrow_{\mathcal{R}} \varphi$ *replaces a predicate atom* $A(v_1, \ldots, v_{\alpha(A)})$ *of* ψ *with the formula* $\rho\theta$*, where:*

1. $A(t_1, \ldots, t_{\alpha(A)}) \leftarrow \rho$ *is a rule in* $\mathcal{R}$,
2. $\theta : \{t_1, \ldots, t_{\alpha(A)}\} \cap \mathcal{V} \to \mathcal{V}$ *is an injective substitution such that, if* $t_i \in \mathcal{V}$*, then* $\theta(t_i) = v_i$ *for all* $1 \leq i \leq \alpha(A)$,
3. *if* $t_i \in C$ *is a constant, then* $v_i = t_i$*, for all* $1 \leq i \leq \alpha(A)$.

Assume that the free variables of ρ are $\mathrm{fv}(\rho) = \{x_1, \ldots, x_n\} \subseteq \{t_1, \ldots, t_{\alpha(A)}\}$, then $\rho\theta$ is the formula, where each free variable x_i is substituted by $\theta(x_i)$, for $1 \leq i \leq n$. An example of inductive predicates can be found at the end of this chapter.

In the following, we denote by $\leftarrow^*_{\mathcal{R}}$ the reflexive and transitive closure of $\leftarrow_{\mathcal{R}}$ and $\psi \leftarrow^{pl}_{\mathcal{R}} \phi$ specifies an unfolding $\psi \leftarrow^*_{\mathcal{R}} \phi$ such that ϕ is predicateless.

Now we want to define a valuation of the formulae in the separation logic $\mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ that valuates the formulae on BIP configurations.

Definition 17 (Semantics)**.** *We define the valuation of the separation logic* $\mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ *inductively over the structure of the* $\langle C, I\rangle$*-formulae. Let* $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$ *be a BIP configuration, then*

- $(\mathfrak{S}, \varsigma, \nu) \models \mathtt{emp}$ *iff* $(\mathfrak{S}, \varsigma, \nu) \in \Sigma^{\emptyset}_{\langle C,I\rangle}$,
- $(\mathfrak{S}, \varsigma, \nu) \models C_i(x)$ *iff* $C_i^{\mathfrak{S}} = \{\nu(x)\}$, $C_j^{\mathfrak{S}} = \emptyset$ *for all* $C_j \in C \setminus \{C_i\}$ *and* $I_i^{\mathfrak{S}} = \emptyset$ *for all* $I_i \in I$,

- $(\mathfrak{S},\varsigma,\nu) \models I_i(x_1,\ldots,x_{\alpha(i)})$ *iff* $I_i^{\mathfrak{S}} = \{(\nu(x_1),\ldots,\nu(x_{\alpha(i)}))\}$, $I_j^{\mathfrak{S}} = \emptyset$ *for all* $I_j \in I \setminus \{I_i\}$ *and* $C_i^{\mathfrak{S}} = \emptyset$ *for all* $C_i \in C$, *and*
- $(\mathfrak{S},\varsigma,\nu) \models \text{state}(x,s)$ *iff* $s \in \mathbb{S}_i$ *for some* $C_i \in C$, $\nu(x) \in C_i^{\mathfrak{S}}$ *and* $\varsigma(\nu(x),C_i) = s$.
- $(\mathfrak{S},\varsigma,\nu) \models \texttt{true}$ *is always valid, and*
- $(\mathfrak{S},\varsigma,\nu) \models A(t_1,\ldots,A_{\alpha(A)})$ *iff* $(\mathfrak{S},\varsigma,\nu) \models \phi$ *for some unfolding* $A(t_1,\ldots,A_{\alpha(A)}) \leftarrow_{\mathcal{R}} \phi$.

Furthermore, we define

- $(\mathfrak{S},\varsigma,\nu) \models \neg\phi$ *iff* $(\mathfrak{S},\varsigma,\nu) \not\models \phi$,
- $(\mathfrak{S},\varsigma,\nu) \models \phi * \psi$ *iff there exist* $(\mathfrak{S}_0,\varsigma_0,\nu_0),(\mathfrak{S}_1,\varsigma_1,\nu_1) \in \Sigma_{\langle C,I\rangle}$ *such that* $(\mathfrak{S}_0,\varsigma_0,\nu_0) \bullet (\mathfrak{S}_1,\varsigma_1,\nu_1) = (\mathfrak{S},\varsigma,\nu)$, $(\mathfrak{S}_0,\varsigma_0,\nu_0) \models \phi$ *and* $(\mathfrak{S}_1,\varsigma_1,\nu_1) \models \psi$,
- $(\mathfrak{S},\varsigma,\nu) \models \phi \wedge \psi$ *iff* $(\mathfrak{S},\varsigma,\nu) \models \phi$ *and* $(\mathfrak{S},\varsigma,\nu) \models \psi$, *and*
- $(\mathfrak{S},\varsigma,\nu) \models \exists x.\phi$ *iff there exists* $u \in \mathcal{U}$ *such that* $\nu' := \nu[x \leftarrow u]$ *and* $(\mathfrak{S},\varsigma,\nu') \models \phi$.

It is $\phi,\psi \in \text{SL}^{\text{BIP}}_{\mathcal{R}}\langle C,I\rangle$, $C_i \in C$ *for* $1 \le i \le n$, $I_j \in I$ *for* $1 \le j \le m$, $A \in \mathcal{P}$, $x,x_1,\ldots,x_{\alpha(j)} \in \mathcal{V}$ *and* $s \in \mathbb{S}_i$ *for* $1 \le i \le n$ *and* $t_1,\ldots,t_{\alpha(A)} \in \mathcal{V} \cup C$.

When defining the existential quantifier, we write the expression $\nu[x \leftarrow u]$. This means that we obtain a new function where all variables map to the same elements as in ν except for x, which is mapped to u.

We see that `emp` specifies all BIP configurations where the relations over the signature $\langle C,I\rangle$ are empty, thus configurations without any components and interactions. Unless stated otherwise, the specification of components and interactions signifies that they are the only components and interactions in the model. For example, models of the formula $C_i(x)$ contain exactly one component of type C_i and no other components or interactions. The spatial conjunction $*$ specifies that one property should hold on one part of a BIP configuration and another property should hold on the other part. Models of the formula $C_i(x) * \texttt{true}$ contain a component of type C_i on one part of the BIP system and the other part of the system is not specified, hence they may contain an arbitrary number of other components and interactions.

A BIP configuration is a model of a predicate symbol $A(t_1,\ldots,t_{\alpha(A)})$ if it is a model of an unfolding of the predicate symbol.

Example 9 (Dining Philosophers). We define a BIP configuration $(\mathfrak{S}_2,\varsigma_2,\nu_2)$ that is represented in Figure 2.7 and analyze variations of the formulae given in Example 8. Let the BIP system $\mathfrak{S}_2$ be defined as

$$\begin{aligned}\mathfrak{S}_2 = \langle\ &\textsc{philo}^{\mathfrak{S}_2} = \{u_2\},\ \textsc{fork}^{\mathfrak{S}_2} = \{u_1,u_3\},\\ &\textsc{take\_left}^{\mathfrak{S}_2} = \{(u_2,u_1\},\textsc{take\_right}^{\mathfrak{S}_2} = \{(u_2,u_3)\},\\ &\textsc{put\_down}^{\mathfrak{S}_2} = \{(u_2,u_3,u_1)\}\ \rangle.\end{aligned}$$

Furthermore $\nu_2 : \mathcal{V} \to \mathcal{U}$ is an arbitrary variable mapping with $\nu_2(x) = u_1$, $\nu_2(y) = u_2$ and $\nu_2(z) = u_3$ and $\varsigma_2 : \mathcal{U} \times C \to \mathbb{S}$ is an arbitrary state snapshot with $\varsigma_2(u_3,\textsc{fork}) = \textsc{busy}$. We observe of which formulae the BIP configuration is a model:

- The BIP configuration $(\mathfrak{S}_2, \varsigma_2, \nu_2)$ is not an element of the set of units ${\Sigma_{\langle C,I\rangle}}^{\emptyset}$, since the relations in the BIP system $\mathfrak{S}_2$ are not empty and $(\mathfrak{S}_2, \varsigma_2, \nu_2)$ contains components and interactions. Thus $(\mathfrak{S}_2, \varsigma_2, \nu_2) \not\models \mathtt{emp}$.

- The configuration $(\mathfrak{S}_2, \varsigma_2, \nu_2)$ is not a model of $\textsc{philo}(y)$ either, since there exist other components and interactions, e.g. the relation ${\textsc{fork}}^{\mathfrak{S}_2}$ is not empty. Let us define another BIP structure $\mathfrak{S}$, where the only non-empty relation is ${\textsc{philo}}^{\mathfrak{S}} = \{u_2\}$, then $(\mathfrak{S}, \varsigma_2, \nu_2)$ is a model, since $\nu_2(u_2) = y$. We write $(\mathfrak{S}, \varsigma_2, \nu_2) \models \textsc{philo}(y)$.

- It is $\nu_2(z) = u_3$ and $u_3 \in {\textsc{fork}}^{\mathfrak{S}_2}$. Furthermore $\varsigma_2(u_3, \textsc{fork}) = \textsc{busy}$ and hence $(\mathfrak{S}_2, \varsigma_2, \nu_2) \models \text{state}(z, \textsc{busy})$. But $(\mathfrak{S}_2, \varsigma_2, \nu_2) \not\models \textsc{fork}(z)$, since e.g. the relation ${\textsc{philo}}^{\mathfrak{S}_2}$ is non-empty. Thus $(\mathfrak{S}_2, \varsigma_2, \nu_2) \not\models \textsc{fork}(z) \wedge \text{state}(z, \textsc{busy})$.

- Again, it is easy to see that $(\mathfrak{S}_2, \varsigma_2, \nu_2)$ is not a model of $\textsc{philo}(y) * \textsc{take\_left}(y, z) * \textsc{fork}(z)$, since e.g. the relation ${\textsc{fork}}^{\mathfrak{S}_2}$ contains more than a single element. We can slightly adapt the formula and write $\textsc{philo}(y) * \textsc{take\_left}(y, z) * \textsc{fork}(z) * \mathtt{true}$. Then

$$(\mathfrak{S}_2, \varsigma_2, \nu_2) \models \textsc{philo}(y) * \textsc{take\_left}(y, z) * \textsc{fork}(z) * \mathtt{true},$$

  since there exist BIP systems $\mathfrak{S}_{21}$ and $\mathfrak{S}_{22}$ such that

$$(\mathfrak{S}_{21}, \varsigma_2, \nu_2) \models \textsc{philo}(y) * \textsc{take\_left}(y, z) * \textsc{fork}(z)$$
$$\text{and } (\mathfrak{S}_{21}, \varsigma_2, \nu_2) \models \mathtt{true}.$$

  Namely

$$\begin{aligned}
\mathfrak{S}_{21} = \langle\ & {\textsc{philo}}^{\mathfrak{S}_{21}} = \{u_2\},\ {\textsc{fork}}^{\mathfrak{S}_{21}} = \{u_3\}, \\
& {\textsc{take_left}}^{\mathfrak{S}_{21}} = \{(u_2, u_3)\}, {\textsc{take_right}}^{\mathfrak{S}_{21}} = \emptyset, \\
& {\textsc{put_down}}^{\mathfrak{S}_{21}} = \emptyset\ \rangle \text{ and} \\
\mathfrak{S}_{22} = \langle\ & {\textsc{philo}}^{\mathfrak{S}_{21}} = \emptyset,\ {\textsc{fork}}^{\mathfrak{S}_{22}} = \{u_1\}, \\
& {\textsc{take_left}}^{\mathfrak{S}_{22}} = \emptyset, {\textsc{take_right}}^{\mathfrak{S}_{22}} = \{(u_2, u_1)\}, \\
& {\textsc{put_down}}^{\mathfrak{S}_{22}} = \{(u_2, u_3, u_1)\}\ \rangle.
\end{aligned}$$

  It is straightforward to check that the correlating relations are disjoint and hence $(\mathfrak{S}_{21}, \varsigma_2, \nu_2) \bullet (\mathfrak{S}_{22}, \varsigma_2, \nu_2)$ is defined and equals $(\mathfrak{S}_2, \varsigma_2, \nu_2)$.

- Last but not least, we have

$$\begin{aligned}
(\mathfrak{S}_2, \varsigma_2, \nu_2) \models\ & \textsc{fork}(x) * \textsc{take_right}(y, x) * \textsc{philo}(y) \\
& * \textsc{take_left}(y, z) * \textsc{fork}(z) * \textsc{put_down}(y, x, z).
\end{aligned}$$

  If we set $\nu_2' := \nu_2[a \leftarrow \nu_2(x)]$, then

$$\begin{aligned}
(\mathfrak{S}_2, \varsigma_2, \nu_2') \models\ & \textsc{fork}(a) * \textsc{take_right}(y, a) * \textsc{philo}(y) \\
& * \textsc{take_left}(y, z) * \textsc{fork}(z) * \textsc{put_down}(y, a, z)
\end{aligned}$$

and hence $(\mathfrak{S}_2, \varsigma_2, \nu_2)$ models $\exists a.\textsc{fork}(a) * \textsc{take\_right}(y,a) * \textsc{philo}(y) * \textsc{take\_left}(y,z) * \textsc{fork}(z) * \textsc{put\_down}(y,a,z)$.

In the rest of this chapter, we define the implication between formulae and specify the sets of components and interactions in a symbolic configuration. Furthermore we give an example for inductive predicates on the *dining philosophers problem*.

Definition 18 (Implication of Formulae)**.** *Let $\phi, \psi \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ be formulae. Then we define*

$$\phi \models \psi \quad \textit{iff} \quad (\mathfrak{S}, \varsigma, \nu) \models \phi \textit{ implies } (\mathfrak{S}, \varsigma, \nu) \models \psi$$

for all BIP configurations $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$.

Definition 19. *Let ψ be a symbolic configuration over a signature $\langle C, I\rangle$ of the form given in Definition 14. If ψ is quantifier-free and predicateless, then we define the set of component axioms*

$$\Gamma(\psi) = \{C_{i_p}(x_p) \mid 1 \le p \le k\}$$

and the set of interaction atoms

$$\Sigma(\psi) = \{I_{j_q}(x_q) \mid 1 \le q \le \ell\}.$$

We extend these notations to

- *existentially quantified symbolic configurations $\phi = \exists x_1 \ldots \exists x_k.\psi$, where $x_1, \ldots, x_k \in \mathcal{V}$ are assumed to be pairwise distinct and distinct from the free variables* $\mathrm{fv}(\phi)$*, as*

  $$\Gamma_\theta(\phi) := \Gamma(\psi\theta) \quad \textit{and} \quad \Sigma_\theta(\phi) := \Sigma(\psi\theta),$$

  where $\theta : \{x_1, \ldots, x_k\} \cup \mathrm{fv}(\phi) \to \mathcal{V}$ is an injective substitution, and

- *finite disjunctions $\pi = \bigwedge_{i=1}^{n} \phi_i$ of symbolic configurations, as*

  $$\Gamma_\theta(\pi) := \bigcup_{i=1}^{n} \Gamma_\theta(\phi_i) \quad \textit{and} \quad \Sigma_\theta(\pi) := \bigcup_{i=1}^{n} \Sigma_\theta(\phi_i),$$

  where $\theta : \bigcup_{i=1}^{n} \mathrm{fv}(\phi_i) \to \mathcal{V}$ is an injective substitution.

For a predicate atom $\mathcal{A} := A(x_1, \ldots, x_{\alpha(A)})$, we define

$$\Gamma_\theta(\mathcal{A}) := \bigcup \{\, \Gamma_\theta(\phi) \mid \mathcal{A} \leftarrow_{\mathcal{R}} \phi,\ \phi \textit{ is predicateless} \,\}, \textit{ and}$$
$$\Sigma_\theta(\mathcal{A}) := \bigcup \{\, \Sigma\theta(\phi) \mid \mathcal{A} \leftarrow_{\mathcal{R}} \phi,\ \phi \textit{ is predicateless} \,\}$$

and generalize the notation to finite disjunctions of symbolic configurations that, moreover, contain predicate atoms. Note that $\Gamma_\theta(\pi)$ and $\Sigma_\theta(\pi)$ are possibly infinite in the latter case.

Now we look at an inductive predicate for the *dining philosophers problem* that represents a table with forks and philosophers that are connected by some interactions.

Example 10 (Dining Philosophers). Inductive predicates enable the definition of tables for the *dining philosophers problem* of arbitrary but finite size. We start by defining a predicate $\text{seat}^{\text{R}}(f_0, f_1)$ that encapsulates a philosopher, a fork, and some interactions:

$$\begin{aligned}\text{seat}^{\text{R}}(f_0, f_1) \leftarrow \exists p_1 \,.\, &\textsc{take\_left}(p_1, f_0) * \textsc{philo}(p_1) * \textsc{take\_right}(p_1, f_1) \\ &* \textsc{fork}(f_1) * \textsc{put\_down}(p_1, f_1, f_0).\end{aligned}$$

There exists only one rule for this predicate, hence the predicate symbol is actually only a placeholder for a predicateless symbolic configuration.

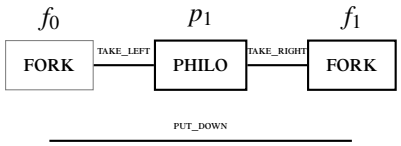

Figure 3.1.: A seat for a right-handed philosopher.

We can use the previous predicate to define a simple chain of right-handed philosophers by linking several seats together:

$$\begin{aligned}\text{chain}(f_0, f_0) &\leftarrow \texttt{emp} \\ \text{chain}(f_0, f_n) &\leftarrow \exists f_1 \,.\, \text{seat}^{\text{R}}(f_0, f_1) * \text{chain}(f_1, f_n).\end{aligned}$$

This predicate contains two rules and, since the second rule can be applied inductively, the predicate can represent chains of arbitrary but finite size. We define the set $\mathcal{R}$ such that it contains exactly the three previously defined rules. An exemplary unfolding might lead to

$$\begin{aligned}\text{chain}(f_0, f_n) \leftarrow^*_{\mathcal{R}} \exists p_1, p_2, f_1 \,.\, &\textsc{take\_left}(p_1, f_0) * \textsc{philo}(p_1) * \textsc{take\_right}(p_1, f_1) \\ &* \textsc{fork}(f_1) * \textsc{put\_down}(p_1, f_1, f_0) \\ &* \textsc{take\_left}(p_2, f_1) * \textsc{philo}(p_2) * \textsc{take\_right}(p_2, f_n) \\ &* \textsc{fork}(f_n) * \textsc{put\_down}(p_2, f_n, f_1) \\ &* \text{chain}(f_1, f_n),\end{aligned}$$

which represents a chain of two philosophers and corresponding forks and relations.

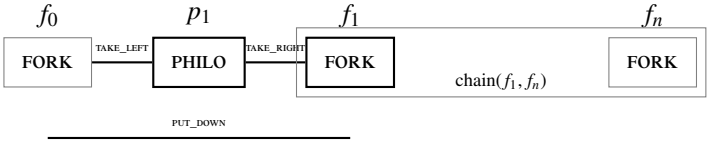

Figure 3.2.: A recursive chain of right-handed philosophers that is unfolded once.

Now, a table is a chain where the first and the last fork are equal. Hence, $\text{chain}(f_0, f_0)$ is a table with zero or more philosophers and forks that alternate each other and are connected

by suitable relations.

Separation logic on BIP configurations $\mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ is a logic derived from separation logic on heaps which enables us to check properties on BIP configurations; whether they are empty, whether they contain certain components, and whether some components are connected in a special way. Moreover, we can check the current state of components and define predicates with systems of inductive definitions to enable more general formulae.

In the next chapter, we define the programming language $\mathcal{L}\langle C, I\rangle$ that is used to write down reconfiguration programs on BIP configurations. The separation logic on BIP configurations enables us to specify pre- and postconditions for those programs (see Chapter 5) and hence prove properties of programs.

4. Reconfiguration Language for BIP

In Chapter 2, we defined BIP configurations as triples containing a BIP system, a state snapshot and a variable mapping. Then we specified the separation logic on BIP $\mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ that is validated on BIP configurations in Chapter 3. It is used to specify properties like the existence of a component of some type, the state of a component or the existence of an interaction between components. Now, we want to define a language that enables us to describe reconfiguration programs on BIP configurations.

DR-BIP While BIP configurations represent settings that occur in the BIP framework, there exists an extension which is called the *dynamically reconfigurable* behavior, interaction, priority framework (DR-BIP). This extended framework uses motifs that combine component and interaction types with reconfiguration programs (see [BBBS18]). Those programs specify how the current BIP system should be transformed into a new system, thus how and if components and interactions may be added or deleted. In Example 5, we defined a BIP system that models a setting of the *dining philosophers problem*. An exemplary reconfiguration program might predefine how to remove a philosopher from the table. This is not merely the action to remove the component PHILO, but one also needs to ensure that the component PHILO is in the correct state (typically state THINKING, since the philosopher should not leave the table if she holds one or two forks) and that the incident interactions and a corresponding fork are deleted or reconnected.

The first step is to define the atomic actions for the addition and deletion of single components and interactions. Furthermore, we want to check properties and then combine actions sequentially or by non-deterministic choice. For this purpose, we specify the programming language $\mathcal{L}\langle C, I\rangle$ that enables us to alter BIP configurations over the signature $\langle C, I\rangle$. Most of the actions are pretty straightforward — only the sequential composition needs further discussion, because it combines two atomic actions into a non-atomic action and hence the states of the components in the BIP configuration can change in between, see Section 2.3.

In this chapter, we define the syntax and semantics of the programming language $\mathcal{L}\langle C, I\rangle$ over the signature $\langle C, I\rangle$, which is used to describe reconfiguration programs on BIP configurations. Furthermore, we assume that the universe $\mathcal{U}$ and the set of variables $\mathcal{V}$ are fixed, countably infinite sets and that $\langle C, I\rangle$ is a signature. For the syntax of the commands, we need the definition of *downward closure* on predicates and $\langle C, I\rangle$-formulae.

Definition 20 (Downward Closure)**.** *Let $p \in \mathcal{P}(\Sigma)$ be a predicate over a separation algebra. Then p is* downward closed *iff $\sigma \in p$ implies that $\sigma' \in p$ for all $\sigma' \preceq \sigma$.*
A $\langle C, I\rangle$-formula ϕ is downward closed if the predicate $p := \{(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle} \mid (\mathfrak{S}, \varsigma, \nu) \models \phi\}$ over the set of BIP configurations is downward closed.

Remember that the order $\preceq$ is defined through the operation $\bullet$ in the separation algebra (see Definition 1). We give the definition of the syntax of the programming language $\mathcal{L}\langle C, I\rangle$ via the Backus-Naur normal form.

Definition 21 (Syntax of the Language $\mathcal{L}\langle C, I\rangle$)**.** *The programming language $\mathcal{L}\langle C, I\rangle$ over the signature $\langle C, I\rangle$ is defined by*

$$\begin{aligned}\ell ::= {} & \texttt{new}(C_i, x) \mid \texttt{delete}(C_i, x) \mid \texttt{connect}(I_j, x_1, \ldots, x_{\alpha(j)}) \mid \\ & \texttt{disconnect}(I_j, x_1, \ldots, x_{\alpha(j)}) \mid \texttt{skip} \mid \texttt{when}\, \phi\, \texttt{do}\, \ell \mid \\ & \texttt{with}\, \psi\, \texttt{do}\, \ell \mid \ell; \ell' \mid \ell + \ell' \mid \ell^*\end{aligned}$$

with $\ell, \ell' \in \mathcal{L}\langle C, I\rangle$, $C_i \in C$ for $1 \leq i \leq n$, $I_j \in I$ for $1 \leq j \leq m$, $x, x_1, \ldots, x_{\alpha(j)} \in \mathcal{V}$ and $\phi, \psi \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ are formulae, where ϕ is downward closed.
The elements in $\mathcal{L}\langle C, I\rangle$ are called programs*, if they are constructed via other elements in $\mathcal{L}\langle C, I\rangle$, or* commands*, if they are primitive.*

The commands $\texttt{new}(C_i, x)$ and $\texttt{delete}(C_i, x)$ signify the addition and deletion of a component of type C_i that is referenced by the variable x. Similarly, the commands $\texttt{connect}(I_j, x_1, \ldots, x_{\alpha(j)})$ and $\texttt{disconnect}(I_j, x_1, \ldots, x_{\alpha(j)})$ stand for the addition and deletion of interactions of type I_j that connect components $x_1, \ldots, x_{\alpha(j)}$. The command `skip` denotes that the BIP configuration does not change. Furthermore, the command $\texttt{when}\, \phi\, \texttt{do}\, \ell$ checks whether the current BIP configuration models the downward closed formula ϕ; if it is a model then the program ℓ is executed directly (the components cannot change their states in between the check and the first execution in ℓ), otherwise it halts silently. Since ϕ is downward closed, it cannot describe the existence of components or interactions, since the empty BIP configuration contains no components or interactions and it is a subconfiguration of each BIP configuration. The command $\texttt{with}\, \psi\, \texttt{do}\, \ell$ combines a check for some properties (like $\texttt{when}\, \phi\, \texttt{do}\, \ell$) with a variable assignment. It tries to map the free variables in the formula ψ to elements in the universe $\mathcal{U}$ such that the current BIP configuration models ψ. If it succeeds, then the program ℓ is executed directly, otherwise it also halts silently.

All of the commands described above are *atomic actions*, hence the states of the components do not change while one of those commands is executed. However, the composition of two or more such commands is not atomic anymore. There are three ways to compose commands: The semicolon combines two commands sequentially, the plus symbol should signify that one of the commands is executed and the choice is non-deterministic. The star is the Kleene operator, thus the command gets executed an arbitrary but finite number of times.

Example 11 (Dining Philosophers). Let $\langle C, I\rangle$ be the signature of the *dining philosophers problem* as defined in Example 5. Then, an exemplary program in the language $\mathcal{L}\langle C, I\rangle$ could be:

Listing 4.1: Create Lonely Philosopher.

```
1 new(PHILO,y);
2 new(FORK,x);
3 new(FORK,z);
```

```
4 connect(TAKE_LEFT,y,x);
5 connect(TAKE_RIGHT,y,z);
6 connect(PUT_DOWN,y,x,z);
```

This program should create a philosopher y and two forks x and z that are arranged such that the philosopher can think, be hungry or eat. See Figure 2.7 for an illustration.

Before we define the semantics of the programming language formally, we define the set of modified variables that will be needed in the reconfiguration rules in Chapter 5.

Definition 22 (Modified Variables)**.** *The set* $\text{Modifies}(\ell) \subset \mathcal{V}$ *of variables that are* modified *by a program* $\ell \in \mathcal{L}\langle C, I\rangle$ *is defined inductively as*

- $\text{Modifies}(\texttt{new}(C_i, x)) = \{x\}$, $\text{Modifies}(\texttt{when}\ \phi\ \texttt{do}\ \ell) = \text{Modifies}(\ell)$, *and* $\text{Modifies}(\texttt{with}\ \psi\ \texttt{do}\ \ell) = \text{fv}(\psi) \cup \text{Modifies}(\ell)$,
- *for any other atomic command* ℓ*, it is* $\text{Modifies}(\ell) = \emptyset$*, and*
- $\text{Modifies}(\ell \star \ell') = \text{Modifies}(\ell) \cup \text{Modifies}(\ell')$ *for* $\star \in \{;, +\}$ *and* $\text{Modifies}(\ell^*) = \text{Modifies}(\ell)$,

where $C_i \in C$ *for* $1 \leq i \leq n$, $\ell, \ell' \in \mathcal{L}\langle C, I\rangle$*, and* $\phi, \psi \in \text{SL}^{\text{BIP}}_{\mathcal{R}}\langle C, I\rangle$ *with* ϕ *is downward closed.*

The programming language $\mathcal{L}\langle C, I\rangle$ describes actions on the set of BIP configurations $\Sigma_{\langle C,I\rangle}$ unified with a new element `error`. This element is returned if the program tries to delete non-existent components or interactions. We extend the set of predicates $\mathcal{P}(\Sigma_{\langle C,I\rangle})$ accordingly by adding a new predicate $\top$, that represents every predicate $p \subseteq \Sigma_{\langle C,I\rangle} \cup \{\texttt{error}\}$ that contains the element `error`. Furthermore, we set

$$\mathcal{P}(\Sigma_{\langle C,I\rangle})^{\top} = \mathcal{P}(\Sigma_{\langle C,I\rangle}) \cup \{\top\}$$

and extend the partial order on the set of predicates by $q \subset \top$ for every $q \in \mathcal{P}(\Sigma_{\langle C,I\rangle})$.

Now we can define the operational semantics of the programming language $\mathcal{L}\langle C, I\rangle$ on BIP configurations by giving transition relations $[\![\ell]\!] \subseteq \Sigma_{\langle C,I\rangle} \times (\Sigma_{\langle C,I\rangle} \cup \{\texttt{error}\})$ for the commands $\ell \in \mathcal{L}\langle C, I\rangle$.

Definition 23 (Operational Semantics of the Programming Language $\mathcal{L}\langle C, I\rangle$)**.** *We give the operational semantics for each command of the language* $\mathcal{L}\langle C, I\rangle$*, where* $\ell : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma', \nu')$ *states that the execution of the program* ℓ *on the BIP configuration* $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$ *leads to the configuration* $(\mathfrak{S}', \varsigma', \nu') \in \Sigma_{\langle C,I\rangle}$.

$$\frac{u \in \mathcal{U}, u \notin C_i^{\mathfrak{S}} \qquad \mathfrak{S}' = \langle\, C_1^{\mathfrak{S}}, \ldots, C_i^{\mathfrak{S}} \cup \{u\}, \ldots, C_n^{\mathfrak{S}}, I_1^{\mathfrak{S}}, \ldots, I_m^{\mathfrak{S}} \,\rangle}{\texttt{new}(C_i, x) : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma[(u, C_i) \leftarrow s_i^0], \nu[x \leftarrow u])}$$

$$\frac{\nu(x) \in C_i^{\mathfrak{S}} \qquad \mathfrak{S}' = \langle\, C_1^{\mathfrak{S}}, \ldots, C_i^{\mathfrak{S}} \setminus \{\nu(x)\}, \ldots, C_n^{\mathfrak{S}}, I_1^{\mathfrak{S}}, \ldots, I_m^{\mathfrak{S}} \,\rangle}{\texttt{delete}(C_i, x) : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma, \nu)}$$

$$\frac{\nu(x) \notin C_i^{\mathfrak{S}}}{\texttt{delete}(C_i, x) : (\mathfrak{S}, \varsigma, \nu) \to \texttt{error}}$$

$$\frac{\mathfrak{S}' = \langle\, C_1^{\mathfrak{S}}, \ldots, C_n^{\mathfrak{S}}, I_1^{\mathfrak{S}}, \ldots, I_i^{\mathfrak{S}} \cup \{(\nu(x_1), \ldots, \nu(x_{\alpha(i)}))\}, \ldots, I_m^{\mathfrak{S}} \,\rangle}{\texttt{connect}(I_i, x_1, \ldots, x_{\alpha(i)}) : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma, \nu)}$$

$$\frac{\begin{array}{c}(\nu(x_1), \ldots, \nu(x_{\alpha(i)})) \in I_i^{\mathfrak{S}} \\ \mathfrak{S}' = \langle\, C_1^{\mathfrak{S}}, \ldots, C_n^{\mathfrak{S}}, I_1^{\mathfrak{S}}, \ldots, I_i^{\mathfrak{S}} \setminus \{(\nu(x_1), \ldots, \nu(x_{\alpha(i)}))\}, \ldots, I_m^{\mathfrak{S}} \,\rangle\end{array}}{\texttt{disconnect}(I_i, x_1, \ldots, x_{\alpha(i)}) : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma, \nu)}$$

$$\frac{(\nu(x_1), \ldots, \nu(x_{\alpha(i)})) \notin I_i^{\mathfrak{S}}}{\texttt{disconnect}(I_i, x_1, \ldots, x_{\alpha(i)}) : (\mathfrak{S}, \varsigma, \nu) \to \texttt{error}}$$

$$\frac{}{\texttt{skip} : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}, \varsigma, \nu)}$$

$$\frac{(\mathfrak{S}, \varsigma, \nu) \models \phi \qquad \ell : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma', \nu')}{\texttt{when}\ \phi\ \texttt{do}\ \ell : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma', \nu')}$$

$$\frac{\begin{array}{c}\mathrm{fv}(\psi) = \{x_1, \ldots, x_i\} \quad u_1, \ldots, u_i \in \mathcal{U} \\ \nu' = \nu[x_1 \leftarrow u_1, \ldots, x_i \leftarrow u_i] \quad (\mathfrak{S}, \varsigma, \nu') \models \psi \quad \ell : (\mathfrak{S}, \varsigma, \nu') \to (\mathfrak{S}', \varsigma', \nu'')\end{array}}{\texttt{with}\ \psi\ \texttt{do}\ \ell : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma', \nu'')}$$

$$\frac{\begin{array}{c}\ell_0 : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma', \nu') \\ \texttt{havoc} : (\mathfrak{S}', \varsigma', \nu') \to (\mathfrak{S}', \varsigma'', \nu') \quad \ell_1 : (\mathfrak{S}', \varsigma'', \nu') \to (\mathfrak{S}'', \varsigma''', \nu'')\end{array}}{\ell_0; \ell_1 : (\mathfrak{S}, \varsigma, \nu) \to \ell_1 : (\mathfrak{S}'', \varsigma''', \nu'')}$$

$$\frac{\ell_i : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma', \nu')}{\ell_0 + \ell_1 : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma', \nu')}\ \textit{for } i = 0, 1$$

$$\frac{}{\ell^* : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}, \varsigma, \nu)}$$

$$\frac{\begin{array}{c}\ell : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma', \nu') \\ \texttt{havoc} : (\mathfrak{S}', \varsigma', \nu') \to (\mathfrak{S}', \varsigma'', \nu') \quad \ell^* : (\mathfrak{S}', \varsigma'', \nu') \to (\mathfrak{S}'', \varsigma''', \nu'')\end{array}}{\ell^* : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}'', \varsigma''', \nu'')}$$

All the triples $(\mathfrak{S}, \varsigma, \nu), (\mathfrak{S}', \varsigma', \nu')$*, etc. are BIP configurations over the signature* $\langle C, I \rangle$*,* ϕ, ψ *are* $\langle C, I \rangle$*-formulae, where* ϕ *is downward closed,* $x, x_1, \ldots, x_{\alpha(j)} \in \mathcal{V}$ *are variables and* $\ell, \ell_0, \ell_1 \in \mathcal{L}\langle C, I \rangle$ *are programs.*

The previous definition starts with the transition relation for the statement $\texttt{new}(C_i, x)$, where $C_i \in C$ is a component type and $x \in \mathcal{V}$ is a variable. If the initial state is a BIP configuration $(\mathfrak{S}, \varsigma, \nu)$, then the final state is the configuration $(\mathfrak{S}', \varsigma[(u, C_i) \leftarrow s_i^0], \nu[x \leftarrow u])$. The statement adds a new component x of type C_i and describes this new component internally by an element $u \in \mathcal{U}$, which is currently unused for this component type. It changes the BIP system $\mathfrak{S}$ to the system $\mathfrak{S}'$ by extending the relation $C_i^{\mathfrak{S}}$ such that $C_i^{\mathfrak{S}'} =$

$C_i^{\mathfrak{S}} \cup \{u\}$. Furthermore, it sets the state of the component, which is described by u, to the initial state $s_i^0 \in \mathbb{S}_i$ of this type, and alters the variable mapping such that x maps to u.

The deletion of components is quite intuitive; if the component does exist, then the matching element $\nu(x) = u$ is deleted from the relation that belongs to the component type, and if the component does not exist then it returns `error`.

The transition relations for the statements `connect` and `disconnect` are similar to the previous transition relations (although `connect` is even simpler than `new` because it assumes that the components that should get connected already exist). The transition relation `when` ϕ `do` ℓ checks whether the initial BIP configuration is a model of ϕ and executes ℓ directly in this case. And `with` ψ `do` ℓ is an extension of the previous command that tries to map the free variables $\{x_1, \ldots, x_i\}$ of the formula ψ (see Definition 15) such that the resulting BIP configuration is a model of ψ.

Before going into the semantics of composed programs, we look at examples of actions on BIP configurations.

Example 12 (Dining Philosophers). The component and interaction types for the dining philosophers problem were already specified in Example 3 and Example 4. We look at two actions on BIP configurations over the signature $\langle C, I \rangle$ of the dining philosophers problem:

- Suppose that we start with an empty BIP configuration $(\mathfrak{S}, \varsigma, \nu) \in \Sigma^{\emptyset}_{\langle C,I \rangle}$, where all the relations in the BIP system $\mathfrak{S}$ are empty. If we execute the program

  $$\texttt{new}(\textsc{philo}, y)$$

  then one possible final state is the BIP configuration

  $$(\mathfrak{S}', \varsigma[(u_2, \textsc{philo}) \leftarrow \textsc{thinking}], \nu[y \leftarrow u_2]), \text{ for } u_2 \in \mathcal{U},$$

  where the only non-empty relation in $\mathfrak{S}'$ is $\textsc{philo}^{\mathfrak{S}'} = \{u_2\}$, the state of the philosopher y is the initial state THINKING and the variable mapping ν is defined such that it maps the variable y to the element u_2. This is also illustrated in Figure 4.1.

- Suppose now that we start in the BIP configuration $(\mathfrak{S}, \varsigma, \nu)$, where the BIP system contains one component of type PHILO described by the element $u_2 \in \mathcal{U}$, and component of type FORK described by the element $u_1 \in \mathcal{U}$ and one interaction of type TAKE_LEFT described by the tuple (u_2, u_1) (hence it connects the philosopher and the fork). Furthermore, both components are in their initial states. If we execute the command

  $$\texttt{disconnect}(\textsc{take\_left}, y, x)$$

  then the new BIP configuration is $(\mathfrak{S}', \varsigma, \nu)$, where the relations in the BIP system $\mathfrak{S}'$ contain only the component u_2 of type PHILO and the component u_1 of type FORK, but no interaction.

The transition relations for sequential composition $(\ell_0; \ell_1)$, non-deterministic choice $(\ell_0 + \ell_1)$, and the Kleene star (ℓ^*) are defined as expected, except for the relation `havoc`, which is in between every two sequentially composed statements. In the next chapters we also want to use the following notation:

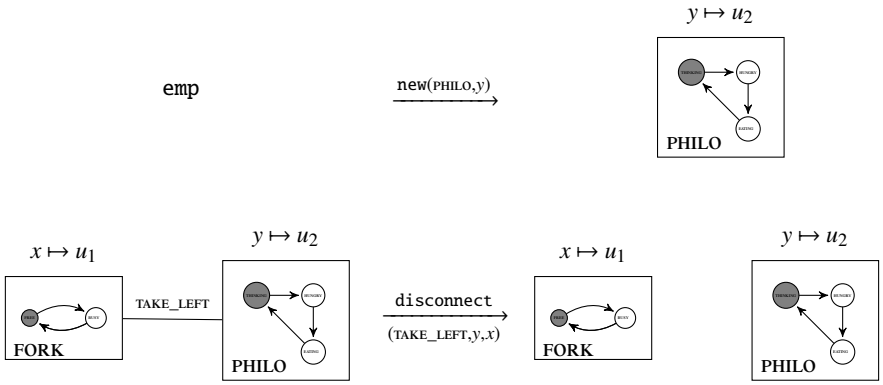

Figure 4.1.: The upper part illustrates the addition of a new component of type PHILO to an empty BIP configuration and the lower part shows the execution of the commands `disconnect`(TAKE_LEFT, y, x) on a BIP configuration.

Definition 24 (Valuation of Programs)**.** *For a program $\ell \in \mathcal{L}\langle C, I\rangle$ we define the function*

$$[\![\ell]\!] : \Sigma_{\langle C,I\rangle} \to \mathcal{P}(\Sigma_{\langle C,I\rangle})^{\top},$$

where $(\mathfrak{S}', \varsigma', \nu') \in [\![\ell]\!]((\mathfrak{S}, \varsigma, \nu))$, for $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$, if and only if the execution of ℓ on $(\mathfrak{S}, \varsigma, \nu)$ might lead to the BIP configuration $(\mathfrak{S}', \varsigma', \nu')$ according to the semantics given in Definition 23.

The relation `havoc` alters only the function ς', thus only the states of the components in the BIP configuration. The intuition behind `havoc` is that commands are atomic but a sequence of commands may not be atomic. Thus the system keeps "working"; interactions may fire and components may change states. Take a look at Section 2.3 for a more precise analysis, where we defined the terms *closed* interaction and *enabled* interaction and a transition function $\rightsquigarrow_c$ for BIP configurations.

We define the semantics of `havoc` and since numerous interactions may fire between the execution of two commands ℓ_0 and ℓ_1, we define the semantics iteratively.

Definition 25 (Operational Semantics `havoc`)**.** *The operational semantics of* `havoc` *in the same form as in Definition 23 as:*

$$\frac{}{\texttt{havoc} : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}, \varsigma, \nu),}$$

$$\frac{(\mathfrak{S}, \varsigma, \nu) \overset{\mathcal{A}}{\rightsquigarrow}_c (\mathfrak{S}, \varsigma', \nu) \qquad \texttt{havoc} : (\mathfrak{S}, \varsigma', \nu) \to (\mathfrak{S}, \varsigma'', \nu)}{\texttt{havoc} : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}, \varsigma'', \nu),}$$

where $(\mathfrak{S}, \varsigma, \nu), (\mathfrak{S}, \varsigma', \nu), (\mathfrak{S}, \varsigma'', \nu) \in \Sigma_{\langle C,I\rangle}$ are BIP configurations, $\mathcal{A} \in \Sigma$ is an interaction atom and $\overset{\mathcal{A}}{\rightsquigarrow}_c$ is the function defined in Definition 12.

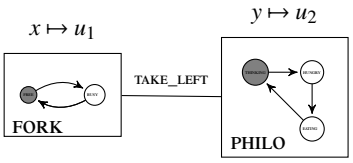

Figure 4.2.: The component x is in state FREE and the component y is in state THINKING, hence the interaction is enabled.

Example 13 (Dining Philosophers). Let us assume that we have a BIP configuration $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$ that contains a philosopher and a fork, which are connected by an interaction of type TAKE_LEFT (see Figure 4.2). The state of the philosopher is THINKING and the state of the fork is FREE.

The only possible transition starting from the current state snapshot is $(\mathfrak{S}, \varsigma, \nu) \overset{\text{TAKE\_LEFT}}{\rightsquigarrow_c} (\mathfrak{S}, \varsigma', \nu)$, where ς' specifies that the philosopher is in state HUNGRY and the fork is in state BUSY. Furthermore, there exists no transition starting from $(\mathfrak{S}, \varsigma', \nu)$. Hence, the execution of havoc leads to a set containing exactly the two BIP configurations $(\mathfrak{S}, \varsigma, \nu)$ and $(\mathfrak{S}, \varsigma', \nu)$.

In this section, we have defined the programming language $\mathcal{L}\langle C, I\rangle$ on the set of BIP configurations that allows us to define reconfiguration programs. In the next chapter, we combine the separation logic $\mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ with the programming language $\mathcal{L}\langle C, I\rangle$ to state and prove pre- and postconditions of programs in $\mathcal{L}\langle C, I\rangle$.

5. Rules for the Verification of Reconfigurations

We have defined BIP configurations, the separation logic on BIP, and the programming language $\mathcal{L}\langle C, I\rangle$ that can be used to describe reconfiguration programs on BIP configurations. We want to be able to verify the partial correctness of those programs using separation logic. For that purpose, we adapt the notion of Hoare triples and define axioms and inference rules.

The first section of this chapter gives axioms and inference rules for reasoning about static BIP configurations. We call this set of inference rules the *reconfiguration rules* and prove their soundness. The soundness of the *frame rule* is proven in the second section. The next section states the axioms and inference rules for reasoning about concurrent BIP configurations (the firing of interactions). This set of rules is called *havoc rules*, since `havoc` is also the name of the relation that describes the state changes, and we also prove their soundness. And lastly, the fourth section contains composition rule for concurrent BIP configurations and the prove of its soundness.

5.1. Reconfiguration Rules

In this section, we give a set of axioms Ax and inference rules for static BIP configurations and prove their soundness. We start by defining *Hoare triples* that resemble the triples defined in [Hoa69], *inference rules*, and *soundness*.

Definition 26 (Hoare Triple)**.** *A* Hoare triple *is a triple*

$$\{\, P \,\}\ \ell\ \{\, Q \,\},$$

where $P, Q \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ *are formulae and* $\ell \in \mathcal{L}\langle C, I\rangle$ *is a program. We call* P *the* precondition *and* Q *the* postcondition*. Furthermore, the Hoare triple is* valid*, if for all* $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$

$$(\mathfrak{S}, \varsigma, \nu) \models P \quad \textit{implies} \quad (\mathfrak{S}', \varsigma', \nu') \models Q, \quad \textit{for all} (\mathfrak{S}', \varsigma', \nu') \in [\![\ell]\!](\mathfrak{S}, \varsigma, \nu).$$

Definition 27 (Inference Rule)**.** *An* inference rule *is a rule*

$$\frac{P_1 \qquad \dots \qquad P_n}{C,}$$

where $P_1, \dots, P_n$ *are* premises *in form of Hoare triples or assertions and* C *is the* conclusion *in form of a Hoare triple.*

Definition 28 (Soundness)**.** *We write* $Ax \vdash \{\ P\ \}\ \ell\ \{\ Q\ \}$ *to state that a Hoare triple can be derived via a set of inference rules from the set of axioms* Ax*. Then the rules are* sound *if every proof-theoretic consequence of some axioms* Ax *is also a semantic consequence of those axioms, hence if*

$$\mathrm{Ax} \vdash \{\ P\ \}\ \ell\ \{\ Q\ \} \quad \textit{implies} \quad \mathrm{Ax} \models \{\ P\ \}\ \ell\ \{\ Q\ \}$$

for $\langle C, I\rangle$*-formulae* P *and* Q *and a program* $\ell \in \mathcal{L}\langle C, I\rangle$*.*

We start by defining the set of axioms Ax and state that the universe $\mathcal{U}$ and the set of variables $\mathcal{V}$ are fixed, countably finite sets. Moreover, the signature $\langle C, I\rangle$ is arbitrary but fixed throughout the rest of this chapter. The axioms are Hoare triples for atomic commands in our programming language $\mathcal{L}\langle C, I\rangle$.

Definition 29 (Small Axioms)**.** *The set of axioms* Ax *contains the Hoare triples*

$$\{\ \texttt{emp}\ \}\ \texttt{new}(C_i, x)\ \{\ C_i(x) \wedge \mathrm{state}(x, s_i^0)\ \},$$
$$\{\ C_i(x)\ \}\ \texttt{delete}(C_i, x)\ \{\ \texttt{emp}\ \},$$
$$\{\ \texttt{emp}\ \}\ \texttt{connect}(I_j, x_1, \ldots, x_{\alpha(j)})\ \{\ I_j(x_1, \ldots, x_{\alpha(j)})\ \},$$
$$\{\ I_j(x_1, \ldots, x_{\alpha(j)})\ \}\ \texttt{disconnect}(I_j, x_1, \ldots, x_{\alpha(j)})\ \{\ \texttt{emp}\ \},\ \textit{and}$$
$$\{\ P\ \}\ \texttt{skip}\ \{\ P\ \},$$

where $C_i \in C$ *and* $I_j \in I$*, and* $x, x_1, \ldots, x_{\alpha(j)} \in \mathcal{V}$*.*

Theorem 2. *The* small axioms Ax *defined in Definition 29 are valid.*

Proof. We prove the validity in the case that a BIP configuration is mapped to another BIP configuration. The case that a BIP configuration is mapped to the `error` state can be suppressed, since we define our rules for partial correctness only.

- Let $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$ be a BIP configuration such that $(\mathfrak{S}, \varsigma, \nu) \models \texttt{emp}$. We use the operational semantics of Definition 23 and obtain

$$\texttt{new}(C_i, x) : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma', \nu')$$
$$\text{such that } \mathfrak{S}' = \langle\ C_1^{\mathfrak{S}}, \ldots, C_i^{\mathfrak{S}} \cup \{u\}, \ldots, C_n^{\mathfrak{S}}, I_1^{\mathfrak{S}}, \ldots, I_m^{\mathfrak{S}}\ \rangle = \langle\ \emptyset, \ldots, C_i^{\mathfrak{S}'} = \{u\}, \ldots, \emptyset\ \rangle,$$
$$\varsigma' = \varsigma[(u, C_i) \leftarrow s_i^0] \text{ and } \nu' = \nu[x \leftarrow u]$$

  for some $u \in \mathcal{U}$. Then $(\mathfrak{S}', \varsigma', \nu') \models C_i(x)$ and $(\mathfrak{S}', \varsigma', \nu') \models \mathrm{state}(x, s_i^0)$ according to the semantics given in Definition 17. Thus $(\mathfrak{S}', \varsigma', \nu') \models C_i(x) \wedge \mathrm{state}(x, s_i^0)$.

- Assume $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$ such that $(\mathfrak{S}, \varsigma, \nu) \models C_i(x)$. Then $\mathfrak{S} = (\emptyset, \ldots, C_i^{\mathfrak{S}} = \{\nu(x)\}, \ldots, \emptyset)$ and using the operational semantics we obtain

$$\texttt{delete}(C_i, x) : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma, \nu)$$
$$\text{such that } \mathfrak{S}' = \langle\ C_1^{\mathfrak{S}}, \ldots, C_i^{\mathfrak{S}} \setminus \{\nu(x)\}), \ldots, C_n^{\mathfrak{S}}, I_1^{\mathfrak{S}}, \ldots, I_m^{\mathfrak{S}}\ \rangle = \langle\ \emptyset, \ldots, \emptyset\ \rangle.$$

  Then $(\mathfrak{S}', \varsigma, \nu) \models \texttt{emp}$.

- The proof of the validity for the axioms `connect` and `disconnect` follows analoguously to the proof of the validity of `new` and `delete`.
- Assume $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$ such that $(\mathfrak{S}, \varsigma, \nu) \models P$. Then

  $$\texttt{skip} : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}, \varsigma, \nu),$$

  hence $(\mathfrak{S}, \varsigma, \nu) \models P$ and the axiom is valid.

It follows that the small axioms are valid. □

Most of the inference rules are standard and inspired by the rules for *abstract separation logic* given in [COY07]. Only the rules for $\texttt{with}\,\psi\,\texttt{do}$, sequential composition and the *frame rule* need special attention.

Definition 30 (Basic Constructs)**.** *The basic constructs are*

$$\frac{\{\, P \wedge \phi \,\}\ \ell\ \{\, Q \,\}}{\{\, P \,\}\ \texttt{when}\ \phi\ \texttt{do}\ \ell\ \{\, Q \,\},}\ (\texttt{when}\ \phi\ \texttt{do})$$

$$\frac{\{\, \exists y_1, \ldots, y_i.\ P \wedge \psi[x_1/y_1, \ldots, x_i/y_i] * \texttt{true} \,\}\ \ell\ \{\, Q \,\}}{\{\, P \,\}\ \texttt{with}\ \psi\ \texttt{do}\ \ell\ \{\, Q \,\}}\ (\texttt{with}\ \psi\ \texttt{do})$$
where $\{x_1, \ldots, x_i\} \in \mathrm{fv}(\psi)$ *and* $y_1, \ldots, y_n \in \mathcal{V}$*,*

$$\frac{\{\, P \,\}\ \ell_0\ \{\, P' \,\} \qquad \{\, P' \,\}\ \texttt{havoc}\ \{\, Q' \,\} \qquad \{\, Q' \,\}\ \ell_1\ \{\, Q \,\}}{\{\, P \,\}\ \ell_0; \ell_1\ \{\, Q \,\},}\ (;)$$

$$\frac{\{\, P \,\}\ \ell_0\ \{\, Q \,\} \qquad \{\, P \,\}\ \ell_1\ \{\, Q \,\}}{\{\, P \,\}\ \ell_0 + \ell_1\ \{\, Q \,\},}\ (+)$$

$$\frac{\{\, P \,\}\ \ell\ \{\, P \,\} \qquad \{\, P \,\}\ \texttt{havoc}\ \{\, P \,\}}{\{\, P \,\}\ \ell^*\ \{\, P \,\},}\ (^*)$$

where $P, P', Q, Q', \phi, \psi \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ *are formulae, where* ϕ *is downward closed, and* $\ell, \ell_0, \ell_1 \in \mathcal{L}\langle C, I\rangle$ *are programs.*

Theorem 3. *The* basic constructs *given in Definition 30 are sound.*

Proof. We prove the soundness of the inference rules one by one.

- Let $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$ be a BIP configuration such that $(\mathfrak{S}, \varsigma, \nu) \models P$ and suppose that the premise $\{\, P \wedge \phi \,\}\ \ell\ \{\, Q \,\}$ is true. If $(\mathfrak{S}, \varsigma, \nu) \models \phi$ then it follows from the operational semantics of the programming language (given in Definition 23) that

  $$\texttt{when}\ \phi\ \texttt{do}\ \ell : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma', \nu'),$$
  $$\text{where } \ell : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma', \nu').$$

  Since $(\mathfrak{S}, \varsigma, \nu) \models P \wedge \phi$, we derive from the premise that $(\mathfrak{S}', \varsigma', \nu') \models Q$.

- Let $(\mathfrak{S},\varsigma,\nu) \in \Sigma_{\langle C,I\rangle}$ be a BIP configuration such that $(\mathfrak{S},\varsigma,\nu) \models P$ and we assume that the premise $\{\ \exists y_1,\ldots,y_i.\ P \wedge \psi[x_1/y_1,\ldots,x_i/y_i] * \texttt{true}\ \}\ \ell\ \{\ Q\ \}$ holds. Then we use the operational semantics and obtain

$$\texttt{when}\ \psi\ \texttt{do}\ \ell : (\mathfrak{S},\varsigma,\nu) \to (\mathfrak{S}',\varsigma',\nu'')$$

if there exists a $\nu' = \nu[x_1 \leftarrow u_1,\ldots,x_i \leftarrow u_i]$ for $\mathrm{fv}(\psi) = \{x_1,\ldots,x_i\}$ and some elements $u_1,\ldots,u_i \in \mathcal{U}$ such that $(\mathfrak{S},\varsigma,\nu') \models P \wedge \psi * \texttt{true}$ and $\ell : (\mathfrak{S},\varsigma,\nu') \to (\mathfrak{S}',\varsigma',\nu'')$.

It follows via the premise that $(\mathfrak{S}',\varsigma',\nu'') \models Q$.

- Assume $(\mathfrak{S},\varsigma,\nu) \in \Sigma_{\langle C,I\rangle}$ such that $(\mathfrak{S},\varsigma,\nu) \models P$ and we presume that the premises $\{\ P\ \}\ \ell_0\ \{\ P'\ \}$, $\{\ P'\ \}\ \texttt{havoc}\ \{\ Q'\ \}$ and $\{\ Q'\ \}\ \ell_1\ \{\ Q\ \}$ hold. From the operational semantics we obtain

$$\begin{gathered}\ell_0;\ell_1 : (\mathfrak{S},\varsigma,\nu) \to (\mathfrak{S}'',\varsigma''',\nu'')\\ \text{with } \ell_0 : (\mathfrak{S},\varsigma,\nu) \to (\mathfrak{S}',\varsigma',\nu'),\ \texttt{havoc} : (\mathfrak{S}',\varsigma',\nu') \to (\mathfrak{S}',\varsigma'',\nu')\\ \text{and } \ell_1 : (\mathfrak{S}',\varsigma'',\nu') \to (\mathfrak{S}'',\varsigma''',\nu'').\end{gathered}$$

Since $(\mathfrak{S},\varsigma,\nu) \models P$, we derive $(\mathfrak{S}',\varsigma',\nu') \models P'$, furthermore $(\mathfrak{S}',\varsigma'',\nu') \models Q'$ and $(\mathfrak{S}'',\varsigma''',\nu'') \models Q$ from the premises. Thus the rule is sound.

- Assume $(\mathfrak{S},\varsigma,\nu) \in \Sigma_{\langle C,I\rangle}$ such that $(\mathfrak{S},\varsigma,\nu) \models P$. We suppose that the premises $\{\ P\ \}\ \ell_i\ \{\ Q\ \}$ hold for $i = 0,1$. It follows from the operational semantics that

$$\begin{gathered}\ell_0 + \ell_1 : (\mathfrak{S},\varsigma,\nu) \to (\mathfrak{S}',\varsigma',\nu')\\ \text{if } \ell_i : (\mathfrak{S},\varsigma,\nu) \to (\mathfrak{S}',\varsigma',\nu') \text{ for an } i \in \{0,1\}\end{gathered}$$

and, since $\{\ P\ \}\ \ell_i\ \{\ Q\ \}$, we know that $(\mathfrak{S}',\varsigma',\nu') \models Q$ and the rule is sound.

- Assume $(\mathfrak{S},\varsigma,\nu) \in \Sigma_{\langle C,I\rangle}$ such that $(\mathfrak{S},\varsigma,\nu) \models P$. We suppose that the premises $\{\ P\ \}\ \ell\ \{\ P\ \}$ and $\{\ P\ \}\ \texttt{havoc}\ \{\ P\ \}$ hold and we show by natural induction that $\{\ P\ \}\ \ell^n\ \{\ P\ \}$ follows for any $n \in \mathbb{N}_0$.
  - Base Case: Suppose $n = 0$, then $(\mathfrak{S},\varsigma,\nu) \models P$.
  - Induction Hypothesis: $\{\ P\ \}\ \ell^n\ \{\ P\ \}$ holds for some arbitrary but fixed $n \in \mathbb{N}_0$.
  - Induction Step: Assume that $\{\ P\ \}\ \ell^n\ \{\ P\ \}$ holds. We know from the operational semantics that

$$\begin{gathered}\ell^* : (\mathfrak{S},\varsigma,\nu) \to (\mathfrak{S}'',\varsigma''',\nu'')\\ \text{if } \ell : (\mathfrak{S},\varsigma,\nu) \to (\mathfrak{S}',\varsigma',\nu'),\ \texttt{havoc} : (\mathfrak{S}',\varsigma',\nu') \to (\mathfrak{S}',\varsigma'',\nu')\\ \text{and } \ell^* : (\mathfrak{S}',\varsigma'',\nu') \to (\mathfrak{S}'',\varsigma''',\nu'').\end{gathered}$$

    and derive from the premises that $(\mathfrak{S}',\varsigma',\nu') \models P$, $(\mathfrak{S}',\varsigma'',\nu') \models P$ and from the induction hypothesis we obtain $(\mathfrak{S}'',\varsigma''',\nu'') \models P$.

Hence the rule follows by natural induction.

It follows that the basic constructs are sound. □

Whereas the basic constructs state partial correctness through analysing the structure of a program ℓ, the following rules study the structure of the pre- and postconditions in the Hoare triples. This includes the conjunction and intersection of conditions and implications.

Definition 31 (Structural Rules)**.** *The structural rules are*

- *the* rules for conjunction and disjunction*:*

$$\frac{\{\, P_i \,\}\ \ell\ \{\, Q_i \,\},\ \textit{all } i \in I}{\{\, \bigvee_{i\in I} P_i \,\}\ \ell\ \{\, \bigvee_{i\in I} Q_i \,\}}\ (\vee) \qquad \frac{\{\, P_i \,\}\ \ell\ \{\, Q_i \,\},\ \textit{all } i \in I}{\{\, \bigwedge_{i\in I} P_i \,\}\ \ell\ \{\, \bigwedge_{i\in I} Q_i \,\}}\ (\wedge)$$

- *and the* rule of consequence*:*

$$\frac{P' \subseteq P \qquad \{\, P \,\}\ \ell\ \{\, Q \,\} \qquad Q \subseteq Q'}{\{\, P' \,\}\ \ell\ \{\, Q' \,\}.}\ (c)$$

Theorem 4. *The rules for conjunction and disjunction and the rule of consequence given in Definition 31 are sound.*

Proof.

- Let $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$ such that $(\mathfrak{S}, \varsigma, \nu) \models \bigvee_{i\in I} P_i$. Then $(\mathfrak{S}, \varsigma, \nu) \models P_i$ for some $i \in I$ and we derive from the premises that

$$\ell : (\mathfrak{S}, \varsigma, \nu) \to (\mathfrak{S}', \varsigma', \nu')$$

  and $(\mathfrak{S}', \varsigma', \nu') \models Q_i$. Hence $(\mathfrak{S}', \varsigma', \nu') \models \bigvee_{i\in I} Q_i$.

- The *rule for disjunction* and the *rule of consequence* follow similarly.

□

The *frame rule* is also a structural rule, but since its definition and the proof of its soundness require multiple pages, we specify it separately. First, we define a subset of the programming language $\mathcal{L}\langle C, I\rangle$ that contains all the programs in the language $\mathcal{L}\langle C, I\rangle$ that omit `with` ψ `do`, sequential composition and the Kleene star.

Definition 32 (Subset of the Programming Language)**.** *We define the subset* $\mathcal{L}_X\langle C, I\rangle \subset \mathcal{L}\langle C, I\rangle$ *of the programming language by*

$$\begin{aligned}\ell \ ::= \ & \texttt{new}(C_i, x) \mid \texttt{delete}(C_i, x) \mid \texttt{connect}(I_j, x_1, \ldots, x_{\alpha(j)}) \mid \\ & \texttt{disconnect}(I_j, x_1, \ldots, x_{\alpha(j)}) \mid \texttt{skip} \mid \texttt{when}\ \phi\ \texttt{do}\ \ell \mid \ell + \ell',\end{aligned}$$

where $\ell, \ell' \in \mathcal{L}_X\langle C, I\rangle$, $C_i \in C$ *for* $1 \le i \le n$, $I_j \in I$ *for* $1 \le j \le m$, $x, x_1, \ldots x_{\alpha(j)} \in \mathcal{V}$ *and* $\phi \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ *is downward closed.*

We define the *frame rule* on the programs in the subset $\mathcal{L}_X\langle C, I\rangle$.

Definition 33 (Frame Rule)**.** *The* frame rule *is*

$$\frac{\{\, P \,\}\ \ell\ \{\, Q \,\}}{\{\, P * F \,\}\ \ell\ \{\, Q * F \,\},}\ (*)$$

where $P, Q, F \in \mathrm{SL}^{\mathrm{BIP}}_{\mathcal{R}}\langle C, I\rangle$ *are formulae and* $\ell \in \mathcal{L}_X\langle C, I\rangle$ *such that* $\mathrm{Modifies}(\ell) \cap \mathrm{fv}(F) = \emptyset$*, hence the program* ℓ *does not modify any free variable in* F*.*

The frame rule can only be applied for atomic programs $\ell \in \mathcal{L}_X\langle C, I\rangle$ and a frame that is not altered by the program ℓ. We may apply the frame rule (and the axiom for `connect`) in the following proof (compare with Figure 4.2):

$$\frac{\{\, \texttt{emp} \,\}\ \texttt{connect}(\textsc{take\_left}, y, x)\ \{\, \textsc{take\_left}(y, x) \,\}}{\{\, \textsc{fork}(x) * \textsc{philo}(y) \,\}\ \texttt{connect}(\textsc{take\_left}, y, x)\ \{\, \textsc{fork}(x) * \textsc{philo}(y) * \textsc{take\_left}(y, x) \,\}.}\ (*)$$

But it may not be applied to prove the Hoare triple

$$\begin{gathered}\{\, \mathrm{chain}(f_0, f_n) \,\} \\ \texttt{new}(\textsc{philo}, p);\ \texttt{connect}(\textsc{take\_right}, p, f_0);\ \texttt{new}(\textsc{fork}, f) \\ \{\, \textsc{fork}(f) * \textsc{philo}(p) * \textsc{take\_left}(p, f_0) * \mathrm{chain}(f_0, f_n) \,\}\end{gathered}$$

(see Figure 3.2), because interactions could fire in the frame (which is the $\mathrm{chain}(f_0, f_n)$) in between the composition of the commands.

5.2. Relaxing Locality

The *frame rule* originates from separation logic on heaps and uses the *locality property* to simplify the verification of programs on heaps. It states that if a heap is a model of some assertion P and a program alters the cells, then the same program would only alter a special part of a heap that is a model of $P * F$, namely the part that models P (assuming that we use the theory given in [IO01], where all the programs are local). The rule was generalized in [COY07] for *abstract separation logic* and proven to be sound and complete for local actions on separation algebras (see Definition 4 for the definition of *locality*).

In our case, we cannot guarantee that all programs $\ell \in \mathcal{L}\langle C, I\rangle$ are local since the action `havoc` and the atomic actions `new` and `with` ψ `do` are non-local. The action `havoc` cannot be adapted, hence the frame rule is restricted such that it only applies to programs that do not trigger `havoc` implicitly. A restricted version of `with` ψ `do` is the action `when` ϕ `do`, where ϕ is restricted to downward closed formulae. Hence we exclude the first action. Furthermore, we generalize the notion of locality to *X-locality* and show that the action `new` is X-local.

In this section, we show which atomic actions are local, we define X-locality and prove that most of the actions are X-local, and finally, we show the soundness of the frame rule defined in Definition 33. First, we show the locality of some of the actions.

Lemma 1. *The atomic actions* `delete`*,* `connect`*,* `disconnect`*,* `skip`*, and* `when` ϕ `do` ℓ *given in Definition 23 are local actions, if* $\ell \in \mathcal{L}\langle C, I\rangle$ *is a local action.*

Proof. We start by proving the locality of the action `delete`: Let $(\Im_0, \varsigma_0, \nu_0), (\Im_1, \varsigma_1, \nu_1) \in \Sigma_{\langle C,I\rangle}$ be two BIP configurations such that $(\Im_0, \varsigma_0, \nu_0) \bullet (\Im_1, \varsigma_1, \nu_1)$ is defined. Then $\nu_0 = \nu_1$ and hence we can distinguish three cases for the action $\texttt{delete}(C_i, x)$:

- Suppose $\nu_0(x) \in C_i^{\mathfrak{S}_0}$. Then

  $$[\![\texttt{delete}]\!](C_i, x)((\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1)) = [\![\texttt{delete}]\!](C_i, x)((\mathfrak{S}_0, \varsigma_0, \nu_0)) * \{(\mathfrak{S}_1, \varsigma_1, \nu_1)\},$$

  since the result is a BIP configuration $(\mathfrak{S}, \varsigma, \nu)$, where $\mathfrak{S} = \langle\, C_1^{\mathfrak{S}}, \ldots, C_n^{\mathfrak{S}}, I_1^{\mathfrak{S}}, \ldots, I_m^{\mathfrak{S}} \,\rangle$ contains the elements of $\mathfrak{S}_0$ and $\mathfrak{S}_1$ minus the element $\nu_0(x)$, $\varsigma = \varsigma_0$ and $\nu = \nu_0$.

- Suppose $\nu_0(x) \in C_i^{\mathfrak{S}_1}$. Then

  $$[\![\texttt{delete}]\!](C_i, x)((\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1))$$

  results in the same structure as in the first case, but

  $$[\![\texttt{delete}]\!](C_i, x)((\mathfrak{S}_0, \varsigma_0, \nu_0)) * \{(\mathfrak{S}_1, \varsigma_1, \nu_1)\} = \top * \{(\mathfrak{S}_1, \varsigma_1, \nu_1)\} = \top,$$

  since $\nu_0(x) \notin C_i^{\mathfrak{S}_0}$ and hence the operational semantics map $(\mathfrak{S}_0, \varsigma_0, \nu_0)$ to the `error` state. The subset relation holds, since $p \subset \top$ for every $p \in \mathcal{P}(\Sigma_{\langle C,I \rangle})$.

- Suppose that $\nu_0(x) \notin C_i^{\mathfrak{S}_0} \cup C_i^{\mathfrak{S}_1}$, then both

  $$[\![\texttt{delete}]\!](C_i, x)((\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1)) = \top$$
  $$\text{and } [\![\texttt{delete}]\!](C_i, x)((\mathfrak{S}_0, \varsigma_0, \nu_0)) = \top$$

  map $(\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1)$ resp. $(\mathfrak{S}_0, \varsigma_0, \nu_0)$ to the `error` state. Hence

  $$[\![\texttt{delete}]\!](C_i, x)((\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1)) = [\![\texttt{delete}]\!](C_i, x)((\mathfrak{S}_0, \varsigma_0, \nu_0)) * \{(\mathfrak{S}_1, \varsigma_1, \nu_1)\}.$$

Thus `delete` is a local action. The locality condition for the actions `connect` and `disconnect` can be proven similarly. And the action `skip` is local, since

$$[\![\texttt{skip}]\!]((\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1)) = [\![\texttt{skip}]\!]((\mathfrak{S}_0, \varsigma_0, \nu_0)) * \{(\mathfrak{S}_1, \varsigma_1, \nu_1)\}$$

for any BIP configurations $(\mathfrak{S}_0, \varsigma_0, \nu_0), (\mathfrak{S}_1, \varsigma_1, \nu_1) \in \Sigma_{\langle C,I \rangle}$.

And last but not least, we assume that $(\mathfrak{S}_0, \varsigma_0, \nu_0), (\mathfrak{S}_1, \varsigma_1, \nu_1) \in \Sigma_{\langle C,I \rangle}$ are BIP configurations such that $(\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1)$ is defined. We analyze two cases for the action $\texttt{when}\ \phi\ \texttt{do}\ \ell$:

- Assume that $(\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1) \models \phi$, then $(\mathfrak{S}_0, \varsigma_0, \nu_0) \models \phi$, because ϕ is downward closed. Hence

  $$[\![\texttt{when}\ \phi\ \texttt{do}\ \ell]\!]((\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1)) = [\![\ell]\!]((\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1))$$
  $$\subseteq [\![\ell]\!]((\mathfrak{S}_0, \varsigma_0, \nu_0)) * \{(\mathfrak{S}_1, \varsigma_1, \nu_1)\} = [\![\texttt{when}\ \phi\ \texttt{do}\ \ell]\!]((\mathfrak{S}_0, \varsigma_0, \nu_0)) * \{(\mathfrak{S}_1, \varsigma_1, \nu_1)\}.$$

- Now we assume that $(\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1) \not\models \phi$, but $(\mathfrak{S}_0, \varsigma_0, \nu_0) \models \phi$. Then it is

  $$[\![\texttt{when}\ \phi\ \texttt{do}\ \ell]\!]((\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1)) = \emptyset \subseteq [\![\texttt{when}\ \phi\ \texttt{do}\ \ell]\!]((\mathfrak{S}_0, \varsigma_0, \nu_0)) * \{(\mathfrak{S}_1, \varsigma_1, \nu_1)\}.$$

Hence the locality of the action $\texttt{when}\ \phi\ \texttt{do}\ \ell$ follows. □

Most of the atomic actions are local, but the action new is an exception. This is due to the non-deterministic choice of the element $u \in \mathcal{U}$ that uniquely defines the new component.

Lemma 2. *The atomic action* new *given in Definition 23 is non-local.*

Proof. We show that the action $\texttt{new}(C_i, x)$ is non-local by assuming that the locality condition holds and giving a counterexample.
Let $(\mathfrak{S}^{\emptyset}, \varsigma, \nu) \in \Sigma^{\emptyset}_{\langle C,I\rangle}$ be an empty BIP configuration and $u \in \mathcal{U}$ an arbitrary but fixed element. Furthermore $\varsigma(u, C_i) \neq s_i^0$ and $\nu(x) \neq u$. Then $(\mathfrak{S}^{\emptyset}, \varsigma, \nu) \bullet (\mathfrak{S}^{\emptyset}, \varsigma, \nu) = (\mathfrak{S}^{\emptyset}, \varsigma, \nu)$ is defined and

$$(\mathfrak{S}', \varsigma[(u, C_i) \leftarrow s_i^0], \nu[x \leftarrow u]) \in \texttt{new}(C_i, x)((\mathfrak{S}^{\emptyset}, \varsigma, \nu) \bullet (\mathfrak{S}^{\emptyset}, \varsigma, \nu)) = \texttt{new}(C_i, x)((\mathfrak{S}^{\emptyset}, \varsigma, \nu)).$$

But $(\mathfrak{S}', \varsigma[(u, C_i) \leftarrow s_i^0], \nu[x \leftarrow u]) \notin \texttt{new}(C_i, x)((\mathfrak{S}^{\emptyset}, \varsigma, \nu)) * \{(\mathfrak{S}^{\emptyset}, \varsigma, \nu)\} = \emptyset$, since the concatenation via the operation $*$ contains only elements, where $\varsigma[(u, C_i) \leftarrow s_i^0] = \varsigma$ and $\nu[x \leftarrow u] = \nu$ hold. This is not true by the definition of ς and ν for any $u \in \mathcal{U}$ and hence new is non-local. □

We want to specify a relaxation of the locality condition that holds for the action new and that allows us to prove the soundness of the frame rule.

The counterexample in the proof of Lemma 2 indicates the problem with the original locality condition — the concatenation via the operation $*$ fails because new alters the total functions ς and ν. We bypass this by defining a *lifted set*, which contains all BIP configurations that differ from original BIP configurations only at certain fixed positions in the functions ς and ν. This lifted set will be used for the specification of X-locality in Definition 35.

Definition 34 (Lifted Set)**.** *Let $X \subset \mathcal{V} \times C$ be a set of tuples (x, C_i), $\mathcal{V}(X) = \{x \mid (x, C_i) \in X\}$ is the set of variables used in X and $P \in \mathcal{P}(\Sigma_{\langle C,I\rangle})$ a set of BIP configurations. Then we define the* lift *of P on the set X as*

$$\begin{aligned} P{\uparrow_X} := \{(\mathfrak{S}, \varsigma', \nu') \mid\ & (\mathfrak{S}, \varsigma, \nu) \in P, \\ & \nu'(y) = \nu(y) \text{ for all } y \in \mathcal{V} \setminus \mathcal{V}(X) \\ & \text{and for all } (u, C_i) \in \mathcal{U} \times C \\ & \quad \text{if } (u, C_i) \neq (\nu'(x), C_i) \text{ for all } (x, C_i) \in X, \text{ then } \varsigma'(u, C_i) = \varsigma(u, C_i)\}. \end{aligned}$$

We give an abstract definition of X-*locality* first. If the definition of the lifted set is adapted, then it can also be used in the store-heap model for the assignment x:=y, which is not local but can be proven to be X-local for $X \subset \mathcal{V}$.

Definition 35 (X-Locality)**.** *Let X be a set such that the lift $P{\uparrow_X}$ of any set $P \in \mathcal{P}(\Sigma)$ is defined for a separation algebra Σ. Then an action $f : \Sigma \to \mathcal{P}(\Sigma)^{\top}$ is* X-local *iff for all $\sigma_0, \sigma_1 \in \Sigma$ such that $\sigma_0 \bullet \sigma_1$ is defined, $f(\sigma_0 \bullet \sigma_1) \subseteq f(\sigma_0) * \{\sigma_1\}{\uparrow_X}$.*

The sequential composition, non-deterministic choice, and iteration of X-local actions on static systems (thus without implicit state changes) are again X-local actions. Look at Appendix A for a proof.

The application of X-locality on BIP configurations leads to more insights. We can show that a local action is also an X-local action for a certain X.

Lemma 3. *Let $f : \Sigma_{\langle C,I\rangle} \to \mathcal{P}(\Sigma_{\langle C,I\rangle})^\top$ be a local action. Then f is also an $\emptyset$-local action.*

Proof. The action f is local, thus $f(\sigma_0 \bullet \sigma_1) \subseteq f(\sigma_0) * \{\sigma_1\} = f(\sigma_0) * \{\sigma_1\}\uparrow_\emptyset$, since $\{\sigma_1\}\uparrow_\emptyset = \{\sigma_1\}$. Hence f is $\emptyset$-local. □

Since we have shown that `delete`, `connect`, `disconnect`, `skip`, and `when` ϕ `do` are local actions, they are also $\emptyset$-local actions. In Appendix A we show that `when` ϕ `do` ℓ is also an X-local action if ℓ is X-local. Furthermore, `new` is an $\{(x, C_i)\}$-local action.

Lemma 4. *The action* `new` *given in Definition 21 is $\{(x, C_i)\}$-local.*

Proof. Let $(\mathfrak{S}_0, \varsigma_0, \nu_0), (\mathfrak{S}_1, \varsigma_1, \nu_1) \in \Sigma_{\langle C,I\rangle}$ be two BIP configurations such that $(\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1)$ is defined. Then

$$\begin{aligned}&\texttt{new}(C_i, x)((\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1))\\ = \{(\mathfrak{S}, \varsigma, \nu) \mid \mathfrak{S} = \mathfrak{S}_0 \uplus \mathfrak{S}_1 \uplus \{u\},\ &\nu = \nu_0[x \leftarrow u] \text{ and } \varsigma = \varsigma_0[(u, C_i) \to s_i^0],\ u \notin C_i^{\mathfrak{S}_0 \cup \mathfrak{S}_1}\},\end{aligned}$$

which equals

$$\begin{aligned}\{(\mathfrak{S}, \varsigma, \nu) \mid \mathfrak{S} = \mathfrak{S}_0 \uplus \{u\},\ &\nu = \nu_0[x \leftarrow u] \text{ and } \varsigma = \varsigma_0[(u, C_i) \to s_i^0],\ u \notin C_i^{\mathfrak{S}_0}\}\\ &*\{(\mathfrak{S}_1, \varsigma_1, \nu_1)\}\uparrow_{\{(x,C_i)\}}\\ = \texttt{new}(C_i, x)((\mathfrak{S}_0, \varsigma_0, \nu_0)) &* \{(\mathfrak{S}_1, \varsigma_1, \nu_1)\}\uparrow_{\{(x,C_i)\}},\end{aligned}$$

since the concatenation via the operation $*$ is only possible if $u \notin C_i^{\mathfrak{S}_1}$. Thus $\texttt{new}(C_i, x)$ is $\{(x, C_i)\}$-local. □

Now we can combine all of the previous results about locality and X-locality and prove the soundness of the *frame rule*.

Theorem 5. *The* frame rule *given in Definition 33 is sound.*

Proof. We start by proving that the program ℓ is X-local for some set $X \subset \mathcal{V} \times C$. Then we use the X-locality to show the frame rule.

Since $\ell \in \mathcal{L}_X\langle C, I\rangle$, we have to show that each of the atomic commands is X-local and furthermore, that compositions via the non-deterministic choice + are X-local: The X-locality of the atomic commands follows via Lemma 3, Lemma 4, and Lemma 15. Furthermore, the composition of X-local programs via non-deterministic choice is again X-local, see Lemma 13. Hence every program $\ell \in \mathcal{L}_X\langle C, I\rangle$ is X-local for some minimal set $X \subset \mathcal{V} \times C$.

Suppose that $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$ is a BIP configuration such that there exist $(\mathfrak{S}_P, \varsigma_P, \nu_P)$, $(\mathfrak{S}_R, \varsigma_R, \nu_R) \in \Sigma_{\langle C,I\rangle}$ with

$$(\mathfrak{S}, \varsigma, \nu) = (\mathfrak{S}_P, \varsigma_P, \nu_P) \bullet (\mathfrak{S}_R, \varsigma_R, \nu_R),$$

$(\mathfrak{S}_P, \varsigma_P, \nu_P) \models P$ and $(\mathfrak{S}_R, \varsigma_R, \nu_R) \models R$. It follows that $(\mathfrak{S}, \varsigma, \nu) \models P * R$. Furthermore we assume that the premise $\{ P \}\ \ell\ \{ Q \}$ holds, hence $[\![\ell]\!](\mathfrak{S}_P, \varsigma_P, \nu_P) \models Q$. Then it follows from the X-locality of the program ℓ that

$$[\![\ell]\!]((\mathfrak{S}_P, \varsigma_P, \nu_P) \bullet (\mathfrak{S}_R, \varsigma_R, \nu_R)) \subseteq \underbrace{[\![\ell]\!]((\mathfrak{S}_P, \varsigma_P, \nu_P))}_{\models Q} * \{(\mathfrak{S}_R, \varsigma_R, \nu_R)\}\!\uparrow_X .$$

Without loss of generality, we can assume that X is chosen as in Lemma 14. Then it follows that $\mathcal{V}(X) \subseteq \text{Modifies}(\ell)$. Furthermore, we require that $\text{Modifies}(\ell) \cap \text{fv}(R) = \emptyset$, hence $\mathcal{V}(X) \cap \text{fv}(R) = \emptyset$. It follows that $\{(\mathfrak{S}_R, \varsigma_R, \nu_R)\}\!\uparrow_X \models R$ and finally

$$[\![\ell]\!]((\mathfrak{S}_P, \varsigma_P, \nu_P) \bullet (\mathfrak{S}_R, \varsigma_R, \nu_R)) \subseteq \underbrace{[\![\ell]\!]((\mathfrak{S}_P, \varsigma_P, \nu_P))}_{\models Q} * \underbrace{\{(\mathfrak{S}_R, \varsigma_R, \nu_R)\}\!\uparrow_X}_{\models R} \models Q * R.$$

Hence we have proven the soundness of the frame rule. □

5.3. Havoc Rules

In this section, we give rules for reasoning about executing interactions on BIP configurations (respectively the state changes that occur if interactions are triggered) and prove their soundness.

Throughout the whole section, we assume that a signature $\langle C, I \rangle$ is fixed. Furthermore, the universe $\mathcal{U}$, the set of variables $\mathcal{V}$, and the set of predicate symbols $\mathcal{P}$ are countably infinite sets.

In previous chapters, we have looked at *concrete semantics* for the state transitions on BIP configurations. An interaction $I_j(x_1, \ldots, x_{\alpha(j)})$ is called *closed* if the matching components for each variable x_i, $1 \leq i \leq \alpha(j)$, are also specified (and no component is missing). In concrete semantics, an interaction can only fire if it is closed and some other conditions hold. Contrarily, in the *open semantics*, an interaction can also fire if it is not *closed* (hence if it is *open*). In both semantics, an interaction can only fire if it is *enabled*, hence if all the connected components are in a state such that there exists a transition (specified by the behavior of the component via the component type) from the current state labeled by the port where the interaction is connected. The only difference between the upcoming definition of *open semantics* and the previous definition for *closed semantics* (see Definition 12) is the specification whether the interaction must be *closed* or not. The other conditions are equal.

Definition 36 (Open Semantics)**.** *Let* $\Sigma = \{I_j(x_1, \ldots, x_{\alpha(j)}) \mid I_j \in I,\ x_1, \ldots, x_{\alpha(j)} \in \mathcal{V}\}$ *be the set of all interaction atoms. Then we define a transition function* $\rightsquigarrow_o \colon \Sigma_{\langle C,I \rangle} \times \Sigma \to \Sigma_{\langle C,I \rangle}$*, where it is*

$$(\mathfrak{S}, \varsigma, \nu) \overset{I_j(x_1,\ldots,x_{\alpha(j)})}{\rightsquigarrow_o} (\mathfrak{S}, \varsigma', \nu), \text{ if and only if}$$

1. $a := (\nu(x_1), \ldots, \nu(x_{\alpha(j)})) \in I_j$ *is an interaction,*

2. *a is enabled in* $(\mathfrak{S}, \varsigma, \nu)$*,*

3. $\varsigma' = \varsigma[(u_k, C_k) \leftarrow s'_k \mid u_k \in a, u_k \in C_k^{\mathfrak{S}}, I_j^k \in \mathbb{P}_k]$ *is the state snapshot, where* s'_k *is defined in Definition 11.*

We call such a transition $(\mathfrak{S}, \varsigma, \nu) \overset{I_j(x_1,\ldots,x_{\alpha(j)})}{\rightsquigarrow_o} (\mathfrak{S}, \varsigma', \nu)$ *a* havoc step.

The notation is extended to

$$(\mathfrak{S}, \varsigma, \nu) \overset{w}{\rightsquigarrow_o} (\mathfrak{S}, \varsigma', \nu)$$

for a word $w \in \Sigma^*$ if $w = w_1 \cdot \ldots \cdot w_n$ for some $n \in \mathbb{N}_0$, $w_1, \ldots, w_n \in \Sigma$, and $(\mathfrak{S}, \varsigma, \nu) \overset{w_1}{\rightsquigarrow_o} (\mathfrak{S}, \varsigma 1, \nu) \overset{w_2}{\rightsquigarrow_o} \ldots \overset{w_n}{\rightsquigarrow_o} (\mathfrak{S}, \varsigma n, \nu) = (\mathfrak{S}, \varsigma', \nu)$. If an interaction $\mathcal{A} \in \Sigma$ does not fulfill the conditions specified in the previous definition for a BIP configuration $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I \rangle}$, then it cannot be executed, since no transition exists. Note that neither the BIP system $\mathfrak{S}$ nor the variable mapping ν of a BIP configuration $(\mathfrak{S}, \varsigma, \nu)$ is touched. The *havoc steps* only change the state snapshot of a BIP configuration.

We use a different semantic for the *havoc rules* than the concrete semantics (which represents the "real" behavior in BIP frameworks), because local reasoning is simpler with open semantics. The key point is the composition rule ($\bowtie$) that is defined at the end of this section. Figure 5.1 illustrates this difference using the dining philosophers problem.

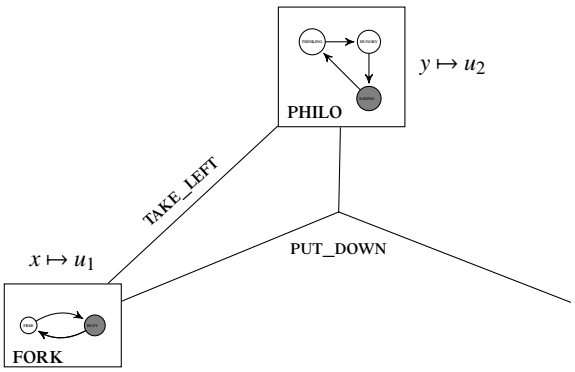

Figure 5.1.: A BIP configuration where the interaction PUT_DOWN(y, z, x) is not complete. It cannot fire in *closed semantics*, but could fire if we consider *open semantics*.

We specify the sets of possible words by using *language expressions*. Those are expressions that resemble regular expressions and represent a language (hence a set of words). If not stated otherwise, the symbol Σ represents an arbitrary set that works as an alphabet for a language.

Definition 37 (Language Expression)**.** *A* language expression L *over the alphabet* Σ *is a term generated inductively by the following syntax:*

$$L ::= \epsilon \mid \mathcal{A} \mid L_1 \cdot L_2 \mid L_1 \cup L_2 \mid L_1^* \mid L_1 \bowtie_{\Sigma_1,\Sigma_2} L_2,$$

where ϵ *denotes the empty word,* $\mathcal{A} \in \Sigma$*,* L_1 *and* L_2 *are language expressions and* $\Sigma_1, \Sigma_2 \subseteq \Sigma$*.*

We specify the *support alphabet* of a language expression, which contains all the symbols that appear in words of the corresponding language.

Definition 38 (Support Alphabet)**.** *The* support alphabet *of a language expression L is the set* $\mathrm{supp}(L)$*, defined inductively as:*

$$\mathrm{supp}(\epsilon) = \emptyset \quad \mathrm{supp}(\mathcal{A}) = \{\mathcal{A}\} \quad \mathrm{supp}(L^*) = \mathrm{supp}(L)$$
$$\mathrm{supp}(L_1 \cdot L_2) = \mathrm{supp}(L_1 \cup L_2) = \mathrm{supp}(L_1 \bowtie_{\Sigma_1,\Sigma_2} L_2) = \mathrm{supp}(L_1) \cup \mathrm{supp}(L_2).$$

Language expressions represent sets of words that we call languages. The connection between a language expression and the language is given in the next definition. Most valuations should not be unexpected, only the valuation of $\bowtie_{\Sigma_1,\Sigma_2}$ might deserve a closer look, since it defines an interleaving of languages.

Definition 39 (Valuation of Language Expressions)**.** *For a language expression L we define its valuation $[\![L]\!]$ inductively as:*

$$[\![\epsilon]\!] = \{\epsilon\} \quad [\![\mathcal{A}]\!] = \{\mathcal{A}\} \quad [\![L_1 \cdot L_2]\!] = \{w_1 \cdot w_2 \mid w_i \in [\![L_i]\!] \text{ for } i \in \{1,2\}\}$$
$$[\![L_1 \cup L_2]\!] = [\![L_1]\!] \cup [\![L_2]\!] \quad [\![L_1^*]\!] = \bigcup_{n \in \mathbb{N}_0} [\![L_1^n]\!]$$
$$[\![L_1 \bowtie_{\Sigma_1,\Sigma_2} L_2]\!] = \{w \in (\Sigma_1 \cup \Sigma_2)^* \mid w{\downarrow_{\Sigma_i}} \in L_i \text{ for } i \in \{1,2\}\},$$

where ϵ denotes the empty word, $\mathcal{A} \in \Sigma$, L_1 and L_2 are language expressions and $\Sigma_1, \Sigma_2 \subseteq \Sigma$. Furthermore, L^n is the concatenation of the language expression L via $(\cdot)$ n times and for every word $w \in \Sigma^*$ and any subset of the alphabet $\Sigma' \subseteq \Sigma$, we denote by $w{\downarrow_{\Sigma'}}$ the projection of the word to the symbols in the alphabet, hence the word obtained by removing all symbols in $\Sigma \setminus \Sigma'$.

To reason about the state transitions via proof rules, we define again a restricted family of triples that resemble Hoare triples. Here, we call them *havoc triples*, the pre- and post-conditions are once more specified via $\langle C, I\rangle$-formulae but, in contrast to the Hoare triples that we used for the reconfiguration rules, this time the middle entry describes possible state changes via language expressions.

Definition 40 (Havoc Triples)**.** *Let P and Q be finite disjunctions of symbolic configurations over a signature $\langle C, I\rangle$ and L be a language expression over an alphabet Σ for some injective substitution θ. The triple $\{\ P\ \}\ L\ \{\ Q\ \}$ stands for the statement:*

for all BIP configurations $(\mathfrak{S}, \varsigma, \nu), (\mathfrak{S}, \varsigma', \nu) \in \Sigma_{\langle C,I\rangle}$,
if $(\mathfrak{S}, \varsigma, \nu) \models P$ and $(\mathfrak{S}, \varsigma, \nu) \overset{w}{\rightsquigarrow}_o (\mathfrak{S}, \varsigma', \nu)$ for some $w \in [\![L]\!]$, then $(\mathfrak{S}, \varsigma', \nu) \models Q$.

In the rest of this chapter, we give proof rules that use *havoc triples* and then prove their soundness. The rules are divided into four groups, where the first group makes use of the structure of the language expression to reason about the triple, the second group analyzes the language (hence the set of words), and the third group reasons via analyzing the pre- and postconditions of a triple. Lastly, we define a composition rule ($\bowtie$) that transfers the concept of locality to the reasoning about firing interactions on BIP configurations.

We start by giving the rules of the first group.

Definition 41 (Proof Rules on the Structure of the Language Expression)**.** *The following proof rules use the structure of the language expression L to analyze the havoc triples:*

$$\frac{\begin{array}{c}\text{for each } x_k \in B, \text{ there exist } s_k, s_k' \in \mathbb{S}_k \text{ such that } s_k \overset{I_j^k}{\rightsquigarrow}_k s_k' \text{ is a transition for } C_k, \\ \text{where } B := \{x_{i_1}, \ldots, x_{i_n}\} \subseteq \{x_1, \ldots, x_{\alpha(j)}\} \text{ as given in Definition 36}\end{array}}{\{\ \bigwedge_{x_k \in B} C_k(x_k) \wedge \text{state}(x_k, s_k)\ \}\ I_j(x_1, \ldots, x_{\alpha(j)})\ \{\ \bigwedge_{x_k \in B} C_k(x_k) \wedge \text{state}(x_k, s_k')\ \}}\ (\mathcal{A})$$

$$\frac{}{\{\ P\ \}\ \epsilon\ \{\ P\ \}}\ (\epsilon) \qquad \frac{\{\ P\ \}\ L_1\ \{\ Q\ \} \qquad \{\ Q\ \}\ L_2\ \{\ R\ \}}{\{\ P\ \}\ L_1 \cdot L_2\ \{\ R\ \}}\ (\cdot)$$

$$\frac{\{\ P\ \}\ L_1\ \{\ Q\ \} \qquad \{\ P\ \}\ L_2\ \{\ Q\ \}}{\{\ P\ \}\ L_1 \cup L_2\ \{\ R\ \}}\ (\cup) \qquad \frac{\{\ P\ \}\ L\ \{\ P\ \}}{\{\ P\ \}\ L^*\ \{\ P\ \}}\ (^*)$$

Here, the formulae P, Q, R are symbolic configurations for the signature $\langle C, I\rangle$ and L, L_1, L_2 are language expressions over an alphabet Σ.

Note that the composition of two language expressions via the $\bowtie$-operator is missing. This is due to the fact that this operator is used in the composition rule ($\bowtie$), which is more complex (see Definition 47 for the corresponding proof rule).

Lemma 5 (Soundness of the Structural Proof Rules)**.** *The structural proof rules given in Definition 41 are sound.*

Proof. We prove the soundness of rules given in Definition 41 and therefore assume that $(\mathfrak{S}, \varsigma, \nu), (\mathfrak{S}, \varsigma', \nu), (\mathfrak{S}, \varsigma_1, \nu), (\mathfrak{S}, \varsigma_2, \nu) \in \Sigma_{\langle C,I\rangle}$ are BIP configurations.

- Suppose that $(\mathfrak{S}, \varsigma, \nu) \models \bigwedge_{x_k \in B} C_k(x_k) \wedge \text{state}(x_k, s_k)$, where the set B is defined in the rule ($\mathcal{A}$). Furthermore, we assume that

  $$(\mathfrak{S}, \varsigma, \nu) \overset{w}{\rightsquigarrow}_o (\mathfrak{S}, \varsigma', \nu)$$

  for a word $w \in [\![I_j(x_1, \ldots, x_{\alpha(j)})]\!] = \{I_j(x_1, \ldots, x_{\alpha(j)})\}$ in the language.

  Since there is only one possible word, it follows via the definition of the transition function $\rightsquigarrow_o$ that $(\mathfrak{S}, \varsigma', \nu) \models \bigwedge_{x_k \in B} C_k(x_k) \wedge \text{state}(x_k, s_k')$ (see Definition 36). Hence the rule ($\mathcal{A}$) is sound.

- Suppose now that $(\mathfrak{S}, \varsigma, \nu) \models P$ and $w \in [\![\epsilon]\!]$. Then

  $$(\mathfrak{S}, \varsigma, \nu) \overset{\epsilon}{\rightsquigarrow}_o (\mathfrak{S}, \varsigma, \nu),$$

  the empty word does not change any states and $(\mathfrak{S}, \varsigma, \nu) \models P$, hence the rule ($\epsilon$) is sound.

- Let $(\mathfrak{S}, \varsigma, \nu) \models P$ and the premises $\{\ P\ \}\ L_1\ \{\ Q\ \}$ and $\{\ Q\ \}\ L_2\ \{\ R\ \}$ hold. Furthermore let $w \in [\![L_1 \cdot L_2]\!]$ be a word. Then w is a concatenation $w = w_1 \cdot w_2$, where $w_i \in [\![L_i]\!]$,

$i = 1, 2$. If

$$(\mathfrak{S}, \varsigma, \nu) \overset{w_1}{\rightsquigarrow}_o (\mathfrak{S}, \varsigma_1, \nu) \overset{w_2}{\rightsquigarrow}_o (\mathfrak{S}, \varsigma_2, \nu) \quad \text{then also} \quad (\mathfrak{S}, \varsigma, \nu) \overset{w}{\rightsquigarrow}_o (\mathfrak{S}, \varsigma_2, \nu)$$

and it follows via the premises that $(\mathfrak{S}, \varsigma_1, \nu) \models Q$ and $(\mathfrak{S}, \varsigma_2, \nu) \models R$. Hence the rule $(\cdot)$ is sound.

For the proof of the rule $(\cup)$, we have a word $w \in [\![L_1 \cup L_2]\!]$, hence $w \in [\![L_i]\!]$, for $i = 1$ or $i = 2$. Then the soundness of the rule follows via the premises. And for the proof of the rule $(^*)$, we analyze a word $w \in [\![L^*]\!]$. There exists an element $n \in \mathbb{N}_0$ such that $w \in [\![L^n]\!]$ and hence we can apply the rule $(\cdot)$ n times to prove the soundness of the rule $(^*)$. □

A *disabled* interaction is an interaction atom such that one of the corresponding components is in a state where there exists no transition from the state labeled by the port where the interaction is connected. Whether or not an interaction is disabled depends on the respective BIP configuration.

Definition 42 (Disabled Interaction †)**.** *Given a symbolic configuration P and an interaction atom $I_j(x_1, \ldots, x_{\alpha(j)})$, we write $\{ P \} \dagger I_j(x_1, \ldots, x_{\alpha(j)})$ for the statement:*

> *Let $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I \rangle}$ be any BIP configuration with $(\mathfrak{S}, \varsigma, \nu) \models P$. Then there exists an element $x_k \in \{x_1, \ldots, x_{\alpha(j)}\}$ such that* state(x_k, s_k) *and there exists no transition $s_k \overset{I_j^k}{\rightsquigarrow}_k s_k'$, $s_k, s_k' \in \mathbb{S}_k$.*

For a set $\Sigma' \in \Sigma$ of interaction atoms, $\{ P \} \dagger \Sigma'$ stands for $\{ P \} \dagger \mathcal{A}$ for each $\mathcal{A} \in \Sigma'$.

In Figure 5.1 the interaction TAKE_LEFT(y, x) is disabled because there exists no transition from the state EATING labeled by TAKE_LEFT^1 for the philosopher x. Now we specify proof rules that analyze the sets of words.

Definition 43 (Proof Rules on the Language Set)**.** *The following proof rules use subset relations for the language $[\![L]\!]$ to analyze the havoc triples:*

$$\frac{\{ P \}\ L_1\ \{ Q \} \qquad [\![L_2]\!] \subseteq [\![L_1]\!]}{\{ P \}\ L_2\ \{ Q \}}\ (\subseteq)$$

$$\frac{\{ P \} \dagger \mathcal{A}}{\{ P \}\ \mathcal{A}\ \{ \texttt{false} \}}\ (\dagger_\perp) \qquad \frac{\{ P \}\ L\ \{ Q \} \qquad \{ Q \} \dagger \mathcal{A} \qquad [\![L]\!] \text{ is prefix-closed}}{\{ P \}\ L \bowtie_{\text{supp}(L),\mathcal{A}} \mathcal{A}^*\ \{ Q \}}\ (\dagger_\bowtie)$$

$$\frac{\{ P \}\ L\ \{ Q \} \qquad \mathcal{A} \notin \text{supp}(L)}{\{ P * \mathcal{A} \}\ L\ \{ Q * \mathcal{A} \}.}\ (\text{supp})$$

The formulae P and Q are symbolic configurations for a signature $\langle C, I \rangle$, $\mathcal{A} \in \Sigma$ is an interaction atom and L, L_1, L_2 are language expressions over the alphabet Σ.

The rule ($\subseteq$) states that if a havoc triple is valid for a language expression L_1 then it is also valid for every language expression whose language is a subset of the language of L_1. The postcondition of a havoc triple whose language contains disabled interactions in each word is `false` (see rule ($\dagger_\perp$)). Furthermore, rule ($\dagger_\bowtie$) states that if we have a triple $\{ P \}\ L\ \{ Q \}$ then we can interweave the words in $[\![L]\!]$ with any words consisting of disabled interactions and the pre- and postcondition remain the same. Last but not least, the pre- and postconditions of a havoc triple can be extended by any interaction atoms that are not in the support of the language expression. We proceed by showing the soundness of the rules.

Lemma 6 (Soundness of Proof Rules on the Language Set)**.** *The proof rules given in Definition 43 are sound.*

Proof. Let $(\mathfrak{S}, \varsigma, \nu), (\mathfrak{S}, \varsigma', \nu) \in \Sigma_{\langle C,I \rangle}$ be BIP configurations.

- Let $(\mathfrak{S}, \varsigma, \nu) \models P$ and we assume that the premises hold. Furthermore $w \in [\![L_2]\!] \subseteq [\![L_1]\!]$ is a word such that
$$(\mathfrak{S}, \varsigma, \nu) \overset{w}{\rightsquigarrow}_o (\mathfrak{S}, \varsigma', \nu).$$
Since $w \in [\![L_1]\!]$, it follows that $(\mathfrak{S}, \varsigma', \nu) \models Q$ and hence via the premise that the rule ($\subseteq$) is sound.

- Let $(\mathfrak{S}, \varsigma, \nu) \models P$ and we suppose that $\{ P \} \dagger \mathcal{A}$ for an interaction atom $I_j(x_1, \ldots, x_{\alpha(j)}) = \mathcal{A} \in \Sigma$. Definition 42 states that the interaction cannot fire, hence one or more of the conditions given in Definition 36 is not fulfilled and there exists no transition from $(\mathfrak{S}, \varsigma, \nu)$ via $\mathcal{A}$. Hence the rule ($\dagger_\perp$) is sound.

- Let $(\mathfrak{S}, \varsigma, \nu) \models P$ and we assume that the premises hold. Furthermore, $w \in [\![L \bowtie_{\mathrm{supp}(L),\mathcal{A}} \mathcal{A}^*]\!]$ is a word, where $w{\downarrow_{\mathrm{supp}(L)}} \in [\![L]\!]$ and $w{\downarrow_{\mathcal{A}}} \in [\![\mathcal{A}^*]\!]$.
  1. Assume that $w{\downarrow_{\mathcal{A}}} = \epsilon$. Then $(\mathfrak{S}, \varsigma, \nu) \overset{w}{\rightsquigarrow}_o (\mathfrak{S}, \varsigma', \nu)$ and it follows via the premise that $(\mathfrak{S}, \varsigma', \nu) \models Q$.
  2. Now assume that $w = u \cdot \mathcal{A} \cdot v$ for $u \in \mathrm{supp}(L)^*$ and $v \in \Sigma^*$. It is $w{\downarrow_{\mathrm{supp}(L)}} = u \cdot v{\downarrow_{\mathrm{supp}(L)}}$ and since $[\![L]\!]$ is prefix-closed, we obtain $u \in [\![L]\!]$. Then $(\mathfrak{S}, \varsigma, \nu) \overset{u}{\rightsquigarrow}_o (\mathfrak{S}, \varsigma', \nu)$ holds, where $(\mathfrak{S}, \varsigma', \nu) \models Q$ because of the premise. But the interaction atom $\mathcal{A}$ is disabled for the BIP configuration $(\mathfrak{S}, \varsigma', \nu)$ and hence the execution of $\mathcal{A}$ is not possible.

  If the execution of a word is possible, then the resulting BIP configuration $(\mathfrak{S}, \varsigma', \nu)$ models Q, and hence the rule ($\dagger_\bowtie$) is sound.

- Let $(\mathfrak{S}_P, \varsigma, \nu) \bullet (\mathfrak{S}_\mathcal{A}, \varsigma, \nu) \models P * \mathcal{A}$ for an interaction atom $\mathcal{A} \in \Sigma$, where $(\mathfrak{S}_P, \varsigma, \nu)$, $(\mathfrak{S}_\mathcal{A}, \varsigma, \nu) \in \Sigma_{\langle C,I \rangle}$, and $\mathcal{A} \notin \mathrm{supp}(L)$. We suppose that the premise $\{ P \}\ L\ \{ Q \}$ holds. Let $w \in [\![L]\!]$ be a word such that there exists a transition
$$(\mathfrak{S}_P, \varsigma, \nu) \bullet (\mathfrak{S}_\mathcal{A}, \varsigma, \nu) \overset{w}{\rightsquigarrow}_o (\mathfrak{S}_P, \varsigma', \nu) \bullet (\mathfrak{S}_\mathcal{A}, \varsigma', \nu).$$
Then there exists a transition $(\mathfrak{S}_P, \varsigma, \nu) \overset{w}{\rightsquigarrow}_o (\mathfrak{S}_P, \varsigma', \nu)$, because $\mathcal{A} \notin [\![L]\!]$ and hence the interaction $\mathcal{A}$ is not triggered by the word w. Thus $(\mathfrak{S}_P, \varsigma', \nu) \models Q$ and $(\mathfrak{S}_P, \varsigma', \nu) \bullet (\mathfrak{S}_\mathcal{A}, \varsigma', \nu) \models Q * \mathcal{A}$ and the proof rule is sound.

Hence the four proof rules are sound. □

Similar to the reconfiguration rules, there exist a rule of consequence (c) and a rule for disjunction ($\vee$). Both rules analyze the pre- and postcondition to reason about the validity of a havoc triple. We also derive rules for the unfolding (and folding) of predicate symbols, which can be seen as special cases of the rule of consequence and the rule for disjunction.

Definition 44 (Proof Rules for Predicates, Disjunction, and Consequence)**.** *The rule of consequence and the rule for disjunction are:*

$$\frac{P \models P' \qquad \{\, P' \,\}\ L\ \{\, Q' \,\} \qquad Q' \models Q}{\{\, P \,\}\ L\ \{\, Q \,\}}\ (c)$$

$$\frac{\left\{\{\, P_i \,\}\ L\ \{\, Q_i \,\}\right\}_{i=1}^{n}}{\{\, \bigvee_{i=1}^{n} P_i \,\}\ L\ \{\, \bigvee_{i=1}^{n} Q_i \,\}.}\ (\vee)$$

The following proof rules unfold resp. fold predicate symbols:

$$\frac{\{\, P * \phi \,\}\ L\ \{\, Q \,\}\ \textit{for all}\ A(t_1,\ldots,t_{\alpha(A)}) \leftarrow_{\mathcal{R}} \phi}{\{\, P * A(t_1,\ldots,t_{\alpha(A)}) \,\}\ L\ \{\, Q \,\}}\ (\text{lu})$$

$$\frac{\{\, P \,\}\ L\ \{\, Q * \phi \,\} \qquad A(t_1,\ldots,t_{\alpha(A)}) \leftarrow_{\mathcal{R}} \phi}{\{\, P \,\}\ L\ \{\, Q * A(t_1,\ldots,t_{\alpha(A)}) \,\}}\ (\text{ru})$$

$$\frac{\{\, P * A(t_1,\ldots,t_{\alpha(A)}) \,\}\ L\ \{\, Q \,\} \qquad A(t_1,\ldots,t_{\alpha(A)}) \leftarrow_{\mathcal{R}} \phi}{\{\, P * \phi \,\}\ L\ \{\, Q \,\}}\ (\text{lf})$$

$$\frac{\{\, P \,\}\ L\ \{\, Q * A(t_1,\ldots,t_{\alpha(A)}) \,\}\ \textit{for all}\ A(t_1,\ldots,t_{\alpha(A)}) \leftarrow_{\mathcal{R}} \phi_i,\ 1 \le i \le k}{\{\, P \,\}\ L\ \{\, \bigvee_{i=1}^{k} Q * \phi_i \,\}.}\ (\text{rf})$$

Here, P, P_i, Q, Q_i, ϕ are $\langle C, I\rangle$-formulae, $1 \le i \le k$, L is a language expression, $A \in \mathcal{P}$ is a predicate symbol, $t_1,\ldots,t_{\alpha(A)} \in \mathcal{V} \cup \mathcal{C}$ are terms.

Note that we often suppress the premise $A(t_1,\ldots,t_{\alpha(A)}) \leftarrow_{\mathcal{R}} \phi$ in the examples, since this follows directly from the rules $\mathcal{R}$ and can be easily reconstructed. We show the soundness of the rules.

Lemma 7 (Proof Rules for Predicates, Disjunction, and Consequence)**.** *The proof rules given in Definition 44 are sound.*

Proof. Let $(\mathfrak{S}, \varsigma, \nu), (\mathfrak{S}, \varsigma', \nu) \in \Sigma_{\langle C,I\rangle}$ be BIP configurations. We prove the soundness of the rules (c) and ($\vee$) first.

- Suppose that $(\mathfrak{S},\varsigma,\nu) \models P$ and the premises in the (c) rule hold. Furthermore $w \in [\![L]\!]$ is a word. Then $(\mathfrak{S},\varsigma,\nu) \models P'$,

$$(\mathfrak{S},\varsigma,\nu) \overset{w}{\rightsquigarrow}_o (\mathfrak{S},\varsigma',\nu)$$

and $(\mathfrak{S},\varsigma',\nu) \models Q'$ (this follows via the premise). Furthermore $(\mathfrak{S},\varsigma',\nu) \models Q$ and hence the rule (c) is sound.

- Suppose that $(\mathfrak{S},\varsigma,\nu) \models \bigvee_{i=1}^{n} P_i$ and $w \in [\![L]\!]$ is a word. We assume that the premises in the $(\vee)$ rule hold. Then $(\mathfrak{S},\varsigma,\nu) \models P_i$, $(\mathfrak{S},\varsigma,\nu) \overset{w}{\rightsquigarrow}_o (\mathfrak{S},\varsigma',\nu)$ and $(\mathfrak{S},\varsigma',\nu) \models Q_i$ for some $1 \leq i \leq n$. Hence $(\mathfrak{S},\varsigma',\nu) \models \bigvee_{i=1}^{n} Q_i$ and the rule $(\vee)$ is sound.

We show the soundness of the rule (lu) via a prooftree using the rules (c) and $(\vee)$:

$$\dfrac{P * A(t_1,\ldots,t_{\alpha(A)}) \models \bigvee_{A(t_1,\ldots,t_{\alpha(A)}) \leftarrow_{\mathcal{R}} \phi} \phi \qquad \dfrac{\{\ P * \phi\ \}\ L\ \{\ Q\ \} \text{ for all } A(t_1,\ldots,t_{\alpha(A)}) \leftarrow_{\mathcal{R}} \phi}{\{\ \bigvee_{A(t_1,\ldots,t_{\alpha(A)}) \leftarrow_{\mathcal{R}} \phi} P * \phi\ \}\ L\ \{\ Q\ \}}\ (\vee)}{\{\ P * A(t_1,\ldots,t_{\alpha(A)})\ \}\ L\ \{\ Q\ \},}\ (c)$$

which is true because $P * A(t_1,\ldots,t_{\alpha(A)}) \models \bigvee_{A(t_1,\ldots,t_{\alpha(A)}) \leftarrow_{\mathcal{R}} \phi} \phi$ holds.

The soundness of the rule (ru) follows via the rule of consequence (c), because $Q * \phi \models Q * A(t_1,\ldots,t_{\alpha(A)})$ if $A(t_1,\ldots,t_{\alpha(A)}) \leftarrow_{\mathcal{R}} \phi$. The same holds for the rule (lf) (but here we use the implication $P * \phi \models P * A(t_1,\ldots,t_{\alpha(A)})$). And last but not least let $\phi_1,\ldots,\phi_k$ be formulae such that $A(t_1,\ldots,t_{\alpha(A)}) \leftarrow_{\mathcal{R}} \phi_i$ holds for every $1 \leq i \leq k$. Then we use the rule (c) and the rule $(\vee)$ to obtain the soundness of the rule (rf):

$$\dfrac{\dfrac{\{\ P\ \}\ L\ \{\ Q * A(t_1,\ldots,t_{\alpha(A)})\ \} \qquad Q * A(t_1,\ldots,t_{\alpha(A)}) \models Q * \phi_i \text{ for all } 1 \leq i \leq k}{\{\ P\ \}\ L\ \{\ Q * \phi_i\ \} \text{ for all } 1 \leq i \leq k}\ (c)}{\{\ P\ \}\ L\ \{\ \bigvee_{i=1}^{k} Q * \phi_i\ \}.}\ (\vee)$$

Hence all the rules are sound. □

5.4. Frontier and Composition Rule

The rest of this chapter contains the composition rule $(\bowtie)$ and preparation for this rule. The idea is that if one sequence of interactions is triggered on one part of a distributed system and another sequence is triggered on the other part, then the sequences can be interleaved to a new sequence of interactions that are triggered on the whole system (if certain conditions hold).

An exemplary application of such a rule would be the chain that is illustrated in Figure 3.2, where it makes sense to reason about the $\text{seat}^{\text{R}}(f_0, f_1)$ part and the $\text{chain}(f_1, f_n)$ part separately. Let us look at a smaller example to analyze the problem a bit further. Suppose that we have two BIP configurations; the one given in Figure 5.1, which we call $(\mathfrak{S}_0,\varsigma_0,\nu_0)$, and the completion $(\mathfrak{S}_1,\varsigma_1,\nu_1)$, given in Figure 5.2. If we want to reason about the two BIP configurations separately, then the reasoning about the fork z would be quite limited, since there is no interaction that can fire. On the composed BIP configuration $(\mathfrak{S}_0,\varsigma_0,\nu_0) \bullet (\mathfrak{S}_1,\varsigma_1,\nu_1)$

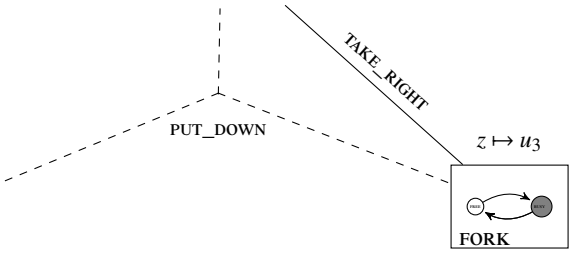

Figure 5.2.: A BIP configuration that indicates the *frontier* to the BIP configuration in Figure 5.1 by a dashed line. The frontier contains exactly one interaction PUT_DOWN(y, z, x).

there are many possible sequences — there are even sequences that contain the interaction TAKE_RIGHT(y, z) multiple times, but if we look at $(\mathfrak{S}_1, \varsigma_1, \nu_1)$ individually then we cannot derive that. Hence we add all the interactions in $(\mathfrak{S}_0, \varsigma_0, \nu_0)$ that connect components in $(\mathfrak{S}_1, \varsigma_1, \nu_1)$ and call the added interactions a *frontier*. In this case, we add the interaction PUT_DOWN(y, x, z) and analyze the fork z and two interactions. The same happens on the other side (we add the interaction TAKE_RIGHT(y, z) as a frontier to the configuration $(\mathfrak{S}_0, \varsigma_0, \nu_0)$) and then we interweave the possible sequences to create new sequences.

The definition of the frontier is simple if we know that it is finite. But there are cases where a straightforward definition yields an infinite frontier (and hence an infinite formula, which we want to avoid). Hence we define an equivalence relation on the interactions in the frontier and consider only representatives of the equivalence classes.

We start with the definition of the equivalence relation.

Definition 45 (Equivalence Class of Interaction Atoms)**.** *Let $\mathcal{A} := I_j(x_1, \ldots, x_{\alpha(j)})$ and $\mathcal{B} := I_j(y_1, \ldots, y_{\alpha(j)})$ be two interaction atoms for $I_j \in I$. They are* equivalent modulo a set Γ of component atoms *iff for each $1 \leq i \leq \alpha(j)$ either*

1. *$x_i = y_i$ and $C_i(x_i) \in \Gamma$ for some component type $C_i \in C$ that has the corresponding port $I^i_j \in \mathbb{P}_i$, or*
2. *$\{C_i(x_i), C_i(y_i)\} \cap \Gamma = \emptyset$ for all component types $C_i \in C$ that do not have the corresponding port, hence where $I^i_j \notin \mathbb{P}_i$.*

We write $\mathcal{A} \simeq_\Gamma \mathcal{B}$ in this case.

Since $\simeq_\Gamma$ is a binary relation that is reflexive, symmetric, and transitive, it is indeed an equivalence relation. The set $\Sigma\langle C, I\rangle := \{I_j(x_1, \ldots, x_{\alpha(j)}) \mid I_j \in I,\ x_1, \ldots, x_{\alpha(j)} \in \mathcal{V}\}$ is infinite even though the set of interaction types is finite. But the partition $\Sigma\langle C, I\rangle / \simeq_\Gamma$ is finite, if Γ is finite. We write $[.]_\Gamma$ as the unique representative of each equivalence class.

Lemma 8. *If the set of component atoms Γ is finite, then the partition $\Sigma\langle C, I\rangle / \simeq_\Gamma$ is also finite.*

Proof. Suppose that $\Sigma\langle C, I\rangle / \simeq_\Gamma$ is infinite. Since the number of interaction types itself is finite, it follows that there exists an interaction type $I_j \in I$ such that there is an infinite

number of equivalence classes for interaction atoms of the form $I_j(x_1, \ldots, x_{\alpha(j)})$. We fix I_j as an interaction type until the end of the proof and look at the different equivalence classes.

Let $X := \{x_i \mid C_i(x_i) \in \Gamma\} \subseteq \mathcal{V}$ be the set of all variables that are used in Γ. The set X is finite since Γ is finite. We analyze the set of tuples $(X \cup \{\_\})^{\alpha(j)}$, where the finite set X is extended by another element _. This set is also finite and hence the set of tuples of arity $\alpha(j)$ is finite.

The tuples can be used to describe the equivalence classes in $\Sigma\langle C, I\rangle / \simeq_\Gamma$ in the following way: A tuple $(x_1, \ldots, x_{\alpha(j)})$ represents the set

$$\{I_j(y_1, \ldots, y_{\alpha(j)}) \mid y_i = x_i \text{ if } x_i \in X, \text{ and } y_i \in \mathcal{V} \setminus X \text{ if } x_i = \_, \text{ for } 1 \leq i \leq \alpha(x)\}.$$

It follows directly from the definition of equivalence modulo a set Γ (see Definition 45) that all the interaction atoms in the same set are equivalent. Furthermore, the sets are pairwise disjoint for distinct tuples $x_1, \ldots, x_{\alpha(j)}$. And since the set of tuples $(X \cup \{\_\})^{\alpha(j)}$ is finite, the set of equivalence classes for the interaction type I_j is finite. Hence we have contradicted the first assumption and the partition is finite. □

The equivalence relation is well-defined for our purposes since two equivalent interaction atoms have the same effect on the states of the components. This is proven in the next lemma.

Lemma 9. *Let P be a symbolic BIP configuration, $\Gamma = \Gamma_\theta(P)$ the set of component atoms, and $\mathcal{A} := I_j(x_1, \ldots, x_{\alpha(j)})$ and $\mathcal{B} := I_j(y_1, \ldots, y_{\alpha(j)})$ interaction atoms in $\Sigma_\theta(P)$. If $(\mathfrak{S}, \varsigma, \nu)$, $(\mathfrak{S}, \varsigma_0, \nu), (\mathfrak{S}, \varsigma_1, \nu) \in \Sigma_{\langle C,I\rangle}$ and $(\mathfrak{S}, \varsigma, \nu) \models P$, then*

$$(\mathfrak{S}, \varsigma, \nu) \overset{\mathcal{A}}{\rightsquigarrow}_o (\mathfrak{S}, \varsigma_0, \nu) \text{ and } (\mathfrak{S}, \varsigma, \nu) \overset{\mathcal{B}}{\rightsquigarrow}_o (\mathfrak{S}, \varsigma_1, \nu) \quad \textit{implies} \quad \varsigma_0(\nu(x), C_i) = \varsigma_1(\nu(x), C_i)$$

for each component atom $C_i(x) \in \Gamma$.

Before we prove the lemma, we need to justify why we can assume that a variable mapping ν is injective: Suppose that P is a symbolic configuration and $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$ a model of P. Then we can assume that the restriction $\nu{\downarrow_{\mathrm{fv}(P)}}$ of the variable mapping ν to the free variables in P is injective. Otherwise, we could choose an arbitrary total order on the set of variables and define a substitution $\theta : \mathrm{fv}(P) \to \mathcal{V}$, where $\theta(x) = \min\{y \in \mathrm{fv}(P) \mid \nu(y) = \nu(x)\}$. Then it is $(\mathfrak{S}, \varsigma, \nu) \models P$ if and only if $(\mathfrak{S}, \varsigma, \nu\theta) \models P\theta$.

Proof. Let $C_i(x) \in \Gamma$ be a component atom. Throughout the proof, we use the definition of the equivalence relation (Definition 45) frequently. There are two possible cases:

- Suppose that $\nu(x) = \nu(x_i)$ for some $1 \leq i \leq \alpha(A)$. Since ν is injective, it is $\nu(x) = \nu(x_i)$. It is $\mathcal{A} \simeq_\Gamma \mathcal{B}$, and hence there are again two cases:
  - It is $x_i = y_i$ and the port $I_j^i \in \mathbb{P}_i$ is specified by the component type C_i. Since there exist executions for the words $\mathcal{A}$ and $\mathcal{B}$ for the BIP configuration $(\mathfrak{S}, \varsigma, \nu)$ and the transition system described by C_i is deterministic, there exists exactly one transition $\nu(x_i, C_i) \overset{I_j^i}{\rightsquigarrow} s_i'$ for a state $s_i' \in \mathbb{S}_i$. Now we know that $\varsigma_{\mathcal{A}}(\nu(x_i), C_i) = \varsigma_{\mathcal{B}}(\nu(y_i), C_i) = s_i'$ for $s_i' \in \mathbb{S}_i$.

– The other case it not possible, since $C_i(x) = C_i(x_i)$, the sets $\{C_i(x_i), C_i(y_i)\}$ and Γ cannot be disjoint.

- Suppose now that $\nu(x) \notin \{\nu(x_1), \ldots, \nu(x_{\alpha(A)})\}$, then it follows that $x \notin \{x_1, \ldots, x_{\alpha(A)}\}$. Furthermore $x \notin \{y_1, \ldots, y_{\alpha(A)}\}$: If we assume that $x = y_k$ for some $1 \leq k \leq \alpha(A)$, then $C_i(y_k) = C_i(x)$ and hence the sets $\{C_i(x_k), C_i(y_k)\}$ and Γ are not disjoint. Thus $x_k = y_k$ and $\nu(x_k) = \nu(y_k)$, which is a contradiction to the first assumption. Hence it follws that $x \notin \{y_1, \ldots, y_{\alpha(A)}\}$, thus $\nu(x) \notin \{\nu(y_1), \ldots, \nu(y_{\alpha(A)})\}$ (since ν is injective) and furthermore $\varsigma(\nu(x), C_i) = \varsigma_{\mathcal{A}}(\nu(x), C_i) = \varsigma_{\mathcal{B}}(\nu(x), C_i)$.

Hence the statement follows. □

Now we finally define the *frontier* of two symbolic configurations formally. Remember that – intuitively – the frontier of the symbolic configuration P for the configuration Q contains all the interaction atoms in Q that connect components defined in P (or rather their equivalence classes, but we have shown in Lemma 9 that this makes no difference for our purposes).

Definition 46. *Given predicateless symbolic configurations P and Q and an injective substitution θ, we define $\mathcal{F}_\theta(P, Q) := \iota_1 * \cdots * \iota_m$, where*

$$\{\iota_1, \ldots, \iota_m\} := \{[I_j(x_1, \ldots, x_{\alpha(j)})]_{\Gamma_\theta(P)} \mid I_j(x_1, \ldots, x_{\alpha(j)}) \in \Sigma_\theta(Q), \\ \text{and } C_i(x_i) \in \Gamma_\theta(P) \wedge I_j^i \in \mathbb{P}_i \text{ for some } 1 \leq i \leq \alpha(j)\}.$$

Given finite disjunctions of symbolic configurations P_k, $k = 1, 2$, we define

$$\mathcal{F}_\theta(P_1, P_2) := \bigvee \{\mathcal{F}_\theta(p_1, p_2) \mid P_k \leftarrow^{pl}_{\mathcal{R}} \phi_k,\ p_i \text{ is a disjunct of } \phi_k,\ k = 1, 2\}.$$

Now we show that $\mathcal{F}_\theta(P_1, P_2)$ is indeed always a finite formula and use that $\leftarrow^{pl}_{\mathcal{R}}$ unfolds all the predicate symbols until the result is a predicateless symbolic configuration.

Lemma 10. *For any finite disjunctions of symbolic configurations P_k, $k = 1, 2$, and any injective substitution θ, the set $\{\mathcal{F}_\theta(p_1, p_2) \mid P_k \leftarrow^{pl}_{\mathcal{R}} \phi_k,\ p_i \text{ is a disjunct of } \phi_k,\ k = 1, 2\}$ is finite.*

Proof. Let $\mathcal{F}_\theta(p_1, p_2) = \iota_1 * \cdots * \iota_m$ be an element of the set, where p_1 is a disjunct of some formula ϕ_1 and p_2 a disjunct of some ϕ_2. For each $1 \leq i \leq m$, ι_i stands for an equivalence class $[I_j(x_1, \ldots, x_{\alpha(j)})]_{\Gamma_\theta(P_1)}$ for $I_j \in I$ and $x_1, \ldots, x_{\alpha(j)} \in \mathcal{V}$, where $I_j(x_1, \ldots, x_{\alpha(j)}) \in \Sigma_\theta(\phi_2) \subseteq \Sigma_\theta(P_2)$ is an interaction atom in P_2 and there exists an element $1 \leq k \leq \alpha(A)$ such that $C_k(x_k) \in \Gamma_\theta(P_1)$. The substitution θ is injective, hence there exists exactly one variable $y \in \mathrm{fv}(P_1) \cap \mathrm{fv}(P_2)$ such that $\theta(y) = x_k$. Since the set $\mathrm{fv}(P_1) \cup \mathrm{fv}(P_2)$ is finite (unfolding only adds existentially quantified variables), its intersection is also finite. Hence there are finitely many component atoms $C_k(x_k)$ such that there exists a corresponding interaction atom in $\Sigma_\theta(P_2)$. Since the number of interaction types is finite and the number of such component atoms is also finite, it follows that the number of possible equivalence classes of interaction atoms in the frontier is also finite. Hence the set is finite. □

The composition rule ($\bowtie$) aims to be an equivalent of the *frame rule* in the set of havoc rules. If two languages hold on two different BIP configurations and the frontiers, then their interleaved words should be valid for the composition of the configurations.

Definition 47 (Proof Rule for Composition)**.** *We define the proof rule*

$$\frac{\{\, P_i * \mathcal{F}_\theta(P_i, P_{3-i}) \,\}\ \tilde{L}_i\ \{\, Q_i * \mathcal{F}_\theta(P_i, P_{3-i}) \,\} \qquad i = 1,2}{\{\, P_1 * P_2 \,\}\ L_1 \bowtie_{\Sigma_1,\Sigma_2} L_2\ \{\, Q_1 * Q_2 \,\},} \ (\bowtie)$$

where

- θ *is an injective substitution,*
- $\Sigma_i = \{I_j(x_1, \ldots, x_{\alpha(j)}) \in \Sigma_\theta(P_1 * P_2) \mid C_k(x_k) \in \Gamma_\theta(P_i)$ *for some* $1 \leq k \leq \alpha(j)\}$*,*
- $\tilde{L}_i$ *is obtained from* L_i *by replacing each interaction atom* $\iota \in \Sigma_i$ *with its* $\simeq_{\Gamma_\theta(P_i)}$*-representative, namely* $[\iota]_{\Gamma_\theta(P_i)}$*, and*
- P_1, P_2, Q_1, Q_2 *are* $\langle C, I\rangle$*-formulae and* L_1, L_2 *are language expressions over the alphabet* Σ*.*

Last but not least, we prove the soundness of the composition rule.

Lemma 11. *The* ($\bowtie$) *rule is sound.*

Let $X \subseteq \mathcal{V} \times C$ be a set of variables, ν a variable mapping and ς, ς' two state snapshots. We say that ς' agrees with ς over $\nu(X)$ if $\varsigma'(\nu(x), C_i) = \varsigma(\nu(x), C_i)$ for all $(x, C_i) \in X$.

Proof. We prove the soundness of the rule by analyzing the structure of a word in the language $[\![L_1 \bowtie_{\Sigma_1,\Sigma_2} L_2]\!]$:

Let $(\mathfrak{S}, \varsigma, \nu) \in \Sigma_{\langle C,I\rangle}$ be a BIP configuration such that $(\mathfrak{S}, \varsigma, \nu) \models P_1 * P_2$. We assume that the premises hold and that $w \in [\![L_1 \bowtie_{\Sigma_1,\Sigma_2} L_2]\!]$ is a word such that

$$(\mathfrak{S}, \varsigma, \nu) \overset{w}{\rightsquigarrow}_o (\mathfrak{S}, \varsigma', \nu)$$

for a configuration $(\mathfrak{S}, \varsigma', \nu) \in \Sigma_{\langle C,I\rangle}$. Then we can write $w = \nu_1 u_1 \nu_2 u_2 \cdots \nu_n u_n \nu_{n+1}$, where $u_1 \cdots u_n = w{\downarrow_{\Sigma_1}} \in [\![L_1]\!]$ and $\nu_1 \cdots \nu_{n+1} \in (\Sigma_2 \setminus \Sigma_1)^*$. Then there exists a sequence of configurations $(\mathfrak{S}, \varsigma_0, \nu), \ldots, (\mathfrak{S}, \varsigma_n, \nu), (\mathfrak{S}, \tau_1, \nu), \ldots, (\mathfrak{S}, \tau_{n+1}, \nu) \in \Sigma_{\langle C,I\rangle}$ such that

$$(\mathfrak{S}, \varsigma_0, \nu) \overset{\nu_1}{\rightsquigarrow}_o (\mathfrak{S}, \tau_1, \nu) \overset{u_1}{\rightsquigarrow}_o (\mathfrak{S}, \varsigma_1, \nu) \overset{\nu_2}{\rightsquigarrow}_o (\mathfrak{S}, \tau_2, \nu) \overset{u_2}{\rightsquigarrow}_o \ldots \overset{u_n}{\rightsquigarrow}_o (\mathfrak{S}, \varsigma_n, \nu) \overset{\nu_{n+1}}{\rightsquigarrow}_o (\mathfrak{S}, \tau_{n+1}, \nu), \tag{5.1}$$

where $\varsigma = \varsigma_0$ and $\varsigma' = \tau n + 1$. Since $(\mathfrak{S}, \varsigma, \nu) \models P_1 * P_2$, there exist BIP structures $\mathfrak{S}_1$ and $\mathfrak{S}_2$ such that $\mathfrak{S}_1 \bullet \mathfrak{S}_2 = \mathfrak{S}$ and $(\mathfrak{S}_i, \varsigma, \nu) \models P_i$, $i = 1,2$. Hence $(\mathfrak{S}_i, \varsigma, \nu) \models \phi_i$ for some predicateless unfolding $P_i \leftarrow^{pl}_{\mathcal{R}} \phi_i$ for $i = 1,2$. Let θ be an injective substitution and consider the sets $X_i := \{(x, C) \mid C(x) \in \Gamma_\theta(\phi_i)\}$. Note that X_1 and X_2 are disjoint, since P_1 and P_2 may not imply the exact same components.

Let F_{12} be the structure such that $(F_{12}, \varsigma, \nu) \models \mathcal{F}_\theta(P_1, P_2)$ for any state snapshot ς. This is well-defined since the frontier only contains interaction atoms. Moreover, it is easily seen

that F_{12} is defined solely by $\mathcal{F}_\theta(P_1, P_2)$ and ν. Let ς_i^1 be the state snapshot that agrees with ς_i over $\nu(X_1)$ and with ς_0 over $\nu(\mathcal{V} \setminus X_1)$ for $1 \leq i \leq n$. Moreover, let $\tilde{u}_i$ be the sequence obtained from u_i by replacing every interaction atom $\iota \in \Sigma_1$ with its $\simeq_{\Gamma_\theta(\phi_1)}$-representative $[\iota]_{\Gamma_\theta(\phi_1)}$. By Lemma 9, we obtain the following sequence of havoc steps:

$$(\mathfrak{S}_1 \bullet F_{12}, \varsigma_0^1, \nu) \overset{\tilde{u}_1}{\rightsquigarrow}_o (\mathfrak{S}_1 \bullet F_{12}, \varsigma_1^1, \nu) \overset{\tilde{u}_2}{\rightsquigarrow}_o \ldots \overset{\tilde{u}_n}{\rightsquigarrow}_o (\mathfrak{S}_1 \bullet F_{12}, \varsigma_n^1, \nu),$$

where $(\mathfrak{S}_1, \varsigma_0^1, \nu) \models \phi_1$, $(F_{12}, \varsigma_0^1, \nu) \models \mathcal{F}_\theta(P_1, P_2)$ and $\tilde{u}_1 \tilde{u}_2 \ldots \tilde{u}_n \in \tilde{L}_1$. To see that the above is indeed a valid havoc sequence, it is enough to observe that v_i does not change the state of any component from $C(x)$ with $(x, C) \in X_1$), thus in Equation 5.1 the state snapshot τ_{i+1} agrees with ς_i over $\nu(X_1)$ for $1 \leq i \leq n$. By the premise, we obtain $(\mathfrak{S}_1 \bullet F_{12}, \varsigma_n^1, \nu) \models Q_1 * \mathcal{F}_\theta(P_1, P_2)$. Hence $(\mathfrak{S}_1, \varsigma_n^1, \nu) \models Q_1$. By a symmetric construction, we obtain $(\mathfrak{S}_2, \varsigma_n^2, \nu) \models Q_2$, where the sequence $\varsigma_1^2, \ldots, \varsigma_n^2$ is obtained in a similar way as $\varsigma_1^1, \ldots, \varsigma_n^1$. Since ς' agrees with ς_n^i over $\nu(X_i)$, for $i = 1, 2$, we get $(\mathfrak{S}_1 \bullet \mathfrak{S}_2, \varsigma', \nu) \models Q_1 * Q_2$, as required. □

6. Reconfigurations on Token Rings

The token ring is a simple example for the usage of the DR-BIP framework (see [BBBS18]). This chapter analyses if and how we can prove the (partial) correctness of some reconfiguration programs on the token ring.

We start by specifying the signature for components and interactions in the token ring. Then we define inductive predicates that describe valid token rings via chains of connected components. Afterwards, we state reconfiguration programs for the deletion and addition of a component in the token ring. Finally, we prove the partial correctness of the reconfiguration programs.

The definition of the *token ring* that we use throughout this chapter is similar to the definition of the dynamic token ring in [BBBS18]:

> "A *token ring* consists of two or more identical components interconnected using uni-directional communication links according to a ring topology. A number of *tokens* are circulating within the ring. A component is *busy* when it holds a token and *idle* otherwise."

A *busy* component can pass the token through the outgoing link to another component only if this other component is *idle*. Deviating from the definition in [BBBS18], we say that a component may be deleted from the token ring if the resulting ring contains at least one busy and one idle component. We call such a token ring a *valid token ring*, since it is deadlock-free[1]. Generally, the components that enter the token ring have no token.

6.1. BIP Configurations, Predicates, and Programs

We start by defining the types of components and interactions in the token ring.

Definition 48 (Signature of the Token Ring)**.** *We define a signature* $\langle C, I\rangle$ *with one component type* C *and one interaction type* I *with arity* 2.
The component type $C = (\mathbb{P}, \mathbb{S}, s^0, \rightsquigarrow)$ *is a tuple, where* $\mathbb{P} = \{I^1, I^2\}$, $\mathbb{S} = \{\text{H}, \text{T}\}$ *and the initial state is* $s^0 = \text{H}$. *Furthermore, the transitions are* $\rightsquigarrow = \{(\text{T}, I^1, \text{H}), (\text{H}, I^2, \text{T})\}$.

Here, the state H is short for *hole* and if a component is in state H, then we say that it is idle. The state T is short for *token* and represents that a component has a token; we say that the component is busy. A component can "receive" a token from another component, if the interaction at port I^2 fires, and "pass" a token, if the interaction at port I^1 fires.

[1] A valid token ring is deadlock-free since there exists at least one pair of a busy and an idle component that are connected such that the busy component may pass the token to the idle component. Hence interactions may fire and there is no deadlock. This can also be proven with the tool described in [BI20].

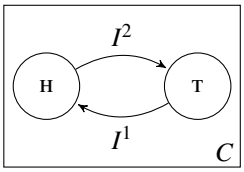

Figure 6.1.: The state diagram describes the behavior of instances of the component type C.

A *valid token ring* is a token ring with at least one idle and one busy component. We want to be able to check whether a BIP configuration models a *valid token ring*, hence we give a formal definition using (inductive) predicates. First, we define a chain of components $\text{chain}(x,z,h,t)$, where $h,t \in \mathbb{N}_0$ are natural numbers. We define the predicate inductively and use pattern matching, while intelligently substituting the pattern $h-1$ with the natural number $h-1 \in \mathbb{N}_0$ if possible:

$$\begin{aligned}
\text{chain}(x,z,1,1) &\leftarrow \exists y\,.\, C(x) * I(x,y) * C(y) * I(y,z) \wedge \text{state}(x,\text{H}) \wedge \text{state}(y,\text{T}),\\
\text{chain}(x,z,1,1) &\leftarrow \exists y\,.\, C(x) * I(x,y) * C(y) * I(y,z) \wedge \text{state}(x,\text{T}) \wedge \text{state}(y,\text{H}),\\
\text{chain}(x,z,h,t) &\leftarrow \exists y\,.\, C(x) * I(x,y) * \text{chain}(y,z,h-1,t) \wedge \text{state}(x,\text{H}),\\
\text{chain}(x,z,h,t) &\leftarrow \exists y\,.\, C(x) * I(x,y) * \text{chain}(y,z,h,t-1) \wedge \text{state}(x,\text{T}).
\end{aligned}$$

Note that there exists no matching for $\text{chain}(x,z,0,t)$ or $\text{chain}(x,z,h,0)$ for any $h,t \in \mathbb{N}_0$, since that ensures that each chain contains at least one busy and one idle component. Now the definition of the valid token ring is straightforward:

$$\text{token\_ring}(x) \leftarrow \exists h,t\,.\, \text{chain}(x,x,h,t). \tag{6.1}$$

For illustration, we define a symbolic BIP configuration, which is a valid token ring.

Example 14. Let B be a symbolic BIP configuration defined as

$$\begin{aligned}
B = &\exists a,b,c,d\,.\, C(a) * C(b) * C(c) * C(d)\\
&* I(a,b) * I(b,c) * I(c,d) * I(d,a)\\
&\wedge \text{state}(a,\text{T}) \wedge \text{state}(b,\text{H}) \wedge \text{state}(c,\text{T}) \wedge \text{state}(d,\text{T}).
\end{aligned}$$

This symbolic BIP configuration is depicted in Figure 6.2 and is a valid token ring since there exist at least one idle and one busy component. If the interaction $I(a,b)$ fires then the component a changes its state via the transition $\text{T} \overset{I^1}{\rightsquigarrow} \text{H}$ and the component b changes its state via the transition $\text{H} \overset{I^2}{\rightsquigarrow} \text{T}$. The interaction $I(c,d)$ connects two components in state T and the component d has no transition from state T with label I^2, hence the interaction cannot fire.

We propose reconfiguration programs for the deletion and addition of components in the token ring. For the addition of a component into the token ring we need to replace an interaction with two new interactions and the component. No hole or token can "get lost". To simplify the proof of the program, we assume that we insert the component next to a

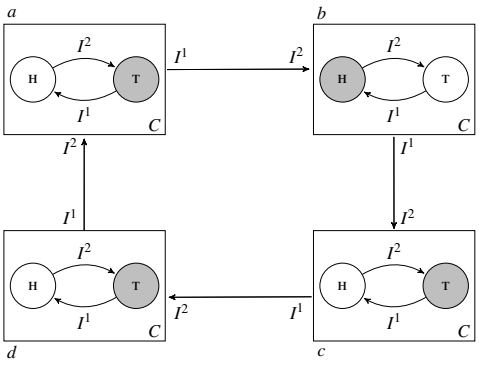

Figure 6.2.: A *valid token ring*, where three components are in state T and one component is in state H.

busy component.

Listing 6.1: New Component.

```
1 with C(x) ∗ I(x,z) ∧ state(x,T) do
2   disconnect(I,x,z);
3   new(C,y);
4   connect(I,x,y);
5   connect(I,y,z)
```

For the deletion of a busy component, it needs to be ensured that the remaining chain has at least one busy and one idle component. It is crucial to disconnect the interactions first and to keep the component from passing the token to the next component because then the remaining chain might not have any idle component.

Listing 6.2: Delete Component in State T.

```
1 with I(x,y) ∗ C(y) ∗ I(y,z) ∧ state(y,T) do
2   disconnect(I,y,z);
3   disconnect(I,x,y);
4   delete(C,y);
5   connect(I,x,z)
```

Example 15 (Order of the Deletion of Interactions). Assume that we want to delete the component a in Figure 6.2. We could delete the interaction between d and a first and this would be an atomic action. Afterwards, we would delete the interaction between a and b, but in between those two interactions, some interactions could be triggered via `havoc`, since we compose the two actions through sequential composition (see Definition 23). Only

the interaction between components a and b may fire, and if it does fire, then component a becomes idle and component b becomes busy. Hence there is no idle component unequal to component a and the deletion of component a would not result in a valid token ring. Indeed the new token ring would be in a deadlock since no interaction can fire (no token can be "passed"). Hence we need to make sure that the components a and b are disconnected first.

For the deletion of an idle component, the order of the disconnection also matters. This time, it is crucial to disconnect the incoming interaction first, because otherwise the remaining chain may contain no busy component.

Listing 6.3: Delete Component in State H.

```
1 with I(x,y) * C(y) * I(y,z) ∧ state(y,H) do
2   disconnect(I,x,y);
3   disconnect(I,y,z);
4   delete(C,y);
5   connect(I,x,z)
```

For the proofs of the reconfiguration programs, we define some more predicates that simplify the notation. We specify a ring token_ring$^{\text{H}}$ that contains at least two idle and one busy component and a ring token_ring$^{\text{T}}$ that contains at least one idle and two busy components:

$$\text{token\_ring}^{\text{H}}(x) \leftarrow \exists y,h,t\,.\; C(x) * I(x,y) * \text{chain}(y,x,h,t) \wedge \text{state}(x,\text{H}),$$
$$\text{token\_ring}^{\text{T}}(x) \leftarrow \exists y,h,t\,.\; C(x) * I(x,y) * \text{chain}(y,x,h,t) \wedge \text{state}(x,\text{T}).$$

Furthermore, we define a predicate chain* that implies more general chains:

$$\begin{aligned}
\text{chain}^*(x,x,0,0) &\leftarrow \texttt{emp},\\
\text{chain}^*(x,z,1,0) &\leftarrow C(x) * I(x,z) \wedge \text{state}(x,\text{H}),\\
\text{chain}^*(x,z,0,1) &\leftarrow C(x) * I(x,z) \wedge \text{state}(x,\text{T}),\\
\text{chain}^*(x,z,h,t) &\leftarrow \text{chain}(x,z,h,t).
\end{aligned}$$

6.2. Proofs of the Reconfiguration Programs

We start by proving the correctness of the reconfiguration programs for the addition of a new component. We prove that the execution of the reconfiguration on a deadlock-free token ring results again in a deadlock-free token ring.

Theorem 6. *Let $P_{\texttt{new}}$ be the program given in Listing 6.1. Then the reconfiguration program $P_{\texttt{new}}$ is correct, meaning that*

$$\{\,\text{token\_ring}(a)\,\}\; P_{\texttt{new}}\; \{\,\exists a.\,\text{token\_ring}(a)\,\}.$$

Proof. We apply the rule for sequential composition (given in Definition 30) first and obtain the following proof sketch:

$\{\ \text{token\_ring}(a)\ \}$
$\{\ \exists x, z.\ \ C(x) * I(x,z) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\mathtt{T})\ \}$
with $C(x) * I(x,z) \wedge \text{state}(x,\mathtt{T})$ **do**
  `disconnect(I,x,z)`
$\{\ C(x) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\mathtt{T})\ \}$
  `havoc`
$\{\ C(x) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\mathtt{T})\ \}$
  `new(C,y)`
$\{\ C(x) * C(y) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\mathtt{T}) \wedge \text{state}(y,\mathtt{H})\ \}$
  `havoc`
$\{\ C(x) * C(y) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\mathtt{T}) \wedge \text{state}(y,\mathtt{H})\ \}$
  `connect(I,y,z)`
$\{\ C(x) * C(y) * I(y,z) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\mathtt{T}) \wedge \text{state}(y,\mathtt{H})\ \}$
  `havoc`
$\{\ C(x) * C(y) * I(y,z) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\mathtt{T}) \wedge \text{state}(y,\mathtt{H})\ \}$
  `connect(I,x,y)`
$\{\ C(x) * I(x,y) * C(y) * I(y,z) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\mathtt{T}) \wedge \text{state}(y,\mathtt{H})\ \}$
$\{\ \exists x.\ \ \text{token\_ring}(x)\ \}$

It is important to note that the sequential composition implies that an implicit `havoc` should be considered in between two commands. We prove the small steps one by one:

- The implication $\text{token\_ring}(a) \models \exists x, z.\ C(x) * I(x,z) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\mathtt{T})$ is proven in Appendix B, Lemma 19.

- We prove the Hoare triple

$$\begin{array}{c}\{\ C(x) * I(x,z) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\mathtt{T})\ \}\\ \mathtt{with}\ C(x) * I(x,z) \wedge \text{state}(x,\mathtt{T})\ \mathtt{do}\ \mathtt{disconnect}(I,x,z)\\ \{\ C(x) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\mathtt{T})\ \}\end{array}$$

  via the *reconfiguration rules* given in Section 5.1. We apply the rule for $\mathtt{with}\ \psi\ \mathtt{do}$ first and afterwards the frame rule, which is applicable since $\mathtt{disconnect}(I,x,z) \in \mathcal{L}_X\langle C, I\rangle$ and the modified tuples of variables in the program do not intersect with the tuples of variables in the frame:

$$\dfrac{\dfrac{\{\ I(x,z)\ \}\ \mathtt{disconnect}(I,x,z)\ \{\ \mathtt{emp}\ \}}{\{\ C(x) * I(x,z) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\mathtt{T})\ \}\ \mathtt{disconnect}(I,x,z)}\ (*)}{\dfrac{\{\ C(x) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\mathtt{T})\ \}}{\{\ C(x) * I(x,z) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\mathtt{T})\ \}}\ (\mathtt{with}\ \psi\ \mathtt{do})}$$
$$\mathtt{with}\ C(x) * I(x,z) \wedge \text{state}(x,\mathtt{T})\ \mathtt{do}\ \mathtt{disconnect}(I,x,z)$$
$$\{\ C(x) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\mathtt{T})\ \}.$$

- It is $C(x) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\text{T}) \equiv \text{chain}^\text{c}(z,x,h,t)$, where chain^c is defined in Section B.2. Furthermore, it is shown in Theorem 9 that $\text{chain}^\text{c}(z,x,h,t)$ is invariant under `havoc`. Hence the following Hoare triple is valid:

$$\begin{array}{c}\{\ C(x) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\text{T})\ \} \\ \texttt{havoc} \\ \{\ C(x) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\text{T})\ \}.\end{array}$$

- The next Hoare triple follows via the frame rule and the axiom for the addition of new components:

$$\dfrac{\{\ \texttt{emp}\ \}\ \texttt{new}(C,y)\ \{\ C(y)\ \}}{\{\ C(x) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\text{T})\ \}}\ (*)$$
$$\begin{array}{c}\texttt{new}(C,y) \\ \{\ C(x) * C(y) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\text{T}) \wedge \text{state}(y,\text{H})\ \}.\end{array}$$

- This havoc triple follows with

$$\begin{aligned} & C(x) * C(y) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\text{T}) \wedge \text{state}(z,\text{H}) \\ \equiv\ & \text{chain}^\text{c}(z,x,h,t) * C(y). \end{aligned}$$

  We set $\Sigma_c = \Sigma_\theta(\text{chain}^\text{c}(z,x,h,t))$ for an injective substitution θ, $\Sigma_y = \Sigma(C(y)) = \emptyset$ and $\Sigma = \Sigma_c \cup \Sigma_y$. Then it is $\Sigma^* = \Sigma_c^* \bowtie_{\Sigma_c,\Sigma_y} \Sigma_y^*$ and we use the rules $(\bowtie)$, (ϵ) and Theorem 9 to derive

$$\dfrac{\dfrac{\text{follows by Theorem 9}}{\{\ \text{chain}^\text{c}(z,x,h,t)\ \}\ \Sigma_c^*\ \{\ \text{chain}^\text{c}(z,x,h,t)\ \}} \quad \dfrac{\dfrac{}{\{\ C(y)\ \}\ \epsilon\ \{\ C(y)\ \}}\ (\epsilon)}{\{\ C(y)\ \}\ \Sigma_y^*\ \{\ C(y)\ \}}}{\dfrac{\{\ \text{chain}^\text{c}(z,x,h,t) * C(y)\ \}\ \Sigma_c^* \bowtie_{\Sigma_c,\Sigma_y} \Sigma_y^*\ \{\ \text{chain}^\text{c}(z,x,h,t) * C(y)\ \}}{\{\ \text{chain}^\text{c}(z,x,h,t) * C(y)\ \}\ \Sigma^*\ \{\ \text{chain}^\text{c}(z,x,h,t) * C(y)\ \}.}}\ (\bowtie)$$

- This Hoare triple follows via the frame rule and the axiom for the connection of two components via an interaction:

$$\dfrac{\{\ \texttt{emp}\ \}\ \texttt{connect}(I,y,z)\ \{\ I(y,z)\ \}}{\{\ C(x) * C(y) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\text{T}) \wedge \text{state}(y,\text{H})\ \}}\ (*)$$
$$\begin{array}{c}\texttt{connect}(I,y,z) \\ \{\ C(x) * C(y) * I(y,z) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\text{T}) \wedge \text{state}(y,\text{H})\ \}.\end{array}$$

- The next havoc triple follows with

$$\begin{aligned} & C(x) * C(y) * I(y,z) * \text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\text{T}) \wedge \text{state}(y,\text{H}) \\ \equiv\ & C(y) * I(y,z) * \text{chain}^*(z,x,h,t) \wedge \text{state}(y,\text{H}) \\ \equiv\ & \text{chain}^\text{c}(y,x,h+1,t) \end{aligned}$$

  and Theorem 9.

- The proof of this Hoare triple is similar to the proof of the previous triple:

$$\frac{\{\ \texttt{emp}\ \}\ \texttt{connect}(I,x,y)\ \{\ I(x,y)\ \}}{\{\ C(x)*C(y)*I(y,z)*\text{chain}^*(z,x,h,t-1)\wedge\text{state}(x,\text{T})\wedge\text{state}(y,\text{H})\ \}}\ (*)$$
$$\texttt{connect}(I,x,y)$$
$$\{\ C(x)*I(x,y)*C(y)*I(y,z)*\text{chain}^*(z,x,h,t-1)\wedge\text{state}(x,\text{T})\wedge\text{state}(y,\text{H})\ \}.$$

- Last but not least we show the final implication by folding the chain two times:

$$\begin{aligned}&C(x)*I(x,y)*C(y)*I(y,z)*\text{chain}^*(z,x,h,t-1)\wedge\text{state}(x,\text{T})\wedge\text{state}(y,\text{H})\\ \models\ &C(x)*I(x,y)*\text{chain}^*(y,x,h+1,t-1)\wedge\text{state}(x,\text{T})\\ \models\ &\text{chain}(x,x,h+1,t)\ \models\ \text{token\_ring}(x)\models\exists x.\ \text{token\_ring}(x),\end{aligned}$$

  since $t\geq 1$ and $h+1\geq 1$.

Hence we have proven the correctness of the program. □

Now we prove the correctness of the reconfiguration program for the deletion of a component with a token. Similar to the previous proof, we show this next theorem by applying the rule for sequential composition and then proving each of the premises one by one.

Theorem 7. *Let $P_{\texttt{delete,T}}$ be the program given in Listing 6.2. Then the reconfiguration program $P_{\texttt{delete,T}}$ is correct, meaning that*

$$\{\ \text{token\_ring}^{\text{T}}(a)\ \}\ P_{\texttt{delete,T}}\ \{\ \exists a.\,\text{token\_ring}(a)\ \}.$$

Proof. Again, we apply the rule for sequential composition first. To make the proof sketch more readable, we define $F := [\text{chain}^*(z,x,h-1,t)\wedge\text{state}(x,\text{H})]\vee[\text{chain}^*(z,x,h,t-1)\wedge\text{state}(x,\text{T})]$ for $1\leq h,t\in\mathbb{N}_0$ and obtain:

$\{\ \text{token\_ring}^{\text{T}}(a)\ \}$
$\{\ \exists x,y,z.\ \ C(x)*I(x,y)*C(y)*I(y,z)*F\wedge\text{state}(y,\text{T})\ \}$
with $I(x,y)*C(y)*I(y,z)\wedge\text{state}(y,\text{T})$ **do**
  `disconnect(I,y,z)`
$\{\ C(x)*I(x,y)*C(y)*F\wedge\text{state}(y,\text{T})\ \}$
  `havoc`
$\{\ C(x)*I(x,y)*C(y)*F\wedge\text{state}(y,\text{T})\ \}$
  `disconnect(I,x,y)`
$\{\ C(x)*C(y)*F\wedge\text{state}(y,\text{T})\ \}$
  `havoc`
$\{\ C(x)*C(y)*F\wedge\text{state}(y,\text{T})\ \}$
  `delete(C,y)`
$\{\ C(x)*F\ \}$
  `havoc`
$\{\ C(x)*F\ \}$
  `connect(I,x,z)`
$\{\ C(x)*I(x,z)*F\ \}$
$\{\ \exists x.\ \ \text{token\_ring}(x)\ \}$

We prove each of the resulting Hoare triples individually:

- The implication $\text{token\_ring}^{\text{T}}(a) \models \exists x, y, z.\ C(x) * I(x,y) * C(y) * I(y,z) * F \wedge \text{state}(y, \text{T})$ is proven in Appendix B, Lemma 20.
- We prove the Hoare triple

$$\begin{gathered}\{\ C(x) * I(x,y) * C(y) * I(y,z) * F \wedge \text{state}(y,\text{T})\ \} \\ \texttt{with}\ I(x,y) * C(y) * I(y,z) \wedge \text{state}(y,\text{T})\ \texttt{do}\ \texttt{disconnect}(I,y,z) \\ \{\ C(x) * I(x,y) * C(y) * F \wedge \text{state}(y,\text{T})\ \}\end{gathered}$$

  via the *reconfiguration rules* given in Section 5.1. We apply the rule for $\texttt{with}\,\psi\,\texttt{do}$ first and afterwards the frame rule, which is again applicable since $\texttt{disconnect}(I,x,z) \in \mathcal{L}_X\langle C, I\rangle$ and the modified tuples of variables in the program do not intersect with the tuples of variables in the frame:

$$\dfrac{\dfrac{\{\ I(y,z)\ \}\ \texttt{disconnect}(I,y,z)\ \{\ \texttt{emp}\ \}}{\{\ C(x) * I(x,y) * C(y) * I(y,z) * F \wedge \text{state}(y,\text{T})\ \}}\ (*)}{\begin{gathered}\texttt{disconnect}(I,y,z) \\ \{\ C(x) * I(x,y) * C(y) * F \wedge \text{state}(y,\text{T})\ \}\end{gathered}}\ (\texttt{with}\ \psi\ \texttt{do})$$

$$\begin{gathered}\texttt{with}\ I(x,y) * C(y) * I(y,z) \wedge \text{state}(y,\text{T})\ \texttt{do}\ \texttt{disconnect}(I,y,z) \\ \{\ C(x) * I(x,y) * C(y) * F \wedge \text{state}(y,\text{T})\ \}.\end{gathered}$$

- The havoc triple follows via

$$\begin{aligned}& C(x) * I(x,y) \wedge \text{state}(y,\text{T}) * F \\ \equiv\ & C(x) * I(x,y) \wedge \text{state}(y,\text{T}) * ([\text{chain}^*(z,x,h-1,t) \wedge \text{state}(x,\text{H})] \\ & \vee [\text{chain}^*(z,x,h,t-1) \wedge \text{state}(x,\text{T})]) \\ \equiv\ & \text{chain}^*(z,y,h,t) * C(y) \wedge \text{state}(y,\text{T}) \\ \equiv\ & \text{chain}^{\text{c}}(z,y,h,t+1)\end{aligned}$$

  and Theorem 9, since $\text{chain}^{\text{c}}(z,y,h,t+1)$ is `havoc` invariant.
- The next Hoare triple follows via the frame rule and the axiom for the disconnection of components:

$$\dfrac{\{\ I(x,y)\ \}\ \texttt{disconnect}(I,x,y)\ \{\ \texttt{emp}\ \}}{\{\ C(x) * I(x,y) * C(y) * F \wedge \text{state}(y,\text{T})\ \}\ \texttt{disconnect}(I,x,y)\ \{\ C(x) * C(y) * F \wedge \text{state}(y,\text{T})\ \}.}\ (*)$$

- Analogous to the proof in Theorem 6 we can use the rules $(\bowtie)$ and (ϵ) and Theorem 9 to derive this havoc triple.
- This Hoare triple follows via the frame rule and the axioms for the deletion of components:

$$\frac{\{\, C(y) \wedge \text{state}(y, \text{T}) \,\}\ \texttt{delete}(C, y)\ \{\, \texttt{emp} \,\}}{\{\, C(x) * C(y) * F \wedge \text{state}(y, \text{T}) \,\}\ \texttt{delete}(C, y)\ \{\, C(x) * F \,\}.}\ (*)$$

- This havoc triple follows again with Theorem 9 and since

$$C(x) * F \equiv \text{chain}^{\text{c}}(z, x, h, t).$$

- This triple follows via the frame rule and the axioms for the connection of two components via an interaction:

$$\frac{\{\, \texttt{emp} \,\}\ \texttt{connect}(I, x, z)\ \{\, I(x, z) \,\}}{\{\, C(x) * F \wedge \text{state}(y, \text{T}) \,\}\ \texttt{connect}(I, x, z)\ \{\, C(x) * I(x, z) * F \,\}.}\ (*)$$

- Last but not least the final implication holds, since $h, t \geq 1$ and

$$\begin{aligned} &C(x) * I(x, z) * ([\text{chain}^*(z, x, h-1, t) \wedge \text{state}(x, \text{H})] \vee [\text{chain}^*(z, x, h, t-1) \wedge \text{state}(x, \text{T})]) \\ &\models \text{chain}^*(x, x, h, t) \models \text{chain}(x, x, h, t) \models \text{token\_ring}(x) \models \exists x.\ \text{token\_ring}(x). \end{aligned}$$

Hence we have proven the Hoare triple correct. □

The reconfiguration program in Listing 6.3 should only get executed if the ring contains at least two components in state H. Hence the theorem is analogous, but we change the precondition to a token ring with at least two components without tokens.

Theorem 8. *Let $P_{\texttt{delete},\text{H}}$ be the program given in Listing 6.3. Then the reconfiguration program given by $P_{\texttt{delete},\text{H}}$ is correct, meaning that*

$$\{\, \text{token\_ring}^{\text{H}}(a) \,\}\ P_{\texttt{delete},\text{H}}\ \{\, \exists a.\, \text{token\_ring}(a) \,\}.$$

The proof of this theorem is analogous to the proof of Theorem 7, only the states H and T need to be exchanged and the order of the deletion of the interactions is turned around.

7. Conclusion

In this work, we have used separation logic to verify the correctness of reconfiguration programs on component-based distributed systems based on the BIP framework. For that purpose, we have defined BIP configurations that model such systems and define the states of the components. The interactions in the BIP configurations may execute and follow the concrete semantics (where interactions can only fire if they are complete and enabled). Furthermore, configurations can be composed via a cancellative operation $\bullet$. Then we specified a separation logic on BIP that allows us to state the existence of components and interactions, check the states of components, and define parametric systems via inductive predicates. Moreover, we proposed a reconfiguration language that contains commands for the creation and deletion of components and interactions, commands that check properties, and commands for the composition of programs. In the following, we have specified Hoare triples and gave axioms and reconfiguration rules for the reasoning about programs (while disregarding the firing of interactions). Then we defined another set of axioms and the havoc rules that are used for the reasoning about the execution of interactions in a BIP configuration. We have proven the soundness of all defined inference rules. Lastly, we created reconfiguration programs for the token ring, defined pre- and postconditions, and proved the correctness of the programs via the inference rules.

The reconfiguration rules were mainly inspired by the inference rules for abstract separation logic in [COY07]. However, we could not apply the abstract rules directly, since they were given for local programs on static resources and our programs are not generally local and the BIP configurations are concurrent resources (because interactions can fire nondeterministically). Hence we needed to adapt the rules for sequential composition and the Kleene star such that they consider the possible state changes through execution of interactions. Furthermore, we generalized locality to X-locality and proved the frame rule for X-local programs.

The havoc rules enable us to reason about the firing of interactions and provide a way to prove Hoare triples for composed programs automatically. The rules use a completely different approach than the one for concurrent separation logic given in [COY07]. Where they construct traces with race checks fully syntactically and check the traces afterwards, we prove the set of all possible words directly via the inference rules.

We conclude that the adaption of the inference rules in abstract separation logic leads to a restriction of the frame rule that limits its application to atomic programs. In order to enable local reasoning for arbitrary programs, we propose the havoc rules. Amongst others, they contain the composition rule ($\bowtie$) that allows the local reasoning on concurrent BIP configurations (without the execution of commands). By combining the two sets of rules, we are able to reason locally on composed reconfiguration programs. We have successfully proven the correctness of reconfiguration programs on token rings of arbitrary size. The verification of the correctness of reconfiguration programs on other parametric systems should be

similar.

For future work, it would be interesting to verify the correctness of reconfiguration programs for systems that are more complex than a token ring and we have already started by modeling the dining philosophers problem and proving the correctness of a reconfiguration program. Furthermore, we would like to prove the completeness of the inference rules and automatize the proofs.

Bibliography

[BBB+11] Ananda Basu, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. Rigorous component-based system design using the BIP framework. *IEEE Softw.*, 28(3):41–48, 2011.

[BBBS18] Rim El Ballouli, Saddek Bensalem, Marius Bozga, and Joseph Sifakis. Programming dynamic reconfigurable systems. In Kyungmin Bae and Peter Csaba Ölveczky, editors, *Formal Aspects of Component Software - 15th International Conference, FACS 2018, Pohang, South Korea, October 10-12, 2018, Proceedings*, volume 11222 of *Lecture Notes in Computer Science*, pages 118–136. Springer, 2018.

[BDP11] James Brotherston, Dino Distefano, and Rasmus Lerchedahl Petersen. Automated cyclic entailment proofs in separation logic. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2011.

[BI20] Marius Bozga and Radu Iosif. Verifying safety properties of inductively defined parameterized systems. *CoRR*, abs/2008.04160, 2020.

[COY07] Cristiano Calcagno, Peter W. O'Hearn, and Hongseok Yang. Local action and abstract separation logic. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 366–378. IEEE Computer Society, 2007.

[DHA09] Robert Dockins, Aquinas Hobor, and Andrew W. Appel. A fresh look at separation algebras and share accounting. In Zhenjiang Hu, editor, *Programming Languages and Systems, 7th Asian Symposium, APLAS 2009, Seoul, Korea, December 14-16, 2009. Proceedings*, volume 5904 of *Lecture Notes in Computer Science*, pages 161–177. Springer, 2009.

[Dij71] Edsger W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, 1:115–138, 1971.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[IO01] Samin S. Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In Chris Hankin and Dave Schmidt, editors, *Conference Record*

of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001, pages 14–26. ACM, 2001.

[O'H19] Peter W. O'Hearn. Separation logic. *Commun. ACM*, 62(2):86–95, 2019.

[ORY01] Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In Laurent Fribourg, editor, *Computer Science Logic, 15th International Workshop, CSL 2001. 10th Annual Conference of the EACSL, Paris, France, September 10-13, 2001, Proceedings*, volume 2142 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2001.

[Rey02] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, pages 55–74. IEEE Computer Society, 2002.

[ZC09] Michael Zhivich and Robert K. Cunningham. The real cost of software errors. *IEEE Secur. Priv.*, 7(2):87–90, 2009.

A. X-Locality

Here, we give a few lemmata that are used in Section 5.2 or are interesting additional statements. The first statement analyzes the operation that lifts sets.

Lemma 12. *Let $X, Y \subset \mathcal{V} \times C$ be sets and $P \in \mathcal{P}(\Sigma_{\langle C,I \rangle})$ a set of BIP configurations. Then*

$$(P\uparrow_X)\uparrow_Y = P\uparrow_{X\cup Y}$$
$$\text{and } P\uparrow_X \cup P\uparrow_Y \subseteq P\uparrow_{X\cup Y}\,.$$

Proof. $(P\uparrow_X)\uparrow_Y \subseteq P\uparrow_{X\cup Y}$: Each element $(\mathfrak{S},\varsigma,\nu) \in (P\uparrow_X)\uparrow_Y$ differs on $\mathcal{V}(Y)$ in ν and on $\{(\nu(x),C_i) \mid (x,C_i) \in Y\}$ in ς from a BIP configuration $(\mathfrak{S}',\varsigma',\nu') \in P\uparrow_X$, which itself differs on $\mathcal{V}(X)$ in ν' and $\{(\nu'(x),C_i) \mid (x,C_i) \in X\}$ in ς' from a BIP configuration $(\mathfrak{S}'',\varsigma'',\nu'') \in P$. Hence $(\mathfrak{S},\varsigma,\nu)$ differs from a BIP configuration $(\mathfrak{S}'',\varsigma'',\nu'') \in P$ on $\mathcal{V}(X \cup Y)$ in ν and on $\{(\nu(x),C_i) \mid (x,C_i) \in X \cup Y\}$ in ς and $(\mathfrak{S},\varsigma,\nu) \in P\uparrow_{X\cup Y}$.
$P\uparrow_{X\cup Y} \subseteq (P\uparrow_X)\uparrow_Y$: The opposite direction follows analoguously.

$P\uparrow_X \cup P\uparrow_Y \subseteq P\uparrow_{X\cup Y}$: Each element $(\mathfrak{S},\varsigma,\nu) \in P\uparrow_X$ differs on $\mathcal{V}(X)$ in ν and on $\{(\nu(x),C_i) \mid (x,C_i) \in X\}$ in ς from an element $(\mathfrak{S}',\varsigma',\nu') \in P$, thus it differs on at most $\mathcal{V}(X \cup Y)$ in ν and on at most $\{\nu(x),C_i) \mid (x,C_i) \in X \cup Y\}$ in ς from $(\mathfrak{S}',\varsigma',\nu')$. □

We analyze how abstract X-locality carries over when using sequential composition $f;g$, non-deterministic choice $f+g$ and iteration f^*. Here, the sequential composition is a simple composition without `havoc`. Similarly, f^* is an iteration without any `havoc` in between.

Lemma 13. *Let $f,g : \Sigma \to \mathcal{P}(\Sigma)^\top$ be X- resp. Y-local, $\{\sigma\}\uparrow_{X\cup Y} = \{\sigma\}\uparrow_Y\uparrow_X$ and $\{\sigma\}\uparrow_X \cup \{\sigma\}\uparrow_Y \subseteq \{\sigma\}\uparrow_{X\cup Y}$ for any $\sigma \in \Sigma$. Furthermore ; is a simple sequential composition without any implicit state changes (the same holds for the Kleene star $(\cdot)^*$).*
Then $f;g$ and $f+g$ are $X \cup Y$-local and f^* is X-local.

Proof. Let $\sigma_0,\sigma_1 \in \Sigma$ be BIP configurations such that the concatenation $\sigma_0 \bullet \sigma_1$ is defined.

- We prove the $X \cup Y$-locality of $f;g$. It is

$$f;g(\sigma_0 \bullet \sigma_1) = f(g(\sigma_0 \bullet \sigma_1))$$
$$\subseteq f(g(\sigma_0) * \{\sigma_1\}\uparrow_Y) \subseteq f(g(\sigma_0)) * \{\sigma_1\}\uparrow_Y\uparrow_X,$$

  where we use the Y-locality of g, and the X-locality on every two elements $\sigma_0' \in g(\sigma_0)$ and $\sigma_1' \in \{\sigma_1\}\uparrow_Y$ with $\sigma_0' \bullet \sigma_1'$ defined. Since $\{\sigma_1\}\uparrow_Y\uparrow_X = \{\sigma_1\}\uparrow_{X\cup Y}$, the composition $f;g$ is $X \cup Y$-local.

- We prove the $X \cup Y$-locality of $f + g$. It is

$$f + g(\sigma_0 \bullet \sigma_1) = f(\sigma_0 \bullet \sigma_1) \cup g(\sigma_0 \bullet \sigma_1)$$
$$\subseteq (f(\sigma_0) * \{\sigma_1\}\!\uparrow_X) \cup (g(\sigma_0) * \{\sigma_1\}\!\uparrow_Y) \subseteq (f + g)(\sigma_0) * \{\sigma_1\}\!\uparrow_{X \cup Y} .$$

  The last step deserves a closer look: If $\sigma_0' \bullet \sigma_1' \in f(\sigma_0) * \{\sigma_1\}\!\uparrow_X$, then $\sigma_0' \in f(\sigma_0) \subseteq (f + g)(\sigma_0) = f(\sigma_0) \cup g(\sigma_0)$ and $\sigma_1' \in \{\sigma_1\}\!\uparrow_X \subseteq \{\sigma_1\}\!\uparrow_{X \cup Y}$. Analoguously for g and hence the subset relation holds and $f + g$ is $X \cup Y$-local.

- Last but not least we prove the X-locality of f^*. Since

$$f^* = \bigcup_{n \in \mathbb{N}_0} f^n = f^0 + f^1 + f^2 + \ldots, \quad \text{where } f^n = \underbrace{f; f; \ldots; f}_{n \text{ times}}.$$

  Hence f^* is a composition via ; and + of the X-local action f and is thus X-local.

Thus $f; g$ and $f + g$ are $X \cup Y$-local actions and f^* is an X-local action. □

Since Lemma 12 shows that the equality and the subset-relation on the lifted sets hold for BIP configurations, we can infer that the sequential composition, non-deterministic choice and Kleene operator X-local actions are X-local actions, if no action `havoc` is implicitly executed in between commands.

This is an interesting result, but in our case the composition of commands is not atomic, hence the action `havoc` is implicitly executed and the sequential composition and the Kleene operator of X-local actions are *not* X-local. Then again, the non-deterministic choice of two X-local actions is X-local.

In the proof of the previous lemma, it is obvious that we can choose a *minimal* set $Z \subset \mathcal{V} \times C$ such that an action $\ell \in \mathcal{L}\langle C, I\rangle$ is Z-local. We show that the variables in the set Z are a subset of the set of variables that are modified by the program ℓ, if $\ell \in \mathcal{L}_X\langle C, I\rangle$.

Lemma 14. *Let $\ell \in \mathcal{L}_X\langle C, I\rangle$ be a program. Then there exists a set $Z \subset \mathcal{V} \times C$ such that ℓ is Z-local and $\mathcal{V}(Z) \subseteq$ Modifies(ℓ).*

Proof. We show this via a structural induction on the program $\ell \in \mathcal{L}_X\langle C, I\rangle$ and look at the different actions:

- Let $\ell =$ `new`(C_i, x) for some component type $C_i \in C$ and a variable $x \in \mathcal{V}$. Then we have shown in Lemma 4 that the action is $\{(x, C_i)\}$-local and $\mathcal{V}(\{(x, C_i)\}) = \{x\} \subseteq \{x\} =$ Modifies(ℓ).

- If ℓ equals `delete`(C_i, x), `connect`$(I_j, x_1, \ldots, x_{\alpha(j)})$, `disconnect`$(I_j, x_1, \ldots, x_{\alpha(j)})$ or `skip`, then ℓ is local, hence $\emptyset$-local and $\mathcal{V}(\emptyset) = \emptyset =$ Modifies(ℓ).

- Let $\ell =$ `when` ϕ `do` ℓ' for some X-local program $\ell' \in \mathcal{L}_X\langle C, I\rangle$, then ℓ is also X-local and, since Modifies(`when` ϕ `do` ℓ') = Modifies(ℓ'), we derive from the induction hypothesis that $\mathcal{V}(X) \subseteq$ Modifies(ℓ') = Modifies(`when` ϕ `do` ℓ').

- If $\ell = \ell_0 + \ell_1$, then Lemma 13 shows that ℓ is $X \cup Y$ local, if ℓ_0 is X-local and ℓ_1 is Y-local ($X, Y \subseteq \mathcal{V} \times C$). Furthermore we assume that $\mathcal{V}(X) \subseteq \text{Modifies}(\ell_0)$ and $\mathcal{V}(Y) \subseteq \text{Modifies}(\ell_1)$. Then it follows that

$$\mathcal{V}(X \cup Y) = \mathcal{V}(X) \cup \mathcal{V}(Y) \subseteq \text{Modifies}(\ell_0) \cup \text{Modifies}(\ell_1) = \text{Modifies}(\ell).$$

Hence the statement is correct. □

Lastly, we show that the command $\texttt{when}\ \phi\ \texttt{do}$ is X-local.

Lemma 15. *Let $\ell \in \mathcal{L}_X\langle C, I\rangle$ be an X-local action. Then $\texttt{when}\ \phi\ \texttt{do}\ \ell$ is X-local.*

Proof. Let $(\mathfrak{S}_0, \varsigma_0, \nu_0), (\mathfrak{S}_1, \varsigma_1, \nu_1) \in \Sigma_{\langle C,I\rangle}$ be two BIP configurations such that $(\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1)$ is defined. Then

$$\begin{aligned}
&[\![\texttt{when}\ \phi\ \texttt{do}\ \ell]\!]((\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1)) \\
&= \{\, [\![\ell]\!]((\mathfrak{S}_0, \varsigma_0, \nu_0) \bullet (\mathfrak{S}_1, \varsigma_1, \nu_1)) \mid (\mathfrak{S}, \varsigma, \nu)_0 \bullet (\mathfrak{S}, \varsigma, \nu)_1 \models \phi \,\} \\
&\overset{X\text{-local}}{=} \{\, [\![\ell]\!](\mathfrak{S}_0, \varsigma_0, \nu_0) * \{(\mathfrak{S}_1, \varsigma_1, \nu_1)\}\!\uparrow_X \mid (\mathfrak{S}, \varsigma, \nu)_0 \bullet (\mathfrak{S}, \varsigma, \nu)_1 \models \phi \,\} \\
&\underset{\text{closed}}{\overset{\phi\ \text{downward}}{\subseteq}} \{\, [\![\ell]\!](\mathfrak{S}_0, \varsigma_0, \nu_0) * \{(\mathfrak{S}_1, \varsigma_1, \nu_1)\}\!\uparrow_X \mid (\mathfrak{S}, \varsigma, \nu)_0 \models \phi \,\} \\
&= \{\, [\![\ell]\!](\mathfrak{S}_0, \varsigma_0, \nu_0) \mid (\mathfrak{S}, \varsigma, \nu)_0 \models \phi \,\} * \{(\mathfrak{S}_1, \varsigma_1, \nu_1)\}\!\uparrow_X \\
&= [\![\texttt{when}\ \phi\ \texttt{do}\ \ell]\!](\mathfrak{S}_0, \varsigma_0, \nu_0) * \{(\mathfrak{S}_1, \varsigma_1, \nu_1)\}\!\uparrow_X,
\end{aligned}$$

hence $\texttt{when}\ \phi\ \texttt{do}\ \ell$ is an X-local action. □

B. Proofs for Token Ring Example

In this chapter, we prove a few statements that are necessary for the proofs in Chapter 6. In the first section, we show a few implications of formulae in separation logic on BIP. And in the second section, we show the invariance under `havoc` of the predicate chain^{c}.

B.1. Implications

The following lemmata are used in the proofs of the correctness of the reconfiguration programs in Chapter 6.

Lemma 16. *The implication*

$$\text{chain}^*(a,d,h,t) \models \exists x,z\,.\ \text{chain}^*(a,x,h_0,t_0) * C(x) * I(x,z) * \text{chain}^*(z,d,h_1,t_1) \wedge \text{state}(x,\text{T})$$

holds for some $x,z \in \mathcal{V}$ and $h,t \in \mathbb{N}_0$, $t \geq 1$, and $h_0,h_1,t_0,t_1 \in \mathbb{N}_0$. Furthermore, it is $h_0+h_1 = h$ and $t_0+t_1+1 = t$.

Proof. The lemma is proven via natural induction on the sum of h and t for $h+t \geq 1$.

- Base Case: Let $h+t=1$, hence $t=1$ and $h=0$. Then there exists only one possible unfolding of the predicate chain^*, namely

  $$\text{chain}^*(a,d,0,1) \models C(a) * I(a,d) \wedge \text{state}(a,\text{T}).$$

  We obtain

  $$C(a) * I(a,d) \wedge \text{state}(a,\text{T}) \models \text{chain}^*(a,a,0,0) * C(a) * I(a,d) * \text{chain}^*(d,d,0,0) \wedge \text{state}(a,\text{T})$$

  and the hypotheses follows.

- Induction Hypothesis: There exists an arbitrary but fixed $n \in \mathbb{N}_0$ such that for any $h,t \in \mathbb{N}_0$ with $h+t=n$ and $t \geq 1$ the implication holds.

- Induction Step: Suppose that $\tilde{h}+\tilde{t} = n+1$, $\tilde{h},\tilde{t} \in \mathbb{N}_0$ and $\tilde{t} \geq 1$. Generally, there exist two possible unfoldings for the predicate chain^*.

  - First Case: Suppose that

    $$\text{chain}^*(a,d,\tilde{h},\tilde{t}) \models \exists b\,.\ C(a) * I(a,b) * \text{chain}^*(b,d,\tilde{h}-1,\tilde{t}) \wedge \text{state}(a,\text{H}),$$

then the induction hypothesis is valid on $\text{chain}^*(b,d,\tilde{h}-1,\tilde{t})$, since $\tilde{h}-1+\tilde{t}=n$ and hence

$$\begin{aligned}
&\exists b\,.\, C(a)*I(a,b)*\text{chain}^*(b,d,\tilde{h}-1,\tilde{t})\wedge\text{state}(a,\text{H})\\
\models&\exists b,x,z\,.\, C(a)*I(a,b)*\text{chain}^*(b,x,\tilde{h}_0,\tilde{t}_1)*C(x)*I(x,z)*\text{chain}^*(z,d,\tilde{h}_1,\tilde{t}_1)\\
&\wedge\text{state}(a,\text{H})\wedge\text{state}(x,\text{T})\\
\models&\exists x,z\,.\,\text{chain}^*(a,x,\tilde{h}_0+1,\tilde{t}_1)*C(x)*I(x,z)*\text{chain}^*(z,d,\tilde{h}_1,\tilde{t}_1)\wedge\text{state}(x,\text{T})
\end{aligned}$$

and the hypothesis follows.

– Second Case: Suppose that

$$\begin{aligned}
&\text{chain}^*(a,d,\tilde{h},\tilde{t})\\
\models&\exists b\,.\, C(a)*I(a,b)*\text{chain}^*(b,d,\tilde{h},\tilde{t}-1)\wedge\text{state}(a,\text{T})\\
\models&\exists b\,.\,\text{chain}^*(a,a,0,0)*C(a)*I(a,b)*\text{chain}^*(b,d,\tilde{h},\tilde{t}-1)\wedge\text{state}(a,\text{T}),
\end{aligned}$$

where the second implication follows, since $\text{chain}^*(a,a,0,0)$ unfolds to `emp`.

Hence the induction hypothesis is correct for all $1\leq h+t\in\mathbb{N}_0$ and $t\leq 1$. □

In the following lemmata and proofs we use the proof rules that are also used for cyclic proofs (see [BDP11]).

Lemma 17. *It holds that*

$$\text{chain}^*(a,b,h_0,t_0)*\text{chain}^*(b,c,h_1,t_1)\models\text{chain}^*(a,c,h_0+h_1,t_0+t_1)$$

for variables $a,b,c\in\mathcal{V}$ and natural numbers $h_0,h_1,t_0,t_1\in\mathbb{N}_0$.

Proof. We show the statement via natural induction over the sum $h_0+t_0\in\mathbb{N}_0$.

- Base Case: Suppose that $0=:n\in\mathbb{N}_0$, $a,b,c\in\mathcal{V}$ are some variables and $h_0,t_0,h_1,t_1\in\mathbb{N}_0$ are natural numbers such that $h_0+t_0=n$. Hence $h_0=t_0=0$ and

$$\dfrac{\dfrac{\texttt{true}}{\texttt{emp}*\text{chain}^*(a,c,h_1,t_1)\models\text{chain}^*(a,c,h_1,t_1)}}{\text{chain}^*(a,b,0,0)*\text{chain}^*(b,c,h_1,t_1)\models\text{chain}^*(a,c,h_1,t_1).}\,(\text{lu})$$

- Induction Hypothesis: There exists an arbitrary but fixed $n\in\mathbb{N}_0$ such that for all $h_0,h_1,t_0,t_1\in\mathbb{N}_0$ with $h_0+t_0=n$ it follows that

$$\text{chain}^*(a,b,h_0,t_0)*\text{chain}^*(b,c,h_1,t_1)\models\text{chain}^*(a,c,h_0+h_1,t_0+t_1)$$

  for variables $a,b,c\in\mathcal{V}$.

- Induction Step: Let $h_0,h_1,t_0,t_1\in\mathbb{N}_0$ be natural numbers such that $h_0+t_0=n+1$. Furthermore $a,b,c\in\mathcal{V}$ are variables. Then the statement follows via a prooftree, where we use the induction hypothesis in step (IH):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\texttt{true}}{C(a)*I(a,a')*\text{chain}^*(a',c,h_0+h_1-1,t_0+t_1)\wedge\text{state}(a,\text{H})}}{\models C(a)*I(a,a')*\text{chain}^*(a',c,h_0+h_1-1,t_0+t_1)\wedge\text{state}(a,\text{H})}\,(\exists)}{\models \exists a'\,.\,C(a)*I(a,a')*\text{chain}^*(a',c,h_0+h_1-1,t_0+t_1)\wedge\text{state}(a,\text{H})}\,(\text{ru})}{\models \text{chain}^*(a,c,h_0+h_1,t_0+t_1)}\,(\text{IH})}{C(a)*I(a,a')*\text{chain}^*(a',b,h_0-1,t_0)*\text{chain}^*(b,c,h_1,t_1)\wedge\text{state}(a,\text{H})}$$

analogous proof for state(a,T)

$$\dfrac{\models \text{chain}^*(a,c,h_0+h_1,t_0+t_1)}{\text{chain}^*(a,b,h_0,t_0)*\text{chain}^*(b,c,h_1,t_1)\models\text{chain}^*(a,c,h_0+h_1,t_0+t_1).}\,(\text{lu})$$

Note that if either h_0 or t_0 are equal to 0, then there exists only one possible unfolding at step (lu). Hence only one of the branches applies and the proof is even simpler. We will leave it out here, because it is redundant.

We have shown the statement via natural induction. □

Lemma 18. *It holds that*

$$\begin{aligned}\text{chain}^*(a,c,h,t)\models\ \exists b\,.\,&[\text{chain}^*(a,b,h,t-1)*C(b)*I(b,c)\wedge\text{state}(b,\text{T})]\\ &\vee[\text{chain}^*(a,b,h-1,t)*C(b)*I(b,c)\wedge\text{state}(b,\text{H})])\end{aligned}$$

for variables $a,c\in\mathcal{V}$ and natural numbers $h,t\in\mathbb{N}_0$ such that $h+t\geq 1$.

Proof. Similar to the proof of Lemma 17 we show the statement via natural induction over the sum $1\leq h+t\in\mathbb{N}_0$.

- Base Case: Suppose that $1=:n\in\mathbb{N}_0$, $a,c\in\mathcal{V}$ are variables and $h,t\in\mathbb{N}_0$ are natural numbers such that $h+t=n$. Then either ($h=1$ and $t=0$) or ($h=0$ and $t=1$). Since the proof is analogous, we show only the first case:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\texttt{true}}{C(a)*I(a,c)\wedge\text{state}(a,\text{H})\models\texttt{emp}*C(a)*I(a,c)\wedge\text{state}(a,\text{H})}\,(\text{ru})}{C(a)*I(a,c)\wedge\text{state}(a,\text{H})\models\text{chain}^*(a,a,0,0)*C(a)*I(a,c)\wedge\text{state}(a,\text{H})}\,(\exists)}{C(a)*I(a,c)\wedge\text{state}(a,\text{H})\models\exists b\,.\,\text{chain}^*(a,b,0,0)*C(b)*I(b,c)\wedge\text{state}(a,\text{H})}\,(\text{lu})}{\text{chain}^*(a,c,1,0)\models\exists b\,.\,\text{chain}^*(a,b,0,0)*C(b)*I(b,c)\wedge\text{state}(a,\text{H})}\,(\text{ru})$$

$$\text{chain}^*(a,c,1,0)\models\exists b\,.\,[\text{chain}^*(a,b,0,-1)*C(b)*I(b,c)\wedge\text{state}(a,\text{T})]\vee[\text{chain}^*(a,b,0,0)*C(b)*I(b,c)\wedge\text{state}(a,\text{H})].$$

- Induction Hypothesis: There exists an arbitrary but fixed $1\leq n\in\mathbb{N}_0$ such that for all $h,t\in\mathbb{N}_0$ with $h+t=n$ it follows that

$$\begin{aligned}\text{chain}^*(a,c,h,t)\models\ \exists b\,.\,&[\text{chain}^*(a,b,h,t-1)*C(b)*I(b,c)\wedge\text{state}(b,\text{T})]\\ &\vee[\text{chain}^*(a,b,h-1,t)*C(b)*I(b,c)\wedge\text{state}(b,\text{H})])\end{aligned}$$

  for variables $a,c\in\mathcal{V}$.

- Induction Step: Let $h,t\in\mathbb{N}_0$ be natural numbers such that $h+t=n+1$. Furthermore $a,c\in\mathcal{V}$ are variables. Then the statement follows via a prooftree, where we use the induction hypothesis in step (IH):

$$\dfrac{\dfrac{\dfrac{\dfrac{\texttt{true}}{C(a)*I(a,b_1)*F\wedge\text{state}(a,\text{H})\models C(a)*I(a,b_1)*F\wedge\text{state}(a,\text{H})}}{C(a)*I(a,b_1)*F\wedge\text{state}(a,\text{H})\models\exists b_1,b_2\,.\,C(a)*I(a,b_1)*F\wedge\text{state}(a,\text{H})}\,(\exists)}{C(a)*I(a,b_1)*F\wedge\text{state}(a,\text{H})}\,(\text{ru})$$

$$\models\exists b_2\,.\,[\text{chain}^*(a,b_2,h,t-1)*C(b_2)*I(b_2,c)\wedge\text{state}(b_2,\text{T})]$$
$$\vee[\text{chain}^*(a,b_2,h-1,t)*C(b_2)*I(b_2,c)\wedge\text{state}(b_2,\text{H})]$$

$$\dfrac{}{C(a)*I(a,b_1)*\text{chain}^*(b_1,c,h-1,t)\wedge\text{state}(a,\text{H})}\,(\text{IH})$$

$$\models\exists b_2\,.\,[\text{chain}^*(a,b_2,h,t-1)*C(b_2)*I(b_2,c)\wedge\text{state}(b_2,\text{T})]$$
$$\vee[\text{chain}^*(a,b_2,h-1,t)*C(b_2)*I(b_2,c)\wedge\text{state}(b_2,\text{H})]$$

analogous proof for $\text{state}(a,\text{T})$

$$\dfrac{}{\text{chain}^*(a,c,h,t)\models\exists b_2\,.\,[\text{chain}^*(a,b_2,h,t-1)*C(b_2)*I(b_2,c)\wedge\text{state}(b_2,\text{T})]}\,(\text{lu})$$
$$\vee[\text{chain}^*(a,b_2,h-1,t)*C(b_2)*I(b_2,c)\wedge\text{state}(b_2,\text{H}),$$

and $F := [\text{chain}(b_1,b_2,h-1,t-1)*C(b_2)*I(b_2,c)\wedge\text{state}(b_2,\text{T})]\vee[\text{chain}(b_1,b_2,h-2,t)*C(b_2)*I(b_2,c)\wedge\text{state}(b_2,\text{H})]$.

We have shown the statement via natural induction. □

Lemma 19. *The implication*

$$\text{token\_ring}(a)\models\exists x,z\,.\,C(x)*I(x,z)*\text{chain}^*(z,x,h,t-1)\wedge\text{state}(x,\text{T})$$

holds for a variable $a\in\mathcal{V}$ and natural numbers $1\leq h,t\in\mathbb{N}_0$.

Proof. We know that $\text{token\_ring}(a)$ unfolds to $\text{chain}(a,a,h,t)$ for $1\leq h,t\in\mathbb{N}_0$. Hence it follows via Lemma 16 that

$$\text{token\_ring}(a)\models\exists x,z\,.\,\text{chain}^*(a,x,h_0,t_0)*C(x)*I(x,z)*\text{chain}^*(z,a,h_1,t_1)\wedge\text{state}(x,\text{T})$$

for natural numbers $h_0,h_1,t_0,t_1\in\mathbb{N}_0$. Furthermore it follows via Lemma 17 that $\text{chain}^*(a,x,h_0,t_0)*\text{chain}^*(z,a,h_1,t_1)\models\text{chain}^*(z,x,h_0+h_1,t_0+t_1)$ and hence

$$\text{token\_ring}(a)\models\exists x,z\,.\,C(x)*I(x,z)*\text{chain}^*(z,x,h_0+h_1,t_0+t_1)\wedge\text{state}(x,\text{T}).$$

Furthermore we know from Lemma 16 that $h_0+h_1=h$ and $t_0+t_1+1=t$, thus the implication follows. □

Lemma 20. *The implication*

$$\text{token\_ring}^{\text{T}}(a)\models\ \exists x,y,z\,.\,C(x)*I(x,y)*C(y)*I(y,z)*\Big([\text{chain}^*(z,x,h,t-1)\wedge\text{state}(x,\text{T})]$$
$$\vee\,[\text{chain}^*(z,x,h-1,t)\wedge\text{state}(x,\text{H})]\Big)\wedge\text{state}(y,\text{T})$$

holds for a variables $a\in\mathcal{V}$ and natural numbers $1\leq h,t\in\mathbb{N}_0$.

Proof. We know that $\text{token\_ring}^{\text{T}}(a)$ unfolds to $\exists b,h,t\ .\ C(a)*I(a,b)*\text{chain}(b,a,h,t)\wedge\text{state}(a,\text{T})$ with $1\leq h,t\in\mathbb{N}_0$. It holds that $\text{chain}(b,a,h,t)\models\text{chain}^*(b,a,h,t)$ and it follows

via Lemma 18 that

$$\begin{aligned}
\text{token_ring}^{\text{T}}(a) \models\ & \exists b,c,h,t\ .\ C(a) * I(a,b) \\
& * \Big([\text{chain}^*(b,c,h,t-1) * C(c) * I(c,a) \wedge \text{state}(c,\text{T})] \\
& \vee [\text{chain}^*(b,c,h,t-1) * C(c) * I(c,a) \wedge \text{state}(c,\text{T})] \Big) \\
& \wedge \text{state}(a,\text{T}) \\
\models\ & \exists b,c,h,t\ .\ C(c) * I(c,a) * C(a) * I(a,b) \\
& * \Big([\text{chain}^*(b,c,h,t-1) \wedge \text{state}(c,\text{T})] \\
& \vee [\text{chain}^*(b,c,h-1,t) \wedge \text{state}(c,\text{H})] \big) \\
& \wedge \text{state}(a,\text{T}).
\end{aligned}$$

The second implication simply changes the order of the formula. If we match the variables in the formula, then the assertion follows. □

B.2. Invariance Under Havoc

We define a new predicate that describes a closed chain:

$$\begin{aligned}
\text{chain}^{\text{c}}(x,x,1,0) &\leftarrow C(x) \wedge \text{state}(x,\text{H}) \\
\text{chain}^{\text{c}}(x,x,0,1) &\leftarrow C(x) \wedge \text{state}(x,\text{T}) \\
\text{chain}^{\text{c}}(x,y,h,t) &\leftarrow \exists z\ .\ C(x) * I(x,z) * \text{chain}^{\text{c}}(z,y,h-1,t) \wedge \text{state}(x,\text{H}) \\
\text{chain}^{\text{c}}(x,y,h,t) &\leftarrow \exists z\ .\ C(x) * I(x,z) * \text{chain}^{\text{c}}(z,y,h,t-1) \wedge \text{state}(x,\text{T})
\end{aligned}$$

Theorem 9. *The closed chain* $\text{chain}^{\text{c}}(x,y,h,t)$ *is invariant under* `havoc` *for* $x,y \in \mathcal{V}$ *and* $h,t \in \mathbb{N}_0$ *and the following triple is correct:*

$$\{\ \text{chain}^{\text{c}}(x,y,h,t)\ \}\ \texttt{havoc}\ \{\ \text{chain}^{\text{c}}(x,y,h,t)\ \}.$$

Proof. The following proof is from a draft between Radu Iosif, Marius Bozga and me. We prove the theorem via a very large prooftree and in order to minimize the trees, we write $C^s(x) := C(x) \wedge \text{state}(x,s)$ for $s \in \{\text{H},\text{T}\}$. We write $\Sigma = \Sigma_\theta(\text{chain}^{\text{c}}(x,y,h,t))$, where θ is an injective substitution mapping all existentially quantified variables from every unfolding of $\text{chain}^{\text{c}}(x,y,h,t)$ to unique names. The first step applies the (lu) rule:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\{\, C^h(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h-1,t) \,\}\ \Sigma^*\ \{\, \mathrm{chain}^{\mathrm{c}}(x,y,h,t) \,\}\ \textbf{(A)}}}{\dfrac{\vdots}{\{\, C^t(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h,t-1) \,\}\ \Sigma^*\ \{\, \mathrm{chain}^{\mathrm{c}}(x,y,h,t) \,\}\ \textbf{(B)}}}}{\dfrac{\dfrac{\dfrac{}{\{\, C^h(x) \,\}\ \Sigma^*\ \{\, C^h(x) \,\}}\ (\epsilon)}{\{\, C^h(x) \,\}\ \Sigma^*\ \{\, \mathrm{chain}^{\mathrm{c}}(x,x,1,0) \,\}}\ (\mathrm{ru})}}{\dfrac{\dfrac{\dfrac{}{\{\, C^t(x) \,\}\ \Sigma^*\ \{\, C^t(x) \,\}}\ (\epsilon)}{\{\, C^t(x) \,\}\ \Sigma^*\ \{\, \mathrm{chain}^{\mathrm{c}}(x,x,0,1) \,\}}\ (\mathrm{ru})}}{\{\, \mathrm{chain}^{\mathrm{c}}(x,y,h,t) \,\}\ \Sigma^*\ \{\, \mathrm{chain}^{\mathrm{c}}(x,y,h,t) \,\}.}\ (\mathrm{lu})$$

Note that $\Sigma(C^h(x)) = \Sigma(C^t(x)) = \emptyset$ and $\{\epsilon\}$ is the only language on the $\emptyset$ alphabet, which takes care of the last two subgoals. The **(A)** subgoal is proven below:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\{C^h(x)\} \dagger \Sigma \setminus \{I(x,z)\}}{\{\, C^h(x) \,\}\ (\Sigma \setminus \{I(x,z)\})^*\ \{\, C^h(x) \,\}}\ (\dagger_{\bowtie})}{\left\{ C^h(x) * I(x,z) \right\}}\ (\mathrm{supp})}{(\Sigma \setminus \{I(x,z)\})^*}\ \textbf{(D)}}{\left\{ C^h(x) * I(x,z) \right\}}}{\{C^h(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h-1,t)\}}}{\{I(x,z)\}}}{\{\, C^h(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h-1,t) \,\}\ \Sigma^*\ \{\, C^h(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h-1,t) \,\}}\ (\dagger_{\bowtie})}{\{\, C^h(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h-1,t) \,\}\ \Sigma^*\ \{\, \exists z' \,.\, C^h(x) * I(x,z') * \mathrm{chain}^{\mathrm{c}}(z',y,h-1,t) \,\}}\ (\mathrm{c})}{\{\, C^h(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h-1,t) \,\}\ \Sigma^*\ \{\, \mathrm{chain}^{\mathrm{c}}(x,y,h,t) \,\}}\ (\mathrm{ru})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{\, C^h(x) \,\}\ \Sigma^*\ \{\, C^h(x) \,\}}\ (\epsilon)}{\{\, C^h(x) * I(x,z) \,\}\ \epsilon\ \{\, C^h(x) * I(x,z) \,\}}\ (\dagger_{\bowtie})}{\{\, C^h(x) \,\}\ (\Sigma \setminus \{I(x,z)\})^*\ \{\, C^h(x) \,\}}\ (\mathrm{supp})}{\left\{ C^h(x) * I(x,z) \right\}}}{(\Sigma \setminus \{I(x,z)\})^*}\ \textbf{(D)}}{\left\{ C^h(x) * I(x,z) \right\}}}{\{C^h(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h-1,t)\}}}{\{I(x,z)\}}}{\{\, C^h(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h-1,t) \,\}\ \Sigma^*\ \{\, C^h(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h-1,t) \,\}}\ (\dagger_{\bowtie})}{\{\, C^h(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h-1,t) \,\}\ \Sigma^*\ \{\, \exists z' \,.\, C^h(x) * I(x,z') * \mathrm{chain}^{\mathrm{c}}(z',y,h-1,t) \,\}}\ (\mathrm{c})}{\{\, C^h(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h-1,t) \,\}\ \Sigma^*\ \{\, \mathrm{chain}^{\mathrm{c}}(x,y,h,t) \,\}}\ (\mathrm{ru})$$

The rule (ru) replaces the predicate atom $\mathrm{chain}^{\mathrm{c}}(x,y,h,t)$ occurring on the right hand side of the goal by one of its rules and a subsequent application of the consequence rule (c) instantiates the existentially quantified variable z' with the z variable occurring on the left hand side of the Hoare triple. Further, the language is restricted from Σ^* to $(\Sigma \setminus \{I(x,z)\})^*$ by an application of the $(\dagger_{\bowtie})$ rule. Note that $[\![(\Sigma \setminus \{I(x,z)\})^*]\!]$ is prefix-closed and that $[\![(\Sigma \setminus \{I(x,z)\})^* \bowtie \{I(x,z)\}^*]\!] = [\![\Sigma^*]\!]$. Next, the right hand side goal

$$\left\{C^h(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h-1,t)\right\}\ (\Sigma \setminus \{I(x,z)\})^*\ \left\{C^h(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h-1,t)\right\}$$

is reduced by an application of rule $(\bowtie)$ with:

$$\begin{array}{lcllcl} \mathcal{P}_1 & = & C^h(x) * I(x,z) & \mathcal{F}(\mathcal{P}_1,\mathcal{P}_2) & = & \mathtt{emp} \\ \mathcal{P}_2 & = & \mathrm{chain}^{\mathrm{c}}(z,y,h-1,t) & \mathcal{F}(\mathcal{P}_2,\mathcal{P}_1) & = & I(x,z) \end{array}$$

to the following subgoals:

$$\left\{C^h(x) * I(x,z)\right\}\ (\Sigma \setminus \{I(x,z)\})^*\ \left\{C^h(x) * I(x,z)\right\} \text{ and}$$
$$\{I(x,z) * \text{chain}^{\text{c}}(z,y,h-1,t)\}\ (\Sigma \setminus \{I(x,z)\})^*\ \{I(x,z) * \text{chain}^{\text{c}}(z,y,h-1,t)\}$$

The left subgoal is proven again using ($\dagger_{\bowtie}$). The right subgoal is reduced to the inductive hypothesis (*) $\{\text{chain}^{\text{c}}(z,y,h-1,t)\}\ (\Sigma \setminus I(x,z))^*\ \{\text{chain}^{\text{c}}(z,y,h-1,t)\}$ using the (supp) rule, since $I(x,z) \notin \text{supp}((\Sigma \setminus I(x,z))^*)$.

We prove now the remaining subgoal **(B)**. Note that $[\![(\Sigma \setminus \{I(x,z)\})^* \cdot I(x,z) \cdot \Sigma^*]\!] \cup [\![(\Sigma \setminus \{I(x,z)\})^*]\!] = [\![\Sigma^*]\!]$ and apply rule (∪) splitting **(B)** into two subgoals:

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{similar to } \mathbf{(C)}}{\{C^t(x) * I(x,z) * \text{chain}^{\text{c}}(z,y,h,t-1)\}\ (\Sigma \setminus \{I(x,z)\})^*\ \{C^t(x) * I(x,z) * \text{chain}^{\text{c}}(z,y,h,t-1)\}}\ \text{(c)}}{\{\exists z'\ .\ C^t(x) * I(x,z') * \text{chain}^{\text{c}}(z',y,h,t-1)\}\ (\Sigma \setminus \{I(x,z)\})^*\ \{C^t(x) * I(x,z) * \text{chain}^{\text{c}}(z,y,h,t-1)\}}\ \text{(ru)} \qquad \dfrac{\{C^t(x) * I(x,z) * \text{chain}^{\text{c}}(z,y,h,t-1)\}\ (\Sigma \setminus \{I(x,z)\})^* \cdot I(x,z) \cdot \Sigma^*\ \{\text{chain}^{\text{c}}(x,y,h,t)\}}{\{C^t(x) * I(x,z) * \text{chain}^{\text{c}}(z,y,h,t-1)\}\ \Sigma^*\ \{\text{chain}^{\text{c}}(x,y,h,t)\}}\ (\cup)}\ \mathbf{(E)}$$

The left subgoal is reduced to a Hoare triple similar to **(C)**, by an application of the rule (ru), followed by an instantiation of the existentially quantified variable z' by rule (c). The right subgoal **(E)** is proven below, by first unfolding the definition of $\text{chain}^{\text{c}}(z,y,h,t-1)$ on the left hand side:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\left\{C^t(x) * I(x,z) * C^h(z) * I(z,u) * \text{chain}^{\text{c}}(u,y,h-1,t-1)\right\}\ (\Sigma \setminus \{I(x,z)\})^* \cdot I(x,z) \cdot \Sigma^*\ \{\text{chain}^{\text{c}}(x,y,h,t)\}} \qquad \dfrac{\vdots}{\{C^t(x) * I(x,z) * C^t(z) * I(z,u) * \text{chain}^{\text{c}}(u,y,h,t-2)\}\ (\Sigma \setminus \{I(x,z)\})^* \cdot I(x,z) \cdot \Sigma^*\ \{\text{chain}^{\text{c}}(x,y,h,t)\}}}{\dfrac{\vdots}{\left\{C^t(x) * I(x,z) * C^h(z)\right\}\ (\Sigma \setminus \{I(x,z)\})^* \cdot I(x,z) \cdot \Sigma^*\ \{\text{chain}^{\text{c}}(x,z,1,1)\}} \qquad \dfrac{\vdots}{\{C^t(x) * I(x,z) * C^t(z)\}\ (\Sigma \setminus \{I(x,z)\})^* \cdot I(x,z) \cdot \Sigma^*\ \{\text{chain}^{\text{c}}(x,z,0,2)\}}}{\{C^t(x) * I(x,z) * \text{chain}^{\text{c}}(z,y,h,t-1)\}\ (\Sigma \setminus \{I(x,z)\})^* \cdot I(x,z) \cdot \Sigma^*\ \{\text{chain}^{\text{c}}(x,y,h,t)\}}\ \text{(lu)}$$

First we prove the two subgoals of **(E)** that involve no predicates:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\left\{C^t(x) * I(x,z) * C^h(z)\right\}}\,(\epsilon)}{\epsilon}\quad \dfrac{\{C^t(x) * I(x,z) * C^h(z)\}}{\Sigma \setminus \{I(x,z)\}}\,(\dagger)}{\left\{C^t(x) * I(x,z) * C^h(z)\right\}}}{(\Sigma \setminus \{I(x,z)\})^*}\,\textbf{(F)}\quad \dfrac{\dfrac{}{\left\{C^t(x) * I(x,z) * C^h(z)\right\}}\,(\alpha)}{I(x,z)}\quad \dfrac{\dfrac{\{C^h(x) * I(x,z) * C^t(z)\}}{I(x,z)}\quad \dfrac{\dfrac{}{\left\{C^h(x) * I(x,z) * C^t(z)\right\}}\,\textbf{(F)}}{(\Sigma \setminus \{I(x,z)\})^*}}{\left\{C^h(x) * I(x,z) * C^t(z)\right\}}\,(\dagger_{\bowtie})}{\Sigma^*}}{\left\{C^h(x) * I(x,z) * C^t(z)\right\}}\,(\cdot)$$

$$\dfrac{\left\{C^t(x) * I(x,z) * C^h(z)\right\}\ (\Sigma \setminus \{I(x,z)\})^* \cdot I(x,z) \cdot \Sigma^*\ \left\{C^h(x) * I(x,z) * C^t(z)\right\}}{\left\{C^t(x) * I(x,z) * C^h(z)\right\}\ (\Sigma \setminus \{I(x,z)\})^* \cdot I(x,z) \cdot \Sigma^*\ \left\{C^h(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,z,0,1)\right\}}\,(\mathrm{ru})$$

$$\dfrac{\left\{C^t(x) * I(x,z) * C^h(z)\right\}\ (\Sigma \setminus \{I(x,z)\})^* \cdot I(x,z) \cdot \Sigma^*\ \left\{\exists z'\ .\ C^h(x) * I(x,z') * \mathrm{chain}^{\mathrm{c}}(z',z,0,1)\right\}}{\left\{C^t(x) * I(x,z) * C^h(z)\right\}\ (\Sigma \setminus \{I(x,z)\})^* \cdot I(x,z) \cdot \Sigma^*\ \{\mathrm{chain}^{\mathrm{c}}(x,z,1,1)\}}\,(\mathrm{ru})$$

$$(\mathrm{c})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\{C^t(x)*I(x,z)*C^t(z)\}\ \epsilon\ \{C^t(x)*I(x,z)*C^t(z)\} \qquad \{C^t(x)*I(x,z)*C^t(z)\}\dagger\Sigma}{\{C^t(x)*I(x,z)*C^t(z)\}\ \Sigma^*\ \{C^t(x)*I(x,z)*C^t(z)\}}\ (\dagger_{\bowtie})}{\{C^t(x)*I(x,z)*C^t(z)\}\ (\Sigma\setminus\{I(x,z)\})^*\cdot I(x,z)\cdot\Sigma^*\ \{C^t(x)*I(x,z)*C^t(z)\}}\ (\subseteq)}{\{C^t(x)*I(x,z)*C^t(z)\}\ (\Sigma\setminus\{I(x,z)\})^*\cdot I(x,z)\cdot\Sigma^*\ \{C^t(x)*I(x,z')*\mathrm{chain}^{\mathrm{c}}(z,z,0,1)\}}\ (\mathrm{ru})}{\{C^t(x)*I(x,z)*C^t(z)\}\ (\Sigma\setminus\{I(x,z)\})^*\cdot I(x,z)\cdot\Sigma^*\ \{\exists z'\ .\ C^t(x)*I(x,z')*\mathrm{chain}^{\mathrm{c}}(z',z,0,1)\}}\ (\mathrm{c})}{\{C^t(x)*I(x,z)*C^t(z)\}\ (\Sigma\setminus\{I(x,z)\})^*\cdot I(x,z)\cdot\Sigma^*\ \{\mathrm{chain}^{\mathrm{c}}(x,z,0,2)\}}\ (\mathrm{ru})$$

We move on to the remaining subgoals (with predicates):

$$\dfrac{\dfrac{\text{similar to } \mathbf{(D)}}{\{C^t(x)*I(x,z)\}\ (\Sigma\setminus\{I(x,z)\})^*\ \{C^t(x)*I(x,z)\}} \qquad \dfrac{\text{similar to } \mathbf{(C)}}{\begin{array}{c}\left\{\boxed{\mathrm{I(x,z)}}*C^h(z)*I(z,u)*\mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1)\right\}\\ (\Sigma\setminus\{I(x,z)\})^*\\ \left\{\boxed{\mathrm{I(x,z)}}*C^h(z)*I(z,u)*\mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1)\right\}\end{array}}}{\left\{C^t(x)*I(x,z)*C^h(z)*I(z,u)*\mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1)\right\}\ (\Sigma\setminus\{I(x,z)\})^*\ \left\{C^t(x)*I(x,z)*C^h(z)*I(z,u)*\mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1)\right\}}\ (\bowtie)$$

$$\dfrac{\dfrac{\dfrac{}{\left\{C^t(x)*I(x,z)*C^h(z)\right\}\ I(x,z)\ \left\{C^h(x)*I(x,z)*C^t(z)\right\}}\ (\alpha)}{\left\{C^t(x)*I(x,z)*C^h(z)*\boxed{\mathrm{I(z,u)}}\right\}\ I(x,z)\ \left\{C^h(x)*I(x,z)*C^t(z)*\boxed{\mathrm{I(z,u)}}\right\}}\ (\mathrm{supp}) \qquad \dfrac{\dfrac{}{\{I(z,u)*\mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1)\}}\ (\epsilon)}{\begin{array}{c}\epsilon\\ \{I(z,u)*\mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1)\}\end{array}}}{\left\{C^t(x)*I(x,z)*C^h(z)*I(z,u)*\mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1)\right\}\ I(x,z)\ \left\{C^h(x)*I(x,z)*C^t(z)*I(z,u)*\mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1)\right\}\ \mathbf{(G)}}\ (\bowtie)$$

$$\dfrac{\dfrac{\dfrac{\mathbf{(A)}}{\left\{C^h(x)*I(x,z)*\mathrm{chain}^{\mathrm{c}}(z,y,h-1,t)\right\}\ \Sigma^*\ \{\mathrm{chain}^{\mathrm{c}}(x,y,h,t)\}}}{\left\{C^h(x)*I(x,z)*C^t(z)*I(z,u)*\mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1)\right\}\ \Sigma^*\ \{\mathrm{chain}^{\mathrm{c}}(x,y,h,t)\}}\ (\mathrm{lf})}{\left\{C^t(x)*I(x,z)*C^h(z)*I(z,u)*\mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1)\right\}\ (\Sigma\setminus\{I(x,z)\})^*\cdot I(x,z)\cdot\Sigma^*\ \{\mathrm{chain}^{\mathrm{c}}(x,y,h,t)\}}\ (\cdot)$$

$$\dfrac{\dfrac{\dfrac{\text{similar to } \mathbf{(C)}}{\{C^t(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h,t-1)\}\ (\Sigma \setminus \{I(x,z)\})^*\ \{C^t(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h,t-1)\}}\ (\mathrm{lf})}{\{C^t(x) * I(x,z) * C^t(z) * I(z,u) * \mathrm{chain}^{\mathrm{c}}(u,y,h,t-2)\}\ (\Sigma \setminus \{I(x,z)\})^*\ \{C^t(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h,t-1)\}}\ (\mathrm{lf})}{\{C^t(x) * I(x,z) * C^t(z) * I(z,u) * \mathrm{chain}^{\mathrm{c}}(u,y,h,t-2)\}\ (\Sigma \setminus \{I(x,z)\})^* \left\{ \begin{array}{c} C^t(x) * I(x,z) * C^t(z) * I(z,u) * \mathrm{chain}^{\mathrm{c}}(u,y,h,t-2) \\ \vee \\ C^t(x) * I(x,z) * C^h(z) * I(z,u) * \mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1) \end{array} \right\}}\ (\mathrm{rf})$$

$$\dfrac{\dfrac{\{C^t(x) * I(x,z) * C^t(z) * I(z,u) * \mathrm{chain}^{\mathrm{c}}(u,y,h,t-2)\} \dagger I(x,z)}{\{C^t(x) * I(x,z) * C^t(z) * I(z,u) * \mathrm{chain}^{\mathrm{c}}(u,y,h,t-2)\}\ I(x,z)\ \{\bot\}}\ (\dagger_\bot) \quad \dfrac{\mathbf{(G)}}{\begin{array}{c}\left\{C^t(x) * I(x,z) * C^h(z) * I(z,u) * \mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1)\right\} \\ I(x,z) \\ \left\{C^h(x) * I(x,z) * C^t(z) * I(z,u) * \mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1)\right\}\end{array}}}{\left\{ \begin{array}{c} C^t(x) * I(x,z) * C^t(z) * I(z,u) * \mathrm{chain}^{\mathrm{c}}(u,y,h,t-2) \\ \vee \\ C^t(x) * I(x,z) * C^h(z) * I(z,u) * \mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1) \end{array} \right\} I(x,z)\ \left\{C^h(x) * I(x,z) * C^t(z) * I(z,u) * \mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1)\right\}}\ (\vee)$$

$$\dfrac{\dfrac{\dfrac{\mathbf{(A)}}{\left\{C^h(x) * I(x,z) * \mathrm{chain}^{\mathrm{c}}(z,y,h-1,t)\right\}\ \Sigma^*\ \{\mathrm{chain}^{\mathrm{c}}(x,y,h,t)\}}\ (\mathrm{lf})}{\left\{C^h(x) * I(x,z) * C^t(z) * I(z,u) * \mathrm{chain}^{\mathrm{c}}(u,y,h-1,t-1)\right\}\ \Sigma^*\ \{\mathrm{chain}^{\mathrm{c}}(x,y,h,t)\}}\ (\mathrm{lf})}{\{C^t(x) * I(x,z) * C^t(z) * I(z,u) * \mathrm{chain}^{\mathrm{c}}(u,y,h,t-2)\}\ (\Sigma \setminus \{I(x,z)\})^* \cdot I(x,z) \cdot \Sigma^*\ \{\mathrm{chain}^{\mathrm{c}}(x,y,h,t)\}}\ (\cdot)$$

We have finally proven the invariance of $\mathrm{chain}^{\mathrm{c}}(x,y,h,t)$ under havoc. □

Acknowledgements

First and foremost, I am profoundly grateful to Joost-Pieter Katoen for introducing me to Radu Iosif and offering to supervise my Bachelor thesis. This would not have been possible without his time and support.

Furthermore, I would like to thank Radu Iosif and Marius Bozga for their motivation, for the many afternoons that we spent on discussions in front of a whiteboard or via Zoom, for their time, support and encouragement. I had a great time working with you!

Special thanks to Moritz Reitmeier for his endless support in every aspect of the process! And last but not least, I would like to thank my family for encouraging and motivating me relentlessly and making this education possible for me.