Formal Modeling and Analysis of Mobile Ad hoc Networks

Fatemeh Ghassemi Esfahani

The work reported in this thesis has been carried out at the Vrije Universiteit Amsterdam.

VRIJE UNIVERSITEIT

Formal Modeling and Analysis of Mobile Ad hoc Networks

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. V. Subramaniam,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Bètawetenschappen
op maandag 19 maart 2018 om 15.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Fatemeh Ghassemi Esfahani

geboren te Teheran, Iran

promotor: prof.dr. W.J. Fokkink
copromotor: dr. B. Luttik

Acknowledgments

I would like to deeply thank my supervisor Wan Fokkink for his endless support and patience and for his guidance on my path towards becoming a professional. It lightened my way, without it I think I could never have reached this point. His support was as a door to survival at the hardest points of my life. I always think about my lucky day I was introduced to him by Marjan Sirjani at a winter school at IPM in Tehran, January 2007. I also warmly thank my co-supervisor Bas Luttik for his constructive comments to revise and improve the thesis.

I thank Behnaz Yousefi for her enthusiasm and interest in research and collaboration. Without her support to implement the tool, the presented theories would not have been applicable. This complemented our joy in doing research, and motivated us to continue it as much as we can. Co-authors Mohammad Reza Mousavi, Ramtin Khosravi, and Behnaz Yousefi contributed to parts of my thesis. I enjoyed our fruitful discussions and hope to continue our cooperations in the future. Thanks go to the Software Engineering Department of the University of Tehran for their support in reducing my load whenever it was possible.

I gratefully acknowledge the members of the reading committee of my thesis, Farhad Arbab, Jan Bergstra, Jan Friso Groote, Massimo Merro, and Femke van Raansdonk, for their constructive comments and questions.

Rena Bakhshi and Cynthia Kop shared their office with me during my visit to the Vrije Universiteit Amsterdam in October 2009. Our friendly discussions are still a part of my best memories during this visit.

My special thanks go to my parents and Hassan's parents. Without their support to look after Rayan I would not have been able to focus on my work. I would like to express my highest gratitude to my beloved ones, Hassan and Rayan. They help me tolerate all the pressures and calm down when I am fully stressed. To Hassan, for being understanding, listening to my complaints when I encountered a flaw in a proof, and much more... To Rayan, for tolerating my absence when he needed me.

Contents

Introduction 1

Applicability of wireless communications is rapidly growing in areas such as home networks and satellite transmissions due to their high accessibility and low cost. Wireless communication has a broadcasting nature, as messages sent by each node can be received by all nodes in its transmission range, called *local broadcast*. Therefore, by paying the cost of one transmission, several nodes may receive the message, which leads to lower energy consumption for the sender and throughput improvement [43].

Mobile ad hoc networks (MANETs) consist of several portable hosts with no pre-existing infrastructure, such as routers in wired networks or access points in managed (infrastructure) wireless networks. Two nodes can effectively communicate if they are located in the communication range of each other. For unicasting a message to a specific node beyond the transmission range of a node, it is needed to relay the message by some intermediate nodes to reach the desired destination. In such networks, nodes can freely change their locations, so the network topology is constantly changing. Due to the lack of any pre-designed infrastructure and global network topology information, network functions such as routing protocols are devised in a completely distributed manner and adaptive to cope with topology changes. Topology-dependent behavior of wireless communication, distribution and adaptation requirements make the design of MANET protocols complicated and more in need of formal modeling and verification for any possible topology and its changes so that it can be trusted to work correctly. For instance, MANET protocols like the Ad hoc On Demand Distance Vector (AODV) routing protocol [125] have been revised as new failure scenarios were experienced or errors were found in the protocol design [25, 62, 120]. Some works model and analyze the correctness of MANET protocols using existing formal frameworks such as SPIN [25, 50, 151] and UPPAAL [32, 63, 80, 113, 151, 152]. The modeling approach using existing formalisms can be summarized as follows: The underlying topology is modeled by a two-dimensional array of Booleans, mobility by explicit manipulation of this matrix, and local broadcast by unicasting to all nodes with whom the sending node is presently connected, using the connectivity matrix. Verification tends to be based on model checking techniques restricted to a pre-specified mobility scenario. Lack of support for compositional modeling and arbitrary topology changes motivates us to develop a new ap-

proach, tailored to the domain of MANETs, with a primitive for local broadcast and support of arbitrary mobility to examine resistance/adaptation of MANET protocols to changes of the underlying topology.

This chapter is structured as follows. In Section 1.1, the main problem of the thesis is stated. In Section 1.2, the analysis approaches of MANET protocols are explained. The main modeling challenges of MANET protocols that should be supported are presented in Section 1.3. In Section 1.4, the works on analysis of MANETs are explored. The main objectives followed in this thesis are introduced in Section 1.5. Finally in Sections 1.6 and 1.7, we explain the structure and origin of each chapter, respectively.

1.1 Problem Statement

Domain-specific modeling and verification frameworks are advantageous as they reduce the labor of modeling by providing suitable means to specify special concepts in the domain and the level of abstraction that a modeler should focus on. The means of wireless communication in MANETs, which depends on the underlying topology, together with topology changes trigger a new demand for a domain specific framework which supports modular and compositional specification as well as analysis of such networks. Since mobility is the intrinsic characteristic of MANET nodes, a protocol should be robust in spite of any arbitrary topology changes. Therefore, the domain specific framework should support analyzing the correctness of protocols taking topology changes into account.

Regarding the Open Systems Interconnection (OSI) model, network protocols can be classified into seven layers: physical, data link, network, transport, session, presentation, and application. However, in practise, there are only four layers: data link, network, transport, and application [59]. Each layer provides a set of services to the layer directly above it. The tailored framework should provide suitable abstractions for the protocols on which we are going to focus our analysis. For instance, the wireless communication service varies at the different layers. The data link layer is responsible for transferring data across the physical link. It consists of two sublayers: Logical Link Control sublayer (LLC) and Media Access Control sublayer (MAC). LLC is mainly responsible for multiplexing packets to their protocol stacks identified by their IPs, while MAC manages accesses to prevent/reduce conflicts of messages that are sent simultaneously over the shared media. In contrast, communication at the network layer uses the service of the MAC sublayer, and hence provides point-to-point communication between two nodes that are not directly connected through appropriate routing of messages.

We focus on the *formal modeling and analysis of protocols above the data link layer* such as routing protocols. Therefore, two main problems are distinguished: 1) the modeling of topology, its changes and its interference with the wireless communication to minimize the problem of state space explosion for small net-

works, 2) Formal analysis of a protocol taking topology changes into account to identify scenarios leading to its malfunction. These two problems are closely intertwined: to reduce the effect of modeling the topology on the size of the semantic model, the level of abstraction is increased, which requires novel ideas to either exploit existing analysis techniques or provide a tailored analysis technique. Considering protocols at the data link layer requires the modeling of conflicts among the messages which make the framework much more complex. Furthermore, to model a protocol at a higher level, there is a need to abstractly specify the services of each layer. For instance, to model an application layer protocol like leader election, routing services of the network layer can be abstractly modeled by specifying the flooding-based routing protocol as a part of the model.

1.2 Analysis Approaches for MANET Protocols

Our analysis of MANET protocols targets qualitative properties. For instance, "Will all nodes finally have a leader which is the one with the maximum value in their connected component?" or "Will always data sent by a source to a destination be successfully received by the destination if there exists an end-to-end route between them for a sufficiently long period of time?". We remark that properties satisfied by MANETs tend to be weaker than those of wired networks. This is due to the topology-dependent behavior of communication, and consequently the need for multi-hop communication between nodes.

Generally speaking, analysis approaches of network protocols can be classified into two classes: simulation and formal verification. Protocols applied in MANETs are usually tested by means of simulation. In simulation, a network of nodes is modeled and then run for a set of scenarios in a specific simulation environment. In each scenario, the set of events generated by the nodes are specified. The simulation environment can take into account the physical area in which nodes are located, the time duration of simulation, the physical characteristics of nodes, and a node mobility model. Such a model defines the speed and movement direction of a node at each time. As the number of scenarios are restricted, by simulation one cannot explore all conditions that are required to hold for such systems. Formal verification methods can be used to model such networks, and then verify them using (semi-)automated model checking or theorem prover techniques or rigorous analytical approaches.

The model checking technique is a successful approach in verifying small- to medium-sized systems. In this approach, the desired properties of a system are verified on the basis of a suitable model of the system. As the complexity of a system grows, it encounters the state space explosion problem, meaning that the state space of a system grows exponentially with the number of components in the system. The mix of broadcast behavior and mobility in MANETs leads to state space explosion, hampering the application of such technique for even small-sized networks, e.g., a network of four nodes each deploying a leader

election protocol. There are several formal frameworks that aid in preparing the model of a system in such a way that the resulting state space remains small. For instance, in actor-based modeling languages, the coarse-grained executions of message handlers prevent unnecessary interleaving of the internal actions of a node with others. The level of abstraction in these languages is very useful to designers for the rapid development of MANET protocols, as they only focus on how messages should be processed, regardless of how they are received or queued. However, the fine-grained executions of message handlers in a process algebraic framework make the generation of semantic models very challenging. Furthermore, techniques like symmetry reduction [35], counter abstraction [56], and partial-order reduction [124] alleviate the state space explosion problem.

The theorem proving technique is applicable for large networks. In this approach, a proof is derived for the desired assertion about the system, given a set of axioms (about a system and modeling framework) and inference rules. Although this approach can be mechanized by a set of available theorem prover tools, it requires a considerable amount of user involvement. On the other hand, the axioms about a system can be proved with the help of model checking technique with relatively little effort. For instance, a predicate about MANETs can be verified automatically for a fixed set of topologies using a model checking tool. The combination of model checking and theorem proving techniques allows to prove a predicate about a MANET protocol with regard to arbitrary topologies [25].

In a rigorous analytical approach, a mathematical model of a system is built using a set of abstractions based on mathematical network theory, optimization, statical mechanics, graph theory and so on. For instance, in [10] gossip protocols are evaluated using the mean field abstraction technique [13] inspired from theory of epidemics. This approach scales well when the size of a network increases, but provides a very abstract view on the system.

1.3 Modeling Issues and Challenges

To model protocols above the data link layer, the core services of this layer, i.e., local broadcast as the primitive means of communication, should be supported by the modeling language either by including a local broadcast primitive or by being able to deal with such a notation. The same discussion holds for other services such as unicast and multicast. To this aim, the attributes of such services should be considered: Wireless communications at this layer are *non-blocking*, i.e., the sender broadcasts irrespective to the readiness of its receivers and is *asynchronous*, i.e., the received packets are buffered at the receivers, and (the data link layer of) each node processes the packet if it is an intended destination (in case of unicast/multicast). Thus if a node is busy processing a message, it will receive the message and process it later.

We remark that if two disconnected nodes broadcast simultaneously with a

common node in their range, the node cannot receive both messages, called the hidden node problem. To prevent/avoid such a problem, some services are provided at the layer. Hence, unicast can be either lossy or reliable. When unicast is *reliable*, the intended receiver successfully receives the packet. In other words, message delivery is *guaranteed* to the ready receivers. A reliable local broadcast can be implemented by unicasting to all the neighbors of a node. Defining the local broadcast primitive as non-blocking in a reliable setting is not straightforward.

Another service of the data link layer is *neighbor discovery*, by which a node becomes aware of its connection topology. The neighbor discovery protocol is implemented by periodically sending *hello* messages and acknowledging such messages received from a neighbor. Modeling such a service in the framework obviates the explicit modeling of this service as a part of the specification, and consequently makes modeling of protocols easier. Modeling such services in not easy when the underlying topology is not explicitly modeled by the specification.

The behavior of local broadcast depends on the topology concept. Mobility is the intrinsic characteristic of MANET nodes and network protocols have no control over the movement of MANET nodes, and hence, to analyze the behavior of a protocol in the presence of mobility, topology changes should be *arbitrary*. However, arbitrary changes of topologies make state spaces grow with a factor of 2^{n^2} which prohibits inspection of MANETs of even small sizes. Hence, either some suitable abreactions should be provided to avoid such a growth or some reduction techniques should be provided. The behavior of a network mainly depends on the local knowledge of the nodes about the network. Hence, data values and inference procedures over such data should be supported by the modeling framework. However, data makes analysis of MANETs via model checking technique challenging due to the infinite domain of data. Mobility, arbitrary changes of topology, and data modeling are all the source of state space explosion. Hence, providing reduction techniques to generate the semantic model is inevitable to make model checking feasible in the domain of MANETs. Furthermore, MANET protocols are supposed to be applied in large-scale networks. Hence, theorem prover technique will form an important ingredient for analysis of such networks. Providing a suitable mechanism in a modeling framework to make application of such a technique feasible is an important issue.

1.4 Related Work

This section provides a brief overview of the formal verification techniques that have been applied to analyze MANET protocols.

The SPIN model checker has been applied in [25, 50, 151] to verify routing protocols in ad hoc networks, namely the Wireless Adaptive Routing Protocol (WARP) [103], Lightweight Underlay Network Ad hoc Routing [142] (LUNAR), and Distance Vector Routing [125] (AODV) Protocol. In a SPIN model, node

connectivity is modeled with the help of PROMELA channels, one for each node. Mobility is modeled by the *case selection* instruction provided by PROMELA, for modeling nondeterminism. In the initialization section, possible links to other neighbours are defined as different cases that all will be checked for a model. Since it does not provide a specific technique to reduce the state space, its state space grows very fast and it is only feasible to check small topologies. Therefore, models would be limited to use fewer number of nodes. In [25], the theorem prover HOL[1] was used to generalize the desired properties of AODV to networks with any number of nodes. As we mentioned earlier, this approach needs a considerable amount of user involvement and cannot be easily exercised.

UPPAAL has been applied in [32, 63, 80, 113, 151, 152] to verify real-time aspects of routing protocols in MANETs as a network of timed automata. In UPPAAL, connectivity is modeled through a set of arrays of booleans, while changing topology is modeled by a separate automaton which manipulates the arrays. In [151], a case study was carried out to evaluate two model checkers, SPIN and UPPAAL. Due to state space explosion, the analysis was limited to some special mobility scenarios (as a part of the specification).

The backoff algorithm for MANETs was verified using PEPA, a stochastic process algebra, in [130]. In this approach the topology of the network is static and broadcast is implemented by unicasting to all nodes connected to the sender. As explained in [57], from a theoretical point view, compositionality is not preserved if broadcast is encoded in calculi based on point-to-point communications.

As we explained before, existing standard tools cannot be used in a compositional way, i.e., increasing the network size triggers a need for revising and adjusting the modeled topology and mobility scenarios. Lack of support for compositional modeling and arbitrary topology changes has motivated new approaches with a primitive for local broadcast and support of arbitrary mobility in an algebraic way. These approaches include CBS# [121], CWS [117], CMAN [78, 79], CMN [115, 116], bKlaim [122], ω-calculus [137], SCWN [81], CSDT [104], AWN [64] and its timed extension [30], and the broadcast psi-calculi [27]. We compare these approaches in detail in Section 3.7 and discuss how each approach tackles the modeling challenges of MANETs, as discussed in Section 1.3: local broadcast, neighbor discovery, data, the underlying topology, mobility, and their analysis approach.

There are different approaches [2, 51, 52] with the aim to analyze networks with an infinite number of nodes, where nodes execute an instance of a network process. A network configuration is represented as a graph in which each individual node represents a state of the process. The behavior of a process is modeled by an automaton. The network configuration transforms due to either the process evolution at a network node or the topology reconfiguration. Verification of safety properties, reaching to an undesirable configuration starting from an initial configuration, is parameterized due to any possible number of nodes

[1] http://hol.sourceforge.net/

and connectivity among them. It is proved that the problem of parameterized safety properties, the so-called *control states reachability problem*, is undecidable. However, that problem turns out to be decidable for the class of bounded path graphs [51, 52]. Decidability of the problem was also considered when configurations evolve due to discrete/continuous clocks at processes [2]. Furthermore, an inductive approach based on reduction to prove compositional invariants for the dynamic process networks was presented in [119]. This approach reduces the calculation of a compositional invariant to a smallest representative network through setting up a collection of local symmetry relations between nodes, specifically defined for each problem. The computed non-dynamic compositional invariant on the representative network is generalized for the entire dynamic network family when the non-dynamic invariant is preserved by any reaction to a dynamic change in the network. Another approach is based on graph transformation systems, where network configurations are hypergraphs and transitions are specified by graph rewriting rules, modeling the dynamic behavior of a protocol. Safety properties are symbolically specified by graph patterns, a generalized form of hypergraphs with negative conditions. To verify that any bad configuration, specified by a pattern, is not reachable from an initial configuration, a *symbolic backward reachability analysis* is introduced, which is not guaranteed to terminate due to the undecidability of the problem [134]. In this analysis, an over-approximation of the set of configurations preceding a bad configuration is computed until no new preceding configuration be generated, and then it is checked that this set contains no initial configuration. The preceding configurations are approximately computed from a given pattern and a set of graph rules. Due to the existence of negative conditions in the pattern, such a preceding set cannot be computed exactly.

1.5 Assumptions, Objective, and Results

We explained our overall research question in Section 1.1: the formal modeling and analysis of MANET protocols taking topology changes into account so that their state spaces grow only moderately, and verification approaches can cope with the topology and it arbitrary changes. To address this problem while the mentioned challenges are covered, we formulate a number of smaller research questions and how they are addressed in this thesis.

As network protocols have no control over the movement of MANET nodes, topology changes cannot be specified as a part of the specification. The first objective of this thesis is to investigate how the arbitrary mobility of nodes can be addressed compactly at the semantic level such that not only it supports efficient analysis of protocols through model checking, but also modeling frameworks can be easily defined over it. To this aim, we introduce constrained labeled transition systems (CLTSs) in which transitions are annotated by constraints over the underlying topology, called network constraints, to restrict the behaviors of a

network to those topologies that satisfy the constraints.

With the aim to design/analyze MANET protocols above the data link layer such as routing protocols, the second objective of this thesis is providing frameworks applicable for modeling and analyzing real-world MANET protocols above the data link layer. To this end, two modeling approaches, which are different in their computation model, are followed: the algebraic and actor-based approaches. The former yields an equational reasoning technique which is the best choice when the number of nodes is infinite, to overcome the state space explosion problem. Furthermore, it can be orthogonally extended with equational abstract data types [55], following the approach of [84, 85]. The actor-based approach supports a higher level of abstraction due to its asynchronous computation model and efficient semantic generation due to its coarse-grained executions of message handlers. However it restricts us to model checking techniques, which raises new objectives: how the semantic model of a specification can be generated efficiently to make model checking techniques feasible, and how the model checking technique can be extended to the proposed CLTSs. We cover the former challenge by extending the counter abstraction reduction technique [56] by considering the topological relations among the nodes in a network. The latter objective is satisfied by introducing a new logic and its model checking algorithm, which efficiently verifies topology-dependent behaviors of MANETs.

In both frameworks, we provide local broadcast (of the MAC layer) as the means of communication. With the aim to detect malfunctions of a MANET protocol caused by conceptual mistakes in the protocol design, rather than by an unreliable communication, we consider local broadcast to be reliable and non-blocking while message delivery is guaranteed for the ready receivers in the range. We assume that unicast and multicast can be modeled through local broadcast and message parameters. Furthermore, we do not consider timing issues of protocols. We restrict our models to ones with a fixed and known number of nodes, and hence they boil down to finite-state problems, making model checking feasible.

Software product line (SPL) engineering has become an established trend in software development, where a family of similar software products with minor differences, called *variability*, are developed in tandem, instead of developing each specific software product separately [128]. Product Line Calculus of Communicating Systems (PL-CCS) [89, 90] is an extension of Milner's Calculus of Communicating Systems (CCS) [118] for behavioral modeling of SPLs, in which variability can be explicitly modeled by a binary variant operator. The semantic models of PL-CCS capture the behavior of a product family at once for all the products whose transitions are annotated by restrictions over the configuration of SPL. A MANET protocol can be interpreted as a product family that behaves according to the underlying topology configuration. Similarities between the two semantic models generate the next objective: can the results of our algebraic framework be reused/extended to the semantic model of PL-CCS to furnish this framework with a proper equational theory.

1.6 Organization of Chapters

The thesis is organized in seven chapters. We explain in some detail the contents of each chapter.

Chapter 2: Preliminaries. This chapter introduces our semantic model, CLTSs. Initially, this semantic model only supports lossy communications by only considering the connectivity restrictions over the underlying network. It explains how the mobility is implicitly and compactly addressed at the semantic model. Then, computed network process theory (*CNT*), which was introduced for the modeling and verifying MANET protocols, is briefly reviewed.

Chapter 3: Reliable Restricted Broadcast Process Theory. This chapter extends CLTSs with negative constraints over the topology. Then, it modifies the core operators of *CNT* to support reliable communication, while a new operator abstracting the neighbor discovery service is introduced. Finally to illustrate the applicability of our framework, a leader election protocol for MANETs is specified.

Chapter 4: Wireless-Rebeca (WRebeca). This chapter introduces our actor-based modeling framework, based on the Rebeca language. It provides two reduction techniques to minimize semantic models to make their verification amenable. Then, it evaluates effectiveness of the proposed reduction technique through two routing protocols: flooding-based and AODV. It demonstrates the usability of the framework through the modeling and analysis of AODV, and reports a loop formation scenario in AODV, found by our analysis tool. This has led to an adaptation of the AODVv2 protocol, published in version 13.

Chapter 5: Model Checking MANETSs. This chapter introduces a branching-time temporal logic to specify the topology-dependent behaviors of MANET protocols. Then, it presents its model checking algorithm over CLTSs. The applicability of the logic and its model checking algorithm is inspected through the verification of a set of properties for the leader election and AODV protocols.

Chapter 6: Product Line Process Theory. This chapter briefly explain PL-CCS, and studies different notions of behavioral equivalence for PL-CCS. These notions enable reasoning about the behavior of SPLs at different levels of abstraction. It discusses the compositionality property and the mutual relationship among these notions. It further shows how the strengths of these notions can be consolidated in an equational reasoning method. Finally, it formulates notions of behavioral equivalence that are characterized by the property specification language for PL-CCS, called multi-valued modal μ-calculus.

Chapter 7: Concluding Remarks. This chapter summarizes the main results of this thesis and discusses future work.

Together with Chapter 2, all the Chapters 3-6 can be read separately; each

includes its own specific background, related work, and conclusion.

1.7 Origins of the Chapters

Chapter 2 is based on the journal papers [73, 74], written together with Wan Fokkink and Ali Movaghar. These journal papers are the extension of the conference papers [71, 72], respectively.

Chapter 3 is composed of the journal paper [69] together with Sections 3, 4, and subsections 7.1.1 and 7.1.3 of the journal paper [70], both written together with Wan Fokkink.

Chapter 4 is based on the journal paper [156], written together with Behnaz Yousefi and Ramtin Khosravi.

Chapter 5 is composed of Sections 5, 6 and subsections 7.1.2 of the journal paper [70].

Chapter 6 is based on the journal paper [75], written together with Mohammad Reza Mousavi.

Preliminaries 2

In this chapter we introduce our proposed semantic model, Constrained Labeled Transition Systems (CLTSs), suitable for modeling the behavior of MANETs. Our semantic model is achieved by enriching the labels of Labeled Transition Systems (LTSs), the classical semantic model of systems, with network constraints. Network constraints specify restrictions over the underlying topology of the network for which the behavior, i.e., the transition, is valid. We explain how the enriched semantic model can abstractly address mobility and topology-dependent behavior of MANETs. Next, we introduce Computed Network Process Theory (*CNT*), the framework suitable for modeling and verification of MANETs in an algebraic approach, using the notion of branching computed network bisimilarity. We provide a sound and complete axiomatization for networks with finite-state behaviors. Furthermore, we show how by equational theory one can reason about MANETs consisting of a finite but unbounded set of nodes, in which all nodes deploy the same protocol. At the end we briefly explain the Actors, a model of concurrent computation for developing parallel, distributed and mobile systems, and then Rebeca, an actor-based modeling language.

2.1 Labeled Transition Systems and Semantic Equivalence Relations

Labeled transition systems are the classical semantic models used to formally model hardware and software systems. A labeled transition system (LTS) is defined by the quadruple $\langle S, \rightarrow, L, s_0 \rangle$ where S is a set of states, $\rightarrow \subseteq S \times L \times S$ a set of transitions, L a set of labels, and s_0 the initial state. Let $s \xrightarrow{\alpha} t$ denote $(s, \alpha, t) \in \rightarrow$. Many equivalence relations are defined over LTSs to reason about the behaviors of two systems [147]. Strong bisimulation [118] is the finest relation which expresses that two LTSs are equivalent if they produce the same set of actions (observable behavior) and have the same branching structure

Definition 2.1 (Strong Bisimilarity)**.** A binary relation $\mathcal{R} \subseteq S \times S$ is called a strong bisimulation if and only if, for any $s_1,\ s_1',\ s_2$, and s_2' and $\alpha \in L$, the following transfer conditions hold:

- $s_1 \mathcal{R} s_2 \wedge s_1 \xrightarrow{\alpha} s_1' \Rightarrow (\exists s_2' \in S : s_2 \xrightarrow{\alpha} s_2' \wedge s_1' \mathcal{R} s_2')$,
- $s_1 \mathcal{R} s_2 \wedge s_2 \xrightarrow{\alpha} s_2' \Rightarrow (\exists s_1' \in S : s_1 \xrightarrow{\alpha} s_1' \wedge s_1' \mathcal{R} s_2')$.

Two states s and t are called strongly bisimilar, denoted by $s \sim t$, if and only if there is a strong bisimulation relating s and t.

To reason about systems whose internal behaviors have been abstracted by turning such behaviors into un-observable τ-transitions, some coarser relations are used, which ignore such transitions. Branching bisimilarity [76] is the finest relation. Let $\xrightarrow{\tau}{}^*$ be reflexive and transitive closure of τ-transitions:

- $t \xrightarrow{\tau}{}^* t$;
- $t \xrightarrow{\tau}{}^* s$, and $s \xrightarrow{\tau} r$, then $t \xrightarrow{\tau}{}^* r$.

Definition 2.2 (Branching Bisimilarity)**.** A binary relation $\mathcal{R} \subseteq S \times S$ is called a branching bisimulation if and only if, for any $s_1,\ s_1',\ s_2$, and s_2' and $\alpha \in L$, the following transfer conditions hold:

- $s_1 \mathcal{R} s_2 \wedge s_1 \xrightarrow{\alpha} s_1' \Rightarrow ((\alpha = \tau \wedge s_1' \mathcal{R} s_2) \vee (\exists s_2', s_2'' \in S : s_2 \xrightarrow{\tau}{}^* s_2'' \xrightarrow{\alpha} s_2' \wedge s_1 \mathcal{R} s_2'' \wedge s_1' \mathcal{R} s_2'))$,
- $s_1 \mathcal{R} s_2 \wedge s_2 \xrightarrow{\alpha} s_2' \Rightarrow ((\alpha = \tau \wedge s_1 \mathcal{R} s_2') \vee (\exists s_1', s_1'' \in S : s_1 \xrightarrow{\tau}{}^* s_1'' \xrightarrow{\alpha} s_1' \wedge s_1'' \mathcal{R} s_2 \wedge s_1' \mathcal{R} s_2'))$.

Two states s and t are called branching bisimilar, denoted by $s \simeq_{br} t$, if and only if there exists a branching bisimulation relating s and t.

2.2 Semantic Model: Constrained Labeled Transition Systems

Communication in wireless networks tends to be based on local (also called restricted) broadcast: Only nodes that are located in the transmission area of a sender can receive messages from this sender. A node B is *directly connected to* a node A if B is located within the transmission range of A. This asymmetric connectivity relation between nodes introduces a topology concept. A *topology* is a function $\gamma : Loc \to I\!P(Loc)$ with $\forall \ell \in Loc\,(\ell \notin \gamma(\ell))$, where Loc denotes a finite set of (hardware) addresses A, B, C ranged over by ℓ. The set Loc is extended with the unknown address $?$ to represent the address of a node which is still not known or concealed from an external observer. For instance, the leader address of a node can be initialized to this value. Furthermore, to define the semantics of communicating nodes in terms of restrictions over the topology in a compositional way, the semantics of receive actions can be defined through an unknown sender, which will be replaced by a known address when the receive actions are composed with the corresponding send action at a specific node (see Section 2.3.2).

*Constrained labeled transition system*s (CLTSs) provide a semantic model for the operational behavior of MANETs. A transition label is a pair of an action and a *network constraint*, restricting the range of possible underlying topologies. A network constraint $\mathcal{C}$ is a set of connectivity pairs $\rightsquigarrow: Loc \times Loc$. For instance, $B \rightsquigarrow A$ denotes that A is connected to B directly and consequently A can receive data sent by B (note that the direction of $\rightsquigarrow$ implies the direction of data flow), while $? \rightsquigarrow A$ denotes that A should be in the range of a node with an unknown address to receive data. We write $\{B \rightsquigarrow A, C\}$ instead of $\{B \rightsquigarrow A,\ B \rightsquigarrow C\}$. This constraint accompanies the send action of B. Exclusion of $B \rightsquigarrow D$ from $\{B \rightsquigarrow A, C\}$ implies that (1) D is not connected to B, or (2) it was connected but missed the sent data of B due to noises in the environment, or (3) we currently don't know anything about D due to its failure and thus its link is of no importance. We remark that the relation $\rightsquigarrow$ need not be reflexive, symmetric, and transitive.

Let $\mathbb{C}(Loc)$ denote the set of network constraints that can be defined over the network addresses in Loc. A network constraint $\mathcal{C}$ is said to be *well-formed* if $\forall \ell \in Loc\ (\ell \rightsquigarrow? \notin \mathcal{C})$, because the receivers should be always known. Let $\mathbb{C}^v(Loc) \subseteq \mathbb{C}(Loc)$ denote the set of well-formed network constraints that can be defined over the network addresses in Loc. Each well-formed network constraint $\mathcal{C}$ represents the set of network topologies that satisfy the connectivity pairs in $\mathcal{C}$, i.e., $\Gamma(\mathcal{C}) = \{\gamma \mid \mathcal{C} \subseteq \mathcal{C}^+_\Gamma(\gamma)\}$, where $\mathcal{C}^+_\Gamma(\gamma) = \{\ell \rightsquigarrow \ell' \mid \ell' \in \gamma(\ell)\}$ extracts all one-hop connectivity information from γ. So the empty network constraint $\{\}$ denotes all possible topologies over Loc.

Let Msg denote a set of messages communicated over a network and ranged over by $\mathfrak{m}$. Let Act be the network send and receive actions with signatures $nsnd : Msg \times Loc$ and $nrcv : Msg$, respectively. The send action $nsnd(\mathfrak{m}, \ell)$ denotes that the message $\mathfrak{m}$ is transmitted from a node with the address ℓ, while the receive action $nrcv(\mathfrak{m})$ denotes that the message $\mathfrak{m}$ is ready to be received. Let $Act_\tau = Act \cup \{\tau\}$, ranged over by η.

Definition 2.3. A *CLTS* is defined by a tuple $\langle S, \Lambda, \rightarrow, s_0 \rangle$, with S a set of states, $\Lambda \subseteq \mathbb{C}^v(Loc) \times Act_\tau$, $\rightarrow \subseteq S \times \Lambda \times S$ a transition relation, and $s_0 \in S$ the initial state. A transition $(s, (\mathcal{C}, \eta), s') \in \rightarrow$ is denoted by $s \xrightarrow{(\mathcal{C},\eta)} s'$.

Generally speaking, the transition $s \xrightarrow{(\mathcal{C},\eta)} s'$ expresses that a MANET protocol in state s with an underlying topology $\gamma \in \Gamma(\mathcal{C})$ can perform action η to evolve to state s'.

2.2.1 Unfolding a CLTS into an LTS

Mobility of nodes is modeled implicitly by a CLTS. For example, consider the CLTS in left of the Fig. 2.1. In state s_0, A sends a req message; in case B is connected to A and their communication is successful, there is a transition to state s_1, otherwise to state s_2. In other words, B may be disconnected from A, or connected to A, but B has not successfully received the message. Thus, s_1 represents that

a message has been received by B. In this state, B sends a rep message; in case A is connected to B and their communication is successful, there is a transition to state s_0, otherwise to state s_2, which is a deadlock state. In each state, the underlying topology can implicitly change, and a behavior (i.e., transition) is only possible if the underlying topology belongs to the network constraint that the behavior under consideration is restricted to. Thus the CLTS models the behavior of the MANET compactly while mobility is modeled implicitly.

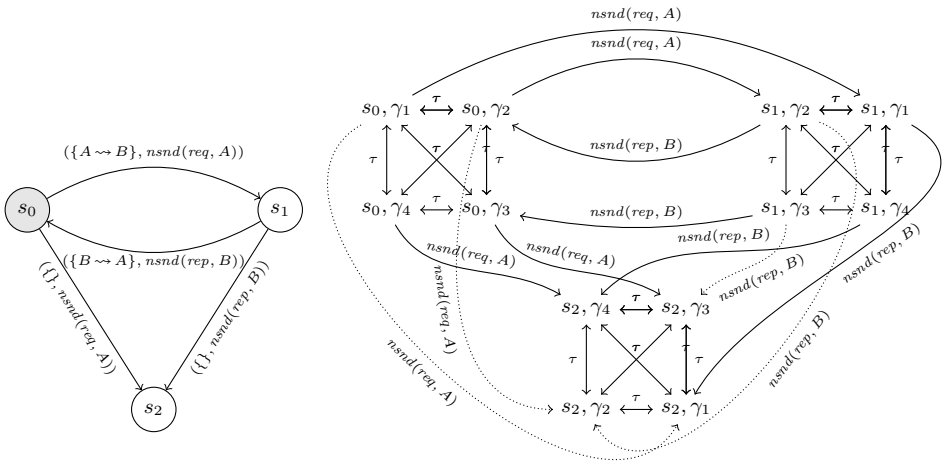


Figure 2.1: A CLTS and its unfolded LTS. The τ-transitions model arbitrary mobility of nodes. The dotted transitions distinguish unsuccessful communications despite of node connectivity.

The given CLTS is unfolded into an LTS, whereby the network constrains are omitted and the underlying topology is made explicit in each state. Transformations of the topology are modeled by means of τ-transitions. We present the unfolding of a CLTS into an LTS here, to clarify the meaning of CLTSs, and to illustrate how the unfolding grows as the number of nodes in the MANET increases.

Let $Loc = \{A, B\}$, so that the possible topologies are $\gamma_1 = \{A \mapsto \{B\}, B \mapsto \emptyset\}$, $\gamma_2 = \{A \mapsto \{B\}, B \mapsto \{A\}\}$, $\gamma_3 = \{A \mapsto \emptyset, B \mapsto \{A\}\}$, and $\gamma_4 = \{A \mapsto \emptyset, B \mapsto \emptyset\}$. The three states s_0, s_1, s_2 are paired with the four possible topologies $\gamma_1, \gamma_2, \gamma_3, \gamma_4$, leading to twelve states in total, as shown in right of the Fig. 2.1. Formally speaking, the given CLTS $\langle S, \Lambda, \rightarrow, s_0 \rangle$, where $\Lambda \subseteq \mathbb{C}^v(Loc) \times Act_\tau$ and $\rightarrow \subseteq S \times \Lambda \times S$, is unfolded into the LTS $\langle S \times \Gamma, Act_\tau, \rightarrow, s_0 \times \Gamma \rangle$, where Γ is the set of possible topologies over Loc and $\rightarrow \subseteq S \times \Gamma \times Act_\tau \times S \times \Gamma$ is the set of

transitions such that $s, \gamma \xrightarrow{\eta} s', \gamma$ if $s \xrightarrow{\mathcal{C},\eta} s'$ with $\gamma \in \Gamma(\mathcal{C})$, and $s, \gamma \xrightarrow{\tau} s, \gamma'$ for all $\gamma' \in \Gamma \setminus \{\gamma\}$. As illustrated, the number of states and transitions grows exponentially as the number of nodes increases, since the number of topologies is exponential in the number of nodes.

2.3 Computed Network Process Theory

Computed Network Process Theory (*CNT*) is an extension of *Restricted Broadcast Process Theory* (*RBPT*) [71] with so-called computed network terms and auxiliary operators. It provides a sound and complete axiom system, modulo so-called rooted (branching computed) network bisimilarity.

Network protocols (in particular MANET protocols) rely on data. To separate the manipulation of data from processes, we make use of equational abstract data types [55]. Data is specified by equational specifications: one can declare data types (so-called *sorts*) and functions working upon these data types, and describe the meaning of these functions by equational axioms. Following the approach of [84, 85], we consider *CNT* with equational abstract data types. Consequently, the set of addresses Loc, messages Msg, and network constraints $\mathbb{C}$ can be treated as data sorts in *CNT* framework with well-defined functions over them.

The semantics of the data part (of a specification), denoted by $\mathbb{D}$, is defined in the same way as in [85]. It should contain the $Bool$ domain with distinct T and F constants, Loc, Msg, and $\mathbb{C}$ domains.

We first explain the set of data types considered in our framework, and then define the *CNT* operators and their axioms.

2.3.1 Data Types

To provide the axioms of computed network terms, we define appropriate functions over the sorts Loc, Msg, and $\mathbb{C}$. We use μCRL notation to define data types: *sort* declares sort names, *func* specifies constructor and *map* non-constructor functions, *var* declares variable names, and *rew* defines the behavior of non-constructor functions by means of rewrite rules. We assume that the function $if : Bool \times D \times D \to D$ is defined for all data sorts D, which returns the second parameter if the boolean parameter equals true, otherwise the third parameter is returned.

The data sort $Bool$ is used in the conditional operator construct to allow data values to influence the behavior of a process. This data sort is defined by two constructors T and F. The conventional operators $\wedge$, $\vee$ and $\neg$ can be defined over it straightforwardly. The data sort Nat specifies the natural numbers by two constructors functions: the constant 0 and the unary function $succ$. We use $1, 2, \ldots$ for $succ(0), succ(succ(0)), \ldots$. The definitions of the non-constructor functions $+$, $>$, $\geq$ and eq are straightforward.

$sort$	Msg	$sort$	Loc
$func$	$req : Loc \to Msg$	$func$	$? :\to Loc$
	$rep : Loc \times Loc \to Msg$		$adr : Loc \to Loc$
map	$isType_{req} : Msg \to Bool$	map	$eq : Loc \times Loc \to Bool$
	$eq : Msg \times Msg \to Bool$		$>: Loc \times Loc \to Bool$
var	$\ell, \ell_1, \ell_2, \ell_3, \ell_4 : Loc$		
rew	$eq(req(\ell_1), req(\ell_2)) = eq(\ell_1, \ell_2)$	$sort$	$\mathbb{C}$
	$eq(rep(\ell_1, \ell_2), rep(\ell_3, \ell_4)) =$	$func$	$empNC :\to \mathbb{C}$
	$\quad eq(\ell_1, \ell_3) \wedge eq(\ell_2, \ell_4)$		$con : Loc \times Loc \times \mathbb{C} \to \mathbb{C}$
	$eq(req(\ell_1), rep(\ell_2, \ell_3)) = F$		
	$eq(rep(\ell_1, \ell_2), req(\ell_3)) = F$	map	$union : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$
	$isType_{req}(req(\ell)) = T$		$subs : Loc \times Loc \times \mathbb{C} \to \mathbb{C}$
	$isType_{req}(rep(\ell_1, \ell_2)) = F$		$include : \mathbb{C} \times \mathbb{C} \to Bool$

Figure 2.2: Data sorts used in the *CNT* framework.

The data sort definitions of Loc, Msg, and $\mathbb{C}$ are given in Fig. 2.2. The network addresses are generated from the constant $?$ and the unary function adr. We use $A, B, \ldots$ to denote $adr(?), adr(adr(?)), \ldots$. The functions eq and $>$ compare two network addresses. The network constraints are generated from the constant $empNC$ and the con function which adds a connectivity pair to network constraints. The function $union$ merges two network constraints such that the redundant connections are removed and the connectivity pairs are sorted in terms of the connected addresses (i.e. the second parameter in con). The function $subs$ substitutes the address in its first parameter for all occurrences of the address in its second parameter. The function $include$ examines if the connectivity pairs of a network constraint are included in another. We write $\mathcal{C}_1 \cup \mathcal{C}_2$, $\mathcal{C}_1 \subseteq \mathcal{C}_2$ and $\mathcal{C}[\ell/\ell']$ instead of $union(\mathcal{C}_1, \mathcal{C}_2)$, $include(\mathcal{C}_1, \mathcal{C}_2)$ and $subs(\mathcal{C}, \ell, \ell')$, respectively. We also write $\{\}$, $\{A \rightsquigarrow B\}$, $\{A \rightsquigarrow B, C\}$ for $empNC$, $con(A, B, empNC)$, $con(A, B, con(A, C, empNC))$.

A message can carry data parameters. For instance, in Fig. 2.2, the message $req : Loc \to Msg$ has one parameter of type Loc. The function eq compares two messages. For each message name m defined in Msg, a function $isType_m : Mag \to Bool$ is defined which examines if a message term is constructed by the message name m.

2.3.2 *CNT* Syntax and Semantics

Let D denote a data sort; u, v and d range over closed and open data terms of sort D, respectively. We consider b is of type $Bool$. Let $d[d_1/d_2]$ denote substitution of d_2 by d_1 in the data term d; this can be extended to computed network terms. Let $\mathcal{A}$ denote a countably infinite set of process names which are used as recursion variables in recursive specifications. This set can be split into two disjoint subsets $\mathcal{A}_p$ and $\mathcal{A}_n$ to denote process names for protocols and networks, respectively.

Without loss of generality we assume that process names and messages have exactly one parameter. The syntax of *CNT* is:

$$t ::= 0 \mid \beta.t \mid t + t \mid [b]t \diamond t \mid \sum_{d:D} t \mid \mathfrak{A}(d)\,, \mathfrak{A}(d:D) \stackrel{def}{=} t \mid [\![t]\!]_\ell \mid t \mid t \mid t \between t \mid t \parallel t \mid (\nu\ell)t \mid \tau_m(t) \mid \partial_m(t)$$

0 defines a deadlock process. The prefix operator in $\beta.t$ denotes a process which performs β and then behaves as t. The action β can be of two types:

- $rcv(\mathfrak{m})$ and $snd(\mathfrak{m})$ actions, denoted by α, which model protocol receive and send actions respectively. They model the interaction of a protocol with its underlying *MAC* layer;
- $(\mathcal{C}, nrcv(\mathfrak{m}))$, $(\mathcal{C}, nsnd(\mathfrak{m}, \ell))$ and $(\mathcal{C}, \tau)$ actions, denoted by $(\mathcal{C}, \eta)$, where the first two actions are called the network receive and send actions respectively. They model the interaction of multiple *MAC* layers in a MANET. An action $(\mathcal{C}, \eta)$ represents the behavior η for the set of topologies specified by $\mathcal{C}$.

The process $t_1 + t_2$ behaves non-deterministically as t_1 or t_2. The conditional construct $[b]t_1 \diamond t_2$ behaves as t_1 when $\mathbb{D} \models b = T$ and as t_2 when $\mathbb{D} \models b = F$. The summation $\sum_{d:D} t$, which binds the name d to t, defines a non-deterministic choice among $t[u/d]$ for all closed $u \in D$. A process name is declared by $\mathfrak{A}(d : D) \stackrel{def}{=} t$, where $\mathfrak{A} \in \mathcal{A}$, and d is a variable name that may appear free in t, meaning that it is not within the scope of a sum operator in t. Computed network terms are considered modulo α-conversion of bound names. The function fn, which returns the set of free names, is defined over computed network terms as usual. A term is closed if the set of its free names is empty. The deployment of a process t at a network address $\ell \neq ?$ is specified as $[\![t]\!]_\ell$, which defines a single-node MANET. The parallel composition $t_1 \parallel t_2$ defines two MANETs that communicate by local broadcast; if there is a connectivity between nodes of t_1 and t_2 they may communicate, or the send/receive actions of t_1 and t_2 are interleaved. *CNT* borrows from the process algebra *ACP* [21] the operators *left merge* ($\between$) and *communication merge* ($|$) to axiomatize parallel composition. Hiding $(\nu\ell)t$ conceals the activities of a node with the address ℓ by renaming this address to $?$ in network send/receive actions. For each message type $m : D \to Msg$, the unary operators τ_m and ∂_m are defined; *Abstraction* $\tau_m(t)$ renames network send/receive actions over messages of type m to τ, and *encapsulation* $\partial_m(t)$ forbids receiving messages of type m and renames them to 0. We use $\tau_{\{m_1,\ldots,m_n\}}(t)$ and $\partial_{\{m_1,\ldots,m_n\}}(t)$ to denote $\tau_{m_1}(\ldots(\tau_{m_n}(t))\ldots)$ and $\partial_{m_1}(\ldots(\partial_{m_n}(t))\ldots)$ respectively. We will use MANET, network and computed network term interchangeably.

Intuitively a computed network term is grammatically well-defined if processes deployed at a network address, called protocols, are defined by protocol action prefix, choice, summation, conditional, deadlock operators and process names. We say t' occurs in the *context* of an operator if t' is a subterm of term

t which is the operand of that operator. Formally, a computed network term t is grammatically well-defined if the following conditions are met:

- If $t \equiv [\![t']\!]_\ell$, then t' has no network prefix action $(\mathcal{C}, \eta)$, deployment $[\![]\!]$, parallel $\|$, left merge $\between$, communication merge $|$, hiding $(\nu\ell)$, abstraction τ_m, encapsulation ∂_m, and process name $\mathfrak{A}(d)$ such that $\mathfrak{A} \in \mathcal{A}_n$.
- If $t \equiv rcv(m(d)).t'$, then it should be immediately preceded by a summation like $\sum_{d:D}$, where $m : D \to Msg$.
- If $t \equiv \alpha.t'$, then it should be in the context of a deployment operator. This rule prevents terms like $snd(m(d)).[\![t]\!]$ or $snd(m(d).0 \parallel 0$.
- If $t \equiv \mathfrak{A}(d)$ where $\mathfrak{A} \in \mathcal{A}_p$, then it should be in the context of a deployment operator. Furthermore it should be defined by an equation like $\mathfrak{A}(d : D) \stackrel{def}{=} t'$ such that t' has no network prefix action $(\mathcal{C}, \eta)$, deployment $[\![]\!]$, parallel $\|$, left merge $\between$, communication merge $|$, hiding $(\nu\ell)$, abstraction τ_m, encapsulation ∂_m, and process name $\mathfrak{B}(d)$ such that $\mathfrak{B} \in \mathcal{A}_n$. Moreover, each occurrence of $\mathfrak{A}$ should be in the context of an α prefix action in t'.
- If $t \equiv \mathfrak{A}(d)$ where $\mathfrak{A} \in \mathcal{A}_n$, then it should not be in the context of a deployment operator. Furthermore it should be defined by an equation like $\mathfrak{A}(d : D) \stackrel{def}{=} t'$ such that t' is well-defined.

The first, third, and forth rules ensure that protocols are only defined by protocol action prefix, choice, summation, conditional, deadlock operators and process names. From now on we will only consider computer network terms that are well-defined. For example, $[\![X(A)]\!]_A \parallel [\![Y(B)]\!]_B$ is a well-defined computed network term, where $X(adr : Loc) \stackrel{def}{=} snd(req(adr)).X(adr)$ and $Y(adr : Loc) \stackrel{def}{=} \sum_{lx:Loc} rcv(req(lx)).snd(rep(adr, lx)).Y(adr)$. The process name X defines a protocol that sends req messages iteratively, while Y receives a req and then sends a rep message.

The semantics of *CNT* is defined by using structural operational semantic (SOS) rules, following the approach of [126]. Given some data model $\mathbb{ID}$ and a MANET, the SOS rules in Table 2.1 induce a CLTS with transitions of the form $t \xrightarrow{(\mathcal{C},\eta)} t'$, where η denotes actions of the form $\{nrcv(\mathfrak{m}), nsnd(\mathfrak{m}, \ell)\} \cup \{\tau\}$. Assume that $\alpha \in \{rcv(\mathfrak{m}), snd(\mathfrak{m})\}$, and β denotes either α or $(\mathcal{C}, \eta)$. We remark that the symmetric counterparts of the rules $Choice$, Bro, Par, and $Sync_2$ are also present, but have been omitted here for brevity.

$Prefix$ indicates execution of a prefix action. The non-deterministic behavior of the choice operator is specified by $Choice$ in terms of its operands. The behavior of the sum operator is defined by substituting any closed term in D for d, as explained by the rule Sum. A process behaves as t_1 if the condition evaluates

Table 2.1: Semantics of *CNT* operators

$$\frac{}{\beta.t \xrightarrow{\beta} t} : \mathit{Prefix} \quad \frac{t_1 \xrightarrow{\beta} t_1'}{t_1 + t_2 \xrightarrow{\beta} t_1'} : \mathit{Choice} \quad \frac{t[u/d] \xrightarrow{\beta} t'}{\sum_{d:D} t \xrightarrow{\beta} t'} : \mathit{Sum},\ u \in D$$

$$\frac{t_1 \xrightarrow{\beta} t_1'}{[b]t_1 \diamond t_2 \xrightarrow{\beta} t_1'} : \mathit{Then},\ \mathbb{D} \models b = T \quad \frac{t_2 \xrightarrow{\beta} t_2'}{[b]t_1 \diamond t_2 \xrightarrow{\beta} t_2'} : \mathit{Else},\ \mathbb{D} \models b = F$$

$$\frac{t \xrightarrow{snd(\mathfrak{m})} t'}{[\![t]\!]_\ell \xrightarrow{(\{\},nsnd(\mathfrak{m},\ell))} [\![t']\!]_\ell} : \mathit{Inter}_1 \quad \frac{t \xrightarrow{rcv(\mathfrak{m})} t'}{[\![t]\!]_\ell \xrightarrow{(\{?\rightsquigarrow\ell\},nrcv(\mathfrak{m}))} [\![t']\!]_\ell} : \mathit{Inter}_2$$

$$\frac{t_1 \xrightarrow{(\mathcal{C}_1,\ nsnd(\mathfrak{m},\ell))} t_1' \quad t_2 \xrightarrow{(\mathcal{C}_2,\ nrcv(\mathfrak{m}))} t_2'}{t_1 \parallel t_2 \xrightarrow{(\mathcal{C}_1 \cup \mathcal{C}_2[\ell/?],\ nsnd(\mathfrak{m},\ell))} t_1' \parallel t_2'} : \mathit{Bro}$$

$$\frac{t_1 \xrightarrow{(\mathcal{C}_1,nrcv(\mathfrak{m}))} t_1' \quad t_2 \xrightarrow{(\mathcal{C}_2,nrcv(\mathfrak{m}))} t_2'}{t_1 \parallel t_2 \xrightarrow{(\mathcal{C}_1 \cup \mathcal{C}_2,nrcv(\mathfrak{m}))} t_1' \parallel t_2'} : \mathit{Recv}$$

$$\frac{t_1 \xrightarrow{\beta} t_1'}{t_1 \parallel t_2 \xrightarrow{\beta} t_1' \parallel t_2} : \mathit{Par} \quad \frac{t \xrightarrow{(\mathcal{C},\eta)} t'}{t \xrightarrow{(\mathcal{C}',\eta)} t'} :\ \mathit{Exe},\ \mathcal{C} \subseteq \mathcal{C}' \quad \frac{t \xrightarrow{\beta} t'}{(\nu\ell)t \xrightarrow{\beta[?/\ell]} (\nu\ell)t'} : \mathit{Hid}$$

$$\frac{t_1 \xrightarrow{(\mathcal{C}_1,nrcv(\mathfrak{m}))} t_1' \quad t_2 \xrightarrow{(\mathcal{C}_2,nrcv(\mathfrak{m}))} t_2'}{t_1 \mid t_2 \xrightarrow{(\mathcal{C}_1 \cup \mathcal{C}_2,nrcv(\mathfrak{m}))} t_1' \parallel t_2'} : \mathit{Sync}_1$$

$$\frac{t[u/d] \xrightarrow{\beta} t'}{\mathfrak{A}(u) \xrightarrow{\beta} t'} :\ \mathit{Inv},\ \mathfrak{A}(d:D) \stackrel{def}{=} t \quad \frac{t_1 \xrightarrow{(\mathcal{C},\eta)} t_1'}{t_1 \between t_2 \xrightarrow{(\mathcal{C},\eta)} t_1' \parallel t_2} :\ \mathit{LExe}$$

$$\frac{t_1 \xrightarrow{(\mathcal{C}_1,nsnd(\mathfrak{m},\ell))} t_1' \quad t_2 \xrightarrow{(\mathcal{C}_2,nrcv(\mathfrak{m}))} t_2'}{t_1 \mid t_2 \xrightarrow{(\mathcal{C}_1 \cup \mathcal{C}_2[\ell/?],nsnd(\mathfrak{m},\ell))} t_1' \parallel t_2'} : \mathit{Sync}_2$$

$$\frac{t \xrightarrow{(\mathcal{C},\eta)} t'}{\partial_m(t) \xrightarrow{(\mathcal{C},nrcv(\mathfrak{m}))} \partial_m(t')} : \mathit{Encap},\ \eta \neq nrcv(\mathfrak{m}) \vee \mathit{isType}_m(\mathfrak{m}) = F$$

$$\frac{t \xrightarrow{(\mathcal{C},\eta)} t'}{\tau_m(t) \xrightarrow{(\mathcal{C},\eta)} \tau_m(t')} : \mathit{Abs}_1,\ \eta \notin \{nrcv(\mathfrak{m}, nsnd(\mathfrak{m},\ell))\} \vee \mathit{isType}_m(\mathfrak{m}) = F$$

$$\frac{t \xrightarrow{(\mathcal{C},\eta)} t'}{\tau_m(t) \xrightarrow{(\mathcal{C},\tau)} \tau_m(t')} : \mathit{Abs}_2,\ \eta \in \{nrcv(\mathfrak{m}, nsnd(\mathfrak{m},\ell))\} \wedge \mathit{isType}_m(\mathfrak{m}) = T$$

to true ($Then$), otherwise it behaves as t_2 ($Else$). Inv defines the behavior of a process name in terms of the right-hand side of its definition $\mathfrak{A}(d : D) \stackrel{def}{=} t$.

$Inter_1$ denotes that a single node can perform the send actions of a protocol at this node under any valid topology, and its network address is appended to this action. $Inter_2$ denotes a single node performing a receive action, under the restriction that the node must be connected to some sender (denoted by $?$). These two rules convert protocol actions to their corresponding network actions by abstracting away from the service of the data link layer.

Rule Bro specifies how a communication occurs between a receiving and a sending process. This rule combines the network constraints, while the unknown location (in the network constraint of the receiving process) is set to the concrete address of the sender. Rule $Recv$ synchronizes the receive actions of processes t_1 and t_2 on message $\mathfrak{m}$, while combining together their (dis)connectivity information in network constraints $\mathcal{C}_1$ and $\mathcal{C}_2$. In Bro and $Recv$ it is required that the union of network constraints on the transition in the conclusion be well-formed. By Par, a process evolves when a subprocess evolves. Exe explains that a behavior that is possible for a network constraint, is also possible for a more restrictive network constraint. As the communication merge operator defines a successful synchronization between two computed networks, its behavior is defined by $Sync_1$ and $Sync_2$, indicating synchronization between receive actions or a send and receive action, respectively. $LExe$ defines that in a term composed by the left merge, the left computed network succeeds to perform the initial action, and then the resulting term proceeds as in parallel composition.

Rule Hid replaces every occurrence of ℓ in the network constraint and action of β by $?$, and hence hides activities of a node with the address ℓ from external observers. The encapsulation operator ∂_m disallows all network receive actions on messages of type m, as specified by $Encap$. According to $Abs_{1,2}$, the abstraction operator τ_m converts all network send and receive actions with a message of type m to τ and leaves other actions unaffected.

2.3.3 Rooted Branching Computed Network Bisimilarity

Computed network terms are considered modulo rooted branching computed network bisimilarity [73]. To define this equivalence relation, we introduce the following notations:

- $\Rightarrow$ denotes the reflexive and transitive closure of unobservable actions:
  - $t \Rightarrow t$;
  - if $t \xrightarrow{(\mathcal{C},\tau)} t'$ for some arbitrary network constraint $\mathcal{C}$ and $t' \Rightarrow t''$, then $t \Rightarrow t''$.
- $t \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t'$ iff $t \xrightarrow{(\mathcal{C},\eta)} t'$ or $t \xrightarrow{(\mathcal{C}[\ell/?],\eta[\ell/?])} t'$ and η is of the form $nsnd(\mathfrak{m}, ?)$ for some $\mathfrak{m}$.

Intuitively $t \Rightarrow t'$ expresses that after a number of topology changes, t can behave like t'. Furthermore, an action like $(\{? \rightsquigarrow B\}, nsnd(req(?), ?))$ can be matched to an action like $(\{A \rightsquigarrow B\}, nsnd(req(A), A))$, which is its $\langle - \rangle$ counterpart.

Definition 2.4. A binary relation $\mathcal{R}$ on computed network terms is a branching computed network simulation if $t_1 \mathcal{R} t_2$ and $t_1 \xrightarrow{(\mathcal{C},\eta)} t_1'$ implies that either:

- η is of the form $nrcv(\mathfrak{m})$ or τ, and $t_1' \mathcal{R} t_2$; or
- there are t_2' and t_2'' such that $t_2 \Rightarrow t_2'' \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t_2'$, where $t_1 \mathcal{R} t_2''$ and $t_1' \mathcal{R} t_2'$.

$\mathcal{R}$ is a branching computed network bisimulation if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are branching computed network simulations. Two terms t_1 and t_2 are branching computed network bisimilar, denoted by $t_1 \simeq_b t_2$, if $t_1 \mathcal{R} t_2$ for some branching computed network bisimulation relation $\mathcal{R}$.

Definition 2.5. Two terms t_1 and t are *rooted branching computed network bisimilar*, written $t_1 \simeq_{rb} t_2$, if:

- $t_1 \xrightarrow{(\mathcal{C},\eta)} t_1'$ implies there is a t_2' such that $t_2 \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t_2'$ and $t_1' \simeq_b t_2'$;
- $t_2 \xrightarrow{(\mathcal{C},\eta)} t_2'$ implies there is a t_1' such that $t_1 \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t_1'$ and $t_1' \simeq_b t_2'$.

Rooted branching computed network bisimilarity is an equivalence relation and constitutes a congruence with respect to the *CNT* operators; see [73]. Intuitively two computed network terms are equivalent if they send and receive the same set of messages for a set of topologies. However a receiving action which does not change the sending behavior of a node can be removed. Therefore, an only receiving MANET (after its first action) is equivalent to deadlock. It should be noted that a node like $[\![Y(B)]\!]_B$ is not branching bisimilar to the sending node $[\![Y'(B)]\!]_B$ where $Y'(adr : Loc) \stackrel{def}{=} \sum_{lx:Loc} snd(rep(adr, lx)).Y'(adr)$, since the latter sends iff it receives a request message while the former always sends.

2.3.4 Axioms

We define the behavior of operators through their axioms over closed terms, which are sound with respect to rooted branching computed network bisimilarity. The axioms of the choice, conditional and summation operator are given in Table 2.2. The axioms Ch_{1-4}, Con_{1-2} and Sum_{1-4} are standard (cf. [111]). The axiom Ch_5 is new in our framework, denoting that a network send action that originates from a node of which the address is unknown can be removed if there is a same action originating from a node with a known address. The axiom Ch_6 explains that a smaller set of network constraints allows more behavior.

Axioms for deployment, left and communication merge, and parallel operators are given in Table 2.3. The axioms Dep_{3-5}, Br, LM_{1-4} and $S_{1-3,5}$ are

Table 2.2: Axioms for choice, conditional and summation operators.

Ch_1	$0 + t = t$	Sum_1	$\sum_{d:D} t = t, d \notin fn(t)$
Ch_2	$t_1 + t_2 = t_2 + t_1$	Sum_2	$\sum_{d:D} t = \sum_{e:D} t[e/d]$
Ch_3	$t_1 + (t_2 + t_3) = (t_1 + t_2) + t_3$	Sum_3	$\sum_{d:D} t = \sum_{d:D} t + t[u/d]$
Ch_4	$t + t = t$		
Sum_4	$\sum_{d:D}(t_1 + t_2) = \sum_{d:D} t_1 + \sum_{d:D} t_2$		
Con_1	$[b]t_1 \diamond t_2 = t_1,\ \mathbb{D} \models b = T$	Con_2	$[b]t_1 \diamond t_2 = t_2,\ \mathbb{D} \models b = F$
Ch_5	$(\mathcal{C}, nsnd(\mathfrak{m}, ?)).t + \langle(\mathcal{C}, nsnd(\mathfrak{m}, ?))\rangle.t = \langle(\mathcal{C}, nsnd(\mathfrak{m}, ?))\rangle.t$		
Ch_6	$(\mathcal{C}_1, \eta).t + (\mathcal{C}_2, \eta).t = (\mathcal{C}_1, \eta).t,\ \mathcal{C}_1 \subseteq \mathcal{C}_2$		

standard. Dep_1 expresses that when a protocol sends a message (denoted by snd), the message is sent into the network (denoted by $nsnd$), irrespective of the underlying topology (expressed by $\{\}$). Dep_2 expresses that when a protocol receives a message (denoted by rcv), it should receive it from the network (denoted by $nrcv$) while it is connected to some sender whose address is unknown (expressed by $\{? \rightsquigarrow \ell\}$). It should be noted that Dep_5 preserves the second and third well-definedness rule given in Section 2.3.2.

The axioms $Sync_{1-3}$ explain the synchronization of two MANETs. The sending MANET $(\mathcal{C}_1, nsnd(\mathfrak{m}_1, \ell)).t_1$ can communicate with the receiving MANET $(\mathcal{C}_2, nrcv(\mathfrak{m}_2)).t_2$, if the receiving addresses (denoted by $\mathcal{C}_2$) are also connected to the sender ℓ (denoted by $\mathcal{C}_1 \cup \mathcal{C}_2[\ell/?]$). Likewise two receiving MANETs synchronize on a message when the receiving addresses of both MANETs are connected to the same unknown address (denoted by $\mathcal{C}_1 \cup \mathcal{C}_2$). Two sending MANETs cannot synchronize due to their signal collision. When a MANET is communicating through a τ action, it cannot be synchronized with another MANET, as indicated by axiom S_4.

We return to the example at the end of Section 2.3.2. The behavior of $[\![X(A)]\!]_A \parallel [\![Y(B)]\!]_B$ can be calculated as follows:

$$
\begin{aligned}
&[\![X(A)]\!]_A \parallel [\![Y(B)]\!]_B = \\
&\quad [\![X(A)]\!]_A \between [\![Y(B)]\!]_B + [\![Y(B)]\!]_B \between [\![X(A)]\!]_A + [\![X(A)]\!]_A \mid [\![Y(B)]\!]_B \\
&[\![X(A)]\!]_A = (\{\}, nsnd(req(A), A)).[\![X(A)]\!]_A \\
&[\![Y(B)]\!]_B = \textstyle\sum_{lx:Loc}(\{? \rightsquigarrow B\}, nrcv(req(lx))).[\![snd(rep(B, lx)).Y(B)]\!]_B \\
&[\![X(A)]\!]_A \between [\![Y(B)]\!]_B = (\{\}, nsnd(req(A), A)).[\![X(A)]\!]_A \parallel [\![Y(B)]\!]_B \\
&[\![Y(B)]\!]_B \between [\![X(A)]\!]_A = \\
&\quad \textstyle\sum_{lx:Loc}(\{? \rightsquigarrow B\}, nrcv(req(lx))).[\![snd(rep(B, lx)).Y(B)]\!]_B \parallel [\![X(A)]\!]_A \\
&[\![X(A)]\!]_A \mid [\![Y(B)]\!]_B = \\
&\quad (\{A \rightsquigarrow B\}, nsnd(req(A), A)).[\![X(A)]\!]_A \parallel [\![snd(rep(B, A)).Y(B)]\!]_B
\end{aligned}
$$

The axioms of hiding and encapsulation are given in Table 2.4. The *hiding* operator $(\nu\ell)\_$ conceals the address of a node with the address ℓ from exter-

Table 2.3: Axioms for deployment, left and communication merge, and parallel operators assuming $d \notin fn(t_2)$.

$$
\begin{array}{ll}
Dep_1\ [\![snd(\mathfrak{m}).t]\!]_\ell = (\{\}, nsnd(\mathfrak{m},\ell)).[\![t]\!]_\ell & Dep_4\ [\![0]\!]_\ell = 0 \\
Dep_2\ [\![rcv(\mathfrak{m}).t]\!]_\ell = (\{? \rightsquigarrow \ell\}, nrcv(\mathfrak{m})).[\![t]\!]_\ell & Dep_5\ [\![\sum_{d:D} t]\!]_\ell = \sum_{d:D} [\![t]\!]_\ell \\
Dep_3\ [\![t_1 + t_2]\!]_\ell = [\![t_1]\!]_\ell + [\![t_2]\!]_\ell & \\
Dep_6\ [\![\mathfrak{A}(u)]\!]_\ell = [\![t[u/d]]\!]_\ell,\ \mathfrak{A}(d:D) \stackrel{def}{=} t & \\
 & \\
Br \quad t_1 \parallel t_2 = t_1 \between t_2 + t_2 \between t_1 + t_1 \mid t_2 & S_1\ t_1 \mid t_2 = t_2 \mid t_1 \\
LM_1\ (\mathcal{C},\eta).t_1 \between t_2 = (\mathcal{C},\eta).(t_1 \parallel t_2) & S_2\ (t_1 + t_2) \mid t_3 = t_1 \mid t_3 + t_2 \mid t_3 \\
LM_2\ (t_1 + t_2) \between t_3 = t_1 \between t_3 + t_2 \between t_3 & S_3\ 0 \mid t = 0 \\
LM_3\ 0 \between t = 0 & S_4\ (\mathcal{C},\tau).t_1 \mid t_2 = 0 \\
LM_4\ (\sum_{d:D} t_1) \between t_2 = \sum_{d:D} t_1 \between t_2 & S_5\ (\sum_{d:D} t_1) \mid t_2 = \sum_{d:D} t_1 \mid t_2 \\
 & \\
Sync_1\ (\mathcal{C}_1, nsnd(\mathfrak{m}_1,\ell)).t_1 \mid (\mathcal{C}_2, nrcv(\mathfrak{m}_2)).t_2 = & \\
\qquad [eq(\mathfrak{m}_1,\mathfrak{m}_2)](\mathcal{C}_1 \cup \mathcal{C}_2[\ell/?], nsnd(\mathfrak{m}_1,\ell)).(t_1 \parallel t_2) \diamond 0 & \\
Sync_2\ (\mathcal{C}_1, nrcv(\mathfrak{m}_1)).t_1 \mid (\mathcal{C}_2, nrcv(\mathfrak{m}_2)).t_2 = & \\
\qquad = [eq(\mathfrak{m}_1,\mathfrak{m}_2)](\mathcal{C}_1 \cup \mathcal{C}_2, nrcv(\mathfrak{m}_1)).(t_1 \parallel t_2) \diamond 0 & \\
Sync_3\ (\mathcal{C}_1, nsnd(\mathfrak{m}_1,\ell_1)).t_1 \mid (\mathcal{C}_2, nsnd(\mathfrak{m}_2,\ell_2)).t_2 = 0 &
\end{array}
$$

nal observers. Therefore, the behavior of a hidden node deploying process X, $(\nu C)[\![X(C)]\!]_C$, is $(\{\}, nsnd(req(?),?)).(\nu C)[\![X(C)]\!]_C$.

The axioms $Abs_{1,2}$ rename η actions carrying messages of type m to τ. The axiom Ecp_2 explains that the *encapsulation* operator renames network receive actions of messages of type m to 0. For example,

$$
\begin{array}{l}
\partial_{req}([\![X(A)]\!]_A \parallel [\![Y(B)]\!]_B) = \\
\quad (\{\}, nsnd(req(A),A)).\partial_{req}([\![X(A)]\!]_A \parallel [\![Y(B)]\!]_B) + \\
\quad (\{A \rightsquigarrow B\}, nsnd(req(A),A)).\partial_{req}([\![X(A)]\!]_A \parallel [\![snd(rep(B,A)).Y(B)]\!]_B)
\end{array}
$$

Axiom T_1 removes a receive action that does not affect the behavior of a network, while T_2 removes a τ action which preserves the behavior of a network after some topology changes. The remaining axioms in this table are standard.

Each process name, defined by the equation $\mathfrak{A} \stackrel{def}{=} t$, specifies some specific processes, called solutions. The process term $t_{\mathfrak{A}}$ is a solution of the equation $\mathfrak{A} \stackrel{def}{=} t$ if the replacement of $\mathfrak{A}$ by $t_{\mathfrak{A}}$ on the both sides of the equation results in equal terms, i.e. $t_{\mathfrak{A}} \simeq_{rb} t[t_{\mathfrak{A}}/\mathfrak{A}]$. We are interested in equations with exactly one solution. We define a guardedness criterion for network names to ensure that a network name $\mathfrak{A}$ specified by the equation $\mathfrak{A} \stackrel{def}{=} t$ has a unique solution, denoted by $rec\mathfrak{A} \cdot t$. A free occurrence of a network name A in t is called *guarded* if this occurrence is in the scope of an action prefix operator (not $(\mathcal{C},\tau)$ prefix) and not in the scope of an abstraction operator [9]; in other words, there is a subterm $(\mathcal{C},\eta).t'$ in t such that $\eta \neq \tau$, and $\mathfrak{A}$ occurs in t'. $\mathfrak{A}$ is *(un)guarded* in t if (not) every free occurrence of $\mathfrak{A}$ in t is guarded. A *CNT* term t is *guarded* if for every

Table 2.4: Axiomatization of hiding, abstraction and encapsulation operators.

Res_1	$(\nu\ell)(t_1 + t_2) = (\nu\ell)t_1 + (\nu\ell)t_2$	Res_3	$(\nu\ell)0 = 0$
Res_2	$(\nu\ell)(\mathcal{C}, \eta).t = (\mathcal{C}[?/\ell], \eta[?/\ell]).(\nu\ell)t$	Res_4	$(\nu\ell)\sum_{d:D} t = \sum_{d:D}(\nu\ell)t$
Ecp_1	$\partial_m((\mathcal{C}, nsnd(\mathfrak{m}, \ell)).t) = (\mathcal{C}, nsnd(\mathfrak{m}, \ell)).\partial_m(t)$		
Ecp_2	$\partial_m((\mathcal{C}, nrcv(\mathfrak{m})).t) = [\neg isType_m(\mathfrak{m})](\mathcal{C}, nrcv(\mathfrak{m})).\partial_m(t) \diamond 0$		
Abs_1	$\tau_m((\mathcal{C}, nrcv(\mathfrak{m})).t) = [isType_m(\mathfrak{m})](\mathcal{C}, \tau).\tau_m(t) \diamond (\mathcal{C}, nrcv(\mathfrak{m})).\tau_m(t)$		
Abs_2	$\tau_m((\mathcal{C}, nsnd(\mathfrak{m}, \ell)).t) = [isType_m(\mathfrak{m})](\mathcal{C}, \tau).\tau_m(t) \diamond (\mathcal{C}, nsnd(\mathfrak{m}, \ell)).\tau_m(t)$		
Abs_3	$\tau_m(t_1 + t_2) = \tau_m(t_1) + \tau_m(t_2)$	Ecp_3	$\partial_m(t_1 + t_2) = \partial_m(t_1) + \partial_m(t_2)$
Abs_4	$\tau_m(0) = 0$	Ecp_4	$\partial_m(0) = 0$
Abs_5	$\tau_m(\sum_{d:D} t) = \sum_{d:D} \tau_m(t)$	Ecp_5	$\partial_m(\sum_{d:D} t) = \sum_{d:D} \partial_m(t)$
T_1	$(\mathcal{C}, \eta).((\mathcal{C}', nrcv(\mathfrak{m})).t + t) = (\mathcal{C}, \eta).t$		
T_2	$(\mathcal{C}, \eta).((\mathcal{C}', \tau).(t_1 + t_2) + t_2) = (\mathcal{C}, \eta).(t_1 + t_2)$		

subterm $rec\mathfrak{A} \cdot t'$, $\mathfrak{A}$ is guarded in t'. This guardedness criterion ensures that any guarded recursive term has a unique solution. To understand why $(\mathcal{C}, \tau)$ does not ensure that a recursion has one solution, consider the following example: $rec\mathfrak{A} \cdot (\mathcal{C}, \tau).\mathfrak{A}$ has solutions like $(\mathcal{C}, \tau).0$ and $(\mathcal{C}, \tau).(\mathcal{C}', nsnd(req(A), A)).0$, while they are not rooted branching computed network bisimilar.

Axioms for process names are given in Table 2.5. $Unfold$ and $Fold$ express existence and uniqueness of a solution for the equation $\mathfrak{A} \stackrel{def}{=} t$, which correspond to Milner's standard axioms, and the *Recursive Definition Principle* (*RDP*) and *Recursive Specification Principle* (*RSP*) in *ACP*. $Unfold$ states that each recursive operator has a solution (whether it is guarded or not), while $Fold$ states that each guarded recursive operator has at most one solution. So $[\![X(A)]\!]_A$ and $[\![Y(B)]\!]_B$ can be converted to

$$\begin{aligned}[\![X(A)]\!]_A &= rec\mathfrak{X} \cdot (\{\}, nsnd(req(A), A) \cdot \mathfrak{X}\\ [\![Y(B)]\!]_B &= rec\mathfrak{Y} \cdot \textstyle\sum_{lx:Loc}(\{? \rightsquigarrow A\}, nrcv(req(lx))) \cdot\\ &\qquad (\{\}, nsnd(rep(B, lx), B)) \cdot \mathfrak{Y}\end{aligned}$$

Hence, the protocol names deployed on network nodes are the solutions of network names in a computed network specification. Furthermore, the behavior of $[\![X(A)]\!]_A \parallel (\nu C)[\![X(C)]\!]_C$, by application of axioms $Dep_{1,2}$, Res_2, Br, LM_1, $Sync_1$ and Ch_5, equals $(\{\}, nsnd(req(A), A)).[\![X(A)]\!]_A \parallel (\nu C)[\![X(C)]\!]_C$. This indicates that $[\![X(A)]\!]_A \parallel (\nu C)[\![X(C)]\!]_C = [\![X(A)]\!]_A$, since both are a solution of $\mathfrak{X} \stackrel{def}{=} (\{\}, nsnd(req(A), A)).\mathfrak{X}$ by axiom $Fold$. Intuitively, the hidden node C does not change the behavior of $[\![X(A)]\!]_A$ from the point view of an external observer, since it assumes the action of C belongs to A. However, a recursive term in the scope of an abstraction would become unguarded, as we will explain below. Axioms Ung, $WUng_1$ and $WUng_2$ make it possible to turn each unguarded recursion into a guarded one.

Table 2.5: Axioms for process names.

$rec\mathfrak{A} \cdot t = t\{rec\mathfrak{A} \cdot t/\mathfrak{A}\}$	*Unfold*
$t_1 = t_2\{t_1/\mathfrak{A}\} \Rightarrow t_1 = rec\mathfrak{A} \cdot t_2$, if A is guarded in t_2	*Fold*
$rec\mathfrak{A} \cdot (\mathfrak{A} + t) = rec\mathfrak{A} \cdot t$	*Ung*
$rec\mathfrak{A} \cdot ((\mathcal{C}, \tau).((\mathcal{C}', \tau).t' + t) + s) =$ $rec\mathfrak{A} \cdot ((\mathcal{C}, \tau).(t' + t) + s)$, if $\mathfrak{A}$ is unguarded in t'	$WUng_1$
$rec\mathfrak{A} \cdot ((\mathcal{C}, \eta_\tau).(\mathfrak{A} + t) + s) =$ $rec\mathfrak{A} \cdot ((\mathcal{C}, \eta_\tau).(t + s) + s), \eta_\tau \in \{nrcv(\mathfrak{m}), \tau\}$	$WUng_2$
$\tau_m(rec\mathfrak{A} \cdot t) = rec\mathfrak{A} \cdot \tau_m(t)$, if $\mathfrak{A}$ is serial in t	*Hid*

Axiom *Hid* expresses that the abstraction operator can be moved inside and outside of a recursion operator, when $\mathfrak{A}$ is serial in t. The free network name $\mathfrak{A}$ is *serial* in t, if it does not occur in the scope of parallel, communication merge, left merge, restriction, encapsulation, and abstraction operators in t. This side condition is required to preserve the soundness of the axiom as explained in [9] (This axiom was also considered in [146], which needs a side condition to be sound in the context of *CCS*.) It should be noted that the abstraction operator can make a guarded recursion unguarded. Thus by applying axiom *Hid*, we can move the operator inside the recursion operator and apply its effect, which may result in unguarded recursion. Then by moving it out and applying $WUng_1$ and $WUng_2$, we can convert it to a guarded one. Finally, by applying *Unfold*, we can remove the abstraction operator completely.

Theorem 2.6. *The axiomatization is sound, i.e. for all closed computed network terms t_1 and t_2, if $t_1 = t_2$ then $t_1 \simeq_{rb} t_2$.*

Our axiomatization is also ground-complete for terms with a finite-state CLTS. For example, $rec\mathfrak{W} \cdot (\{\}, nsnd(req(A), A)).\mathfrak{W} \parallel \sum_{lx:Loc}(\{? \rightsquigarrow B\}, nrcv(req(lx))).\mathfrak{W}$ produces an infinite-state transition system, since at each recursive call, a new parallel operator is generated. Thus, its equality to $rec\mathfrak{H} \cdot (\{\}, nsnd(req(A), A)).\mathfrak{H}$ can not be proved by our axiomatization.

Theorem 2.7. *The axiomatization is ground-complete, i.e. for all closed finite-state computed network terms t_1 and t_2, $t_1 \simeq_{rb} t_2$ implies $t_1 = t_2$.*

The proofs of the above theorems are presented in [73].

2.3.5 Symbolic Verification

To verify MANET protocols for large networks, or if one needs to deal with infinite data domains, model checking is not readily applicable. Since MANETs often consist of an arbitrary set of nodes that run the same protocols, we develop a symbolic verification technique for such networks within the *CNT* framework, based on the cones and foci method [67, 68, 86]. This technique works on a restricted

class of specifications, called *linear computed network equations*, in which the states are data objects, and rephrases the question whether the system specification and implementation are equivalent in terms of proof obligations on relations between data objects. We exploit our equations to convert the parallel composition of an arbitrary number of similar processes, modulo some data parameters, to a single linear equation using the Composition Theorem from [87]. The Composition Theorem however is based on the assumption that communications are restricted to two processes. Since in our framework broadcast communication is an essential ingredient, we generalize the Composition Theorem to this setting. The linear equation representing the MANET of similar nodes and the desired external behavior of this network (also expressed by a linear equation) are taken as input to the symbolic verification technique, which reduces the question of their behavioral equivalence to proving data equalities.

Linear Computed Network Equations and Invariants A linear computed network equation (LCNE) is a computed network term consisting of only action prefix, summation and conditional operators; it does not contain any parallel, encapsulation, abstraction and hiding operators. An LCNE is basically a vector of data parameters together with a list of condition, action and effect triples, describing for each state under which condition an action may happen and what is its effect on the vector of data parameters. Each computed network term can be transformed into an LCNE using the axioms (cf. [143]).

Without loss of generality, we assume that each message constructor has exactly one parameter. Let the set of (concrete) actions be Act^c, ranged over by $\eta(-)$, defined as:

$$Act^c = \{nsnd(m(-), \ell), nrcv(m(-)) | m : D_m \to Msg,\ \ell \in Loc\}$$

Definition 2.8. A linear computed network equation is a *CNT* specification of the form

$$A(d : D) \stackrel{def}{=} \sum_{\eta : Act^c \cup \{\tau\}} \sum_{e:E} [h_\eta(d,e)](\mathcal{C}_\eta(d,e), \eta(f_\eta(d,e))).A(g_\eta(d,e)) \diamond 0$$

where $h_\eta : D \times E \to Bool$, $\mathcal{C}_\eta : D \times E \to \mathbb{C}$, $f_\eta : D \times E \to D_m$ and $g_\eta : D \times E \to D$ for each $\eta \in Act^c \cup \{\tau\}$.

The LCNE in Definition 2.8 has exactly one *CLTS* as its solution (modulo strong bisimilarity). In this *CLTS*, the states are data elements $d : D$, where D may be a Cartesian product of n data types, i.e. $(d_1, \ldots, d_n)$, the transition labels are the network send and receive actions of messages parameterized with data, and the transition constraints are network constraints parameterized with data. The LCNE expresses that state d can send/receive message $\eta(f_\eta(d,e))$ for the set of topologies specified by $\mathcal{C}_\eta(d,e)$ to end up in state $g_\eta(d,e)$ under the condition that $h_\eta(d,e)$ is true.

Definition 2.9. A mapping $\mathcal{I} : D \to Bool$ is an invariant for an LCNE, written as in Definition 2.8, if for all $\eta \in Act^c \cup \{\tau\}$, $d : D$ and $e : E$,

$$\mathcal{I}(d) \wedge h_\eta(d, e) \Rightarrow \mathcal{I}(g_\eta(d, e)).$$

Invariants can be used to characterize the set of reachable states of an LCNE. Namely, if $\mathcal{I}(d)$ and it is possible to perform $\eta(f_\eta(d, e))$ (since $h_\eta(d, e)$ holds), then $\mathcal{I}$ holds in the resulting state $g_\eta(d, e)$.

Equivalence Checking by using State Mappings The system implementation and specification, both given in linear format, are branching computed network bisimilar if a state mapping ϕ exists from implementation to specification which satisfies the transfer conditions of a branching computed network bisimulation. An invariant $\mathcal{I}$ can be imposed; then the transfer conditions only need to hold in states where $\mathcal{I}$ is true, and consequently equivalence between implementation and specification is only guaranteed to hold in states where $\mathcal{I}$ is true.

We omit the abstraction operator $\tau_{\widetilde{M}}$ to ensure that the recursive specification of $Impl$ has a finite-state behavior. However we consider its effect in our equivalence relation (regarding the serial condition of the axiom Hid). The set of communications over $\widetilde{M}$ is defined by $I_{\widetilde{M}}$ as

$$\{nsnd(\mathcal{C}, \mathfrak{m}, \ell), nrcv(\mathcal{C}, \mathfrak{m}) | \exists m \in \widetilde{M} \cdot isType_m(\mathfrak{m})\}.$$

Recall that $\langle\eta\rangle$ denotes η or $\eta[\ell/?]$ for some ℓ when η is of the form $nsnd(\mathfrak{m}, ?)$. Depending on the value of $\langle\eta\rangle$, for any arbitrary binary relation $\odot$, $r_\eta(e, d) \odot r'_{\langle\eta\rangle}(e, d')$ holds iff $r_\eta(e, d) \odot r'_\eta(e, d')$ or $r_\eta(e, d)[\ell/?] \odot r'_{\eta[\ell/?]}(e, d')$ holds.

Proposition 2.10. *Let the LCNE Imp be of the form*

$$Imp(d : D) \stackrel{def}{=} \sum_{\eta \in Act^c \cup \{\tau\}} \sum_{e:E} [h_\eta(d, e)](\mathcal{C}_\eta(d, e), \eta(f_\eta(d, e))).Imp(g_\eta(d, e)) \diamond 0$$

Furthermore, let the LCNE $Spec$ be of the form

$$Spec(d' : D') \stackrel{def}{=} \sum_{\eta \in Act^c \setminus I_{\widetilde{M}}} \sum_{e:E} [h'_\eta(d', e)](\mathcal{C}'_\eta(d', e), \eta(f'_\eta(d', e))).Spec(g'_\eta(d', e)) \diamond 0$$

Let $\mathcal{I} : D \to Bool$ be an invariant for Imp, and $\phi : D \to D'$ a state mapping. If for all $\eta \in Act^c \setminus I_{\widetilde{M}}$ and $\eta_\tau \in I_{\widetilde{M}}$, ϕ satisfies the following conditions:

1. $\forall e : E(h_{\eta_\tau}(d, e) \Rightarrow \phi(d) = \phi(g_{\eta_\tau}(d, e)))$;
2. $\forall e : E$, $h_\eta(d, e)$ *implies that either η is a receive action and $\phi(d) = \phi(g_\eta(d, e))$, or $h'_{\langle\eta\rangle}(\phi(d), e)$ holds for some $\langle\eta\rangle$ such that $f_\eta(d, e) = f'_{\langle\eta\rangle}(\phi(d), e)$, $\mathcal{C}'_{\langle\eta\rangle}(\phi(d), e) \subseteq \mathcal{C}_\eta(d, e)$, and $\phi(g_\eta(d, e)) = g'_{\langle\eta\rangle}(\phi(d), e)$;*

3. $\forall e : E$, $h'_\eta(\phi(d), e)$ *implies that either* η *is a receive action such that* $\phi(d) = g'_\eta(\phi(d), e)$, *or there exists* d^* *such that* $d \xrightarrow{\eta_{\tau_1}}_{\mathcal{C}_1} \ldots \xrightarrow{\eta_{\tau_n}}_{\mathcal{C}_n} d^*$, *where* $\eta_{\tau_1}, \ldots, \eta_{\tau_n} \in I_{\widetilde{M}}$, *and for some* $\langle\eta\rangle$, $h_{\langle\eta\rangle}(d^*, e)$ *holds with* $f_{\langle\eta\rangle}(d^*, e) = f'_\eta(\phi(d), e)$, $\mathcal{C}_{\langle\eta\rangle}(d^*, e) \subseteq \mathcal{C}'_\eta(\phi(d), e)$, *and* $\phi(g_{\langle\eta\rangle}(d^*, e)) = g'_\eta(\phi(d), e))$;

then for all $d : D$ *with* $\mathcal{I}(d)$, $\tau_{\widetilde{M}}(Imp(d)) \simeq_b Spec(\phi(d))$.

See [74] for the proof. Since each state of the specification defines the external behavior of the implementation with regard to any possible topology changes, the mapped state of the implementation should not be changed by τ-transitions, as implied by the first criterion. The second criterion implies that for any receive action that does not change the behavior of the implementation, either $\phi(d) = \phi(g_\eta(d, e))$ holds or the specification has the same receive action that mimics its behavior. Furthermore for send actions it implies that there exists some action $\langle\eta\rangle$ in the specification which mimics the behavior of the implementation on $(\mathcal{C}_\eta(d, e), \eta(f_\eta(d, e)))$ by performing $(\mathcal{C}'_{\langle\eta\rangle}(\phi(d), e), \langle\eta\rangle(f_\eta(d, e)))$ where $\mathcal{C}'_{\langle\eta\rangle}(\phi(d), e) \subseteq \mathcal{C}_\eta(d, e)$. Similarly, the third criterion implies that for any receive action of specification that does not change the behavior either $\phi(d) = g'_\eta(\phi(d), e)$ or there exists a state d^*, reachable after a set of communications over abstracted messages (which preserves the behavior of the implementation in the state d), which mimics the behavior of the specification on $(\mathcal{C}'_\eta(\phi(d), e), \eta(f'_\eta(\phi'(d), e)))$ by performing $(\mathcal{C}_{\langle\eta\rangle}(d^*, e), \langle\eta\rangle(f'_\eta(\phi(d), e)))$ where $\mathcal{C}_{\langle\eta\rangle}(d^*, e) \subseteq \mathcal{C}'_\eta(\phi(d), e)$. The first criterion together with the first part of the second and third criteria (i.e., conditions on receive actions which do not change the behavior) enforce the first transfer condition while the second part of the second and third criteria imply the second transfer condition of branching computed network bisimulation.

Due to mobility of nodes, MANET protocols usually contain mechanisms to examine if a node connection to some other node exists or not. For instance, a node may examine whether it is still connected to its next hop for a destination in a routing protocol, or to its leader in a leader election protocol. Such mechanisms are modeled by non-deterministic behavior in the protocol specification, which restarts some part of the process (like route discovery in a routing protocol). Due to such mechanisms, in each state of the implementation, the observable behavior may change after a set of τ-transitions. On the other hand, since we assume arbitrary mobility for MANET nodes, each state of the specification defines the behavior of a MANET for any possible topology change. Therefore, we lack a collection of so-called focus points [68, 86]: states in the implementation that can be matched to some state in the specification with the same observable behavior.

To illustrate the application of Proposition 2.10, let us show that $\forall n : Nat \cdot N(n) \simeq_b M(n)$, where

$$\begin{aligned} N(n : Nat) &\stackrel{def}{=} [n \geq 1](\{\}, nsnd(data(B), A)).N(n+1) \diamond 0 + \\ &\quad\ [n \geq 1](\{\}, nsnd(data(B), ?)).N(n+2) \diamond 0 \\ M(b : Bool) &\stackrel{def}{=} [eq(b, T)](\{\}, nsnd(data(B), A)).M(b) \diamond 0 \end{aligned}$$

it suffices to show that $\phi(n) = if(n \geq 1, T, F)$ satisfies the second and third conditions of Proposition 2.10 (as there is no abstraction):

- When $n \geq 1$ holds, two actions $\eta_1 \equiv nsnd(data(-), A)$ and $\eta_2 \equiv \eta_1[?/A]$ are possible. For the first action, $f_{\eta_1}(n) = B$, $\mathcal{C}_{\eta_1}(n) = \{\}$, and $g_{\eta_1}(n) = n + 1$. The we have that $\phi(n) = T$, $h'_{\eta_1}(T)$, $f_{\eta_1}(n) = f'_{\eta_1}(T)$, $\mathcal{C}'_{\eta_1}(T) \subseteq \mathcal{C}_{\eta_1}(n)$, and $\phi(g_{\eta_1}(n)) = g'_{\eta_1}(T)$. For the second action, $f_{\eta_2}(n) = B$, $\mathcal{C}_{\eta_2}(n) = \{\}$, and $g_{\eta_2}(n) = n + 2$. The only action of M is again matched to this action, since $\langle nsnd(data(-), ?)\rangle = nsnd(data(-), A)$, $f_{\eta_2}(n) = f'_{\langle \eta_2 \rangle}(T)$, $\mathcal{C}'_{\langle \eta_2 \rangle}(T) \subseteq \mathcal{C}_{\eta_2}(n)$, and $\phi(g_{\eta_2}(n)) = g'_{\langle \eta_2 \rangle}(T)$.
- The only action of M when $eq(\phi(n \geq 1), T))$ is $\eta \equiv nsnd(data(-), A)$, and the same action is enabled in N when $n \geq 1$, with the same parameter and network constraint.

Linearization of Uniform MANETs In practice a MANET often consists of an arbitrary set of similar nodes: each node is identified by a unique network address, and deploys the same protocols. In this section we show how our symbolic verification approach can be exploited to verify such networks. To this aim, we first provide a general recursive specification for MANETs with similar nodes, and then derive a LCNE as a solution of the recursive specification, using the *CNT* axioms, data axioms and induction. The derived linear equation is strongly bisimilar to the original recursive equation.

Without loss of generality, we assume that each message constructor has exactly one parameter. We assume that each process $P(\ell, d)$ is defined using a linear process equation (LPE) [24] of the form:

$$P(\ell : Loc, d : D) \stackrel{def}{=} \sum_{m \in Msg} \sum_{e:E_m} [h_{m_s}(\ell, d, e)] snd(m(f_{m_s}(\ell, d, e))).P(\ell, g_{m_s}(\ell, d, e)) \diamond 0 + [h_{m_r}(\ell, d, e)] rcv(m(f_{m_r}(\ell, d, e))).P(\ell, g_{m_r}(\ell, d, e)) \diamond 0 \quad (2.1)$$

where $h_{m_s/m_r} : Loc \times D \times E_m \to Bool$, $f_{m_s/m_r} : Loc \times D \times E_m \to D_m$ and $g_{m_s/m_r} : Loc \times D \times E_m \to D$ for each $m \in Msg$.

As we do not want to fix the addresses of nodes in the MANET beforehand, we use two auxiliary data sorts: $LocList$ which is a list of network addresses of nodes, and similar to the approach of [87], $DTable$ which is a table indexed by network addresses, where each entry maintains the state of the node at the corresponding network address. We also exploit for each $m \in Msg$ an auxiliary data sort $EList_m$, which is a list of elements of sort E_m, the auxiliary data type used in functions of messages (see equation 2.1).

The sort $LocList$ is defined below. Lists are generated from the empty list $empL$ and add, which places a new address in the list. The function has examines if an element belongs to the list; $include$ examines if the first list is included in the second list; $remove$ removes an address from the list; $head$ returns the first

element of the list; $size$ returns the length of the list; $nodup$ examines if the list has no duplicated item; and eq compares two lists.

To increase readability, we write binary functions in infix manner, and use symbols $\emptyset$, $\triangleright$, $\in$, $\subseteq$, $\setminus$, $|\,|$ and $\ell l[0]$ for $empL$, add, has, $include$, $remove$, $size$ and $head(\ell l)$, respectively. The data sort $EList_m$ for $m \in Msg$ is defined in the same way as $LocList$, but using the constant $empE_m$.

$$
\begin{array}{lll}
sort & LocList & \\
func & empL :\rightarrow LocList & \\
 & add : Loc \times LocList \rightarrow LocList & \\
map & has : Loc \times LocList \rightarrow Bool & \\
 & include, eq : LocList \times LocList \rightarrow Bool & \\
 & remove : LocList \times Loc \rightarrow LocList & \\
 & head : LocList \rightarrow Loc & \\
 & size : LocList \rightarrow Nat & \\
 & nodup : LocList \rightarrow Bool & \\
var & \ell l, \ell l_1, \ell l_2 : LocList, \ell, \ell_1, \ell_2 : Loc & \\
rew & has(\ell, empL) = F & LA_1 \\
 & has(\ell_1, add(\ell_2, \ell l)) = if(eq(\ell_1, \ell_2), T, has(\ell_1, \ell l)) & LA_2 \\
 & include(empL, \ell l) = T & LA_3 \\
 & include(add(\ell, \ell l_1), \ell l_2) = has(\ell, \ell l_2) \wedge include(\ell l_1, \ell l_2) & LA_4 \\
 & remove(empL, \ell) = empL & LA_5 \\
 & remove(add(\ell_1, \ell l), \ell_2) = & \\
 & \quad if(eq(\ell_1, \ell_2), remove(\ell l, \ell_2), add(\ell_1, remove(\ell l, \ell_2))) & LA_6 \\
 & head(add(\ell, \ell l)) = \ell & LA_7 \\
 & size(empL) = 0 & LA_9 \\
 & size(add(\ell, \ell l)) = size(\ell l) + 1 & LA_{10} \\
 & nodup(empL) = T & LA_{11} \\
 & nodup(add(\ell, \ell l)) = \neg has(\ell, \ell l) \wedge nodup(\ell l) & LA_{12} \\
 & eq(\ell l_1, \ell l_2) = include(\ell l_1, \ell l_2) \wedge include(\ell l_2, \ell l_1) & LA_{13}
\end{array}
$$

Tables are generated from the constant $empT$ and an operation upd, which places a new entry in the table. The function get gets an entry from the table using its index. The function $upd\_all_{g_m}(\ell \triangleright \ell l, e \triangleright el, dt)$ updates the list of entries $\ell \triangleright \ell l$ in the table using the function $g_m : Loc \times D \times E_m \rightarrow D$; the entry ℓ is updated by $g_m(\ell, get(\ell, dt), e)$, which uses the network address ℓ, the previous value at the entry, and an auxiliary value e. Intuitively this function is helpful to update a set of receiver nodes that communicate with a sender through message m. Similarly the function $and\_all_{h_m, f_m, f'_m}(\ell_1 \triangleright \ell l, \ell_2, e_2, e_1 \triangleright el, dt)$ examines a boolean expression on a list of entries $\ell_1 \triangleright \ell l$ using functions $h_m : Loc \times D \times E_m \rightarrow Bool$ and $f_m, f'_m : Loc \times D \times E_m \rightarrow D_m$; for each entry ℓ_1, it examines if $h_m(\ell_1, get(\ell_1, dt), e_1)$ evaluates to true and if $f'_m(\ell_1, get(\ell_1, dt), e_1)$ is equal to $f_m(\ell_2, get(\ell_2, dt), e_2)$. Intuitively this function is helpful to examine if a set of nodes can synchronize with each other upon receiving a message of type m, i.e., whether the conditions of their actions are true (examined by h_m) and their

message parameters are equal to each other (examined by f_m, f'_m).

$$
\begin{array}{lll}
sort & DTable & \\
func & empT :\rightarrow DTable & \\
 & upd : Loc \times D \times DTable \rightarrow DTable & \\
map & get : Loc \times DTable \rightarrow D & \\
 & upd\_all_{g_m} : LocList \times EList_m \times DTable \rightarrow DTable & \\
 & and\_all_{h_m,f_m,f'_m} : LocList \times Loc \times E_m \times EList_m \times DTable \rightarrow Bool & \\
var & \ell, \ell_1, \ell_2 : Loc, \ell l : LocList, & \\
 & d : D, dt : DTable, & \\
 & e, e_1, e_2 : E_m, el : EList_m & \\
rew & get(\ell_1, upd(\ell_2, d, dt)) = if(eq(\ell_1, \ell_2), d, get(\ell_1, dt)) & TA_1 \\
 & upd\_all_{g_m}(empL, el, dt) = dt & TA_2 \\
 & upd\_all_{g_m}(add(\ell, \ell l), add(e, el), dt) = & \\
 & \quad upd(\ell, g_m(\ell, get(\ell, dt), e), upd\_all_{g_m}(\ell l, el, dt)) & TA_3 \\
 & and\_all_{h_m,f_m,f'_m}(empL, \ell, e, el, dt) = T & TA_4 \\
 & and\_all_{h_m,f_m,f'_m}(add(\ell_1, \ell l), \ell_2, e_2, add(e_1, el), dt) = & \\
 & \quad and(h_m(\ell_1, get(\ell_1, dt), e_1), and(eq(f_m(\ell_2, get(\ell_2, dt), e_2), & \\
 & \quad\quad f'_m(\ell_1, get(\ell_1, dt), e_1)), and\_all_{h_m,f_m,f'_m}(\ell l, \ell_2, e_2, el, dt))) & TA_5
\end{array}
$$

Axioms TA_{2-5} are schematic and can be defined for all functions g_{m_s/m_r}, h_{m_s/m_r}, f_{m_s/m_r} in equation (2.1) for any $m \in Msg$.

In the remainder we write $dt[\ell]$ instead of $get(\ell, dt)$. The following network recursive specification puts nodes deploying process P at network addresses of ℓl in parallel.

$$
\begin{array}{l}
Manet(\ell l : LocList, dt : DTable) \stackrel{def}{=} \\
\quad [eq(\ell l, \emptyset)]0 \diamond [\![P(\ell l[0], dt[\ell l[0]])]\!]_{\ell l[0]} \parallel Manet(\ell l \setminus \ell l[0], dt).
\end{array}
\tag{2.2}
$$

Below we present the core lemma of this section. It gives an expansion of $Manet$, where all operators for parallelism have been removed. The resulting network has the list ℓl and the table dt as parameters. In essence, the complexity of the computed network $Manet$ is now encoded using the list and table operations.

Lemma 2.11 says that in the network X, the node with network address $k \in \ell l$ may send the message m, parameterized by data from this node, if it is ready to send (as indicated by $h_{m_s}(k, dt[k], e)$) to a list ℓ_s (without duplicates) of receiver nodes with addresses in $\ell l \setminus k$ that are all ready to receive such a message (examined by $and\_all_{h_{m_r},f_{m_s},f_{m_r}}$). Table entries with indices in ℓ_s and k are updated as a result of this communication (using $upd\_all_{g_{m_r}}$). The function $\mathcal{C}(\ell, \ell_s) = \{\ell \rightsquigarrow \ell' | \ell' \in \ell_s\}$ specifies the network constraint for this behavior of the network, indicating there is a communication link from ℓ to each node in ℓ_s. Nodes in the network X may also receive a message m from an unknown address ?; the receiving nodes must have network addresses in ℓ_s, where $\ell_s \subseteq \ell l \wedge \neg eq(\ell_s, \emptyset) \wedge nodup(\ell_s)$, and must be ready to receive such a message (examined

by $and\_all_{h_{m_r},f_{m_r},f_{m_r}}$). All table entries with indices in ℓ_s are updated as a result of this receive action (using $upd\_all_{g_{m_r}}$).

Lemma 2.11. *The MANET $Manet$ as defined in equations 2.1 and 2.2 is a solution for the MANET X in equation 2.3 below.*

$$
\begin{array}{l}
X(\ell l : LocList, dt : DTable) \stackrel{def}{=} \\
\quad \sum_{m\in Msg} \sum_{k:Loc} \sum_{\ell_s:LocList} \sum_{e:E_m} \sum_{el:EList_m} \\
\qquad [k \in \ell l \wedge \ell_s \subseteq \ell l \setminus k \wedge nodup(\ell_s) \wedge |\ell_s| = |el| \wedge \\
\qquad\quad h_{m_s}(k, dt[k], e) \wedge and\_all_{h_{m_r},f_{m_s},f_{m_r}}(\ell_s, k, e, el, dt)] \\
\qquad\quad (\mathcal{C}(k, \ell_s), nsnd(m(f_{m_s}(k, dt[k], e)), k)). \\
\qquad\qquad X(\ell l, upd(k, g_{m_s}(k, dt[k], e), upd\_all_{g_{m_r}}(\ell_s, el, dt))) \diamond 0 + \\
\quad \sum_{m\in Msg} \sum_{\ell_s:LocList} \sum_{el:EList_m} \\
\qquad [\ell_s \subseteq \ell l \wedge \neg eq(\ell_s, \emptyset) \wedge nodup(\ell_s) \wedge |\ell_s| = |el| \wedge \\
\qquad\quad and\_all_{h_{m_r},f_{m_r},f_{m_r}}(\ell_s, \ell_s[0], el[0], el, dt)] \\
\qquad\quad (\mathcal{C}(?, \ell_s), nrcv(m(f_{m_r}(\ell_s[0], dt[\ell_s[0]], el[0])))). \\
\qquad\qquad X(\ell l, upd\_all_{g_{m_r}}(\ell_s, el, dt)) \diamond 0.
\end{array}
\tag{2.3}
$$

See [74] for the proof. The following Composition Theorem is a corollary of Lemma 2.11 and the axiom $Fold$.

Theorem 2.12. $Manet(\ell l, dt) = X(\ell l, dt)$.

2.4 Actor Model and the Rebeca Language

The actor model [4, 93] has been introduced for the purpose of modeling concurrent and distributed applications. It is an agent-based language introduced by Hewitt [93], extended by Agha to an object-based concurrent computation model [4]. An actor model consists of a set of actors communicating with each other through unicasting asynchronous messages. Each computation unit, modeled by an actor, has a unique address and mailbox. Messages sent to an actor are stored in its mailbox. Each actor is defined through a set of message handlers, called *methods*, to specify the actor behavior upon processing of each message (see Fig. 2.3). In this model, message delivery is guaranteed but is not in-order. This policy implicitly abstracts from effects of the network, i.e., delays over different routing paths, message conflicts, etc., and consequently makes it a suitable modeling framework for concurrent and distributed applications. Another semantic property of this model is atomic execution of methods and fairness in scheduling actors. Intuitively, the time to execute a method is considered negligible compared to the delay of the network. Consequently, instructions of methods are not interleaved, and hence execution of methods becomes atomic in the semantic model.

Rebeca [139] is an actor-based modeling language which aims to bridge the gap between formal verification techniques and the real-world software engineering of concurrent and distributed applications. It provides an operational

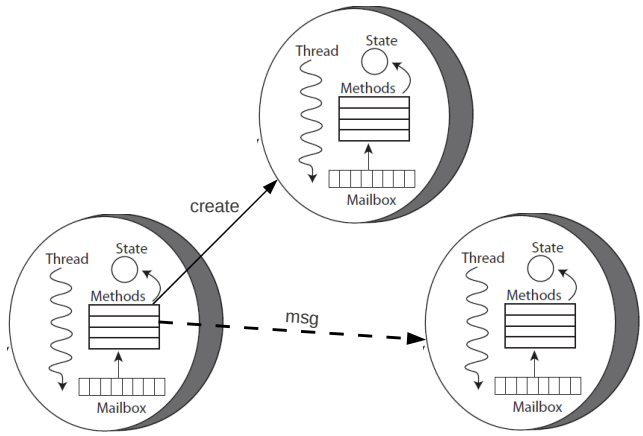


Figure 2.3: Each actor has its own thread of control, mailbox, and address. Upon processing a message, it may create a new actor, send messages to other actors, or update its state variables [100].

interpretation of the actor model through a Java-like syntax, which makes it easy to learn and use. Rebeca is supported by a robust model checking tool, named Afra[1], which takes advantage of various reduction techniques [97, 133] to make efficient verification possible. Due to its design principle it is possible to extend the core language based on the desired domain [138]. For example, different extensions have been introduced in various domains such as probabilistic systems [149], real-time systems [131], software product lines [132], and broadcasting environment [155]. With the aim of reducing the state space, message delivery is considered in-order, and thus each actor mailbox is modeled through a FIFO queue.

In Rebeca, actors are the computation units of the system, called rebecs (short for reactive objects), which are instances of the defined *reactive classes* in the model. Rebecs communicate with each other only through asynchronous message passing. Every sent message eventually will be received and processed by its *potential* receivers. In Rebeca, the rebecs defined as the *known rebecs* of a sender, the sender itself using the "self" keyword, or the sender of the message currently processed using the keyword "sender" are considered as the potential receivers.

Every reactive class has three major parts, first the *known rebecs* to specify the neighbors of the rebec, second the *state variables* to maintain the state of the rebec, and third the *message servers* to indicate the reactions of the rebec on received messages. The local state of a rebec is defined in terms of its state variables together with its message queue. Whenever a rebec receives a message

[1] http://www.rebeca-lang.org/wiki/pmwiki.php/Tools/Afra

```
1   reactiveclass MNode
2   {
3       knownrebecs
4       {
5         MNode next;
6       }
7     statevars
8     {
9       int my_i;
10      boolean done;
11    }

13    msgsrv initial(int j,
          boolean starter)
14    {
15      my_i = j;
16      if(starter) {
17        done = true;
18        next.send(my_i);
19      } else
20        done = false;
21    }
23    msgsrv send(int i)
24    {
25      if (i < my_i) {
26        if (!done) {
27          done = true;
28          next.send(my_i);
29        }
30      } else {
31        my_i = i;
32        done = true;
33      }
34    }
35  }
36  main
37  {
38    MNode n1(n2):(1,false);
39    MNode n2(n3):(2,false);
40    MNode n3(n4):(3,true);
41    MNode n4(n1):(4,false);
42  }
```

Figure 2.4: An example in Rebeca: Max-algorithm with 4 nodes in a ring topology.

which has no corresponding message server to respond to, it simply discards the message. Each rebec has at least one message server called “initial”, which acts like a constructor in object-oriented languages and performs the initialization tasks.

A rebec is said to be *enabled* if and only if it has at least one message in its queue. The computation takes place by removing a message from the queue and executing its corresponding message server atomically, after which the rebec proceeds to process the other messages in its queue (if any). Processing a message may have the following consequences:

- it may modify the value of the state variables of the executing rebec, or
- some messages may be sent to other rebecs.

Each Rebeca model consists of two parts, the *reactive classes* part and the *main* part. In the *main* part the instances of the reactive classes are created initially while their known rebecs and local variables are initialized.

As an example, Figure 2.4 illustrates a simple max finding algorithm modeled in Rebeca, referred to as “Max-Algorithm”. Every node in the network with a ring topology contains an integer value and they intend to find the maximum value of all nodes in a distributed manner. Each node knows its next neighbour in the

ring topology. The `initial` message server has a parameter, named `starter`. The rebec with the `starter` value *true* initiates the algorithm by sending the first message to its next neighbour. Whenever a node receives a value from its preceding neighbour, it compares this value with its current value and one of the following scenarios happens:

- if it has not sent its value yet and its value is greater than the received one, it sends its value to its next neighbour;
- if its current value is less than the received one, it gives up sending its value and updates its current value to the received one;
- if it has already sent its value, it only checks whether it must updates its value.

This protocol does not work on MANETs as nodes give up to resend their value after their first send. The Max-Algorithm should find the maximum value among the connected nodes in MANETs. To this aim, if a node moves and connects to new nodes, it has to re-send its value as its value may be the maximum value in the currently connected nodes.

Reliable Restricted Broadcast Process Theory

3

Most frameworks for the formal analysis of mobile ad hoc network (MANET) protocols, such as [64, 78, 79, 81, 104, 115, 121, 122, 137] similar to *CNT*, focus on protocols above the data link layer; hence they support the core services of this layer, which means that local broadcast is the primitive means of communication. Wireless communication at this layer is *non-blocking*, i.e., the sender broadcasts irrespective of the readiness of its receivers, and is *asynchronous*, i.e., received packets are buffered at the receiver. The data link layer of a node processes the packet if it is an intended destination. While a node is busy processing a message, it can still receive messages, buffer them and process them later. However, if two different nodes broadcast simultaneously with a common node in their range, the latter node cannot receive both messages and drops one of them, which is called the hidden node problem. We say that wireless communication is *reliable* if the intended receivers successfully receive the packet. In other words, message delivery is guaranteed to all connected neighbors.

Although lossy communication is an integral part of MANETs, mimicking it faithfully in a formal framework can hamper the formal analysis of MANET protocols. To obtain a deeper understanding of a malfunctioning of such a protocol due to a conceptual mistakes in its design rather than unreliable communication, it may be helpful to consider communication reliable, meaning that the possibility of the hidden node problem is omitted from the framework [62, 64]. Therefore we modify the core of *CNT* by considering communication among the nodes in the network to be reliable and refer to the process algebra as *Reliable Restricted Broadcast Process Theory (RRBPT)*.

Somewhat surprisingly, all the results do not carry over in a straightforward fashion from *CNT* to *RRBPT*. In a lossy setting, the non-blocking property of local broadcast communication is an immediate consequence of the rule Par and its counterpart in Table 2.1 for the parallel composition expressing that if a node is not ready to participate in a communication, then we can assume that either it was disconnected from the sender or it was connected but has lost the message. However, in the reliable setting, to guarantee the non-blocking property, nodes should always be input-enabled. The input-enabledness feature is ensured

through the *RRBPT* SOS rules, where the main difference between *RRBPT* and *RBPT* is: in *RRBPT*, nodes lose a communication only when they are disconnected and are always input-enabled. To express such conditions, we extend network constraints with disconnectivity pairs. The introduction of such information makes it possible to reason about topology-dependent behaviors of MANETs at CLTS-level through model checking techniques, as will be discussed in Chapter 5. *RRBPT* provides a sensing operator which allows to change the control flow of a process depending on the status of node connectivity with other nodes. This operator abstracts away from the neighbor discovery service by which a node becomes aware of its connection topology. The neighbor discovery protocol is implemented by periodically sending *hello* messages and acknowledging such messages received from a neighbor. The novel *sensing* operator, inspired by the work in [104], obviates the explicit modeling of this service, and consequently makes modeling of protocols using such a service easier. The neighbor discovery service is modeled implicitly in the semantics in [104]. To this aim, an arbitrary subset of a node's neighbors is considered as the neighbors discovered by such a protocol. We highlight challenges of bringing input-enabledness feature in the semantics of *RRBPT* in the presence of the sensing operator. Furthermore, the behavioral equivalence relation of *CNT* setting is not a congruence with respect to parallel composition anymore. To support the desired distinguishing power, we provide a new bisimulation relation which guarantees the congruence property for MANETs.

The reliable framework, *RRBPT*, can be extended in the same way as *RBPT* with computed network terms and the auxiliary operators *left merge* ($\between$) and *communication merge* ($|$) to provide a sound and complete axiomatization for the parallel composition. However, the input-enabledness feature and the new sensing operator require new auxiliary operators to assist their axiomatization (which enable linearization of processes at the syntactic level to take benefit of symbolic verification). To this aim, we discuss the appropriate axioms of *RRBPT*. We utilize our axioms to analyze the correctness of protocols at the syntactic level. To this aim, we facilitate the specification of the protocol behaviors preconditioned to multihop constraints and then introduce a new notion of refinement among protocol implementations and their specifications. Such a relation abstracts away from a sequence of multi-hop communications, leading to an application-level action preconditioned by a multi-hop constraint over the topology. Therefore, the correctness of a protocol (with a finite-state behavior) is accomplished by proving that the implementation rewritten into a recursive specification by our axiomatization (made of only dynamic operators) refines the specification. We demonstrate the applicability of our framework by analyzing and proving the correctness of a simple routing protocol inspired by the AODV protocol.

This chapter is organized as follows. Section 3.1 extends the network constraints with negative pairs, and explains how they are helpful in giving semantics to the reliable communication. Section 3.2 introduces the syntax and semantics of *RRBPT*. Sections 3.3 and 3.4 provide the appropriate notion of behavioral

equivalence and axioms in the reliable setting, respectively. We demonstrate the applicability of our axioms by analyzing a simple routing protocol in Section 3.5. Later, we study a leader election protocol devised for MANETs in Section 3.6 and discuss how its semantic model can be derived by mCRL2 toolset and to what extend it can be analyzed. We review and compare the related process algebraic frameworks in depth in Section 3.7 before concluding the chapter.

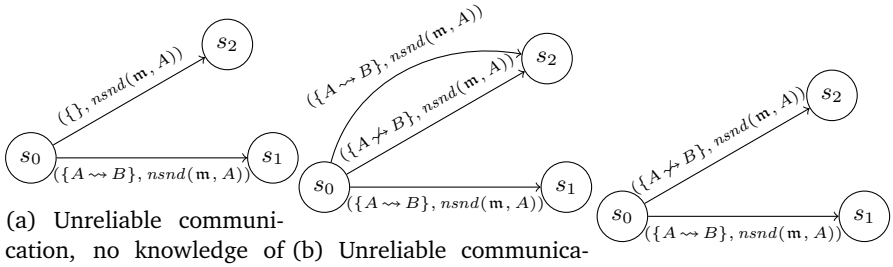
3.1 Extending Network Constraints

A network constraint $\mathcal{C}$ is a set of connectivity pairs $\rightsquigarrow$: $Loc \times Loc$ and disconnectivity pairs $\not\rightsquigarrow$: $Loc \times Loc$. In this setting, non-existence of (dis)connectivity information between two addresses implies lack of information about this link (which can e.g. be helpful when the link has no effect on the evolution of the network). For instance, $B \rightsquigarrow A$ denotes that A is connected to B directly and consequently A can receive data sent by B as before, while $B \not\rightsquigarrow A$ denotes that A is not connected to B directly and consequently cannot receive any message from B. We write $\{B \rightsquigarrow A, C,\ B \not\rightsquigarrow D, E\}$ instead of $\{B \rightsquigarrow A,\ B \rightsquigarrow C,\ B \not\rightsquigarrow D,\ B \not\rightsquigarrow E\}$.

A network constraint $\mathcal{C}$ is said to be *well-formed* if $\forall \ell, \ell' \in Loc\ (\ell \rightsquigarrow \ell' \notin \mathcal{C} \vee \ell \not\rightsquigarrow \ell' \notin \mathcal{C})$. Let $\mathbb{C}^v(Loc)$ denote the set of well-formed network constraints that can be defined over the network addresses in Loc. We define an ordering on network constraints. We say that $\mathcal{C}_1 \preccurlyeq \mathcal{C}_2$ iff $\mathcal{C}_2 \subseteq \mathcal{C}_1$ or $\exists\, \ell \in Loc\, (\mathcal{C}_2[\ell/?] \subseteq \mathcal{C}_1)$, where $d[d_1/d_2]$ denotes the substitution of d_1 for d_2 in d; this can be extended to process terms. For instance, $\{B \rightsquigarrow A\} \preccurlyeq \{? \rightsquigarrow A\}$ and $\{B \rightsquigarrow A,\ B \rightsquigarrow C\} \preccurlyeq \{B \rightsquigarrow A\}$. Each well-formed network constraint $\mathcal{C}$ represents the set of network topologies that satisfy the (dis)connectivity pairs in $\mathcal{C}$, i.e., $\Gamma(\mathcal{C}) = \{\gamma \mid \mathcal{C}_\Gamma(\gamma) \preccurlyeq \mathcal{C}\}$ where $\mathcal{C}_\Gamma(\gamma) = \{\ell \rightsquigarrow \ell' \mid \ell' \in \gamma(\ell)\} \cup \{\ell \not\rightsquigarrow \ell' \mid \ell' \notin \gamma(\ell)\}$ extracts all one-hop (dis)connectivity information from γ. So the empty network constraint $\{\}$ still denotes all possible topologies over Loc. The negation $\neg\,\mathcal{C}$ of network constraint $\mathcal{C}$ is obtained by negating all its (dis)connectivity pairs. Clearly, if $\mathcal{C}$ is well-formed then so is $\neg\,\mathcal{C}$.

3.1.1 Reliable versus Unreliable Communication

Different behaviors for the communication primitives can be defined with the help of appropriate network constraints (see Fig. 3.1). Suppose that node A sends a message, and B is the only neighbor (waiting to receive). If B is connected to A and communication is reliable, then B will receive the message, as indicated in Fig. 3.1c. If on the other hand communication is unreliable, then B may or may not receive the message, as modeled in Fig. 3.1b. In this case we can infer the possible neighbors of A that did not receive the message. Such information can be hidden by merging information $A \rightsquigarrow B$ and $A \not\rightsquigarrow B$ and using the general network constraint $\{\}$, as in Fig. 3.1a. By using appropriate SOS rules, either of the three behaviors in Fig. 3.1 can be defined. In the *CNT* framework, the behavior of communication is as the one depicted in Fig. 3.1a.



(a) Unreliable communication, no knowledge of neighbors

(b) Unreliable communication

(c) Reliable communication

Figure 3.1: Modeling different communication behaviors: s_0 represents a state in which A broadcasts its data, s_1 a state after a successful transmission of data from A to B, and s_2 a state after an unsuccessful communication.

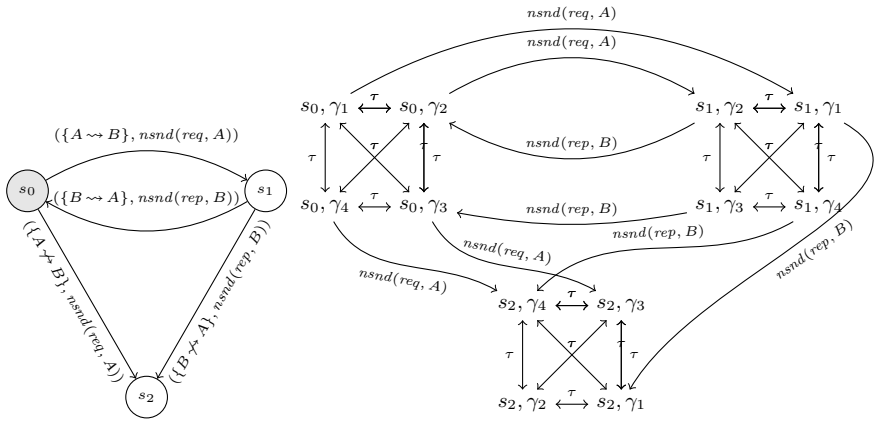
3.1.2 Unfolding a CLTS into an LTS

Mobility of nodes is still modeled implicitly through the semantic models. However, in this setting, the network constraint at each transition implying a send action subsumes (dis)connectivity information between the sender and other network nodes (see Fig. 3.1c). Hence, the CLTS of Fig. 2.1 is revised as the one in Fig. 3.2a. State s_1 still represents that a message has been received by B. The CLTS indicates that the network moves to the deadlock state s_2 if either B is disconnected from A at the moment that A is transmitting req or A is disconnected from B at the moment that B is transmitting rep. The corresponding unfolded LTS is shown in Fig. 3.2b for $Loc = \{A, B\}$. Thus, the possible topologies are $\gamma_1 = \{A \mapsto \{B\}, B \mapsto \emptyset\}$, $\gamma_2 = \{A \mapsto \{B\}, B \mapsto \{A\}\}$, $\gamma_3 = \{A \mapsto \emptyset, B \mapsto \{A\}\}$, and $\gamma_4 = \{A \mapsto \emptyset, B \mapsto \emptyset\}$. The three states s_0, s_1, s_2 are paired with the four possible topologies $\gamma_1, \gamma_2, \gamma_3, \gamma_4$, leading to twelve states as before.

3.2 Syntax and semantics of *RRBPT*

Recall that Msg denotes a set of messages communicated over a network, ranged over by $\mathfrak{m}$. Furthermore, $\mathcal{A}$ denotes a countably infinite set of process names which are used as recursion variables in recursive specifications. Recall that $nsnd : Msg \times Loc$ and $nrcv : Msg$ represent *network send* and *network receive* actions and $snd, rcv : Msg$ *protocol send* and *protocol receive* actions. Let $IAct$ be a set of internal actions, ranged over by i. The syntax of *RRBPT* is given by the following grammar, where ℓ ranges over Loc:

$$t ::= 0 \mid \alpha.t \mid t + t \mid [\![t]\!]_\ell \mid t \parallel t \mid \mathfrak{A}, \mathfrak{A} \stackrel{def}{=} t \mid$$
$$sense(\ell, t, t) \mid (\nu\ell)t \mid \tau_{\mathfrak{m}}(t) \mid \partial_{\mathfrak{m}}(t)$$



(a) A CLTS. (b) The corresponding unfolded LTS.

Figure 3.2: A CLTS and its unfolded LTS. The τ-transitions model mobility of nodes.

We recall from Section 2.3.2 that the deadlock process is modeled by 0. The process $\alpha.t$ performs action α and then behaves as process t, where α is either an internal action or a protocol send/receive action $snd(\mathfrak{m})/rcv(\mathfrak{m})$. We remark that *RRBPT* does not include computed network terms in comparison with *CNT*, so prefixed-actions of the form $(\mathcal{C}, \eta)$ are not supported. Internal actions are useful in modeling the interactions of a process with other applications running on the same node. Protocol send/receive actions specify the interaction of a process with its data-link layer protocols: these protocols are responsible for transferring messages reliably throughout the network. These actions are turned into their corresponding network ones via the semantics (see Section 3.2.1): the send action $nsnd(\mathfrak{m}, \ell)$ denotes that the message $\mathfrak{m}$ is transmitted from a node with the address ℓ, while the receive action $nrcv(\mathfrak{m})$ denotes that the message $\mathfrak{m}$ is ready to be received.

The process $t_1 + t_1$ behaves non-deterministically as t_1 or t_2. The simplest form of a MANET is a node, represented by the deployment operator $[\![t]\!]_\ell$, denoting process t deployed on a node with the known network address $\ell \neq ?$ (where $?$ denotes the unknown address). A MANET can be composed by putting MANETs in parallel using $\|$; the nodes communicate with each other by reliable restricted broadcast. A process name is specified by $\mathfrak{A} \stackrel{def}{=} t$ where $\mathfrak{A} \in \mathcal{A}$ is a name.

As a running example, $P \stackrel{def}{=} init.snd(req).rcv(rep).succ.P$ denotes a process that recursively broadcasts a message req after performing the internal action $init$, waits to receive a rep and then performs an internal action $succ$; and $Q \stackrel{def}{=} rcv(req).snd(rep).Q$ a process that recursively receives a message req and then replies by sending rep. The internal action $init$ represents initialization of a route discovery from an upper layer application, and $succ$ a route discovery notification

to an upper layer application. The network process $[\![P]\!]_A \parallel [\![Q]\!]_B$ specifies an ad hoc network composed of two nodes with network addresses A and B deploying processes P and Q, respectively.

MANET protocols may behave based on the (non-)existence of a link. A neighbor discovery service can be implemented at the network layer, by periodically sending *hello* messages and acknowledging such messages received from a neighbor. The sensing operator $sense(\ell', t_1, t_2)$ examines the status of the link from the node with address ℓ' to the node, say with address ℓ, that the sensing is executed on; in case of its existence it behaves as t_1, and otherwise as t_2. For instance, the term $[\![sense(\ell', t_1, t_2)]\!]_\ell$ examines the existence of the link $\ell' \rightsquigarrow \ell$, and then behaves accordingly. The hide operator $(\nu\ell)t$ conceals the address ℓ in the process t, by renaming this address to ? in network send/receive actions. For each message $\mathfrak{m} \in Msg$, the abstraction operator $\tau_{\mathfrak{m}}(t)$ renames network send/receive actions over message $\mathfrak{m}$ to τ, and the encapsulation operator $\partial_{\mathfrak{m}}(t)$ forbids receiving the message $\mathfrak{m}$. Let $\tau_{\{\mathfrak{m}_1,\ldots,\mathfrak{m}_n\}}(t)$ and $\partial_{\{\mathfrak{m}_1,\ldots,\mathfrak{m}_n\}}(t)$ denote $\tau_{\mathfrak{m}_1}(\ldots(\tau_{\mathfrak{m}_n}(t))\ldots)$ and $\partial_{\mathfrak{m}_1}(\ldots(\partial_{\mathfrak{m}_n}(t))\ldots)$.

For example, $\tau_{Msg}(\partial_{Msg}([\![P]\!]_A \parallel [\![Q]\!]_B))$ specifies an isolated MANET that cannot receive any message from the environment, while its communications (i.e. send actions) are abstracted away.

Terms should be grammatically well-defined, meaning that each process deployed at a network address should be only defined by action prefix, choice, sense and process names.

3.2.1 Operational Semantics

Let $PAct$ and $NAct$ denote the set of protocol and network send and receive actions respectively, and $IAct$ the set of internal actions. We assume that $\alpha \in PAct \cup IAct$, $\eta \in NAct \cup IAct \cup \{\tau\}$, $i \in IAct$, and $\iota \in IAct \cup \{\tau\}$. The SOS rules in Table 3.1 together with the rules $Prefix$, $Choice$, Inv, $Encap$, $Abs_{1,2}$, Exe, $Recv$, and Bro of Table 2.1 induce a CLTS with transitions of the form $t \xrightarrow{\beta} t'$, where $\beta \in \mathbb{C}^v(Loc) \times (NAct \cup IAct \cup \{\tau\})$. In these rules, $t \not\xrightarrow{(\mathcal{C},\ rcv(\mathfrak{m}))}$ denotes that there exists no t' such that $t \xrightarrow{(\mathcal{C}',\ rcv(\mathfrak{m}))} t'$ and $\mathcal{C}' \preccurlyeq \mathcal{C}$. Rules Int and $Sen_{1,2}$ are new in comparison with the rules of *CNT* framework. Table 3.1 replaces the rule $Inter_1$ of Table 2.1 by Snd and $Inter_2$ by Rcv_{1-3}. Furthermore, it overwrites rule Par for only unobservable and internal actions and extends application of the rule Exe to $(\mathcal{C}, \alpha)$-like actions by the rule Exe'.

Rule $Prefix'$ assigns an empty network constraint to each prefixed action, which may be accumulated by further constraints through application of rules $Rcv_{1,2}$ or $Sen_{1,2}$. We remark that the rule $Prefix$ of Table 2.1 is still valid for computed network terms. The rule Int indicates that a node progresses when the deployed process on the node performs an internal action. Interaction between the process t and its data-link layer is specified by the rule Snd and Rcv_{1-3}: when t broadcasts a message, it is delivered to the nodes in its transmission range

Table 3.1: Semantics of *RRBPT* operators.

$$\frac{}{\alpha.t \xrightarrow{(\{\},\alpha)} t} : \mathit{Prefix}' \qquad \frac{t_1 \xrightarrow{(\mathcal{C},\alpha)} t_1'}{\mathit{sense}(\ell,t_1,t_2) \xrightarrow{(\{?\rightsquigarrow\ell\}\cup\mathcal{C},\alpha)} t_1'} : \mathit{Sen}_1$$

$$\frac{t \xrightarrow{(\mathcal{C},\ rcv(\mathfrak{m}))}\!\!\!\!\!\!\not\quad}{[\![t]\!]_\ell \xrightarrow{(\mathcal{C}[\ell/?],\ nrcv(\mathfrak{m}))} [\![t]\!]_\ell} : \mathit{Rcv}_3 \qquad \frac{t_2 \xrightarrow{(\mathcal{C},\alpha)} t_2'}{\mathit{sense}(\ell,t_1,t_2) \xrightarrow{(\{?\not\rightsquigarrow\ell\}\cup\mathcal{C},\alpha)} t_2'} : \mathit{Sen}_2$$

$$\frac{t \xrightarrow{(\mathcal{C},\ rcv(\mathfrak{m}))} t'}{[\![t]\!]_\ell \xrightarrow{(\mathcal{C}[\ell/?]\cup\{?\rightsquigarrow\ell\},\ nrcv(\mathfrak{m}))} [\![t']\!]_\ell} : \mathit{Rcv}_1 \qquad \frac{t \xrightarrow{(\mathcal{C},\ i)} t'}{[\![t]\!]_\ell \xrightarrow{(\mathcal{C},\ i)} [\![t']\!]_\ell} : \mathit{Int}$$

$$\frac{t \xrightarrow{(\mathcal{C},\ snd(\mathfrak{m}))} t'}{[\![t]\!]_\ell \xrightarrow{(\mathcal{C}[\ell/?],\ nsnd(\mathfrak{m},\ell))} [\![t']\!]_\ell} : \mathit{Snd} \qquad \frac{t_1 \xrightarrow{(\mathcal{C},\iota)} t_1'}{t_1 \parallel t_2 \xrightarrow{(\mathcal{C},\iota)} t_1' \parallel t_2} : \mathit{Par}$$

$$\frac{t \xrightarrow{(\mathcal{C},\alpha)} t'}{t \xrightarrow{(\mathcal{C}',\alpha)} t'} :\ \mathit{Exe}',\ \mathcal{C}' \preccurlyeq \mathcal{C} \qquad \frac{t \xrightarrow{(\mathcal{C},\ rcv(\mathfrak{m}))} t'}{[\![t]\!]_\ell \xrightarrow{(\mathcal{C}[\ell/?]\cup\{?\not\rightsquigarrow\ell\},\ nrcv(\mathfrak{m}))} [\![t]\!]_\ell} : \mathit{Rcv}_2$$

disregarding their readiness. Rcv_1 specifies that a process t with an enabled receive action can perform it successfully if it has a link to a sender (not currently known). In contrast, an enabled receive action cannot be performed if the node is disconnected from the sender (not currently known). However, to make the node *input-enabled* and consequently *non-blocking*, the node still performs its receive action but its state is unchanged, as explained in Rcv_2. If a protocol does not have any enabled receive action $rcv(\mathfrak{m})$ for the network constraint $\mathcal{C}$, then receiving the message has no effect on the node behavior, as explained by Rcv_3. Consequently, this rule makes nodes *input-enabled*, meaning that a node not ready to receive a message will drop it. Therefore, $[\![P]\!]_A$ has a $(\{\}, nrcv(rep))$-transition by application of this rule.

Rules $\mathit{Sen}_{1,2}$ explain the behavior of the sense operator. In case there is a link to the node with the address ℓ from the node that is running the sense operator, and currently its address is unknown, then it behaves like t_1; in case this link is not present, it behaves like t_2. Therefore, the link status is combined with the network constraint $\mathcal{C}$ generated by its first or second term argument, as given by $\mathit{Sen}_{1,2}$ respectively. For instance, $\mathit{sense}(B, rcv(req).0, snd(req).0)$ generates two transitions: by *Prefix*′ and Sen_1, it generates a $(\{? \rightsquigarrow B\},\ rcv(req))$-transition, and by *Prefix*′ and Sen_2, a $(\{? \not\rightsquigarrow B\},\ snd(req))$-transition.

In rules *Snd* and Rcv_1, the network constraint $\mathcal{C}$ may have unknown addresses due to the sensing operators, which are replaced by the address of development operator, i.e., $\mathcal{C}[\ell/?]$. Therefore, by application of *Prefix*′, Sen_1, and

Rcv_1, the following transition results:

$$[\![sense(B, rcv(req).0, snd(req).0)]\!]_A \xrightarrow{(\{A \rightsquigarrow B,\ ? \rightsquigarrow A\},\ nrcv(req))} [\![0]\!]_A.$$

We remark that the rule Exe' is essential for the rule Rcv_3. Without such a rule, Rcv_3 would derive a self-loop with the label $(\{B \not\rightsquigarrow A\}, rcv(req))$ for $[\![Q]\!]_B$ as $Q \xrightarrow{(\{B \not\rightsquigarrow A\}, rcv(req))}$. While $[\![Q]\!]_B$ also has a $(\{? \rightsquigarrow B\}, nrcv(req))$-transition leading to the behavior $[\![y]\!]_B$, where $y \equiv snd(rep).Q$, by the application of Rcv_1. For topologies that are in common between the two network constraints, $\{? \rightsquigarrow B\}$ and $\{B \not\rightsquigarrow A\}$, $[\![Q]\!]_B$ have a non-deterministic behavior on receiving req (it may process it or drop it). By Exe', Q induces a $(\{B \not\rightsquigarrow A, ? \rightsquigarrow B\}, rcv(req))$-transition leading to the behavior y. Finally, $\{B \not\rightsquigarrow A, ? \rightsquigarrow B\} \preceq \{B \not\rightsquigarrow A\}$ makes that the premise $Q \xrightarrow{(\{B \not\rightsquigarrow A\}, rcv(req))}$ does not hold, and hence the behavior of $[\![Q]\!]_B$ is deterministic on receive actions.

The SOS rules of receive actions and parallel composition have been modified compared to the lossy framework [73, 74]: Rcv_{1-3} specify the locality of a receiver node with respect to the sender (connected, disconnected, unknown). Par prevents evolution of sub-networks on network actions and enforces all nodes to specify their localities with respect to the sender before evolving the whole network via $Recv$ or Bro rules. By Par, a process evolves when a subprocess evolves by performing only an internal or silent action. The symmetric counterpart of the rule Par holds.

For instance, the MANET $[\![sense(B, rcv(req).0, snd(req).0)]\!]_A \parallel [\![snd(req).0]\!]_B$ can generate the $(\{B \rightsquigarrow A\},\ nsnd(req, B))$ transition induced by the deduction tree below, where $x \equiv sense(B, rcv(req).0, snd(req).0)$ and $z \equiv snd(req).0$:

$$\dfrac{\dfrac{\dfrac{}{rcv(req).0 \xrightarrow{(\{\},\ rcv(req))} 0}:Sen_1}{x \xrightarrow{(\{? \rightsquigarrow B\},\ rcv(req))} 0}:Rcv_1}{[\![x]\!]_A \xrightarrow{(\{A \rightsquigarrow B,\ ? \rightsquigarrow A\},\ nrcv(req))} [\![0]\!]_A}:Bro, Snd$$
$$[\![x]\!]_A \parallel [\![z]\!]_B \xrightarrow{(\{B \rightsquigarrow A, A \rightsquigarrow B\},\ nsnd(req, B))} [\![0]\!]_A \parallel [\![0]\!]_B$$

The derived CLTS of our running example $\partial_{Msg}([\![P]\!]_A \parallel [\![Q]\!]_B)$ is given in Fig. 3.3.

3.3 Rooted Branching Reliable Computed Network Bisimilarity

It can be easily shown that rooted branching computed network bisimilarity does not constitute a congruence with respect to the *RRBPT* operators. We still want that a receiving MANET (after its first action) be equivalent to deadlock. In this

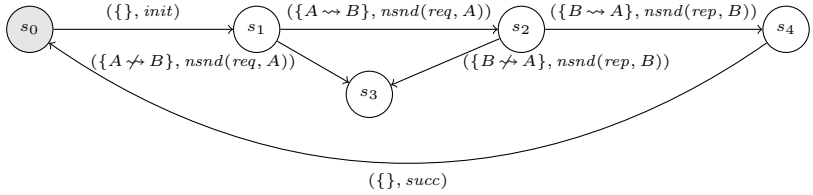


Figure 3.3: The CLTS associated to $\partial_{Msg}([\![P]\!]_A \parallel [\![Q]\!]_B)$.

setting, still $[\![0]\!]_A \simeq_b [\![rcv(\mathfrak{m}).0]\!]_A$, but $[\![0]\!]_A \parallel [\![snd(\mathfrak{m}).0]\!]_B \not\simeq_b [\![rcv(\mathfrak{m}).0]\!]_A \parallel [\![snd(\mathfrak{m}).0]\!]_B$, since by application of $Rcv_{1,2}$, Snd, and Bro:

$$[\![rcv(\mathfrak{m}).0]\!]_A \parallel [\![snd(\mathfrak{m}).0]\!]_B \xrightarrow{(\{B \not\rightsquigarrow A\}, nsnd(\mathfrak{m},B))} [\![rcv(\mathfrak{m}).0]\!]_A \parallel [\![0]\!]_B$$
$$[\![rcv(\mathfrak{m}).0]\!]_A \parallel [\![snd(\mathfrak{m}).0]\!]_B \xrightarrow{(\{B \rightsquigarrow A\}, nsnd(\mathfrak{m},B))} [\![0]\!]_A \parallel [\![0]\!]_B$$

while by application of Rcv_3, Snd, Bro:

$$[\![0]\!]_A \parallel [\![snd(\mathfrak{m}).0]\!]_B \xrightarrow{(\{\}, nsnd(\mathfrak{m},B))} [\![0]\!]_A \parallel [\![0]\!]_B$$

which cannot be matched to any transition of $[\![rcv(\mathfrak{m}).0]\!]_A \parallel [\![snd(\mathfrak{m}).0]\!]_B$ according to the second condition of Definition 2.4. However, we observe that the $(\{\}, nsnd(\mathfrak{m}, B))$-transition can be matched to the transition sets of actions $(\{B \not\rightsquigarrow A\}, nsnd(\mathfrak{m}, B))$ and $(\{B \rightsquigarrow A\}, nsnd(\mathfrak{m}, B))$, as the network constraints $\{B \not\rightsquigarrow A\}$ and $\{B \rightsquigarrow A\}$ provide a partitioning of $\{\}$ while the resulting states of their corresponding transitions are equivalent. Thus, we revise our Definition 2.4 by generalizing its second condition.

Intuitively, two MANETs are equivalent if they have the same observable behaviors for all possible underlying topologies. In the lossy setting, the observable behaviors exclude receive actions, as the node $[\![rcv(a).snd(a).0]\!]_A$ can be distinguished from $[\![rcv(a).0]\!]_A$ due to its capability to send a after its receipt. However, the capability of receiving messages implicitly defines a restriction on the underlying topology. For instance, the sending action $snd(a)$ in $[\![rcv(a).snd(a).0]\!]_A$ is only possible if the node in question was previously connected to a sender and successfully received a. Thus to distinguish $[\![rcv(a).snd(a).0]\!]_A$ from $[\![snd(a).0]\!]_A$, receive actions are included in the observables in the reliable setting. Furthermore, as dropping a message may have the same effect as its processing (as explained above), a transition cannot be matched in the same way as in Definition 2.4 and it may be matched to multiple transitions.

A partitioning of a network constraint $\mathcal{C}$ consists of network constraints $\mathcal{C}_1$, $\ldots$, $\mathcal{C}_n$ such that $\forall i, j \leq n\,(i \neq j \Rightarrow \Gamma(\mathcal{C}_i) \cap \Gamma(\mathcal{C}_j) = \emptyset) \wedge \bigcup_{k=1}^{n} \Gamma(\mathcal{C}_k) = \Gamma(\mathcal{C})$ where $\Gamma(\mathcal{C})$ denotes the set of topologies represented by the network constraint $\mathcal{C}$. Recall that $\langle(\mathcal{C}, \eta)\rangle$ is the counterpart of $(\mathcal{C}, \eta)$, meaning that it denotes $(\mathcal{C}, \eta)$ or $(\mathcal{C}[\ell/?], \eta[\ell/?])$ if η is of the form $nsnd(\mathfrak{m}, ?)$. We use $\langle\mathcal{C}\rangle$ to denote $\mathcal{C}$ or $\mathcal{C}[\ell/?]$.

Definition 3.1. A binary relation $\mathcal{R}$ on computed network terms is a branching reliable computed network simulation if $t_1 \mathcal{R} t_2$ and $t_1 \xrightarrow{(\mathcal{C},\eta)} t_1'$ implies that either:

- η is a τ action, and $t_1' \mathcal{R} t_2$; or
- there are $s_1'', \ldots, s_k''$ and $s_1', \ldots, s_k'$ for some $k > 0$ such that $\forall i \leq k (t_2 \Rightarrow s_i'' \xrightarrow{\langle\langle \mathcal{C}_i,\eta \rangle\rangle} s_i'$, with $t_1 \mathcal{R} s_i''$ and $t_1' \mathcal{R} s_i'$), and $\langle \mathcal{C}_1 \rangle, \ldots, \langle \mathcal{C}_k \rangle$ constitute a partitioning of $\langle \mathcal{C} \rangle$.

$\mathcal{R}$ is a branching reliable computed network bisimulation if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are branching reliable computed network simulations. Two terms t_1 and t_2 are branching reliable computed network bisimilar, denoted by $t_1 \simeq_{br} t_2$, if $t_1 \mathcal{R} t_2$ for some branching reliable computed network bisimulation relation $\mathcal{R}$.

Theorem 3.2. *Branching reliable computed network bisimilarity is an equivalence.*

Trivially $(t_1 \simeq_b t_2) \Rightarrow (t_1 \simeq_{br} t_2)$.

Definition 3.3. Two terms t_1 and t are *rooted branching reliable computed network bisimilar*, written $t_1 \simeq_{rbr} t_2$, if:

- $t_1 \xrightarrow{(\mathcal{C},\eta)} t_1'$ implies there is a t_2' such that $t_2 \xrightarrow{\langle\langle \mathcal{C},\eta \rangle\rangle} t_2'$ and $t_1' \simeq_{br} t_2'$;
- $t_2 \xrightarrow{(\mathcal{C},\eta)} t_2'$ implies there is a t_1' such that $t_1 \xrightarrow{\langle\langle \mathcal{C},\eta \rangle\rangle} t_1'$ and $t_1' \simeq_{br} t_2'$.

Corollary 3.4. *Rooted branching reliable computed network bisimilarity is an equivalence.*

See Section A.1 for the proof of Theorem 3.2. Corollary 3.4 is an immediate result of Theorem 3.2 and Definition 3.3.

Theorem 3.5. *Rooted branching reliable computed network bisimilarity is a congruence for RRBPT operators.*

See Section A.2 for the proof.

3.4 Axioms

To provide a sound and complete axiomatization for closed *RRBPT* terms with respect to rooted branching reliable computed network bisimilarity, the framework should be extended with the computed network terms, i.e., $(\mathcal{C}, \eta).t$ which expresses that action η is possible for topologies belonging to $\mathcal{C}$. This prefix operator is helpful to transform protocol send/receive actions into their corresponding network ones (see axioms $Dep_{1,2}$ in Section 2.3.4). Furthermore, it

should contain the operators *left merge* ($\between$) and *communication merge* ($|$) to axiomatize parallel composition. Note that the interleaving semantics for parallel composition is only valid for internal and unobservable actions (see SOS rule Par). Furthermore, we need the conditional and sum operators to enforce input-enabledness. To axiomatize the behavior of nodes while being input-enabled, we also exploit two novel auxiliary operators.

RRBPT is extended with new operators and called *Reliable Computed Network Process Theory* (*RCNT*). Its syntax contains:

$$\begin{aligned} t ::= \; & 0 \mid \beta.t \mid t + t \mid \mathfrak{A}\,, \mathfrak{A} \stackrel{def}{=} t \mid \; t \mid t \mid t \between t \mid t \parallel t \mid rec\mathfrak{A} \cdot t \\ & sense(\ell, t, t) \mid (\nu\ell)t \mid \tau_{\mathfrak{m}}(t) \mid \partial_{\mathfrak{m}}(t) \mid \ell : t : t \mid \mathcal{C} \rhd t \mid [\![t]\!]_{\ell} \end{aligned}$$

The prefix operator in $\beta.t$ again denotes a process which performs β and then behaves as t. The action β can now be of two types: either an internal action or a send/receive action $snd(\mathfrak{m})/rcv(\mathfrak{m})$, denoted by α, or actions of the form $(\mathcal{C}, nrcv(\mathfrak{m}))$, $(\mathcal{C}, nsnd(\mathfrak{m}, \ell))$ and $(\mathcal{C}, \tau)$, denoted by $(\mathcal{C}, \eta)$, where the first two actions are called the network receive and send actions, respectively. The new operator $\ell : t_1 : t_2$, so-called *local deployment*, defines the behavior of process t_2 deployed at the network address ℓ while it only considers the input-enabledness feature with regard to the behavior of t_1. In cases that it should drop a message (i.e., processing the message has not been defined by t_2), it behaves as t_1. This operator is helpful to axiomatize the behavior of the deployment operator in the reliable setting. To axiomatize the behavior of the $sense$ operator, the framework is extended with the *topology restriction* operator $\mathcal{C} \rhd t$ which restricts the behavior of t by taking restrictions of $\mathcal{C}$ into account.

Due to the input-enabledness feature of nodes, their behavior is recursive: upon receiving a message for which no receive action has been defined, a node drops the message. To this aim, we exploit the recursion operator $rec\mathfrak{A} \cdot t$, which specifies the *solution* of the process name $\mathfrak{A}$, defined by the equation $\mathfrak{A} \stackrel{def}{=} t$. As we are interested in equations with exactly one solution, we define a guardedness criterion for network names similar to *CNT* (see Section 2.3.4). A free occurrence of a network name A in t is called *guarded* if this occurrence is in the scope of an action prefix operator (not $(\mathcal{C}, \tau)$ prefix) and not in the scope of an abstraction operator [9]; in other words, there is a subterm $(\mathcal{C}, \eta).t'$ in t such that $\eta \neq \tau$, and $\mathfrak{A}$ occurs in t'. $\mathfrak{A}$ is *(un)guarded* in t if (not) every free occurrence of $\mathfrak{A}$ in t is guarded. A *RCNT* term t is *guarded* if for every subterm $rec\mathfrak{A} \cdot t'$, $\mathfrak{A}$ is guarded in t'. This guardedness criterion ensures that any guarded recursive term has a unique solution.

A term is grammatically well-defined if its processes deployed at a network address through either a network or local deployment operator, are only defined by action prefix, choice, sense, and process names.

The SOS rules of the new operators are given in Table 3.2 together with the rules $Sync_{1,2}$ of Table 2.1. In these rules, $t \stackrel{nrcv(\mathfrak{m})}{\not\longrightarrow}$ denotes that there exists no

t' such that $t \xrightarrow{(\mathcal{C}',\ nrcv(\mathfrak{m}))} t'$ for some network constraint $\mathcal{C}'$. The behavior of the local deployment operator is almost similar to the deployment operator. Its rules $Inter'_1$ and $Inter'_2$ are almost the same as Snd and Rcv_1, respectively. However, it substitutes $Inter'_3$ for Rcv_2 by which it only adds transitions containing the disconnectivity pair $?\not\rightarrow\ell$ for those possible receive actions of t_2 (generated by Rcv_1). Rules $Choice'$, Inv', and $Sen'_{1,2}$ are also the same as $Choice$, Inv, and $Sen_{1,2}$, receptively. The difference between the rules of the deployment and local deployment is that the behavior of the deployment operator is derived regarding the behavior of the deployed protocol by the rules Snd and Rcv_{1-3} while the behavior of the local deployment is defined in terms of the structure of the deployed protocol. Rules $Sen'_{3,4}$ make the behavior of $sense(\ell', t_1, t_2)$ input-enabled toward receive actions that are possible by t_1 but not t_2 and vice versa. The constraints of the topology restriction operator $\mathcal{C} \rhd t$ is added to the behaviors of t as explained by the rule TR.

The main differences of extended *RCNT* with *CNT* are that its deployed nodes are input-enabled and its communication primitive is reliable. We use the notation $\sum_{m\in M} t$ to define $t[m_1/\mathfrak{m}] + \ldots + t[m_k/\mathfrak{m}]$, where $M = \{m_1, \ldots, m_k\}$. Furthermore, $if(b, t_1, t_2)$ behaves as t_1 if the condition b holds and otherwise as t_2.

The axioms regarding the deployment, left and communication merge, and parallel operators are given in Table 3.3. The axioms Br, $LM_{2,3}$ and S_{1-4} are standard (cf. [111]). The axiom Ch_5 denotes that a network send action whose sender address is unknown can be removed if its counterpart action exists. The axiom Ch_6 explains that a more liberal network constraint allows more behavior. Axioms Dep_{0-7}, $LM'_{1,2}$, and $TRes_{1-5}$ are new in comparison with the lossy setting of *CNT*. The axiom $(\mathcal{C}, \eta).t_1 \between t_2 = (\mathcal{C}, \eta).(t_1 \parallel t_2)$ has been replaced by $LM'_{1,2}$ which only allow internal or unobservable actions of the left operand to be performed.

To axiomatize the behavior of a node considering the input-enabledness feature, we need to find the messages that it cannot currently respond to and then add a summand which receives those message without processing them. To this aim, axiom Dep_0 expresses the behavior of $[\![t]\!]_\ell$ as a recursive specification which drops messages that it does not handle with the help of the auxiliary function $Message(t, \mathcal{S})$, and the behavior of t with the help of the local deployment operator $\ell : \mathfrak{Q} : t$. The function $Message(t, \mathcal{S})$ returns the set of messages that can be

Table 3.2: Semantics of the new operators of *RCNT*

$$\frac{}{\ell : t_1 : snd(\mathfrak{m}).t_2 \xrightarrow{(\{\},\ nsnd(\mathfrak{m},\ell))} [\![t_2]\!]_\ell} : Inter'_1$$

$$\frac{}{\ell : t_1 : rcv(\mathfrak{m}).t_2 \xrightarrow{(\{?\rightsquigarrow\ell\},\ nrcv(\mathfrak{m}))} [\![t_2]\!]_\ell} : Inter'_2$$

$$\frac{}{\ell : t_1 : rcv(\mathfrak{m}).t_2 \xrightarrow{(\{?\not\rightsquigarrow\ell\},\ nrcv(\mathfrak{m}))} t_1} : Inter'_3 \quad \frac{t \xrightarrow{(\mathcal{C}',\eta)} t'}{\mathcal{C} \rhd t \xrightarrow{(\mathcal{C}'\cup\mathcal{C},\eta)} t'} : TR$$

$$\frac{\ell : t_3 : t_i \xrightarrow{(\mathcal{C},\ \eta)} t'_i \quad i \in \{1,2\}}{\ell : t_3 : t_1 + t_2 \xrightarrow{(\mathcal{C},\eta)} t'_i} : Choice' \quad \frac{t[rec\mathfrak{A} \cdot t/\mathfrak{A}] \xrightarrow{(\mathcal{C},\eta)} t'}{rec\mathfrak{A} \cdot t \xrightarrow{(\mathcal{C},\eta)} t'} : Rec$$

$$\frac{\ell : t_1 : t_2 \xrightarrow{(\mathcal{C},\ \eta)} t'_2}{\ell : t_1 : \mathfrak{A} \xrightarrow{(\mathcal{C},\eta)} t'_2} : Inv',\ \mathfrak{A} \stackrel{def}{=} t_2 \qquad \frac{t_1 \xrightarrow{(\mathcal{C},\iota)} t'_1}{t_1 \between t_2 \xrightarrow{(\mathcal{C},\iota)} t'_1 \parallel t_2} : LExe$$

$$\frac{\ell : t_3 : t_1 \xrightarrow{(\mathcal{C},\eta)} t'_1}{\ell : t_3 : sense(\ell', t_1, t_2) \xrightarrow{(\{\ell\rightsquigarrow\ell'\}\cup\mathcal{C},\eta)} t'_1} : Sen'_1$$

$$\frac{\ell : t_3 : t_2 \xrightarrow{(\mathcal{C},\eta)} t'_2}{\ell : t_3 : sense(\ell', t_1, t_2) \xrightarrow{(\{\ell\not\rightsquigarrow\ell'\}\cup\mathcal{C},\eta)} t'_2} : Sen'_2$$

$$\frac{\ell : t_3 : t_1 \xrightarrow{nrcv(\mathfrak{m})} \quad \ell : t_3 : t_2 \xrightarrow{(\mathcal{C},\ nrcv(\mathfrak{m}))} t'_2}{\ell : t_3 : sense(\ell', t_1, t_2) \xrightarrow{(\{\ell\rightsquigarrow\ell'\},\ nrcv(\mathfrak{m}))} t_3} : Sen'_3$$

$$\frac{\ell : t_3 : t_1 \xrightarrow{(\mathcal{C},\ nrcv(\mathfrak{m}))} t'_1 \quad \ell : t_3 : t_2 \xrightarrow{nrcv(\mathfrak{m})}}{\ell : t_3 : sense(\ell', t_1, t_2) \xrightarrow{(\{\ell\not\rightsquigarrow\ell'\},\ nrcv(\mathfrak{m}))} t_3} : Sen'_4$$

currently processed by t and is defined using structural induction:

$$
\begin{array}{l}
Message(0, \mathcal{S}) = \emptyset \\
Message(i.t, \mathcal{S}) = \emptyset,\ i \in IAct \\
Message(snd(\mathfrak{m}).t, \mathcal{S}) = \emptyset \\
Message(rcv(\mathfrak{m}).t, \mathcal{S}) = \{\mathfrak{m}\} \\
Message(t_1 + t_2, \mathcal{S}) = Message(t_1, \mathcal{S}) \cup Message(t_2, \mathcal{S}) \\
Message(sense(\ell, t_1, t_2), \mathcal{S}) = Message(t_1, \mathcal{S}) \cup Message(t_2, \mathcal{S}) \\
Message(\mathfrak{A}, \mathcal{S}) = Message(t, \mathcal{S} \cup \{\mathfrak{A}\}),\ \mathfrak{A} \notin \mathcal{S}, \mathfrak{A} \stackrel{def}{=} t \\
Message(\mathfrak{A}, \mathcal{S}) = \emptyset,\ \mathfrak{A} \in \mathcal{S}
\end{array}
$$

where $\mathcal{S}$ keeps track of process names whose right-hand definitions have been examined. We remark that Dep_0 extends the deployment behavior of the lossy setting with the input-enabledness feature with the help of operator $\ell : \mathfrak{Q} : t$. The axioms Dep_{1-7} specify the behavior of the operator $\ell : t_1 : t_2$. Axiom Dep_1 defines the interaction between the network and data link layers. The protocol send action (at the network layer) is transformed into its network version (at the data link layer). Axiom Dep_2 indicates that when ℓ is connected to a sender (which is unknown yet), the receive action is successful and its behavior proceeds as $[\![t]\!]_\ell$. Otherwise, the receive action is unsuccessful and its behavior is defined by t'. Axioms $Dep_{3,4,5}$ express the effect of the local deployment on choice, deadlock, and process names, respectively while axioms $Dep_{6,7}$ define its effect on the prefixed internal actions and $sense$ operator, respectively.

The behavior of the topology restriction operator is defined by the axioms $TRes_{1-5}$ in Table 3.3. Axiom $TRes_1$ considers the restrictions of $\mathcal{C}_1$ by integrating its restrictions with $\mathcal{C}_2$ in the computed network term $(\mathcal{C}_2, \eta).t$ if $\mathcal{C}_1 \cup \mathcal{C}_2$ is well-formed. Axiom $TRes_2$ defines that topology restriction can be distributed over the choice operator. Axiom $TRes_3$ expresses that the topology restriction operator can be moved inside and outside of a recursion operator. Axioms $TRes_{4,5}$ explain that the topology restriction operator has no effect on a process name and deadlock, respectively.

For instance, the behavior of the MANET $[\![sense(B, rcv(req).0, snd(req).0)]\!]_A$, where $Msg = \{req, rep\}$, can be simplified as:

$$
\begin{array}{l}
[\![sense(B, rcv(req).0, snd(req).0)]\!]_A =^{Dep_0} \\
\quad rec\mathfrak{Q} \cdot (\{\}, nrcv(rep)).\mathfrak{Q} + A : \mathfrak{Q} : sense(B, rcv(req).0, snd(req).0) =^{Dep_8} \\
\quad rec\mathfrak{Q} \cdot (\{\}, nrcv(rep)).\mathfrak{Q} + (\{B \not\rightsquigarrow A\}, nrcv(req)).\mathfrak{Q} + \\
\quad\quad \{B \rightsquigarrow A\} \rhd A : \mathfrak{Q} : rcv(req).0 + \{B \rightsquigarrow A\} \rhd A : \mathfrak{Q} : snd(req).0 =^{Dep_{1,2}} \\
\quad rec\mathfrak{Q} \cdot (\{\}, nrcv(rep)).\mathfrak{Q} + (\{B \not\rightsquigarrow A\}, nrcv(req)).\mathfrak{Q} + \\
\quad\quad \{B \rightsquigarrow A\} \rhd ((\{? \rightsquigarrow A\}, nrcv(req)).[\![0]\!]_A + (\{? \not\rightsquigarrow A\}, nrcv(req)).\mathfrak{Q}) + \\
\quad\quad\quad \{B \not\rightsquigarrow A\} \rhd (\{\}, nsnd(req, A)).[\![0]\!]_A =^{TRes_{1,2}} \\
\quad rec\mathfrak{Q} \cdot ((\{\}, nrcv(rep)).\mathfrak{Q} + (\{B \not\rightsquigarrow A\}, nrcv(req)).\mathfrak{Q} +
\end{array}
$$

Table 3.3: Axioms for the choice, deployment, left and communication merge, and parallel operators. The sets M_1 and M_2 denote $Message(t_2,\emptyset) \setminus Message(t_1,\emptyset)$ and $Message(t_1,\emptyset) \setminus Message(t_2,\emptyset)$ respectively.

$$
\begin{array}{ll}
Dep_0 & [\![t]\!]_\ell = rec\mathfrak{Q}\cdot\sum_{\mathfrak{m}'\notin Message(t,\emptyset)}(\{\},nrcv(\mathfrak{m}')).\mathfrak{Q}+\ell:\mathfrak{Q}:t \\
Dep_2 & \ell:t':rcv(\mathfrak{m}).t=(\{?\not\rightsquigarrow\ell\},nrcv(\mathfrak{m})).t'+(\{?\rightsquigarrow\ell\},nrcv(\mathfrak{m})).[\![t]\!]_\ell \\
Dep_7 & \ell:t_3:sense(\ell',t_1,t_2)=\sum_{\mathfrak{m}'\in M_1}(\{\ell\rightsquigarrow\ell'\},nrcv(\mathfrak{m}')).t_3 \\
 & +\sum_{\mathfrak{m}'\in M_2}(\{\ell\not\rightsquigarrow\ell'\},nrcv(\mathfrak{m}')).t_3+\{\ell\rightsquigarrow\ell'\}\rhd\ell:t_3:t_1 \\
 & +\{\ell\not\rightsquigarrow\ell'\}\rhd\ell:t_3:t_2
\end{array}
$$

$$
\begin{array}{llll}
Dep_1 & \ell:t':snd(\mathfrak{m}).t=(\{\},nsnd(\mathfrak{m},\ell)).[\![t]\!]_\ell & Dep_6 & \ell:t':i.t=(\{\},i).[\![t]\!]_\ell \\
Dep_3 & \ell:t_3:t_1+t_2=\ell:t_3:t_1+\ell:t_3:t_2 & Dep_4 & \ell:t:0=0 \\
Dep_5 & \ell:t':\mathfrak{A}=\ell:t':t,\ \mathfrak{A}\stackrel{def}{=}t & &
\end{array}
$$

$$
\begin{array}{llll}
TRes_1 & \mathcal{C}_1\rhd(\mathcal{C}_2,\eta).t=(\mathcal{C}_1\cup\mathcal{C}_2,\eta).t,\ \text{if}\ \mathcal{C}_1\cup\mathcal{C}_2\in\mathbb{C}^v(Loc) & & \\
TRes_2 & \mathcal{C}\rhd(t_1+t_2)=(\mathcal{C}\rhd t_1)+(\mathcal{C}\rhd t_2) & TRes_4 & \mathcal{C}\rhd\mathfrak{A}=\mathfrak{A} \\
TRes_3 & \mathcal{C}\rhd rec\mathfrak{A}\cdot t=rec\mathfrak{A}\cdot(\mathcal{C}\rhd t) & TRes_5 & \mathcal{C}\rhd 0=0
\end{array}
$$

$$
\begin{array}{llll}
Br & t_1\parallel t_2=t_1\between t_2+t_2\between t_1+t_1\mid t_2 & S_1 & t_1\mid t_2=t_2\mid t_1 \\
LM_1' & (\mathcal{C},\eta).t_1\between t_2=0,\ \eta\notin IAct\cup\{\tau\} & S_2 & (t_1+t_2)\mid t_3=t_1\mid t_3+t_2\mid t_3 \\
LM_2 & (t_1+t_2)\between t_3=t_1\between t_3+t_2\between t_3 & S_3 & 0\mid t=0 \\
LM_3 & 0\between t=0 & S_4 & (\mathcal{C},\eta).t_1\mid t_2=0,\ \eta\in IAct\cup\{\tau\} \\
LM_2' & (\mathcal{C},\eta).t_1\between t_2=(\mathcal{C},\eta).(t_1\parallel t_2),\ \eta\in IAct\cup\{\tau\} & &
\end{array}
$$

$$
\begin{array}{ll}
Sync_1 & (\mathcal{C}_1,nsnd(\mathfrak{m}_1,\ell)).t_1\mid(\mathcal{C}_2,nrcv(\mathfrak{m}_2)).t_2= \\
 & \quad if((\mathfrak{m}_1=\mathfrak{m}_2),(\mathcal{C}_1\cup\mathcal{C}_2[\ell/?],nsnd(\mathfrak{m}_1,\ell)).t_1\parallel t_2,0) \\
Sync_2 & (\mathcal{C}_1,nrcv(\mathfrak{m}_1)).t_1\mid(\mathcal{C}_2,nrcv(\mathfrak{m}_2)).t_2= \\
 & \quad if((\mathfrak{m}_1=\mathfrak{m}_2),(\mathcal{C}_1\cup\mathcal{C}_2,nrcv(\mathfrak{m}_1)).t_1\parallel t_2,0) \\
Sync_3 & (\mathcal{C}_1,nsnd(\mathfrak{m}_1,\ell_1)).t_1\mid(\mathcal{C}_2,nsnd(\mathfrak{m}_2,\ell_2)).t_2=0
\end{array}
$$

$$
\begin{array}{l}
(\{B\rightsquigarrow A,?\rightsquigarrow A\},nrcv(req)).[\![0]\!]_A+ \\
\quad(\{B\rightsquigarrow A,?\not\rightsquigarrow A\},nrcv(req)).\mathfrak{Q}+(\{B\not\rightsquigarrow A\},nsnd(req,A)).[\![0]\!]_A)
\end{array}
$$

Furthermore, the following should be added to the axioms:

$$
\begin{array}{ll}
T_3 & (\mathcal{C}',\eta).((\mathcal{C}_1,\eta).t+(\mathcal{C}_2,\eta).t+t')=(\mathcal{C}',\eta).((\mathcal{C},\eta).t+t') \\
 & \quad\text{iff}\ \exists\ell,\ell'\in Loc,\exists\mathcal{C}\in\mathbb{C}^v(Loc)\cdot(\mathcal{C}_1=\mathcal{C}\cup\{\ell\rightsquigarrow\ell'\}\ \wedge\ \mathcal{C}_2=\mathcal{C}\cup\{\ell\not\rightsquigarrow\ell'\}
\end{array}
$$

which accumulates the network constraints that constitute a partitioning.

It is not hard to see that the axioms Ch_{1-6} of Table 2.2, Table 3.3, Table 2.4 extended with the axiom T_3 while T_1 has been excluded, and Table 2.5 (while η_τ in the axiom $WUng_2$ only implies to τ) provide a sound axiomatization of *RCNT*. This can be checked by verifying soundness for each axiom individually. Furthermore, our axiomatization is also ground-complete for terms with a finite-state CLTS, but not for infinite-state CLTSs. See sections A.3 and A.4 for the proofs of the soundness and completeness of our axiomatization respectively.

3.5 Case Study: a Simple Routing Protocol

In MANETs, nodes communicate through others via a multi-hop communication. Hence, nodes act as routers to make the communication possible among not directly connected nodes. We illustrate the applicability of our axioms in the analysis of MANET protocols through a simple routing protocol inspired by the AODV protocol.

3.5.1 Protocol Specification

The protocol consists of three processes P, M, and Q, each specifying the behavior of a node as the source (that finds a route to a specific destination), middle node (that relays messages from the source to the destination), and destination. The description of these process are given in Figure 3.4.

$$
\begin{array}{rcl}
P & \stackrel{def}{=} & sense(B, snd(data_B).P, snd(req).P_1) \\
P_1 & \stackrel{def}{=} & [rcv(rep_C).P_2 + rcv(rep_B).P + snd(req).P_1] \\
P_2 & \stackrel{def}{=} & sense(C, rcv(error).P + snd(data_C).P_2, snd(req).P_1) \\
M & \stackrel{def}{=} & rcv(req).snd(req).M_1 \\
M_1 & \stackrel{def}{=} & rcv(rep_B).snd(rep_C).M_2 + snd(req).M_1 \\
M_2 & \stackrel{def}{=} & sense(B, rcv(data_C).snd(data_B).M_2, snd(error).snd(req).M_1) \\
Q & \stackrel{def}{=} & rcv(req).snd(rep_B).Q + rcv(data_B).deliver.Q
\end{array}
$$

Figure 3.4: The specification of processes P, M, and Q as a part of our simple routing protocol.

Process P, deployed at the address A, uses the neighbor discovery service of the data link layer to examine if it has a direct link to the destination with the address B. If it is connected, then it sends its data directly by broadcasting the message $data_B$; otherwise, it initiates the route discovery procedure by sending the message req, then behaving as P_1. This process waits until it receives a reply from a middle name with the address C or B. In the former case, it behaves as P_2 which indicates that A sends it data through C as long as C is connected to A. In the latter case, it behaves as P which indicates that A sends it data as long as B is directly connected to A.

Process M relays req messages to find a route to B and then behaves as M_1. This process waits until it receives a reply. To model waits with a timeout, it non-deterministically sends a request again. Upon receiving a reply from C it behaves as M_2, indicating that it relays data messages of A as long as it has a

link to C. Finally, process Q sends a reply upon receiving a request message and receives data messages.

To simplify the route maintenance procedure of AODV, the middle node takes advantage of the sensing operator when it behaves as M_2. Whenever it finds out that it has no link to C, it sends an error message to its upstream node, i.e., A, to inform it that its route to B through C is not valid. After sending and receiving the error message, they both initiate the route discovery procedure by sending a request message.

The network with the three nodes of a source, middle, and destination is specified by

$$\mathcal{N} \equiv \tau_{Msg}(\partial_{Msg}(\llbracket P \rrbracket_A \parallel \llbracket M \rrbracket_C \parallel \llbracket Q \rrbracket_B)).$$

Analyzing $(\nu A)(\nu B)(\nu C)\mathcal{N}$, whose network addresses have been abstracted away, reveals that it is rooted branching bisimilar to a process with livelock behavior[1]. Possibly a livelock occurs where data is not delivered to B as no route can found to B. Such behavior may be the result of a conceptual mistake in the protocol design or the impossibility of communication between A and B due to their disconnectivity. We propose a technique in Section 3.5.2 to discover only those faulty behaviors that are due to an incorrect protocol design.

The network $\partial_{Msg}(\llbracket P \rrbracket_A \parallel \llbracket M \rrbracket_C \parallel \llbracket Q \rrbracket_B)$ can be simplified as:

$$\begin{aligned}
&\partial_{Msg}(\llbracket P \rrbracket_A \parallel \llbracket M \rrbracket_C \parallel \llbracket Q \rrbracket_B) = \qquad (3.1)\\
&(\{A \rightsquigarrow B\}, nsnd(data_B, A)).\partial_{Msg}(\llbracket P \rrbracket_A \parallel \llbracket M \rrbracket_C \parallel \llbracket deliver.Q \rrbracket_B) +\\
&(\{A \not\rightsquigarrow B, A \rightsquigarrow C\}, nsnd(req, A)).\partial_{Msg}(\llbracket P_1 \rrbracket_A \parallel \llbracket snd(req).M_1 \rrbracket_C \parallel \llbracket Q \rrbracket_B) +\\
&(\{A \not\rightsquigarrow B, A \not\rightsquigarrow C\}, nsnd(req, A)).\partial_{Msg}(\llbracket P_1 \rrbracket_A \parallel \llbracket M \rrbracket_C \parallel \llbracket Q \rrbracket_B).
\end{aligned}$$

Next, we simplify $\partial_{Msg}(\llbracket P_1 \rrbracket_A \parallel \llbracket snd(req).M_1 \rrbracket_C \parallel \llbracket Q \rrbracket_B)$ as

$$\begin{aligned}
&\partial_{Msg}(\llbracket P_1 \rrbracket_A \parallel \llbracket snd(req).M_1 \rrbracket_C \parallel \llbracket Q \rrbracket_B) = \qquad (3.2)\\
&(\{A \rightsquigarrow B\}, nsnd(req, A)).\partial_{Msg}(\llbracket P_1 \rrbracket_A \parallel \llbracket snd(req).M_1 \rrbracket_C \parallel \llbracket snd(rep_B).Q \rrbracket_B) +\\
&(\{A \not\rightsquigarrow B\}, nsnd(req, A)).\partial_{Msg}(\llbracket P_1 \rrbracket_A \parallel \llbracket snd(req).M_1 \rrbracket_C \parallel \llbracket Q \rrbracket_B) +\\
&(\{C \rightsquigarrow B\}, nsnd(req, C)).\partial_{Msg}(\llbracket P_1 \rrbracket_A \parallel \llbracket M_1 \rrbracket_C \parallel \llbracket snd(rep_B).Q \rrbracket_B) +\\
&(\{C \not\rightsquigarrow B\}, nsnd(req, C)).\partial_{Msg}(\llbracket P_1 \rrbracket_A \parallel \llbracket M_1 \rrbracket_C \parallel \llbracket Q \rrbracket_B).
\end{aligned}$$

Now, we continue by extending $\partial_{Msg}(\llbracket P_1 \rrbracket_A \parallel \llbracket M_1 \rrbracket_C \parallel \llbracket snd(rep_B).Q \rrbracket_B)$:

$$\begin{aligned}
&\partial_{Msg}(\llbracket P_1 \rrbracket_A \parallel \llbracket M_1 \rrbracket_C \parallel \llbracket snd(rep_B).Q \rrbracket_B) =\\
&(\{\ \}, nsnd(req, A)).\partial_{Msg}(\llbracket P_1 \rrbracket_A \parallel \llbracket M_1 \rrbracket_C \parallel \llbracket snd(rep_B).Q \rrbracket_B) +\\
&(\{\ \}, nsnd(req, C)).\partial_{Msg}(\llbracket P_1 \rrbracket_A \parallel \llbracket M_1 \rrbracket_C \parallel \llbracket snd(rep_B).Q \rrbracket_B) +\\
&(\{B \rightsquigarrow A, C\}, nsnd(rep_B, B)).\partial_{Msg}(\llbracket P \rrbracket_A \parallel \llbracket snd(rep_C).M_2 \rrbracket_C \parallel \llbracket Q \rrbracket_B) +
\end{aligned}$$

[1] The network has been specified in the mCRL2 language [82], as it is explained in Section 3.6.2. The code and its derived transition system modulo branching bisimilarity by the mCRL2 toolset is available at `fghassemi.adhoc.ir/codeFI17.zip`

$$(\{B \rightsquigarrow A, B \not\rightsquigarrow C\}, nsnd(rep_B, B)).\partial_{Msg}([\![P]\!]_A \parallel [\![M_1]\!]_C \parallel [\![Q]\!]_B) +$$
$$(\{B \not\rightsquigarrow A, B \rightsquigarrow C\}, nsnd(rep_B, B)).\partial_{Msg}([\![P_1]\!]_A \parallel [\![snd(rep_C).M_2]\!]_C \parallel [\![Q]\!]_B).$$

By simplifying the term $\partial_{Msg}([\![P]\!]_A \parallel [\![snd(rep_C).M_2]\!]_C \parallel [\![Q]\!]_B)$, which indicates that A and C have found a direct route to B, we reach $\partial_{Msg}([\![P]\!]_A \parallel [\![M_2]\!]_C \parallel [\![Q]\!]_B)$:

$$\partial_{Msg}([\![P]\!]_A \parallel [\![snd(rep_C).M_2]\!]_C \parallel [\![Q]\!]_B) =$$
$$(\{\ \}, nsnd(rep_C, C)).\partial_{Msg}([\![P]\!]_A \parallel [\![M_2]\!]_C \parallel [\![Q]\!]_B) +$$
$$(\{A \rightsquigarrow B\}, nsnd(data_B, A)).\partial_{Msg}([\![P]\!]_A \parallel [\![snd(rep_C).M_2]\!]_C \parallel [\![deliver.Q]\!]_B) +$$
$$(\{A \not\rightsquigarrow B\}, nsnd(req, A)).\partial_{Msg}([\![P_1]\!]_A \parallel [\![snd(rep_C).M_2]\!]_C \parallel [\![Q]\!]_B).$$

By extending $\partial_{Msg}([\![P]\!]_A \parallel [\![M_2]\!]_C \parallel [\![Q]\!]_B)$, we have:

$$\partial_{Msg}([\![P]\!]_A \parallel [\![M_2]\!]_C \parallel [\![Q]\!]_B) =$$
$$(\{A \rightsquigarrow B\}, nsnd(data_B, A)).\partial_{Msg}([\![P]\!]_A \parallel [\![M_2]\!]_C \parallel [\![deliver.Q]\!]_B) +$$
$$(\{A \not\rightsquigarrow B\}, nsnd(req, A)).\partial_{Msg}([\![P_1]\!]_A \parallel [\![M_2]\!]_C \parallel [\![Q]\!]_B).$$

Finally extending $\partial_{Msg}([\![P_1]\!]_A \parallel [\![M_2]\!]_C \parallel [\![Q]\!]_B)$ results:

$$\partial_{Msg}([\![P_1]\!]_A \parallel [\![M_2]\!]_C \parallel [\![Q]\!]_B) =$$
$$(\{A \rightsquigarrow B\}, nsnd(req, A)).\partial_{Msg}([\![P_1]\!]_A \parallel [\![M_2]\!]_C \parallel [\![snd(rep_B).Q]\!]_B) +$$
$$(\{A \not\rightsquigarrow B\}, nsnd(req, A)).\partial_{Msg}([\![P_1]\!]_A \parallel [\![M_2]\!]_C \parallel [\![Q]\!]_B) +$$
$$(\{C \not\rightsquigarrow B\}, nsnd(error, C)).\partial_{Msg}([\![P_1]\!]_A \parallel [\![snd(req).M_1]\!]_C \parallel [\![Q]\!]_B).$$

The following scenario, found by above equations, is valid for a topology in which A has only a multi-hop link to B via C, but B has a direct link to A:

$$\partial_{Msg}([\![P]\!]_A \parallel [\![M]\!]_C \parallel [\![Q]\!]_B)$$
$$\xrightarrow{(\{A \not\rightsquigarrow B, A \rightsquigarrow C\}, nsnd(req, A))} \partial_{Msg}([\![P_1]\!]_A \parallel [\![snd(req).M_1]\!]_C \parallel [\![Q]\!]_B)$$
$$\xrightarrow{(\{C \rightsquigarrow B\}, nsnd(req, C))} \partial_{Msg}([\![P_1]\!]_A \parallel [\![M_1]\!]_C \parallel [\![snd(rep_B).Q]\!]_B)$$
$$\xrightarrow{(\{B \rightsquigarrow A, C\}, nsnd(rep_B, B))} \partial_{Msg}([\![P]\!]_A \parallel [\![snd(rep_C).M_2]\!]_C \parallel [\![Q]\!]_B)$$
$$\xrightarrow{(\{\ \}, nsnd(rep_C, C))} \partial_{Msg}([\![P]\!]_A \parallel [\![M_2]\!]_C \parallel [\![Q]\!]_B)$$
$$\xrightarrow{(\{A \not\rightsquigarrow B\}, nsnd(req, A))} \partial_{Msg}([\![P_1]\!]_A \parallel [\![M_2]\!]_C \parallel [\![Q]\!]_B)$$
$$\xrightarrow{(\{A \not\rightsquigarrow B\}, nsnd(req, A))} \partial_{Msg}([\![P_1]\!]_A \parallel [\![M_2]\!]_C \parallel [\![Q]\!]_B)$$
$$\dots$$

The reason is found in the specification of M_2 which does not handle request messages, and hence, for such a topology no data will be received by B although there is a path form A to B and from B to A. Therefore, we revise M_2 as:

$$M_2 \stackrel{def}{=} sense(B, rcv(data_C).snd(data_B).M_2 + rcv(req).snd(rep_C).M_2,$$
$$snd(error).snd(req).M_1)$$

The path above also exists in the lossy setting, but with all disconnectivity pairs removed from the network constraints. However, an exhaustive and therefore expensive inspection of this path is needed to determine that it is due to a design error. The first transition carries the label $(\{A \not\rightsquigarrow B, A \rightsquigarrow C\}, nsnd(req, A))$ in the reliable setting, meaning that B is not ready to receive, and the label $(\{A \rightsquigarrow C\}, nsnd(req, A))$ in the lossy setting. The latter label indicates that either B was not ready to receive or it was not connected to A. So in the lossy setting one has to examine the origin state to find out if B had an enabled receive action or not. The concept of not being ready to receive is treated in the same way as a lossy communication. Since only the former may be due to a conceptual design in the protocol, finding design errors is not straightforward in the lossy setting. In general the lossy setting will produce a large number of possible error traces that all need to be examined exhaustively, while the reliable setting will produce no spurious error traces.

3.5.2 Protocol Analysis

The properties of wireless protocols, specially MANETs, tends to be weaker in comparison with wired protocols. For instance, the simple property of packet delivery from node A to B is specified as "if there is a path from A to B for a long enough period of time, any packet sent by A, will be received by B" [64]. The topology-dependent behavior of communication, and consequently the need for multi-hop communication between nodes, make their properties preconditioned by the existence of some paths among nodes.

To investigate the topology-dependent properties of MANETs by equational reasoning, it is necessary to enrich our process theory *RCNT* to specify behaviors constrained by multi-hop constraints. To this aim, we extend the action prefix operator of *RCNT* with actions that are paired with multi-hop constraints.

Viewing a network topology as a directed graph, a multi-hop constraint is represented as a set of multihop connectivity pairs $\dashrightarrow$: $Loc \times Loc$ and $\not\dashrightarrow$: $Loc \times Loc$. For instance, $A \dashrightarrow C$ denotes there exists a multi-hop connection from A to C, and consequently C can indirectly receive data from A while $A \not\dashrightarrow C$ denotes that there exists no multi-hop connection from A to C and hence C can not receive data from A. Let $\mathbb{M}(Loc)$ denote the set of multi-hop constraints that can be defined over network addresses in Loc, ranged over by $\mathcal{M}$. Each multi-hop constraint $\mathcal{M}$ represents the set of network topologies that satisfy multi-hop connectivity pairs in $\mathcal{M}$, i.e. $\gamma \in \Gamma(\mathcal{M})$ iff for each $\ell \dashrightarrow \ell'$ in $\mathcal{M}$ there is a multi-hop connection from ℓ to ℓ' in γ, and for each $\ell \not\dashrightarrow \ell'$ in $\mathcal{M}$, there is no multi-hop connection from ℓ to ℓ' in γ.

We extend *RCNT* with terms like $(\mathcal{M}, \iota).t$, where $\iota \in IAct \cup \{\tau\}$, denotes that the action ι is possible if the underlying topology belongs to the multi-hop network constraint $\mathcal{M}$. To define a well-formed *RCNT* term, the rule which restricts the application of the new prefixed-actions to sequential processes, is added to the previous ones. Furthermore, a term cannot have two summands such that

one is prefixed by an action of the form $(\mathcal{C}, \eta)$ and the other by an action of the form $(\mathcal{M}, \iota)$. So terms with an action of the form $(\mathcal{M}, \iota)$ only contain action prefix (with multi-hop constraints), choice and recursion operators.

To reason about the correctness of a MANET protocol, its behavior can be abstractly specified by observable internal actions with the required conditions on the underlying topology, i.e., ι-actions with multi-hop constraints. Intuitively, each communication of a protocol implementation triggers an internal action. Such communications are abstracted away by τ-transitions. Therefore, we define a novel preorder relation to examine if a protocol refines its specification. To this aim, a sequence of τ-transitions is allowed to precede an action that is matched to an action of the specification, as long as the accumulated network constraints of the τ-transitions satisfy the multi-hop network constraint of the matched action. By accumulating network constraints, it is ensured that the set of the links that make the multi-hop connections indicated by the multi-hop constraint hold, will be stable during the communications. Hence our preorder relation is parametrized by a network constraint to reflect such accumulated network constraints.

To provide such a relation, we use the notation $\stackrel{\mathcal{C}}{\Rightarrow}$ which is the reflexive and transitive closure of τ-relations while their network constraints are accumulated:

- $t \stackrel{\{\,\}}{\Longrightarrow} t$;
- if $t \xrightarrow{(\mathcal{C},\tau)} t'$ for some arbitrary network constraint $\mathcal{C}$ and $t' \stackrel{\mathcal{C}'}{\Rightarrow} t''$, then $t \stackrel{\mathcal{C}'\cup\mathcal{C}}{\Longrightarrow} t''$, where $\mathcal{C}' \cup \mathcal{C}$ is well-formed.

Furthermore, the network constraint $\mathcal{C}$ satisfies the multi-hop constraint $\mathcal{C}$, denoted by $\mathcal{C} \models \mathcal{M}$ iff $\exists\, \gamma \in \Gamma(\mathcal{C})\, (\gamma \models \mathcal{M})$. We remark that a network constraint like $\{A \not\rightsquigarrow B\}$ may satisfy both multi-hop constraints $\{A \dashrightarrow B\}$ and $\{A \not\dashrightarrow B\}$, but $\{A \rightsquigarrow B\}$ only satisfies $\{A \dashrightarrow B\}$.

Definition 3.6. A binary relation $\mathcal{R}_{\mathcal{C}}$ on $RCNT$ terms is a refinement relation if $t\ \mathcal{R}_{\mathcal{C}}\ s$ implies:

- if $t \xrightarrow{(\mathcal{C}',\eta)} t'$, where $\mathcal{C} \cup \mathcal{C}' \in \mathbb{C}^v(Loc)$, then
  - $\eta = \tau$ and $t'\ \mathcal{R}_{\mathcal{C}\cup\mathcal{C}'}\ s$, or
  - there is an s' such that $s \xrightarrow{(\mathcal{C},\eta)} s'$, and $t'\ \mathcal{R}_{\mathcal{C}\cup\mathcal{C}'}\ s'$, or
  - $\eta = \iota$ for some $\iota \in IAct \cup \{\tau\}$ and there is an s' such that $s \xrightarrow{(\mathcal{M},\iota)} s'$ with $\mathcal{C} \cup \mathcal{C}' \models \mathcal{M}$ and $t'\ \mathcal{R}_{\mathcal{C}\cup\mathcal{C}'}\ s'$;
- if $s \xrightarrow{(\mathcal{M},\iota)} s'$, then there are t'' and t' such that $t \stackrel{\mathcal{C}'}{\Rightarrow} t'' \xrightarrow{(\mathcal{C}'',\iota)} t'$ with $\mathcal{C} \cup \mathcal{C}' \cup \mathcal{C}'' \models \mathcal{M}$ and $t'\ \mathcal{R}_{\mathcal{C}\cup\mathcal{C}'\cup\mathcal{C}''}\ s'$ where $\mathcal{C} \cup \mathcal{C}' \cup \mathcal{C}'' \in \mathbb{C}^v(Loc)$;
- if $s \xrightarrow{(\mathcal{C},\eta)} s'$, then there is a t' such that $t \xrightarrow{(\mathcal{C}',\eta)} t'$ with $t'\ \mathcal{R}_{\mathcal{C}\cup\mathcal{C}'}\ s'$.

The protocol t refines the specification s, denoted by $t \sqsubseteq s$, if $t \; \mathcal{R}_{\{\,\}} \; s$ holds for some refinement relation $\mathcal{R}_{\{\,\}}$.

Theorem 3.7. *Refinement is a preorder relation and has the precongruence property.*

See Section A.5 for its proof. The following proposition is helpful to prove refinement between two processes:

Proposition 3.8. *Suppose $\iota \in IAct$. The following rules holds*

$$(\mathcal{C}, \tau).t \sqsubseteq (\mathcal{M}, \iota).s \Leftrightarrow \mathcal{C} \rhd t \sqsubseteq (\mathcal{M}, \iota).s \wedge \mathcal{C} \models \mathcal{M}$$
$$(\mathcal{C}, \iota).t \sqsubseteq (\mathcal{M}, \iota).s \Leftrightarrow \mathcal{C} \rhd t \sqsubseteq s.$$

These rules correspond to the transfer conditions of Definition 3.6, and their proofs are discussed in Section A.5. Instead of finding a refinement relation and showing that it satisfies the transfer conditions of Definition 3.6 that its proof process is managed at the semantic level, we propose a proof process at the syntactic level. To show that $t \sqsubseteq s$, where s and t are closed terms for the specification and implementation, by our axiomatization, both s and t are rewritten into a set of summands like $(\mathcal{C}_1, \iota_1).t_1 + \ldots + (\mathcal{C}_n, \iota_n).t_n$ and $(\mathcal{M}_1, \iota'_1).s_1 + \ldots + (\mathcal{M}_m, \iota'_m).s_m$ for some $n \geq 0$ and $m \geq 0$. The proof of $t \sqsubseteq s$ can be reduced to proving that:

$$\begin{aligned}\forall i \leq n\,,\, \mathcal{C}_i \in \mathbb{C}^v(Loc)\,(\exists!\, j \leq m\,(\iota_i = \iota'_j \wedge \mathcal{C}_i \models \mathcal{M}_j \wedge \mathcal{C}_i \rhd t_i \sqsubseteq s_j) \\ \vee(\mathcal{C}_i \rhd t_i \sqsubseteq s))\end{aligned}$$

using the precongruence property of our refinement for the choice operator and the rules of Proposition 3.8. The first disjunct represents the third case of the first transfer condition while the second disjunct represents the first conditions of both transfer conditions of Definition 3.6. This proof process proceeds until we reach to a predicate $\mathcal{C}' \rhd t' \;\sqsubseteq\; s'$ to prove for which either we have previously examined $\mathcal{C}'' \rhd t' \;\sqsubseteq\; 's'$ where $\mathcal{C}' \preccurlyeq \mathcal{C}''$, or it holds trivially.

To analyze the correctness of our simple routing protocol, we investigate if it has the packet delivery property. To this end, we verify whether $\mathcal{N} \equiv \tau_{Msg}(\partial_{Msg}([\![P]\!]_A \parallel [\![M]\!]_C \parallel [\![Q]\!]_B))$ refines $\mathfrak{S}$, where $\mathfrak{S}$ is defined as:

$$\begin{aligned}\mathfrak{S} \equiv rec\mathfrak{O} \cdot (\{A \dashrightarrow B, B \dashrightarrow A\}, deliver).\mathfrak{O} + (\{A \not\dashrightarrow B\}, \tau).0 \\ +(\{A \dashrightarrow B, B \not\dashrightarrow A\}, \tau).0.\end{aligned}$$

Therefore, we exploit the provided equations in Section **??** to prove such a refinement at the syntactic level. To this aim, we match all the resulting terms of τ-transitions to $\mathfrak{S}$ as long as their accumulated network constraints do not exclusively satisfy one of the multi-hop constraints $\{A \dashrightarrow B, B \dashrightarrow A\}$, $\{A \dashrightarrow B, B \not\dashrightarrow A\}$ or $\{A \not\dashrightarrow B\}$. Otherwise, if the accumulated network constraint of a τ-transition only satisfies $\{A \not\dashrightarrow B\}$ or $\{A \dashrightarrow B, B \not\dashrightarrow A\}$, the resulting term of the τ-transition will be matched to 0.

Thus, we use Equation 3.1 to show that:

$$\begin{aligned}
&\tau_{Msg}(\partial_{Msg}(\llbracket P\rrbracket_A \parallel \llbracket M\rrbracket_C \parallel \llbracket Q\rrbracket_B)) \sqsubseteq \mathfrak{S} \Leftrightarrow\\
&\{A \rightsquigarrow B\} \rhd \tau_{Msg}(\partial_{Msg}(\llbracket P\rrbracket_A \parallel \llbracket M\rrbracket_C \parallel \llbracket deliver.Q\rrbracket_B)) \sqsubseteq \mathfrak{S} \wedge\\
&\{A \not\rightsquigarrow B, A \rightsquigarrow C\} \rhd \tau_{Msg}(\partial_{Msg}(\llbracket P_1\rrbracket_A \parallel \llbracket snd(req).M_1\rrbracket_C \parallel \llbracket Q\rrbracket_B)) \sqsubseteq \mathfrak{S} \wedge \quad (3.3)\\
&\{A \not\rightsquigarrow B, A \not\rightsquigarrow C\} \rhd \tau_{Msg}(\partial_{Msg}(\llbracket P_1\rrbracket_A \parallel \llbracket M\rrbracket_C \parallel \llbracket Q\rrbracket_B)) \sqsubseteq 0
\end{aligned}$$

To prove the refinement relation 3.3, we use the Equation 3.2 to show that

$$\begin{aligned}
&\{A \not\rightsquigarrow B, A \rightsquigarrow C\} \rhd \tau_{Msg}(\partial_{Msg}(P_1\rrbracket_A \parallel \llbracket snd(req).M_1\rrbracket_C \parallel \llbracket Q\rrbracket_B)) \sqsubseteq \mathfrak{S} \Leftrightarrow\\
&\{A \not\rightsquigarrow B, A \rightsquigarrow C, C \rightsquigarrow B\} \rhd \tau_{Msg}(\partial_{Msg}(\llbracket P_1\rrbracket_A \parallel \llbracket M_1\rrbracket_C \parallel \llbracket snd(rep_B).Q\rrbracket_B)) \sqsubseteq \mathfrak{S} \wedge\\
&\{A \not\rightsquigarrow B, A \rightsquigarrow C, C \not\rightsquigarrow B\} \rhd \tau_{Msg}(\partial_{Msg}(\llbracket P_1\rrbracket_A \parallel \llbracket M_1\rrbracket_C \parallel \llbracket Q\rrbracket_B)) \sqsubseteq 0 \qquad (3.4)
\end{aligned}$$

The refinement relation (3.4) trivially holds as it can be proved with the help of our axiomatization, especially the rules *Fold* and $TRes_{1,2}$, that $\{A \not\rightsquigarrow B, A \rightsquigarrow C, C \not\rightsquigarrow B\} \rhd \tau_{Msg}(\partial_{Msg}(\llbracket P_1\rrbracket_A \parallel \llbracket M_1\rrbracket_C \parallel \llbracket Q\rrbracket_B))$ is the answer to the equation $\mathfrak{Q} \stackrel{def}{=} (\{A \not\rightsquigarrow B, A \rightsquigarrow C, C \not\rightsquigarrow B\}, \tau).\mathfrak{Q}$, and trivially

$$rec\mathfrak{Q} \cdot (\{A \not\rightsquigarrow B, A \rightsquigarrow C, C \not\rightsquigarrow B\}, \tau).\mathfrak{Q} \sqsubseteq 0.$$

So, it can be easily proved that $\tau_{Msg}(\partial_{Msg}(\llbracket P\rrbracket_A \parallel \llbracket M\rrbracket_C \parallel \llbracket Q\rrbracket_B)) \sqsubseteq \mathfrak{S}$.

3.6 Case Study: Leader Election Algorithm

To illustrate the applicability of our framework, we specify a leader election protocol for MANETs introduced in [150] and discuss how its corresponding CLTS can be derived to analyze the protocol using the existing model checking tools. To this aim, *RRBPT* is extended with abstract data types, similar to *CNT*, and its messages and process names are parameterized with data. We add two operators that intertwine processes with data: *sum* to let receive actions range over the data values of their message parameters (to be able to receive any value from the environment), and *condition* to let behaviors depend on data values. The process $\sum_{d:D} t$ is equivalent to $t[u_1/d] + t[u_2/d] + \ldots$ for any values $u_1, u_2, \ldots$ over the data domain D. The process $[b]t_1 \diamond t_2$ behaves as t_1 if b evaluates to true, and otherwise as t_2.

3.6.1 Protocol Specification

Leader election algorithms aim at electing a unique leader from a fixed set of nodes. In the context of MANETs, such protocols must consider arbitrary topology changes and aim at finding a unique leader within a strongly connected component (SCC) [150].

The algorithm operates by first "growing" and then "shrinking" a spanning tree rooted at the source node that initiates the election algorithm, using the so-called echo algorithm [31, 66]. After the spanning tree has shrunk completely,

the source node will have the required information to elect the leader. The spanning tree is constructed by broadcasting an $election$ message from the source node to the leaf nodes. The parent of each node in the spanning tree is the node from whom it receives the $election$ message for the first time. After receiving the $election$ message, the node broadcasts $election$ to its other neighbors. The spanning tree is shrunk by broadcasting ack messages from the leaf nodes to the source node. Each node after gathering ack messages from all its children, determines the node with the highest value in its subtree, and sends this information to its parent through an ack message. Since nodes may crash, each node keeps track of the status of children from whom it should still receive an ack, by periodically sending and receiving $probe$ and $reply$ messages. The source node can determine the leader when it has gathered ack messages from all its children. It announces the leader by means of a $leader$ message, which is forwarded down the spanning tree.

Due to node crashes and mobility of nodes, a spanning tree may be partitioned or two spanning trees may merge. A node triggers a fresh leader election process when it gets disconnected from its leader or restarts from the crash state. Thus more than one node can initiate the leader election process independently, leading to concurrent computations. In [150], concurrent computations are handled by requiring that at any time each node participates in at most one computation. To achieve this, each spanning tree is identified by a pair of its source node identifier id and a sequence number num, which is incremented each time a node starts a fresh computation, called *computation index*, and all messages except $leader$ are tagged with $\langle id, num \rangle$. A total order is defined on computation indices. Each node belongs to a spanning tree from which it received the $election$ message with the highest computation index. When two disjoint MANETs connect, the algorithm allows MANETs to terminate their ongoing computations and then exchange their leader through $leader$ messages; the combined MANET adopts the leader with the higher value.

In the specification and verification of the protocol, we abstracted away exchanges of $probe$ and $reply$ messages: each node may non-deterministically conclude that it has received ack messages from all its children and stop processing subsequent ack messages. Furthermore, with the help of the sensing operator, each node adapts its behavior in case it gets disconnected from its parent. To make the state space finite we assume that each node can disconnect from its leader only once. We define a total order on the network addresses of nodes and for simplicity assume that the value of a node is the same as its network address.

Our specification uses as abstract data types the Booleans $Bool$ (with domain values T and F), addresses Loc, natural numbers Nat, and computation indices $CompInd$. The variables maintained by each node are: $elec$ and ack of type $Bool$, where $elec$ is true while the node is involved in a computation, and ack is true if the node has not yet sent an ack message to its parent; lid, max and $parent$ of type Loc, where lid denotes the address of the supposed leader (which is updated when the node receives a $leader$ message), max is the highest value

the node has encountered in a computation, and $parent$ the address of the node's parent in the spanning tree; src defines the computation index of the spanning tree of which the node currently is a member. Values of $CompInd$ are pairs of a sequence number and an address, paired by the constructor function dc. To access the items of src, for the sake of readability, we use a dot notation like $src.id$; num of type Nat denotes the node's sequence number.

The specification of the algorithm is given in Fig. 3.5. The pre-conditions to send and receive $election$, ack and $leader$ messages and parameter updates match with the specification given in [150] with two minor differences. We elaborate on each summand of the specification, and highlight points where our specification differs from [150].

In the first summand, in case a node has a leader different from itself (denoted by $(id \neq lid) \wedge (lid \neq ?)$) and is disconnected from its leader (examined by the sense operator), the node initiates the election algorithm by broadcasting a leader message. The conjunct "$num < 1$" prevents an infinite growth of the state space due to an increase of num values within computation indices. An election process is also initiated, as indicated by the second summand, when a node has no leader and is not already participating in an election process (i.e., $\neg elec \wedge (lid = ?)$). This is the case when a node restarts from the crash state. After sending the message $election$, $elec$ and ack are set to true, while max is set to the node's identifier. Furthermore, $parent$ and lid are set to unknown, while src is set to $dc(num, id)$.

The third summand specifies how a node handles received $election$ messages. Such a message is processed if the receiving node has not sent its ack yet and the message belongs to an election process with a higher computation index than what the node has seen so far (i.e., $elec \wedge dc(num1, id1) > src \wedge ack$), or the node has just recovered from a crash and hence has no leader and has not participated in an election process yet (i.e., $\neg elec \wedge (lid = ?)$). In those cases the node joins the election process, by setting its src to $dc(num1, id1)$, and setting its parent to the node $id2$ it received the message from.

The fourth summand specifies when a node sends its ack message. A non-root node that has not sent its ack message before (indicated by $(src.id \neq id) \wedge ack$) announces the maximum identifier max it knows by sending the message $ack(src, max)$, where src identifies the election process in which the node is involved. Then ack is set to false.

Upon receiving an ack message, as specified by the fifth summand, it is processed if the receiving node has not previously sent its ack and the value contained in the ack message (i.e., mid) is greater than the maximum value the node has seen so far. Then the node updates its maximum identifier from max to mid. (Although max is updated upon receipt of an ack message, in [150] a node announces the maximum value it knows by sending the message $ack(src, lid)$.)

The sixth summand specifies that a root node announces the node with the largest value it knows as the leader, if it has not previously announced it (i.e.,

$$
\begin{array}{l}
node(id : Loc, elec : Bool, ack : Bool, lid : Loc, max : Loc, parent : Loc, \\
\quad src : CompInd, num : Nat) \stackrel{def}{=} \\
\quad [(id \neq lid) \wedge (lid \neq ?) \wedge num < 1] \\
\quad\quad sense(lid, node(id, elec, ack, lid, max, parent, src, num), \\
\quad\quad\quad snd(election(dc(num, id), id)). \\
\quad\quad\quad\quad node(id, T, T, ?, id, ?, dc(num, id), num + 1)) \diamond 0 \; + \\
\quad [\neg elec \wedge (lid = ?) \wedge num < 1] \\
\quad\quad snd(election(dc(num, id), id)). \\
\quad\quad\quad node(id, T, T, ?, id, ?, dc(num, id), num + 1) \diamond 0 \; + \\
\quad \sum_{num1:Nat} \sum_{id1:Loc} \sum_{id2:Loc} rcv(election(dc(num1, id1), id2)). \\
\quad\quad [(\neg elec \wedge (lid = ?)) \vee (elec \wedge dc(num1, id1) > src \wedge ack)] \\
\quad\quad\quad snd(election(dc(num1, id1), id)). \\
\quad\quad\quad\quad node(id, T, T, lid, id, id2, dc(num1, id1), num) \diamond \\
\quad\quad\quad node(id, elec, ack, lid, max, parent, src, num) \; + \\
\quad [ack \wedge (src.id \neq id)](snd(ack(src, max)). \\
\quad\quad node(id, elec, F, lid, max, parent, src, num))) \diamond 0 \; + \\
\quad \sum_{mid:Loc} rcv(ack(src, mid)).[ack \wedge (mid > max)] \\
\quad\quad node(id, elec, ack, lid, mid, parent, src, num) \diamond \\
\quad\quad node(id, elec, ack, lid, max, parent, src, num) \; + \\
\quad [(src.id = id) \wedge ack](snd(leader(max)). \\
\quad\quad node(id, F, F, max, max, parent, src, num)) \diamond 0 \; + \\
\quad [(parent \neq ?) \wedge \neg ack \wedge elec] \\
\quad\quad sense(parent, node(id, elec, ack, lid, max, parent, src, num), \\
\quad\quad\quad snd(leader(max)).node(id, F, F, max, max, id, src, num)) \diamond 0 \; + \\
\quad \sum_{li:Loc} rcv(leader(li)).([\neg elec \wedge lid > li)] \\
\quad\quad snd(leader(lid)).node(id, elec, ack, lid, max, parent, src, num) \diamond (\\
\quad\quad\quad [(\neg ack \wedge elec \wedge li \geq max) \vee (\neg elec \wedge li > lid)] \\
\quad\quad\quad\quad snd(leader(li)).node(id, F, ack, li, max, parent, src, num) \diamond \\
\quad\quad\quad\quad node(id, elec, ack, lid, max, parent, src, num))) \; + \\
\quad [(lid \neq ?) \wedge \neg elec] finish(lid, id).node(id, elec, ack, lid, max, parent, src, num) \diamond 0 \; + \\
\quad [(lid \neq ?) \wedge (lid \neq id) \wedge \neg elec] \\
\quad\quad sense(lid, \; snd(leader(lid)).node(id, elec, ack, lid, max, parent, src, num), \\
\quad\quad\quad node(id, elec, ack, lid, max, parent, src, num)) \; + \\
\quad [lid = id]snd(leader(lid)).node(id, elec, ack, lid, max, parent, src, num) \diamond 0
\end{array}
$$

Figure 3.5: Specification of the leader election algorithm for MANETs [150].

$(src.id = id) \wedge ack$). Then it sets its lid to max and terminates the election process by setting $elec$ and ack to false.

The seventh summand specifies the case that a node previously sent ack and is waiting to receive the $leader$ from its parent (i.e., $(parent \neq ?) \wedge \neg ack \wedge elec$), but gets disconnected from its parent (examined by the sense operator). The node then acts as the root of its subtree by announcing the leader.

The eighth summand specifies how a node handles received $leader$ messages. In case such a message contains an identifier higher than the node's leader and the node's maximum value (after it informed its parent about this maximum value, i.e., $\neg ack \wedge elec$), it adopts that identifier as its leader, and announces the new leader to inform its neighbors. Otherwise it announces its own leader to the sender.

With the aim of model checking certain properties, to observe the leader of a node, a self-loop is added to states with the termination status (i.e., $(lid \neq ?) \wedge \neg elec$). It performs the action $finish$, parameterized with the node's leader and identifier. This is specified by the ninth summand.

The tenth summand specifies that non-leader nodes periodically announce their leader, if they are still connected to their parent (examined by the sense operator), Owing to these messages, if two spanning trees are joined, nodes in one spanning tree are informed about the leader of the other spanning tree. As specified by the eighth summand, the leader with the higher identifier is selected as the leader of the merger of the two spanning trees.

Similarly, leader nodes periodically announce their existence, as specified by the last summand. (In [150], upon receiving a $leader(li)$ message while $\neg elec \wedge lid > lid$, the node wrongly sends $leader(max)$ instead of $leader(lid)$.)

A MANET of four nodes is specified by the parallel composition of $[\![node(\ell, F, F, ?, \ell, ?, dc(0, ?), 0)]\!]_\ell$ for $\ell \in \{A, B, C, D\}$. Furthermore, we close the network on all message types and abstract away communication actions:

$$\tau_{Msg}(\partial_{Msg}([\![node(A, F, F, ?, ?, ?, dc(0, ?), 0)]\!]_A \parallel \ldots \parallel [\![node(D, F, F, ?, ?, ?, dc(0, ?), 0)]\!]_D))$$

3.6.2 Tool Support

We exploit the mCRL2 toolset [83] to convert *RRBPT* specifications into LTSs, where actions are parameterized by network constraints. Hence, existing model checker tools can be used to inspect desired properties over the resulting LTSs. The framework of mCRL2 integrates process and abstract data specifications. Network constraints and calculations over them can thus be specified as data terms and functions, respectively.

The main differences between mCRL2 and *RRBPT* are on the deployment and sensing operators of *RRBPT* and their parallel composition. For an *RRBPT* process

term t deployed at address ℓ, assume $encode(t, \ell, \mathcal{C})$ denotes the corresponding code in mCRL2, defined inductively by:

$$\begin{aligned} encode(t_1 + t_2, \ell, \mathcal{C}) &= encode(t_1, \ell, \mathcal{C}) + encode(t_2, \ell, \mathcal{C}) \\ encode([b]t_1 \diamond t_2, \ell, \mathcal{C}) &= b \rightarrow encode(t_1, \ell, \mathcal{C}) \diamond encode(t_2, \ell, \mathcal{C}) \\ encode(0, \ell, \mathcal{C}) &= 0 \end{aligned}$$

where $b \rightarrow t_1 \diamond t_2$ behaves as t_1 if b evaluates to true, and otherwise as t_2. We postpone the explanation of the parameter $\mathcal{C}$ until the explanation of the encoding of the action prefix and $sense$ operator.

To model pairs of network constraints and actions in the semantics of *RRBPT*, we parameterize network send and receive actions in mCRL2 with network constraints as follows. In mCRL2, actions α and β are synchronized if their synchronization is defined, denoted by $\alpha \mid \beta = \gamma$, and they agree on the number and values of their parameters. By contrast, in *RRBPT* two actions are synchronized if they are either two $nrcv$ actions or an $nrcv$ and an $nsnd$ action, and they agree on the message part, while some calculations are performed on their network constraints (see rules Bro and $Recv$). To model the effect of rules Snd and Rcv_{1-3}, and take care of computations over network constraints defined by rules Bro and $Recv$, we define a set of actions $nsnd_j, nrcv_j : \mathbb{C} \times Msg \times Loc$, where $1 \leq j \leq n$ with n the number of nodes. For network addresses Loc, assume L_c, L_d and ℓ such that $Loc = L_c \cup L_d \cup \{\ell\}$ and $L_c \cap L_d = \emptyset$ and $\ell \notin L_c \cup L_d$. Furthermore, let $\mathcal{C}(L_c, L_d, \ell)$ denote the network constraint $\{\ell \rightsquigarrow \ell' \mid \ell' \in L_c\} \cup \{\ell \not\rightsquigarrow \ell' \mid \ell' \in L_d\}$, and $|S|$ the size of set S. The action $nrcv_j(\mathcal{C}(L_c, L_d, \ell), \mathfrak{m}, \ell)$, where $|L_c| = j$, denotes that when the message $\mathfrak{m}$ is sent by the node with address ℓ, j nodes with addresses in L_c are ready to receive it because they are connected to the sender, while nodes with addresses in L_d cannot receive it as they are disconnected from the sender. And $nsnd_j(\mathcal{C}(L_c, L_d, \ell), \mathfrak{m}, \ell)$, where $|L_c| = j$, denotes that the node with address ℓ has sent the message $\mathfrak{m}$, while j nodes with addresses in L_c have received it, but nodes with addresses in L_d cannot receive it as they are disconnected from the sender.

In encoding *RRBPT* processes in mCRL2, deployment operators are removed, and protocol send and receive actions are modeled by calling $sndProcess(\mathfrak{m}, \ell, \mathcal{C})$ and $rcvProcess(\mathfrak{m}, \ell, \mathcal{C})$ respectively, which are defined below. Here ℓ is the address of deployment and network constraint $\mathcal{C}$ defines the status of the links in the network; as a result the network behavior is restricted accordingly. This latter parameter is set to empty to allow any possible behavior of the network. The processes unfold send and receive actions regarding how the node that runs the

actions can be synchronized in the network.

$$
\begin{aligned}
&encode(rcv(m).t, \ell, \mathcal{C}) = rcvProcess(\mathfrak{m}, \ell, \mathcal{C}).encode(t, \ell, \{\})\\
&encode(snd(m).t, \ell, \mathcal{C}) = sndProcess(\mathfrak{m}, \ell, \mathcal{C}).encode(t, \ell, \{\})\\
&rcvProcess(\mathfrak{m}, \ell, \mathcal{C}') = \textstyle\sum_{\ell' \in Loc}(\ell = \ell') \rightarrow\\
&\quad (\textstyle\sum_{L_c \subseteq Loc(\ell' \in L_c)} \sum_{L_d \subseteq Loc} \sum_{\ell'' \in Loc(\ell'' \neq \ell')} \sum_{j>0}\\
&\qquad (\neg \mathcal{C}' \cap \mathcal{C}(L_c, L_d, \ell'') = \emptyset) \rightarrow nrcv_j(\mathcal{C}(L_c, L_d, \ell''), \mathfrak{m}, \ell'') \diamond 0) \diamond 0\\
&sndProcess(\mathfrak{m}, \ell, \mathcal{C}') = \textstyle\sum_{\ell' \in Loc}(\ell = \ell') \rightarrow\\
&\quad (\textstyle\sum_{L_c \subseteq Loc} \sum_{L_d \subseteq Loc} \sum_{j \geq 0}(\neg \mathcal{C}' \cap \mathcal{C}(L_c, L_d, \ell') = \emptyset) \rightarrow\\
&\qquad nsnd_j(\mathcal{C}(L_c, L_d, \ell'), \mathfrak{m}, \ell') \diamond 0) \diamond 0
\end{aligned}
$$

To model the process term $sense(\ell', t_1, t_2)$ deployed at the node with address ℓ, a $sense$ action is performed, parameterized by a network constraint indicating the link status. Furthermore, the appropriate link status is added to the network constraint to encode t_1 and t_2:

$$
\begin{aligned}
&encode(sense(\ell', t_1, t_2), \ell, \mathcal{C}) = sense(\{\ell' \rightsquigarrow \ell\}).encode(t_1, \ell, \mathcal{C} \cup \{\ell' \rightsquigarrow \ell\}) +\\
&\quad sense(\{\ell' \not\rightsquigarrow \ell\}).encode(t_2, \ell, \mathcal{C} \cup \{\ell' \not\rightsquigarrow \ell\})
\end{aligned}
$$

The accumulated network constraints restrict communication of the network as explained in encoding the receive actions. The $sense$ action of a node is synchronized with $sensed$ actions of other nodes to ensure that all nodes adhere to the same restrictions on the network. To this aim, a set of $sensed$ actions with different network constraints is added to each process name definition. For instance, the process definition $X(d : D) \stackrel{def}{=} t$ is encoded as:

$$
\begin{aligned}
X(d : D, x : Loc, c : \mathbb{C}) = &\; encode(t, x, c) +\\
&\textstyle\sum_{\ell, \ell' \in Loc} sensed(\{\ell \rightsquigarrow \ell'\}).X(d, x, \{\ell \rightsquigarrow \ell'\}) +\\
&\qquad sensed(\{\ell \not\rightsquigarrow \ell'\}).X(d, x, \{\ell \not\rightsquigarrow \ell'\}).
\end{aligned}
$$

Consequently, $encode(X(u), \ell, \mathcal{C}) = X(u, \ell, \mathcal{C})$.

The *RRBPT* term $\partial_{Msg}([\![t_1]\!]_{\ell_1} \parallel \ldots \parallel [\![t_n]\!]_{\ell_n})$, where Msg is the set of all messages and t_is are input-enabled, is modeled by the mCRL2 operators renaming ρ, allow ∂, and parallel $\parallel$, where $\rho_{\{a \rightarrow b\}}$ renames the action name a to b, and ∂_A renames action $a \notin A$ to deadlock:

$$
\begin{aligned}
&\rho_{\{\forall i \leq n(nsnd_{ii} \rightarrow nsnd, csense_n \rightarrow sense)\}}(\partial_{\{\forall j \leq n(nsnd_j), csense_n, finish\}}\\
&\qquad ((encode(t_1, \ell_1, \{\}) \parallel \ldots \parallel encode(t_n, \ell_n, \{\}))))
\end{aligned}
$$

where the synchronization set is defined by $nsnd_j \mid nrcv_j \mid \ldots \mid nrcv_j = nsnd_{jj}$ and $sense \mid sensed \mid \ldots \mid sensed = csense_n$, and the number of actions $nrcv_j$ and

$sensed$ should be j and $n-1$, respectively. The LTS resulting from this encoding contains labels of the form $nsnd(\mathcal{C}, \mathfrak{m}, \ell)$, $sense(\mathcal{C})$, and internal actions.

We encoded the leader election case study from Section 3.6.[2]

3.6.3 Protocol Analysis

We can verify properties like "all the nodes eventually have a leader" over the generated LTSs using the mCRL2 toolset. However, we cannot inspect that "each selected leader is the one with the highest index in the SCC of the node". To verify such a property (with any classical model checker), the specification of the node should be revised to maintain the list of its neighbors. Such a revision makes the state space explode as each node has 2^{n-1} possibilities for its neighbors and hence, the state space grows with a factor of $(2^{n-1})^n$ for a network with n nodes. Table 3.4 expresses the size of the generated state space when the mobility is modeled implicitly through the semantics. Revising the protocol specification for the networks with three and four nodes yields state spaces that are 4^3 and 8^4 times larger, respectively. We remark that to inspect such a property with the technique of Section 3.5.2, we have to provide a specification which defines the leader for each pattern of SCCs.

Another approach is modeling the underlying topology as a part of the semantic states, while mobility is addressed by explicit manipulation of the topology (similar to the unfolded CLTS). Thus, the number of states grows with the factor of 2^{n^2-n} which denotes the number of topologies among n nodes. Therefore, this approach yields state spaces that are 8 and 64 times larger in the number of states for the network of three and four nodes, respectively.

Table 3.4: The size of state space for MANETs deploying the leader protocol modulo strong bisimilarity.

no. nodes	no. states	no. transitions
3	$4,278$	$33,536$
4	$357,024$	$3,928,890$

Before reduction modulo strong bisimilarity, our approach yields state spaces with $37,992$ states and $381,912$ transitions and $11,922,272$ states and $150,908,802$ transitions for the network of three and four nodes, respectively.

To verify properties that depend on the topological arrangements of nodes, we introduce a new logic and its model checking algorithm in Chapter 5.

3.7 Related Work

Related calculi to ours are CBS# [121], CWS [117], CMAN [78, 79], CMN and its timed version [115, 116], bKlaim [122], ω-calculus [137], SCWN [81],

[2]This encoding is available at `http://fghassemi.adhoc.ir/leaderelection.zip`.

CSDT [104], AWN [64] and its timed extension [30], and the broadcast psi-calculi [27]. We shortly go through these calculi, with a focus on their approach in specifying a node, modeling topology and mobility, and supporting the neighbor discovery service. Furthermore, we discuss their capabilities to faithfully support the properties of wireless communication at this layer, i.e., being non-blocking and asynchronous. Finally, we examine their purpose of verification, and compare our behavioral equivalence relation to those with a reliable setting.

3.7.1 Modeling Issues

CBS# [121], an extension of Calculus of Broadcasting Systems [129], provides a framework for specification and security analysis of communication protocols for MANETs. In this approach, the mobility is modeled implicitly in the semantics. The operational semantics is parameterized by a set of connectivity graphs, each imposing a set of connections between nodes in a network. Each transition of a MANET is parameterized by a connectivity graph. In other words, the connectivity graph defines the behavior of a network at each step, while in our approach the behavior of a network defines the set of topologies under which such a behavior can occur. Consequently we merge all transitions, and their corresponding topologies leading to the same state, into a transition labeled by network constraints. Thus our approach results in a more compact LTS.

CWS [117] (Calculus for Wireless Systems) is a channel-based algebra for modeling MAC-layer protocols, for which interferences are an essential aspect. In this approach the physical characteristics of nodes such as their physical location and transmission ranges are considered, while locations of nodes are static. CMN [115] (Calculus of Mobile ad hoc Networks), inspired by CWS to model MANETs above the MAC-layer, concerns modeling the mobility of nodes explicitly in the semantics. To this aim, for each node a physical location is specified and the underlying topology is derived by a function d, which takes two locations and computes their distance. If the distance is smaller than a pre-defined value, nodes at those locations are connected. Mobility is modeled by changing the location of the node to an arbitrary location, which may lead to state space explosion if locations are drawn from a real coordinate system.

CMAN [78] (Calculus of Mobile Ad hoc Networks) provides an approach for modeling of MANETs where for each node a connectivity set (its neighbors) is specified. Mobility of a node is modeled explicitly in the semantics by manipulation of the connectivity set of all effected nodes, by adding/removing this node to/from their connectivity set. Later, in [79], CMAN was equipped by a static location binding operator to limit the arbitrary mobility of nodes in the scope of the operator, so that a MANET can be verified for a specific mobility scenario.

bKlaim [122], centered around the tuple space paradigm, provides a process calculus with asynchronous local broadcast; broadcast messages are output into the tuple spaces of neighboring nodes to the sending node. This framework has no specific networking paradigm in mind, and provides a simple calculus

for a more general study of broadcast. Following the same principle as CBS# in modeling topology changes, it constructs mobility-preserving finite abstract transition systems as the semantic model using the notion of *exposed actions* (at each state) which are the immediately available actions and visible data. A state represents the behavior of networks whose exposed action is a preorder of that state's exposed action.

The ω-calculus [137], a conservative extension of the π-calculus, provides an approach to specify MANETs in the same vein as CMAN, but models connectivity information, called process interface, at the specification level by the group concept; a group is a maximal clique in a connectivity graph, and two nodes can communicate if they belong to the same group. Node mobility is captured through the dynamic creation of new groups and dynamically changing process interfaces, using appropriate rules in the semantics. By defining a mobility invariant which constrains node mobilities, one can derive the model of a MANET and verify it against a mobility scenario. In contrast, using our approach, one can verify the model of a MANET against different mobility scenarios by defining predicates over network constraints.

In [81], a simple process calculus with a broadcast operator was extended by realistic mobility models in an orthogonal way, so-called SCWN (a Simple Calculus for Wireless Networks). This approach, which can be understood as a generalization of CWS, is appropriate to verify properties under a specific mobility model, in contrast to the arbitrary mobility model. To this aim, the specification of a node is equipped with a mobility function which determines the movement trajectories of a node over time and consequently its neighbors; the semantics incorporates a notion of global time passing and is parameterized by a mobility model which manipulates the mobility function of a node. This method is based on computations of the transmission range of a sender using physical locations of nodes to derive the real underlying topology. This approach suffers from state space explosion because of its real-time delay transitions, which may be resolved by using a discrete time delay (as proposed by its inventors).

CSDT [104] was inspired by previous works on issues such as broadcasting, movement and separating between a node's control and topology information. However, it provides two additional features: broadcasting at multiple transmission ranges, i.e., normal and high, and neighbor discovery so that nodes remain aware of their connection topology to successfully complete their tasks. To mimic the behavior of a neighborhood-discovery protocol at the semantics, for each node, an arbitrary subset of the node's neighbors is associated as the believed neighbors; these sets are updated due to topology changes. Topology and its changes are modeled in the same way as CMN with the difference that the notions of locations and their interconnections are captured as a graph.

AWN [64] is a process algebra for specification and verification of wireless routing protocols with a set of rich data structures to support necessary data types of mesh routing protocols like routing tables, conditional unicast and local broadcast. Each node is specified by parallel composition of two processes, one

is responsible for message handling and maintaining the protocol data such as routing tables while the other manages the queuing of messages as they arrive. Hence, the node is naturally always input-enabled, and local broadcast is deliberately non-blocking. However, it is discussed how AWN can be make non-blocking by using a semantic rule as Rcv_2 in our case (or the rules of CBS#).

Broadcast psi-calculus [27] extends the psi-calculus [17], a parametric framework for extensions of the π-calculus, with primitives for broadcast communication, namely broadcast channels and connectivity predicates to regulate which channel can send or receive from which broadcast channel. It allows to specify arbitrary data structures and logical assertions for facts about data. Connectivity among node can be specified as assertions using connectivity predicates which can be inquired in the specification (neighbor discovery). Similarly, mobility can be modeled by generation of assertions as a part of the specification (a so-called topology controller). Hence, standard mobility models over a discretized finite space can be specified by a user. However, due to its parameters and involvement of user to specify mobility and connectivity assertions, its application is cumbersome.

Table 3.5: Comparison between related algebras: n refers to the name of the node, ℓ to the logical address, l to the physical location, s to the data store, r to the transmission range, σ to the connectivity set, and g to the connected groups of a node. Finally, f denotes the mobility function, which defines the location of a node at time t, and T is a timeout when the mobility function is updated. N and H are believed normal-range and high-range neighbors, and Γ is the topology controller specified by guarded assertions.

	Node specification	Comm.	Conn. of nodes	Mobility	Neighbor discovery
CBS#	$\ell[p, s]$	reliable	$-$	implicit	-
CWS	$n[p]^c_{l,r}$	reliable	$d(l, l')$	$-$	-
CMN	$n[p]^c_{l,r}$	lossy	$d(l, l')$	explicit	-
CMAN	$[p]^\sigma_\ell$	lossy	σ	explicit	-
bKlaim	$\ell :: p \parallel \ell :: s$	reliable	-	implicit	-
ω-calculus	$p : g$	lossy	g	explicit	-
SCWN	$\ell[p]^T_f$	lossy	$area_n(f(t))$	explicit	-
CSDT	$p : [\![n, l, N, H]\!]$	reliable	$range(l)$	explicit	semantics
AWN	$\ell : p : \sigma$	reliable	σ	explicit	-
bpsi-calculi	$p \mid \Gamma$	lossy	Γ	Γ	syntax
RRBPT	$[\![p]\!]_\ell$	reliable	$-$	implicit	syntax

In Table 3.5, we have compared our core algebra, *RRBPT*, with the related ones in terms of node specification, how connectivity information is specified or derived, and how mobility is modeled. We classify these approaches in addressing mobility into three groups: explicit, implicit, and syntax. In *explicit* modeling of mobility in the semantics, the underlying topology is modeled as a part of

the semantics state, and mobility is modeled by performing transitions (with an unobservable action) between states, by the application of mobility rules which manipulate the topology model. In *implicit* modeling, each state is a representative of all possible topologies a network can meet and a network can be at any of these topologies. CLTSs model mobility implicitly, and as explained in Section 3.1.2, they are unfolded by explicitly modeling mobility at the semantic level. Finally, by *syntax*, we mean topology changes are specified as a part of the specification.

Among all these approaches, only [122] is intrinsically asynchronous. The non-blocking property is a consequence of either nodes being *input-enabled* or the communication primitives being lossy. In the former case, the asynchronous property is achieved through abstract data specifications [55] in line with the approach from [84, 85], in which the sum operator plays a pivotal role. Each process is then parametrized by a variable of the queue type with a summand which receives all possible messages (if the queue is empty). Conversely, this property of the communication cannot be obtained in the frameworks with a lossy communication primitive such as CMN, CMAN, ω-calculus, SCWN, and the broadcast psi-calculi.

To make a process input-enabled while communication is synchronous, three approaches are followed. In the first approach, followed by AWN, the semantics is equipped with a rule similar to our Rcv_2 with a negative premise which expresses that if a node is not ready to receive, the message is simply ignored [64]. Due to our implicit modeling of topology, the negative premise of our rule is more complicated to characterize the unreadiness of nodes regarding the underlying topology. In the second approach, followed by CSDT, counterparts for the rules Bro and $Recv$ are defined with negative premises to cover cases when a process cannot participate in the communication message [104]. The third approach, provided by CBS#, eliminates negative premises, to remain within the de Simone format of structural operational semantics [64], in favor of actions which discard messages [121]. Therefore, the semantics is augmented by rules that trigger the ignore actions for any sending node, receiving nodes for disconnected locations, and deadlock. Furthermore, the rules Bro and $Recv$ are modified to cover cases when a process ignores a message.

3.7.2 Analysis Approaches

The analysis techniques supported by these frameworks, except bKlaim and AWN, are based on a behavioral congruence relation. However, only equations between networks were defined by using structural congruence in these approaches. None of them provides a complete axiomatization for their algebra of MANETs which are helpful in linearizing a given specification to be symbolically verified when the number of nodes is unbounded (see Section 2.3.5). Among the reliable settings, only CSDT provides a behavioral equivalence relation, based on the notion of observational congruence: the receive and send actions are observable while

transitions changing the underlying topology are treated as unobservable. However, due to implicit modeling of topology and mobility, our behavioral equivalence relation has been parametrized with network constraints while it considers the branching structure of MANETs.

A compositional framework for proving invariant properties is given in [29]. To this aim, global invariants stated at the level of individual nodes are lifted to the network of nodes. Therefore, AWN framework and their compositional technique can be mechanized in Isabelle/HOL. The resulting framework was applied in [28] to prove loop freedom of the AODV routing protocol. The properties of AODV, expressed as state/transition invariants, were proved for a network with a bounded number of nodes by checking that they are preserved under any execution of any line of the specification [148].

CBS# and bKlaim are supported by control flow analysis to establish the security properties of a broadcast network. In this approach, specifications are statically inspected to predict approximations to the set of values which may be transmitted/stored by nodes during their executions. The difference in these two works is that analysis of CBS# is restricted to one specific connectivity graph and hence, the influence of topology changes is not exposed in its results.

Model checking is used as a diagnostic tool to locate conceptual design errors in finite representations of a MANET, and hence complements other formal methods-based protocol modeling and verification techniques, such as process algebra. Since physical locations are considered in CMN and SCWN, they suffer from the state space explosion problem even for a simple network. However, in approaches with explicit modeling of topology changes the same problem exists, as the topology changes randomly. As an example, the state space of the simple leader election protocol in Section 3.6 yields $37,992$ states and $381,912$ transitions and $11,922,272$ states and $150,908,802$ transitions for the network of three and four nodes in the compact form of CLTS, respectively. Hence, to apply model checking in the context of these approaches, the arbitrary mobility should be restricted, which is possible in ω-calculus using its mobility invariant concept. We will discuss in Section 5.5, how model checking is applied to AWN, which can be also extended to algebras, e.g. CMAN, ω-calculus, and CSDT. bKlaim introduces a new logic to support efficient model checking over its semantic model, i.e., abstract transition systems. We postpone further explanation about its logic to Section 5.5, after providing our logic.

3.8 Conclusion

We introduced the reliable framework *RRBPT*, suitable to specify and verify MANETs, with the aim to catch errors in design decisions. We discussed the required changes at the semantic model by extending the network constraints with negative connectivity links. Furthermore, we revised the equivalence relation of the lossy setting to preserve required behavior in the reliable framework. Then we

demonstrated which axioms should be revised/added/removed for the reliable setting. We provided an analysis approach at the syntactic level, exploiting a pre-congruence relation and our axiomatization. We applied our analysis approach to a simple routing protocol to prove that it correctly finds routes among connected nodes. Implementation of the framework using mCRL2 was discussed. At the end, a comparison among related calculi was given.

Wireless Rebeca 4

As we discussed in the previous chapter, the synchronous spirit of communication primitives in process calculi prevents natural modeling of nodes: either the modeler needs to ensure their input-enabledness in the specifications or input-enabledness should be built into the semantics. From the verification point of view, in a computational model based on actors, the coarse-grained execution of message handlers reduces the state space of a model substantially. This policy, inherited from the world of object-oriented languages, is justified by the fact that the internal computation and intermediate values of variables are not observable due to encapsulation of objects. These considerations motivate our direction towards actor-based modeling frameworks.

To reap the benefits of the distributed and asynchronous concurrent computation model of actors, we considered the Rebeca language to model such networks. However, the only means of communication in Rebeca (see Section 2.4) is the pair-wise communication between two known nodes. We interpret the known-rebec concept as existence of a one-hop link to initially set up the underlying topology: if A is known to B, then there is a directed communication link from B to A, and so B can send messages to A. In synchronous algebraic frameworks, unicast could be easily simulated by broadcast communication and messages that are parametrized by the receiver address; a receiver handles a message if it is the intended address of the message. However in Rebeca, as a result of a broadcast communication, its message is added to the queue of rebecs that are currently connected to the sender. Therefore, simulating unicast with broadcast causes the state space to grow unnecessarily. Hence, we extend Rebeca not only with asynchronous reliable local broadcast but also with multicast and unicast communications to abstract away from the data link layer services. The reliable unicast communication service of the data link layer provides feedback (to its upper layer applications) in case of (un)successful delivery. Therefore, the new framework provides *conditional unicast* to support modeling protocol behaviors in each scenario (in the semantic model, the status of the underlying topology defines the behavior of actors). The result is a formalism that we call it *wRebeca*.

Since in wRebeca only one-hop communications are considered (in contrast to the original actor model), the assumption about the unpredictability of multi-hop communications (with different delays) is not valid anymore, and thus the

message delivery is in-order and is guaranteed for connected receivers. Consequently, each actor mailbox is modeled through a queue. The core language of wRebeca was first introduced in [153] as an extension of *broadcast* Rebeca [155] in which each actor mailbox was modeled by a bag. Using bags instead of FIFO may result in spurious counterexamples during model checking which cannot occur in reality. We have extended and revised its syntax and provided its formal semantics by two-tiered operational rules (in terms of the semantics of the statements) to not only faithfully support the extension but also fix the flaws.

The resulting framework provides a suitable means to model the behavior of ad hoc networks without the need to consider asynchronous communications handled by message storages in the computation model. However, to minimize the effect of message storages on the growth of the state space, we exploit techniques to reduce it. Since nodes can communicate through broadcast and a limited form of multicast/unicast, it is possible to consider actors that have the same neighbors and local states as identical according to the counter abstraction technique [11, 56, 127]. Therefore, the states whose number of actors with the same neighbors and local state are the same for each local state value, will be aggregated, thus the state space is reduced considerably. The reduced semantics is strongly bisimilar to the original one.

To examine resistance and adaptation of MANET protocols to changes of the underlying topology, we address mobility via arbitrary changes of the topology at the semantic level. However, these random changes make the state space grow exponentially and the model checking technique infeasible even for small networks while the proposed counter abstraction technique becomes invalid. To this end, we consolidate next states which are only different in their topologies, and consequently derive CLTSs (see Section 2.2 and 3.1). We establish that the reduced semantics (i.e., CLTS) is branching bisimilar to the original one, and consequently a set of properties such as ACTL-X [48] are preserved. Therefore, our framework becomes applicable to verify some important properties of real-world MANET protocol, e.g., loop freedom, in the presence of mobility in a unified model (versus generating a model for each mobility scenario).

We illustrate the applicability of our approach through the specification and verification of two MANET protocols, namely a simple flooding-based protocol and AODV. We present a complete and accurate model of the core functionalities of a recent version of AODVv2 protocol (version 11), and investigate its properties like loop freedom. We detect scenarios over which the loop freedom property is violated due to maintaining multiple unconfirmed next hops for a route without checking them to be loop free. We have communicated this scenario to the AODV group and they have confirmed it. In response, their route information evaluation was modified, published in version 13 of the draft.[1] More specifically, the rule for adding a new route if current routes to the same address have an "Unconfirmed" status, was changed. Furthermore, we verify the monotonic in-

[1] https://tools.ietf.org/html/draft-ietf-manet-aodvv2-13.

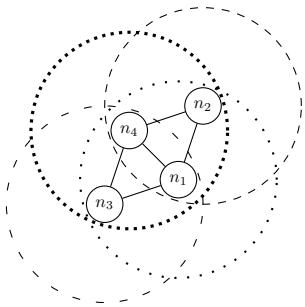
crease of sequence numbers and packet delivery properties using existing model checkers.

The chapter is structured as follows. Section 4.1 briefly explains the idea behind the counter abstraction technique and its relation to the symmetry reduction technique. Section 4.2 discusses how the main modeling challenges of MANETs are addressed in wRebeca. Section 4.3 presents our extension to Rebeca for modeling MANETs. In Section 4.4, we generate the state space compactly with the aim of efficient model checking. To illustrate the applicability of our approach, we specify the core functionalities of AODVv2-11 in Section 4.5. Then, in Section 4.6, we exemplify the efficiency of our state space generation by means of two case studies: the AODV and the flooding-based routing protocol. We illustrate our tool and analysis of the models through verification of AODV. Finally, we conclude the chapter by comparing the actor-based framework with the algebraic ones in Section 4.8.

4.1 Counter Abstraction

Since model checking is the main approach of verification in Rebeca, we need to overcome state space explosion. One way to tackle this problem is through applying reduction techniques such as symmetry reduction [35] and counter abstraction [56]. Counter abstraction is a form of symmetry reduction and, in case of full symmetry, it can be used to avoid the *constructive orbit problem*, according to which finding a unique representative of each state is NP-hard [35]. The idea of using counters and counter abstraction in model checking was first introduced in [56].However, the term of *counter abstraction* was first presented in [127] for the verification of parameterized systems and further used in different studies such as [11, 101].

The idea of counter abstraction is to record the global state of a system as a vector of counters, one for each local state. Each counter denotes the number of components currently residing in the corresponding local state. In our work, by "components" we mean the actors of the system. This technique turns a model with an exponential size in n, i.e. m^n, into one of a size polynomial in n, i.e. $\binom{n+m-1}{m}$, where n and m denote the number of components and local states, respectively. Two global states S and S' are considered identical up to permutation if for every local state s, the number of components residing in s is the same in the two states S and S', as permutation only changes the order of elements. For example, consider a system which consists of three components that each have only one variable v_i of boolean type. Three global states $(true, true, false)$, $(false, true, true)$, and $(true, false, true)$ are equivalent and can be abstracted into one global state represented as $(true : 2, false : 1)$.



(a) The network

MNode n_1 $(n_2, n_3, n_4) : (1, \textit{false})$
MNode n_2 $(n_1, n_4) : (2, \textit{false})$
MNode n_3 $(n_1, n_4) : (3, \textit{true})$
MNode n_4 $(n_2, n_3, n_1) : (4, \textit{false})$

(b) Syntactic definition during instantiation

Figure 4.1: A sample of an initial topology and its corresponding syntactic and semantic representations

4.2 Modeling Topology and Mobility

In this section, we discuss issues brought up by extending Rebeca to model and verify MANETs, and our solutions to overcome these challenges. We assume that the number of nodes is fixed (to make the state space finite as explained in [52]).

Every rebec represents a node in the MANET model. A node can communicate only with those located in its communication range, so-called *connected* nodes. In this approach, we consider the topology as a part of the states and randomly change the underlying topology at the semantic level. Recall that a topology is modeled by a function $\gamma : Loc \to I\!P(Loc)$ with $\forall \ell \in Loc\,(\ell \notin \gamma(\ell))$. For simplicity, we assume that the transmission ranges of all nodes are equal. Therefore, connectivity is a bidirectional concept, and hence, the resulting topology modeled by the function γ will be symmetric; if $\ell \in \gamma(\ell') \Leftrightarrow \ell' \in \gamma(\ell)$. We remark that the presented framework and techniques in this chapter are also valid for asymmetric communications. Changing the topology is considered an unobservable action, modeled by a τ-transition, which alters the topology function. Hence, each τ-transition represents a set of (bidirectional) link setups/breakdowns in the underlying topology.

To set up the initial topology of the network, the *known-rebecs* definitions, provided by the Rebeca language, are extended to address the connectivity of rebecs. Fig. 4.1a shows the communication range of the nodes in a simple network. To configure the initial topology of this network, *known-rebecs* of each rebec should be defined as shown in Fig. 4.1b during its instantiation (cf. Fig. 2.4).

For a network of n nodes, there are $2^{((n\times n)-n)/2}$ possible (symmetric) topologies. Considering all these topologies may lead to state space explosion. Hence, we use network constraints to characterize the set of possible topologies. A topology γ is called *valid* for the network constraint $\mathcal{C}$, denoted as $\gamma \vDash \mathcal{C}$, if $\gamma \in \Gamma(\mathcal{C})$.

If the only valid topology for a network constraint is equal to the initial topol-

ogy, then the underlying topology will be static. This case can be useful for modeling WMNs with stable mesh routers with no mesh clients.

4.3 wRebeca: Syntax and Semantics

In this section, we extend the syntax of Rebeca introduced in Section 2.4, with conditional unicast and multicast, topology constraint, and known rebecs to set up the initial topology. Next, we provide the semantics of wRebeca models in terms of LTSs.

4.3.1 Syntax

The grammar of wRebeca is presented in Fig. 4.2. It consists of two major parts: reactive classes and main part. The definition of reactive classes is almost similar to the one in Rebeca. However, the main part is augmented with the $\mathrm{ConstraintPart}$, where constraints are introduced to reduce all possible topologies in the network. The instances of the declared reactive classes are defined in the main part, before the $\mathrm{ConstraintPart}$, by indicating the name of a reactive class and an arbitrary rebec name along with two sets of parentheses divided by the character ":". The parameters between the first couple of parentheses define the neighbors of the rebec in the initial topology. The parameters between the second couple of parentheses is pass values to the initial message server. Rebecs here communicate through broadcast, multicast, and unicast. In the broadcast statement, we simply use the message server name along with its parameters without specifying the receivers of a message. In contrast, when unicasting/multicasting a message, we also need to specify the receiver/receivers of the message. However, there is no delivery guarantee, depending on the location of the receiver. In case of unicasting, the sender can react based on the delivery status. Let $\mathsf{unicast}(\mathrm{Rec}, M(\mathrm{List}(Expr)))$ indicate $\mathsf{unicast}(\mathrm{Rec}, M(\mathrm{List}(Expr)))$ succ :{} unsucc :{} when the delivery status has no effect on the rebec behavior.

In addition to communication statements, there are assignment, conditional, and loop statements. The first one is used to assign a value to a variable. The second is used to branch based on the evaluation of an expression: if the expression evaluates to $true$, then the if part, and otherwise the else part will be executed. Let if $(Expr)$ Block denote if $(Expr)$ Block else $\{\ \}$. Finally, the third is used to execute a set of statements iteratively as long as the loop condition, i.e., the boolean expression $Expr$, holds. Furthermore, break can be used to terminate its nearest enclosing loop statement and transfer the control to the next statement. For the sake of readability, we use for $(T\ x = Expr_1;\ Expr_2;\ Expr_3)\{\ \mathrm{Statement}^*\ \}$ to denote $T\ x = Expr_1;\mathsf{while}\ (Expr_2)\{\ \mathrm{Statement}^*\ Expr_3\ \}$. A variable can be defined in the scope of message servers as a statement similar to programming languages.

$$
\begin{aligned}
\text{Model} &::= \text{ReactiveClass}^{+}\ \text{Main}\\
\text{Main} &::= \mathsf{main}\ \{\text{RebecDecl}^{+}\ \text{ConstraintPart}\ \}\\
\text{List(X)} &::= \langle X, \rangle^{*} X \mid \epsilon\\
\text{RebecDecl} &::= C\ R\ (\text{List}(R))\ :\ (\text{List}(Expr));\\
\text{ConstraintPart} &::= \mathsf{constraint}\ \{\text{Constraint}\}\\
\text{Constraint} &::= \text{ConstrainDec} \mid !\ \text{ConstrainDec} \mid\\
&\quad\ \mathsf{and}(\text{Constraint}\ ,\ \text{Constraint})\\
\text{ConstrainDec} &::= \mathsf{con}(R\ ,\ R) \mid\ \mathsf{true}\\
\text{ReactiveClass} &::= \mathsf{reactiveclass}\ C\ \{\ \text{StateVars MsgServer}^{*}\ \}\\
\text{StateVars} &::= \mathsf{statevars}\ \{\ \text{VarDecl}^{*}\ \}\\
\text{MsgServer} &::= \mathsf{msgsrv}\ M(\text{List}(T\ V))\ \{\ \text{Statement}^{*}\ \}\\
\text{VarDecl} &::= T\ V;\\
\text{Statement} &::= \text{VarDecl} \mid \text{Assign} \mid \text{Conditional} \mid \text{Loop} \mid \text{Broadcast} \mid\\
&\quad\ \text{Multicast} \mid \text{Unicast} \mid \mathsf{break};\\
\text{Assign} &::= V = Expr;\\
\text{Conditional} &::= \mathsf{if}\ (Expr)\ \text{Block}\ \mathsf{else}\ \text{Block}\\
\text{Block} &::= \text{Statement} \mid \{\ \text{Statement}^{*}\ \}\\
\text{Loop} &::= \mathsf{while}(Expr)\ \text{Block}\\
\text{Broadcast} &::= M(\text{List}(Expr));\\
\text{Multicast} &::= \mathsf{multicast}(V\ , M(\text{List}(Expr)));\\
\text{Unicast} &::= \mathsf{unicast}(\text{Rec}\ , M(\text{List}(Expr)))\ \mathsf{succ}:\text{Block}\ \mathsf{unsucc}:\text{Block}\\
\text{Rec} &::= \mathsf{self} \mid V
\end{aligned}
$$

Figure 4.2: wRebeca language syntax: Angle brackets ($\langle\ \rangle$) are used as metaparentheses. Superscript * indicates zero or more times repetition. The symbols C, R, T, M, and V denote the set of classes, rebec names, types, message server and variable names, respectively. The symbol $Expr$ denotes an expression, which can be an arithmetic or a boolean expression.

A given wRebeca model is called *well-formed* if no state variable is also declared in the scope of a message server as a local variable, no two state variables, message servers or rebec classes have identical names, identifiers of variables, message servers and classes do not clash, and all rebec instance accesses and variable accesses occur over declared/specified ones. Furthermore, a message can be sent to a specific/all rebec(s) if its corresponding message handler has been defined in the receiver rebec(s) and the number and type of actual parameters correctly match the formal ones in the message server specification. Each `break` should occur within a loop statement. Furthermore, the initial topology should satisfy the network constraint and be symmetric, i.e., if n_1 is the known rebec of n_2, then n_2 should be the known rebec of n_1. By default, the network constraint is `true` if no network constraint is defined, and all the nodes are disconnected if no initial topology is defined.

Example: Fig. 4.3 illustrates a revised version of Max-Algorithm introduced in 2.4. Every node in the network contains an integer value and intends to find the maximum value of all nodes in its vicinity in a distributed manner by flooding the value. The `MNode` message server acts as a constructor and has a parameter, named `starter`. The rebec with the `starter` value $true$ initiates the algorithm by broadcasting the first message. Whenever a node receives a value, it updates its value if it is less then the received one. Furthermore, it rebroadcast the its value if the received one is different. To reduce the number of transferred messages, each message contains a counter, called `hopNum`, which shows how many times it has been re-broadcast. If the `hopNum` is more than the specified bound, it quits re-broadcasting.

4.3.2 Semantics

The formal semantics of a well-formed wRebeca model is expressed as an LTS. In the following, we formally define the states, transitions, and initial states of the semantic model generated for a given wRebeca specification. To this aim, the given specification is decomposed into its constituent components, i.e., rebec instances, reactive classes, initial topology, and network constraint represented by the wRebeca model $\mathcal{M}$. The topology silently transforms while satisfying the given network constraint. As explained in Section 2.4, message server executions are atomic; their executions are not interleaved with executions of message servers of other rebecs. Intuitively, the global states of a wRebeca model are defined by the local states of its rebecs and the underlying topology. Consequently, a state transition occurs either upon atomic execution of a message server (i.e., when a rebec processes its corresponding message in its queue), or at a random change in the topology (modeled through unobservable τ-transitions).

Let V denote the set of variables ranged over by x, and Val denote the set of all possible values for the variables, ranged over by e. Furthermore, we assume that the set of types T consists of the integer and boolean data types, i.e., $T =$

```
1   reactiveclass MNode
2   {
3     statevars
4     {
5       int my_i;
6     }

8     msgsrv MNode(int j, boolean
          starter)
9     {
10      my_i = j;
11      if (starter) send(my_i,0);
12    }
13    msgsrv send(int i, int
          hopNum)
14    {
15      if (i != my_i){
16        if (i>my_i) my_i = i;
17        if (hopNum<3) {
18          hopNum++;
19          send(my_i,hhopNum);
20        }
21      }
22    }
23    main
24    {
25      MNode n1(n2,n3,n4):
26                  (1, false);
27      MNode n2(n1,n4):(2,false);
28      MNode n3(n1,n4):(3,true);
29      MNode n4(n2,n3,n1)
30                  :(4, false);

32      constraint
33      {
34        and(con(n1,n2),!con(n1,n3))
35      }
36    }
```

Figure 4.3: Revised max-algorithm in a network consisting of four nodes

$\{int, bool\}$. We consider the default value $0 \in Val$ for the integer and boolean variables since the boolean values $true$ and $false$ can be modeled by 1 and 0 in the semantics, respectively. The variable assignment in each scope can be modeled by the valuation function $V \rightarrow Val$ ranged over by θ. An assignment can be extended by writing $\theta \cup \{y \mapsto e\}$. This can only be done if θ is not defined on y. To manage value assignments regarding scope management, we specify the set of all environments as $Env = Stack(V \rightarrow Val)$, ranged over by υ. Let $upd(\upsilon, \{y \mapsto e\})$ extend the variable assignments of the current scope, i.e, the top of the stack, by $\{y \mapsto e\}$ if the stack is not empty. Assume $Stack()$ denotes an empty environment. By entering into a scope, the environment υ is updated by $push(\theta, \upsilon)$ where θ is empty if the scope belongs to a block (which will be extended by the declarations in the block). Upon exiting from the scope, it is updated by $pop(\upsilon)$ which removes the top of the stack. Let $eval(expr, \upsilon)$ denote the value of the expression $expr$ in the context of environment υ, and $\upsilon[x := e]$ the environment identical to υ except that x, which belongs to the domain of the topmost assignment of υ, is updated to e.

Assume $Seq(D)$ denotes the set of all sequences of elements in D; we use notations $\langle d_1 \dots d_n \rangle$ and ϵ for a non-empty and empty sequence, respectively. Note that the elements in a sequence may be repeated. A FIFO queue of elements of D can be viewed as a $Seq(D)$. For instance, $\langle 2\ 3\ 2\ 4 \rangle \in Seq(\mathbb{N})$ denotes a FIFO queue of natural numbers where its head is 2. For a given FIFO queue $f : Seq(D)$, assume $f \triangleright d$ denotes the sequence obtained by appending d to the end of f, while

$d \triangleright f$ denotes the sequence with head d and tail f.

A wRebeca model is defined through a set of reactive classes, rebec instances, an initial topology, and a network constraint. Let C denote the set of all reactive classes in the model ranged over by c, R the set of rebec instances ranged over by r, and $\mathbb{C}$ the set of network constraints ranged over by $\mathcal{C}$. Assume Γ is the set of all topologies ranged over by γ. Each reactive class c is described by a tuple $c = \langle V_c, M_c \rangle$, where V_c is the set of class state variables and M_c the set of message types ranged over by m that its instances can respond to. We assume that for each class c, we have the state variable $self \in V_c$, and $initial \in M_c$ which can be seen as its constructor in object-oriented languages. For the sake of simplicity, we assume that messages are parameterized with one argument, so Msg_c, where $M_c = Val \rightarrow Msg_c$ defines the set of all messages that rebec instances of the reactive class c can respond to. The formal parameter of a message can be accessed by $fm : M_c \rightarrow V$. Let $Statement$ denote the set of statements ranged over by σ, δ (we use σ^*, δ^* to denote a sequence of statements), and $body : M_c \rightarrow Seq(Statement)$ specify the sequence of statements executed by a message server. A block, denoted by β, is either defined by a statement or a sequence of statements surrounded by braces.

A rebec instance r is specified by the tuple $\langle c, e_0 \rangle$ where $c \in C$ is its reactive class, and e_0 defines the value passed to the message c which is initially put in the rebec's queue. We assume a unique identifier is assigned to each rebec instance. Let $I = \{1 \dots n\}$ denote a finite set of all rebec identifiers ranged over by i and j. Furthermore, we use r_i to denote the rebec instance r with the assigned identifier i. A rebec in wRebeca, like Rebeca, holds its received messages in a FIFO queue.

All rebecs of the model form a closed system, denoted by $\mathcal{M} = \langle \|_{i \in I} r_i, C, \gamma_0, \mathcal{C} \rangle$, where $r_i = \langle c, e_0^i \rangle$ for some $c \in C$ and $\mathcal{C} \in \mathbb{C}$. By default, $\mathcal{C} = true$ and $\forall i, j \leq n((i \neq j \Rightarrow j \in \gamma_0(i) \wedge i \in \gamma_0(j))$ if no network constraint and initial topology were defined. The (global) state of the $\mathcal{M}$ is defined in terms of rebec's local states and the underlying topology.

Definition 4.1. The semantics of a wRebeca model $\mathcal{M} = \langle \|_{i \in I} r_i, C, \gamma_0, \mathcal{C} \rangle$ is expressed by the LTS $\langle S, L, \rightarrow, s_0 \rangle$ where

- $S \subseteq S_1 \times \ldots \times S_n \times \Gamma$ is the set of global states such that $(s_1, \ldots, s_n, \gamma) \in S$ iff $\gamma \vDash \mathcal{C}$, and $S_i = Env \times FIFO_i$ is the set of local states of rebec $r_i = \langle c, e_0^i \rangle$ where $FIFO_i = Seq(Msg_c)$ models a FIFO queue of messages sent to the rebec r_i. Therefore, each s_i can be denoted by the pair (ν_i, f_i). We use the dot notations $s_i.\nu$ and $s_i.f$ to access the environment and FIFO queue of rebec i, respectively.
- $L = Act \cup \{\tau\}$ is the set of labels, where $Act = \bigcup_{c \in C} Msg_c$.
- The transition relation $\rightarrow \subseteq S \times L \times S$ is the least relation satisfying the SOS rules in Tables 4.1 and 4.2.

Table 4.1: wRebeca SOS rules

Term: $\nu_i, f_1, \ldots, f_n, \epsilon \rightsquigarrow_\gamma \nu_i, f_1, \ldots, f_n, \top$

Assign: $\nu_i, f_1, \ldots, f_n, x := expr; \rightsquigarrow_\gamma \nu_i[x := eval(expr, \nu_i)], f_1, \ldots, f_n, \top$

VDecl: $\nu_i, f_1, \ldots, f_n, T\ x; \rightsquigarrow_\gamma upd(\nu_i, \{x \mapsto 0\}), f_1, \ldots, f_n, \top$

Block: $$\frac{push(\emptyset, \nu_i), f_1, \ldots, f_n, \sigma^* \rightsquigarrow_\gamma \nu_i', f_1', \ldots, f_n', \zeta}{\nu_i, f_1, \ldots, f_n, \{\sigma^*\} \rightsquigarrow_\gamma pop(\nu_i'), f_1', \ldots, f_n', \zeta}$$

Cond$_1$: $$\frac{eval(expr, \nu_i) = true \quad \nu_i, f_1, \ldots, f_n, \beta_1 \rightsquigarrow_\gamma \nu_i', f_1', \ldots, f_n', \zeta}{\nu_i, f_1, \ldots, f_n, if\ expr\ \beta_1\ else\ \beta_2 \rightsquigarrow_\gamma \nu_i', f_1', \ldots, f_n', \zeta}$$

Cond$_2$: $$\frac{eval(expr, \nu_i) = false \quad \nu_i, f_1, \ldots, f_n, \beta_2 \rightsquigarrow_\gamma \nu_i', f_1', \ldots, f_n', \zeta}{\nu_i, f_1, \ldots, f_n, if\ expr\ \beta_1\ else\ \beta_2 \rightsquigarrow_\gamma \nu_i', f_1', \ldots, f_n', \zeta}$$

Loop$_1$: $$\frac{\begin{array}{c} eval(expr, \nu_i) = true \\ \nu_i, f_1, \ldots, f_n, \beta \rightsquigarrow_\gamma \nu_i', f_1', \ldots, f_n', \top \\ \nu_i', f_1', \ldots, f_n', while(expr)\ \beta \rightsquigarrow_\gamma \nu_i'', f_1'', \ldots, f_n'', \top \end{array}}{\nu_i, f_1, \ldots, f_n, while(expr)\ \beta \rightsquigarrow_\gamma \nu_i'', f_1'', \ldots, f_n'', \top}$$

Loop$_2$: $$\frac{\begin{array}{c} eval(expr, \nu_i) = true \\ \nu_i, f_1, \ldots, f_n, \beta \rightsquigarrow_\gamma \nu_i', f_1', \ldots, f_n', \bot \end{array}}{\nu_i, f_1, \ldots, f_n, while(expr)\ \beta \rightsquigarrow_\gamma \nu_i', f_1', \ldots, f_n', \top}$$

Loop$_3$: $$\frac{eval(expr, \nu_i) = false}{\nu_i, f_1, \ldots, f_n, while(expr)\ \beta \rightsquigarrow_\gamma \nu_i, f_1, \ldots, f_n, \top}$$

Seq$_1$: $$\frac{\begin{array}{c} \nu_i, f_1, \ldots, f_n, \sigma_1 \rightsquigarrow_\gamma \nu_i', f_1', \ldots, f_n', \top \\ \nu_i', f_1', \ldots, f_n', \sigma_2^* \rightsquigarrow_\gamma \nu_i'', f_1'', \ldots, f_n'', \zeta \end{array}}{\nu_i, f_1, \ldots, f_n, \sigma_1\sigma_2^* \rightsquigarrow_\gamma \nu_i'', f_1'', \ldots, f_n'', \zeta}$$

Seq$_2$: $\nu_i, f_1, \ldots, f_n, break;\ \sigma^* \rightsquigarrow_\gamma \nu_i, f_1, \ldots, f_n, \bot$

Handle: $$\frac{\begin{array}{c} s_i.f = m(e) \triangleright f_i \wedge \forall k \neq i (f_k = s_k.f) \\ \nu_i = push(\{fm(m) \mapsto e\}, s_i.\nu) \\ \nu_i, f_1, \ldots, f_n, body(m) \rightsquigarrow_\gamma\ \nu_i', f_1', \ldots, f_n', \top \end{array}}{(s_1, \ldots, s_n, \gamma) \xrightarrow{m(e)} (s_1', \ldots, s_n', \gamma)}, \text{ where } \forall k \neq i (s_k' = (s_k.\nu, f_k')) \wedge s_i' = (pop(\nu_i'), f_i')$$

Mov $(s_1, \ldots, s_n, \gamma) \xrightarrow{\tau} (s_1, \ldots, s_n, \gamma')$, where $\gamma' \models \mathcal{C}$

Table 4.2: SOS rules of communication in wRebeca

$$BCast: \quad \nu_i, f_1, \ldots, f_n, m(expr); \rightsquigarrow_\gamma \nu_i, f'_1, \ldots, f'_n, \top \text{ , where } [\forall k \leq n(k \neq i \wedge (k \in \gamma(i)) \Rightarrow f'_k = f_k \triangleright m(eval(expr, v_i))][f'_k = f_k])$$

$$MCast: \quad \nu_i, f_1, \ldots, f_n, multicast(rcvs, expr); \rightsquigarrow_\gamma \nu_i, f'_1, \ldots, f'_n, \top \text{ , where } \forall k \leq n(k \in rcvs \wedge k \in \gamma(i)) \Rightarrow [f'_k = m(eval(expr, v_i)) \triangleright f_k][f'_k = f_k])$$

$$UCast_1: \quad \frac{(j \in \gamma(i)) \quad f'_j = f_j \triangleright m(eval(expr, v_i)) \wedge \forall k \neq j(f'_k = f_k) \quad \nu_i, f'_1, \ldots, f'_n, \beta_1 \rightsquigarrow_\gamma \nu'_i, f''_1, \ldots, f''_n, \zeta}{\nu_i, f_1, \ldots, f_n, unicast(j, m(expr))\ succ : \beta_1\ unsucc : \beta_2 \rightsquigarrow_\gamma \nu'_i, f''_1, \ldots, f''_n, \zeta}$$

$$UCast_2: \quad \frac{(j \notin \gamma(i)) \quad \nu_i, f_1, \ldots, f_n, \beta_2 \rightsquigarrow_\gamma \nu'_i, f'_1, \ldots, f'_n, \zeta}{\nu_i, f_1, \ldots, f_n, unicast(j, m(expr))\ succ : \beta_1\ unsucc : \beta_2 \rightsquigarrow_\gamma \nu'_i, f'_1, \ldots, f'_n, \zeta}$$

- s_0 is the initial state, and is defined as the combination of initial states of rebecs and the initial topology, i.e., $s_0 = (s_0^1, \ldots, s_0^n, \gamma_0)$, where for the rebec $r_i = \langle c, e_0^i \rangle$, $s_0^i = (push(\theta_0, stack()), \langle c(e_0^i) \rangle)$ which denotes that the class variables (i.e., V_c) are initialized to the default value, denoted by θ_0, and its queue includes only the message $c(e_0^i)$, and $\gamma_0 \vDash \mathcal{C}$.

To describe the semantics of transitions in wRebeca in Table 4.1, we exploit an auxiliary transition relation $\rightsquigarrow_\gamma \subseteq (Env \times FIFO_1 \times \ldots \times FIFO_n \times Seq(Statement)) \rightarrow (Env \times FIFO_1 \times \ldots \times FIFO_n \times \{\top, \bot\})$ to address the effect of statement executions on the given environment of the rebec (which executes the statements) and the queue of all rebecs. Upon execution, the statements are either successfully terminated, denoted by $\top$, or abnormally terminated, denoted by $\bot$. Let ζ range over $\{\top, \bot\}$. Rule $Term$ explains that an empty statement terminates successfully. The effect of an assignment statement, i.e., $x := expr;$, is that the value of variable x is updated by $eval(expr, \nu_i)$ in ν_i as explained by the rule $Assign$. The variable declaration $T\ x;$ extends the variable valuation corresponding to the current scope by the value assignment $x \mapsto 0$, where 0 is the default value for the types of T, as explained in the rule $VDecl$. The behavior of a block is expressed by the rule $Block$, based on the behavior of the statements (in its scope) on the environment $push(\emptyset, \nu_i)$, where the empty valuation function may be extended by the declarations in the scope (by rule $VDecl$). Thereafter, to find the effect of the block, the last scope is popped from the environment. Rules $Cond_{1,2}$ specify the effect of the if statement: If $eval(expr, \nu_i)$ evaluates to $true$, its effect is defined by the effect of executing the if part, otherwise the $else$ part. Rules $Loop_{1-3}$

explain the effect of the $while$ statement; if the loop condition evaluates to $true$, the effect of the $while$ statement is defined in terms of the effect of its body by the rules $Loop_{1,2}$, otherwise it terminates immediately as specified by the rule $Loop_3$. If the body of the $while$ statement terminates successfully, the effect of the $while$ statement is defined in terms of the effect of the $while$ statement on the resulting environment and queues of its body execution as explained by $Loop_1$. Rule $Loop_2$ expresses that if the body of the $while$ statement terminates abnormally (due to a $break$ statement) while its condition evaluates to $true$, then it terminates successfully while taking the effect of its body execution into account. The effect of a sequence of statements is specified by the rules $Seq_{1,2}$. Upon successful execution of a statement, the effect of its next statements is considered (rule Seq_1). A $break$ statement makes all its next statements be abandoned (rule Seq_2).

The rule $Handle$ expresses that the execution of a wRebeca model progresses when a rebec processes the first message of its queue. In this rule, the message $m(e)$ is processed by the rebec r_i as $s_i.f = m(e) \triangleright f_i$. To process this message, its corresponding message server, i.e. $body(m)$ is executed. The effect of its execution is captured by the transition relation $\rightsquigarrow_\gamma$ on the environment of r_i, updated by the variable assignment $\{fm \mapsto e\}$ for the scope of the message server of m, and the queue of all rebecs while message $m(e)$ is removed from the queue of r_i. The rule Mov specifies that the underlying topology is implicitly changed at the semantic level, and the new topology satisfies $\mathcal{C}$.

The semantics of communication statements is given in Table 4.2. The expression $b \Rightarrow [C_1][C_2]$ in the side conditions of rules $BCast$ and $MCast$ abbreviates $(b \Rightarrow C_1) \wedge (\neg b \Rightarrow C_2)$. The effects of broadcast and multi-cast communications are specified by the rules $BCast$ and $MCast$, respectively: the message $m(eval(expr, \nu_i))$ is appended to the queue of all connected nodes to the sender in case of broadcast, and all connected nodes among the specified receivers (i.e., $rcvs$) in case of multi-cast. Rules $UCast_{1,2}$ express the effect of unicast communication upon its delivery status. If the communication was successful (i.e., the sender was connected to the receiver), the message is appended to the queue of the receiver while the effect of the $succ$ part is also considered (rule $UCast_1$), otherwise only the effect of the $unsucc$ part is considered (rule $UCast_2$).

Example: Consider the global state $(s_1, s_2, s_3, s_4, \gamma)$ such that $s_1 = ((\{\{my\_i \mapsto 3\}\}, \epsilon)$, $s_2 = (\{\{my\_i \mapsto 2\}\}, \epsilon)$, $s_3 = (\{\{my\_i \mapsto 3\}\}, \epsilon)$, $s_4 = (\{\{my\_i \mapsto 3\}\}, \langle send(3,0)\rangle)$, and γ as the one defined in Fig. 4.1a for the wRebeca model in Fig. 4.3 where $\{\{my\_i \mapsto i\}\}$ denotes $push(\{my_i \mapsto i\}, Stack())$. Regarding our rules, the following transition is derived:

$$\dfrac{\dfrac{\dfrac{\nu_2, \overrightarrow{\epsilon}, hopNum++ \rightsquigarrow_\gamma \nu_3, \overrightarrow{\epsilon}, \top \quad \nu_3, \overrightarrow{\epsilon}, send(4,1) \rightsquigarrow_\gamma \nu_3, \overrightarrow{\langle send(4,1)\rangle}, \epsilon, \top}{\nu_2, \overrightarrow{\epsilon}, hopNum++;\ send(4,1); \rightsquigarrow_\gamma \nu_3, \overrightarrow{\langle send(4,1)\rangle}, \epsilon, \top}\ Seq_1}{\nu_1, \overrightarrow{\epsilon}, \{hopNum++; send(4,1)\} \rightsquigarrow_\gamma \nu_4, \overrightarrow{\langle send(4,1)\rangle}, \epsilon, \top \quad : (*)}\ Block$$

The following inference tree uses the result of the first tree, denoted by $(*)$, as a part of its premise to derive the result $(**)$. This result is used by the next tree as a part of its premise to derive the transition.

$$\dfrac{\dfrac{\dfrac{eval(hopNum < 3, \nu_1) = true \qquad (*)}{\nu_1, \overrightarrow{\epsilon}, if\,(hopNum < 3)\ldots \rightsquigarrow_\gamma \nu_4, \overrightarrow{\langle send(4,1)\rangle}, \epsilon, \top}\, Cond_1 \quad \nu_1, \overrightarrow{\epsilon}, if\,(i > my\_i)\ldots \rightsquigarrow_\gamma \nu_1, \overrightarrow{\epsilon}, \top}{\nu_1, \overrightarrow{\epsilon}, \{if\,(i > my\_i)\ldots\} \rightsquigarrow_\gamma \nu_4, \overrightarrow{\langle send(4,1)\rangle}, \epsilon, \top \ : (**)}\, Seq_1, Block$$

$$\dfrac{\dfrac{eval(i! = my\_i, \nu_1) = true \qquad (**)}{\nu_1, \overrightarrow{\epsilon}, if\ (i! = my\_i \ldots \rightsquigarrow_\gamma \nu_1', \overrightarrow{\langle send(4,1)\rangle}, \epsilon, \epsilon, \top}\, Cond_1}{(s_1, s_2, s_3, s_4, \gamma) \xrightarrow{send(4,1)} (s_1', s_2', s_3', s_4', \gamma)}\, Handle$$

where $\overrightarrow{\epsilon}$ and $\overrightarrow{\langle send(4,1)\rangle}$ abbreviate $\epsilon, \epsilon, \epsilon, \epsilon$ (i.e., all the queues are empty) and $\langle send(4,1)\rangle, \langle send(4,1)\rangle, \langle send(4,1)\rangle$, respectively, $\nu_1 = push(\{i \mapsto 3, hopNum \mapsto 0, \}, \{\{my\_i \mapsto 3\}\})$, $\nu_2 = push(\emptyset, \nu_1)$, $\nu_3 = \nu_2[hopNum := 1]$, $\nu_4 = pop(\nu_3)$, $\nu_1' = pop(\nu_4)$, $s_1' = (\{\{my\_i \mapsto 3\}\}, \langle send(4,1)\rangle)$, $s_2' = (\{\{my\_i \mapsto 2\}\}, \langle send(4,1)\rangle)$, $s_3' = (\{\{my\_i \mapsto 3\}\}, \langle send(4,1)\rangle)$, and $s_4' = (\{\{my\_i \mapsto 4\}\}, \epsilon)$. By the rule $Handle$, the message $send(3,0)$ in the queue of n_4 is processed. To this aim, the body of its message server, i.e., $if\ (i! = my\_i$ is executed. Since $eval(i! = my\_i, \nu_1) = true$, by the rule $Cond_1$, the $true$ part (i.e., the sequence of $if\ (i > my\_i)\ \ldots$ and $if\ (hopNum < 3)\ \ldots$) is executed. Due to $eval(hopNum < 3, \nu_1) = true$, by the rule $Cond_1$, the if part is executed.

4.4 Semantic Reduction Techniques

We extend application of the counter abstraction technique to wRebeca models when the topology is static. To this end, the local states of rebecs and their neighborhoods are considered. Later, we inspect the soundness of the counter abstraction technique in the presence of mobility. As a consequence, we propose a reduction technique based on removal of τ-transitions. Recall that the topology is static when the only valid topology of the network constraint is equal to the initial topology.

4.4.1 Applying Counter Abstraction

Assume S_c is the set of local states that the instances of the reactive class c can take (i.e., $S_c = Env_c \times FIFO_c$) and I is the set of rebec identifiers. To apply counter abstraction, rebecs with an identical local state and neighbors that are topologically equivalent are counted together. Two nodes $i, j \in I$ are said to be *topologically equivalent*, denoted by $i \approx_\gamma j$, iff $\forall k \in I \setminus \{i, j\}(k \in \gamma(i) \wedge k \in \gamma(j)$. Intuitively, two topologically equivalent nodes have the same neighbors (except

themselves). So if either one broadcasts, the same set of nodes (except themselves) will receive, and if they are also connected to each other, their counterpart (that is symmetric to the sender) will receive. In a set of pairwise topologically equivalent nodes, all nodes are either all connected to each other, or all disconnected, because they have the same neighbors (except themselves). Therefore, a set of pairwise topologically equivalent nodes will affect the same nodes when either one broadcasts. Hence, a set of pairwise topologically equivalent nodes with an identical local state can be aggregated. To this aim, nodes of the underlying topology are partitioned into the maximal sets of pairwise topologically equivalent nodes, denoted by $\mathcal{N}_1, \ldots, \mathcal{N}_\ell$. We define the set of *distinct local states* as $S^d = \bigcup_{c \in C} S_c$, and the set of topology equivalence classes as $\mathbb{T} = \{\mathcal{N}_1, \ldots, \mathcal{N}_\ell\}$. Consequently, each global state $(s_1, \ldots, s_n, \gamma)$ is abstracted into a vector of elements $(s_i^d, \mathcal{N}_i) : c_i$ where $s_i^d \in S^d$, $\mathcal{N}_i \in \mathbb{T}$, and c_i is the number of nodes in the topology equivalence class $\mathcal{N}_i$ that reside in the very local state s_i^d. The reduced global state, called *abstract global state*, is presented as follows, where n and m denote the number of all rebecs and distinct local states (i.e., $m = \left|S^d\right|$), respectively:

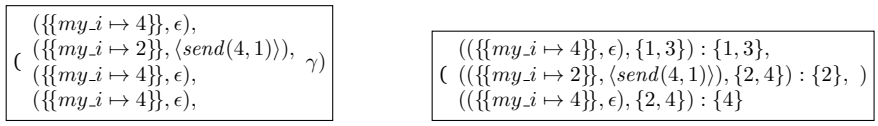
$$S = ((s_1^d, \mathcal{N}_1) : c_1, \ldots, (s_k^d, \mathcal{N}_k) : c_k),\ \forall i \leq k (c_i > 0 \wedge \mathcal{N}_i \in \mathbb{T}), \sum_{i=1}^{k} c_i = n,\ k \leq n$$

For instance, nodes $n_1,\ n_4$, and $n_2,\ n_3$ in Fig. 4.1a have the same neighbors, so if their state variables and queue contents are the same, then they can be counted together.

Recall that when the underlying topology is static, a global state may only change upon processing a message by a rebec, since in wRebeca the bodies of message servers execute atomically. Thus, its corresponding abstract global state may also only change upon processing a message by a rebec.

Counting abstraction is beneficial when the reactive classes do not have a variable that will be assigned uniquely to its instances, such as "unique address" as a state variable. (Note that in the semantics, rebecs have identifiers which are not a part of their local states.) For example, counter abstraction is effective on the specification of the max-algorithm given in Fig. 4.3.

The reduction takes place on-the-fly while constructing the state space. To this end, each global state $(s_1, \ldots, s_n, \gamma)$ is transformed into the form $((s_1^d, \mathcal{N}_1) : n_1, (s_2^d, \mathcal{N}_2) : n_2, \ldots, (s_k^d, \mathcal{N}_k) : n_k)$ such that $n_i \subseteq \mathcal{N}_i$ is the set of node identifiers that are pairwise topologically equivalent with the local state equal to s_i^d, where $\mathcal{N}_i \in \mathbb{T}$. This new presentation of the global state is called *transposed global state*. The sets n_i are leveraged to update the states of the potential receivers (known by the underlying topology) when a communication occurs. To generate the abstract global states, each transposed global state is processed by taking an arbitrary node from the set assigned to a distinct local state and a topology equivalence class if the distinct local state consists of a non-empty queue. The next transposed global state is computed by executing the message handler of



(a) Before applying counter abstraction (b) After applying counter abstraction

Figure 4.4: A global state and its corresponding transposed global state: assume $\{\{i \mapsto e\}\}$ denotes $push(\{i \mapsto e\}, Stack())$

.

the head message in the queue. This is repeated for all the pairs of a distinct local state and a topology equivalence class of the transposed global state. After generating all the next transposed global states of a transposed state, the transposed state is transformed into its corresponding abstract global state by replacing each n_i by $|n_i|$. A transposed global state is processed only if its corresponding abstract global state has not been previously computed. During state space generation, only the abstract global states are stored. Fig. 4.4 illustrates a global state and its corresponding transposed global state, where γ is defined as in Fig. 4.1a. It is assumed that the network consists of four nodes of the reactive class $MNode$ as specified in Fig. 4.3. Each row in Fig. 4.4a represents a local state, i.e., valuation of the local state variable and message queue, while each row in Fig. 4.4b represents a distinct local state and a set of pairwise topologically equivalent identifiers together with those nodes of the set that reside in that distinct local state. As the topology is static, it can be removed from the abstract/transposed global states. Furthermore, each topology equivalence class of nodes can be represented by its unique representative, e.g., the one with the minimum identifier.

The following theorem states that applying counter abstraction preserves semantic properties of the model modulo strong bisimilarity. To this aim, we prove that states that are counted together are strongly bisimilar. For instance, the global state similar to the one in Fig. 4.4a except that the distinct local states of nodes 2 and 4 are swapped, is mapped into the same abstract global state that corresponds to Fig. 4.4b.

Theorem 4.2 (Soundness of Counting Abstraction)**.** *Assume two global states S_1 and S_2 such that for every pair of a distinct local state $s^d \in S^d$ and a topology equivalence class $\mathcal{N} \in \mathbb{T}$, the number of pairwise topologically equivalent nodes of $\mathcal{N}$ that have the local state s^d are the same in S_1 and S_2. Then S_1 and S_2 are strong bisimilar.*

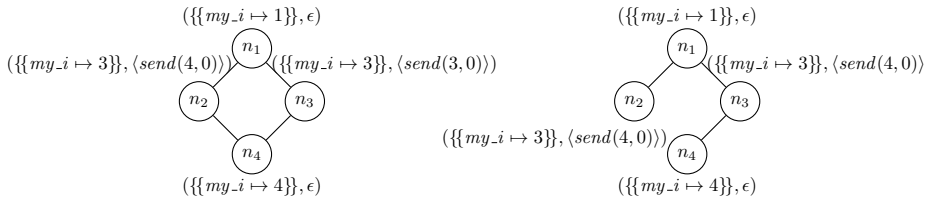
Proof. Since the topology is static, the only transitions of these states are the result of processing messages in their rebec queues. Suppose $S_1 \xrightarrow{m(e)} S_1'$ since there is a node i with the local state (ν_i, f_i) in the topology equivalence class $\mathcal{N}$,

where $m(e)$ is the head of f_i using the semantic rule $Handle$ in Table 4.1. Assume that i belongs to the set of pairwise topologically equivalent nodes $\mathcal{N}_1 \subseteq \mathcal{N}$, where $((\nu_i, f_i), \mathcal{N}) : \mathcal{N}_1$ is an element of the transposed global state corresponding to S_1. Due to the assumption, there exists a set of pairwise topologically equivalent nodes $\mathcal{N}_2 \subseteq \mathcal{N}$ in S_2 with the distinct local state (ν_i, f_i) where $|\mathcal{N}_1| = |\mathcal{N}_2|$. We choose an arbitrary node j in $\mathcal{N}_2$ and prove that it triggers the same transition as i. We claim that for each pair of distinct local state s_k^d and topology equivalence class $\mathcal{N}'$, the number of nodes $nb_i \subseteq \mathcal{N}'$ that are a neighbor of i and reside in the local state s_k^d, is the same to the number of nodes $nb_j \subseteq \mathcal{N}'$ that are a neighbor of j with the local state s_k^d. Assume for the arbitrary transposed global state element $(s_l^d, \mathcal{N}'')$ that, without loss of generality, nb_i has more pairwise topologically equivalent nodes than nb_j in $(s_l^d, \mathcal{N}'')$. As the links are bidirectional, due to the definition of abstract/transposed global states, i is the neighbor of nodes in $\mathcal{N}''$. Furthermore, as the topology is the same for S_1 and S_2 and $i, j \in \mathcal{N}$, then j is also the neighbor of nodes in $\mathcal{N}''$. However, due to the assumption, the number of pairwise topologically equivalent nodes of $\mathcal{N}''$ in S_1 and S_2 that have the distinct local state s_l^d are the same. So there are a set of pairwise topologically equivalent nodes of $\mathcal{N}''$ with the local state s_l^d that are not in nb_j, which contradicts the fact that j is the neighbor of nodes in $\mathcal{N}''$.

As both i and j handle the same message, they execute the same message server, and consequently the effects on their own local state and their neighbors will be the same. Therefore, $S_2 \xrightarrow{m(e)} S_2'$ while $\forall (s_o^d, \mathcal{N}^*)$ the number of pairwise topologically equivalent nodes from the equivalence class $\mathcal{N}^*$ in S_2' that reside in the distinct local state s_o^d is the same as the number of pairwise topologically equivalent nodes from the equivalence class $\mathcal{N}^*$ in S_1' that reside in the distinct local state s_o^d. The same argument holds when $S_2 \xrightarrow{m(e)} S_2'$. □

As mentioned before, the reduction is only applicable if the network is static. This is due to the fact that if node neighborhoods may change, then nodes which are in the same equivalence class in some state may no longer be equivalent in the next state. Consider the max-algorithm protocol (Fig. 4.3) for the two topologies shown in Fig. 4.5a and Fig. 4.5b (satisfying the network constraint in Fig. 4.5c). Nodes N_2 and N_3 are topologically equivalent under topology 1, but not under topology 2.

To illustrate that counter abstraction is not applicable to systems with a dynamic topology, Fig. 4.6 shows a part of the state space of the max-algorithm with a change in the underlying topology (from Fig. 4.5a (γ_1) to Fig. 4.5b (γ_2) with/without applying counter abstraction, where only these two topologies are possible. As predicted, the reduced state space (on the left) is not strongly bisimilar (see Section 2.1 for the definition) to the original state space (on the right). During transposed global state generation, the next state is only generated for node 2 with the distinct local state $(\{\{my\_i \mapsto 3\}\}, \langle send(4,0)\rangle)$ from the equivalence class $\{2,3\}$. Therefore, it is obvious that the next states in the left LTS



(a) Topology 1 (b) Topology 2

$$\mathsf{and}(\mathsf{con}(n_1, n_2), \mathsf{and}(\mathsf{con}(n_1, n_3), \mathsf{and}(\mathsf{con}(n_3, n_4), \mathsf{and}(!\mathsf{con}(n_1, n_4), !\mathsf{con}(n_2, n_3)))))$$

(c) An example of network topology constraint

Figure 4.5: Two possible topologies for the given constraint on the max-algorithm protocol

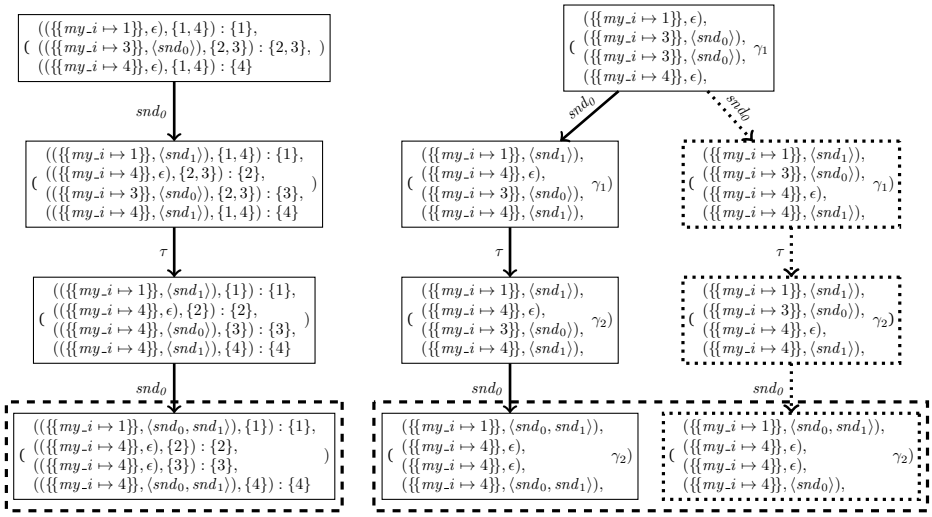


Figure 4.6: Comparing a part of the max-algorithm 's state space with/without applying counter abstraction in a dynamic network. The two states enclosed by a dashed border (in the right) are not strongly bisimilar to the one in the left since in the right figure there is a global state (dotted bordered) in which only one node has snd_0, snd_1 messages in its queue while in the left figure there are two nodes with queues containing snd_0, snd_1 messages. Note that snd_0 and snd_1 stand for $send(4,0)$ and $send(4,1)$, respectively.

of Fig. 4.6 can be matched to the states with the solid borders in the right LTS. However, the solid bordered states are not strongly bisimilar to the dotted ones in the right LTS. The reduced LTS should be strongly bisimilar to its original one to preserve all properties of its original model.

To take a better advantage of the reduction technique, the message storages

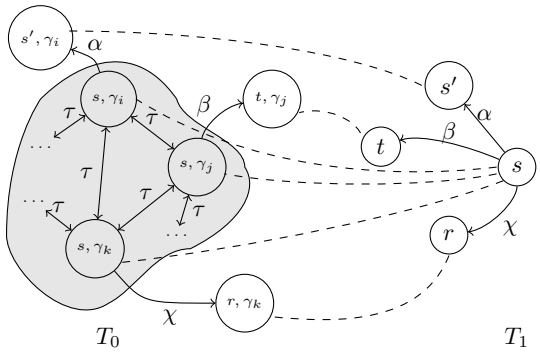


Figure 4.7: Relation $\mathcal{R}$ matches states (s, γ) of T_0 to s of T_1.

can be modeled as bags. However, such an abstraction results in more interleaving of messages which do not necessarily happen in reality, and hence, an effort to inspect if a given trace (of the semantic model) is a valid scenario in reality is needed. This effort is only tolerable if the state space reduces substantially.

4.4.2 Eliminating τ-Transitions

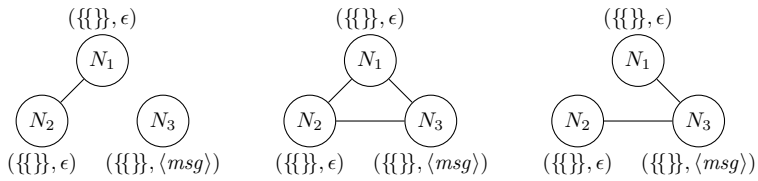
Instead of maintaining the underlying topology as a part of states to derive the MANET behavior and modifying it at each state, modeled by τ-transitions, the behavior of a MANET can be derived with respect to all possible topologies while no topology is kept in the states. To this end, all τ-transitions are eliminated and only those that correspond to processing of messages are kept. The following theorem expresses that removal of τ-transitions and topology information from the global states preserves properties of the original model modulo branching bisimulation, such as ACTL-X [48]. In fact, by exploiting a result from [48] about the correspondence between the equivalence induced by ACTL-X and branching bisimulation, the ACTL-X fragments of μ-calculus and CACTL, which will be introduced in Chapter 5, are also preserved. We show in Section 4.6.3 that important properties of MANET protocols can be still verified over reduced state spaces.

Theorem 4.3 (Soundness of τ-Transition Elimination). *For an LTS* $T_0 \equiv \langle S \times \Gamma, \rightarrow, L, (s_0, \gamma_0)\rangle$*, assume that* $(s,\gamma) \xrightarrow{\alpha} (t,\gamma') \Rightarrow (\gamma = \gamma') \vee (\alpha = \tau \wedge s = t)$*, and* $\forall \gamma, \gamma' \in \Gamma : (s,\gamma) \xrightarrow{\tau} (s,\gamma')$*. If* $T_1 \equiv \langle S, \rightarrow', L, s_0\rangle$*, where* $\rightarrow' = \{(s,\alpha,t) \mid ((s,\gamma),\alpha,(t,\gamma)) \in \rightarrow\}$*, then* $(s_0,\gamma_0) \simeq_{br} s_0$.

Proof. Construct $\mathcal{R} = \{((s,\gamma), s) | s \in S, \gamma \in \Gamma\}$ as shown in Figure 4.7. We show that $\mathcal{R}$ is a branching bisimulation. To this aim, we show that it satisfies the transfer conditions of Definition 2.2. For an arbitrary pair $((s,\gamma), s) \in \mathcal{R}$, assume $(s,\gamma) \xrightarrow{\alpha} (t,\gamma')$.

If $\alpha = \tau$, then two cases can be distinguished: (1) either $\gamma \neq \gamma'$, and hence by definition of T_0, $s = t$ holds which concludes $(t, \gamma')\ \mathcal{R}\ s$, (2) or $\gamma = \gamma'$ and by definition of T_1, $s \overset{\alpha}{\rightarrow}' t$, and $(t, \gamma')\ \mathcal{R}\ t$.

If $\alpha \neq \tau$, then by definition of T_0, $\gamma = \gamma'$ and hence by definition of T_1, $s \overset{\alpha}{\rightarrow}' t$, and $(t, \gamma')\ \mathcal{R}\ t$. Whenever $s \overset{\alpha}{\rightarrow}' t$, then by definition of T_1 there exists γ' such that $(s, \gamma') \overset{\alpha}{\rightarrow} (t, \gamma')$ and hence, $(t, \gamma')\ \mathcal{R}\ t$. Consequently $\mathcal{R}$ is a branching bisimulation relation. □

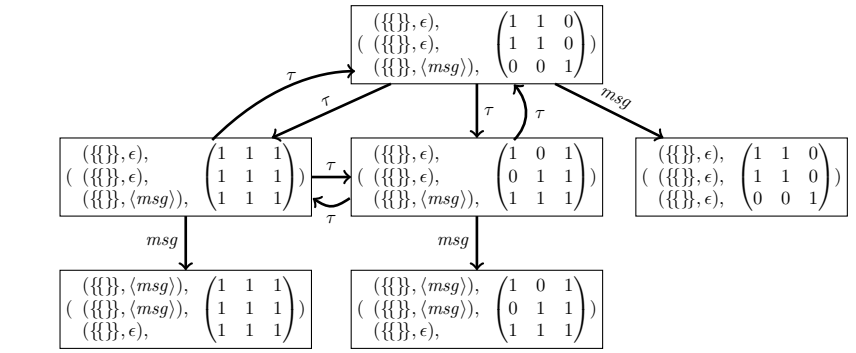


(a) Topology γ_1 (b) Topology γ_2 (c) Topology γ_3

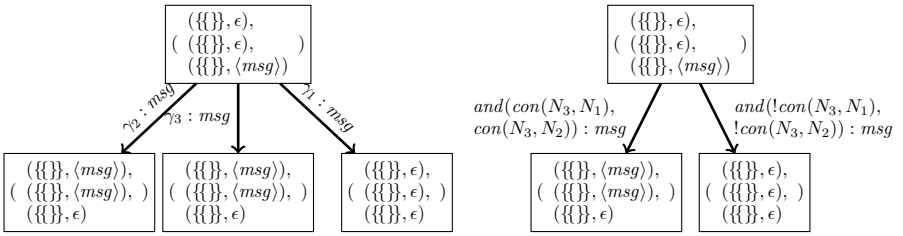
Figure 4.8: All possible topologies considered during state space generation of Fig. 4.9

As an example, consider a network which consists of three nodes, which are the instances of a reactive class with no state variable and only one message, msg. The message server msg has only one statement to broadcast the message msg to its neighbors. We assume that the set of all possible topologies is restricted by a network constraint to the three topologies depicted in Fig. 4.8. Consider the global state in which only N_3 has one msg in its queue.

The state space of the above imaginary model before reduction is presented in Fig. 4.9a, where transitions take place by processing messages or changing the topology. Fig. 4.9b illustrates the state space after eliminating τ-transitions and topology information. Connectivity information is removed from the global states, as in each state its transitions are derived for all possible topologies. In this approach, transition labels are paired with the topology to denote the topology-dependent behavior of transitions. The two transitions labeled with γ_2 and γ_3 can be merged by characterizing the links that make communication from N_3 to N_1 and N_2; i.e., from the sender to the receivers. Such links can be characterized by the network constraints depicted in Fig. 4.9c. In this model, a state is representative of all possible topologies. The resulting semantic model is a CLTS, introduced in Section 2.2, with extended network constraints, introduced in Section 3.1.



(a) State space before reduction



(b) Reduced state space after eliminating τ-transitions and topology information

(c) Reduced state space with labels characterized by network constraints

Figure 4.9: State space before and after applying reduction

4.5 Modeling the AODVv2 Protocol

To illustrate the applicability of the proposed modeling language, the AODVv2-11[2] protocol has been modeled[3]. The AODV is a popular routing protocol for wireless ad hoc networks, first introduced in [125], and later revised several times.

In this algorithm, routes are constructed dynamically whenever requested. Every node has its own routing table to maintain information about the routes of the received packets. When a node receives a packet (whether it is a route discovery or data packet), it updates its own routing table to keep the shortest and freshest path to the source or destination of the received packet. Three different tables are used to store information about neighbors, routes and received messages:

- neighbor table: keeps the adjacency states of the node's neighbors. The neighbor state can be one of the following values:

[2] https://tools.ietf.org/html/draft-ietf-manet-aodvv2-11

[3] The specification of AODVv2-11 in [153] maintains only one route for each destination and it abstracts away from the neighbor table and the state of a route.

  - *Confirmed: indicates that a bidirectional link to that neighbor exists. This state is achieved either through receiving a rrep message in response to a previously sent rreq message, or a RREP_Ack message as a response to a previously sent rrep message (which requested an RREP_Ack) to that neighbor.*
  - *Unknown: indicates that it is currently unknown whether there is a link to that neighbor. Initially, the states of the links to the neighbors are unknown.*
  - *Blacklisted: indicates that the link to that neighbor is unidirectional. When a node has failed to receive the RREP_Ack message in response to its rrep message to that neighbour, the neighbor state is changed to blacklisted. Hence, it stops forwarding any message to it for an amount of time, ResetTime. After reaching the ResetTime, the neighbor's state will be set to unknown.*

- route table: contains information about discovered routes and their status. The following information is maintained for each route:
  - *SeqNum: destination sequence number*
  - *route_state: the state of the route to the destination which can have one of the following values:*
    - unconfirmed*: when the neighbor state of the next hop is unknown;*
    - active*: when the link to the next hop has been confirmed, and the route is currently used;*
    - idle*: when the link to the next hop has been confirmed, but it has not been used in the last active interval;*
    - invalid*: when the link to the next hop is broken, i.e., the neighbor state of the next hop is blacklisted.*
  - *Metric: indicates the cost or quality of the route, e.g., hop count, the number of hops to the destination*
  - *NextHop: IP address of the next hop to the destination*
  - *Precursors (optional feature): the list of the nodes interested in the route to the destination, i.e., upstream neighbors.*

- route message table, also known as *RteMsg Table*: contains information about previously received route messages such as $rreq$ and $rrep$, so that it can be determined whether the new received message is worth processing or redundant. Each entry of this table contains the following information:
  - *MessageType: which can be either* $rreq$ *or* $rrep$
  - *OrigAdd: IP address of the originator*
  - *TargAdd: IP address of the destination*

- *OrigSeqNum: sequence number of the originator*
- *TargSeqNum: sequence number of the destination*
- *Metric*

When one node, the source, intends to send a package to another, the destination, it looks in its routing table for a valid route to that destination, i.e., a route of which the route state is not invalid. If there is no such route, it initiates a route discovery procedure by broadcasting a $rreq$ message. The freshness of the requested route is indicated through the sequence number of the destination that the source is aware of. Whenever a node initiates a route discovery, it increases its own sequence number, with the aim to define the freshness of its route request. Every node upon receiving this message checks its routing table to find a route to the requested destination. If there is such a path or the receiver is in fact the destination, it informs the sender through unicasting a $rrep$ message. However, an acknowledgment from the receiver (of $rrep$) is requested whenever the neighbor state of the next hop is *unconfirmed*. Otherwise, it re-broadcasts the $rreq$ message to examine if any of its neighbors has a valid path. Meanwhile, a reverse forwarding path is constructed to the source over which $rrep$ messages are going to be communicated later. In case a node receives a $rrep$ message, if it is not the source, it forwards the $rrep$ after updating its routing table with the received route information. Whenever a node fails to receive the requested acknowledgment, it uses a $rerr$ message to inform all its neighbors intending to use the broken link to forward their packets.

In our model, each node is represented through a rebec (actor), identified by an IP address, with a routing table and a sequence number (sn). In addition, every node keeps track of the adjacency status to its neighbors by means of a neighbor table, through the $neigh\_state$ array, where $neigh\_state[i] = true$ indicates that it is adjacent to the node with the IP address i, while $false$ indicates that its adjacency status is either *unknown* or *blacklisted* (since timing issues are not taken into account, these two statuses are considered the same). As the destinations of any two arbitrary rows of a routing table are always different, the routing table has at most n rows, where n is the number of nodes in the model. It should be noted that more than one next hop for each destination is maintained to increase the probability of packet delivery; if one route gets broken, there may be other routes that can be used as an alternative. Therefore, the routing table is modeled by a set of arrays, namely, dsn, $route\_state$, $hops$, $nhops$, and $pres$, to represent the $SeqNum$, $route\_state$, $Metric$, $NextHop$, and $Precursors$ columns of the routing table, respectively. The arrays dsn and $route\_state$ are of size n, while the arrays $hops$, $nhops$, and $pers$ are of size $n \times n$.

- dsn: destination sequence number, for instance, $dsn[i]$ keeps the sequence number of the destination with IP i
- $route\_state$: an integer that refers to the state of the route to the destination and can have one of the following values:

  - $route\_state[i] = 0$: all the routes to the destination i are *unconfirmed*;
  - $route\_state[i] = 1$: there is a *valid* route via a next hop that its link has been confirmed, the route state in the protocol is either **active** or **idle**; since we abstract from the timing issues, these two states are depicted as one. Although there can exist more than one *unconfirmed* route to each destination, there can be only one *valid* route to each destination. When a route state to a destination gets changed to *valid*, all the routes to the same destination are removed from routing table;
  - $route\_state[i] = 2$: all the routes to the destination i are *invalid*, their links to the next hops are broken;

- $hops$: the number of hops to the destination for different routes, for example $hops[i][j]$ indicates the number of hops to the destination i from over the j-th route.

- $nhop$: IP address of the next hop to the destination for different routes,for instance $nhops[i][j]$ contains the next hop of the j-th route to the destination with the IP address i.

- $pres$: an array that indicates which of the nodes are interested in the routes to the destination, for example $pres[i][j] = true$ indicates that the node with the IP address j is interested in the routes to the node with the IP address i.

Since we have considered a row for each destination in our routing table, to indicate whether the node has any route to each destination until now, we initially set $dsn[i]$ to -1 which implies that the node has never known any route to the node with the IP address i. We refer to all the arrays mentioned above as *routing arrays*. Initially all integer cells of arrays are set to -1 and all boolean cells are set to $false$. To model expunging a route, its corresponding next hop and hop count entries in the arrays $nhops$ and $hops$ are set to -1. Since we have only considered one node as the destination and one node as the source, the information in $rreq$ and $rrep$ messages has no conflict and consequently the route message table can be abstracted away. In other words, the routing table information can be used to identify whether the newly received message has been seen before or not, as the stored routes towards the source represent information about $rreq$s and the routes towards the destination represent $rrep$s.

Note that $rreq$ and $rrep$, i.e., all route messages, carry route information to their source and destination, respectively. Therefore, a bidirectional path is constructed while these messages travel through the network. Whenever a node receives a route message, it processes incoming information to determine whether it offers any improvement to its known existing routes. Then, it updates its routing table accordingly in case of an improvement. The processes of evaluating and updating the routing table are explained in the following subsections.

4.5.1 Evaluating Route Messages

Every received route message contains a route and consequently is evaluated to check for any improvement. Note that a $rreq$ message contains a route to its source while a $rrep$ message contains a route to its destination. Therefore, as the routes are identified by their destinations (denoted by des), in the former case, the destination of the route is the originator of the message (i.e., $des = oip\_$), and in the latter, it is the destination of the message (i.e., $des = dip\_$). The routing table must be evaluated if one of the following conditions is realized:

1. no route to the destination has existed, i.e., $dsn[des] = -1$
2. there are some routes to the destination, but all their route states are *unconfirmed*
3. there is a valid or invalid route to the destination in the routing table and one of following conditions holds:
   - the sequence number of the incoming route is greater than the existing one
   - the sequence number of the incoming route is equal to the existing one, however the hop count of the incoming route is less than the existing one (the new route offers a shorter path).

4.5.2 Updating the Routing Table

The routing table is updated as follows:

- if no route to the destination has existed, i.e., $dsn[des] = -1$, the incoming route is added to the routing table.
- if the route states of existing routes to the destination are *unconfirmed*, the new route is added to the routing table.
- if the route state of the existing route is *valid* while the next hop's neighbor state of the incoming route is *unknown*, the new route is added to the routing table since it may offer an improvement in the future and turn into *confirmed*.
- if the existing route state is *invalid* and the neighbor state of the next hop of the incoming route is *unknown*, the existing route should be updated with information of the received one.
- if the next hop's neighbor state of the incoming route is *confirmed*, the existing route is updated with new information and all other routes with the route state *unconfirmed* are expunged from the routing table.

As described earlier, there are three types of route discovery packets: $rreq$, $rrep$ and $rerr$. There is a message server for handling each of these packet types:

- rec_rreq is responsible for processing a route discovery request message;
- rec_rrep handles a reply request message;
- rec_rerr updates the routing table in case an error occurs over a path and informs the interested nodes about the broken link.

There are also two message servers for receiving and sending a data packet. All these message servers will be discussed thoroughly in the following subsections.

4.5.3 $rreq$ Message Server

This message server processes a received route discovery request and reacts based on its routing table, shown in Fig. 4.12. The $rreq$ message has the following parameters: $hops\_$ and $maxHop$ as the number of hops and the maximum number of hops, $dsn\_$ as the destination sequence number, and $oip\_$, $osn\_$, dip, and $sip\_$ respectively refer to the IP address and sequence number of the originator, the IP address of the destination, and the IP address of the sender. Whenever a node receives a route request, i.e., $rec\_rreq(hops\_, dip\_, dsn\_, oip\_, osn\_, sip\_, maxHop)$ message, it checks incoming information with the aim to improve the existing route or introduce a new route to the destination, and then updates its routing table accordingly (see also Sections 4.5.1 and 4.5.2). During processing a $rreq$ message, a backward route, from the destination to the originator is built by manipulating the routing arrays with the index $oip\_$. Similarly, while processing a $rrep$ message, it constructs a forwarded route to the destination by addressing the routing arrays with the index $dip\_$. Therefore, the procedure of evaluating the new route and updating the routing table is the same for both $rreq$ and $rrep$ messages, except for different indices $oip\_$ and $dip\_$, respectively.

Updating the routing table: Fig. 4.10 depicts this procedure which includes both evaluating the incoming route and updating the routing table (the code is the body of if-part in the line 7 of Fig. 4.12). If no route exists to the destination, the received information is used to update the routing table and generate discovery packets, lines (1-12). The route state is set based on the neighbor status of the sender: if its neighbor status is *confirmed*, the route state is set to *valid*, otherwise to *unconfirmed*. The next hop is set to the sender of the message, i.e., $nhop[oip\_][0] = sip\_$. If a route exists to the destination (i.e., $oip\_$), one of the following conditions happens:

- the route state is *unconfirmed*, lines (15-44): it either updates the routing table if there is a route with a next hop equal to the sender, or adds the incoming route to the first empty cells of $nhop$ and $hops$ arrays. If the

neighbor status of the sender is *confirmed*, then all other routes with the same destination are expunged while the route state is set to *valid*, lines (28-37).

- the route state is *invalid* or it is valid, but the neighbor status of the sender is *confirmed*, lines (48-60): if the incoming message contains a greater sequence number, or an equal sequence number with a lower hop count, then it updates the current route while a new discovery message is generated.
- the route state is *valid* and the neighbor status of the sender is *unknown*, lines (64-72): the incoming route is added to the routing table and a new discovery message is generated if it provides a fresher or shorter path.

In these cases, if a new discovery message should be generated (when the node has no route as fresh as the route request), the auxiliary boolean variable gen_msg is set to $true$. In Fig. 4.12, after updating the routing table, if a new message should be generated, indicated by $if\ (gen\_msg = true)$, it rebroadcasts the $rreq$ message with the increased hop count if the node is not the destination, lines (51-54). Otherwise, it increases its sequence number and replies to the next hop(s) toward the originator of the route request, $oip\_$, based on its routing table. Before unicasting $rrep$ messages, next hops toward the destination, $dip\_$, and the sender are set as interested nodes to the route toward the originator, $oip\_$, lines (17-23). It unicasts each $rrep$ message to its next hops one by one until it gets an ack from one, lines (24-44); ack reception is modeled implicitly through successful delivery of unicast, i.e., the $succ$ part. If it receives an ack, it updates the route state to *valid* and the neighbor status of the next hop to *confirmed* and stops unicasting $rrep$ messages. If it doesn't receive a $RREP\_Ack$ message from the next hop when the route state is *valid*, it initiates the error recovery procedure.

Error Recovery Procedure: The code for this procedure is illustrated in Fig. 4.11 (its code is the body of if-part in line 44 of Fig. 4.12). As explained earlier, this procedure is initiated when a node doesn't receive a $RREP\_Ack$ message from the next hop of the route with state *valid*. Then, it updates its route state to *invalid* and adds the sequence number of the originator to the array of invalidated sequence numbers, denoted by dip_sqn. Furthermore, it adds all the interested nodes in the current route to the list of affected neighbors, denoted by $affected\_neighbours$, lines (3-7). It invalidates other valid routes that use the same broken next hop as their next hops, adds their sequence numbers to the invalidated array and sets the nodes interested in those routes as affected neighbors, lines (8-26). Finally, it multicasts a $rerr$ message which contains the destination IP address, the node IP address, and the invalidated sequence numbers to the affected neighbors, line 27.

```
1  if (dsn[oip_]==-1) {
2      dsn[oip_]=osn_;
3      if (neigh_state[sip_]==true) { route_state[oip_]=1; }
4      else  { route_state[oip_]=0; }
5      hops[oip_][0]=hops_;nhop[oip_][0]=sip_;
6      gen_msg = true;
7  } else {
8      if (route_state[oip_]==0) {
9          dsn[oip_]=osn_;
10         route_num = 0;
11         for(int i=0;i<4;i++){
12             if (nhop[oip_][i]==-1 || nhop[oip_][i]==sip_) {
13                 route_num = i; break; }}
14         if (neigh_state[sip_]==true) {
15             route_state[oip_]=1;
16             for(int i=0;i<4;i++) {
17                 hops[oip_][i]=-1;nhop[oip_][i]=-1;}
18             hops[oip_][0]=hops_;nhop[oip_][0]=sip_;
19         } else {
20             route_state[oip_]=0;
21             hops[oip_][route_num]=hops_;
22             nhop[oip_][route_num]=sip_; }
23     }  else {
24         if (route_state[oip_]==2 || neigh_state[sip_]==true) {
25             /* update the existing route */
26             if ((dsn[oip_]==osn_ && hops[oip_][0]>hops_) || dsn[oip_]<osn_ ){
27                 dsn[oip_]=osn_;
28                 if (neigh_state[sip_]==true) route_state[oip_]=1;
29                   else route_state[oip_]=0;
30                 hops[oip_][0]=hops_;
31                 nhop[oip_][0]=sip_;
32                 gen_msg = true; }
33         } else {
34       route_num = 0;
35       for(int i=0;i<4;i++){
36         if (nhop[oip_][i]==-1 || nhop[oip_][i]==sip_){
37           route_num = i; break;}}
38       if ((dsn[oip_]==osn_ && hops[oip_][0]>hops_) || dsn[oip_]<osn_ ) {
39         dsn[oip_]=osn_;
40         hops[oip_][route_num]=hops_;
41         nhop[oip_][route_num]=sip_;
42         gen_msg = true;}}}}
```

Figure 4.10: Updating the routing table

```
1  if(route_state[oip_]==1){
2    route_state[oip_]=2;
3    dip_sqn[oip_]=dsn[oip_];
4    for(int k=0;k<4;k++){
5      if(pre[oip_][k]==true) { affected_neighbours[k]=true; } }
6    for(int j=0;j<4;j++){
7      for(int r=0;r<4;r++){
8        if(nhop[oip_][r]!=-1 && nhop[j][0]==nhop[oip_][r]) {
9          route_state [j]= 2;
10         dip_sqn[j]=dsn[j];
11         for(int k=0;k<4;k++){
12           if(pre[j][k]==true) { affected_neighbours[k]=true; } }
13       break;}
14     }
15   }
16   multicast(affected_neighbours, rec_rerr(dip_, ip, dip_sqn));
17 }
```

Figure 4.11: The error recovery procedure

4.5.4 *rrep* Message Server

This message server, shown in Fig. 4.13, processes the received reply messages and also constructs the route forward to the destination. At first, it updates the routing table and decides whether the message is worth processing, as previously mentioned for *rreq* messages, and constructs the route, but this time to the destination (its code is similar to the one in Fig. 4.10 except that $dip\_$ is used instead of $oip\_$, and is given at line 6 of Fig. 4.13). This message is sent backwards till it reaches the source through the reversed path constructed while broadcasting the *rreq* messages. When it reaches the source, it can start forwarding data to the destination. In case the node is not the originator of the route discovery message, it updates the array of interested nodes, lines (17-23). Then, it unicasts the message to the next hop(s), on the reverse path to the originator, lines (24-42). Based on the AODVv2 protocol, if connectivity to the next hop on the route to the originator is not confirmed yet, the node must request a Route Reply Acknowledgment (*RREP_Ack*) from the intended next hop router. If a *RREP_Ack* is received, then the neighbor status of the next hop and route state must be updated to *confirmed* and *valid*, respectively, lines (30-36), otherwise the neighbor status of the next hop remains *unknown*, lines (37-40). This procedure is modeled through *conditional unicast* which enables the model to react based on the delivery status of the unicast message so that $succ$ models the part where the *RREP_Ack* is received while $unsucc$ models the part where it fails to receive an acknowledgment from the next hop. In case the unicast is unsuccessful and the

```
1  msgsrv rec_rreq(int hops_,int dip_ ,int dsn_ ,int oip_ ,int osn_ ,int sip_,int
       maxHop)
2  {
3    int[]  dip_sqn=new int[4];
4    int route_num;
5    bool[] affected_neighbours=new bool[4];
6    bool gen_msg = false;
7    if(ip!=oip_){
8    //evaluate and update the routing table
9    }
10   if(gen_msg==true){
11     if(ip==dip_){
12       bool su = false;
13       pre[dip_][sip_]=true;
14       for(int i=0;i<4;i++){
15         int nh = nhop[dip_][i];
16         if(nh!=-1) { pre[oip_][nh]=true; }
17       }
18       for(int i=0;i<4;i++){
19         if(nhop[oip_][i]!=-1){
20         int n_hop = nhop[oip_][i];
21         sn  = sn+1;
22         /* unicast a RREP towards oip of the RREQ */
23         unicast(n_hop,rec_rrep(0 , dip_ , sn , oip_ , self))
24           succ:{
25             route_state[oip_]=1;
26             neigh_state[n_hop]=true;
27             su = true;
28             break;
29           }
30           unsucc:{neigh_state[n_hop]=false;}
31         }
32       }
33       if(su==false && route_state[oip_]==1){
34         /* error recovery procedure */
35       }
36     } else {
37       hops_ = hops_+1;
38       if(hops_<maxHop) { rec_rreq(hops_,dip_,dsn_,oip_,osn_,self,maxHop); }
39     }
40   }
41 }
```

Figure 4.12: The *rreq* message server

route state is valid, the error recovery procedure will be followed, lines (43-46).

4.5.5 $rerr$ Message Server

This message server, shown in Fig. 4.14, processes the received error messages and informs those nodes that depend on the broken link. When a node receives a $rerr$ message, it must invalidate those routes using the broken link as their next hops and sends the $rerr$ message to those nodes interested in the invalidated routes. This message has only two parameters: $sip\_$ which indicates the IP address of the sender, and rip_rsn, which contains the sequence number of those destinations which have become unaccessible from the $sip\_$.

For all the *valid* routes to the different destinations, it examines whether the next hop of the route to the destination is equal to $sip\_$ and the sequence number of the route is smaller than the received sequence number, line 10. In case the above conditions are satisfied, the route is invalidated, lines (11-19), and a $rerr$ message is sent to the affected nodes, line 21.

4.5.6 $newpkt$ Message Server

Whenever a node intends to send a data packet, it creates a rec_newpkt which has only two parameters, $data$ and $dip\_$. The code for this message server is shown in Fig. 4.15. If it is the destination of the message, it delivers the message to itself, lines (4-7). Otherwise, if it has a valid route to the destination, it sends data using that route, lines (12-16). If it has no valid route, it increases its own sequence number and broadcasts a route request message, lines (18-26). In addition, if a route to the destination is not found within $RREQ\_WAIT\_TIME$, the node retries to send a new $rreq$ message after increasing its own sequence number. Since we abstracted away from time, we model this procedure through the $resend\_rreq$ message server which attempts to resend a $rreq$ message while the node sequence number is smaller than 3 (to make the state space finite).

4.6 Evaluation

In this section, we will review the results obtained from efficiently constructing the state spaces for the two wRebeca models, the flooding and AODV protocol. Also, we briefly introduce our tool and its capabilities. Then, the loop freedom invariant is defined and one possible loop scenario is demonstrated. Finally, two properties that must hold for the AODV protocol are expressed and checked with regard to the AODV model.

Example: The flooding protocol is one of the earliest methods used for routing in wireless networks. The flooding protocol modeled in wRebeca is presented in Fig. 4.16. Every node upon receiving a packet checks whether it is the packet's destination. If so it processes the message, otherwise it broadcasts the message to its neighbors.

```
1  msgsrv rec_rrep(int hops_ ,int dip_ ,int dsn_ ,int oip_ ,int sip_) {
2    int[]  dip_sqn=new int[4];
3    bool[] affected_neighbours=new bool[4];
4    bool gen_msg = false;
5    int n_hop,route_num;
6    /*evaluate and update the routing table */
7    if(gen_msg==true){
8      if(ip==oip_){
9        /* this node is the originator of the corresponding RREQ */
10       /* a data packet may now be sent */
11     } else {
12       hops_  = hops_+1;
13       bool su = false;
14       pre[dip_][sip_]=true;
15       for(int i=0;i<4;i++){
16         int nh = nhop[dip_][i];
17         if(nh!=-1){ pre[oip_][nh]=true; }
18       }
19       for(int i=0;i<4;i++){
20         if(nhop[oip_][i]!=-1){
21           n_hop = nhop[oip_][i];
22           unicast(n_hop,rec_rrep(hops_,dip_,dsn_,oip_, self))
23           succ:{
24             route_state[oip_]=1;
25             neigh_state[n_hop]=true;
26             su = true;
27             break;
28           }
29           unsucc: {neigh_state[n_hop]=false;}
30         }
31       }
32       if(su==false && route_state[oip_]==1){
33       /* error recovery procedure */
34       }
35     }
36   }
37 }
```

Figure 4.13: The rrep message server

```
1  msgsrv rec_rerr(int source_, int sip_ ,int[] rip_rsn) {
2    int[]  dip_sqn=new int[4];
3    bool[] affected_neighbours=new bool[4];
4    if(ip!=source_) {
5      //regenerate rrer for invalidated routes
6      for(int i=0;i<4;i++){
7        int rsn=rip_rsn[i];
8        if(route_state[i]==1 && nhop[i][0]==sip_ && dsn[i]<rsn && rsn!=0){
9          route_state [i]= 2;
10         dip_sqn[i]=dsn[i];
11         for(int j=0;j<4;j++){
12           if(pre[i][j]==true){ affected_neighbours[j]=true; }
13         }
14       }
15     }
16     multicast(affected_neighbours, rec_rerr(source_, self ,dip_sqn));
17   }
18 }
```

Figure 4.14: The rerr message server

4.6.1 State Space Generation

Static Network . Consider a network with a static topology, in other words the network constraint is defined so that it leads to only one valid topology. We illustrate the applicability of our counter abstraction technique on the flooding routing protocol. In contrast to the intermediate nodes on a path (the ones except the source and destination), the source and destination nodes cannot be aggregated with other nodes, due to their local states. However, in the case of the AODV protocol, no two nodes can be counted together due to the unique variables of IP address and routing table of each node. As the number of intermediate nodes with the same neighbors increases, more reduction takes place. We have chosen four fully connected network topologies to show the power of our reduction technique when the number of intermediate nodes increases from one to four.

Table 4.3 shows the number of states when running the flooding protocol on different networks with different topologies, distinguished by different number of intermediate nodes. By applying counter abstraction reduction, the intermediate nodes are aggregated as they have the same role in the protocol. However, the effectiveness of this technique depends on the network topology and the modeled protocol.

Dynamic network. In this case, the topology is constantly changing, in other words there is more than one possible topology. The resulting state spaces before

```
1  msgsrv rec_newpkt(int data ,int  dip_) {
2    int[]   dip_sqn=new int[4];
3    bool[] affected_neighbours=new bool[4];
4    if (ip==dip_){
5      /* the DATA packet is intended for this node */
6    }
7    else{
8      /* the DATA packet is not intended for this node */
9      store[dip_]=data;
10     if (route_state[dip_]==1){
11       /* valid route to dip*/
12       /* forward packet */
13     }else{
14       /* no valid route to dip*/
15       /* send a new rout discovery request*/
16       if (sn<3){
17         sn++;
18         unicast( self ,resend_rreq(dip_));
19         rec_rreq (0,dip_,dsn[dip_], self ,sn, self ,4);
20       }
21     }
22   }
23 }
24 msgsrv resend_rreq(int dip_){
25   if (sn<3){
26     sn++;
27     unicast( self ,resend_rreq(dip_));
28     rec_rreq (0,dip_,dsn[dip_], self ,sn, self ,4);
29   }
30 }
```

Figure 4.15: The rec_newpkt message server

Table 4.3: Comparing the size of states (St.) and transitions (Tr.) with/without applying counter abstraction reduction for different number of intermediate (inter.) nodes

No. of inter. nodes	St. after reduction	Tr. after reduction	St. before reduction	Tr. before reduction
1	24	24	36	36
2	226	133	574	276
3	3,689	912	13,197	2,441
4	71,263	6,649	321,419	21,466

```
1  reactiveclass Node
2  {
3    statevars
4    {
5      boolean destination;
6    }
7    msgsrv initial (boolean
         source,boolean dest)
8    {
9      destination=dest;
10     if (source==true)
11       relay_packet(55,0);
12   }
13   msgsrv relay_packet(int
         data,int hopNum)
14   {
15     if (destination==true)
16       deliver_packet(data);
17     else if (hopNum<3)
18     {
19       hopNum++;
20       relay_packet(data,hopNum);
21     }
22   }
23   msgsrv deliver_packet(int data)
24   {
25     // do nothing
26   }
27 }

29 main
30 {
31   Node node0 (node1)
32     :(true, false);
33   Node node1
34     (node0,node2,node3)
35   :(false, false);
36   Node node2 (node1,node3)
37     :(false, false);
38   Node node3 (node1,node2)
39     :(false,true);

41 }
```

Figure 4.16: The flooding protocol [153]

and after eliminating τ-transitions are compared for the two case studies while the topology is constantly changing for networks of 4 and 5 nodes, as shown in Table 4.4. Table 4.5 depicts the constraints used to generate the state spaces and the number of topologies that each constraint results in. Constraints are chosen randomly here, just to show the effectiveness of our reduction technique. To this aim, we have randomly removed a (fixed) link from the network constraints. Nevertheless, constraints can be chosen wisely to limit the network topologies to those which are prone to lead to an erroneous situation, i.e., violation of a correctness property like loop freedom. However, it is also possible to check the model against all topologies by not defining any constraint. In other words, a modeler at first can focus on some suspicious network topologies and after resolving the raised issues check the model for all possible topologies. There are also some networks that have certain constraints on how the topology can change, e.g., node 1 can never get into the communication range of node 2. These restrictions on topology changes can be reflected by constraints too. The sizes of state spaces are compared under different network constraints resulting in different numbers of valid topologies. Eliminating τ-transitions and topology information manifestly reduces the number of states and transitions even without restriction on the possible topologies. Therefore, it makes MANET protocol verification possible in an efficient manner. Note that in case the size of the net-

Table 4.4: Comparing the size of states (St.) and transitions (Tr.) before/after applying τ-transition elimination reduction for different numbers of nodes and valid topologies (topo.)

No. of	nodes	Valid topo.	St. before reduction	Tr. before reduction	St. after reduction	Tr. after reduction
Flooding protocol	4	4	2,119	11,724	541	1,652
	4	8	4,431	42,224	567	1,744
	4	16	10,255	179,936	655	2,192
	4	32	22,255	747,200	710	2,765
	4	64	44,495	2,917,728	710	3,145
AODV protocol	4	4	3,007	16,380	763	1,969
	4	8	12,327	113,480	1,554	3,804
	4	16	35,695	610,816	2,245	5,549
	4	32	93,679	3,097,792	2,942	7,596
	4	64	258,447	16,797,536	4,053	10,629
	5	16	-	-	165,959	598,342

Table 4.5: Applied network constraints

No. of nodes	No. of valid topologies	constraint
4	4	$and(and(con(node0, node1), con(node0, node3)),$ $and(con(node2, node3), con(node1, node3)))$
4	8	$and(and(con(node0, node1),$ $con(node0, node3)), con(node2, node3))$
4	16	$and(con(node0, node1), con(node2, node3))$
4	32	$con(node0, node1)$
5	16	$and(and(con(node0, node1), and(con(node0, node3),$ $con(node4, node1))), and(con(node2, node3),$ $and(con(node1, node3), con(node2, node4))))$

work was increased from four to five, we couldn't generate its state space without applying reduction due to the memory limitation on a computer with 8GB RAM.

4.6.2 Tool Support

The presented modeling language and reduction techniques are supported by a tool, implemented in Java[4]. A screen-shot of this tool is given in Fig. 4.17. After opening a model, the tool extracts the information of the reactive classes, such as the state variables and message servers, and also the main part including

[4]The tool and the source codes of examples are available at https://github.com/b-yousefi/wRebeca

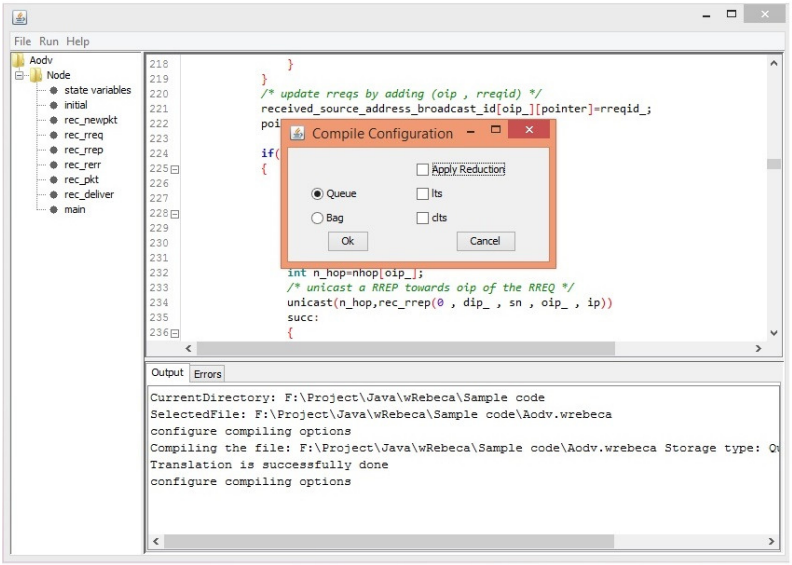


Figure 4.17: A screen-shot of the wRebeca tool with the *compilation info* window to configure the state space generator

the rebec declarations and the network constraint. Then it generates several classes in the Java language based on the obtained information and compiles them together with some abstract and base classes (common in all models), for example *global state* and *topology*, to build an engine that constructs the model state space upon its execution. Before compiling, a user can select the reduction method. To take advantage of all hardware capabilities, we have implemented our state space generation algorithm in a multi-threaded way to leverage the power of multi-core CPUs.

During state space generation, information about the state variables and transitions are stored as an LTS in the Aldebaran format[5]. This LTS can be evaluated by tools such as the mCRL2 toolset[6]. For example, one can express desired properties in μ-calculus [112] and verify them. Also, as explained in Section 4.4, labels are extended with network constraints as defined in Section 3.6.2 so that the reduced LTS is turned into a CLTS.

4.6.3 Model Checking of the AODV Protocol Properties

There are different ways to check a given property on a wRebeca model. Invariant properties can be evaluated while generating the state space by checking

[5] http://cadp.inria.fr/man/aldebaran.html
[6] http://www.mcrl2.org/

each reached global state against defined invariants. Furthermore, the resulting state space can be model checked by tools supporting the Aldebaran format such as mCRL2.

Checking the loop freedom invariant Loop freedom is one of the well-known properties which must hold for all routing protocols such as the AODV protocol. For example, consider the routes to a destination x in the routing tables of all nodes, where $node_0$ has a route to x with the next hop $node_1$, $node_1$ has a route to x with the next hop $node_2$, and $node_2$ has a route to x with the next hop $node_0$. The given example constructs a loop which consists of the three nodes, $node_0$, $node_1$, and $node_2$. A state is considered *loop free* if the collective routing table entries of all nodes for each pair of a source and destination do not form a loop. As was mentioned earlier in AODVv2-11, each route may have more than one next hop when the adjacency states of the next hops are *unconfirmed*. Therefore, while the loop freedom of a state is checked, one must take into account all next hops stored for each route. Then, for each next hop it must be checked whether it leads to a loop or not. A routing protocol deployed on a network is called loop free if all of its states are loop free. In other words, the loop freedom property of a protocol is an invariant (which can be easily specified by the ACTL-X fragment of μ-calculus, and hence, is preserved by the τ-elimination reduction method). To facilitate checking such invariants, we extend our state space generator engine (produced by our tool) to check the loop freedom property of each newly generated global state on-the-fly. To this aim, we specified a recursive function to determine whether in a global state the next hops in different nodes collectively lead to a loop scenario, as shown in Fig. 4.18, as a part of the state generator class. Whenever a new state is reached, before proceeding any further, it is checked whether no loop is formed on the forward/backward routes between the source and destination, by calling $loop\_freedom(4, 1, \text{new Set}\langle int\rangle(1))$ and $loop\_freedom(1, 4, \text{new Set}\langle int\rangle(4))$, as $node_4$ and $node_1$ are the destination and source respectively. If the loop freedom condition is violated, the $loop\_freedom$ function returns $false$, and the state generator engine doesn't process the new global state and it returns a path which has led to that global state. The function $loop\_freedom$ has three parameters: des refers to the destination of the route, cur refers to the IP address of the current node which is going to be processed, and $visited$ is the list of IP addresses of those nodes which have been processed.

Although keeping more than one next hop for each route may increase the route availability, it compromises the validity of the routing tables by violating the loop freedom invariant in a network of at least four nodes with a dynamic topology. Consider the network topology shown in Fig. 4.1a. The following scenario explains steps that lead to the invariant violation.

1. $node_2$ initiates a route discovery procedure for destination $node_3$ by broadcasting a $rreq$ message.

```
1 bool loop_freedom(des:int, cur:int, visited:Set<int>){
2   for(int i=0; i<n; i++)
3   if ((state.node(cur).nhops[des][i]!=-1) &&
4     (!visited.contains(state.node(cur).nhops[des][i]))&&
5       loop_freedom(des,i,visited.add(i)))
6     return true;
7   else
8     return false;
9 }
```

Figure 4.18: Checking the loop freedom property on a global state: we have used a dot notation to access the array $nhops$ of the rebec with the identifier i, i.e., $state.node(i)$, where $state$ is the newly generated global state

2. $node_1$ and $node_4$ upon receiving the $rreq$ message, add a route to their routing tables towards $node_2$ and store $node_2$ as their next hop. Since it is the first time that these nodes have received a message from $node_2$, the neighbor state of $node_2$ is set to *unconfirmed*. Therefore, the route state is *unconfirmed*.

3. As $node_1$ and $node_4$ are not the intended destination of the route request, they rebroadcast the $rreq$ message.

4. $node_1$ receives the $rreq$ message sent by $node_4$ and since the route to $node_2$ is *unconfirmed* it adds $node_4$ as a new next hop to $node_2$.

5. $node_4$ also adds $node_1$ as the new next hop towards $node_2$ after processing the $rreq$ sent by $node_1$. At this point a loop is formed between $node_1$ and $node_4$.

6. $node_3$ receives the $rreq$ message sent by $node_1$ and since it is the destination, it sends a $rrep$ message towards $node_1$.

7. $node_2$ moves out of the $node_1$ and $node_4$ communication ranges.

8. $node_1$ receives the $rrep$ message sent by $node_3$ and as the route state towards $node_2$ is *unconfirmed* it multicasts the $rrep$ message to the existing next hops, $node_2$ and $node_4$. Since $node_4$ is adjacent to $node_1$, it receives the message and then sends an ack to $node_1$. Therefore, $node_1$ sets the neighbor state of $node_4$ to *confirmed* and subsequently the route state towards $node_2$ to *valid*. Then it expunges the next hop $node_2$ from which it has not received an ack.

9. $node_4$ by receiving the $rrep$ message from $node_1$ multicasts it to its next hops towards $node_1$ and $node_2$, and similar for $node_1$. It updates its routing table by validating $node_1$ as its next hop to $node_2$.

The scenario was found for the first time in the wRebeca model with the network constraint resulting in four topologies as indicated in Table 4.5. During the state-space generation of this model whose state-space generator was extended with the function $loop\_freedom$, it stopped immediately when such a scenario was found. We remark that this scenario can be also found by other models in the table with different number of topologies if their state-space generators are extended accordingly. However, the chance of finding a loop scenario is larger in networks with more nodes. Furthermore, we can generalize the scenario to all networks with the same connectivity when the communications occur, and the same mobility scenario. This scenario proves that this version of AODV is incorrect, reported by us to the AODV group. We have also modeled and verified the next two versions, AODVv2-13 and AODVv2-16, and they still violate the loop-freedom property as reported in [154].

We believe that other loop scenarios of different classes can be found if the state-space generator continues when the given invariant is violated.

Checking the properties by mCRL2 Sequence numbers are used frequently by the AODV protocol to evaluate the freshness of routes. Therefore, it is important that each node's sequence number increases monotonically. To this end, we manually configured the state-space generator to add two self-loops to each state, one with the label $src\_sn(x)$ and the other with the label $info\_i\_dsn(y, z)$. The former monitors the sequence number of the source node, where x is sn of the source node. The latter traces the destination sequence number of routes to the source and destination for each node i (i.e., the backward and forward routes to the destination of our model), where y and z are $dsn[src]$ and $dsn[dst]$ of node i, respectively. Properties, specifying the monotonic increase of sequence numbers, are expressed through the ACTL-X fragment of μ-calculus as shown in Fig. 4.19. The first formula asserts a monotonic increase of the source sequence number. The second formula assures the destination sequence numbers stored in the routing table of $node_i$ are increased monotonically, and must hold for all nodes in the model.

4.7 Related Work

There are several studies that verify the loop-freedom property of AODV and its variants. A scenario leading to a loop was first discovered in [25].

The algebraic framework AWN [64] has been used in [54, 96, 144, 148] to model and analyze different versions of AODV. In [54], dynamic MANET on-demand (DYMO) routing protocol (also known as AODVv2) is modeled, it is shown that how it solves some problems discovered in AODV and how it fails to

$$\forall x, y : \mathbb{N}((x > 0 \wedge x < 4 \wedge y > 0 \wedge y < x) \Rightarrow [src\_sn(x).true^*.src\_sn(y)]false)$$

$$\begin{aligned}\forall x, y, m, n : \mathbb{N}((&x \geq 0 \wedge x < 4 \wedge y \geq 0 \wedge y < 4 \wedge \\ &m \geq 0 \wedge m < 4 \wedge n \geq 0 \wedge n < 4 \wedge \\ &(m < x \vee n < y)) \Rightarrow \\ &[true^*.info\_i\_dsn(x, y).true^*.info\_i\_dsn(m, n)]false)\end{aligned}$$

Figure 4.19: μ-calculus properties verified by mCRL2

address all the shortcomings. In [96], some ambiguities in the AODV RFC are discussed to show which interpretations are loop free. In [144] it is shown that monotonically increasing sequence numbers, by themselves, do not guarantee loop freedom. In [30] a timed extension of AWN is used to show that AODV is not loop-free as data required for routing can expire.

The loop freedom of AODVv2-04 for an arbitrary number of nodes was examined in [120] through an inductive and compositional proof: It provides an inductive invariant and proves that it is held initially and also preserved by every action, either a protocol action or a change in the network, similar to the approach of [148]. They have reported two loop-formation scenarios due to inappropriate setting of timing constants and accepting any valid route when the current route is broken without any further evaluation.

There are number of works that evaluate the performance of AODV and its variants using UPPAAL and its statistical model checking (SMC) extension [42, 61, 95]. In [98] the looping property of AODVv2-16, a recent version of AODV, is investigated and furthermore, its performance is compared to DYMO using UPPAAL SMC model checker [46].

4.8 Conclusion

We conclude this chapter by comparing the actor-based framework with the related process calculi as discussed in Section 3.7. To this aim we extend Table 3.5 with the row of Table 4.6. wRebeca includes all advantages of AWN by providing an asynchronous framework, while queues which manage messages are handled by the underlying computation model. Furthermore, it subsumes advantages of *RRBPT* by providing an automated tool which generates CLTSs, and hence reduces state spaces. The threat of state space explosion is more reduced in comparison with *RRBPT* as it generates CLTSs for coarse-grained execution of wRebeca models. Due to inheriting input-enabled nodes and reliable communication in this framework, properties like packet delivery of protocols can be easily inspected, as opposed to in algebraic frameworks.

We remark that bRebeca was introduced in [155] as an extension to Rebeca, to support broadcast communication which abstracts the *global broadcast com-*

Table 4.6: Characteristic of wRebeca

	Node specification	Comm.	Conn. of nodes	Mobility	Neighbor discovery
wRebeca	$\langle c, e_0^i \rangle$, $c = \langle V_c, M_c \rangle$	reliable	–	explicit/ implicit	-

munications [23]. To abstract away from the effect of the network, the order of receipts for two subsequent broadcast communications is not necessarily the same as their corresponding sends in an actor model. Hence, each actor mailbox was modeled by a bag. The resulting framework is suitable for modeling and efficient verification of broadcasting protocols above the network layer, but not appropriate for modeling MANETs in two respects. Firstly the topology is not defined, and every actor (node) can receive all messages, in other words all nodes are connected to each other. Secondly, as there is no topology defined, mobility cannot be expressed.

The reliable asynchronous communications, and implicit support of message storages make our framework suitable to analyze MANETs with respect to different mobility scenarios. To overcome the state space explosion, we leveraged the counter abstraction technique to analyze ad hoc networks with static topologies. Our reduction technique performs well on protocols with no specific state variable that distinguish each rebec, and topologies with many topologically equivalent nodes. We demonstrated the effectiveness of our approach on the flooding protocol in different network settings. However, mobility ruins the soundness of our counter abstraction. To this end, we eliminated τ-transitions while topology information was removed from global states to considerably reduce the size of the state space. The proposed reduction techniques were integrated into a tool customizable in verifying wRebeca models for different topology dynamism. Invariants can be checked during the state space generation while the resulting output can be fed into existing model checking tools such as mCRL2 and CADP.

Our resulting framework was used successfully to verify real-world and complex protocols such as AODVv2-11. The coarse-grained semantics together with topology abstraction make the model checking technique feasible with regard to such applications. For instance, for a network of four nodes, the state space of AODVv2-11 which implicitly models arbitrary changes of the underlying topology consists of $4,053$ states and $10,629$ transitions (see Table 4.4). The erroneous scenario violating loop freedom, which depends on topology changes, is very hard to find with informal approaches such as simulation or existing formal approaches due to the exhaustive number of topology changes and the state space explosion problem. The wRebeca formal framework with a Java-like syntax and an actor-based computational model (which simplifies the specifications) can be adopted by protocol designers to formally verify their protocols in the early phases of their development.

Model Checking MANETs

5

Properties of MANETs tend to be weaker than of wired networks. This is due to the topology-dependent behavior of communication, and consequently the need for multi-hop communication between nodes. For instance, the important property of *packet delivery* in routing or information dissemination protocols in the context of MANETs becomes: if there exists an end-to-end route between two nodes A and B *for a sufficiently long period of time*, then packets sent by A will eventually be received by B [64]. To verify such properties using an existing temporal logic such as CTL [34], a preprocessing step is required to enrich states with propositions that indicate restrictions over the underlying topology. The large number of topologies and mobility scenarios leads to an exponential blow-up if they are considered explicitly and individually. In [63] this bottleneck was circumvented by considering only specific mobility scenarios.

To remove efforts prior to verification, improve memory usage, gain efficiency, and extend verification to arbitrary mobility changes, we introduce the temporal logic *Constrained Action Computation Tree Logic* (CACTL), which is interpreted over CLTSs. It extends ACTLW [114] (which in turn is based on Action CTL [48]) with topological constraints. In CACTL, the path quantifier operator *All* is parameterized by a constraint on the underlying topologies, so that only paths are considered for which the constraint is satisfied, for example that a multi-hop connection is present. This extension can in principle be employed with regard to any temporal logic.

The new dimension of topology restrictions in the logic requires that the model checking algorithm is modified appropriately. We present a model checking algorithm for CACTL. This approach is flexible and efficient for the specification and verification of mobile behavior, compared to existing approaches. As an example we show how properties of AODV and the leader election protocol can be verified.

The chapter is structured as follows. Section 5.1 explains how network constraints are used to restrict the paths of a semantic mode. Section 5.2 discusses our motivation first and then introduces the syntax and semantics of CACTL. Section 5.3 provides a model checking algorithm and discusses its implementation. Section 5.4 illustrates the expressiveness of CACTL in the analysis of MANET

protocols. Section 5.5 provides an overview on existing approaches to model checking MANETs regarding mobility. Finally, Section 5.6 concludes the chapter.

5.1 Restricting Semantics with Network Constraints

As explained in Section 2.2.1, states in a CLTS do not hold information about the underlying topology. E.g., the transition $t_0 \xrightarrow{(\{A \rightsquigarrow B,\ A \not\rightsquigarrow C\}, \eta_1)} t_1$ implies that at the very moment when t_1 is reached, B is connected to A, and C is not connected to A. In t_0 before this transition and in t_1 after this transition, there is no restriction on the underlying topology. The arbitrary mobility implicitly modeled by the semantics can be restricted through a network constraint $\zeta \in \mathbb{C}(Loc)$. As a result, the CLTS is restricted to transitions that conform to ζ, meaning that their network constraints do not include (dis)connectivity information that contradicts ζ. Conformance between network constraints $\mathcal{C}$ and ζ means that $\neg\mathcal{C} \cap \zeta = \emptyset$.

In this chapter we only consider positive multi-hop network constraints as we are interested in behaviors that depend on the existence of such connections rather than their non-existence. Hence, a multi-hop network constraint is a set of multi-hop connectivity pairs like $\ell \dashrightarrow \ell'$ indicating topologies in which there is a multi-hop connection from ℓ to ℓ' (See Section 3.5.2). By a multi-hop constraint $\mathcal{M}$, the arbitrary mobility can be restricted in the semantics in two ways. First, topology changes in states are restricted to those defined by $\mathcal{M}$. The behavioral model then only allows transitions $\xrightarrow{\mathcal{C},\eta}$ for which there exists a topology $\gamma \in \Gamma(\mathcal{C}) \cap \Gamma(\mathcal{M})$. Thus, starting from a topology in $\mathcal{M}$ in the initial state, each state in any transition sequence has associated with it one or more topologies from $\mathcal{M}$. Second, the links making a topology satisfy $\mathcal{M}$ are restricted to be the same in all associated topologies. A path $t_0(\mathcal{C}_1, \eta_1)t_1(\mathcal{C}_2, \eta_2)t_2 \ldots$ is said to be valid for a multi-hop constraint $\mathcal{M}$ if there exists a sequence $\gamma, \gamma_1, \gamma_2, \ldots \in \Gamma(\mathcal{M})$ such that $\gamma_i \in \Gamma(\mathcal{C}_i)$ and γ is a subgraph of γ_i for all $i = 1, 2, \ldots$. $\mathcal{C}^+_\Gamma(\gamma) \cap \neg\mathcal{C}_i = \emptyset$ means the links of γ, extracted by $\mathcal{C}^+_\Gamma$, are not required to be disconnected in topologies satisfying $\mathcal{C}_i$, and consequently there exists a $\gamma_i \in \mathcal{C}_i$ such that γ is a subgraph of γ_i. For instance, with the first approach the path $t_0 \xrightarrow{(\{A \rightsquigarrow B,\ A \not\rightsquigarrow C\}, \eta_1)} t_1 \xrightarrow{(\{B \not\rightsquigarrow A,\ B \not\rightsquigarrow C\}, \eta_2)} t_2$ is allowed for $\{A \dashrightarrow C\}$, since both $\{A \rightsquigarrow B,\ A \not\rightsquigarrow C\}$ and $\{B \not\rightsquigarrow A,\ B \not\rightsquigarrow C\}$ have a topology in common with $\{A \dashrightarrow C\}$. But it is not allowed with the second approach, since the constraint of the first transition forbids a direct connection from C to A, while the constraint of the second transition forbids an indirect connection from C to A via B. The second interpretation of topology restrictions leads to more realistic verifications: a temporal property can be examined under the assumption that a stable multi-hop connection exists.

5.2 Constrained Action Computation Tree Logic (CACTL)

As the behavior of a MANET depends on the topology of the underlying network, the properties of a MANET protocol may depend on constraints on this topology. In verifying whether a temporal logic formula holds for a certain MANET, we may only want to explore paths that satisfy certain connectivity conditions with regard to the underlying topology, such as a direct link between two nodes, or the existence of a multi-hop connection between two nodes. CLTSs provide a suitable platform to verify topology-dependent properties, using the (dis)connectivity information encoded in transition labels. Transitions are traversed to investigate a behavioral property as long as (dis)connectivity information implies topology requirements on which the property depends. To this aim single- and multi-hop constraints are used, taking mobility into account in different ways. Single-hop constraints limit the one-step movements of nodes and hence, restrict transitions that can be traversed, while multi-hop constraints limit the mobility scenarios of nodes and thus, restrict the paths that need to be investigated (see Section 5.1). Since a CLTS is unfolded to an LTS in which mobility is modeled explicitly by pairing a state with its underlying topology (see Section 3.1.2), a behavioral property can be decorated with connectivity conditions while the topology is captured by atomic propositions on states (cf. [64]).

Action Computational Tree Logic (ACTL) [48] parameterizes the temporal operators *next* **X** and *until* **U** from CTL [34] with a set of actions. It provides a general framework for verifying properties in process algebra [47]. ACTLW [114] further enriches the *until* operator from ACTL and adds an (enriched) *unless* operator **W**. We note that in the action-based logic settings, in contrast to CTL, **EW** cannot readily be defined in terms of **AU**. In Section 5.2.2 we will define a temporal logic for MANETs in which the *until* and *unless* operators from ACTLW are preceded by either the path quantifier *Exists* **E** or *All* **A** from ACTLW decorated with multi-hop constraints.

5.2.1 Motivating Example

Consider the CLTS of $\partial_{Msg}([\![P]\!]_A \parallel [\![Q]\!]_B)$, where $Q \stackrel{def}{=} rcv(req).snd(rep).Q$ and $P \stackrel{def}{=} init.snd(req).rcv(rep).succ.P$, in Fig. 3.3. The internal action $init$ denotes a request from an application at node A that initializes a route discovery process to node B, and $succ$ denotes the successful termination of that route discovery process. To verify correctness of the route discovery protocol, taking into account mobility of nodes, any route discovery initialization must terminate successfully whenever there is a multi-hop connection from A and B as well as from B to A. This property can be specified by: *for all mobility scenarios in which* $A \dashrightarrow B$ *and* $B \dashrightarrow A$ *are valid for a sufficiently long period of time, each occurrence of* $init$ *is eventually followed by an occurrence of* $succ$. This property is satisfied by the CLTS in Fig. 3.3. In particular, consider the following two paths, where s_3 is a

deadlock:

$$s_0(\{\}, init)s_1(\{A \not\rightsquigarrow B\}, nsnd(req, A))s_3$$

$$s_0(\{\}, init)s_1(\{A \rightsquigarrow B\}, nsnd(req, A))s_2(\{B \not\rightsquigarrow A\}, nsnd(rep, B))s_3.$$

Although in both paths $init$ is not followed by an occurrence of $succ$, these paths do not violate the property above. Namely, to satisfy $A \dashrightarrow B$ and $B \dashrightarrow A$ in a network of two nodes, A and B should be directly connected to each other. However, these links get disconnected by the network constraint $\{A \not\rightsquigarrow B\}$ in the first and $\{B \not\rightsquigarrow A\}$ in the second path. Therefore, these paths are not valid for the multi-hop constraint $\{A \dashrightarrow B, B \dashrightarrow A\}$.

Suppose a regulator node $[\![R]\!]_C$ with

$$R \stackrel{def}{=} rcv(req).snd(req).R + rcv(rep).snd(rep).R$$

is added. The resulting network $\partial_{Msg}([\![P]\!]_A \parallel [\![Q]\!]_B \parallel [\![R]\!]_C)$ still satisfies the above property. However, if R is replaced by the erroneous protocol

$$E \stackrel{def}{=} rcv(req).snd(req).E + rcv(req).E + rcv(rep).snd(rep).E + rcv(rep).E$$

that may drop a packet it receives, then the new MANET does not satisfy the property. For example, the path which ends in the deadlock state $\partial_{Msg}([\![rcv(rep).P]\!]_A \parallel [\![Q]\!]_B \parallel [\![E]\!]_C$

$$\begin{aligned}&\partial_{Msg}([\![P]\!]_A \parallel [\![Q]\!]_B \parallel [\![E]\!]_C)\\ &\quad(\{\}, init)\partial_{Msg}([\![snd(req).rcv(rep).P]\!]_A \parallel [\![Q]\!]_B \parallel [\![E]\!]_C)\\ &\quad(\{A \rightsquigarrow C,\ A \not\rightsquigarrow B\}, nsnd(req, A))\partial_{Msg}([\![rcv(rep).P]\!]_A \parallel [\![Q]\!]_B \parallel [\![E]\!]_C)\end{aligned}$$

satisfies the multi-hop constraint $\{A \dashrightarrow B,\ B \dashrightarrow A\}$ in a network of three nodes, namely A, B and C, but the occurrence of $init$ is not followed by an occurrence of $succ$. For instance, with regard to the topology $\gamma = \{A \mapsto \{C\}, B \mapsto \{C\}, C \mapsto \{A, B\}\}$ in $\{A \dashrightarrow B,\ B \dashrightarrow A\}$ (note that the network constraints $\{\}$ and $\{A \rightsquigarrow C,\ A \not\rightsquigarrow B\}$ agree with γ), C may drop the req packet.

5.2.2 CACTL Syntax

The temporal logic *Constrained Action Computation Tree Logic* (CACTL) allows to express topology-dependent properties of MANET protocols. The path quantifier *All* is parameterized by a multi-hop constraint over the topology, which specifies the pre-condition required for paths of a state to be inspected (for its evaluation). A typical example of such a constraint is the existence of a (topological) path between two nodes. A pre-condition restricts mobility scenarios, implicitly modeled in the semantics. Therefore, the evaluation of a state formula is restricted to those mobility scenarios satisfying the pre-condition. Such an evaluation only considers the paths of a states that are valid for the given mobility

scenarios. However, the path quantifier *Exists* is defined as before and inspects all paths of a state to find at least one satisfying the specified (path) property. By combining *All* and *Exists* path quantifiers, the decorated version of *Exists*, preconditioned by a multi-hop constraint, can be specified to also include states that have no path for which the pre-condition holds. In other words, if a state has no path for which the pre-condition holds, then it satisfies the formula vacuously.

Moreover, to restrict mobility scenarios in a more concrete way, the satisfaction relation is parameterized with single-hop constraints. This parameter expresses the (non-)existence of communication links; nodes can only move in such a way that the specified links do not change. This is achieved by only traversing transitions that conform to the specified links.

A CACTL formula may include constants true and false, actions $\eta \in \mathit{Act}_\tau$, standard Boolean operators $\neg$, $\wedge$ and $\vee$, path quantifiers $\mathbf{A}^{\,\mu}$ parameterized by constraints over topologies and $\mathbf{E}$, and temporal operators $\mathbf{U}$ and $\mathbf{W}$.

Let $\eta \in \mathit{Act}_\tau$, and $\ell, \ell' \in \mathit{Loc}$. *Action formula* χ, *topology formula* μ, *state formula* ϕ (also called *CACTL formula*), and *path formula* ψ are defined by the grammars:

$$
\begin{array}{lcl}
\chi & ::= & \mathit{true} \mid \eta \mid \neg\chi \mid \chi \wedge \chi' \\
\mu & ::= & \mathit{true} \mid \ell \dashrightarrow \ell' \mid \mu \wedge \mu' \\
\phi & ::= & \mathit{true} \mid \neg\phi \mid \phi \wedge \phi' \mid \mathbf{E}\psi \mid \mathbf{A}^{\,\mu}\psi \\
\psi & ::= & \phi \,{}_{\chi}\mathbf{U}_{\chi'}\, \phi' \mid \phi \,{}_{\chi}\mathbf{W}_{\chi'}\, \phi'
\end{array}
$$

Each temporal operator should be preceded by a path quantifier, forming a *CACTL operator*. Action and path formulae are the same as in ACTLW [114]. Using topology formulae to restrict state formulae and require multi-hop connections are novel. Each topology formula induces the multi-hop constraint made of multi-hop connections participating in the topology formula under consideration. In the reminder of this chapter, we use the notions of topology formula and multi-hop constraint interchangeably. For the topology formula μ and topology γ, assume $\gamma \in \mu$ denotes $\gamma \in \Gamma(\mathcal{M})$, where $\mathcal{M}$ is the multi-hop constraint induced by μ.

Let $T \equiv \langle S, \Lambda, \rightarrow, s_0 \rangle$ be a CLTS. A *path* π in T from a state $t_0 \in S$ is an alternating sequence of states and constraint-action (transition) pairs $t_0(\mathcal{C}_1, \eta_1)t_1(\mathcal{C}_2, \eta_2)t_2 \ldots$ where $\forall i \geq 1\,((t_{i-1}, (\mathcal{C}_i, \eta_i), t_i) \in \rightarrow)$. A path is said to be *maximal* if it either is infinite or ends in a deadlock state, meaning that it has no outgoing transitions. Given a network constraint $\zeta \in \mathbb{C}^v(\mathit{Loc})$, a path is called a *$\zeta$-path* if $\mathcal{C}_i$ conforms to ζ, i.e., $\forall i \geq 1(\neg\mathcal{C}_i \cap \zeta = \emptyset)$. State and path formulae are interpreted with regard to a network constraint ζ, meaning that only maximal ζ-paths are considered in the evaluation of such formulae. A network constraint $\mathcal{C}' \in \mathbb{C}(\mathit{Loc})$ is said to be invalid for (or violate) μ, if $\forall\gamma \in \mu(\mathcal{C}^+_\Gamma(\gamma) \cap \neg\mathcal{C}' \neq \emptyset)$. We say a path violates μ in state t_i, if the accumulated network constraints before reaching t_i violates μ, i.e., $\forall\gamma \in \mu(\mathcal{C}^+_\Gamma(\gamma) \cap \neg\bigcup_{1\leq j\leq i}\mathcal{C}_j \neq \emptyset)$. A path is said to be invalid for μ (or violate μ), if it is invalid in some state over the path.

The path quantifier $\mathbf{E}$ requires that the given path formula is satisfied for at least one maximal ζ-path starting in the given state, while $\mathbf{A}^{\,\mu}$ requires the property holds for all ζ-paths that are valid for the multi-hop constraint induced by the topology formula μ.

A state t that satisfies a CACTL formula ϕ is called a ϕ-state, and a transition with an action from χ is called a χ-transition. A χ-transition that ends in a ϕ-state is called a (χ,ϕ)-transition. A path with an initial ϕ-state and only (χ,ϕ)-transitions is called a (χ,ϕ)-path. The until operator $\phi\,{}_{\chi}\mathbf{U}_{\chi'}\,\phi'$ is satisfied by maximal ζ-paths that start with an initial ϕ-state and perform a finite sequence of (χ,ϕ)-transitions, until a (χ',ϕ')-transition is performed. The unless (or weak until) operator $\phi\,{}_{\chi}\mathbf{W}_{\chi'}\,\phi'$ extends $\phi\,{}_{\chi}\mathbf{U}_{\chi'}\,\phi'$ by moreover allowing infinite ζ-paths that are a (χ,ϕ)-path. We note that, in contrast to CTL, $\mathbf{EW}$ cannot readily be defined in terms of $\mathbf{AU}$, because the states satisfying ϕ and ϕ' must be visited by transitions carrying actions from χ and χ' respectively.

For example, the correctness of the route discovery protocol explained in Section 5.2.1 can be verified by the state formula

$$\mathbf{A}^{\,true}(true\,{}_{\neg init}\mathbf{W}_{init}\,\mathbf{A}^{\,A\dashrightarrow B\wedge B\dashrightarrow A}(true\,{}_{\tau}\mathbf{U}_{succ}\,true))$$

which indicates that an action $init$ is always eventually followed by an action $succ$, possibly after some communications (specified by τ), under the condition that from the moment that $init$ occurs there exists a multi-hop connection from A to B and from B to A. This formula is satisfied by the initial state of the CLTS given in Fig. 3.3 after all communications have been turned into τ with the help of abstraction operator, i.e., by $\tau_{Msg}(\partial_{Msg}(\llbracket P\rrbracket_A \parallel \llbracket Q\rrbracket_B))$.

Other CACTL formulae can be derived from $\mathbf{E}$, $\mathbf{A}$, $\mathbf{U}$ and $\mathbf{W}$ following the approach of [114]:

$$\begin{aligned}
\mathbf{EX}_{\chi}\phi &= \mathbf{E}(true\,{}_{false}\mathbf{U}_{\chi}\,\phi) & \mathbf{AX}_{\chi}\phi &= \mathbf{A}^{\,true}(true\,{}_{false}\mathbf{U}_{\chi}\,\phi)\\
\mathbf{EF}_{\chi}\phi &= \mathbf{E}(true\,{}_{true}\mathbf{U}_{\chi}\,\phi) & \mathbf{AF}^{\mu}_{\chi}\phi &= \mathbf{A}^{\,\mu}(true\,{}_{true}\mathbf{U}_{\chi}\,\phi)\\
\mathbf{EG}_{\chi}\phi &= \mathbf{E}(\phi\,{}_{\chi}\mathbf{W}_{false}\,false) & \mathbf{AG}^{\mu}_{\chi}\phi &= \mathbf{A}^{\,\mu}true(\phi\,{}_{\chi}\mathbf{W}_{false}\,false)
\end{aligned}$$

Intuitively, $\mathbf{AX}_{\chi}\phi$ is satisfied by states of which all maximal ζ-paths start with a (χ,ϕ)-transition, $\mathbf{AF}^{\mu}_{\chi}\phi$ by states of which all maximal ζ-paths either contain a (χ,ϕ)-transition or in some state violate μ, and $\mathbf{AG}^{\mu}_{\chi}\phi$ by states of which all maximal ζ-paths visit only ϕ-states and perform only χ-transitions as long as μ is valid.

We define $\mathbf{E}^{\mu}$ by combining $\mathbf{E}$ and $\mathbf{A}^{\mu}$ as $\mathbf{E}^{\mu}\varphi = \mathbf{E}\varphi \vee \mathbf{A}^{\,\mu}\varphi$. Intuitively, $\mathbf{E}^{\mu}\varphi$ is satisfied by a state that either has at least one maximal ζ-path that satisfies the path formula φ, or all maximal ζ-paths violate μ. Consequently, $\mathbf{EF}^{\mu}_{\chi}\phi$ is satisfied by a state that either has at least one maximal ζ-path which contains a (χ,ϕ)-transition or all its maximal ζ-paths are invalid for μ.

5.2.3 CACTL Semantics

Let $T \equiv \langle S, \Lambda, \rightarrow, s_0 \rangle$ be a CLTS. For a path π of T, assume π_i^s, $\pi_i^{\mathcal{C}}$ and π_i^{η} denote the ith state, network constraint and action on path π, while $len(\pi)$ denotes the number of transitions in π (which is ∞ if π is infinite). So a finite path π is of the form $\pi_0^s(\pi_1^{\mathcal{C}}, \pi_1^{\eta})\pi_1^s \cdots (\pi_{len(\pi)}^{\mathcal{C}}, \pi_{len(\pi)}^{\eta})\pi_{len(\pi)}^s$. Recall that $\mathcal{C}_\Gamma^+(\gamma)$ extract all connectivity relations in topology γ.

Satisfaction by $\eta \in Act_\tau$ of action formula χ (written $\eta \models \chi$), by state t of state formula ϕ under network constraint ζ (written $t \models_\zeta \phi$), and by maximal path π of path formula ψ under network constraint ζ (written $\pi \models_\zeta \psi$), is defined inductively below.

$\eta \models true$	always
$\eta \models \eta'$	iff $\eta = \eta'$
$\eta \models \neg\chi$	iff $\eta \not\models \chi$
$\eta \models \chi \wedge \chi'$	iff $\eta \models \chi \wedge \eta \models \chi'$
$t \models_\zeta true$	always
$t \models_\zeta \neg\phi$	iff $t \not\models_\zeta \phi$
$t \models_\zeta \phi \wedge \phi'$	iff $t \models_\zeta \phi \wedge t \models_\zeta \phi'$
$t \models_\zeta \mathbf{E}\psi$	iff there exists a maximal ζ-path π such that $t = \pi_0^s \wedge \pi \models_\zeta \psi$
$t \models_\zeta \mathbf{A}^{\,\mu}\psi$	iff for all maximal ζ-paths π with $t = \pi_0^s$ either $\pi \models_\zeta \psi$,
	or assume $\psi \equiv \phi\,{}_\chi\mathbf{V}_{\chi'}\,\phi'$, where $\mathbf{V} \in \{\mathbf{U}, \mathbf{W}\}$, $\pi_0^s \models_\zeta \phi \wedge$
	$\exists 1 \leq j \leq len(\pi)\,(\forall\, 1 \leq i \leq j\,(\pi_i^s \models_\zeta \phi \wedge \pi_i^\eta \models \chi) \wedge$
	$\quad \forall\gamma \in \mu\,(\mathcal{C}_\Gamma^+(\gamma) \cap \bigcup_{k=1}^{j} \neg\pi_k^{\mathcal{C}} \neq \emptyset))$
$\pi \models_\zeta \phi\,{}_\chi\mathbf{U}_{\chi'}\,\phi'$	iff $\pi_0^s \models_\zeta \phi$ and $\exists 1 \leq j \leq len(\pi)\,($
	$\forall\, 1 \leq i < j\,(\pi_i^s \models_\zeta \phi \wedge \pi_i^\eta \models \chi) \wedge (\pi_j^s \models_\zeta \phi' \wedge \pi_j^\eta \models \chi'))$
$\pi \models_\zeta \phi\,{}_\chi\mathbf{W}_{\chi'}\,\phi'$	iff either $\pi \models_\zeta \phi\,{}_\chi\mathbf{U}_{\chi'}\,\phi'$,
	or $\pi_0^s \models_\zeta \phi$ and $\forall 1 \leq i \leq len(\pi)\,(\pi_i^s \models_\zeta \phi \wedge \pi_i^\eta \models \chi)$

The semantics of the until operator in ACTL, ACTLW and CACTL is somewhat different from CTL: $\mathbf{E}(\phi_\chi\mathbf{U}_{\chi'}\phi')$ in ACTLW requires to perform at least one action from χ' to reach a state satisfying ϕ', while $\mathbf{E}(\phi\mathbf{U}\phi')$ in CTL is also satisfied if the first state already satisfies ϕ'. Similar to ACTL but in contrast to ACTLW, our semantics of the until operator explicitly distinguishes silent from visible actions.

5.3 Model Checking Algorithms

We adapt the CTL model checking algorithm (see [36, 53]) to CACTL. Model checking a CACTL formula φ on a CLTS under $\zeta \in \mathbb{C}$ starts with the smallest sub-

formulae and works outwards toward φ. The procedure $ModelCheck(CLTS, \varphi, \zeta)$ returns those states in $CLTS$ that satisfy φ under ζ. The pseudo code of the model checking algorithm's backbone, where each syntactic form of CACTL's grammar is provided with a dedicated subcall, is given in Fig. 5.1. In the following subsections, we explain the details of each subcall.

Procedure $ModelCheck(CLTS, \varphi, \zeta)$
Output: The states in $CLTS$ that satisfy φ under ζ
switch φ **do**
  case $\phi \wedge \phi'$
    $T_1 := ModelCheck(CLTS, \phi, \zeta)$;
    $T_2 := ModelCheck(CLTS, \phi', \zeta)$;
    return $T_1 \cap T_2$;
  case $\mathbf{E}(\phi \,{}_{\chi}\mathbf{U}_{\chi'}\, \phi')$
    return $CheckEU(CLTS, \chi, \phi, \chi', \phi', \zeta)$
  case $\mathbf{A}^{\mu}(\phi \,{}_{\chi}\mathbf{U}_{\chi'}\, \phi')$
    return $CheckAU(CLTS, \mu, \chi, \phi, \chi', \phi', \zeta)$
  ...
endsw

Figure 5.1: Backbone of the model checking algorithm.

5.3.1 Model Checking EU Formulae

We explain the idea of the subcall $CheckEU$ for the **EU** operator. A ϕ-state satisfies $\mathbf{E}(\phi \,{}_{\chi}\mathbf{U}_{\chi'}\, \phi')$ under ζ if it has a ζ-path that consists of (χ, ϕ)-transitions until a (χ', ϕ')-transition is performed. Our model checking algorithm is based on the CTL model checking algorithm for **EU** [53], where the analysis starts from ϕ'-states and proceeds in a backward fashion over ϕ-states.

First, recursively, the states are determined that satisfy ϕ as well as those that satisfy ϕ'. Then, ϕ-states with an outgoing (χ', ϕ')-transition which conforms to ζ are searched for. Later, we move backward over (χ, ϕ)-transitions which conform to ζ in a breadth-first fashion. Backward movement is stopped when no new ϕ-state can be explored. All ϕ-states visited in the backward analysis satisfy $\mathbf{E}(\phi \,{}_{\chi}\mathbf{U}_{\chi'}\, \phi')$ under ζ. The pseudocode of the algorithm is given in Fig. 5.2.

The FIFO queue $explore$ initially contains the states from which the backward analysis starts: ϕ-states with an outgoing (χ', ϕ')-transition that conforms to ζ. Backward movement over (χ, ϕ)-transitions that conform to ζ is assisted by initially finding such transitions, denoted by transition relation $\rightarrow'$. States visited during backward movement are stored in the set $visited$. The correctness of the algorithm is implied by the fact that $\forall s \in explore\,(s \models_{\zeta} \mathbf{E}(\phi \,{}_{\chi}\mathbf{U}_{\chi'}\, \phi'))$.

As an example we verify $\mathbf{E}(true \,{}_{a \vee \tau}\mathbf{U}_{b}\, \mathbf{E}(true \,{}_{a \vee \tau}\mathbf{U}_{c}\, true))$ under the empty constraint $\{\}$ over the CLTS given in Fig. 5.3. First the states satisfying $\mathbf{E}(true \,{}_{a \vee \tau}\mathbf{U}_{c}\, true)$

Procedure $CheckEU(CLTS, \chi, \phi, \chi', \phi', \zeta)$
Output: The states that satisfy $\mathbf{E}(\phi \, {}_{\chi}\mathbf{U}_{\chi'} \, \phi')$ under ζ in
$CLTS \equiv \langle S, \Lambda, \rightarrow, s_0 \rangle$
$T_\phi := ModelCheck(CLTS, \phi, \zeta)$;
$T_{\phi'} := ModelCheck(CLTS, \phi', \zeta)$;
$explore := \{s \in T_\phi \mid (s, (\mathcal{C}, \eta), t) \in \rightarrow,\ t \in T_{\phi'},\ \eta \models \chi',\ \zeta \cap \neg\mathcal{C} = \emptyset\}$;
$visited := \emptyset$;
$\rightarrow' := \{(t, (\mathcal{C}, \eta), t') \in \rightarrow \mid t, t' \in T_\phi,\ \eta \models \chi,\ \zeta \cap \neg\mathcal{C} = \emptyset\}$;
while $explore \neq \emptyset$ **do**
  $t_0 := dequeue(explore)$;
  $visited := visited \cup \{t_0\}$;
  for all $(s, (\mathcal{C}, \eta), t_0) \in \rightarrow'$ **do**
    if $s \notin explore \cup visited$ **then** $enqueue(s, explore)$;
return $visited$;

Figure 5.2: Model checking **EU** formulae.

are found. The breadth-first backward search starts from s_7, as this is the only state with an outgoing c-transition. The transitions from s_4 to s_6 and from s_7 to s_9 are removed because they do not carry an action from $\{a, \tau\}$. The verification of the inner **EU** formula ends with $visited = \{s_6, s_7\}$. To verify the outer **EU** formula, the breadth-first backward analysis starts from s_4, because from this state there is a b-transition to s_6 where the inner **EU** formula holds. Again the transitions from s_4 to s_6 and from s_7 to s_9 are removed. The search computes $visited = \{s_0, s_1, s_2, s_4, s_5\}$, which is returned as the result of the verification of the outer **EU** formula.

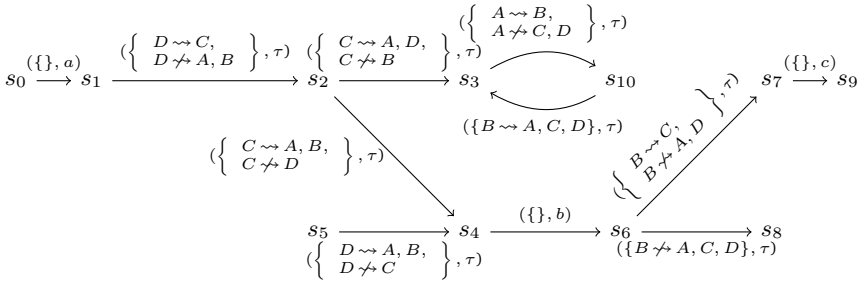


Figure 5.3: The CLTS to be checked for $\mathbf{E}(true \, {}_{a \vee \tau}\mathbf{U}_b \, \mathbf{E}(true \, {}_{a \vee \tau}\mathbf{U}_c \, true))$.

The worst-case time and memory complexity to check an **EU** formula is $O(V + E)$, where V and E are the number of states and transitions of the underlying CLTS. By contrast, model checking such a CTL formula using the standard algorithm with the approach of [64], where the topology is modeled as a part

of states, has a worst-case time and memory complexity of $O(4^{\frac{n^2-n}{2}} \times (V+E))$, where n is the number of nodes; 4 originates from the four types of links, and $\frac{n^2-n}{2}$ is the number of possible links among the n nodes. We note however that our model checking algorithm for **AU** formulae, which will be explained in the next section, faces a similar exponential blow-up.

A bottleneck for the performance can be the check whether $s \notin \mathit{explore} \cup \mathit{visited}$. In general the generated state space is so large that the bulk of it has to be stored on disk, and disk accesses to perform the aforementioned check are very expensive. In line with [91], this overhead can be alleviated considerably by exploiting a Bloom filter to reduce the number of redundant disk accesses in cases where the state s was not yet generated, i.e., is not present on disk.

5.3.2 Model Checking AU Formulae

The procedure $\mathit{CheckAU}$ model checks the **AU** operator. In general a ϕ-state satisfies a formula $\mathbf{A}^{\,\mu}(\phi\ {}_{\chi}\mathbf{U}_{\chi'}\ \phi')$ under ζ if all its ζ-paths consist of (χ,ϕ)-transitions until either a (χ',ϕ')-transition is performed or the topology formula μ is violated. Paths that violate the **AU** formula can be searched for with a backward analysis in a depth-first fashion. A state t may be visited for different values of network constraints, gathered over different paths. The backward analysis should be redone for each network constraint $\mathcal{C}$, except when t was visited earlier for a network constraint $\mathcal{C}'$ with $\mathcal{C}' \subseteq \mathcal{C}$.

We start the backward search from ϕ-states that violate the **AU** formula because they have either (1) an outgoing transition (which conforms to ζ) that is neither a (χ',ϕ')- nor a (χ,ϕ)-transition, or (2) no outgoing transition, or (3) an infinite (χ,ϕ)-path that does not violate μ (and conforms to ζ). All preceding states on backward (χ,ϕ)-paths that do not violate μ (with transitions that conform to ζ) violate $\mathbf{A}^{\,\mu}(\phi\ {}_{\chi}\mathbf{U}_{\chi'}\ \phi')$. Furthermore, $\neg\phi$-states trivially violate this formula.

For the sake of efficiently finding ϕ-states of type (3), we assume *strong fairness*, meaning that we only consider infinite paths on which each infinitely often enabled transition is infinitely often executed. Considering only (χ,ϕ)-transitions that conform to ζ, we determine *terminal* strongly connected components (SCCs) (i.e., SCCs from which no other SCC can be reached). Owing to strong fairness, each infinite (χ,ϕ)-path is guaranteed to end up in such a terminal SCC, and traverse each transition in this SCC. We check for each terminal SCC whether its accumulated network constraints do not violate μ.

The pseudocode of the algorithm is given in Fig. 5.4. The procedure Clean removes all transitions that do not conform to ζ, and turns states that can perform a transition (which conforms to ζ) that is neither a (χ',ϕ')- nor a (χ,ϕ)-transition into deadlock states, by removing all their outgoing transitions. This way ϕ-states of type (1) are cast to type (2).

The procedure TSCC computes the terminal SCCs of (χ,ϕ)-transitions, and

Procedure $CheckAU(CLTS, \mu, \chi, \phi, \chi', \phi', \zeta)$
Output: The states that satisfy $\mathbf{A}^{\mu}(\phi\, {}_{\chi}\mathbf{U}_{\chi'}\, \phi')$ under ζ in
$CLTS \equiv \langle S, \Lambda, \rightarrow, s_0 \rangle$
$T_\phi := ModelCheck(CLTS, \phi, \zeta)$;
$T_{\phi'} := ModelCheck(CLTS, \phi', \zeta)$;
$\rightarrow' := Clean(CLTS, \chi, T_\phi, \chi', T_{\phi'}, \zeta)$;
$constraint := TSCC(\rightarrow', \chi, T_\phi)$;
$start := Start(\rightarrow', \mu, \chi, T_\phi, \chi', T_{\phi'}, constraint)$;
$violate := start$;
$visited := \{t \mapsto \{\gamma \in \mu \mid \mathcal{C}^+_\Gamma(\gamma) \cap \neg constraint(t) = \emptyset\} \mid t \in start\} \cup \{(t \mapsto \emptyset) \mid t \notin start\}$;
while $start \neq \emptyset$ **do**
  choose a $t_0 \in start$;
  $start := start \setminus \{t_0\}$;
  $explore := \{(t_0, \{\gamma \in \mu \mid \mathcal{C}^+_\Gamma(\gamma) \cap \neg constraint(t) = \emptyset\})\}$;
  while $explore \neq \emptyset$ **do**
    $(t_1, \Gamma_1) := pop(explore)$;
    $violate := violate \cup \{t_1\}$;
    for all $(t_2, (\mathcal{C}_2, \eta), t_1) \in \rightarrow'$ **do**
      $\Gamma := \{\gamma \in \Gamma_1 \mid \mathcal{C}^+_\Gamma(\gamma) \cap \neg \mathcal{C}_2 = \emptyset\}$;
      if $\Gamma \not\subseteq visited(t_2) \wedge \Gamma \neq \emptyset$ **then**
        $push((t_2, \Gamma \setminus visited(t_2)), explore)$;
        $upd(visited, t_2, \Gamma)$;
return $T_\phi \setminus violate$;

Figure 5.4: Model checking **AU** formulae.

simultaneously collects the network constraints on the transitions in each such terminal SCC. It returns a function $constraint$ that maps each state in a terminal SCC of (χ, ϕ)-transitions to the accumulation of all network constraints on the transitions in this terminal SCC; states that are not in such a terminal SCC are mapped to the empty network constraint. Next the procedure $Start$ returns the set *start* of states that are in a terminal SCC of (χ, ϕ)-transitions in which (I) none of the states can perform a (χ', ϕ')-transition, and (II) the accumulation of all network constraints on the transitions do not violate μ.

The backward movement, which every time starts from a state in *start*, is performed in a depth-first fashion over (χ, ϕ)-transitions which conform to ζ. During this backward movement, network constraints on transitions are accumulated. The backward search proceeds as long as the accumulated network constraints do not violate μ. Visited states violate the **AU** formula; they are stored in the set *violate*, which initially consists of the states in *start*. We remark that the accumulated network constraint $\mathcal{C}_1 \cup \mathcal{C}_2$ does not violate μ, implied by

$\exists\gamma \in \mu\,(\mathcal{C}_\Gamma^+(\gamma) \cap \neg(\mathcal{C}_1 \cup \mathcal{C}_2) = \emptyset)$, if and only if $\exists\gamma \in \{\gamma' \in \mu \mid \mathcal{C}_\Gamma^+(\gamma') \cap \neg\mathcal{C}_1 = \emptyset\}\,(\mathcal{C}_\Gamma^+(\gamma) \cap \neg\mathcal{C}_2 = \emptyset)$. Hence, instead of accumulating network constraints over the paths and then checking whether they do not violate μ, the topologies that lead to the satisfaction of μ are initially computed and refined while passing a transition. Therefore, the function $visited$ maps each visited state to the set of topologies for which the backward analysis was performed; states that have not yet been visited are mapped to the empty set. Hence, the backward search proceeds as long as the set of topologies (which makes μ true along the path) is not empty. Furthermore, the backward analysis is redone for each visited state s with the set of topologies Γ if $\Gamma \not\subseteq visited(s)$.

Each backward search is coordinated by means of the stack *explore*, which initially consists of the state t_0 in $start$ where the search starts, paired with topologies that make μ true with respect to the accumulation $constraint(t_0)$ of network constraints in the terminal SCC of t_0. Repeatedly the top (t_1, Γ_1) of $explore$ is popped, and t_1 is added to $violate$. Furthermore, for each incoming transition $(t_2, (\mathcal{C}_2, \eta), t_1)$ of t_1, Γ_1 is refined into Γ with respect to $\mathcal{C}_2$ to denote the topologies for which the state t_2 has been visited. Then, it is checked whether state t_2 has been visited for a new topology. If this is the case, then the pair $(t_2, \Gamma \setminus visited(s))$ is added to the stack $explore$, where $\Gamma \setminus visited(s)$ is the set of newly visited topologies for which the backward analysis should be redone. Furthermore, $visited$ is updated by the new topologies for t_2, denoted by $upd(visited, t_2, \Gamma)$.

Finally, when the backward search has been performed exhaustively, meaning that *start* has become empty, all states in T_ϕ that are not in *violate* satisfy $\mathbf{A}^{\,\mu}(\phi\,{}_\chi\mathbf{U}_{\chi'}\,\phi')$ under ζ.

The correctness of our algorithm is established based on the following assertions, which are straightforward to prove:

1. The states on the $explore$ stack form a (χ, ϕ)-path in the semantic model that conforms to ζ;
2. $\forall(s, \Gamma) \in explore\,(\Gamma \neq \emptyset)$.
3. The sets of topologies on the stack constitute a chain, i.e., for any two consecutive elements (s_1, Γ_1) directly above (s_2, Γ_2) on the stack, $\Gamma_1 \subseteq \Gamma_2$ and s_1 is a descendant of s_2.

For each $s \in violate$, there was (s, Γ) on the top of the stack before s was being added to $violate$. The first assertion implies that there exists a (χ, ϕ)-path that conforms to ζ from s to the state at the bottom of the stack, which belongs to $start$. The second assertion implies that $\Gamma \neq \emptyset$. Therefore, the third assertion implies that the accumulated network constraint over this path does not violate μ. Hence, s indeed violates $\mathbf{A}^{\,\mu}(\phi\,{}_\chi\mathbf{U}_{\chi'}\,\phi')$ under ζ. Conversely, we can be easily shown that if a state violates $\mathbf{A}^{\,\mu}(\phi\,{}_\chi\mathbf{U}_{\chi'}\,\phi')$, then it will be in the set $violate$ as it has a path to a state in $start$.

Our model checking algorithm is inspired by the CTL model checking algorithm for $\mathbf{EG}\phi$ formulae [36], where the backward search starts from SCCs with ϕ-states, i.e., states satisfying $\mathbf{EG}\phi$. We remark that the states in our terminal SCCs violate $\mathbf{AU}$. Furthermore, the classical CTL model checker for $\mathbf{AU}$ formulae is based on a forward analysis in a depth-first fashion; a state satisfies $\mathbf{AU}$ if none of its successors violates $\mathbf{AU}$ [53]. However, fairness conditions cannot be addressed during the forward analysis. To this end, the framework is extended with predicates over states. Then states with a fair-path, i.e., a path over which each fairness predicate is infinitely many times satisfied, are labeled with the auxiliary predicate Q (through a backward analysis from fair-SCCs, i.e., SCCs with at least one state satisfying each fairness predicate). The $\mathbf{AU}$ formulae with fairness conditions are found using the $\mathbf{EU}$- and $\mathbf{EG}$-algorithms. The former considers Q-labels while the latter starts the backward search from fair-SCCs with ϕ-states.

Each state is in the worst case visited for all possible accumulated network constraints. If the set Loc contains n addresses, there are $\binom{n-2}{2}$ (dis)connectivity pairs, and consequently the number of possible accumulated network constraints is 2^{n^2-n}. Hence, each state is visited at most 2^{n^2-n} times. This means model checking of $\mathbf{AU}$ formulae is only feasible in case of a small number of addresses. However, since interesting mobility scenarios typically require only few addresses, this is not a serious limitation of our approach.

The time complexity of $\mathbf{AU}$ model checking is $O(2^{n^2-n} \times (V + E))$, with memory usage $O(V + E)$. Since the time and space complexities are linear with respect to the number of states and transitions, model checking MANET protocols with arbitrary mobility is feasible. We recall that model checking CTL formula over semantic models where topology is modeled as a part of states, following the approach of [64], has a time complexity of $O(2^{n^2-n} \times (V+E))$ with memory usage $O(2^{n^2-n} \times (V+E))$. Note that we have not considered the memory usage of $visited$ in favor of the space needed to store topologies as part of states in the latter approach.

As an example, we check the formula $\mathbf{A}^{D \dashrightarrow B}(true\ {}_{a\vee\tau}\mathbf{U}_b\mathbf{E}(true\ {}_{a\vee\tau}\mathbf{U}_c\ true))$ under $\{\}$ over the CLTS given in Fig. 5.3, using our model checking algorithm. First transitions from s_4 to s_6 and from s_7 to s_9 are removed because they do not carry an action from $\{a, \tau\}$. The terminal (χ, ϕ)-SCCs of which the accumulated network constraints do not violate $D \dashrightarrow B$, found by procedure $FindSCC$, are $\{s_3, s_{10}\}$, $\{s_8\}$ and $\{s_9\}$. Therefore, they are collapsed as shown in Fig. 5.5, while $visited(SCC_0) = \{\{A \rightsquigarrow B,\ A \not\rightsquigarrow C, D,\ B \rightsquigarrow A, C, D\}\}$, $visited(SCC_1) = \{\{\}\}$, and $visited(SCC_2) = \{\{\}\}$. Furthermore, the relation SCC is set to $\{(SCC_0, \{s_3, s_{10}\}), (SCC_1, \{\}), (SCC_2, \{\})\}$. The set end contains $\{s_7, SCC_0, SCC_1, SCC_2\}$, since s_7 contains a transition which is neither from $\{a, \tau\}$ nor from $\{b\}$. The backward analysis starts from SCC_0 for $head(visited(SCC_0))$. Upon movement to s_2, the accumulated network constraints are $\{A \rightsquigarrow B,\ A \not\rightsquigarrow C, D,\ B \rightsquigarrow A, C, D,\ C \rightsquigarrow A, D,\ C \not\rightsquigarrow B\}$ are valid for $D \dashrightarrow B$, and consequently it is added to $visited(s_2)$. The backward analysis proceeds to s_0, while $temp = \{SCC_0, s_2, s_1, s_0\}$. The

backward analysis from states SCC_1 and s_7 stops at state s_6, while $visited(s_6) = \{\{B \rightsquigarrow C,\ B \not\rightsquigarrow A, D\}, \{B \not\rightsquigarrow A, C, D\}\}$ and states SCC_1, s_7, and s_6 are added to $temp$. The backward analysis from SCC_2 stops immediately, while it is added to $temp$. Consequently states $\{s_4, s_5\}$ are returned by the algorithm. Note that the path $s_0(\{\}, a)s_1(\{D \not\rightsquigarrow A, B,\ D \rightsquigarrow C\}, \tau)s_2(\{C \rightsquigarrow A, D,\ C \not\rightsquigarrow B\}, \tau)s_3(\{A \rightsquigarrow B,\ A \not\rightsquigarrow C, D\}, \tau)s_{10}$ is a valid path for the multi-hop constraint $D \dashrightarrow B$ that does not end with a $(b, \mathbf{E}(true\ _{a \vee \tau}\mathbf{U}_{B \dashrightarrow C}\ ctrue))$-transition.

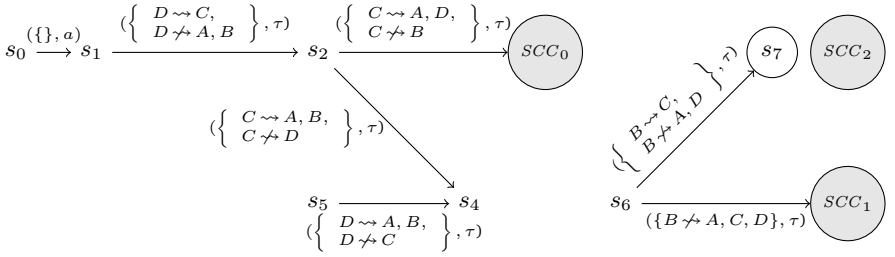


Figure 5.5: The CLTS used during backward analysis to check $\mathbf{A}^{D \dashrightarrow B}(true\ _{a \vee \tau}\mathbf{U}_b\ \mathbf{E}(true\ _{a \vee \tau}\mathbf{U}_c\ true))$: the white and gray states are of the first and second types respectively.

5.3.3 Model Checking EW Formulae

Formula $\mathbf{E}(\phi\ _{\chi}\mathbf{W}_{\chi'}\ \phi')$ is satisfied by ϕ-states that (1) either have a ζ-path of (χ, ϕ)-transitions that end with a (χ', ϕ')-transition, or (2) have a (χ, ϕ) ζ-path.

States of first type are found by calling procedure $CheckEU$. To find states of second type, terminal SCCs made of (χ, ϕ)-transitions are search for. Preceding states over (χ, ϕ)-transitions that conform to ζ satisfy $\mathbf{E}(\phi\ _{\chi}\mathbf{W}_{\chi'}\ \phi')$. To this aim, CLTS is restricted to ϕ-states that were not found by $CheckEU$ and χ-transitions which conform to ζ. The resulting CLTS is decomposed into SCCs, while each SCC is collapsed into a single state. A backward analysis in a breath-first fashion is performed from each terminal state. States visited during backward analysis satisfy $\mathbf{E}(\phi\ _{\chi}\mathbf{W}_{\chi'}\ \phi')$ under ζ and so do states within visited collapsed SCCs.

The pseudocode of the algorithm is given in Fig. 5.6. The procedure $CollapSCC$ (1) computes the SCCs of a CLTS, (2) collapses each SCC to a single state, (3) adds a mapping from each (collapsed) SCC to its states in the relation SCC. The FIFO queue $explore$ initially contains the states where the backward analysis starts from: ϕ-states with a (χ, ϕ)-path whose transitions conform to ζ. State visited during backward movement are stored in the set $visited$. A ϕ-states satisfies $\mathbf{E}(\phi\ _{\chi}\mathbf{W}_{\chi'}\ \phi')$ if either is in $visited$ or within an SCC. The correctness of the algorithm is implied by the correctness of the $\mathbf{EU}$ model checking algorithm and the fact that $\forall s \in explore\ (s \models_{\zeta} \mathbf{E}(\phi\ _{\chi}\mathbf{W}_{\chi'}\ \phi'))$.

Procedure $CheckEW(CLTS, \phi, \chi, \chi', \phi', \zeta)$
Output: The states that satisfy $\mathbf{E}(\phi\, {}_{\chi}\mathbf{W}_{\chi'}\, \phi')$ under ζ in
$CLTS \equiv \langle S, \Lambda, \rightarrow, s_0 \rangle$
$T_\phi := ModelCheck(CLTS, \phi, \zeta)$;
$T_{\phi'} := ModelCheck(CLTS, \phi', \zeta)$;
$visited := CheckEU(CLTS, \phi, \chi, \chi', \phi', \zeta)$;
$\rightarrow' := \{(t, (\mathcal{C}, \eta), t') \in \rightarrow \mid t, t' \in T_\phi \setminus visited,\ \eta \models \chi,\ \zeta \cap \neg\mathcal{C} = \emptyset\}$;
$\rightarrow'' := CollapSCC(\rightarrow')$;
$explore := \{s \mid s \text{ has no outgoing transition } \in \rightarrow''\}$;
while $explore \neq \emptyset$ **do**
  $t_0 := head(explore)$;
  $visited := visited \cup \{t_0\}$;
  $dequeue(explore)$;
  for all $(s, (\mathcal{C}, \eta), t_0) \in \rightarrow''$ **do**
    if $s \notin explore \cup visited$ **then** $enqueue(s, explore)$;
$result := \{t \in T_\phi \mid t \in visited \vee \exists\, s\, (t \in SCC(s))\}$;
return $result$;

Figure 5.6: Model checking **EW** formulae.

The time complexity to check an **EW** formula is $O(V + E)$. However model checking CTL formula over semantic models where topology is modeled as a part of states, following the approach of [64], has a time complexity of $O(2^{n^2-n} \times (V + E))$.

5.3.4 Model Checking AW Formulae

Model checking **AW** is similar to **AU**. In general a ϕ-state satisfies a formula $\mathbf{A}^{\mu}(\phi\, {}_{\chi}\mathbf{W}_{\chi'}\, \phi')$ under ζ if all its ζ-paths consist of (χ, ϕ)-transitions as long as either a (χ', ϕ')-transition is performed or the topology formula μ is violated. A ϕ-state trivially violates this property if it has an outgoing transition (which conforms to ζ) that is neither a (χ', ϕ')- nor a (χ, ϕ)-transition. Preceding states over a (χ, ϕ)-path that does not violate μ, and its transitions conform to ζ, violate $\mathbf{A}^{\mu}(\phi\, {}_{\chi}\mathbf{W}_{\chi'}\, \phi')$ and so do $\neg\phi$-states. Such states can be searched similar to **AU**.

5.4 Protocol Analysis with CACTL

We have implemented the model checking algorithms explained in Section 5.3 in Java[1]. CACTL model checker can be used to express and verify interesting properties of MANET protocols on CLTSs. In this section, we go through two

[1]The codes are available at https://github.com/fghassemi/CACTL

case studies over the AODVv2-11 specified in Section 4.5 and the leader election algorithm specified in Section 3.6.

5.4.1 Checking the Packet Delivery Property of AODV

The important property of *packet delivery* in routing or information dissemination protocols in the context of MANETs is: if there exists an end-to-end route (multi-hop communication path) between two nodes A and C *for a sufficiently long period of time*, then packets sent by A will eventually be received by C [62]. To be able to specify such a property, inspired by [62] we will extend our specification in Section 4.5 to include data packet handling (to forward the packet to its next hop towards the destination) in addition to the route discovery packets and their corresponding handlers. Furthermore, whenever a source node discovers a route to an intended destination, it starts forwarding its data packet through the next hop, specified in its routing table. The data packet is forwarded by intermediate nodes to their next hops. When the data packet reaches the intended destination, it delivers the data to itself by unicasting the $deliver$ message to itself. In case an intermediate node fails to forward the message, the error recovery procedure is initiated as explained in Section 4.5. Consequently, using the following formula, we can verify packet delivery property:

$$\mathbf{A}^{\,true}(true\ {}_{\neg rec\_newpkt(0,4)}\mathbf{W}_{rec\_newpkt(0,4)}\ \mathbf{A}^{\,n_1 \dashrightarrow n_4 \wedge n_4 \dashrightarrow n_1}(true\ {}_{\tau}\mathbf{U}_{deliver()}\ true))$$

It expresses that as long as there is a stable multi-hop path from n_1 to n_4 and vice versa (specified by $n_1 \dashrightarrow n_4 \wedge n_4 \dashrightarrow n_1$), any $rec\_newpkt(0,4)$ message is followed by a $delivery()$ message after passing τ-transitions which abstract away from other message communications. By model checking the resulting CLTS of the AODVv2 model, we found a scenario in which the above property does not hold. We explain this scenario (which is part of a counterexample to the above formula) in a network of three nodes N_1, N_2 and N_3, where node N_3 is always connected to the nodes N_1 and N_2, while the connection between the nodes N_1 and N_2 is transient. Thus, the mobility of nodes leads to the topologies shown in Fig. 4.8b and Fig. 4.8c. Assume the topology is initially as the one in Fig. 4.8b:

- Node N_1 unicasts a $rec\_newpkt(data, N_2)$ to itself, indicating that it wants to send $data$ to node N_2.
- Node N_1 initiates a route discovery procedure by broadcasting an $rreq_{N_1,0}$ message to its neighbors, i.e., nodes N_3 and N_2. Note that $rreq_{a,i}$ refers to an $rreq$ message received from node a with the hop count of i. Each $rreq$ message has more parameters but here only these two parameters are of interest and the other parameters are assumed to be equal for all the $rreq$ messages, i.e., the destination and source sequence numbers, and the source and destination IPs.

- Node N_3 processes the $rreq_{N_1,0}$ and since it is not the destination and has no route to N_2 in its routing table, rebroadcasts the $rreq_{N_3,1}$ message to its neighbors, nodes N_1 and N_2, after increasing the hop count. At this point, node N_2 has two messages in its queue, $rreq_{N_1,0}$ and $rreq_{N_3,1}$.
- Node N_1 moves out of the communication range of node N_2, resulting the network topology shown in Fig. 4.8c.
- Node N_2 takes $rreq_{N_1,0}$ from the head of its queue and updates its routing table by setting N_1 as the next hop in the route towards N_1. As node N_2 is the intended destination for the route discovery message, it unicasts an $rrep$ message towards the originator, N_1, indicating that the route has been built and it can start forwarding the data.
- Since the connection between the nodes N_1 and N_2 is broken, it fails to receive an ack from N_1 and marks the route as *invalid*.
- Node N_2 then takes $rreq_{N_3,1}$ from its queue and since the route state towards N_1 is *invalid*, it evaluates the received route to update the routing table. Since the hop count of the received message is greater than the existing one, it does not update the existing route and the message is discarded.

Although the route through node N_3 to node N_1 seems to be valid, the protocol refuses to employ it to prevent possible loop formation in the future. As we have reported in [154], the packet delivery property is still violated in AODVv2-16 in a scenario different from the above mentioned one.

5.4.2 Verification of the Leader Election Algorithm

The leader election algorithm for MANETs presented in Section 3.6 requires that *eventually* every node is in a stable state, i.e. has a unique leader which is the highest-valued node in its SCC. We show by means of our model checking algorithm that a MANET with four nodes, each deploying the protocol from [150], has a *weak* form of stabilization: if nodes stay strongly connected, then the algorithm converges to a desired state. Assume four node identifiers $A < B < C < D$. We investigate the property that "all nodes eventually choose D as their leader, if they are continuously in the same SCC", specified by the CACTL formula

$$p_1 \equiv \mathbf{AF}^{\mu_1}_{true}(\bigwedge_{x \in Loc} \mathbf{EX}_{finish(D,x)} true)$$

where $\mu_1 \equiv A \dashrightarrow B \wedge B \dashrightarrow C \wedge C \dashrightarrow D \wedge D \dashrightarrow A$.

We note that the strong fairness assumption in the CACTL model checking algorithm is essential for $\mathbf{AU}$ formulae to guarantee that executions will always escape from the self-loops of $finish$ actions.

Next we examine merging scenarios of the protocol: when two distinct SCCs merge, nodes adopt the node with the highest value in both as their leader. We

can verify the property "if an SCC consisting of A and B is connected to an SCC consisting of C and D, then D becomes the leader of all nodes" with the help of the CACTL formula

$$p_2 \equiv \mathbf{AG}^{true}_{true}((\bigwedge_{x\in\{A,B\}} \mathbf{EX}_{finish(B,x)} true) \wedge (\bigwedge_{y\in\{C,D\}} \mathbf{EX}_{finish(D,y)} true) \Rightarrow \mathbf{AF}^{\mu_1}_{true}(\bigwedge_{z\in Loc} \mathbf{EX}_{finish(D,z)} true))$$

The formula expresses that first nodes A, B and nodes C, D elect B and D as their leaders, respectively. As nodes nodes electing the same leader are in the same SCC, it can be inferred that first nodes A, B and nodes C, D elect belong to the same SCC. If these two SCCs merge (indicated by $\mathbf{AF}^{\mu_1}$), then they will eventually converge to the same leader D. The property becomes more complex by examining a scenario in which three SCCs merge. We can verify the property "if an SCC consisting of A and C is first connected to D and later to B, then D becomes the leader of all nodes" with the help of the CACTL formula

$$p_3 \equiv \mathbf{AG}^{true}_{true}((\bigwedge_{x\in\{A,C\}} \mathbf{EX}_{finish(C,x)} true) \wedge (\bigwedge_{y\in\{B,D\}} \mathbf{EX}_{finish(y,y)} true) \Rightarrow \mathbf{AF}^{\mu_2}_{true}((\bigwedge_{z\in\{A,C,D\}} \mathbf{EX}_{finish(D,z)} true) \Rightarrow \mathbf{AF}^{\mu_1}_{true}(\bigwedge_{w\in Loc} \mathbf{EX}_{finish(D,w)} true)))$$

where $\mu_2 \equiv A \dashrightarrow C \wedge C \dashrightarrow D \wedge D \dashrightarrow A$.

We can investigate scenarios in which a node gets disconnected from its parent or leader during or after the leader election phase respectively. One can verify that "a node will not change its leader as long as it is connected to it", specified by the CACTL formula

$$p_4 \equiv \mathbf{AF}^{A \dashrightarrow D \wedge D \dashrightarrow A}_{finish(D,A)} (\mathbf{AG}^{D \dashrightarrow A}_{\neg(leader(A,A)\vee leader(B,A)\vee leader(C,A))} true)$$

meaning that after D was selected as the leader of A when $A \dashrightarrow D \wedge D \dashrightarrow A$ (specified by the outer $\mathbf{AF}$ formula), A never chooses another leader (A, B or C), unless it gets disconnected from D.

We verified properties p_{1-4}; these properties are all satisfied by our model. We used the mCRL2 tool to generate the state space, modulo strong bisimilarity, as explained in Section 3.6.2. The generated LTSs by mCRL2 include network constraint information, and hence can be viewed as CLTSs. Table 5.1 illustrates the verification time on a Corei7 CPU with 8GB RAM.

5.5 Related Work

To investigate properties of MANETs, two approaches are followed in the literature: either a compact semantic model and a new logic are provided, or existing

Table 5.1: Verification times of the CACTL model checker.

No. of nodes	No. of states	No. of transitions	p_1	p_2	p_3	p_4
3	$4,278$	$33,536$	0.303s	0.358s	0.369s	0.275s
4	$357,024$	$3,928,890$	36.554s	53.204s	62.763s	91.190s

logics are used while arbitrary mobility is restricted to tackle the state space explosion problem.

The asynchronous process algebra bKlaim [122] follows the first approach similar to ours, based on finite *abstract labeled transition systems* where the influence of the network topology is preserved: each state, denoted by a multi-set of actions and data, represents networks whose multiplicity of each immediately available action and visible data is at most equal to that state. A variant of CACTL is proposed whose temporal operators are parametrized with a set of topologies to determine which properties hold if movement of nodes is restricted. To cope with the abstraction in the semantics, each logic formula is interpreted to either true/false, or unknown which evaluates to true/false on concrete transition systems. However, topology-dependent behaviors cannot be checked unless all topologies that satisfy the multi-hop condition are provided by the user.

Conversely, AWN [64] verifies topology-dependent behavior properties using CTL [34], by special treatment of (dis)connectivity information in transition labels. This approach can be extended to algebras, e.g. CMAN and ω-calculus, with (dis)connectivity information on transition labels. However, this approach needs auxiliary strategies to extract predicates from the states and transitions, to restrict connectivity changes during model checking and thus limit the state space. These challenges are also tackled with the help of the model checker UPPAAL, by transforming AWN specifications to automata and exploiting an auxiliary automaton, similar to [113], which statically restricts topologies to at most 5 nodes with up to one topology change, and scenarios with two new data packets.

5.6 Conclusion

We presented a branching-time temporal logic CACTL, which is interpreted over CLTSs. It is aimed at expressing topology-dependent behavioral properties of MANET protocols. We moreover introduced a model checking algorithm for CACTL that investigates the temporal behavior of MANETs, while taking into account the underlying topology, represented symbolically by means of network constraints. Scenarios like *after a route found* and *after two distinct SCCs are merged* can be investigated with the help of multi-hop constraints over topologies, which in CACTL are specified on top of path quantifiers.

Advantages of our approach are the flexibility in verifying topology-dependent behavior (without changing the model), and the facility to restrict the mobility

scenarios considered. By nesting path quantifiers, a set of specific topological paths can be specified with the help of topology constraints (without a need to specify how a topology constraint should be inferred). The (dis)connectivity information in CLTS transitions makes it possible to restrict the generality of mobility as desired.

Product Line Process Theory

6

Software product line (SPL) engineering has become an established trend in software development, where a family of similar software products with minor differences are developed in tandem, instead of developing each specific software product separately [128]. SPL engineering benefits from systematic reuse throughout the system life cycle and enables mass development and customization of numerous products. Hence, the development cost and the time to market for an SPL is substantially decreased, compared to the cumulative development cost and time of the isolated products [135]. To this aim, various software engineering activities have to be adapted to cope with the differences among the artifacts for different products, called *variability*. Variability introduces a new complexity dimension and hence, this calls for a genuine treatment of variability in different artifacts (such as requirement specification, architectural design, detailed design, and implementation artifacts). Such a treatment should also allow for a collective analysis of product line behavior (e.g., in testing and verification [58, 140]) to deal with the inherent complexity of SPLs.

At the highest level of abstraction, an SPL can be specified by a set of features that satisfy the specific needs of a particular market segment or mission [40]. A feature identifies "a prominent or distinctive unit of requirement which can be either a user-visible behavior, aspect, quality, or characteristic of a software system" [99]. Hence, a product can be specified by a subset of features. To specify an SPL, the features are organized in a hierarchical model, called a *feature model*. It identifies commonalities and differences among the products of the SPL in terms of their features and it identifies suitable relations among features, such as optional, mandatory, or mutually exclusive. More concrete specifications capture structural and behavioral aspects of an SPL. For instance, the architecture description languages Koala [123] and xADL [45] concentrate on structural modeling of SPLs. Modal transition systems (MTSs) [107] and featured transition systems (FTSs) [38], however, concentrate on behavioral modeling (we refer to [20] for an overview of such behavioral models). MTSs capture the behavior of SPLs by defining state transitions as optional or mandatory, while FTSs annotate transitions with a set of features. Behavioral models typically come equipped

with a product derivation method; e.g., a product, derived from a feature model, can project an FTS onto an LTS.

Formal verification techniques provide strong tools to analyze complex systems to guarantee their correctness. Product Line Calculus of Communicating Systems (PL-CCS) [89, 90] is an extension of Milner's Calculus of Communicating Systems (CCS) [118]. PL-CCS extends CCS by adding the binary variant operator $\oplus_i$ to model behavioral variability in SPLs. More specifically, process term $p_1 \oplus_1 p_2$, where p_1 and p_2 are CCS process terms, specifies a family of two alternative products, namely p_1 or p_2 (the index i in $\oplus_i$ is used to designate repeated choices that have to be made in the same way; when no repetition of indices is present, the indices can be safely ignored). The semantics of a PL-CCS specification is given in terms of three different models: the *flat semantics*, the *unfolded semantics*, and the *configured-transition semantics* [90]. A PL-CCS specification can be turned into a product, specified by a CCS term, by resolving the variability points, i.e., the variant operators, by deciding on whether their right or left process is chosen. The flat semantics of a PL-CCS term is given in terms of the semantics of all derivable products, denoted by CCS terms. A product family LTS (PF-LTS) is an extension of an LTS, where labels and states are paired with configuration vectors that maintain the configuration of variants. PF-LTSs provide the unfolded semantics of PL-CCS terms and are derived through a set of structural rules in a systematic way. The structural rules given in [90] work on a restricted set of PL-CCS terms in order to be compositional. The configured-transition semantics is defined over the unfolded semantics by merging all states that only differ in their configuration parts. This provides the most succinct model of PL-CCS terms. Hence, in the developments to come, we mostly focus on the configured-transition semantics of PL-CCS. In particular, we provide a set of structural rules that derive a configured-transition semantics for PL-CCS terms directly.

Equational reasoning is the cornerstone of the algebraic approach to process theory. To furnish PL-CCS with a proper equational theory, we study a number of notions of behavioral equivalence, based on strong bisimilarity. A summary of these notions, their properties and the results establishing their relationship is depicted in Fig. 6.1. We start with a set of axioms that we expect to be sound for a model of PL-CCS and define the notion of strict strong bisimilarity, which is a natural extension of strong bisimilarity in the SPL setting. Namely, strict strong bisimilarity requires bisimilar product lines to behave bisimilarly for all common configurations. This turns out to be a fully compositional notion for PL-CCS, but too strong a notion for some of our intuitive axioms. For example, strict strong bisimilarity rejects the intuitive axiom $p \oplus_i q = q \oplus_i p$ as it is sensitive to the placement of the processes in a binary variant composition. Subsequently, we introduce a coarser notion, called product line bisimilarity, which does satisfy the desired axioms of PL-CCS. However, this notion is shown to satisfy a weaker compositionality property. Namely, it is compositional for a subset of PL-CCS terms, called fully expanded terms. To remedy the latter issue, we show that

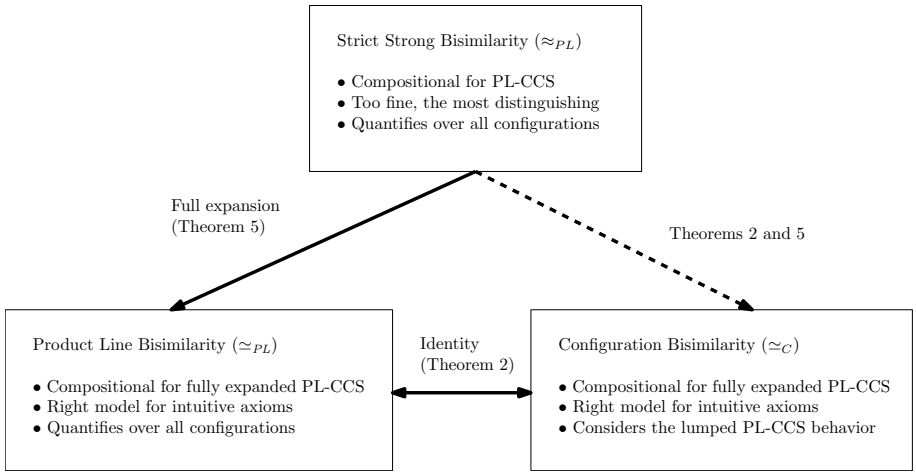


Figure 6.1: Our Notions of Behavioral Equivalence for PL-CCS Specifications

all PL-CCS terms can be rewritten into this subset using a sound transformation, thanks to the strong compositional notion of strict strong bisimilarity. Since strict strong bisimilarity implies product line bisimilarity this transformation is also sound for the latter notion and hence, resolves its compositionality issue.

Owing to the fact that a MANET protocol can be interpreted as a product family that behaves according to the underlying topology configuration, there is a similarity between the semantic models of MANETs and product families, i.e., CLTSs and PF-LTSs (or configured-transition systems): the network constraint of a transition enumerates the topologies for which a behavior is valid while the configuration vector enumerates the configurations of a family. We take advantage of our experience with CLTSs to define configuration bisimilarity, which is an alternative yet equivalent notion for product line bisimilarity. The main motivation for introducing configuration bisimilarity is that it allows for reasoning about the product line behavior as a whole and hence, dispenses with scrutinizing individual product behaviors.

Our axiomatization is useful to identify common parts (i.e., the mandatory parts) among the products of a family, to reorganize the functionality of a family specification to behaviors for which appropriate components exist, to derive products of a family to validate a model in terms of its intended systems (where each product is specified by a CCS term), and to manipulate functionalities assigned to products of a product line at the syntactic level.

Regarding an equational theory for PL-CCS, our work improves upon [89], where a number of algebraic laws to restructure a product line specification were given. However, we are not aware of any complete axiomatization of PL-CCS to date. For example, the laws of [89] are restricted to families with the same number of variants and exclude intuitive equalities like $p \oplus_i q = q \oplus_i p$ and

$p \oplus_i p = p$. It is worth noting that our notion of configuration bisimilarity is reminiscent of the notion of branching bisimilarity introduced in [16].

PL-CCS also comes equipped with a property specification language that is a variant of the multi-valued modal μ-calculus [90]. A configured transition system can also be viewed as a multi-valued modal Kripke structure [90] and hence, formulae in the multi-valued modal μ-calculus can be evaluated naturally in this semantic domain. The corresponding model checking method verifies a family at once, and its result defines the set of products that meet the given property. We show that the multi-valued modal μ-calculus of [90] is the logical characterization of our notion of product line bisimilarity (and hence, also our notion of configuration bisimilarity). This provides another evidence of suitability for our main notions of behavioral equivalence.

The contributions of this chapter can be summarized as follows:

- We introduce a means to specify product lines with infinite behavior, following the approach of [9], by extending PL-CCS with recursive specifications.
- We provide a set of structural rules to derive the configured-transition semantics of PL-CCS directly.
- We study different notions of bisimilarity over the configured-transition semantics. To this end, we provide a set of intuitive axioms that should be satisfied. We prove different compositionality (congruence) results for the notions of bisimilarity and relate them to each other.
- We provide a sound axiomatization of PL-CCS terms modulo product line bisimilarity, which additionally allows one to derive any sound equation on closed terms with finite-state behavior (in technical terms, it is a ground-complete axiomatization).
- We show that the multi-valued modal μ-calculus is the characterizing logic for our product line bisimilarity.

The rest of this chapter is structured as follows. Section 6.1 introduces the PL-CCS syntax and semantics. Section 6.2 defines our notions of behavioral equivalence and relates them. Section 6.3 provides a set of sound and ground-complete axioms to syntactically manipulate product lines. Section 6.4 illustrates the applicability of our axiomatization in the analysis of product lines. Section 6.5 presents the multi-valued modal μ-calculus as well as a logical characterization of product line bisimilarity. Section 6.6 provides an overview of existing approaches on modeling and verification of SPLs. Finally, Section 6.7 concludes the chapter.

6.1 PL-CCS : Syntax and Semantics

To our knowledge, PL-CCS [89, 90] is the first process algebra introduced to formally specify and verify product lines in an algebraic manner. To give a semantics to PL-CCS terms in [89, 90], the binary variant operators are assigned an index in a pre-processing step. Following the same principle, we index the binary variant operator of PL-CCS with a natural number. This allows for defining a unique semantic model for each PL-CCS term and also specifies multiple variation points that should be resolved in the same manner. Moreover, any unnumbered PL-CCS term can be considered an indexed PL-CCS term by assigning arbitrary distinct natural numbers to each and every binary variant operator.

To specify product lines with infinite behavior, we extend PL-CCS with the recursion operator $\langle X|E\rangle$ taken from [9]. It encompasses both the CCS recursion operator $recX{\cdot}t$ (which is specified in our syntax as $\langle X|X \stackrel{def}{=} t\rangle$) and the standard way to express recursion in ACP.

6.1.1 PL-CCS: Syntax

Let $\mathcal{A}$ be the set of process names which are used as recursion variables in recursive specifications and ranged over by A and B. Moreover assume that Σ a finite set of input action labels, $\overline{\Sigma} = \{\overline{a} \mid a \in \Sigma\}$ is the set of output actions, and $\tau \notin \Sigma \cup \overline{\Sigma}$ the unobservable action. Then, the set of all actions Act is defined as $\Sigma \cup \overline{\Sigma} \cup \{\tau\}$. By definition, we have that $\overline{\overline{a}} = a$.

The syntax of PL-CCS comprises deadlock 0, action prefix $a.-$ (for each $a \in Act$), choice $+$, binary variant $\oplus_i$ where $i \in \mathbb{N}$, and parallel composition $\|$. It also includes process renaming $[f]$ where $f : Act \mapsto Act$ is a renaming function with $f(\overline{a}) = \overline{f(a)}$, and $f(\tau) = \tau$, and restriction $\backslash L$ where $L \subseteq Act$. Additionally, it has process names, and recursion operators $\langle A|E\rangle$ where E is a *recursive specification* over $\mathcal{A}$, denoted by $E(\mathcal{A})$ for short and $A \in \mathcal{A}$. A recursive specification is defined by a set of recursive equations that contains precisely one recursive equation $A \stackrel{def}{=} t_A$ for each process name $A \in \mathcal{A}$, where t_A is a term over the PL-CCS signature and process names from $\mathcal{A}$. The PL-CCS syntax is summarized by the following grammar:

$$t ::= \quad 0 \mid a.t \mid t + t \mid t \oplus_i t \mid t \| t \mid t[f] \mid t \setminus L \mid A \mid \langle A|E\rangle$$

Process term $a.t$ denotes the process that first performs action a and then behaves as t. Alternative composition $t_1 + t_2$ nondeterministically behaves as t_1 or t_2. Variant operator $\oplus_i$ defines a behavioral variation point. Family $t_1 \oplus_i t_2$ consists of the two alternative families specified by t_1 and t_2. PL-CCS terms can be composed using the parallel composition operator $\|$. Process term $t_1 \| t_2$ denotes the concurrent execution of two processes t_1 and t_2, of which the actions can be interleaved or synchronized whenever t_1 and t_2 are ready to execute an input and the corresponding output action simultaneously. The process term

$t[f]$ behaves as t, with every action renamed according to the renaming function f. The process term $t \setminus L$ can perform any action that is not included in L. The recursion operator $\langle A|E\rangle$ represents a solution of the recursive specification $E(\mathcal{A})$ where A acts as the initial variable. A solution of a recursive specification $E(\mathcal{A})$ is a set of process terms $\{s_A \mid A \in \mathcal{A}\}$ such that if for all $A \in \mathcal{A}$, s_A is substituted for A, the equations of E correspond to equal elements (in the model of our equational theory), i.e., $s_A = t\{s_X/X|X \in \mathcal{A}\}$, where $A \stackrel{def}{=} t \in E$. The guardedness criterion which will be explained in Section 6.3, for recursive specifications ensures that this solution is unique. As far as unguarded recursions are concerned, following the approach of CCS and [9], we consider the solution that has the least set of transitions. In the remainder of this chapter, we use the notions of process term, product line, and family interchangeably.

Note that the term defining process name A in a recursive specification may include recursive specifications. A term is called *closed*, if every process name A occurs in the scope of a binding recursive specification $E(\mathcal{A})$ such that $A \in \mathcal{A}$. For instance, in the closed term $\langle X|\{X \stackrel{def}{=} a.0 \oplus_1 b.\langle Y|\{Y \stackrel{def}{=} Y + c.X\}\rangle\}\rangle$, X is bound by the outer recursive specification. As usual, we use the notation $t\{s/A\}$ to denote the substitution of a closed term s for every free occurrence of process name A in t. We use $\langle t|E\rangle$, where E is a recursive specification over $\mathcal{A}$, to denote $t\{\langle A|E\rangle/A \mid A \in \mathcal{A}\}$, i.e., t where, for all $A \in \mathcal{A}$, all free occurrences of A in t are replaced by $\langle A|E\rangle$.

By adopting $\langle A|E\rangle$ instead of the CCS recursion operator, we can easily specify SPLs in which a process name definition is shared. For instance, $\langle p_1|\{p_1 \stackrel{def}{=} p_2 \parallel p_2,\ p_2 \stackrel{def}{=} b.0 \oplus_1 c.0\}\rangle$ is equivalent to the CCS notation $\mu X.(\mu Y.b.0 \oplus_1 c.0 \parallel \mu Y.b.0 \oplus_1 c.0)$.

Intuitively, an index is bound in a term if it either occurs directly in the term or indirectly in the definition of any recursion variable occurring in the term. The bound indices of term t, denoted by $bi(t)$, can be found with the help of the auxiliary function fbi:

$$
\begin{array}{ll}
fbi(0,\mathcal{S}) = \emptyset & fbi(t \setminus L,\mathcal{S}) = fbi(t,\mathcal{S}) \\
fbi(a.t,\mathcal{S}) = fbi(t,\mathcal{S}) & fbi(t[f],\mathcal{S}) = fbi(t,\mathcal{S}) \\
fbi(t_1 + t_2) = fbi(t_1,\mathcal{S}) \cup fbi(t_2,\mathcal{S}) & fbi(t_1 \parallel t_2,\mathcal{S}) = fbi(t_1,\mathcal{S}) \cup fbi(t_2,\mathcal{S}) \\
fbi(t_1 \oplus_i t_2,\mathcal{S}) = \{i\} \cup fbi(t_1,\mathcal{S}) \cup fbi(t_2,\mathcal{S}) & fbi(A,\mathcal{S}) = \emptyset \\
\multicolumn{2}{l}{fbi(\langle A|E \cup \{A \stackrel{def}{=} t\}\rangle,\mathcal{S}) = fbi(\langle t|E \cup \{A \stackrel{def}{=} t\}\rangle,\mathcal{S} \cup \{A\}),\ \text{if } A \notin \mathcal{S}} \\
\multicolumn{2}{l}{fbi(\langle A|E \cup \{A \stackrel{def}{=} t\}\rangle,\mathcal{S}) = \emptyset,\ \text{if } A \in \mathcal{S}.}
\end{array}
$$

An index i is *free* in t iff it is not bound. We denote by $t[i/j]$ the term that is obtained by replacing all $\oplus_j$ by $\oplus_i$ in t (we dispense with the inductive definition as it is straightforward).

6.1.2 PL-CCS Semantics

Intuitively, the behavior of a product line family is defined by the cumulative behavior of its products. These products are obtained by resolving the choice in the binary variant operators. The resolution may take place at various points of execution and hence, to record such choices the semantics needs to record whether the choice is unresolved (denoted by $?$), resolved in favor of the left-hand-side product (denoted by L), or resolved in favor of the left-hand-side product (denoted by R). Also resolving one instance of a binary variant operator may resolve the choice for other instances. For instance, configuring the variant j as R in $(a.0 \oplus_i b.0) \oplus_j c.0$, makes it unnecessary to configure the variant i. The configuration status of variation points bound in a process term with maximum index n are recorded in a configuration vector $\nu \in \{L, R, ?\}^n$, where the ith element of the vector is denoted by $\nu|_i$. We denote by Config the set of all possible configuration vectors, ranged over by ν and λ. Expression $\nu|_{i/x}$ denotes the result of replacing the ith element of ν by $x \in \{L, R\}$. A configuration is called *full* when all its elements are configured, i.e., are in $\{L, R\}$. Two configurations ν and λ that do not have any conflict on a variation point are called *consistent*. This concept is formalized below.

Definition 6.1 (Consistent configuration vectors [90])**.** Configuration vectors $\nu, \lambda \in \{L, R, ?\}^n$ are *consistent*, denoted by $\nu \asymp \lambda$, if and only if $\forall i \in \{1, ..., n\} : ((\nu|_i = ?) \vee (\lambda|_i = ?) \vee (\nu|_i = \lambda|_i))$.

Given two consistent configuration vectors ν and λ, their *unification*, denoted by $\nu \odot \lambda$ merges the configurations of their variation points as follows: $(\nu \odot \lambda)|_i = X \in \{L, R, ?\}$ iff either $\nu|_i = X \wedge \lambda|_i = ?$ or $\nu|_i = ? \wedge \lambda|_i = X$ or $\nu|_i = \lambda|_i = X$.

Configuration vector ν' is *more concrete* than ν (or ν is *more abstract* than ν'), denoted by $\nu \sqsubseteq \nu'$, iff $\forall i \in \{1, ..., n\} : ((\nu|_i = ?) \vee (\nu|_i = \nu'|_i))$ [90]. Hence, each configuration vector ν represents a set of configuration vectors $\{\nu' \mid \nu \sqsubseteq \nu'\}$.

We briefly explained the three different semantic models of PL-CCS terms: the *flat semantics*, the *unfolded semantics*, and the *configured-transition semantics* [90]. Since our equivalence relation and the multi-valued modal μ-calculus are defined over the configured-transition semantics, we next elaborate on how this semantics is derived directly using our SOS rules.

The configured-transition semantics induces an LTS, in which the labels are pairs in $\mathit{Act} \times \mathit{Config}$. Formally, a configured-transition system (CTS) is a tuple $\langle S, s_0, \mathit{Act} \times \mathit{Config}, \rightarrow \rangle$, where S is a set of states, $s_0 \in S$ is an initial state and $\rightarrow \subseteq S \times \mathit{Act} \times \mathit{Config} \times S$ is a set of transition relations. The notation $s \xrightarrow{\alpha,\nu} s'$ is used for $(s, (\alpha, \nu), s') \in \rightarrow$ and is representative for all transitions $s \xrightarrow{\alpha,\nu'} s'$, where $\nu \sqsubseteq \nu'$. The CTS semantics of a PL-CCS term t has the set of all terms as its states, t as the initial state, and the least relation satisfying the rules in Fig. 6.2 as its transition relation.

The rule Prefix indicates the execution of a prefix action, where $\nu_?$ denotes a configuration vector in which no element is configured. Choice specifies the

$$\frac{}{a.t \xrightarrow{a,\nu_?} t} : Prefix \qquad \frac{t_1 \xrightarrow{a,\nu} t_1'}{t_1 + t_2 \xrightarrow{a,\nu} t_1'} : Choice$$

$$\frac{t \xrightarrow{a,\nu} t' \quad a \notin L}{t \setminus L \xrightarrow{a,\nu} t' \setminus L} : Res \qquad \frac{t_1 \xrightarrow{a,\nu} t_1' \quad \nu|_i \neq R}{t_1 \oplus_i t_2 \xrightarrow{a,\nu|_{i/L}} t_1'} : Select$$

$$\frac{t_1 \xrightarrow{a,\nu} t_1'}{t_1 \parallel t_2 \xrightarrow{a,\nu} t_1' \parallel t_2} : Par \qquad \frac{\langle t|E\rangle \xrightarrow{a,\nu} t' \quad A \overset{def}{=} t \in E}{\langle A|E\rangle \xrightarrow{a,\nu} t'} : Call$$

$$\frac{t_1 \xrightarrow{a,\nu} t_1' \quad t_2 \xrightarrow{\overline{a},\nu'} t_2' \quad \nu \asymp \nu'}{t_1 \parallel t_2 \xrightarrow{\tau,\nu\odot\nu'} t_1' \parallel t_2'} : Sync \qquad \frac{t \xrightarrow{a,\nu} t'}{t[f] \xrightarrow{f(a),\nu} t'[f]} : Rename$$

Figure 6.2: Operational Semantics to derive configured-transition systems

non-deterministic behavior of the choice operator in terms of its operands. Res defines that term $t \setminus L$ is only allowed to do actions that are not in L. $Select$ defines the behavior of a family in terms of its products, by deciding about the ith variant operator. The side condition prohibits any reconfiguration, if it was previously configured. $Call$ defines the behavior of $\langle A|E\rangle$ in terms of the behavior of the right-hand side of the equation $A \overset{def}{=} t$ in the recursive specification E. Par explains that a process in a parallel composition can proceed independent of the other parallel component. $Sync$ states that two processes in a parallel composition can be synchronized on an action, if both are ready to perform input and output counterparts simultaneously. Since t_1 and t_2 resolve variants in their scopes, their resolutions are unified for their parallel composition in $Sync$. Finally, $Rename$ renames all actions using a function f.

The symmetric versions of rules $Choice$, $Select$ (while each occurrence of R and L should be reversed), and Par are not given explicitly here for the sake of brevity.

Example 6.2. Using the rules in Fig. 6.2, the configured-transition semantics of $\langle X|E\rangle$, where $E = \{X \overset{def}{=} (a.(b.X \oplus_1 c.0) + d.0) \oplus_2 e.0\}$, is given in Fig. 6.3a. The derivation tree inducing the transition labeled by $a, \langle ?, L\rangle$ is given below:

$$\cfrac{\cfrac{\cfrac{\cfrac{}{a.(b.\langle X|E\rangle \oplus_1 c.0) \xrightarrow{a,\langle ?,?\rangle} b.\langle X|E\rangle \oplus_1 c.0}:Prefix}{a.(b.\langle X|E\rangle \oplus_1 c.0) + d.0 \xrightarrow{a,\langle ?,?\rangle} b.\langle X|E\rangle \oplus_1 c.0}:Choice}{(a.(b.\langle X|E\rangle \oplus_1 c.0) + d.0) \oplus_2 e.0 \xrightarrow{a,\langle ?,L\rangle} b.\langle X|E\rangle \oplus_1 c.0}:Select}{\langle X|E\rangle \xrightarrow{a,\langle ?,L\rangle} b.\langle X|E\rangle \oplus_1 c.0}:Call$$

Other transitions are derived in a similar way. On deriving the transitions of $b.\langle X|E\rangle \oplus_1 c.0$ with the help of $Prefix$ and $Select$, only the first variant point can

be configured, and consequently it returns to state $\langle X|E\rangle$ with the action $b, \langle L, ?\rangle$ or state 0 with the action $c, \langle R, ?\rangle$, as shown in Fig. 6.3a.

Returning to state $\langle X|E\rangle$ in Example 6.2, makes it possible to reconfigure any previously configured variant. For the sake of compositionality, resolutions of variants are open to any possible configuration in our SOS rules. However, as it is explained in the next paragraph and Section 6.2.1, in deriving the behavior of a product/a set of related products only consistent resolutions are followed.

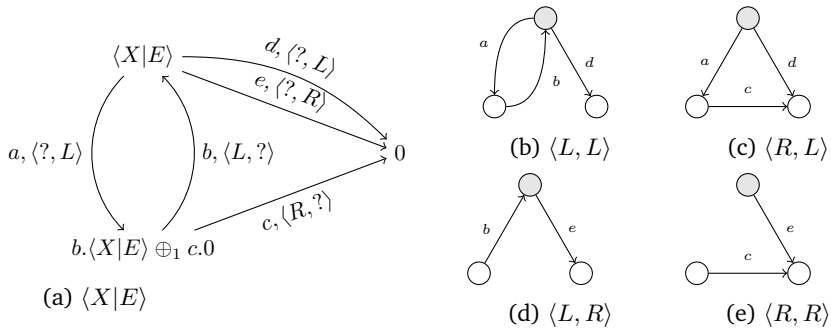


Figure 6.3: The configured-transition system of $\langle X|E\rangle$, and $\Pi(\langle X|E\rangle, \nu)$ for the given configuration vectors

The semantic model of each product of t, identified by the full configuration ν^f, can be derived by removing the transitions from t whose configuration vector is not consistent with ν^f. Let $\Pi(t, \nu^f)$ denote the resulting LTS. Formally speaking, $\Pi(t, \nu^f) \xrightarrow{a} \Pi(t', \nu^f)$ iff $t \xrightarrow{a,\nu} t'$ and $\nu \sqsubseteq \nu^f$. Therefore, only resolutions that are consistent with the full configuration, i.e., $\nu \sqsubseteq \nu^f$, are allowed and consequently reconfiguration is prohibited. See Fig. 6.3b, 6.3c, 6.3d, and 6.3e for the semantic models of products derived from X for the given configuration vectors $\langle L, L\rangle$, $\langle R, L\rangle$, $\langle L, R\rangle$, and $\langle R, R\rangle$ respectively (initial states are highlighted in gray). It should be noted that the semantic models derived for the configuration vectors $\langle L, R\rangle$ and $\langle R, R\rangle$ have only one reachable state from the initial state through the action e.

6.2 Bisimilarity for Product Line

Following the approach of ACP [22], we define a set of axioms (as the main part of our *process theory* or *equational theory*) as primary and then investigate the models that they have. The most intuitive model of a process theory is the algebra of closed terms (the algebra with the same operators of equational theory) modulo a congruence. In this section, we first discuss different equivalence relations to reason about product lines. Next, we discuss the congruence property of the previously defined relations.

Table 6.1: The axioms that product line bisimilarity should support.

$p \oplus_i q = q \oplus_i p,\ i \notin bi(p) \cup bi(q)$	A_1	$(p \oplus_i q) + r = (p + r) \oplus_i (q + r)$	A_5
$(p \oplus_i q) \oplus_j r = p \oplus_i (q \oplus_j r),$ $i \notin bi(r) \wedge j \notin bi(p)$	A_2	$r + (p \oplus_i q) = (r + p) \oplus_i (r + q)$	A_6
$p \oplus_i p = p$	A_3	$r \parallel (p \oplus_i q) = (r \parallel p) \oplus_i (r \parallel q)$	D_1
$a.(p \oplus_i q) = a.p \oplus_i a.q$	A_4	$(p \oplus_i q) \parallel r = (p \parallel r) \oplus_i (q \parallel r)$	D_2

Table 6.1 lists the axioms we have in mind for PL-CCS. We shall look for an appropriate notion of bisimilarity that supports the given equations. Axioms A_{1-3} define commutativity, associativity and idempotency for the binary variant operator. Axioms $A_{1,2}$ ensure that two families are equivalent when they produce the same set of products, irrespective of their orders in variant operators. However, their application is restricted: i should be free in p and q for A_1, while i should be free in r and j should be free in p for A_2. For instance, $a.0 \oplus_1 (b.0 \oplus_1 c.0)$ produces two products $a.0$ and $c.0$, but $(b.0 \oplus_1 c.0) \oplus_1 a.0$ produces $a.0$ and $b.0$, and consequently, as expected they are not equivalent. Axiom A_3 removes a repeated product from a family, and implies that two product families are equivalent iff they produce similar products, irrespective of their multiplicity. Axiom A_4 defines distributivity for prefix over binary variant, while axioms $A_{5,6}$ define distributivity for choice over binary variant. These rules allow for postponing the product selection by factorizing the common initial action/behavior respectively. Axioms $D_{1,2}$ define distributivity for the parallel over the binary variant. These two axioms reveal the difference between the alternative choice and the binary variant. Axioms $A_{5,6}$ and $D_{1,2}$ are useful to reduce redundancy by factorizing common parts.

6.2.1 Equivalence Relation

Strong bisimulation (see Definition 2.1) is very efficient to check and affords a neat theory: many other notions in the branching spectrum can be reduced to it by adding a standard set of axioms [147]. Definition 2.1 can be readily used for CTSs (since CTSs can be considered LTSs with a structure on the set of labels L). However, this simple adoption of Definition 2.1 can lead to some counter-intuitive observations.

For example, according to this definition, $a.0 + b.0$ is strongly bisimilar to $a.0 \oplus_1 b.0$. However, these two processes should not be considered equivalent intuitively. The family $a.0+b.0$ produces one product which has a non-deterministic behavior in performing actions a and b. By contrast, the family $a.0 \oplus_1 b.0$ produces two products, namely $a.0$ and $b.0$, and each product has deterministic behavior by performing solely a or solely b. As another concern, this relation cannot identify $a.(b.0 \oplus_1 c.0)$ and $a.b.0 \oplus_1 a.c.0$, while both have two products $a.b.0$ and $a.c.0$;

consequently, this bisimulation relation does not support axiom A_4. Therefore, the appropriate notion of bisimilarity over families must find for any product in one family a product in another such that their behaviors be strong bisimilar.

A full configuration is called *valid* with respect to term t, if its length is not less than the maximum index in $bi(t)$. For instance, $\langle L\rangle$ is not valid for $b.\langle X|E\rangle \oplus_1 c.0$, where $E = \{(a.(b.X \oplus_1 c.0) + d.0) \oplus_2 e.0\}$, while $\langle L, ?, L\rangle$ is valid. Let $VFConfig(t)$ denote the set of all valid full configurations with respect to t. Intuitively, two product families are equivalent when they produce bisimilar sets of products:

Definition 6.3 (Strict strong bisimulation)**.** Two product line terms s and t are *strictly strongly bisimilar*, denoted by $s \approx_{PL} t$, iff for any valid full configuration $\nu^f \in VFConfig(s) \cap VFConfig(t)$, $\Pi(s, \nu^f) \sim \Pi(t, \nu^f)$.

Strict strong bisimilarity does not support axioms $A_{1,2}$; to see this, observe that $\Pi(a.0 \oplus_1 b.0, \langle L\rangle) \nsim \Pi(b.0 \oplus_1 a.0, \langle L\rangle)$, $\Pi((a.0 \oplus_1 b.0) \oplus_2 c.0, \langle L, R\rangle) \nsim \Pi(a.0 \oplus_1 (b.0 \oplus_2 c.0), \langle L, R\rangle)$ and $\Pi((a.0 \oplus_2 b.0) \oplus_1 c.0, \langle R, L\rangle) \nsim \Pi(a.0 \oplus_2 (b.0 \oplus_1 c.0), \langle R, L\rangle)$. However, axiom A_3 is supported by strict strong bisimilarity, e.g., $(a.0 \oplus_1 b.0) \oplus_2 (a.0 \oplus_1 b.0) \approx_{PL} a.0 \oplus_1 b.0$ since for any $\nu^f \in \{\langle L, R\rangle, \langle L, L\rangle, \langle R, R\rangle, \langle R, L\rangle\}$, $\Pi((a.0 \oplus_1 b.0) \oplus_2 (a.0 \oplus_1 b.0), \nu^f) \sim \Pi(a.0 \oplus_1 b.0, \nu^f)$. See Appendix B.3 for its soundness proof.

To make Definition 6.3 insensitive to the placement of families in a binary variant composition, and consequently to support axioms $A_{1,2}$, the notion of bisimulation can be revised as follows.

Definition 6.4 (Product line bisimulation)**.** Two product line terms s and t are *product line bisimilar*, denoted by $s \simeq_{PL} t$, if and only if:

- $\forall \nu_1^f \in VFConfig(s) \cdot \exists \nu_2^f \in VFConfig(t) \cdot \Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$, and
- $\forall \nu_2^f \in VFConfig(t) \cdot \exists \nu_1^f \in VFConfig(s) \cdot \Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$.

Theorem 6.5. *Product line bisimilarity is an equivalence relation.*

See Section B.1 for the proof. Confining the behaviors of two CTSs into full configurations disallows any reconfiguration of variants whose resolutions are left open in their semantic models (as it was explained in Section 6.1.2).

Matching each and every valid full configuration in the product line bisimilarity is very tedious. This process may require examining all possible pairs of configurations to find a suitable match. (Yet it does eventually reduce to checking strong bisimilarity for a large, but finite, number of CCS processes, which is only decidable for CCS processes with finite-state behaviors [33].) Hence, it would be appealing to have an appropriate notion of bisimilarity that decides the equivalence of PL-CCS terms at once (without quantifying over the full configurations in the notion of product line bisimilarity). We provide such a relation, called configuration bisimulation, next and prove that it coincides with product line bisimilarity. Therefore, all the results for product line bisimilarity also hold for

configuration bisimilarity. Also, to facilitate reasoning about product lines and to establish an algebraic theory for product line processes, we provide a sound and complete axiomatization for product line and configuration bisimilarity. Using this axiomatization, a term can be restructured at the syntactic level to its equivalent terms without the need to generate and compare the state spaces. In the process of providing a sound and complete axiomatization, we use product line bisimilarity which makes our proofs much simpler.

We define a configuration-oriented notion of strong bisimulation inspired by [16], and our notion of branching reliable computed network bisimilarity (see Definition 3.1). In contrast to CLTSs, the transitions of a CTS are interrelated. For instance, the transitions a and b in $a.0 + (b.0 \oplus_1 c.0)$ are related together as they coexist in the same product. Hence, these two transitions cannot be matched independently to prohibit cases where the behaviors of a family is matched partially to the behavior of another family. This correlation among transitions motivates our definition of consistent transition sets. As a first step, we would like to classify the transitions in terms of their configuration vectors; the set of all transitions with consistent configurations is called a *consistent transition set*. Such transitions potentially belong to a family. Subsequently, we define a relation that relates two strongly similar states in terms of their consistent transition sets. In other words, it is also required to relate not only states on the basis of their behaviors, but also their consistent transition sets. Therefore, two states are related if one strongly simulates the other and their consistent transition sets can be matched. The former condition classifies states in terms of their behaviors (i.e., enabled actions) while the latter in terms of the correlation of behaviors (i.e., enabled configuration vectors).

Example 6.6. Consider Fig. 6.4, which illustrates the above mention bisimulation relation by an example. The terms in this figure are not product line bisimilar, as the left one consists of four (behaviorally different) products while the right one consists of two. However, a notion of bisimulation that only considers consistent transition sets in each state (as shown in Fig. 6.4) cannot distinguish them. The consistent transition set $\{(d, \langle L, ?\rangle), (a, \langle ?, ?\rangle)\}$ can be enabled together for a product/family identified by with the configuration vector ν as long as $\nu \sqsubseteq \langle L, ?\rangle \wedge \nu\langle ?, ?\rangle$. For instance, consider the product $d.0 + a.c.0$, which is identified by the full configuration $\langle L, R\rangle$ in the left CTS. It is derived from the consistent transition sets highlighted in Fig. 6.4. These consistent transition sets are matched to the corresponding sets in the right CTS. However, the matched sets do not belong to a family, and consequently cannot specify a product. The reason stems from the related states $b.0 \oplus_2 c.0$ and $b.0$. State $b.0 \oplus_2 c.0$ is reachable by the consistent transition set $\{(d, \langle L, ?\rangle), (a, \langle ?, ?\rangle)\}$ for two products (i.e., $\{\langle L, R\rangle, \langle L, L\rangle\}$). Therefore, its consistent transition set $\{(c, \langle ?, R\rangle)\}$ is enabled for product $\langle L, R\rangle$. However, its related state $b.0$ is only reachable for one product, which does not generate a matching consistent transition set.

To summarize, we need to note for which family a state is reachable in or-

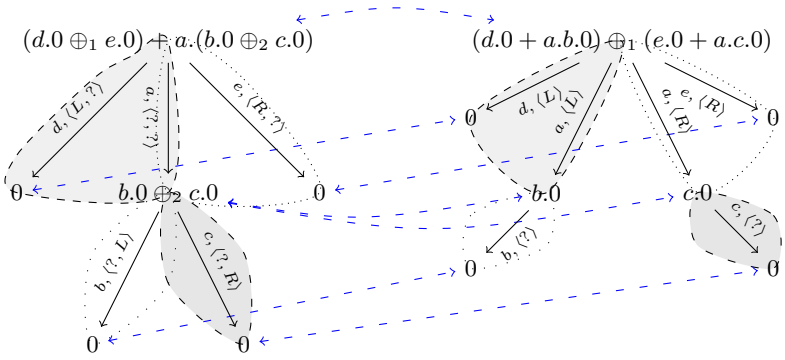


Figure 6.4: An example on defining a bisimulation regarding states and consistent transition sets: The consistent transitions sets of each state have been grouped together by dashed/dotted boundaries

der to match its enabled consistent transition sets with respect to that family. Furthermore, this book-keeping family disallows any reconfiguration of variants which have been previously resolved.

A *partitioning* of a configuration vector ν, denoted by p_ν, is a set of configuration vectors such that $\forall \nu_i, \nu_j \in p_\nu \cdot ((i \neq j) \Rightarrow \nu_i \not\asymp \nu_j)$, and $\forall \nu^f \cdot (\nu \sqsubseteq \nu^f \Rightarrow \exists \nu_j \in p_\nu \cdot (\nu_j \sqsubseteq \nu^f))$. For instance, $\{\langle ?, L\rangle, \langle L, R\rangle, \langle R, R\rangle\}$ is a partitioning of $\langle ?, ?\rangle$. A partitioning of ν is said to be *valid* for the state s, denoted by $Par(s, \nu, p_\nu)$, if and only if its elements (containing a product with a non-deadlock behavior in the state s) are at most as abstract as the configurations of the transitions of s. Formally, the predicate $Par(s, \nu, p_\nu)$ holds if and only if $\forall \nu_1 \in p_\nu \Rightarrow (\exists \alpha, \nu_1', s'(s \xrightarrow{\alpha, \nu_1'} s' \wedge \nu_1' \sqsubseteq \nu_1) \vee \nexists \alpha, \nu_1', s'(s \xrightarrow{\alpha, \nu_1'} s' \wedge \nu_1' \asymp \nu_1))$. This restriction the partitioning of $\langle R, ?, ?\rangle$ in the states $(b.0 \oplus_2 (0 \oplus_3 c.0))$ rejects a partitioning like $\{\langle R, L, ?\rangle, \langle R, R, ?\rangle\}$ but accepts $\{\langle R, L, ?\rangle, \langle R, R, R\rangle, \langle R, R, L\rangle\}$. For an arbitrary configuration vector ν and state s, the partitioning p_ν, such that $Par(s, \nu, p_\nu)$, classifies the transitions of s into a set of consistent transition sets; For any $\nu \in p_{\nu_1}$, transitions of s whose configurations are more abstract than ν constitute a consistent transition set. In the notion of configuration bisimulation defined below, states are related in terms of families. Assume s and t are related for families ν_1 and ν_2, respectively. Furthermore, the consistent transition sets of s defined by p_{ν_1}, where $Par(s, \nu_1, p_{\nu_1})$, are matched to the consistent transition sets of t defined by p_{ν_2}, where $Par(t, \nu_2, p_{\nu_2})$.

Definition 6.7 (Configuration bisimulation)**.** A family $\mathcal{R}$ of binary relations $R_{\nu_1,\nu_2} \subseteq S \times S$, where $\nu_1, \nu_2 \in Config$, is a *configuration simulation relation* if and only if for all $R_{\nu_1,\nu_2} \in \mathcal{R}$, and $s, t \in S$ such that $(s, t) \in R_{\nu_1,\nu_2}$, there exist p_{ν_1} and p_{ν_2}, where $Par(s, \nu_1, p_{\nu_1})$ and $Par(t, \nu_2, p_{\nu_2})$ such that for any $\nu_1' \in p_{\nu_1}$, there exists $\nu_2' \in p_{\nu_2}$ and:

- $s \xrightarrow{\alpha,\nu_1''} s'$ and $\nu_1'' \sqsubseteq \nu_1' \Rightarrow \exists t',\ \nu_2'' \cdot t \xrightarrow{\alpha,\nu_2''} t'$, $\nu_2'' \sqsubseteq \nu_2'$, and $(s',t') \in R_{\nu_1',\nu_2'}$;
- $t \xrightarrow{\alpha,\nu_2''} t'$ and $\nu_2'' \sqsubseteq \nu_2' \Rightarrow \exists s',\ \nu_1'' \cdot s \xrightarrow{\alpha,\nu_1''} s'$, $\nu_1'' \sqsubseteq \nu_1'$, and $(s',t') \in R_{\nu_1',\nu_2'}$,

where $R_{\nu_1',\nu_2'} \in \mathcal{R}$. $\mathcal{R}$ is a configuration bisimulation if $\mathcal{R}$ and $\mathcal{R}^{-1}$, where $R_{\nu_1,\nu_2} \in \mathcal{R} \Leftrightarrow R^{-1}_{\nu_1,\nu_2} \in \mathcal{R}^{-1}$, are configuration simulations. Two states $s,t \in S$ are called configuration bisimilar, denoted by $s \simeq_C t$, if and only if $(s,t) \in R_{\nu_?,\nu_?}$ for some family of configuration bisimulation relation $\mathcal{R}$ such that $R_{\nu_?,\nu_?} \in \mathcal{R}$.

Example 6.8. To inspect if $a.(b.0 \oplus_1 c.0)$ and $a.c.0 \oplus_1 a.b.0$ are configuration bisimilar, we should find a configuration bisimulation having the relation $R_{\langle ? \rangle,\langle ? \rangle} = \{(a.(b.0 \oplus_1 c.0), a.c.0 \oplus_1 a.b.0)\}$. For the configuration vector $\langle ? \rangle$, the partitioning $\{\langle ? \rangle\}$ and $\{\langle L \rangle, \langle R \rangle\}$ are both valid for $a.(b.0 \oplus_1 c.0)$, but only $\{\langle L \rangle, \langle R \rangle\}$ is valid for $a.c.0 \oplus_1 a.b.0$. If we consider the partitioning $\{\langle ? \rangle\}$ for $a.(b.0 \oplus_1 c.0)$, and match $\langle ? \rangle$ to both $\langle R \rangle$ and $\langle L \rangle$ of the partitioning of $a.c.0 \oplus_1 a.b.0$, then $(b.0 \oplus_1 c.0, b.0) \in R_{\langle ? \rangle,\langle R \rangle}$ and $(b.0 \oplus_1 c.0, c.0) \in R_{\langle ? \rangle,\langle L \rangle}$ must hold. However the family $\{R_{\langle ? \rangle,\langle ? \rangle}, R_{\langle ? \rangle,\langle R \rangle}, R_{\langle ? \rangle,\langle L \rangle}\}$ is not a configuration bisimulation, because the elements of the only valid partitioning $\{\langle L \rangle, \langle R \rangle\}$ of $\langle ? \rangle$ in the state $b.0 \oplus_1 c.0$ cannot be both matched to $\langle L \rangle$ for the state $a.c.0$ in the relation $R_{\langle ? \rangle,\langle L \rangle}$. Therefore, by considering the partitioning $\{\langle L \rangle, \langle R \rangle\}$ for $\langle ? \rangle$ in the state $a.(b.0 \oplus_1 c.0)$ and matching the configurations $\langle R \rangle$ and $\langle L \rangle$ of the partitioning for $a.(b.0 \oplus_1 c.0)$ to $\langle L \rangle$ and $\langle R \rangle$ of the partitioning for $a.c.0 \oplus_1 a.b.0$ (and vice versa), the family of relations $R_{\langle ? \rangle,\langle ? \rangle} = \{(a.(b.0 \oplus_1 c.0), a.c.0 \oplus_1 a.b.0)\}$, $R_{\langle R \rangle,\langle L \rangle} = \{(b.0 \oplus_1 c.0, c.0), (0,0)\}$, $R_{\langle L \rangle,\langle R \rangle} = \{(b.0 \oplus_1 c.0, b.0), (0,0)\}$ is achieved which constitutes a configuration simulation relation. However $a.(b.0 \oplus_1 0) \not\simeq_C a.b.0$ as the only valid partitioning of $\langle ? \rangle$ for the state $b.0 \oplus_1 0$ is $\{\langle R \rangle, \langle L \rangle\}$ and then the behavior $b.0 \oplus_1 0$ for the family $\langle R \rangle$ (i.e., the empty consistent transition set) cannot be matched to any behavior of $a.b.0$.

Partitioning the transitions of a state, guided by its consistent transition set, was inspired by our notion of branching reliable computed network bisimilarity, where a network constraint is partitioned (see Definition 3.1), while parameterizing the relation with matched partitioning was inspired by [16].

Theorem 6.9. *For any PL-CCS s and t, $s \simeq_{PL} t \Leftrightarrow s \simeq_C t$.*

Proof. We assume $s \simeq_{PL} t$, we show that $s \simeq_C t$. The assumption $s \simeq_{PL} t$ implies that there is a set of pairs of valid full configurations (ν_1^f, ν_2^f), where $\nu_1^f \in \mathit{VFConfig}(s)$ and $\nu_2^f \in \mathit{VFConfig}(t)$, such that $\Pi(s,\nu_1^f) \sim \Pi(t,\nu_2^f)$. Construct $\mathcal{R} = \{R_{\nu_1^f,\nu_2^f} \mid \Pi(s,\nu_1^f) \sim \Pi(t,\nu_2^f)\}$ where $R_{\nu_1^f,\nu_2^f}$ is defined as:

$$R_{\nu_1^f,\nu_2^f} = \{(s',t') \mid \Pi(s,\nu_1^f) \sim \Pi(t,\nu_2^f) \text{ witnessed by } R' \wedge (\Pi(s',\nu_1^f), \Pi(t',\nu_2^f)) \in R'\}$$

It is easy to check that $\mathcal{R}$ is a configuration bisimulation relation.

We assume $s \simeq_C t$ is witnessed by the configuration bisimulation $\mathcal{R}$, we show that $s \simeq_{PL} t$. To this aim, for any $\nu_1^f \in VFConfig(s)$, we find $\nu_2^f \in VFConfig(t)$ such that $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$ and vice versa. Let $\mathcal{B} = \{\nu_2 \mid R_{\nu_1,\nu_2} \in \mathcal{R} \wedge \nu_1 \sqsubseteq \nu_1^f\}$. Choose $b \in \mathcal{B}$ such that $\nexists \nu \in \mathcal{B} \setminus \{b\} \cdot b \sqsubseteq \nu$. Take a configuration ν_2^f such that $b \sqsubseteq \nu_2^f$. It is trivial that $R' = \{(\Pi(s', \nu_1^f), \Pi(t', \nu_2^f)) \mid (s', t') \in R_{\nu_1,\nu_2} \wedge R_{\nu_1,\nu_2} \in \mathcal{R} \wedge \nu_1 \sqsubseteq \nu_1^f \wedge \nu_2 \sqsubseteq \nu_2^f\}$ is a strong bisimulation witnessing $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$. □

Since product line and configuration bisimilarity coincide, with the aim to provide a sound and complete axiomatization, we use product line bisimilarity which makes our proofs much simpler. All results for product line bisimilarity also hold for configuration bisimilarity.

6.2.2 Congruence Property

In this section, we study the congruence property of strict and product line bisimulation relations, and provide a syntactic restriction over PL-CCS terms that makes product line bisimilarity a congruence with respect to the PL-CCS operators.

Strict strong bisimilarity is a congruence for PL-CCS terms.

Theorem 6.10. *Strict strong bisimilarity is a congruence for the PL-CCS term algebra.*

See Section B.1 for the proof.

The case for product line bisimulation is a bit more intricate. To illustrate the involved issues, observe that $(d.0 \oplus_1 e.0) \parallel (b.0 \oplus_1 c.0) \not\simeq_{PL} (d.0 \oplus_1 e.0) \parallel (c.0 \oplus_1 b.0)$, while $b.0 \oplus_1 c.0 \simeq_{PL} c.0 \oplus_1 b.0$. The reason is that the configurations $\langle R \rangle$ and $\langle L \rangle$ of $b.0 \oplus_1 c.0$ are matched to the configurations $\langle L \rangle$ and $\langle R \rangle$ of $c.0 \oplus_1 b.0$ respectively, but each pair of matched configurations chooses a different product in $d.0 \oplus_1 e.0$. However, $(d.0 \oplus_1 e.0) \oplus_2 (b.0 \oplus_1 c.0) \simeq_{PL} (d.0 \oplus_1 e.0) \oplus_2 (c.0 \oplus_1 b.0)$, since $\oplus_2$ makes the configurations of its operands independent of each other. To guarantee congruence for product line bisimilarity, we impose a constraint on the PL-CCS syntax. In $(d.0 \oplus_1 e.0) \parallel (b.0 \oplus_1 c.0)$, there are two binary variants indexed by 1. Hence, once one resolves the choice between $d.0$ and $e.0$, the same choice has to made between $b.0$ and $c.0$. We call a product line *fully expanded* when all its variants can be configured independently from the configuration of other variation points. The previous example is not fully expanded as its binary variants can not be resolved independently.

This constraint was first introduced in [90] with the different intention of compositionality for their structural rules. We revise their definition to enforce that different binary variant choices can be made independent of each other. To formally define a fully expanded term, we use its *term-dependency graph* which is a directed labeled graph. Its construction for a term t is explained on the term $(a.\langle Y|E\rangle + e.0) \parallel \langle Z|E\rangle$, where $E = \{Y \stackrel{def}{=} b.0 \oplus_1 c.Z, Z \stackrel{def}{=} c.Z \oplus_2 d.Y\}$. Its nodes

comprise the nodes of the parse tree of t together with additional nodes labeled $\langle A_i|E_i\rangle$ and E_i (i.e., the gray and the white nodes in Fig. 6.5, respectively). Its edges comprise the edges of the parse tree of t (i.e., the thick solid edges in Fig. 6.5) plus edges connecting $\langle A_i|E_i\rangle$ to the node labeled A_i in the term-dependency graph of E_i (i.e., the dashed edges in Fig. 6.5), and the edges of recursive specifications E_i. The term-dependency graph of $E_i(\mathcal{A}_i)$ consists of nodes labeled A_i for each $A_i \in \mathcal{A}_i$, together with the nodes of the parse trees for term t_i for each $A_i \stackrel{def}{=} t_i \in E_i$. Its edges comprise the edges of the parse trees (i.e., the thin solid edges in Fig. 6.5) plus the edges connecting A_i to the roots of the parse trees of the corresponding right-hand sides, i.e., t_i (i.e., the thick dashed edges in Fig. 6.5). Additionally, we add edges from leaves of the parse trees labeled A_i to the node labeled with A_i in its binding recursive specification (i.e., the dotted edges in Fig. 6.5).

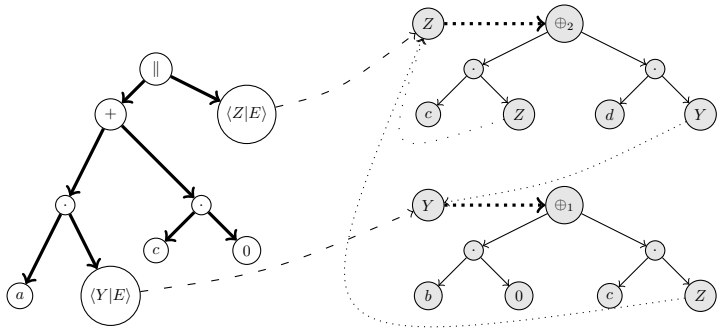


Figure 6.5: The term-dependency graphs of $(a.\langle Y|E\rangle + e.0) \parallel \langle Z|E\rangle$, where $E = \{Y \stackrel{def}{=} b.0 \oplus_1 c.Z, Z \stackrel{def}{=} c.Z \oplus_2 d.Y\}$

Definition 6.11 (Fully expanded terms)**.** A PL-CCS term is *fully expanded* if and only if in its term-dependency graph, for each two distinct simple paths starting at an arbitrary common node and ending at (common or distinct) nodes labeled with $\oplus_i$, they have both passed through a common node that is labeled with $\oplus_j$, for some $j \neq i$.

Recall that a simple path is a path in a graph which does not have repeating vertices. This constraint rules out systems such as $\langle p_1|\{p_1 \stackrel{def}{=} p_2 \parallel p_2,\ p_2 \stackrel{def}{=} b.0 \oplus_1 c.0\}\rangle$, since the parallel composition in the equation of p_1 has two simple paths to $\oplus_1$. However, these paths do not pass through a node labeled with $\oplus_j$, $j \neq 1$, beforehand. Nevertheless, it accepts systems such as $\langle p_4|\{p_4 \stackrel{def}{=} (p_2 \oplus_2 p_2) \parallel (b.0 \oplus_3 c.0), p_2 \stackrel{def}{=} b.0 \oplus_1 c.0\}\rangle$, since all simple paths of the parallel composition in the equation p_4 to $\oplus_1$ have already passed through $\oplus_2$, as shown

in Fig. 6.6. The same holds for simple paths of the node labeled with p_4. Such systems were not accepted by the Definition given in [90]. The term-dependency graph of Fig. 6.5 satisfies the condition of Definition 6.11.

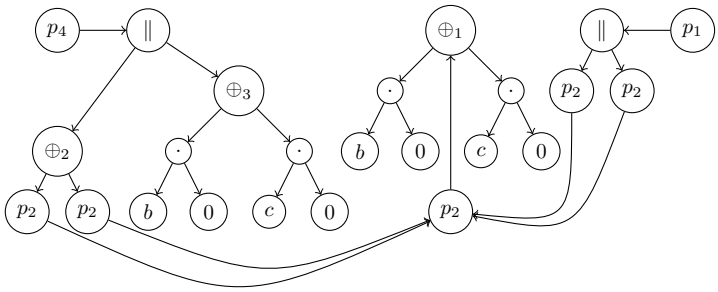


Figure 6.6: The term-dependency graphs of p_1 and p_4; the term-dependency graph of p_2 is shared

Restricting to fully expanded PL-CCS terms prevents compositing two product line bisimilar terms s and t with a term t' with a dependent binary variant which may be resolved incompatible in the terms s and t. Fully expandedness resolves the incompatibility problem between dependent variants.

Theorem 6.12. *Product line bisimilarity is a congruence on the fully expanded PL-CCS term algebra.*

See Section B.1 for the proof. Restricting to fully expanded PL-CCS terms is not important in practice (when terms are manipulated at the syntactic level), since a term can be rewritten using our axioms supported by strict strong bisimilarity into a fully expanded form (see Theorem 6.13 in Section 6.3.2). Note that the side condition of axiom A_2 guarantees that a fully expanded term remains fully expanded after being restructured by this axiom. For instance, although $(a.0 \oplus_1 b.0) \oplus_2 (c.0 \oplus_1 d.0)$ is fully expanded, $a.0 \oplus_1 (b.0 \oplus_2 (c.0 \oplus_1 d.0))$ is not.

6.3 Equational Reasoning on PL-CCS Terms

We extend the axioms given in Section 6.2 to reason about parallel, recursive behaviors with finite-state models, and indexed binary variants. To axiomatize the interleaving behavior of parallel composition and terms with variants with an identical index, we extend the PL-CCS syntax and semantics with new operators in Section 6.3.1.

We provide PL-CCS axioms that are sound with respect to product line bisimilarity in Section 6.3.2. Furthermore, we identify those that are also valid with respect to strict strong bisimilarity. Our axiomatization in Table 6.2 subsumes standard axioms of CCS for choice operator (C_{1-4}, R_{1-4}, E_{1-4}), axioms of ACP corresponding to the well-known elimination theorem [8] ($P_{1-3,5,6}$ and $S_{1,2,4-6}$),

and axioms of CCS to reason about recursive behaviors [145] ($Fold$, $UnFold$, and Ung). We explain that a term can be manipulated using axioms supported by strict strong bisimilarity (without changing the order of operands in binary variants) to be rewritten into a fully expanded form, and then manipulated with all axioms valid for product line bisimilarity. We prove ground-completeness (completeness over closed terms) of our axiomatization for a subset of PL-CCS terms, namely, those with finite-state behaviors in Section 6.3.3.

6.3.1 Extending PL-CCS Framework

Our axiomatization borrows from the process algebra *ACP* [21] two auxiliary operators (left merge and communication merge) to axiomatize the interleaving and the synchronizing behavior of parallel composition, respectively. Furthermore, we extend the process theory with two new sets of indexed operators (left and right selector), to restrict the behavior of a term regarding the configuration of the variant indexed by that number. The SOS rules of operators to derive CTSs are given in Fig. 6.7.

$$\frac{t_1 \xrightarrow{a,\nu} t_1'}{t_1 \between t_2 \xrightarrow{a,\nu} t_1' \parallel t_2} : LMerge \qquad \frac{t_1 \xrightarrow{a,\nu} t_1' \quad t_2 \xrightarrow{\overline{a},\nu'} t_2' \quad \nu \asymp \nu'}{t_1 \mid t_2 \xrightarrow{\tau,\nu\odot\nu'} t_1 \parallel t_2} : Merge$$

$$\frac{t \xrightarrow{a,\nu} t' \quad \nu|_i \neq R}{L(t,i) \xrightarrow{a,\nu} L(t',i)} : LSelect \qquad \frac{t \xrightarrow{a,\nu} t' \quad \nu|_i \neq L}{R(t,i) \xrightarrow{a,\nu} R(t',i)} : RSelect$$

Figure 6.7: SOS rules for auxiliary operators

In the left merge composition $t_1 \between t_2$, the left operand (t_1) performs an action and then continues in parallel with t_2. In the communication merge $t_1 \mid t_2$, both operands are synchronized on their initial actions and then continue in parallel composition. The left selector operator $L(t,i)$ makes all variants indexed by i be configured as left. Therefore, it only allows behaviors whose configurations on the variant indexed by i are consistent with left. The right selector operator $R(t,i)$ behaves symmetrically.

6.3.2 PL-CCS Axiomatization

We proceed to complete the axiomatization of PL-CCS modulo product line bisimilarity. The axioms are given in Table 6.2. Axioms C_{1-4} define commutativity, associativity, idempotence, and unit element for the choice operator.

Axiom P_1 defines the parallel composition of two families in an interleaving semantics, as in the process algebra *ACP* [21]; axiom P_2 explains the behavior of the left merge operator in terms of its left operand, if it can do an action. However, if it cannot do any action, the result is a deadlock, as explained by P_6. Axioms $P_{3,4}$ define left-distributivity of the choice and the binary variant over

the left merge operator, respectively. Axiom P_5 defines right-distributivity of binary variant over the left merge operator. Axiom S_1 defines the commutativity property for the communication merge operator. Axioms $S_{2,3}$ (together with S_1) define distributivity of choice and binary variant over the communication merge operator respectively. Axioms $S_{4,5,6}$ define the behavior of communication merge operator; when the operands are ready to do matched input and output communication actions, they can be synchronized and the result of their synchronization is the unobservable action τ as explained by S_4. However, if either they are not matched (S_5) or one of the operands cannot do any action (S_6), the result is a deadlock.

Table 6.2: The axiomatization of PL-CCS terms: for axioms N_2 and N_4, $i \neq j$.

$p + q = q + p$	C_1	$(p + q) + r = p + (q + r)$	C_2
$p = p + p$	C_3	$0 + p = p$	C_4
$p \parallel q = (p \between q) + (q \between p) + (p \mid q)$	P_1	$p \mid q = q \mid p$	S_1
$a.p \between q = a.(p \parallel q)$	P_2	$(p + q) \mid r = (p \mid r) + (q \mid r)$	S_2
$(p + q) \between r = (p \between r) + (q \between r)$	P_3	$(p \oplus_i q) \mid r = (p \mid r) \oplus_i (q \mid r)$	S_3
$(p \oplus_i q) \between r = (p \between r) \oplus_i (q \between r)$	P_4	$(a.p) \mid (\overline{a}.q) = \tau.(p \parallel q)$	S_4
$p \between (q \oplus_i r) = (p \between q) \oplus_i (p \between r)$	P_5	$(a.p) \mid (b.q) = 0,\ (b \neq \overline{a}) \vee (a = \tau)$	S_5
$0 \between p = 0$	P_6	$0 \mid p = 0$	S_6
$(a.p)[f] = f(a).(p[f])$	R_1	$(a.p) \setminus L = a.(p \setminus L),\ a \notin L$	E_1
$(p + q)[f] = (p[f]) + (q[f])$	R_2	$(a.p) \setminus L = 0,\ a \in L$	E_2
$(p \oplus_i q)[f] = (p[f]) \oplus_i (q[f])$	R_3	$(p + q) \setminus L = (p \setminus L) + q \setminus L)$	E_3
$0[f] = 0$	R_4	$0 \setminus L = 0$	E_4
		$(p \oplus_i q) \setminus L = (p \setminus L) \oplus_i (q \setminus L)$	E_5
$L(p \oplus_i q, i) = L(p, i)$	N_1	$L(p \oplus_j q, i) = L(p, i) \oplus_j L(q, i)$	N_2
$R(p \oplus_i q, i) = R(q, i)$	N_3	$R(p \oplus_j q, i) = R(p, i) \oplus_j R(q, i)$	N_4
$p \oplus_i q = L(p, i) \oplus_i R(q, i)$	N_5	$p = p[k/j],\ k \notin bi(p)$	N_6

$$\langle A | E \cup \{B \stackrel{def}{=} t\}\rangle = \langle\langle A|E\rangle | \{B \stackrel{def}{=} t\}\rangle \qquad Dec$$

$$\langle A | \{A \stackrel{def}{=} t\}\rangle = \langle t | \{A \stackrel{def}{=} t\}\rangle \qquad UnFold$$

$$s = t\{s/A\} \Rightarrow s = \langle A | \{A \stackrel{def}{=} t\}\rangle,\ \ A \text{ is guarded in } t \qquad Fold$$

$$\langle A | \{A \stackrel{def}{=} A + t\}\rangle = \langle A | \{A \stackrel{def}{=} t\}\rangle \qquad Ung$$

$$\langle A | \{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle = \langle A | \{A \stackrel{def}{=} t_1\}\rangle \oplus_i \langle A | \{A \stackrel{def}{=} t_2\}\rangle \qquad Dri$$

Axioms $R_{1,4}$ and $E_{1,2,4}$ define the behavior of renaming and restriction operators respectively. Axioms $R_{2,3}$ and $E_{3,5}$ define distributivity of choice and binary

variant over the renaming and restriction operators, respectively.

Axiom Dec decomposes a recursive specification E made up of (finitely many) multiple equations into several nested recursive specifications made up of a single equation. Axiom $UnFold$ expresses the existence of a solution for any recursive specification E: the constant $\langle A|E\rangle$ is a solution of the recursive specification E. $Fold$ expresses uniqueness of a solution for a guarded recursive specification: if y is a solution for A in E, and E is guarded, then $y = \langle A|E\rangle$. Note that $UnFold$ and $Fold$ are the consequences of *Recursive Definition Principle* (*RDP*) and *Recursive Specification Principle* (*RSP*) in *ACP*. An occurrence of a process name A in t is called *guarded*, if and only if this occurrence is in the scope of an action prefix operator. A recursive specification is called *guarded* if and only if all occurrences of all the process names in the right-hand sides of all its equations are guarded, or it can be rewritten to such a recursive specification using the axioms of the theory and the equations of the specification [9]. This guardedness criterion ensures every guarded recursive specification has a unique solution (in the model of closed PL-CCS terms modulo product line bisimilarity). By application of axiom Ung, it possible to turn the unguarded recursive specification $\{A \stackrel{def}{=} A + t\}$ into a guarded one. By application of axiom Dri, one can derive the products of a recursive specification: the solution of a recursive specification $\langle A|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle$ is $\langle A|\{A \stackrel{def}{=} t_1\}\rangle \oplus_i \langle A|\{A \stackrel{def}{=} t_2\}\rangle$. Axioms $UnFold$, $Fold$, and Ung are standard for CCS terms (with finite state behaviors modulo branching bisimulation) [145].

Axioms N_{1-5} handle binary variants with an identical index. Whenever the left (right) operand is selected in $p \oplus_i q$, then all i-indexed variants in p (q), should select their left (right) operands accordingly. Axiom N_5 removes all occurrences of $\oplus_i$ at the root of $p \oplus_i q$ in operands p and q. In other words, with the help of N_{1-5}, the occurrence of i in subtrees of $p \oplus_i q$ becomes unique. For instance, consider the product line $\langle X|\{X \stackrel{def}{=} b.X \oplus_1 a.0\}\rangle$, with two products $\langle X|\{X \stackrel{def}{=} b.X\}\rangle$ and $\langle X|\{X \stackrel{def}{=} a.0\}\rangle$ obtained using axiom Dri. However, by applying axiom $UnFold$ and substituting for X its defining term, one can derive $\langle X|\{X \stackrel{def}{=} b.X \oplus_1 a.0\}\rangle = (b.(\langle X|\{X \stackrel{def}{=} b.X \oplus_1 a.0\}\rangle) \oplus_1 a.0) = b.(\langle X|\{X \stackrel{def}{=} b.X\}\rangle \oplus_1 \langle X|\{X \stackrel{def}{=} a.0\}\rangle) \oplus_1 a.0$. By axioms A_4 and $N_{1,5}$, $b.(\langle X|\{X \stackrel{def}{=} b.X\}\rangle \oplus_1 \langle X|\{X \stackrel{def}{=} a.0\}\rangle) \oplus_1 a.0$ is reduced to $b.\langle X|\{X \stackrel{def}{=} b.X\}\rangle \oplus_1 a.0$, which derives two products $b.\langle X|\{X \stackrel{def}{=} b.X\}\rangle$ and $a.0$ (that are strongly bisimilar to the products of $\langle X|\{X \stackrel{def}{=} b.X \oplus_1 a.0\}\rangle$). Axiom N_6 changes the index of a binary variant term. For example, $(a.0 \oplus_1 b.0) \oplus_2 (c.0 \oplus_1 d.0) = (a.0 \oplus_1 b.0) \oplus_2 (c.0 \oplus_3 d.0)$. Consequently, axiom A_2 can be applied, resulting in $(a.0 \oplus_1 b.0) \oplus_2 (c.0 \oplus_1 d.0) = a.0 \oplus_1 (b.0 \oplus_2 (c.0 \oplus_3 d.0))$.

It should be noted that $t \approx_{PL} s$ implies $t \simeq_{PL} s$. All axioms, except for $A_{1,2}$ and N_6, are sound for strict strong bisimilarity. Generally speaking, to manipulate a PL-CCS term, as stated in Theorem 6.13, first it can be converted into a fully expanded form with the help of axioms supported by strict strong bisimilarity,

and then manipulated with all axioms.

Theorem 6.13. *Any PL-CCS term t can be rewritten by axioms A_{4-6}, $D_{1,2}$, $P_{4,5}$, $S_{1,3}$, R_3, E_5, Dri, and N_{1-5} into a form that is fully expanded.*

See Section B.2 for the proof. With the help of axioms A_{4-6}, $P_{4,5}$, $S_{1,3}$, R_3, E_5, and Dri, one can convert a PL-CCS term into a term such that its binary variants do not occur in the scope of any CCS operators. Then, by axioms N_{1-5}, the term becomes fully expanded. Therefore, it can be manipulated using all the axioms. The soundness of derivations, when the order of binary variants operands are fixed, follows from Theorem 6.10, and when terms are fully expanded, follows from Theorem 6.12.

Theorem 6.14 (Soundness)**.** *The axiomatization, given in Tables 6.1 and 6.2 (while the application of the axioms $A_{1,2}$ and N_6 is restricted to fully expanded terms), is sound for the term algebra $\mathbb{P}$(PL-CCS)$/\simeq_{PL}$, i.e., for all closed PL-CCS terms t_1 and t_2, $t_1 = t_2$ implies $t_1 \simeq_{PL} t_2$.*

See Section B.3 for the proof. The proof is a consequence of Theorems 6.10, 6.12, 6.13, and the fact that all the axioms except $A_{1,2}$ and N_6 are sound with respect to strict strong bisimilarity.

Example 6.15. Axioms $D_{1,2}$ in Table 6.1 can be derived from the other axioms. The derivation for D_1 is:

$$
\begin{aligned}
r \parallel (p \oplus_i q) &=^{P_1} r \between (p \oplus_i q) + (p \oplus_i q) \between r + r \mid (p \oplus_i q) \\
&=^{P_{4,5},S_{2,3}} (r \between p \oplus_i r \between q) + (p \between r \oplus_i q \between r) + (r \mid p \oplus_i r \mid q) \\
&=^{A_{5,6}} (((r \between p + p \between r + r \mid p) \oplus_i (r \between p + p \between r + r \mid q)) \oplus_i \\
&\quad ((r \between p + q \between r + r \mid p) \oplus_i (r \between p + q \between r + r \mid q))) \oplus_i \\
&\quad (((r \between q + p \between r + r \mid p) \oplus_i (r \between q + p \between r + r \mid q)) \oplus_i \\
&\quad ((r \between q + q \between r + r \mid p) \oplus_i (r \between q + q \between r + r \mid q))) \\
&=^{N_{1,3,5}} (r \between p + p \between r + r \mid q) \oplus_i (r \between q + q \between r + r \mid q) \\
&=^{P_1} (r \parallel p) \oplus_i (r \parallel q).
\end{aligned}
$$

In [89], a set of algebraic laws was provided including A_4, A_6, D_1, and Dri. Their algebraic laws are sensitive to the numbering of variants and the placement of their operands. Therefore, the laws do not support idempotence and commutativity properties of the binary variant (axioms $A_{1,3}$). By prohibiting application of axioms $A_{1,2}$ and N_6 while removing variants with an identical index with the help of axioms N_{1-5}, we can rewrite a term into a fully expanded one. Subsequently, with the help of axioms $A_{1,2}$ and N_6, our axiomatization becomes insensitive to the placement of operands in binary variants or binary variant indices.

6.3.3 Completeness of the Axiomatization for Finite-state Behaviors

We prove that the axiomatization in Table 6.1 and 6.2 is ground-complete for PL-CCS terms with finite-state models modulo product line bisimilarity. Following the approach of [9], to restrict to PL-CCS terms with finite-state transition systems, we provide a syntactical restriction for constants $\langle A|E\rangle$. We consider so-called *finite-state PL-CCS*, denoted by PL-CCS$_f$, which is obtained by extending PL-CCS with essentially finite-state recursive specifications: a recursive specification E is essentially finite-state, if it has only finitely many equations and in the right-hand sides of all equations of E, no process name occurs in the scope of static operators, namely, parallel composition, left- and communication merge, restriction, and renaming operators.

For instance, PL-CCS term $\langle Y|\{Y \stackrel{def}{=} (a.0 \oplus_1 b.0) \parallel c.Y\}\rangle$ is not a finite-state PL-CCS process. To see this, observe that the sequence of transitions $\langle Y|\{Y \stackrel{def}{=} (a.0 \oplus_1 b.0) \parallel c.Y\}\rangle \xrightarrow{c,\langle ?\rangle} (a.0 \oplus_1 b.0) \parallel \langle Y|\{Y \stackrel{def}{=} (a.0 \oplus_1 b.0) \parallel c.Y\}\rangle \xrightarrow{c\langle ?\rangle} (a.0 \oplus_1 b.0) \parallel (a.0 \oplus_1 b.0) \parallel \langle Y|\{Y \stackrel{def}{=} (a.0 \oplus_1 b.0) \parallel c.Y\}\rangle \xrightarrow{c,\langle ?\rangle} \ldots$ can be derived, which leads to an infinite state space. This sequence results from the occurrence of process name Y in the context of a parallel composition.

Proposition 6.16 (Finite-state behaviors)**.** *Consider a PL-CCS$_f$ term t; the transition system for t generated by the SOS rules has only finitely many states.*

This proposition can be proved by resolving the binary variants, which are finite. Therefore, a finite set of CCS terms are derived such that each CCS term generates finitely many states [145].

Theorem 6.17 (Completeness)**.** *The axiomatization, given in Tables 6.1 and 6.2 is ground-complete for the term algebra $\mathbb{IP}$(PL-CCS$_f$)$/\simeq_{PL}$, i.e., for all closed finite-state PL-CCS$_f$ terms t_1 and t_2, $t_1 \simeq_{PL} t_2$ implies $t_1 = t_2$.*

See Section B.4 for the proof.

6.4 Product Line Analysis

The advantage of our sound and complete axiomatization is that we can prove equality of PL-CCS terms at a syntactic level by transforming one term to another. Hence, one does not need to generate the huge state space, which was required to check the notions of bisimilarity introduced in Section 6.2. This process can be facilitated and mechanized with the help of theorem provers, or term rewriting systems. Consequently, terms can be transformed by our axiomatization into a basic form, such as *linear process specifications* in the mCRL2 language [82, 83], over which different analyses can be performed, either manually or using tools. This basic form acts as a symbolic representation of the state space of a model, which is comparatively small. A set of tools such as model checker,

state space visualizer, and behavioral simulator exist that run over this basic representation and can be adapted to our setting. Furthermore, a number of optimization approaches such as τ-confluence reduction [26], that work on the level of this basic format, can simplify it prior to any analysis. The transformation process into a basic form can be mechanized in the same way as [143] within a small amount of time. Similarly, PL-CCS terms can be reduced to their possible products (which are simple CCS terms) in a syntactic way to be validated in terms of their intended properties.

A formal framework for modeling and analyzing SPLs should support modular design, derivation (configuration) of individual systems from a product line model, and restructuring them into various syntactic forms [89]. Our process theory supports them all. In our case, we support a few different forms of restructuring: for example, we support "moving variation points throughout the hierarchical specification of an SPL towards its leaves or its root" [89]. We also support "modeling individual systems using a higher or lower degree of common parts" [89]. Therefore, a designer can model the functionality of an SPL irrespective of the existing components. Later with the aim of reuse, the functionality can be restructured to behaviors for which appropriate components exist. A restructuring mechanism is also appealing when a new functionality (which corresponds to a new feature) is added. We illustrate how our framework supports deriving products or restructuring of SPLs in following sections.

6.4.1 Deriving Products of a Family

Using our axiomatization, one can derive the products of a family, i.e., rewrite a process term into a term which comprises binary variants of CCS terms.

Example 6.18. For instance, consider CCS terms p, q, and s (which naturally do not contain the binary variant operator); the family $(q \oplus_1 (a.(s \oplus_2 p)) \oplus_3 a.s$ can generate three pairwise non-bisimilar products:

$$
\begin{aligned}
&(q \oplus_1 (a.(s \oplus_2 p))) \oplus_3 a.s =^{A_2} \\
&\quad q \oplus_1 ((a.(s \oplus_2 p)) \oplus_3 a.s) =^{A_4} q \oplus_1 ((a.s \oplus_2 a.p) \oplus_3 a.s) =^{A_1} \\
&\quad q \oplus_1 ((a.p \oplus_2 a.s) \oplus_3 a.s) =^{A_2} q \oplus_1 (a.p \oplus_2 (a.s \oplus_3 a.s)) =^{A_3} \\
&\quad q \oplus_1 (a.p \oplus_2 a.s)
\end{aligned}
$$

Example 6.19. We can compare product lines $\langle p_1 | \{p_1 \stackrel{def}{=} p_2 \parallel p_2,\ p_2 \stackrel{def}{=} b.0 \oplus_1 c.0\}\rangle$ and $\langle p_1' | \{p_1' \stackrel{def}{=} b.0 \oplus_1 c.0 \parallel b.0 \oplus_2 c.0\}\rangle$. p_1 generates two non-bisimilar products, while p_1' generates three pairwise non-bisimilar products, concluding $p_1 \neq p_1'$. To see this, observe the following derivations:

$$
\begin{array}{lll}
p_1 & =^{UnFold} & \langle p_2 | \{p_1 \stackrel{def}{=} p_2 \parallel p_2, \\
 & & \quad p_2 \stackrel{def}{=} b.0 \oplus_1 c.0\}\rangle \parallel \langle p_2 | \{p_1 \stackrel{def}{=} p_2 \parallel p_2,\ p_2 \stackrel{def}{=} b.0 \oplus_1 c.0\}\rangle \\
 & =^{UnFold} & (b.0 \oplus_1 c.0) \parallel (b.0 \oplus_1 c.0) \\
 & =^{P_1, C_3} & ((b.0 \oplus_1 c.0) \between (b.0 \oplus_1 c.0)) + ((b.0 \oplus_1 c.0) \mid (b.0 \oplus_1 c.0)) \\
 & =^{P_{4,5}, S_3} & ((b.0 \between b.0 \oplus_1 c.0 \between b.0) \oplus_1 (b.0 \between c.0 \oplus_1 c.0 \between c.0)) + \\
 & & ((b.0 \mid b.0 \oplus_1 c.0 \mid b.0) \oplus_1 (b.0 \mid c.0 \oplus_1 c.0 \mid c.0)) \\
 & =^{N_1, 3, 5} & (b.0 \between b.0) \oplus_1 (c.0 \between c.0) + (b.0 \mid b.0) \oplus_1 (c.0 \mid c.0) \\
 & =^{P_2, S_5, C_{3,4}} & b.(0 \parallel b.0) \oplus_1 c.(0 \parallel c.0) \\
 & =^{P_{1,6}, S_6} & b.b.0 \oplus_1 c.c.0
\end{array}
$$

$$
\begin{array}{lll}
p_1' & =^{UnFold} & (b.0 \oplus_1 c.0) \parallel (b.0 \oplus_2 c.0) \\
 & =^{D_{1,2}} & ((b.0 \parallel b.0) \oplus_2 (b.0 \parallel c.0)) \oplus_1 ((c.0 \parallel b.0) \oplus_2 (c.0 \parallel c.0)) \\
 & =^{P_{1,6}, S_6} & (b.b.0 \oplus_2 (b.c.0 + c.b.0)) \oplus_1 ((c.b.0 + b.c.0) \oplus_2 c.c.0) \\
 & =^{A_{1-3}, C_1} & b.b.0 \oplus_1 ((b.c.0 + c.b.0) \oplus_2 c.c.0)
\end{array}
$$

Next, we show that every PL-CCS term can be rewritten into a normal form comprising binary choices over CCS products.

Theorem 6.20. *Let $\bigoplus_{i \leq n} p_i$ denote $(p_0 \oplus_1 (p_1 \oplus_2 (\ldots \oplus_n p_n) \ldots))$ if $n > 0$, and p_0 if $n = 0$. Using the axiomatization in Tables 6.1 and 6.2, each PL-CCS term t can be rewritten into the form $\bigoplus_{i \leq n} p_i$, where the p_is are CCS processes.*

See Section B.2 for the proof. With the help of axioms A_{4-6}, $D_{1,2}$, $P_{4,5}$, $S_{1,3}$, R_3, E_5, and Dri, one can convert a PL-CCS term into another term such that its binary variants do not occur in scope of any CCS operators. Later by axiom N_{1-6}, the indices of variants become unique. Therefore, by $A_{1,2}$, it can be rewritten into the desired format.

6.4.2 Restructuring a Product Family

With the help of our axiomatization, we can factorize the common parts and simplify the structure of product line terms. We can also identify the mandatory parts of a product line; the parts that exist in any product.

Example 6.21. Consider a $Sensor$ process that is replicated in different parts of a car windscreen $WindScreen$, such as wiper $WipFam$ and fog remover $FogFam$ [90]:

$$
\begin{array}{l}
WindScreen \stackrel{def}{=} WipFam \oplus_1 FogFam \\
WipFam \stackrel{def}{=} Sensor \parallel Wiper \\
FogFam \stackrel{def}{=} Sensor \parallel FogRem
\end{array}
$$

where $Sensor$ detects the different conditions of precipitation, $Wiper$ and $FogRem$ offer different operational modes for wiper arm movement, and windscreen

warmer concerning environmental conditions, respectively. Using our axioms, $WindScreen$ specification is restructured as follows:

$$\begin{aligned} WindScreen &\stackrel{def}{=} WipFam \oplus_1 FogFam \\ &=^{UnFold} (Sensor \parallel Wiper) \oplus_1 (Sensor \parallel FogRem) \\ &=^{D_1} Sensor \parallel (Wiper \oplus_1 FogRem) \end{aligned}$$

The new specification for $WindScreen$ reveals that $Sensor$ is the mandatory part of our windscreen family. The structural specification (i.e., the architecture) of $WindScreen$ consists of two components, a $Sensor$ and a component, which can be either $Wiper$ or $FogRem$.

Assume that the $Sensor$ has two qualities, namely, high and low. The low quality sensor cannot distinguish between *heavy* and *little* rain (specified by hvy and ltl actions) and can only discriminate between *no rain* and *rain* [90]. The high quality sensor can, however, make such distinctions as specified below:

$$Sensor \stackrel{def}{=} SensL \oplus_2 SensH$$

$$\begin{aligned} SensL &\stackrel{def}{=} non.SensL + ltl.Raining + hvy.Raining + \overline{noRain}.SensL \\ Raining &\stackrel{def}{=} non.SensL + ltl.Raining + hvy.Raining + \overline{rain}.Raining \end{aligned}$$

$$\begin{aligned} SensH &\stackrel{def}{=} non.SensH + ltl.Medium + hvy.Heavy + \overline{noRain}.SensH \\ Medium &\stackrel{def}{=} non.SensH + ltl.Medium + hvy.Heavy + \overline{rain}.Medium \\ Heavy &\stackrel{def}{=} non.SensH + ltl.Medium + hvy.Heavy + \overline{hvyRain}.Heavy \end{aligned}$$

The specification of $Sensor$ can be similarly examined to reveal the common behaviors. To this aim, we first restructure $SensL \oplus_2 SensH$ to factor out the common behaviors:

$$\begin{aligned} &SensL \oplus_2 SensH =^{UnFold} \\ &\quad (non.SensL + ltl.Raining + hvy.Raining + \overline{noRain}.SensL) \oplus_2 \\ &\quad (non.SensH + ltl.Medium + hvy.Heavy + \overline{noRain}.SensL) =^{N_{1,3,5}} \\ &\quad ((non.SensL + ltl.Raining + hvy.Raining + \overline{noRain}.SensL) \oplus_2 \\ &\qquad (non.SensH + ltl.Raining + hvy.Raining + \overline{noRain}.SensL)) \oplus_2 \\ &\quad ((non.SensL + ltl.Medium + hvy.Heavy + \overline{noRain}.SensH) \oplus_2 \\ &\qquad (non.SensH + ltl.Medium + hvy.Heavy + \overline{noRain}.SensH)) =^{A_{4,5}} \\ &\quad (non.(SensL \oplus_2 SensH) + ltl.Raining + hvy.Raining + \overline{noRain}.SensL) \oplus_2 \\ &\quad (non.(SensL \oplus_2 SensH) + ltl.Medium + hvy.Heavy + \overline{noRain}.SensH) =^{UnFold} \\ &\quad (non.Sensor + ltl.Raining + hvy.Raining + \overline{noRain}.SensL) \oplus_2 \\ &\quad (non.Sensor + ltl.Medium + hvy.Heavy + \overline{noRain}.SensH) =^{N_{1,3,5},A_{4,5}} \\ &\quad non.Sensor + ltl.(Raining \oplus_2 Medium) + hvy.(Raining \oplus_2 Heavy) + \\ &\qquad \overline{noRain}.Sensor \end{aligned}$$

Similarly $Raining \oplus_2 Medium$ and $Raining \oplus_2 Heavy$ can be examined:

$$Raining \oplus_2 Medium =^{UnFold, N_{1,3,5}, A_{4,5}} non.(SensL \oplus_2 SensH) + ltl.(Raining \oplus_2 Medium) + hvy.(Raining \oplus_2 Heavy) + \overline{rain}.(Raining \oplus_2 Medium)$$

$$Raining \oplus_2 Heavy =^{UnFold, N_{1,3,5}, A_{4,5}} non.(SensL \oplus_2 SensH) + ltl.(Raining \oplus_2 Medium) + hvy.(Raining \oplus_2 Heavy) + (\overline{rain}.(Raining \oplus_2 Heavy) \oplus_2 \overline{hvyRain}.(Raining \oplus_2 Heavy))$$

By applying $Fold$, the new specification of $Sensor$ is obtained as follows:

$$\begin{aligned} Sensor &\stackrel{def}{=} non.Sensor + ltl.RainMed + hvy.RainHvy + \overline{noRain}.Sensor \\ RainMed &\stackrel{def}{=} non.Sensor + ltl.RainMed + hvy.RainHvy + \overline{rain}.RainMed \\ RainHvy &\stackrel{def}{=} non.Sensor + ltl.RainMed + hvy.RainHvy + \\ &\qquad (\overline{rain}.RainHvy \oplus_2 \overline{hvyRain}.RainHvy) \end{aligned}$$

The new specification explains that resolving variability between $SensL$ and $SensH$ can be postponed until the variability between performing output actions $\overline{rain}$ and $\overline{hvyRain}$ is resolved in case it is possible to have heavy rain.

In [37], PL-CCS was compared with FTS in addressing variability via modeling the above-mentioned example. There, it was concluded that modeling in PL-CCS can result in verbose descriptions since common parts have to be duplicated [39]. However, in our experience, PL-CCS facilitates modular design without forcing the designer to factor out common parts. Later the specification can be restructured as illustrated by the above-given example. For instance, actions non, ltl, hvy, and $\overline{noRain}$ are common among $SensL$ and $SensR$, while their behaviors does not change after performing actions non and $\overline{noRain}$. Such common actions and behaviors are factored our by rewriting $SensL \oplus_2 SensR$ to $non.Sensor + ltl.(Raining \oplus_2 Medium) + hvy.(Raining \oplus_2 Heavy) + \overline{noRain}.Sensor$. Therefore, the modeler is not forced to identify common actions and behaviors to derive its model. The semantics of our resulting specification is even more compact than the FTS model of [37] by factorizing out common behaviors as much as possible; the part of behavior in which both sensors does not change their behaviors as long as action hvy is performed, is factored out in our case.

6.5 Logical Characterization

In this section, we show that product line bisimilarity induces the same identification of PL-CCS terms as the multi-valued modal μ-calculus. In this section, we first review the logic and then explain how it characterizes product line bisimilarity.

6.5.1 Multi-Valued Modal μ-Calculus

The multi-valued modal μ-calculus [90] combines Kozen's modal μ-calculus [105] and multi-valued μ-calculus as defined by Grumberg and Shoham [136]. Let $\mathcal{V}$ be the set of propositional variables. The set of multi-valued modal μ-calculus formulae is given by the following grammar:

$$\varphi \ ::= \ true \mid false \mid Z \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \eta Z.\varphi, \ \eta \in \{\nu, \mu\}$$

where $a \in Act$ and $Z \in \mathcal{V}$, and the fixed point quantifiers μ and ν are variable binders.

Let $mv - \mathfrak{L}_\mu$ denote the set of closed formulae generated by the above grammar. The semantics of a formula for a PL-CCS term is the set of configurations satisfying it. All configurations satisfy $true$ in all states. A configuration ν' satisfies a formula $\phi_1 \vee \phi_2$ in state s if it satisfies either ϕ_1 or ϕ_2 in state s. A configuration ν' satisfies a formula $\langle a \rangle \varphi$ in state s if it has a transition $s \xrightarrow{a,\nu} s'$ such that $\nu \sqsubseteq \nu'$ and ν' satisfies φ in state s'. A configuration ν' satisfies a formula $[a]\varphi$ in state s if for all transitions $s \xrightarrow{a,\nu} s'$ such that $\nu \sqsubseteq \nu'$, ν' satisfies φ in state s'. Equations with recursive variables are used to describe properties of behaviors with an infinite depth. For instance, $X \stackrel{def}{=} \langle a \rangle X \vee \langle b \rangle true$ specifies configurations (i.e., products) that satisfy $\langle b \rangle true$ either in the initial state, or the state reached after performing a sequence of a-actions (with consistent configurations). Since an equation may have many solutions, the maximum and minimum solutions are selected by $\nu Z.\phi$ and $\mu Z.\phi$, respectively. Considering Z as a mapping from the states to a set of configurations, $\mu Z.\phi$ is valid for the smallest mapping Z that satisfies the equation $Z = \phi$. Similarly $\nu Z.\phi$ is valid for the largest mapping Z that satisfies equation $Z = \phi$. For instance, the property $\mu X.\langle a \rangle X \vee \langle b \rangle true$ specifies that "eventually an action b follows a (possibly empty) sequence of a actions". This property holds for the configuration $\langle L, L \rangle$ in the state $\langle X|E \rangle$ of CTS in Fig. 6.3a.

The semantics of φ, denoted by $[\![\varphi]\!]$, is a function $\mathcal{S} \to I\!P(Config)$ that defines the set of configurations that satisfy formula φ for each given state. Given an environment $\rho : \mathcal{V} \to (\mathcal{S} \to I\!P(Config))$, which maps free variables in φ to $\mathcal{S} \to I\!P(Config)$, $[\![\varphi]\!]_\rho$ defines the semantics of φ with respect to ρ in Table 6.3. Let $\mathcal{R}_a(s, s') = \{\nu \mid s \xrightarrow{\nu', a} s', \nu' \sqsubseteq \nu\}$ denote the set of configurations for which s has a transition to s' labeled by a.

Example 6.22. Regarding the rules in Table 6.3, the semantics of $[\![\langle b \rangle true]\!]$ in the state $(b.\langle X|E \rangle \oplus_1 c)$ of the CTS in Fig. 6.3a is $\{\langle L, R \rangle, \langle L, L \rangle\}$, since $\mathcal{R}_b(b.\langle X|E \rangle \oplus_1 c.0, \langle X|E \rangle) = \{\langle L, R \rangle, \langle L, L \rangle\}$ and $[\![true]\!](\langle X|E \rangle) = Config$. It can be shown that $f = \{(b.\langle X|E \rangle \oplus_1 c.0) \mapsto \{\langle L, R \rangle, \langle L, L \rangle\}, \langle X|E \rangle \mapsto \{\langle L, L \rangle\}, 0 \mapsto \emptyset\}$ is the semantics of $\mu X.\langle a \rangle X \vee \langle b \rangle true$ since $[\![\langle a \rangle X]\!]_{[X \mapsto f]} = \lambda s. \bigcup \{\mathcal{R}_a(s, s') \cap X(s')\} = \{\langle X|E \rangle \mapsto \{\langle L, L \rangle\}, b.\langle X|E \rangle \oplus_1 c \mapsto \emptyset, 0 \mapsto \emptyset\}$, $[\![\langle b \rangle true]\!] = \{b.\langle X|E \rangle \oplus_1 c \mapsto \{\langle L, R \rangle, \langle L, L \rangle\}, \langle X|E \rangle \mapsto \emptyset, 0 \mapsto \emptyset\}$, $f = [\![\langle a \rangle X \vee \langle b \rangle true]\!]_{[X \mapsto f]}$, and it is the minimum mapping w.r.t. inclusion that satisfies the equation $X = \langle a \rangle X \vee \langle b \rangle true$.

Table 6.3: Semantics of multi-valued modal μ-calculus [90]

$[\![true]\!]_\rho = \lambda s.Config$	$[\![\langle a\rangle\varphi]\!]_\rho = \lambda s.\bigcup_{s'\in S}\mathcal{R}_a(s,s')\cap[\![\varphi]\!]_\rho(s')$
$[\![false]\!]_\rho = \lambda s.\emptyset$	$[\![[a]\varphi]\!]_\rho = \lambda s.\bigcap_{s'\in S}\neg\mathcal{R}_a(s,s')\cup[\![\varphi]\!]_\rho(s')$
$[\![Z]\!]_\rho = \rho(Z)$	$[\![\mu Z.\varphi]\!]_\rho = \bigcap\{f \mid [\![\varphi]\!]_{\rho[Z\mapsto f]}\subseteq f\}$
$[\![\varphi_1\wedge\varphi_2]\!]_\rho = [\![\varphi_1]\!]_\rho\cap[\![\varphi_2]\!]_\rho$	$[\![\nu Z.\varphi]\!]_\rho = \bigcup\{f \mid f\subseteq[\![\varphi]\!]_{\rho[Z\mapsto f]}\}$
$[\![\varphi_1\vee\varphi_2]\!]_\rho = [\![\varphi_1]\!]_\rho\cup[\![\varphi_2]\!]_\rho$	

6.5.2 Relation to Product Line Bisimilarity

Model checking logical formula ϕ over a PL-CCS term is supposed to result in the set of full configurations for which the property holds. Intuitively, two PL-CCS terms are logically equivalent when for each logical formula, there exists a non-empty set of products in one product line satisfying it if and only if there exists such a non-empty set in the other. For instance, $(a.0 + b.0) \oplus_1 b.0$ is not logically equivalent to $a.0 \oplus_1 b.0$ as the logical formula $\langle a\rangle true \wedge \langle b\rangle true$ is satisfied by the former for the configuration $\langle L\rangle$, but it is not satisfied by the latter for any configuration.

Definition 6.23 (Logical equivalence)**.** Two PL-CCS terms s and t are *logically equivalent*, denoted by $s \sim_L t$, iff $\forall\varphi \in mv-\mathfrak{L}_\mu \cdot ([\![\varphi]\!](s) \neq \emptyset \Leftrightarrow [\![\varphi]\!](t) \neq \emptyset)$.

As stated before, product line bisimilarity and logical equivalence coincide. We restrict to PL-CCS terms with finite-state behaviors, following the approach of [49, 92]. However, Theorem 6.24 can be generalized in the same vein as [118] by resorting to infinitary logics. The following theorem states that if two PL-CCS$_f$ terms are not product line bisimilar, then there is a logical formula that can distinguish them.

Theorem 6.24. *For any PL-CCS$_f$ terms s and t, $s \simeq_{PL} t$ iff $s \sim_L t$.*

See Section B.5 for the proof.

6.6 Related Work

There are a vast number of languages to specify software product lines for the purpose of specifying different aspects of variability [1, 7, 15, 38, 39, 44, 60, 65, 77, 89, 90, 94, 106, 108, 110, 141, 157]. Among them some frameworks, such as [1, 44, 60, 157], do not directly address formal reasoning. On the other hand, there are formal frameworks to reason about some aspects of SPLs, such as [7, 15, 38, 39, 65, 77, 89, 90, 94, 106, 108, 110, 141].

Regarding modeling issues, several approaches are classified by [39] in terms of treating variability as either a first class citizen [15, 44, 77, 94, 108, 110, 141] or as part of the behavioral model [1, 60, 65, 90, 106, 157]. In the former approaches, variability is separately modeled and related to other models (data

and behavior), called base models. Therefore, variability is explicitly traceable in base models and its evolution is automatically propagated to the base models. As opposed to other process-algebraic approaches [15, 77, 141], PL-CCS expresses variability as part of its behavioral model; In [15, 77], the cross-tree constraints of feature models are related to the behavior of products using a CCS-like process algebra, while in [141], process terms are tagged with the sets of specific products where they are enabled using a CSP-like process algebra. In [20], some of the fundamental formal behavioral models for SPLs are compared in terms of their expressiveness.

Our approach follows the line of research on process algebra for SPLs [14, 15, 77, 89, 90, 141]. It also offers several reasoning capabilities: model-checking based on a multi-valued modal μ-calculus over CTSs given in [90], and equational reasoning based on a set of rules to restructure PL-CCS terms, extending the approach of [89]. Along these lines, the safety and liveness properties of SPLs as well as consistency of configurations are checked in [15, 77] by encoding their semantic rules in the Maude rewriting logic. In [14], the algebraic framework mCRL2 [82, 83], is used for modular verification of SPLs. There, tailored property-preserving reductions were applied to a product line modeled in mCRL2 using the reduction modulo branching bisimulation of mCRL2 tool set. Two pre-congruence behavioral relations to compare the behavior of a product either against a product or a family are proposed in [141]. In contrast, we offer a set of behavioral equivalence relations over CTSs supported by a sound and complete axiomatization to reason about families at the syntactic level. To this aim, a family can be restructured to another, while its functionality is preserved. Our approach is hence complementary to the aforementioned approaches in the literature.

The work of [109] is analogous to our in that it takes an axiomatic approach to software product lines. However, their approach has a different intention, namely, to axiomatize product family concepts that characterize a generic product line formalism. Their approach defines that all operators of a formalism are distributive over the binary variant operator. Our axioms $D_{1,2}$, $A_{5,6}$, $P_{4,5}$, S_3 (together with S_1), R_3, and E_5 conform to this result. Furthermore, it expresses that binary variant operators with different indices are distributive and provides rules to simplify specifications when two i-indexed variants are directly nested. These rules are derivable in our setting by axioms A_3 and N_{1-6} as follows:

$$
\begin{aligned}
&(P \oplus_j Q) \oplus_i (P \oplus_j O) =^{A_3} \\
&\quad ((P \oplus_j Q) \oplus_i (P \oplus_j O)) \oplus_j ((P \oplus_j Q) \oplus_i (P \oplus_j O)) =^{N_5} \\
&\quad L((P \oplus_j Q) \oplus_i (P \oplus_j O), j) \oplus_j R((P \oplus_j Q) \oplus_i (P \oplus_j O), j) =^{N_{1-4}} \\
&\quad (L(P, j) \oplus_i L(P, j)) \oplus_j (R(Q, j) \oplus_i R(O, j)) =^{N_2, N_4} \\
&\quad L(P \oplus_i P, j) \oplus_j R(Q \oplus_i O, j) =^{A_3, N_5} P \oplus_j (Q \oplus_i O) \\
&P \oplus_i (Q \oplus_i O) =^{N_5} L(P, i) \oplus_i R(Q \oplus_i O, i) =^{N_4} \\
&\quad L(P, i) \oplus_i R(O, i) =^{N_5} P \oplus_i O
\end{aligned}
$$

Similarly, $(P \oplus_j O) \oplus_i (Q \oplus_j O) = (P \oplus_j Q) \oplus_i O$ and $(P \oplus_i Q) \oplus_i O = P \oplus_i O$ can be derived from our equational theory.

Other techniques related to our behavioral equivalence are conformance notions that are used to iteratively refine partial behavioral models; they can consequently be used to relate a product behavior to a family model. Refinement as well as conformance between SPLs modeled by modal I/O automata is studied in [106]. A notion of behavioral conformance on MTS-based specifications is defined in [65], which preserves 3-valued weak μ-calculus. Modal transition systems (MTS) [65] and modal I/O automata [106] capture variability by defining transitions as optional and mandatory. A notion of input-output conformance on FTSs is defined in [18, 19] with the aim of devising a model-based testing trajectory for SPLs. In [41], pre-orders over FTSs preserving LTL properties are given with respect to specific products. As opposed to these approaches, our equivalence relation is defined over CTS and preserves multi-valued modal μ-calculus. Providing a comprehensive and formal comparison of the different notions of SPL pre-orders in the literature is among our future work. In [16], a feature-oriented notion of branching bisimulation over FTSs and its associated minimization algorithm were introduced in order to reduce a model prior to its verification. Our configuration bisimulation was inspired by [16] and adopting its minimization algorithm is also among our future directions for research.

Other approaches mainly use model checking to reason about SPLs; these include checking safety properties over Statecharts in [110], LTL over FTSs [39], CTL over modal I/O automata [108], MHML, a deontic logic interpreted over MTSs [7], and fLTL (an extension of LTL) over FTSs in [38].

6.7 Conclusions and Future Work

We proposed an equational reasoning technique to reason about software product lines at the syntactic level. To this aim, we defined product line bisimilarity by finding a mapping between products of two terms, identified by their configuration vectors. We also introduced a configuration-oriented bisimilarity that compares families at once, based on the idea of partitioning configuration lists inspired by our notion of branching reliable computed network bisimilarity. We proved that product line and configuration bisimilarity coincide. To facilitate checking the bisimilarity relations, we provided a sound and complete axiomatization over closed and finite-state behaviors. We characterized the distinguishing power of our equivalence relations in terms of a multi-valued modal μ-calculus.

Instead of working at the semantic level and finding a mapping between products, one can use our axioms and restructure a term to its equivalent terms, e.g., such that the mandatory and optional parts are factored out separately. The restructuring mechanism can also be initiated to group the functionality of an SPL to behaviors for which appropriate components exist. Furthermore, one can derive the possible products of a term, specified by CCS terms to validate an SPL

model in terms of its various products.

We intend to exploit the PL-CCS process theory as a formal framework for specifying the structural and behavioral aspects of product lines, following the approach of [5]. We intend to investigate a basic form, such as linear process specification in mCRL2, over which different analysis and optimizations can be executed. Then, PL-CCS terms can be automatically transformed into the basic form in the same way of [143]. Finding a minimization algorithm for configuration bisimulation is another line of research. Furthermore, pre-order notions of literature can be compared in the general setting of FTS, which is a very expressive model for SPLs [20].

Concluding Remarks

7

We review the results of this thesis and the objectives that have been achieved to overcome the problems and the challenges as explained in Sections 1.1 and 1.3, respectively. Furthermore, we provide possible future directions of this research.

7.1 Results

We have classified our results from three viewpoints: the basic semantic model for MANETs, formal modeling frameworks for MANETs and results related to each specific framework, and formal analysis of MANETs taking mobility into account.

To address the problem of abstract formal modeling of the topology and its changes to minimize the problem of state space explosion and to make its formal analysis via the model checking technique feasible, the following results have been obtained:

- We introduced network constraints with positive and negative pairs as the symbolic representation of a set of topologies. These constraints are exploited to annotate the transitions of the classical LTSs to compactly address arbitrary mobility of nodes at the semantics. We formally defined how a CLTS is unfolded into an LTS whose states maintain the underlying topology. We showed in Section 3.6 through a case study on a leader election protocol for MANETs that CLTSs prevent the growth of state spaces with a factor of 2^{n^2}, where n is the number of nodes, as opposed to the classical LTSs.

- We gave a novel branching-time temporal logic over CLTSs, as an extension of action-based CTL [48], to specify the topology-dependent properties of MANETs. In this logic, the path quantifier *All* is parametrized with a topology formula that enforces the pre-condition on establishment of the property. The topology formula expresses multi-hop constraints over the topology. We have also defined an efficient model checking algorithm for this logic. Therefore, by specifying an appropriate property one can examine the correct behavior of a protocol for different topological patterns

(without changing the model), which is infeasible through classical model checking approaches.

Therefore, the objective of providing a suitable semantic model addressing the arbitrary mobility of nodes which supports efficient analysis of protocols through model checking has been achieved.

To formally model the protocols above the data link layer, two directions were followed, with a focus on the problem of minimizing the state space explosion specially for formal models of small networks.

- We provided an algebraic framework whose terms derive CLTSs. Thus the problem of state space explosion due to topology and its arbitrary changes is delegated to the semantic model. The behavior of wireless communication with the required properties is guaranteed by the careful computation of network constraints by the SOS rules. The framework was furnished with a sound and complete axiomatization modulo rooted branching reliable computed network bisimilarity. Therefore, it supports necessary means to automatically derive the linear format of a specification following the approach of [143] for symbolic analysis of MANETs, namely the cones and foci method [67, 68, 86] and parametrized boolean equation systems [88]. The former allows to prove that a network composed of a finite but unbounded numbers of uniform nodes is behaviorally equivalent to a specification, while the latter enables to prove a property for a specification with infinite data domains.

- A modeling approach based on the actor computation model was followed which comes up with a high level of abstraction for modelers, and hence, it can be used in a larger community thanks to its simple syntax inherited from Rebeca [139]. However, it does not support compositional modeling of networks as opposed to the algebraic framework. Furthermore, it prohibits a direct generation of the compact CLTS models (to address the issues of topology). Semantic states contain an abstract model of the underlying topology by which the behavior of wireless communication was defined. To minimize the problem of state space explosion and hence, to get the assistance of existing model checking tools, we proposed a reduction technique, based on the counter abstraction technique, whose application is restricted to static networks. However, CLTSs are generated through a two-step state generation tool by first exploring next states and then merging those which only differ in their underlying topology.

To formally analyze MANETs to find mobility scenarios leading to an erroneous behavior, the following results were achieved:

- We extended the model checking technique over CLTSs and demonstrated how the correct behavior of a protocol can be inspected regarding different topological patterns.

- To investigate the topology-dependent properties of MANETs by equational reasoning, we provided an equivalence relation for the reliable setting supported with a sound and complete axiomatization. Furthermore, we provided an analysis approach at the syntactic level, exploiting a precongruence relation and our axiomatization.
- We extended the counter abstraction technique with the idea of partitioning nodes in terms of their topological positions to reduce the classical semantic model of MANETs when the underlying topology constitutes a part of a semantic state.
- We introduced a configuration-oriented bisimilarity that compares product families at once, inspired by our notion of branching reliable computed network bisimilarity, which induces the same identification of PL-CCS terms as the multi-valued modal μ-calculus of [90]. We also provided a sound and complete axiomatization over closed PL-CCS terms with finite-state behaviors.

7.2 Future Work

The following extensions are proposed for this research:

- Extending the algebraic framework with parametrized boolean equations in the same way as [88]: We can symbolically reason about the properties of MANET models with infinite data domains specified by linear formats without any need to generate the state spaces.
- Application of wRebeca to the newer versions of AODV: We can learn experiences by inspecting different versions which are helpful for the protocol design. Our ultimate goal could be to show how formal tools are beneficial for a protocol design.
- Extending wRebeca with timing features following the approach of [102]: We remark that our approach of modeling the topology and timing concepts are orthogonal, and thus it can be easily extended to reason about the performance issues of protocols.
- Extending the model checking algorithm for topology formula with negative topology formula, e.g., $\ell \not\rightarrow \ell'$: To this aim, we should define when a path is valid for a multi-hop constraint $\mathcal{M}$. Instead of finding a subgraph for a positive topology formula, we should find a super-graph which does not satisfy a negative topology formula.
- Generating a counterexample during model checking: To assist the process of debugging, it is essential to improve our model checking algorithm to generate a counterexample. The role of network constraints during the model checking make the generation of counterexamples challenging.

Bibliography

[1] W. van der Aalst, M. Dumas, F. Gottschalk, A. ter Hofstede, M. Rosa, and J. Mendling. Correctness-preserving configuration of business process models. In *Proc. 11th Conference on Fundamental Approaches to Software Engineering*, volume 4961 of *LNCS*, pages 46–61. Springer, 2008.

[2] P. Aziz Abdulla, G. Delzanno, O. Rezine, A. Sangnier, and R. Traverso. On the verification of timed ad hoc networks. In *Proc. 9th Conference on Formal Modeling and Analysis of Timed Systems*, volume 6919 of *LNCS*, pages 256–270. Springer, 2011.

[3] L. Aceto, A. Ingólfsdóttir, K. Larsen, and J. Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, 2007.

[4] G. Agha. *ACTORS - A Model of Concurrent Computation in Distributed Systems*. MIT Press, 1990.

[5] A. Aldini, M. Bernardo, and F. Corradini. *A Process Algebraic Approach to Software Architecture Design*. Springer, 2010.

[6] H. R. Andersen, C. Stirling, and G. Winskel. A compositional proof system for the modal mu-calculus. In *Proc. 9th Annual Symposium on Logic in Computer Science*, pages 144–153. IEEE, 1994.

[7] P. Asirelli, M. ter Beek, A. Fantechi, and S. Gnesi. A logical framework to deal with variability. In *Proc. 8th Conference on Integrated Formal Methods*, volume 6396 of *LNCS*, pages 43–58, 2010.

[8] J. Baeten, T. Basten, and M.A. Reniers. *Process Algebra: Equational Theories of Communicating Processes*. Cambridge University Press, 2010.

[9] J. Baeten and M. Bravetti. A ground-complete axiomatization of finite state processes in process algebra. In *Proc. 16th Conference on Concurrency Theory*, volume 3653 of *LNCS*, pages 248–262. Springer, 2005.

[10] R. Bakhshi, L. Cloth, W. J. Fokkink, and B. R. Haverkort. Mean-field analysis for the evaluation of gossip protocols. In *Proc. on Quantitative Evaluation of SysTems*, pages 247–256. IEEE, 2009.

[11] G. Basler, M. Mazzucchi, T. Wahl, and D. Kroening. Symbolic counter aabstraction for concurrent software. In *Proc. 21st Conference on Computer Aided Verification*, pages 64–78. Springer, 2009.

[12] T. Basten. Branching bisimilarity is an equivalence indeed! *Information Processing Letters*, 58(3):141–147, 1996.

[13] R. J. Baxter. *Exactly Solved Models in Statistical Mechanics*. Academic Press, 1982.

[14] M. ter Beek and E. P. de Vink. Using mCRL2 for the analysis of software product lines. In *Proc. 2nd Workshop on Formal Methods in Software Engineering*, pages 31–37. ACM, 2014.

[15] M. ter Beek, A. Lluch-Lafuente, and M. Petrocchi. Combining declarative and procedural views in the specification and analysis of product families. In *Proc. 17th Software Product Line Conference*, pages 10–17. ACM, 2013.

[16] T. Belder, M. H. ter Beek, and E. P. de Vink. Coherent branching feature bisimulation. In *Proc. 6th Workshop on Formal Methods and Analysis in SPL Engineering*, volume 182 of *EPTCS*, pages 14–30, 2015.

[17] J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *Proc. 24th Annual IEEE Symposium on Logic in Computer Science*, pages 39–48. IEEE, 2009.

[18] H. Beohar and M. R. Mousavi. Spinal test suites for software product lines. In *Proc. 9th Workshop on Model-Based Testing*, volume 141 of *EPTCS*, pages 44–55, 2014.

[19] H. Beohar and M.R. Mousavi. Input-output conformance testing based on featured transition systems. In *Proc. Symposium on Applied Computing*, pages 1272–1278. ACM, 2014.

[20] H. Beohar, M. Varshosa, and M.R. Mousavi. Basic behavioral models for software product lines: Expressiveness and testing pre-orders. *Science of Computer Programming*, 2015.

[21] J. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984.

[22] J. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:21–77, 1985.

[23] D. P. Bertsekas and R. T. G. Gallager. *Data Networks*. Prentice Hall, 1992.

[24] M. Bezem and J. F. Groote. Invariants in process algebra with data. In *Proc. 5th Conference of Concurrency Theory*, volume 836 of *LNCS*, pages 401–416. Springer, 1994.

[25] K. Bhargavan, D. Obradovid, and C. A. Gunter. Formal verification of standards for distance vector routing protocols. *Journal of the ACM*, 49(4):538–576, 2002.

[26] S. Blom and J. van de Pol. State space reduction by proving confluence. In *Proc. 14th Conference on Computer Aided Verification*, volume 2404 of *LNCS*, pages 596–609. Springer, 2002.

[27] J. Borgström, S. Huang, M. Johansson, P. Raabjerg, B. Victor, J. Åman Pohjola, and J. Parrow. Broadcast psi-calculi with an application to wireless protocols. *Software and System Modeling*, 14(1):201–216, 2015.

[28] T. Bourke, R. J. van Glabbeek, and P. Höfner. A mechanized proof of loop freedom of the (untimed) AODV routing protocol. In *Proc. 12th Symposium on Automated Technology for Verification and Analysis*, volume 8837 of *LNCS*, pages 47–63. Springer, 2014.

[29] T. Bourke, R. J. van Glabbeek, and P. Höfner. Showing invariance compositionally for a process algebra for network protocols. In *Proc. 5th Conference on Interactive Theorem Proving*, volume 8558 of *LNCS*, pages 144–159. Springer, 2014.

[30] E. Bres, R. J. van Glabbeek, and P. Höfner. A timed process algebra for wireless networks with an application in routing (extended abstract). In *Proc. 25th European Symposium on Programming*, volume 9632 of *LNCS*, pages 95–122. Springer, 2016.

[31] E. J. H. Chang. Echo algorithms: Depth parallel operations on general graphs. *IEEE Transactions on Software Engineering*, 8(4):391–401, 1982.

[32] S. Chiyangwa and M. Kwiatkowska. A timing analysis of AODV. In *Proc. 7th IFIP Conference on Formal Methods for Open Object-based Distributed Systems*, volume 3535 of *LNCS*, pages 306–321. Springer, 2005.

[33] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121(2):143–148, 1995.

[34] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proc. Workshop on Logic of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1981.

[35] E. M. Clarke, E. A. Emerson, S. Jha, and A. Prasad Sistla. Symmetry reductions in model checking. In *Proc. 10th Conference on Computer Aided Verification*, volume 1427 of *LNCS*, pages 147–158. Springer, 1998.

[36] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2001.

[37] A. Classen. Modelling with FTS: a collection of illustrative examples. Technical Report P-CS-TR SPLMC-00000001, PReCISE Research Center, University of Namur, 2010.

[38] A. Classen, M. Cordy, P. Schobbens, P. Heymans, A. Legay, and J. F. Raskin. Featured transition systems: foundations for verifying variability-intensive systems and their application to LTL model checking. *IEEE Transactions on Software Engineering*, 39(8):1069–1089, 2013.

[39] A. Classen, P. Heymans, P. Schobbens, A. Legay, and J. F. Raskin. Model checking lots of systems: efficient verification of temporal properties in software product lines. In *Proc. 32nd Conference on Software Engineering*, pages 335–344. ACM, 2010.

[40] P. Clements and L. Northrop. *Software product lines: Practices and Patterns*. Addison-Wesley, 2001.

[41] M. Cordy, A. Classen, G. Perrouin, P. Schobbens, P. Heymans, and A. Legay. Simulation-based abstractions for software product-line model checking. In *Proc. 34th Conference on Software Engineering*, pages 672–682. IEEE, 2012.

[42] A. Dal Corso, D. Macedonio, and M. Merro. Statistical model checking of ad hoc routing protocols in lossy grid networks. In *Proc. 7th International Symposium on NASA Formal Methods*, volume 9058 of *LNCS*, pages 112–126. Springer, 2015.

[43] T. Cui, L. Chen, and T. Ho. Proc. 46th IEEE conference on distributed optimization in wireless networks using broadcast advantage. In *Proc. Decision and Control*, pages 5839–5844. IEEE, 2007.

[44] K. Czarnecki and M. Antkiewicz. Mapping features to models: a template approach based on superimposed variants. In *Proc. 4th Conference on Generative Programming and Component Engineering,*, volume 3676 of *LNCS*, pages 422–437. Springer, 2005.

[45] E. M. Dashofy, A. van der Hoek, and R. N. Taylor. An infrastructure for the rapid development of xml-based architecture description languages. In *Proc. 22rd Conference on Software Engineering*, pages 266–276. ACM, 2002.

[46] A. David, K. G. Larsen, A. Legay, M. Mikucionis, and D. Bøgsted Poulsen. Uppaal SMC tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4):397–415, 2015.

[47] R. De Nicola, A. Fantechi, S. Gnesi, and G. Ristori. An action-based framework for verifying logical and behavioural properties of concurrent systems. *Computer Networks and ISDN Systems*, 25(7):761–778, 1993.

[48] R. De Nicola and F. W. Vaandrager. Action versus state based logics for transition systems. In *Semantics of Systems of Concurrent Processes*, volume 469 of *LNCS*, pages 407–419. Springer, 1990.

[49] R. De Nicola and F.W. Vaandrager. Three logics for branching bisimulation. *Journal of the ACM*, 42(2):458–487, 1995.

[50] R. de Renesse and A. H. Aghvami. Formal verification of ad-hoc routing protocols using SPIN model checker. In *Proc. 12th Mediterranean Electrotechnical Conference*, pages 1177–1182. IEEE, 2004.

[51] G. Delzanno, A. Sangnier, R. Traverso, and G. Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In *Proc. Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 18 of *LIPIcs*, pages 289–300. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[52] G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of safety properties in ad hoc network protocols. In *Proc. 1st Workshop on Process Algebra and Coordination*, volume 60 of *EPTCS*, pages 56–65, 2011.

[53] E. A. Emerson E. M. Clarke and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transaction of Programming Languages and Systems*, 8(2):244–263, 1986.

[54] S. Edenhofer and P. Höfner. Towards a rigorous analysis of aodvv2 (dymo). In *Proc. 20th IEEE International Conference on Network Protocols*, pages 1–6. IEEE Computer Society, 2012.

[55] H. Ehrich, J. Loeckx, and M Wolf. *Specification of Abstract Data Types*. John Wiley, 1996.

[56] E. A. Emerson and R. J. Trefler. From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. In *Proc. 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, volume 1703 of *LNCS*, pages 142–156. Springer, 1999.

[57] C. Ene and T. Muntean. Expressiveness of point-to-point versus broadcast communications. In *Proc. 12th Symposium on Fundamentals of Computation Theory*, volume 1684 of *LNCS*, pages 258–268. Springer, 1999.

[58] E. Engström and P. Runeson. Software product line testing - A systematic mapping study. *Information & Software Technology*, 53(1):2–13, 2011.

[59] K. R. Fall and W. R. Stevens. *TCP/IP Illustrated*, volume 1. Addison-Wesley, 2011.

[60] A. Fantechi and S. Gnesi. Formal modeling for product line families engineering. In *Proc. 12th Conference on Software Product Lines*, pages 193–202. IEEE, 2008.

[61] A. Fehnker, R. J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. Tan. Automated analysis of AODV using UPPAAL. In *Proc. 18th Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *LNCS*, pages 173–187. Springer, 2012.

[62] A. Fehnker, R. J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan. A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV. *arXiv preprint arXiv:1312.7645*, 2013.

[63] A. Fehnker, R. J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan. Automated analysis of AODV using UPPAAL. In *Proc. 18th Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *LNCS*, pages 173–187. Springer, 2012.

[64] A. Fehnker, R. J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan. A process algebra for wireless mesh networks. In *Proc. 21st European Symposium on Programming*, volume 7211 of *LNCS*, pages 295–315. Springer, 2012.

[65] D. Fischbein, S. Uchitel, and V. Braberman. A foundation for behavioral conformance in software product line architectures. In *Proc. Workshop on Role of Software Architecture for Testing and Analysis*, pages 39–48. ACM, 2006.

[66] W. J. Fokkink. *Distributed Algorithms: An Intuitive Approach*. MIT Press, 2013.

[67] W. J. Fokkink, J. Pang, and J. van de Pol. Cones and foci: A mechanical framework for protocol verification. *Formal Methods in System Design*, 29(1):1–31, 2006.

[68] W. J. Fokkink, J. Pang, and J. C. van de Pol. Cones and foci: A mechanical framework for protocol verification. *Formal Methods in System Design*, 29(1):1–31, 2006.

[69] F. Ghassemi and W. Fokkink. Reliable restricted process theory. *CoRR*, abs/1705.02600, 2017.

[70] F. Ghassemi and W. J. Fokkink. Model checking mobile ad hoc networks. *Formal Methods in System Design*, 49(3):159–189, 2016.

[71] F. Ghassemi, W. J. Fokkink, and A. Movaghar. Restricted broadcast process theory. In *Proc. 6th Conference on Software Engineering and Formal Methods*, pages 345–354. IEEE, 2008.

[72] F. Ghassemi, W. J. Fokkink, and A. Movaghar. Equational reasoning on ad hoc networks. In *Proc. 3rd Conference on Fundamentals of Software Engineering*, volume 5961 of *LNCS*, pages 113–128. Springer, 2009.

[73] F. Ghassemi, W. J. Fokkink, and A. Movaghar. Equational reasoning on mobile ad hoc networks. *Fundamenta Informaticae*, 103:1–41, 2010.

[74] F. Ghassemi, W. J. Fokkink, and A. Movaghar. Verification of mobile ad hoc networks: An algebraic approach. *Theoretical Computer Science*, 412(28):3262–3282, 2011.

[75] F. Ghassemi and M. R. Mousavi. Product line process theory. *Journal of Logic and Algebraic Programming*, 85(1):200–226, 2016.

[76] R. J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[77] S. Gnesi and M. Petrocchi. Towards an executable algebra for product lines. In *Proc. 16th Software Product Line Conference*, pages 66–73. ACM, 2012.

[78] J. Godskesen. A calculus for mobile ad hoc networks. In *Proc. 9th Conference on Coordination Models and Languages*, volume 4467 of *LNCS*, pages 132–150. Springer, 2007.

[79] J. Godskesen. A calculus for mobile ad-hoc networks with static location binding. In *Proc. 15th Workshop on Expressiveness in Concurrency*, volume 242 of *Electronic Notes in Theoretical Computer Science*, pages 161–183, 2009.

[80] J. Godskesen and O. Gryn. Modeling and verification of security protocols for ad hoc networks using UPPAAL. In *Proc. 18th Nordic Workshop on Programming Theory*, page 3 pages, 2006.

[81] J. Godskesen and S. Nanz. Mobility models and behavioural equivalence for wireless networks. In *Proc. 11th Conference on Coordination Models and Languages*, volume 5521 of *LNCS*, pages 106–122. Springer, 2009.

[82] J. F. Groote, A. Mathijssen, M. Reniers, Y. Usenko, and M. van Weerdenburg. The formal specification language mCRL2. In *Methods for Modelling Software Systems*, volume 06351 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl, 2006.

[83] J. F. Groote and M.R Mousavi. *Modeling and Analysis of Communicating Systems*. MIT Press, 2014.

[84] J. F. Groote and A. Ponse. μCRL: A base for analysing processes with data. In *Proc. 3rd Workshop on Concurrency and Compositionality*, pages 125–130. GMD-Studien Nr. 191, 1991.

[85] J. F. Groote and A. Ponse. Syntax and semantics of μCRL. In *Proc. Workshop on Algebra of Communicating Processes*, Workshops in Computing, pages 26–62. Springer, 1995.

[86] J. F. Groote and J. Springintveld. Focus points and convergent process operators: A proof strategy for protocol verification. *Journal of Logic and Algebraic Programming*, 49(1-2):31–60, 2001.

[87] J. F. Groote and J. van Wamel. The parallel composition of uniform processes with data. *Theoretical Computer Science*, 266(1-2):631–652, 2001.

[88] J. F. Groote and T. A. C. Willemse. Parameterised boolean equation systems. *Theoretical Computer Science*, 343(3):332–369, 2005.

[89] A. Gruler, M. Leucker, and K. D. Scheidemann. Calculating and modeling common parts of software product lines. In *Proc. 12th Software Product Line Conference*, pages 203–212. IEEE, 2008.

[90] A. Gruler, M. Leucker, and K. D. Scheidemann. Modeling and model checking software product lines. In *Proc. 10th Conference on Formal Methods for Open Object-Based Distributed Systems*, volume 5051 of *LNCS*, pages 113–131. Springer, 2008.

[91] M. Hammer and M. Weber. "To store or not to store" reloaded: Reclaiming memory on demand. In *Proc. 11th Workshop on Formal Methods for Industrial Critical Systems*, volume 4346 of *LNCS*, pages 51–66. Springer, 2006.

[92] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[93] C. Hewitt. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8(3):323–364, 1977.

[94] P. Höfner, R. Khédri, and B. Möller. An algebra of product families. *Software and System Modeling*, 10(2):161–182, 2011.

[95] P. Höfner and A. McIver. Statistical model checking of wireless mesh routing protocols. In *Proc. 5th International Symposium on NASA Formal Methods*, volume 7871 of *LNCS*, pages 322–336. Springer, 2013.

[96] Peter Höfner, Robert J. van Glabbeek, Wee Lum Tan, Marius Portmann, Annabelle McIver, and Ansgar Fehnker. A rigorous analysis of aodv and its variants. In *Proc. 15th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 203–212. ACM, 2012.

[97] M. M. Jaghoori, M. Sirjani, M. R. Mousavi, E. Khamespanah, and A. Movaghar. Symmetry and partial order reduction techniques in model checking Rebeca. *Acta Informatica*, 47(1):33–66, 2010.

[98] M. Kamali, M. Merro, and A. Dal Corso. Aodvv2: Performance vs. loop freedom. In *Proc. 44th International Conference on Current Trends in Theory and Practice of Computer Science*, volume 10706 of *LNCS*, pages 337–350. Springer, 2018.

[99] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, CMU/SEI-90-TR-21, 1990.

[100] R. K. Karmani and G. Agha. Actors. In *Encyclopedia of Parallel Computing*, pages 1–11. Springer, 2011.

[101] J. Katoen. Model checking: One can do much more than you think! In *Proc. 4th Conference on Fundamentals of Software Engineering*, volume 7141 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2011.

[102] E. Khamespanah, M. Sirjani, Z. Sabahi-Kaviani, R. Khosravi, and M. Izadi. Timed rebeca schedulability and deadlock freedom analysis using bounded floating time transition system. *Science of Computer Programming*, 98:184–204, 2015.

[103] P. Khengar and A.H. Aghvami. WARP - the wireless adaptive routing protocol. In *Proc. IST Mobile Communications Summit 2001*, pages 480–485, 2001.

[104] D. Kouzapas and A. Philippou. A process calculus for dynamic networks. In *Proc. Conference on Formal Techniques for Distributed Systems*, volume 6722 of *LNCS*, pages 213–227. Springer, 2011.

[105] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[106] K. G. Larsen, U. Nyman, and A. Wasowski. Modal I/O automata for interface and product line theories. In *Proc. 16th European Symposium on Programming Languages and Systems*, volume 4421 of *LNCS*, pages 64–79. Springer, 2007.

[107] K. G. Larsen and B. Thomsen. A modal process logic. In *Proc. 3rd Annual Symposium on Logic in Computer Science*, pages 203–210. IEEE, 1988.

[108] K. Lauenroth, S. Thning, and K. Pohl. Model checking of domain artifacts in produc line engineering. In *Proc. 24th IEEE/ACM Conference on Automated Software Engineering*, pages 269–280. IEEE, 2009.

[109] M. Leucker and D. Thoma. A formal approach to software product families. In *Proc. 5th Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, volume 7609 of *LNCS*, pages 131–145. Springer, 2012.

[110] J. Liu, J. Dehlinger, and R. Lutz. Safety analysis of software product lines using state-based modeling. *Journal of Systems and Software*, 80(11):1879–1892, 2007.

[111] B. Luttik. *Choice Quantification in Process Algebra*. PhD thesis, University of Amsterdam, 2002.

[112] R. Mateescu and M. Sighireanu. Efficient on-the-fly model-checking for regular alternation-free mu-calculus. *Science of Computer Programming*, 46(3):255–281, 2003.

[113] A. McIver and A. Fehnker. Formal techniques for analysis of wireless networks. In *Proc. 2nd Symposium on Leveraging Applications of Formal Methods*, pages 263–270. IEEE, 2006.

[114] R. Meolic, T. Kapus, and Z. Brezocnik. ACTLW - An action-based computation tree logic with unless operator. *Information Sciences*, 178(6):1542–1557, 2008.

[115] M. Merro. An observational theory for mobile ad hoc networks. *Information and Computation*, 207(2):194–208, 2009.

[116] M. Merro, F. Ballardin, and E. A Sibilio. A timed calculus for wireless systems. *Theoretical Computer Science*, 412(47):6585–6611, 2011.

[117] N. Mezzetti and D. Sangiorgi. Towards a calculus for wireless systems. In *Proc. 22nd Conference on Mathematical Foundations of Programming Semantics*, volume 158 of *Electronic Notes in Theoretical Computer Science*, pages 331–353, 2006.

[118] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[119] K. S. Namjoshi and R. J. Trefler. Analysis of dynamic process networks. In *Proc. 21st Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 9035 of *LNCS*, pages 164–178. Springer, 2015.

[120] K. S. Namjoshi and R. J. Trefler. Loop freedom in AODVv2. In *Proc. 35th IFIP Conference on Formal Techniques for Distributed Objects, Components, and Systems*, volume 9039 of *LNCS*, pages 98–112, 2015.

[121] S. Nanz and C. Hankin. A framework for security analysis of mobile wireless networks. *Theoretical Computer Science*, 367(1):203–227, 2006.

[122] S. Nanz, F. Nielson, and H. Nielson. Static analysis of topology-dependent broadcast networks. *Information and Computation*, 208(2):117–139, 2010.

[123] R. C. van Ommering. Koala, a component model for consumer electronics eroduct software. In *Proc. 2ed Conference on Development and Evolution of Software Architectures for Product Families*, volume 1429 of *LNCS*, pages 76–86. Springer, 1998.

[124] D. A. Peled. All from one, one for all: On model checking using representatives. In *Proc. 5th Conference on Computer Aided Verification*, volume 697 of *LNCS*, pages 409–423. Springer, 1993.

[125] C. E. Perkins and E. M. Belding-Royer. Ad-hoc on-demand distance vector routing. In *Proc. 2nd Workshop on Mobile Computing Systems and Applications*, pages 90–100. IEEE, 1999.

[126] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.

[127] A. Pnueli, J. Xu, and L. D. Zuck. Liveness with (0, 1, infty)-counter abstraction. In *Proc. 14th Conference on Computer Aided Verification*, volume 2404 of *LNCS*, pages 107–122. Springer, 2002.

[128] K. Pohl, G. Bockle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.

[129] K. V. S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25(2-3):285–327, 1995.

[130] T. Razafindralambo and F. Valois. Performance evaluation of backoff algorithms in 802.11 ad-hoc networks. In *Proc. 3rd ACM Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks*, pages 82–89. ACM, 2006.

[131] A. H. N. Reynisson, M. Sirjani, L. Aceto, M. Cimini, A. Jafari, A. Ingólfsdóttir, and S. H. Sigurdarson. Modelling and simulation of asynchronous real-time systems using Timed Rebeca. *Science of Computer Programming*, 89:41–68, 2014.

[132] H. Sabouri and R. Khosravi. Delta modeling and model checking of product families. In *Proc. 5th Conferece on Fundamentals of Software Engineering*, volume 8161 of *LNCS*, pages 51–65. Springer, 2013.

[133] H. Sabouri and M. Sirjani. Slicing-based reductions for rebeca. *Electronic Notes in Theoretical Computer Science*, 260:209–224, 2010.

[134] M. Saksena, O. Wibling, and B. Jonsson. Graph grammar modeling and verification of ad hoc routing protocols. In *Proc. 14th Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *LNCS*, pages 18–32. Springer, 2008.

[135] K. Schmid and F. van der Linden. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer, 2010.

[136] S. Shoham and O. Grumberg. Multi-valued model checking games. *Journal of Computer and System Sciences*, 78(2):414–429, 2012.

[137] A. Singh, C. R. Ramakrishnan, and S. A. Smolka. A process calculus for mobile ad hoc networks. *Science of Computer Programming*, 75(6):440–469, 2010.

[138] M. Sirjani and M. M. Jaghoori. Ten years of analyzing actors: Rebeca experience. In *Formal Modeling: Actors, Open Systems, Biological Systems*, volume 7000 of *LNCS*, pages 20–56. Springer, 2011.

[139] M. Sirjani, A. Movaghar, A. Shali, and F. S. de Boer. Modeling and verification of reactive systems using Rebeca. *Fundamenta Informaticae*, 63(4):385–410, 2004.

[140] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake. A classification and survey of analysis strategies for software product lines. *ACM Computing Surveys*, 47(1):6:1–6:45, 2014.

[141] M. Tribastone. Behavioral relations in a process algebra for variants. In *Proc. 18th Software Product Line Conference*, pages 82–91. ACM, 2014.

[142] C. Tschudin, R. Gold, O. Rensfelt, and O. Wibling. LUNAR: a lightweight underlay network ad-hoc routing protocol and implementation. In *Proc. 4th Conference on Next Generation Teletraffic and Wired/Wireless Advanced Networking*, page 300, 2004.

[143] Y. Usenko. *Linearization in μCRL*. PhD thesis, Eindhoven University of Technology, 2002.

[144] R. van Glabbeek, P. Höfner, W. L. Tan, and M. Portmann. Sequence numbers do not guarantee loop freedom: Aodv can yield routing loops. In *Proc. 16th ACM International Conference on Modeling, Analysis & Simulation of Wireless and Mobile Systems*, pages 91–100. ACM, 2013.

[145] R. J. van Glabbeek. A complete axiomatization for branching bisimulation congruence of finite-state behaviours. In *Proc. 18th Symposium on Mathematical Foundations of Computer Science*, volume 711 of *LNCS*, pages 473–484. Springer, 1993.

[146] R. J. van Glabbeek. Notes on the methodology of CCS and CSP. *Theoretical Computer Science*, 177(2):329–349, 1997.

[147] R. J. van Glabbeek. The linear time-branching time spectrum i - the semantics of concrete, sequential processes. In *Handbook of Process Algebra*, pages 3–99. Elsevier, 2001.

[148] R. J. van Glabbeek, P. Höfner, M. Portmann, and W. Lum Tan. Modelling and verifying the AODV routing protocol. *Distributed Computing*, 29(4):279–315, 2016.

[149] M. Varshosaz and R. Khosravi. Modeling and verification of probabilistic actor systems using pRebeca. In *Proc. 14th Conference on Formal Engineering Methods*, volume 7635 of *LNCS*, pages 135–150. Springer, 2012.

[150] S. Vasudevan, J. Kurose, and D. Towsley. Design and analysis of a leader election algorithm for mobile ad hoc networks. In *Proc. 12th Conference on Network Protocols*, pages 350–360. IEEE, 2004.

[151] O. Wibling, J. Parrow, and A. Pears. Automatized verification of ad hoc routing protocols. In *Proc. 24th IFIP Conference on Formal Techniques for Networked and Distributed Systems*, volume 3235 of *LNCS*, pages 343–358. Springer, 2004.

[152] O. Wibling, J. Parrow, and A. Pears. Ad hoc routing protocol verification through broadcast abstraction. In *Proc. 25th IFIP Conference on Formal Techniques for Networked and Distributed Systems*, volume 3731 of *LNCS*, pages 128–142. Springer, 2005.

[153] B. Yousefi. Modeling and analysis of broadcasting actors. Master's thesis, University of Tehran, 2015.

[154] B. Yousefi and F. Ghassemi. An efficient loop-free version of aodvv2. *CoRR*, abs/1709.01786v2, 2017.

[155] B. Yousefi, F. Ghassemi, and R. Khosravi. Modeling and efficient verification of broadcasting actors. In *Proc. 6th Conference on Fundamentals of Software Engineering*, volume 9392 of *LNCS*, pages 69–83, 2015.

[156] B. Yousefi, F. Ghassemi, and R. Khosravi. Modeling and efficient verification of wireless ad hoc networks. *Formal Aspect of Computing*, To appear, 2017.

[157] T. Ziadi, L. Hélouët, and J. M. Jézéquel. Towards a UML profile for software product lines. In *Proc. 5th Workshop on Product-Family Engineering*, volume 3014 of *LNCS*, pages 129–139. Springer, 2003.

Proofs of Chapter 3

A

A.1 Proof of Theorem 3.2

To prove that branching reliable computed network bisimilarity is an equivalence, we exploit semi-branching reliable computed network bisimilarity, following [12].

Definition A.1. A binary relation $\mathcal{R}$ on computed network terms is a semi-branching reliable computed network simulation, if $t_1 \mathcal{R} t_2$ implies that whenever $t_1 \xrightarrow{(\mathcal{C},\eta)} t_1'$:

- either $\eta = \tau$ and there is t_2' such that $t_2 \Rightarrow t_2'$ with $t_1 \mathcal{R} t_2'$ and $t_1' \mathcal{R} t_2'$; or
- there are $s_1'', \ldots, s_k''$ and $s_1', \ldots, s_k'$ for some $k > 0$ such that $\forall i \leq k\,(t_2 \Rightarrow s_i'' \xrightarrow{\langle(\mathcal{C}_i,\eta)\rangle} s_i'$, with $t_1 \mathcal{R} s_i''$ and $t_1' \mathcal{R} s_i')$, and $\langle \mathcal{C}_1 \rangle, \ldots, \langle \mathcal{C}_k \rangle$ constitute a partitioning of $\langle \mathcal{C} \rangle$.

$\mathcal{R}$ is a semi-branching reliable computed network bisimulation if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are semi-branching reliable computed network simulations. Computed networks t_1 and t_2 are semi-branching reliable computed network bisimilar if $t_1 \mathcal{R} t_2$, for some semi-branching reliable computed network bisimulation relation $\mathcal{R}$.

Lemma A.2. *Let t_1 and t_2 be computed network terms, and $\mathcal{R}$ a semi-branching reliable computed network bisimulation such that $t_1 \mathcal{R} t_2$.*

- *If $t_1 \Rightarrow t_1'$ then $\exists t_2' \cdot t_2 \Rightarrow t_2' \wedge t_1' \mathcal{R} t_2'$*
- *If $t_2 \Rightarrow t_2'$ then $\exists t_1' \cdot t_1 \Rightarrow t_1' \wedge t_1' \mathcal{R} t_2'$*

Proof. We only give the proof of the first property. The second property can be proved in a similar fashion. The proof is by induction on the number of $\Rightarrow$ steps from t_1 to t_1':

- Base: Assume that the number of steps equals zero. Then t_1 and t_1' must be equal. Since $t_1 \mathcal{R} t_2$ and $t_2 \Rightarrow t_2$, the property is satisfied.

- Induction step: Assume $t_1 \Rightarrow t_1'$ in n steps, for some $n \geq 1$. Then there is t_1'' such that $t_1 \Rightarrow t_1''$ in $n-1 \Rightarrow$ steps, and $t_1'' \xrightarrow{(\mathcal{C},\tau)} t_1'$. By the induction hypothesis, there exists t_2'' such that $t_2 \Rightarrow t_2''$ and $t_1'' \mathcal{R} t_2''$. Since $t_1'' \xrightarrow{(\mathcal{C},\tau)} t_1'$ and $\mathcal{R}$ is a semi-branching reliable computed network bisimulation, there are two cases to consider:
  - there is t_2' such that $t_2'' \Rightarrow t_2'$, $t_1'' \mathcal{R} t_2'$, and $t_1' \mathcal{R} t_2'$. So $t_2 \Rightarrow t_2'$ such that $t_1' \mathcal{R} t_2'$.
  - or there are $s_1''', \ldots, s_k'''$ and $s_1', \ldots, s_k'$ for some $k > 0$ such that $\forall i \leq k\,(t_2'' \Rightarrow s_i''' \xrightarrow{(\mathcal{C}_i,\tau)} s_i'$, with $t_1'' \mathcal{R} s_i'''$ and $t_1' \mathcal{R} s_i')$, and $\mathcal{C}_1, \ldots, \mathcal{C}_k$ constitute a partitioning of $\mathcal{C}$. By definition, $s_i''' \xrightarrow{(\mathcal{C}_i,\tau)} s_i'$ yields $s_i''' \Rightarrow s_i'$. Consequently for any arbitrary $i \leq k$, $t_2 \Rightarrow s_i'$ such that $t_1' \mathcal{R} s_i'$.

□

Proposition A.3. *The relation composition of two semi-branching reliable computed network bisimulations is again a semi-branching reliable computed network bisimulation.*

Proof. Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be semi-branching reliable computed network bisimulations with $t_1 \mathcal{R}_1 t_2$ and $t_2 \mathcal{R}_2 t_3$. Let $t_1 \xrightarrow{(\mathcal{C},\eta)} t_1'$. It must be shown that

- either $\eta = \tau$ and there is t_3' such that $t_3 \Rightarrow t_3'$ with $t_1 \mathcal{R}_1 \circ \mathcal{R}_2 t_3'$ and $t_1' \mathcal{R}_1 \circ \mathcal{R}_2 t_3'$; or
- $\exists s_1', \ldots, s_k', s_1'', \ldots, s_k''\, \forall i \leq k\,(t_3 \Rightarrow s_i'' \xrightarrow{\langle(\mathcal{C}_i,\eta)\rangle} s_i' \wedge t_1 \mathcal{R}_1 \circ \mathcal{R}_2 s_i'' \wedge t_1' \mathcal{R}_1 \circ \mathcal{R}_2 s_i')$, where $\langle\mathcal{C}_1\rangle, \ldots, \langle\mathcal{C}_k\rangle$ constitute a partitioning of $\langle\mathcal{C}\rangle$.

Since $t_1 \mathcal{R}_1 t_2$, two cases can be considered:

- $\eta = \tau$ and there exists t_2' such that $t_2 \Rightarrow t_2'$ with $t_1 \mathcal{R}_1 t_2'$ and $t_1' \mathcal{R}_1 t_2'$. Lemma A.2 yields that there exists t_3' that $t_3 \Rightarrow t_3'$ with $t_2' \mathcal{R}_2 t_3'$. It immediately follows that $t_1 \mathcal{R}_1 \circ \mathcal{R}_2 t_3'$ and $t_1' \mathcal{R}_1 \circ \mathcal{R}_2 t_3'$.
- there exist $s_1^{**}, \ldots s_j^{**}$, $s_1^* \ldots s_j^*$ for some $j > 0$ such that $\forall i \leq j\,(t_2 \Rightarrow s_i^{**} \xrightarrow{\langle(\mathcal{C}_i,\eta)\rangle} s_i^*$, $t_1 \mathcal{R}_1 s_i^{**}$, $t_1' \mathcal{R}_1 s_i^*)$, and $\langle\mathcal{C}_1\rangle, \ldots, \langle\mathcal{C}_j\rangle$ is a partitioning of $\langle\mathcal{C}\rangle$. Since $t_2 \mathcal{R}_2 t_3$ and $t_2 \Rightarrow s_i^{**}$, Lemma A.2 yields that there are $s_1''', \ldots, s_j'''$ such that $\forall i \leq j\,(t_3 \Rightarrow s_i''' \wedge s_i^{**} \mathcal{R}_2 s_i''')$. Two cases can be distinguished:
  - either $\eta = \tau$ and for some $i \leq j$, $s_i^{**} \xrightarrow{(\mathcal{C}_i,\tau)} s_i^*$ implies there is s_i'' such that $s_i''' \Rightarrow s_i''$ with $s_i^{**} \mathcal{R}_2 s_i''$ and $s_i^* \mathcal{R}_2 s_i''$. It follows immediately that there exists s_i'' such that $t_3 \Rightarrow s_i''$ with $t_1 \mathcal{R}_1 \circ \mathcal{R}_2 s_i''$ and $t_1' \mathcal{R}_1 \circ \mathcal{R}_2 s_i''$; or

– for all $i \leq j$, $s_i^{**} \xrightarrow{\langle(\mathcal{C}_i,\eta)\rangle} s_i^*$ implies there are $s''_{i_1},\ldots,s''_{i_{k_i}}$ and $s'_{i_1},\ldots,s'_{i_{k_i}}$ for some $k_i > 0$ such that $\forall o \leq k_i\,(s''_i \Rightarrow s''_{i_o} \xrightarrow{\langle(\mathcal{C}_{i_o},\eta)\rangle} s'_{i_o}$, $s_i^{**}\mathcal{R}_2 s''_{i_o}$, $s_i^{*}\mathcal{R}_2 s'_{i_o})$, and $\langle\mathcal{C}_{i_1}\rangle,\ldots,\langle\mathcal{C}_{i_{k_i}}\rangle$ is a partitioning of $\langle\mathcal{C}_i\rangle$. Since $t_3 \Rightarrow s''_i$, we have $\forall i \leq j, \forall o \leq k_i\,(t_3 \Rightarrow s''_{i_o} \xrightarrow{\langle(\mathcal{C}_{i_o},\eta)\rangle} s'_{i_o}$ with $t_1\mathcal{R}_1 \circ \mathcal{R}_2 s''_{i_o}$, $t'_1\mathcal{R}_1 \circ \mathcal{R}_2 s'_{i_o})$, and $\{\langle\mathcal{C}_{i_o}\rangle \mid i \leq j, o < k_i\}$ is a partitioning of $\langle\mathcal{C}\rangle$.

□

Corollary A.4. *Semi-branching reliable computed network bisimilarity is an equivalence relation.*

Proposition A.5. *Each largest semi-branching reliable computed network bisimulation is a branching reliable computed network bisimulation.*

Proof. Suppose $\mathcal{R}$ is the largest semi-branching reliable computed network bisimulation for some given CLTSs. Let $t_1\mathcal{R}t_2$, $t_2 \Rightarrow t'_2$, $t_1\mathcal{R}t'_2$ and $t'_1\mathcal{R}t'_2$. We show that $\mathcal{R}' = \mathcal{R} \cup \{(t'_1, t_2)\}$ is a semi-branching reliable computed network bisimulation.

1. If $t'_1 \xrightarrow{(\mathcal{C},\eta)} t''_1$, then it follows from $(t'_1, t'_2) \in \mathcal{R}$ that
   - either $\eta = \tau$ and there exists t''_2 such that $t'_2 \Rightarrow t''_2$ with $t'_1\mathcal{R}t''_2$ and $t''_1\mathcal{R}t''_2$. Finally $t_2 \Rightarrow t'_2$ results $t'_1\mathcal{R}'t''_2$ and $t''_1\mathcal{R}'t''_2$; or
   - there are $s'''_1,\ldots,s'''_k$ and $s''_1,\ldots,s''_k$ for some $k > 0$ such that $\forall i \leq k\,(t'_2 \Rightarrow s'''_i \xrightarrow{\langle(\mathcal{C}_i,\eta)\rangle} s''_i$ with $(t'_1, s'''_i), (t''_1, s''_i) \in \mathcal{R})$ and $\langle\mathcal{C}_1\rangle,\ldots,\langle\mathcal{C}_k\rangle$ is a partitioning of $\langle\mathcal{C}\rangle$. And $t_2 \Rightarrow t'_2$ yields $\forall i \leq k\,(t_2 \Rightarrow s'''_i \xrightarrow{\langle(\mathcal{C}_i,\eta)\rangle} s''_i$, with $(t'_1, s'''_i), (t''_1, s''_i) \in \mathcal{R}')$.
2. If $t_2 \xrightarrow{(\mathcal{C},\eta)} t''_2$, then it follows from $(t_1, t_2) \in \mathcal{R}$ that
   - either $\eta = \tau$, and there exists t''_1 such that $t_1 \Rightarrow t''_1$ with $t''_1\mathcal{R}t_2$ and $t''_1\mathcal{R}t''_2$. Furthermore, $(t_1, t'_2) \in \mathcal{R}$, $t_1 \Rightarrow t''_1$, and Lemma A.2 imply there exists t'''_2 such that $t'_2 \Rightarrow t'''_2$ with $(t''_1, t'''_2) \in \mathcal{R}$. Similarly $(t'_1, t'_2) \in \mathcal{R}$, $t'_2 \Rightarrow t'''_2$, and Lemma A.2 imply there exists t'''_1 such that $t'_1 \Rightarrow t'''_1$ with $(t'''_1, t'''_2) \in \mathcal{R}$. From $(t'''_1, t'''_2) \in \mathcal{R}$, $(t'''_2, t''_1) \in \mathcal{R}^{-1}$, and $(t''_1, t_2) \in \mathcal{R}$, we conclude $(t'''_1, t_2) \in \mathcal{R} \circ \mathcal{R}^{-1} \circ \mathcal{R}$. And from $(t'''_1, t'''_2) \in \mathcal{R}$, $(t'''_2, t''_1) \in \mathcal{R}^{-1}$, and $(t''_1, t''_2) \in \mathcal{R}$, we conclude $(t'''_1, t''_2) \in \mathcal{R} \circ \mathcal{R}^{-1} \circ \mathcal{R}$.
   - or there are $s'''_{1_1},\ldots,s'''_{1_k}$ and $s''_{1_1},\ldots,s''_{1_k}$ for some $k > 0$ such that $\forall i \leq k\,(t_1 \Rightarrow s'''_{1_i} \xrightarrow{\langle(\mathcal{C}_i,\eta)\rangle} s''_{1_i}$ with $(s'''_{1_i}, t_2), (s''_{1_i}, t''_2) \in \mathcal{R})$ and $\langle\mathcal{C}_1\rangle,\ldots,\langle\mathcal{C}_k\rangle$ is a partitioning of $\langle\mathcal{C}\rangle$. Since $(t_1, t'_2) \in \mathcal{R}$ and $t_1 \Rightarrow s'''_{1_i}$, by Lemma A.2, there are $s'''_{2_1},\ldots,s'''_{2_k}$ such that $\forall i \leq k\,(t'_2 \Rightarrow s'''_{2_i}$ and $(s'''_{1_i}, s'''_{2_i}) \in \mathcal{R})$. Since $s'''_{1_i} \xrightarrow{\langle(\mathcal{C}_i,\eta)\rangle} s''_{1_i}$, there are $s^{**}_{2_{i_1}},\ldots,s^{**}_{2_{i_{k_i}}}$ and $s^{*}_{2_{i_1}},\ldots,s^{*}_{2_{i_{k_i}}}$ for

some $k_i > 0$ such that $\forall o \leq k_i\,(s'''_{2_i} \Rightarrow s^{**}_{2_{i_o}} \xrightarrow{\langle(\mathcal{C}_{i_o},\eta)\rangle} s^{*}_{2_{i_o}}$ with $(s'''_{1_i}, s^{**}_{2_{i_o}}), (s''_{1_i}, s^{*}_{2_{i_o}}) \in \mathcal{R}$) and $\langle\mathcal{C}_{i_1}\rangle, \ldots, \langle\mathcal{C}_{i_{k_i}}\rangle$ is a partitioning of $\langle\mathcal{C}_i\rangle$. Since $t'_2 \Rightarrow s'''_{2_i}$ and $s'''_{2_i} \Rightarrow s^{**}_{2_{i_o}}$, we have $\forall i \leq k, o \leq k_i\,(t'_2 \Rightarrow s^{**}_{2_{i_o}})$. By assumption, $(t'_1, t'_2) \in \mathcal{R}$, so by Lemma A.2 there are $s^{**}_{1_1}, \ldots, s^{**}_{1_K}$, where $K = \sum_{i=1}^{k} k_i$, such that $\forall z \leq K\,(t'_1 \Rightarrow s^{**}_{1_z}$ and $(s^{**}_{1_z}, s^{**}_{2_{i_o}}) \in \mathcal{R}$, where $z = (\sum_{j=1}^{i-1} k_j) + o)$. Since $s^{**}_{2_{i_o}} \xrightarrow{\langle(\mathcal{C}_{i_o},\eta)\rangle} s^{*}_{2_{i_o}}$, there are $s^{***}_{1_{z_1}}, \ldots, s^{***}_{1_{z_{k'_z}}}$ and $s'_{1_{z_1}}, \ldots, s'_{1_{z_{k'_z}}}$ for some $k'_z > 0$ such that $\forall j \leq k'_z\,(s^{**}_{1_z} \Rightarrow s^{***}_{1_{z_j}} \xrightarrow{\langle(\mathcal{C}_{i_{o_j}},\eta)\rangle} s'_{1_{z_j}}$ with $(s^{***}_{1_{z_j}}, s^{**}_{2_{i_o}}), (s'_{1_{z_j}}, s^{*}_{2_{i_o}}) \in \mathcal{R}$) and $\langle\mathcal{C}_{i_{o_1}}\rangle, \ldots, \langle\mathcal{C}_{i_{o_{k'_z}}}\rangle$ is a partitioning of $\langle\mathcal{C}_{i_o}\rangle$. And $t'_1 \Rightarrow s^{**}_{1_z}$ yields $\forall i \leq k, o \leq k_i, j \leq k'_z (t'_1 \Rightarrow s^{***}_{1_{z_j}} \xrightarrow{\langle(\mathcal{C}_{i_{o_j}},\eta)\rangle} s'_{1_{z_j}}$ with

$$
\begin{aligned}
(s^{***}_{1_{z_j}}, s^{**}_{2_{i_o}}) \in \mathcal{R} \wedge (s^{**}_{2_{i_o}}, s'''_{1_i}) \in \mathcal{R}^{-1} \wedge (s'''_{1_i}, t_2) \in \mathcal{R} \\
\Rightarrow (s^{***}_{1_{z_j}}, t_2) \in \mathcal{R} \circ \mathcal{R}^{-1} \circ \mathcal{R} \\
(s'_{1_{z_j}}, s^{*}_{2_{i_o}}) \in \mathcal{R} \wedge (s^{*}_{2_{i_o}}, s''_{1_i}) \in \mathcal{R}^{-1} \wedge (s''_{1_i}, t''_1) \in \mathcal{R} \\
\Rightarrow (s'_{1_{z_j}}, t''_2) \in \mathcal{R} \circ \mathcal{R}^{-1} \circ \mathcal{R},
\end{aligned}
$$

where $z = (\sum_{l=1}^{i-1} k_l) + o)$, and $\{\langle\mathcal{C}_{i_{o_j}}\rangle \mid i \leq k, o \leq k_i, j \leq k'_z\}$ is a partitioning of $\langle\mathcal{C}\rangle$.

By Proposition A.3, $\mathcal{R} \circ \mathcal{R}^{-1} \circ \mathcal{R}$ is a semi-branching reliable computed network bisimulation. Since $\mathcal{R}$ is the largest semi-branching reliable computed network bisimulation, and clearly $\mathcal{R} \subseteq \mathcal{R} \circ \mathcal{R}^{-1} \circ \mathcal{R}$, we have $\mathcal{R} = \mathcal{R} \circ \mathcal{R}^{-1} \circ \mathcal{R}$.

So $\mathcal{R}'$ is a semi-branching reliable computed network bisimulation. Since $\mathcal{R}$ is the largest semi-branching reliable computed network bisimulation, $\mathcal{R}' = \mathcal{R}$.

We will now prove that $\mathcal{R}$ is a branching reliable computed network bisimulation. Let $t_1 \mathcal{R} t_2$, and $t_1 \xrightarrow{(\mathcal{C},\eta)} t'_1$. We only consider the case when $\eta = \tau$, because for other cases, the transfer condition of Definition 3.1 and Definition A.1 are the same. Two cases can be distinguished:

1. there exists t'_2 such that $t_2 \Rightarrow t'_2$ with $t_1 \mathcal{R} t'_2$ and $t'_1 \mathcal{R} t'_2$: we proved above that $t'_1 \mathcal{R} t_2$. This agrees with the first case of Definition 3.1.
2. there are $s''_1, \ldots, s''_k$ and $s'_1, \ldots, s'_k$ for some $k > 0$ such that $\forall i \leq k\,(t_2 \Rightarrow s''_i \xrightarrow{\langle(\mathcal{C}_i,\tau)\rangle} s'_i$ with $t_1 \mathcal{R} s''_i$ and $t'_1 \mathcal{R} s'_i)$ and $\langle\mathcal{C}_1\rangle, \ldots, \langle\mathcal{C}_k\rangle$ constitute a partitioning of $\langle\mathcal{C}\rangle$. This agrees with the second case of Definition 2.4.

Consequently $\mathcal{R}$ is a branching reliable computed network bisimulation. □

Since any branching reliable computed network bisimulation is a semi-branching reliable computed network bisimulation, this yields the following corollary.

Corollary A.6. *Two computed network terms are related by a branching reliable computed network bisimulation if and only if they are related by a semi-branching reliable computed network bisimulation.*

Corollary A.7. *Branching reliable computed network bisimilarity is an equivalence relation.*

Corollary A.8. *Rooted branching reliable computed network bisimilarity is an equivalence relation.*

Proof. It is easy to show that rooted branching reliable computed network bisimilarity is reflexive and symmetric. To conclude the proof, we show that rooted branching reliable computed network bisimilarity is transitive. Let $t_1 \simeq_{rbr} t_2$ and $t_2 \simeq_{rbr} t_3$. Since $t_1 \simeq_{rbr} t_2$, if $t_1 \xrightarrow{(\mathcal{C},\eta)} t_1'$, then there is t_2' such that $t_2 \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t_2'$ and $t_1' \simeq_{br} t_2'$. Since $t_2 \simeq_{rbr} t_3$, there is t_3' such that $t_3 \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t_3'$ and $t_2' \simeq_{br} t_3'$. Equivalence of branching reliable computed network bisimilarity yields $t_3 \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t_3'$ with $t_1' \simeq_{br} t_3'$. The same argumentation holds when $t_3 \xrightarrow{(\mathcal{C},\eta)} t_3'$. Consequently the transfer conditions of Definition 3.3 holds and $t_1 \simeq_{rbr} t_3$. □

A.2 Proof of Theorem 3.5

Theorem A.9. *Rooted branching reliable computed network bisimilarity is a congruence for terms with respect to RCNT operators.*

Proof. We need to prove the following cases:

1. $[\![t_1]\!]_\ell \simeq_{rbr} [\![t_2]\!]_\ell$ implies $[\![\alpha.t_1]\!]_\ell \simeq_{rbr} [\![\alpha.t_2]\!]_\ell$;
2. $[\![t_1]\!]_\ell \simeq_{rbr} [\![t_2]\!]_\ell$ and $[\![t_1']\!]_\ell \simeq_{rbr} [\![t_2']\!]_\ell$ implies $[\![t_1 + t_1']\!]_\ell \simeq_{rbr} [\![t_2 + t_2']\!]_\ell$;
3. $[\![t_1]\!]_\ell \simeq_{rbr} [\![t_2]\!]_\ell$ and $[\![t_1']\!]_\ell \simeq_{rbr} [\![t_2']\!]_\ell$ implies that $[\![sense(\ell', t_1, t_1')]\!]_\ell \simeq_{rbr} [\![sense(\ell', t_2, t_2')]\!]_\ell$;
4. $[\![t_1]\!]_\ell \simeq_{rbr} [\![t_2]\!]_\ell$ implies $\ell : t : t_1 \simeq_{rbr} \ell : t : t_2$ for any arbitrary term t;
5. $t_1 \simeq_{rbr} t_2$ implies $(\mathcal{C}, \eta).t_1 \simeq_{rbr} (\mathcal{C}, \eta).t_2$;
6. $t_1 \simeq_{rbr} t_2$ and $t_1' \simeq_{rbr} t_2'$ implies $t_1 + t_1' \simeq_{rbr} t_2 + t_2'$;
7. $t_1 \simeq_{rbr} t_2$ implies $(\nu\ell)t_1 \simeq_{rbr} (\nu\ell)t_2$;
8. $t_1 \simeq_{rbr} t_2$ and $t_1' \simeq_{rbr} t_2'$ implies $t_1 \parallel t_1' \simeq_{rbr} t_2 \parallel t_2'$;

9. $t_1 \simeq_{rbr} t_2$ and $t'_1 \simeq_{rbr} t'_2$ implies $t_1 \between t'_1 \simeq_{rbr} t_2 \between t'_2$;
10. $t_1 \simeq_{rbr} t_2$ and $t'_1 \simeq_{rbr} t'_2$ implies $t_1 \mid t'_1 \simeq_{rbr} t_2 \mid t'_2$;
11. $t_1 \simeq_{rbr} t_2$ implies $\partial_M(t_1) \simeq_{rbr} \partial_M(t_2)$;
12. $t_1 \simeq_{rbr} t_2$ implies $\tau_M(t_1) \simeq_{rbr} \tau_M(t_2)$;
13. $t_1 \simeq_{rbr} t_2$ implies $\mathcal{C} \rhd t_1 \simeq_{rbr} \mathcal{C} \rhd t_2$.

Clearly, if $t_1 \simeq_{rbr} t_2$ then $t_1 \simeq_{br} t_2$ is witnessed by the following branching reliable computed network bisimulation relation:

$$\begin{aligned}\mathcal{R}' = \{\mathcal{R} \mid\ & t_1 \xrightarrow{(\mathcal{C},\eta)} t'_1 \Rightarrow \exists t'_2 \cdot t_2 \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t'_2 \wedge t'_1 \simeq_{br} t'_2 \text{ is witnessed by } \mathcal{R}\}\\ & \cup \{\mathcal{R} \mid\ t_2 \xrightarrow{(\mathcal{C},\eta)} t'_2 \Rightarrow \exists t'_1 \cdot t_1 \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t'_1 \wedge t'_1 \simeq_{br} t'_2 \text{ is witnessed by } \mathcal{R}\}\\ & \cup \{(t_1, t_2)\}.\end{aligned}$$

We prove the cases 1, 2, 4, 7, 10, 11, and 13 since the proof of the cases 3 and 6 are similar to the case 2, the case 5 is similar to the case 1, the cases 8 and 9 are similar to the case 10, and the case 12 is similar to the case 11.

Case 1. The first transitions of $[\![\alpha.t_1]\!]_\ell$ and $[\![\alpha.t_2]\!]_\ell$ are the same with application of the rule Snd (if α is a send action), Rcv_1 (if α is a receive action), or $Rcv_{2,3}$ (for receiving $(\mathcal{C}, nrcv(\mathfrak{m}))$which are not derivable from Rcv_1), and by assumption $[\![t_1]\!]_\ell \simeq_{rbr} [\![t_2]\!]_\ell$ implies $[\![t_1]\!]_\ell \simeq_{br} [\![t_2]\!]_\ell$. Thus the transfer conditions of Definition 3.3 hold.

Case 2. Every transition $[\![t_1 + t'_1]\!]_\ell \xrightarrow{(\mathcal{C},\eta)} t$ owes to $[\![t_1]\!]_\ell \xrightarrow{(\mathcal{C},\eta)} t$ or $[\![t'_1]\!]_\ell \xrightarrow{(\mathcal{C},\eta)} t$ by $Choice$, or is implied by Rcv_3, i.e., $[\![t_1 + t'_1]\!]_\ell \xrightarrow{(\mathcal{C},nrcv(\mathfrak{m}))} [\![t_1 + t'_1]\!]_\ell$ iff there exists no $t_1 + t'_1 \xrightarrow{(\mathcal{C}',rcv(\mathfrak{m}))} t^*$ for some t^* such that $\mathcal{C}' \preccurlyeq \mathcal{C}$. We assume that $\mathcal{C}$ is the greatest network constraint derived by Rcv_3 as the other can be derived by application of Exe. For the former case, $[\![t_1]\!]_\ell \simeq_{rbr} [\![t_2]\!]_\ell$ and $[\![t'_1]\!]_\ell \simeq_{rbr} [\![t'_2]\!]_\ell$ imply there is t' such that $[\![t_2]\!]_\ell \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t'$ or $[\![t'_2]\!]_\ell \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t'$ and $t \simeq_{br} t'$. Thus $[\![t_2 + t'_2]\!]_\ell \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t'$ with $t \simeq_{br} t'$. For the latter case by the rule $Choice$, there exists no $t_1 \xrightarrow{(\mathcal{C}',rcv(\mathfrak{m}))} t^*$ and $t'_1 \xrightarrow{(\mathcal{C}',rcv(\mathfrak{m}))} t^*$ for some t^* such that $\mathcal{C}' \preccurlyeq \mathcal{C}$. Thus by the rule Rcv_3, $[\![t_1]\!]_\ell \xrightarrow{(\mathcal{C},nrcv(\mathfrak{m}))} [\![t_1]\!]_\ell$ and $[\![t'_1]\!]_\ell \xrightarrow{(\mathcal{C},nrcv(\mathfrak{m}))} [\![t'_1]\!]_\ell$. We remark that transitions derived by application of Rcv_3 are those that cannot be derived from $Rcv_{1,2}$. The greatest value of the network constraints of such transitions have no pair in the form of $? \rightsquigarrow \ell$ or $? \not\rightsquigarrow \ell$. This implies that such transitions can not be mimicked by application of $Rcv_{1,2}$ (since they will add constraints of the form $? \rightsquigarrow \ell$ or $? \not\rightsquigarrow \ell$) . Therefore, $[\![t_1]\!]_\ell \simeq_{rbr} [\![t_2]\!]_\ell$ and $[\![t'_1]\!]_\ell \simeq_{rbr} [\![t'_2]\!]_\ell$ imply that $[\![t_2]\!]_\ell \xrightarrow{(\mathcal{C},nrcv(\mathfrak{m}))} [\![t_2]\!]_\ell$ and $[\![t'_2]\!]_\ell \xrightarrow{(\mathcal{C},nrcv(\mathfrak{m}))} [\![t'_2]\!]_\ell$ which can be only derived by application of the rule Rcv_3. Therefore, there exists no $t_2 \xrightarrow{(\mathcal{C}',rcv(\mathfrak{m}))} t^*$ and $t'_2 \xrightarrow{(\mathcal{C}',rcv(\mathfrak{m}))} t^*$ for some t^* such that $\mathcal{C}' \preccurlyeq \mathcal{C}$. Hence, $[\![t_2 + t'_2]\!]_\ell \xrightarrow{(\mathcal{C},nrcv(\mathfrak{m}))} [\![t_2 + t'_2]\!]_\ell$.

Case 4 Suppose that $\ell : t : t_1 \xrightarrow{(\mathcal{C}^*,\eta)} t^*$. Two cases can be distinguished:

- It owes to the application of one of the rules $Inter'_{1-3}$ together with some of the rules $Choice'$, Inv', and $Sen'_{1,2}$. Therefore, t_1 has a subterm in the form of $\alpha.t'_1$ where $t^* \equiv [\![t'_1]\!]_\ell$, η is the network action version of α, and $\mathcal{C}^* = \mathcal{C}^*_1 \cup \mathcal{C}^*_2$ such that $\mathcal{C}^*_1$ is the network constraint added by $Inter'_{1-3}$ and $\mathcal{C}^*_2$ is added by the rules $Sen'_{1,2}$ if they are applied. By application of $Prefix'$ and $Snd/Rcv_{1,2}$ together with the rules $Choice$, Inv, and $Sen_{1,2}$, $[\![t_1]\!]_\ell \xrightarrow{(\mathcal{C}^*,\eta)} [\![t'_1]\!]_\ell$. The assumption $[\![t_1]\!]_\ell \simeq_{rbr} [\![t_2]\!]_\ell$ results that $[\![t_2]\!]_\ell \xrightarrow{\langle(\mathcal{C}^*,\eta)\rangle} [\![t'_2]\!]_\ell$ and $[\![t'_1]\!]_\ell \simeq_{br} [\![t'_2]\!]_\ell$. We remark that a transition of t_1 owing to $Rcv_{1,2}$ cannot be matched to a transition of t_2 generated by Rcv_3. By the rules $Rcv_{1,2}$, two transitions are generated with network constraints of the forms $\{?\rightsquigarrow\ell\}\cup\mathcal{C}$ and $\{?\not\rightsquigarrow\ell\}\cup\mathcal{C}$. Assume that one of these transitions of t_1 is matched to a transition of t_2 generated by Rcv_3 with the network constraint $\mathcal{C}$ (together with Exe). Due to our root condition, t_1 must also generate a transition with the network constraint $\mathcal{C}$ by application of Rcv_3 and this is impossible. Thus $[\![t_2]\!]_\ell \xrightarrow{\langle(\mathcal{C}^*,\eta)\rangle} [\![t'_2]\!]_\ell$ implies that t_2 must have a subterm in the form of $\alpha.t'_2$. Therefore with a same discussion, $\ell : t : t_2 \xrightarrow{\langle(\mathcal{C}^*,\eta)\rangle} [\![t'_2]\!]_\ell$ and $[\![t'_1]\!]_\ell \simeq_{br} [\![t'_2]\!]_\ell$.
- It owes to either Sen_3 or Sen_4 as t_1 has a subterm of the form $sense(\ell', t^*_1, t^{**}_1)$. Assume it was derived by Sen_3 (maybe together with $Choice'$, Inv', and $Sen'_{1,2}$), as the other case can be proved with the same argumentation. Thus, the assumptions $\ell : t : t^*_1 \not\xrightarrow{nrcv(\mathfrak{m})}$ and $\ell : t : t^{**}_1 \xrightarrow{(\mathcal{C},nrcv(\mathfrak{m}))} [\![t^{**\prime}_1]\!]_\ell$, where $\mathcal{C}^* = \{\ell' \rightsquigarrow \ell\} \cup \mathcal{C}_1$, $\eta^* = nrcv(\mathfrak{m})$ and $t^* = t$ hold. These together imply that $sense(\ell', t^*_1, t^{**}_1) \not\xrightarrow{(\{?\rightsquigarrow\ell'\},rcv(\mathfrak{m}))}$ and hence it can be concluded that $t_1 \not\xrightarrow{(\{?\rightsquigarrow\ell'\}\cup\mathcal{C}_1[?/\ell],rcv(\mathfrak{m}))}$. Therefore, by application of Rcv_3, $[\![t_1]\!]_\ell \xrightarrow{(\mathcal{C}^*,nrcv(\mathfrak{m}))} [\![t_1]\!]_\ell$, and by application of the rules Sen_2 and Rcv_1 (maybe together with $Choice$, Inv, and $Sen_{1,2}$), it can be derived that $[\![t_1]\!]_\ell \xrightarrow{(\{\ell'\not\rightsquigarrow\ell\}\cup\mathcal{C}\cup\mathcal{C}_1[\ell/?],nrcv(\mathfrak{m}))} [\![t^{**\prime}_1]\!]_\ell$. The assumption $[\![t_1]\!]_\ell \simeq_{rbr} [\![t_2]\!]_\ell$ implies that $[\![t_2]\!]_\ell \xrightarrow{(\{\ell'\not\rightsquigarrow\ell\}\cup\mathcal{C}\cup\mathcal{C}_1[\ell/?],nrcv(\mathfrak{m}))} [\![t^{**\prime}_2]\!]_\ell$, $[\![t^{**\prime}_1]\!]_\ell \simeq_{br} [\![t^{**\prime}_2]\!]_\ell$, and $[\![t_2]\!]_\ell \xrightarrow{(\mathcal{C}^*,nrcv(\mathfrak{m}))} [\![t_2]\!]_\ell$ hold. Due to the root condition, the negative pair $\ell \not\rightsquigarrow \ell'$ can be only derived when t_2 has a subterm of the form $sense(\ell', t^*_2, t^{**}_2)$, where $t^*_2 \not\xrightarrow{rcv(\mathfrak{m})}$, $t^{**}_2 \xrightarrow{(\mathcal{C},rcv(\mathfrak{m}))} t^{**\prime}_2$. Consequently $\ell : t : t^*_2 \not\xrightarrow{nrcv(\mathfrak{m})}$ and $\ell : t : t^{**}_2 \xrightarrow{(\mathcal{C},nrcv(\mathfrak{m}))} [\![t^{**}_2]\!]_\ell$. By application of Sen_3, $\ell : t : t_2 \xrightarrow{(\mathcal{C}^*,nrcv(\mathfrak{m}))} t$ is achieved.

Case 7. We prove that if $t_1 \simeq_{br} t_2$ then $(\nu\ell)t_1 \simeq_{br} (\nu\ell)t_2$. Let $t_1 \simeq_{br} t_2$ be witnessed by the branching reliable computed network bisimulation relation $\mathcal{R}$. We define $\mathcal{R}' = \{((\nu\ell)t'_1, (\nu\ell)t'_2) | (t'_1, t'_2) \in \mathcal{R}\}$. We prove that $\mathcal{R}'$ is a branching

reliable computed network bisimulation relation. Suppose $(\nu\ell)t_1' \xrightarrow{(\mathcal{C}',\eta')} (\nu\ell)t_1''$ results from the application of Hid on $t_1' \xrightarrow{(\mathcal{C},\eta)} t_1''$. Since $(t_1', t_2') \in \mathcal{R}$, there are two cases; in the first case η is a τ action and $(t_1'', t_2') \in \mathcal{R}$, consequently $((\nu\ell)t_1'', (\nu\ell)t_2') \in \mathcal{R}'$. In second case there are $s_1''', \ldots s_k'''$ and $s_1'', \ldots, s_k''$ for some $k > 0$ such that $\forall i \leq k\,(t_2' \Rightarrow s_i''' \xrightarrow{\langle(\mathcal{C}_i,\eta)\rangle} s_i''$ with $(t_1', s_i'''), (t_1'', s_i'') \in \mathcal{R})$, and $\langle\mathcal{C}_1\rangle \ldots, \langle\mathcal{C}_k\rangle$ is a partitioning of $\langle\mathcal{C}\rangle$. By application of Hid, $\forall i \leq k\,((\nu\ell)t_2' \Rightarrow (\nu\ell)s_i'''$ with $((\nu\ell)t_1', (\nu\ell)s_i''') \in \mathcal{R}')$. There are two cases to consider:

- $\langle(\mathcal{C}_i,\eta)\rangle = (\mathcal{C}_i,\eta)$: Consequently $(\nu\ell)s_i''' \xrightarrow{(\mathcal{C}_i',\eta')} (\nu\ell)s_i''$ where $(\mathcal{C}_i',\eta') = (\mathcal{C}_i,\eta)[?/\ell]$.
- $\langle(\mathcal{C}_i,\eta)\rangle \neq (\mathcal{C}_i,\eta)$: in this case η is of the form $nsnd(\mathfrak{m},?)$, $\eta' = \eta$, and $\mathcal{C}_i' = \mathcal{C}_i[?/\ell]$. If $\langle(\mathcal{C}_i,\eta)\rangle = (\mathcal{C}_i,\eta)[\ell/?]$ then $\langle(\mathcal{C}_i,\eta)\rangle[?/\ell] = (\mathcal{C}_i',\eta')$ holds, otherwise $\langle(\mathcal{C}_i,\eta)\rangle = (\mathcal{C}_i,\eta)[\ell'/?]$, where $\ell' \neq \ell$, and hence $\langle(\mathcal{C}_i,\eta)\rangle[?/\ell]$ is a counterpart of $(\mathcal{C}_i',\eta')$. Consequently $(\nu\ell)s_i''' \xrightarrow{\langle(\mathcal{C}_i',\eta')\rangle} (\nu\ell)s_i''$.

Owing to the fact that a subset of $\mathcal{C}_1[?/\ell], \ldots, \mathcal{C}_k[?/\ell]$ constitutes a partitioning of $\mathcal{C}[\ell/?]$, and according to the discussion above, there are $s_1''', \ldots, s_j'''$ and $s_1'', \ldots, s_j''$ for some $j \leq k$ such that $\forall i \leq j$, $(\nu\ell)t_2' \Rightarrow (\nu\ell)s_i''' \xrightarrow{\langle(\mathcal{C}_i',\eta')\rangle} (\nu\ell)s_i''$ with $((\nu\ell)t_1', (\nu\ell)s_i'''), ((\nu\ell)t_1'', (\nu\ell)s_i'') \in \mathcal{R}')$, and $\langle\mathcal{C}_1'\rangle, \ldots, \langle\mathcal{C}_j'\rangle$ is a partitioning of $\langle\mathcal{C}'\rangle$.

Likewise we can prove that $t_1 \simeq_{rbr} t_2$ implies $(\nu\ell)t_1 \simeq_{rbr} (\nu\ell)t_2$. To this aim we examine the root condition in Definition 3.3. Suppose $(\nu\ell)t_1 \xrightarrow{(\mathcal{C}',\eta')} (\nu\ell)t_1'$. With the same argument as above, $(\nu\ell)t_2 \xrightarrow{\langle(\mathcal{C}',\eta')\rangle} (\nu\ell)t_2'$. Since $t_1' \simeq_{br} t_2'$, we proved that $(\nu\ell)t_1' \simeq_{br} (\nu\ell)t_2'$. Concluding $(\nu\ell)t_1 \simeq_{rbr} (\nu\ell)t_2$.

Case 10. From the three remaining cases, we focus on the most challenging case, which is the communication merge operator $|$, as the other operators are proved in a similar way. First we prove that if $t_1 \simeq_{br} t_2$, then $t_1 \parallel t \simeq_{br} t_2 \parallel t$. Let $t_1 \simeq_{br} t_2$ be witnessed by the branching reliable computed network bisimulation relation $\mathcal{R}$. We define $\mathcal{R}' = \{(t_1' \parallel t', t_2' \parallel t') \mid (t_1', t_2') \in \mathcal{R},\ t' \text{ any computed network term}\}$. We prove that $\mathcal{R}'$ is a branching reliable computed network bisimulation relation. Suppose $t_1' \parallel t \xrightarrow{(\mathcal{C}^*,\eta)} t^*$. There are several cases to consider:

- Suppose η is of the form $nsnd(\mathfrak{m},\ell)$. First let it be performed by t_1', and t participated in the communication. That is, $t_1' \xrightarrow{(\mathcal{C}_1,nsnd(\mathfrak{m},\ell))} t_1''$ and $t \xrightarrow{(\mathcal{C},nrcv(\mathfrak{m}))} t'$ give rise to the transition $t_1' \parallel t \xrightarrow{(\mathcal{C}_1\cup C[\ell/?],nsnd(\mathfrak{m},\ell))} t_1'' \parallel t'$. As $(t_1', t_2') \in \mathcal{R}$ and $t_1' \xrightarrow{(\mathcal{C}_1,nsnd(\mathfrak{m},\ell))} t_1''$, there are $s_1''', \ldots, s_k'''$ and $s_1'', \ldots, s_k''$ for some $k > 0$ such that $\forall i \leq k\,(t_2' \Rightarrow s_i''' \xrightarrow{(\mathcal{C}_{1_i}[\ell'/\ell],nsnd(\mathfrak{m},\ell'))}$

s_i'', where $(\ell =?\vee\ell = \ell')$, with (t_1', s_i'''), $(t_1'', s_i'') \in \mathcal{R})$, and $\mathcal{C}_{1_1}[\ell'/\ell], \ldots, \mathcal{C}_{1_k}[\ell'/\ell]$ is a partitioning of $\mathcal{C}_1[\ell'/\ell]$.

Hence $\forall i \leq k\,(t_2' \parallel t \Rightarrow s_i''' \parallel t \xrightarrow{((\mathcal{C}_{1_i}[\ell'/\ell]\cup C)[\ell'/?], nsnd(\mathfrak{m},\ell'))} s_i'' \parallel t'$ with $(t_1' \parallel t, s_i''' \parallel t), (t_1'' \parallel t', s_i'' \parallel t') \in \mathcal{R}')$, and $(\mathcal{C}_{1_1}[\ell'/\ell]\cup\mathcal{C})[\ell'/?], \ldots, (\mathcal{C}_{1_k}[\ell'/\ell]\cup\mathcal{C})[\ell'/?]$ is a partitioning of $(\mathcal{C}_1[\ell'/\ell] \cup \mathcal{C})[\ell'/?]$.

Now suppose that the send action was performed by t, and t_1' participated in the communication. That is, $t_1' \xrightarrow{(\mathcal{C}_1, nrcv(\mathfrak{m}))} t_1''$ and $t \xrightarrow{(\mathcal{C}, nsnd(\mathfrak{m},\ell))} t'$ give rise to the transition $t_1' \parallel t \xrightarrow{(\mathcal{C}_1\cup\mathcal{C}[\ell/?], nsnd(\mathfrak{m},\ell))} t_1'' \parallel t'$. Since $(t_1', t_2') \in \mathcal{R}$ and $t_1' \xrightarrow{(\mathcal{C}_1, nrcv(\mathfrak{m}))} t_1''$, there are $s_1''', \ldots, s_k'''$ and $s_1'', \ldots, s_k''$ for some $k > 0$ such that $\forall i \leq k\,(t_2' \Rightarrow s_i''' \xrightarrow{(\mathcal{C}_{1_i}, nrcv(\mathfrak{m}))} s_i''$ with $(t_1', s_i'''), (t_1'', s_i'') \in \mathcal{R})$, and $\mathcal{C}_{1_1}, \ldots, \mathcal{C}_{1_k}$ is a partitioning of $\mathcal{C}_1$. Therefore, $\forall i \leq k\,(t_2' \parallel t \Rightarrow s_i''' \parallel t \xrightarrow{(\mathcal{C}_{1_i}\cup\mathcal{C}[\ell/?], nsnd(\mathfrak{m},\ell))} s_i'' \parallel t'$, and $(t_1' \parallel t, s_i''' \parallel t), (t_1'' \parallel t', s_i'' \parallel t') \in \mathcal{R}')$ and $\mathcal{C}_{1_1} \cup\mathcal{C}[\ell/?], \ldots, \mathcal{C}_{1_k} \cup\mathcal{C}[\ell/?]$ constitute a partitioning of $\mathcal{C}_1 \cup\mathcal{C}[\ell/?]$.

- The case where η is a receive action is proved in a similar way to the previous case.
- Suppose η is a τ action. Assume it originates from t_1 by application of Par. Thus $t_1' \xrightarrow{(\mathcal{C},\tau)} t_1''$ and $(t_1', t_2') \in \mathcal{R}$ implies: either $(t_1'', t_2') \in \mathcal{R}$ and consequently $(t_1'' \parallel t, t_2' \parallel t) \in \mathcal{R}'$, or there are $s_1''', \ldots, s_k'''$ and $s_1'', \ldots, s_k''$ for some $k > 0$ such that $\forall i \leq k\,(t_2' \Rightarrow s_i''' \xrightarrow{(\mathcal{C}_i,\tau)} s_i''$ with $(t_1', s_i'''), (t_1'', s_i'') \in \mathcal{R})$, and $\mathcal{C}_1, \ldots, \mathcal{C}_k$ constitute a partitioning of $\mathcal{C}$. Therefore, $\forall i \leq k\,(t_2' \parallel t \Rightarrow s_i''' \parallel t \xrightarrow{(C_i,\tau)} s_i'' \parallel t'$, and $(t_1' \parallel t, s_i''' \parallel t), (t_1'' \parallel t', s_i'' \parallel t') \in \mathcal{R}')$. The case when $t \xrightarrow{(\mathcal{C},\tau)} t'$ implies $t_1' \parallel t \xrightarrow{(\mathcal{C},\tau)} t_1' \parallel t'$ by application of Par is straightforward.
- The case when η is an internal action is easy to prove (similar to the second case of the previous case).

Likewise we can prove that $t_1 \simeq_{rbr} t_2$ implies $t \parallel t_1 \simeq_{rbr} t \parallel t_2$.

Now let $t_1 \simeq_{rbr} t_2$. To prove $t_1 \mid t \simeq_{rbr} t_2 \mid t$, we examine the root condition from Definition 3.3. Suppose $t_1 \mid t \xrightarrow{(\mathcal{C}^*, nsnd(\mathfrak{m},\ell))} t^*$. There are two cases to consider:

- This send action was performed by t_1 at node ℓ, and t participated in the communication. That is, $t_1 \xrightarrow{(\mathcal{C}_1, nsnd(\mathfrak{m},\ell))} t_1'$ and $t \xrightarrow{(\mathcal{C}, nrcv(\mathfrak{m}))} t'$, so that $t_1 \mid t \xrightarrow{(\mathcal{C}_1\cup\mathcal{C}[\ell/?], nsnd(\mathfrak{m},\ell))} t_1' \parallel t'$. Since $t_1 \simeq_{rbr} t_2$, there is a t_2' such that $t_2 \xrightarrow{(\mathcal{C}_1, nsnd(\mathfrak{m},\ell'))} t_2'$ with $(\ell =? \vee \ell = \ell')$ and $t_1' \simeq_{br} t_2'$. Then

$t_2 \mid t \xrightarrow{(\mathcal{C}_1 \cup \mathcal{C}[\ell'/?], nsnd(\mathfrak{m},\ell))} t_2' \parallel t'$. Since $t_1' \simeq_{br} t_2'$, we proved that $t_1' \parallel t' \simeq_{br} t_2' \parallel t'$.

- The send action was performed by t at node ℓ, and t_1 participated in the communication. That is, $t_1 \xrightarrow{(\mathcal{C}_1, nrcv(\mathfrak{m}))} t_1'$ and $t \xrightarrow{(\mathcal{C}, nsnd(\mathfrak{m},\ell))} t'$, so that $t_1 \mid t \xrightarrow{(\mathcal{C}_1 \cup \mathcal{C}[\ell/?], nsnd(\mathfrak{m},\ell))} t_1' \parallel t'$. Since $t_1 \simeq_{rbr} t_2$, there is a t_2' such that $t_2 \xrightarrow{(\mathcal{C}_1, nrcv(\mathfrak{m}))} t_2'$ with $t_1' \simeq_{br} t_2'$. Then $t_2 \mid t \xrightarrow{(\mathcal{C}_1 \cup \mathcal{C}[\ell/?], nsnd(\mathfrak{m},\ell))} t_2' \parallel t'$. Since $t_1' \simeq_{br} t_2'$, we have $t_1' \parallel t' \simeq_{br} t_2' \parallel t'$.

Finally, the case where $t_1 \mid t \xrightarrow{(\mathcal{C}^*, nrcv(\mathfrak{m}))} t^*$ can be easily dealt with. This receive action was performed by both t_1 and t.

Concluding, $t_1 \mid t \simeq_{rbr} t_2 \mid t$. Likewise it can be argued that $t \mid t_1 \simeq_{rbr} t \mid t_2$.

Case 11. We prove that if $t_1 \simeq_{br} t_2$, then $\partial_M(t_1) \simeq_{br} \partial_M(t_2)$. Let $t_1 \simeq_{br} t_2$ be witnessed by the branching reliable computed network bisimulation relation $\mathcal{R}$. We define $\mathcal{R}' = \{(\partial_M(t_1'), \partial_M(t_2')) \mid (t_1', t_2') \in \mathcal{R}\}$. We prove that $\mathcal{R}'$ is a branching reliable computed network bisimulation relation. Suppose that $\partial_M(t_1') \xrightarrow{(\mathcal{C},\eta)} \partial_M(t_1'')$ results from the application of $Encap$ on $t_1' \xrightarrow{(\mathcal{C},\eta)} t_1''$ such that $\eta \neq nrcv(\mathfrak{m}) \vee isType_m(\mathfrak{m}) = F$. Since $(t_1', t_2') \in \mathcal{R}$, two cases can be considered: either η is a τ action and $(t_1'', t_2') \in \mathcal{R}$, or there are $s_1''', \ldots, s_k'''$ and $s_1'', \ldots, s_k''$ for some $k > 0$ such that $\forall i \leq k\, (t_2' \Rightarrow s_i''' \xrightarrow{\langle(\mathcal{C}_i,\eta)\rangle} s_i''$ with $(t_1', s_i'''), (t_1'', s_i'') \in \mathcal{R})$ and $\langle\mathcal{C}_1\rangle, \ldots, \langle\mathcal{C}_k\rangle$ is a partitioning of $\langle\mathcal{C}\rangle$. In the former case, $(\partial_M(t_1''), \partial_M(t_2')) \in \mathcal{R}'$. In the latter case, by application of Par and $Encap$, $\forall i \leq k\, (\partial_M(t_2') \Rightarrow \partial_M(s_i''') \xrightarrow{\langle(\mathcal{C}_i,\eta)\rangle} \partial_M(t_2'')$ with $(\partial_M(t_1'), \partial_M(s_i''')), (\partial_M(t_1''), \partial_M(s_i'')) \in \mathcal{R}')$.

Likewise we can prove that $t_1 \simeq_{rbr} t_2$ implies $\partial_M(t_1) \simeq_{rbr} \partial_M(t_2)$. To this aim we examine the root condition in Definition 3.3. Suppose $\partial_M(t_1) \xrightarrow{(\mathcal{C},\eta)} \partial_M(t_1')$. With the same argument as above, $\partial_M(t_2) \xrightarrow{\langle(\mathcal{C},\eta)\rangle} \partial_M(t_2')$. Since $t_1' \simeq_{br} t_2'$, we proved that $\partial_M(t_1') \simeq_{br} \partial_M(t_2')$. Concluding $\partial_M(t_1) \simeq_{rbr} \partial_M(t_2)$.

Case 13 Suppose that $\mathcal{C} \rhd t_1 \xrightarrow{(\mathcal{C}' \cup \mathcal{C},\eta)} t_1'$ by application of TR since $t_1 \xrightarrow{(\mathcal{C}',\eta)} t_1'$. By assumption $t_1 \simeq_{rbr} t_2$ implies that $t_2 \xrightarrow{(\mathcal{C}',\eta)} t_2'$ and $t_1' \simeq_{br} t_2'$. Therefore, by application of TR, $\mathcal{C} \rhd t_2 \xrightarrow{(\mathcal{C}' \cup \mathcal{C},\eta)} t_2'$, and $t_1' \simeq_{br} t_2'$ concludes that $\mathcal{C} \rhd t_1 \simeq_{rbr} \mathcal{C} \rhd t_2$.

□

A.3 Soundness of *RCNT* Axiomatization

As two rooted branching computed network bisimilar terms are also rooted branching reliable computed network bisimilar, the soundness of axioms which are in common with the lossy setting are established [73]. Thus, to prove the soundness of our axiomatization, it suffices to prove the soundness of each new axiom in

comparison with the lossy setting, i.e., Dep_{0-7}, $TRes_{1-5}$, $LM'_{1,2}$, and T_1, modulo rooted branching reliable computed network bisimilarity.

We focus on the soundness of Dep_0 and T_1, as the soundness of the remaining axioms can be argued in a similar fashion. To prove Dep_0, we show that both sides of the axiom satisfy the transfer conditions of Definition 3.3. In following cases, for the sake of brevity, we write X for $rec\mathfrak{Q}\cdot\sum_{\mathfrak{m}'\notin Message(t,\emptyset)}(\{\}, nrcv(\mathfrak{m}')).\mathfrak{Q}+\ell:\mathfrak{Q}:t$. Assume that $[\![t]\!]_\ell \xrightarrow{(\mathcal{C}^*,\eta)} [\![t']\!]_\ell$. Two cases can be distinguished:

1. It owes to the application $Prefix'$ and $Snd/Rcv_{1,2}$ together with some of the rules $Choice$, Inv, and $Sen_{1,2}$. Therefore t has a subterm of the form of $\alpha.t'$ where η is the network action version of α and $\mathcal{C}^* \equiv \mathcal{C}_1^* \cup \mathcal{C}_2^*$ such that $\mathcal{C}_1^*$ is derived by $Snd/Rcv_{1,2}$ and $\mathcal{C}_2^*$ is derived by $Sen_{1,2}$ if they are applied. By application of $Inter'_1/Inter'_{2,3}$ together with some of the rules $Choice'$, Inv', and $Sen'_{1,2}$, $\ell : X : t \xrightarrow{(\mathcal{C}^*,\eta)} [\![t']\!]_\ell$. Then, by application of Rec and $Choice$, $X \xrightarrow{(\mathcal{C}^*,\eta)} [\![t']\!]_\ell$.

2. It owes to the application of Rcv_3 since $t \not\xrightarrow{(\mathcal{C},rcv(\mathfrak{m}))}$, where $\mathcal{C}^* = \mathcal{C}[\ell/?]$, and there exists no t' such that $t \not\xrightarrow{(\mathcal{C}',\ rcv(\mathfrak{m}))} t' \wedge \mathcal{C}' \preccurlyeq \mathcal{C}$. Two cases can be distinguished:

   - Assume that $m \notin Message(t,\emptyset)$. Thus, by application of Rec, $Choice$ and $Prefix$, $X \xrightarrow{(\mathcal{C},nrcv(\mathfrak{m}))} X$, where $\mathcal{C} = \{\}$.
   - Assume that $m \in Message(t,\emptyset)$ and consequently $t \xrightarrow{(\mathcal{C}'',rcv(\mathfrak{m}))} t'$ for some t'. This only happens when t has a subterm of the form of $sense(\ell', t_1, t_2)$ for some t_1 and t_2. Assume that $t_1 \xrightarrow{(\mathcal{C}'''[?/\ell],rcv(\mathfrak{m}))} t'_1$, where $\{?\rightsquigarrow\ell'\}\cup\mathcal{C}'''[?/\ell]\cup\mathcal{C}_1^*[?/\ell] = \mathcal{C}''$ and $t_2 \not\xrightarrow{(\{\},\ rcv(\mathfrak{m}))}$. Thus, $t \not\xrightarrow{(\{?\rightsquigarrow\ell'\}\cup\mathcal{C}_1^*[?/\ell],rcv(\mathfrak{m}))}$ and $\mathcal{C} \equiv \{?\rightsquigarrow\ell'\}\cup\mathcal{C}_1^*[?/\ell]$. Therefore, $\ell : X : t_1 \xrightarrow{(\mathcal{C}'''\cup\{\ell\rightsquigarrow\ell'\},nrcv(\mathfrak{m}))} [\![t'_1]\!]_\ell$ and $\ell : X : t_2 \not\xrightarrow{nrcv(\mathfrak{m})}$, and by application of Sen_4 (and maybe together with $Choice'$, Inv', and $Sen'_{1,2}$), $\ell : X : t \xrightarrow{(\{\ell\not\rightsquigarrow\ell'\}\cup\mathcal{C}_1^*,nrcv(\mathfrak{m}))} X$. The case $t_2 \xrightarrow{(\mathcal{C}'''[?/\ell],rcv(\mathfrak{m}))} t'_2$, where $\{?\rightsquigarrow\ell'\}\cup\mathcal{C}'''[?/\ell]\cup\mathcal{C}_1^*[?/\ell] = \mathcal{C}''$, and $t_1 \not\xrightarrow{(\{\},\ rcv(\mathfrak{m}))}$ hold, is proved with a similar discussion with application of Sen_3.

We focus on the soundness of T_1. The only transition that the terms $(\mathcal{C}',\eta).(\ (\mathcal{C}_1,\eta).t + (\mathcal{C}_2,\eta).t + t')$ and $(\mathcal{C}',\eta).((\mathcal{C},\eta).t + t')$ in T_1 can do is $\xrightarrow{(\mathcal{C}',\eta)}$ and the resulting terms $(\mathcal{C}_1,\eta).t + (\mathcal{C}_2,\eta).t + t'$ and $(\mathcal{C},\eta).t + t'$ are branching reliable computed network bisimilar, witnessed by the relation $\mathcal{R}$ constructed as follows:

$$\mathcal{R} = \{((\mathcal{C}_1,\eta).t + (\mathcal{C}_2,\eta).t + t', (\mathcal{C},\eta).t + t'), (t,t) \mid t \in RCNT\}$$

The pair $((\mathcal{C}_1,\eta).t + (\mathcal{C}_2,\eta).t + t', (\mathcal{C},\eta).t + t')$ satisfies the transfer conditions of Definition 3.1. Because every initial transition that $(\mathcal{C}_1,\eta).t + (\mathcal{C}_2,\eta).t + t'$ can perform owing to t', $(\mathcal{C},\eta).t + t'$ can perform too. If $(\mathcal{C}_1,\eta).t + (\mathcal{C}_2,\eta).t + t'$ can perform a $(\mathcal{C}_1,\eta)$ or $(\mathcal{C}_2,\eta)$-transition, $(\mathcal{C},\eta).t + t'$ can also perform it by application of Exe. Vice versa, if $(\mathcal{C},\eta).t + t'$ can perform a $(\mathcal{C},\eta)$-transition, then as $\mathcal{C}_1$ and $\mathcal{C}_2$ form a partitioning of $\mathcal{C}$, $(\mathcal{C}_1,\eta).t + (\mathcal{C}_2,\eta).t + t'$ can perform a corresponding $(\mathcal{C}_1,\eta)$- or $(\mathcal{C}_2,\eta)$-transition.

A.4 Completeness of *RCNT* Axiomatization

To define *RCNT* terms with a finite-state behavior, we borrow the syntactical restriction of [73] on recursive terms $rec\mathfrak{A} \cdot t$, following the approach of [9]. We consider so-called *finite-state Reliable Computed Network Theory* (*RCNT*$_f$), obtained by restricting recursive terms $rec\mathfrak{A} \cdot t$ to those that of which the bound network names do not occur in the scope of parallel, communication merge, left merge, hide, encapsulation and abstraction operators in t.

We follow the corresponding proof of [73] to prove the completeness of our axiomatization by performing the following steps:

1. first we show that each *RCNT*$_f$ term can be turned into a *normal form* consisting of only $0, (\mathcal{C},\eta).t', t' + t''$ and $rec\mathfrak{A} \cdot t'$, where $\mathfrak{A}$ is guarded in t';
2. next we define *recursive network specifications* and prove that each guarded recursive network specification has a unique solution;
3. finally we show that our axiomatization is ground-complete for normal forms, by showing that equivalent normal forms are solutions for the same guarded recursive network specification.

Completeness of our axiomatization for all *RCNT*$_f$ terms results from the steps 1 and 3. We only discuss the first step, as others are exactly the same as in the lossy setting.

Proposition A.10. *Each closed term t of RCNT$_f$ whose network names do not occur in the scope of one of the operators $\|, \between, |, (\nu\ell), \tau_M$ or ∂_M for some $\ell \in Loc$ and $M \subseteq Msg$, can be turned into a normal form.*

We prove this by structural induction over the syntax of terms t (possibly open). The base cases of induction for $t \equiv 0$ or $t \equiv \mathfrak{A}$ are trivial because they are in normal form already. The inductive cases of the induction are the following ones:

- if $t \equiv [\![0]\!]_\ell$, then by application of $Dep_{0,4}$ and Ch_1 we have $t = rec\mathfrak{Q} \cdot \sum_{\mathfrak{m}' \notin Msg}(\{\}, nrcv(\mathfrak{m}')).\mathfrak{Q}$, which is in normal form.

- if $t \equiv [\![\alpha.t']\!]_\ell$ or $t \equiv [\![t' + t'']\!]_\ell$ or $[\![sense(\ell', t', t'')]\!]_\ell$ or $[\![\mathfrak{A}]\!]_\ell$, then t can be turned into a normal form by application of axioms $Dep_{0-5,6,7}$ and induction over $[\![t']\!]_\ell$ and $[\![t'']\!]_\ell$.
- if $t \equiv (\mathcal{C}, \eta).t'$ or $t \equiv t' + t''$, then t can be turned into normal form by induction over t' and t''.
- the other cases can be treated in the same way as in [73].

A.5 Proofs of Section 3.5.2

We first prove Theorem 3.7 which indicates that the refinement relation is a preorder relation and has the precongruence property, and then we discuss the proof of Proposition 3.8.

A.5.1 Proof of Theorem 3.7

We first show that the refinement relation is a preorder relation and then discuss its precongruence property. To prove that refinement is a preorder, we must show that it is reflexive and transitive. As it is trivial that Definition 3.6 is reflexive, we focus on its transitivity property.

Regrading the well-formedness conditions imposed on *RCNT* terms, the transitivity property of our refinement relation, i.e., $t_1 \sqsubseteq t_2$ and $t_2 \sqsubseteq s$ implies that $t_1 \sqsubseteq s$, can be only proved when t_1 and t_2 have no prefixed-actions with a multi-hop network constraint. For such terms, Definition 3.6 enforces they mimic the behavior of each other by the first and third conditions.In other words, for reliable computed network terms with no prefixed-actions with multi-hop network constraints, a relation which is strong bisimulation of [118] is also a refinement.

Lemma A.11 (Transitive property). *$t_1 \sqsubseteq t_2$ and $t_2 \sqsubseteq s$ implies that $t_1 \sqsubseteq s$.*

Proof. Assume sets of refinement relations $\mathcal{R}^1_{\mathcal{C}}$ and $\mathcal{R}^2_{\mathcal{C}}$ witnessing $t_1 \sqsubseteq t_2$ and $t_2 \sqsubseteq s$, respectively. We construct a set of refinement relations $\mathcal{R}'_{\mathcal{C}} = \{(t'_1, s') \mid (t'_2, s') \in \mathcal{R}^2_{\mathcal{C}} \wedge t'_1 \, \mathcal{R}^1_{\mathcal{C}} \, t_2\}$ for any well-formed network constraint $\mathcal{C}$. We show that $t'_1 \mathcal{R}'_{\mathcal{C}} s'$ satisfies the transfer conditions of Definition 3.6.

Assume $t'_1 \xrightarrow{(\mathcal{C}', \eta)} t''_1$ where $\mathcal{C} \cup \mathcal{C}' \in \mathbb{C}^v(Loc)$. By assumption $t'_1 \mathcal{R}^1_{\mathcal{C}} t'_2$ implies that $t'_2 \xrightarrow{(\mathcal{C}', \eta)} t''_2$ such that $t''_1 \mathcal{R}^1_{\mathcal{C} \cup \mathcal{C}'} t''_2$. By the assumption $t'_2 \mathcal{R}^2_{\mathcal{C}} s'$, there are three cases to consider:

- $\eta = \tau$ and $t''_2 \; \mathcal{R}^2_{\mathcal{C} \cup \mathcal{C}'} \; s'$. Thus by construction, $t''_1 \; \mathcal{R}'_{\mathcal{C} \cup \mathcal{C}'} \; s'$.
- There is an s'' such that $s' \xrightarrow{(\mathcal{C}, \eta)} s''$, and $t''_2 \; \mathcal{R}^2_{\mathcal{C} \cup \mathcal{C}'} \; s''$. Thus by construction, $t''_1 \; \mathcal{R}'_{\mathcal{C} \cup \mathcal{C}'} \; s'$.

- $\eta = \iota$ for some $\iota \in IAct \cup \{\tau\}$ and there is an s'' such that $s' \xrightarrow{(\mathcal{M},\iota)} s''$ with $\mathcal{C} \cup \mathcal{C}' \models \mathcal{M}$ and $t_2'' \; \mathcal{R}^2_{\mathcal{C}\cup\mathcal{C}'} \; s''$. Thus by construction, $t_1'' \; \mathcal{R}'_{\mathcal{C}\cup\mathcal{C}'} \; s''$.

Assume $s' \xrightarrow{(\mathcal{C}',\eta)} s''$. The assumption $t_2' \; \mathcal{R}^2_{\mathcal{C}\cup\mathcal{C}'} \; s'$ implies that there is a t_2'' such that $t_2' \xrightarrow{(\mathcal{C}',\eta)} t_2''$ with $t_2'' \; \mathcal{R}^2_{\mathcal{C}\cup\mathcal{C}'} \; s''$. By assumption $t_1' \mathcal{R}^1_{\mathcal{C}} t_2'$ implies that $t_1' \xrightarrow{(\mathcal{C}',\eta)} t_1''$ such that $t_1'' \mathcal{R}^1_{\mathcal{C}\cup\mathcal{C}'} t_2''$, and consequently $t_1'' \; \mathcal{R}'_{\mathcal{C}\cup\mathcal{C}'} \; s''$.

Assume $s' \xrightarrow{(\mathcal{M},\iota)} s''$. . The assumption $t_2' \; \mathcal{R}^2_{\mathcal{C}\cup\mathcal{C}'} \; s'$ implies that there are t_2''' and t_2'' such that $t_2' \overset{\mathcal{C}'}{\Rightarrow} t_2''' \xrightarrow{(\mathcal{C}'',\iota)} t_2''$ with $t_2''' \; \mathcal{R}^2_{\mathcal{C}\cup\mathcal{C}'} \; s'$ and $t_2'' \; \mathcal{R}^2_{\mathcal{C}\cup\mathcal{C}'\cup\mathcal{C}''} \; s''$ where $\mathcal{C} \cup \mathcal{C}' \cup \mathcal{C}'' \models \mathcal{M}$. As every transition of t_2' is mimicked by t_1', there are t_1''' and t_1'' such that $t_1' \overset{\mathcal{C}'}{\Rightarrow} t_1''' \xrightarrow{(\mathcal{C}'',\iota)} t_1''$ with $t_1''' \; \mathcal{R}^1_{\mathcal{C}\cup\mathcal{C}'} \; t_2'''$ and $t_1'' \; \mathcal{R}^1_{\mathcal{C}\cup\mathcal{C}'\cup\mathcal{C}''} \; t_2''$. Concluding, there are t_1''' and t_1'' such that $t_1' \overset{\mathcal{C}'}{\Rightarrow} t_1''' \xrightarrow{(\mathcal{C}'',\iota)} t_1''$ with $t_1''' \; \mathcal{R}'_{\mathcal{C}\cup\mathcal{C}'} \; s'$ and $t_1'' \; \mathcal{R}'_{\mathcal{C}\cup\mathcal{C}'\cup\mathcal{C}''} \; s''$ where $\mathcal{C} \cup \mathcal{C}' \cup \mathcal{C}'' \models \mathcal{M}$. □

Theorem A.12. *Refinement is a precongruence for terms with respect to the RCNT operators.*

Proof. Assume that $t_1 \sqsubseteq s_1$ and $t_2 \sqsubseteq s_2$. We first show that $t_1 + t_2 \sqsubseteq s_1 + s_2$. There are sets of refinement relations $\mathcal{R}^1_{\mathcal{C}}$ and $\mathcal{R}^2_{\mathcal{C}}$ witnessing $t_1 \sqsubseteq s_1$ and $t_2 \sqsubseteq s_2$, respectively. We construct a set of refinement relations $\mathcal{R}_{\mathcal{C}} = \mathcal{R}^1_{\mathcal{C}} \cup \mathcal{R}^2_{\mathcal{C}} \cup \{(t_1', s_1 + s_2) \mid t_1' \; \mathcal{R}^1_{\mathcal{C}} \; s_1\} \cup \{(t_2', s_1 + s_2) \mid t_2' \; \mathcal{R}^2_{\mathcal{C}} \; s_2\}$ for any well-formed network constraint $\mathcal{C}$. We show that $\mathcal{R}_{\{\,\}} = \{(t_1 + t_2, s_1 + s_2)\} \cup \mathcal{R}^1_{\{\,\}} \cup \mathcal{R}^2_{\{\,\}}$ satisfies the transfer conditions of Definition 3.6.

Assume $t_1 + t_2 \xrightarrow{(\mathcal{C}',\eta)} t_1'$ owing to $t_1 \xrightarrow{(\mathcal{C}',\eta)} t_1'$, where $\mathcal{C} \cup \mathcal{C}' \in \mathbb{C}^v(Loc)$. By the assumption $t_1 \; \mathcal{R}^1_{\{\,\}} \; s_1$, three cases can be considered:

- $\eta = \tau$ and $t_1' \; \mathcal{R}_{\mathcal{C}\cup\mathcal{C}'} \; s_1$. Thus by construction $t_1' \; \mathcal{R}_{\mathcal{C}\cup\mathcal{C}'} \; s_1 + s_2$.
- There is an s_1' such that $s_1 \xrightarrow{(\mathcal{C},\eta)} s_1'$, and $t_1' \; \mathcal{R}^1_{\mathcal{C}\cup\mathcal{C}'} \; s_1'$. Thus by the rule *Choice*, there is an s_1' such that $s_1 + s_2 \xrightarrow{(\mathcal{C},\eta)} s_1'$ and by construction $t_1' \; \mathcal{R}_{\mathcal{C}\cup\mathcal{C}'} \; s_1'$.
- $\eta = \iota$ for some $\iota \in IAct \cup \{\tau\}$ and there is an s_1' such that $s_1 \xrightarrow{(\mathcal{M},\iota)} s_1'$ with $\mathcal{C} \cup \mathcal{C}' \models \mathcal{M}$ and $t_1' \; \mathcal{R}^1_{\mathcal{C}\cup\mathcal{C}'} \; s_1'$. Thus by the rule *Choice*, there is an s_1' such that $s_1 + s_2 \xrightarrow{(\mathcal{M},\iota)} s_1'$ and by construction $\mathcal{C} \cup \mathcal{C}' \models \mathcal{M}$ and $t_1' \; \mathcal{R}_{\mathcal{C}\cup\mathcal{C}'} \; s_1'$.

The same discussion holds if $t_1 + t_2 \xrightarrow{(\mathcal{C}',\eta)} t_2'$ owing to $t_2 \xrightarrow{(\mathcal{C}',\eta)} t_2'$.

Assume $s_1 + s_2 \xrightarrow{(\mathcal{M},\iota)} s_1'$ owing to $s_1 \xrightarrow{(\mathcal{M},\iota)} s_1'$, where $\mathcal{C} \cup \mathcal{C}' \in \mathbb{C}^v(Loc)$. By assumption $t_1 \; \mathcal{R}^1_{\mathcal{C}} \; s_1$ implies there are t_1'' and t_1' such that $t_1 \overset{\mathcal{C}'}{\Rightarrow} t_1'' \xrightarrow{(\mathcal{C}'',\iota)} t_1'$ with $t_1'' \; \mathcal{R}^1_{\mathcal{C}\cup\mathcal{C}'} \; s_1$ and $t_1' \; \mathcal{R}^1_{\mathcal{C}\cup\mathcal{C}'\cup\mathcal{C}''} \; s_1'$ where $\mathcal{C} \cup \mathcal{C}' \cup \mathcal{C}'' \models \mathcal{M}$. Consequently $t_1 + t_2 \overset{\mathcal{C}'}{\Rightarrow}$

$t_1'' \xrightarrow{(\mathcal{C}'',\iota)} t_1'$ with $t_1'' \ \mathcal{R}_{\mathcal{C}\cup\mathcal{C}'}\ s_1 + s_2$ and $t_1' \ \mathcal{R}_{\mathcal{C}\cup\mathcal{C}'\cup\mathcal{C}''}\ s_1'$ where $\mathcal{C}\cup\mathcal{C}'\cup\mathcal{C}'' \models \mathcal{M}$. The same discussion holds when $s_1 + s_2 \xrightarrow{(\mathcal{M},\iota)} s_2'$ owing to $s_2 \xrightarrow{(\mathcal{M},\iota)} s_2'$.

Assume $s_1 + s_2 \xrightarrow{(\mathcal{C},\eta)} s_1'$ owing to $s_1 \xrightarrow{(\mathcal{C},\eta)} s_1'$. By assumption $t_1 \ \mathcal{R}^1_{\mathcal{C}}\ s_1$ implies there is a t_1' such that $t_1 \xrightarrow{(\mathcal{C}',\eta)} t_1'$ with $t_1' \ \mathcal{R}^1_{\mathcal{C}\cup\mathcal{C}'}\ s_1'$. Hence, there is a t_1' such that $t_1 + t_2 \xrightarrow{(\mathcal{C}',\eta)} t_1'$ with $t_1' \ \mathcal{R}_{\mathcal{C}\cup\mathcal{C}'}\ s_1'$.

The above discussions together yield $t_1 + t_2 \sqsubseteq s_1 + s_2$.

If s_1 and s_2 have no prefixed-action with a multi-hop network constraint, then we must show the following cases:

1. $(\mathcal{C},\eta).t_1 \sqsubseteq (\mathcal{C},\eta).t_2$;
2. $(\nu\ell).t_1 \sqsubseteq (\nu\ell).t_2$;
3. $t_1 \parallel t_2 \sqsubseteq s_1 \parallel s_2$;
4. $t_1 \between t_2 \sqsubseteq s_1 \between s_2$;
5. $t_1 \mid t_2 \sqsubseteq s_1 \mid s_2$;
6. $\partial_M(t_1) \sqsubseteq \partial_M(t_2)$;
7. $\tau(t_1) \sqsubseteq \tau(t_2)$;
8. $\mathcal{C} \rhd t_1 \sqsubseteq \mathcal{C} \rhd t_2$;

The above cases result from the congruence property of strong bisimilarity. As we discussed earlier, for reliable computed network terms with no prefixed-actions with multi-hop network constraints, a relation which is strong bisimulation of [118] is also a refinement.

□

The proof of Theorem 3.7 is an immediate result of Lemma A.11 and Theorem A.12.

A.5.2 Proof of Proposition 3.8

First we show that $(\mathcal{C},\tau).t \sqsubseteq (\mathcal{M},\iota).s \Rightarrow \mathcal{C} \rhd t \sqsubseteq (\mathcal{M},\iota).s \wedge \mathcal{C} \models \mathcal{M}$. The only transition $(\mathcal{C},\tau).t$ can make is $(\mathcal{C},\tau).t \xrightarrow{(\mathcal{C},\tau)} t$. As $\iota \neq \tau$, according to the first case of the first transfer condition of Definition 3.6, $t \ \mathcal{R}_{\mathcal{C}}\ (\mathcal{M},\iota).s$. We construct $\mathcal{R}'_{\{\ \}} = \mathcal{R}_{\mathcal{C}}$ and show that it induces $\mathcal{C} \rhd t \sqsubseteq (\mathcal{M},\iota).s$. This is trivial as any transition $\mathcal{C} \rhd t \xrightarrow{(\mathcal{C}\cup\mathcal{C}',\eta)} t'$ is the result of $t \xrightarrow{(\mathcal{C}',\eta)} t'$. The transition $s \xrightarrow{(\mathcal{M},\iota)} s'$ and the assumption $t \ \mathcal{R}_{\mathcal{C}}\ (\mathcal{M},\iota).s$ imply that here are t'' and t' such that $t \overset{\mathcal{C}'}{\Rightarrow} t'' \xrightarrow{(\mathcal{C}'',\iota)} t'$ with $\mathcal{C}\cup\mathcal{C}'\cup\mathcal{C}'' \models \mathcal{M}$ and $t' \ \mathcal{R}_{\mathcal{C}\cup\mathcal{C}'\cup\mathcal{C}''}\ s'$ where $\mathcal{C}\cup\mathcal{C}'\cup\mathcal{C}'' \in \mathbb{C}^v(Loc)$. Two cases can be discussed:

- $t'' \equiv t$, and $t \stackrel{\{\,\}}{\Longrightarrow} t \xrightarrow{(\mathcal{C}'',\iota)} t'$ with $\mathcal{C} \cup \mathcal{C}'' \models \mathcal{M}$ and $t' \; \mathcal{R}_{\mathcal{C}\cup\mathcal{C}''} \; s'$ where $\mathcal{C}\cup\mathcal{C}'' \in \mathbb{C}^v(Loc)$. Therefore, $\mathcal{C} \rhd t \stackrel{\{\,\}}{\Longrightarrow} \mathcal{C} \rhd t \xrightarrow{(\mathcal{C}''\cup\mathcal{C},\iota)} t'$ with $\mathcal{C}\cup\mathcal{C}'' \models \mathcal{M}$ and $t' \; \mathcal{R}_{\mathcal{C}\cup\mathcal{C}''} \; s'$ where $\mathcal{C} \cup \mathcal{C}'' \in \mathbb{C}^v(Loc)$;
- $t \stackrel{\mathcal{C}'}{\Rightarrow} t''$ is the result of $n > 0$ τ-transitions. Thus there is t^* such that $t \xrightarrow{(\mathcal{C}^*,\tau)} t^* \stackrel{\mathcal{C}^{**}}{\Longrightarrow} t''$ where $\mathcal{C}^* \cup \mathcal{C}^{**} = \mathcal{C}'$. Hence, $\mathcal{C} \rhd t \xrightarrow{(\mathcal{C}^*\cup\mathcal{C},\tau)} t^* \stackrel{\mathcal{C}^{**}}{\Longrightarrow} t''$. Thus, $\mathcal{C} \rhd t \stackrel{\mathcal{C}'}{\Rightarrow} t'' \xrightarrow{(\mathcal{C}'',\iota)} t'$ with $\mathcal{C}\cup\mathcal{C}'\cup\mathcal{C}'' \models \mathcal{M}$ and $t' \; \mathcal{R}_{\mathcal{C}\cup\mathcal{C}'\cup\mathcal{C}''} \; s'$ where $\mathcal{C} \cup \mathcal{C}' \cup \mathcal{C}'' \in \mathbb{C}^v(Loc)$.

Now, we show that $(\mathcal{C},\iota).t \sqsubseteq (\mathcal{M},\iota).s \Rightarrow \mathcal{C} \rhd t \sqsubseteq s$. The only transition $(\mathcal{C},\iota).t$ can make is $(\mathcal{C},\iota).t \xrightarrow{(\mathcal{C},\iota)} t$. As $\iota \neq \tau$ and $\iota \in IAct$, according to the third case of the first transfer condition of Definition 3.6, $t \; \mathcal{R}_{\mathcal{C}} \; s$. We construct $\mathcal{R}'_{\{\,\}} = \mathcal{R}_{\mathcal{C}}$ and show that it induces $\mathcal{C} \rhd t \sqsubseteq s$. This is trivial as any transition $\mathcal{C} \rhd t \xrightarrow{(\mathcal{C}\cup\mathcal{C}',\eta)} t'$ is the result of $t \xrightarrow{(\mathcal{C}',\eta)} t'$. The reverse of the rule can be argued in a similar fashion.

Proofs of Chapter 6

B

B.1 Proofs of Theorems 6.5, 6.10, and 6.12

We first prove that product line bisimilarity is an equivalence relation, and then prove its congruence property on fully expanded PL-CCS terms. Later, we show that strict strong bisimilarity is an equivalence relation and constitutes a congruence on PL-CCS terms

B.1.1 Proof of Theorem 6.5

Theorem 6.5. Product line bisimilarity is an equivalence relation.
Proof. To show that product line bisimilarity is an equivalence, we must show that it is reflexive, symmetric, and transitive. Reflexivity and symmetry follow immediately from the reflexivity and symmetry properties of strong bisimilarity. Hence, it only remains to prove transitivity.

Consider PL-CCS terms s, t, r such that $s \simeq_{PL} t$, and $t \simeq_{PL} r$. Following Definition 6.4, for any valid full configuration ν_1^f with respect to s, there exists a valid full configuration ν_2^f with respect to t such that $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$. For any ν_2^f, there exists a valid full configuration ν_3^f with respect to r such that $\Pi(t, \nu_2^f) \sim \Pi(r, \nu_3^f)$. Transitivity of strong bisimilarity results in $\Pi(s, \nu_1^f) \sim \Pi(r, \nu_3^f)$. The same argument holds for any valid full configuration ν_3^f with respect to r, concluding that $s \simeq_{PL} r$.

B.1.2 Proof of Theorem 6.12

Theorem 6.12. Product line bisimilarity constitutes a congruence on fully expanded PL-CCS terms.
Proof. Consider an arbitrary product line r such that $s \odot r$ and $t \odot r$ are fully expanded, where $\odot \in \{+, \oplus, \|\}$ and $s \simeq_{PL} t$; also consider renaming function ϕ, $L \subseteq Act$; to show that product line bisimilarity is a congruence on fully expanded PL-CCS terms we need to prove the following statements:

1. $\alpha.s \simeq_{PL} \alpha.t$;
2. $s + r \simeq_{PL} t + r$;

3. $s \oplus_i r \simeq_{PL} t \oplus_i r$;
4. $s \setminus L \simeq_{PL} t \setminus L$;
5. $s[\phi] \simeq_{PL} t[\phi]$;
6. $s \parallel r \simeq_{PL} t \parallel r$;

We only prove cases 1, 3, and 6 as the proof of remaining cases is almost identical. Let $\nu \cdot \lambda$ denote the concatenation of two configuration vectors ν and λ by appending the elements of λ at the end of ν. Furthermore, assume that $|\nu|$ denotes the length of configuration vector ν, and $max(S)$ denotes the maximum index in set S, where $max(\emptyset) = 0$. Note that any full configuration ν^f with respect to $s \odot r$ can be written as either $\nu_s \cdot \lambda_1$ or $\nu_r \cdot \lambda_2$ for some $\nu_s \in \mathit{VFConfig}(s)$ and $\nu_r \in \mathit{VFConfig}(r)$. The following two lemmata are required for the proof. In following proofs, we use $\equiv$ to denote syntactic equivalence.

Lemma B.1. *For each PL-CCS term t, $t \xrightarrow{a,\nu} t'$, where $|\nu| \geq max(bi(t))$, implies $t \xrightarrow{a,\nu\cdot\lambda_?} t$.*

Proof. The proof is straightforward by induction on the structure of t. The only interesting cases are given below:

- $t \equiv t_1 \oplus_i t_2$: by SOS rule Select, $t \xrightarrow{a,\nu} t'_1$, since $t_1 \xrightarrow{a,\nu'} t'_1$ and $\nu'|_i \neq R$, and $\nu = \nu'|_{i/L}$. By induction, $t_1 \xrightarrow{a,\nu'\cdot\lambda_?} t'_1$ while $\nu'|_i \neq R \Rightarrow (\nu' \cdot \lambda_?)|_i \neq R$. Consequently by SOS rule Select, $t \xrightarrow{a,\nu\cdot\lambda_?} t'_1$ holds. The same discussion holds when $t \xrightarrow{a,\nu} t'_2$ as the result of $t_2 \xrightarrow{a,\nu'} t'_2$.
- $t \equiv t_1 \parallel t_2$: by SOS rule Sync, we have $t \xrightarrow{\tau,\nu'\odot\nu''} t'_1 \parallel t'_2$, since $t_1 \xrightarrow{a,\nu'} t'_1$ and $t_2 \xrightarrow{\overline{a},\nu''} t'_2$, and $\nu' \asymp \nu''$. By induction, $t_1 \xrightarrow{a,\nu'\cdot\lambda_?} t'_1$ and $t_2 \xrightarrow{\overline{a},\nu''\cdot\lambda_?} t'_2$. Since $(\nu' \cdot \lambda_?) \asymp (\nu'' \cdot \lambda_?)$, then by SOS role Sync, we have $t \xrightarrow{\tau,(\nu'\odot\nu'')\cdot\lambda_?} t'_1 \parallel t'_2$. The same argument holds when $t \xrightarrow{a,\nu} t'_1 \parallel t_2$ or $t \xrightarrow{a,\nu} t_1 \parallel t'_2$ by SOS rule Par as the results of $t_1 \xrightarrow{a,\nu'} t'_1$ or $t_2 \xrightarrow{a,\nu'} t'_2$, respectively.

□

Lemma B.2. *For each PL-CCS term t, $t \xrightarrow{a,\nu} t'$ implies $t \xrightarrow{a,\nu'} t$, where $\nu = \nu' \cdot \lambda_?$ and $|\nu'| \geq max(bi(t))$.*

Proof. By induction on the structure of t. The only interesting cases are:

- $t \equiv t_1 \oplus_i t_2$: by SOS rule *Select*, we have $t \xrightarrow{a,\nu} t_1'$, since $t_1 \xrightarrow{a,\nu'} t'$ and $\nu'|_i \neq R$, and $\nu \equiv \nu'|_{i/L}$. By induction, we obtain $t_1 \xrightarrow{a,\nu''} t'$, where $\nu' \equiv \nu'' \cdot \lambda_?$ and $|\nu''| \geq max(bi(t))$. Since $i \in bi(t)$, then $i < |\nu''|$, and consequently $\nu''|_i \neq R$. Therefore by SOS rule *Select*, $t \xrightarrow{a,\nu''|_{i/L}} t_1'$ holds. The same argument holds when $t \xrightarrow{a,\nu} t_2'$ as the result of $t_2 \xrightarrow{a,\nu'} t_2'$.

- $t \equiv t_1 \parallel t_2$: by SOS rule *Sync*, we obtain $t \xrightarrow{\tau,\nu_1 \odot \nu_2} t_1' \parallel t_2'$, since $t_1 \xrightarrow{a,\nu_1} t_1'$ and $t_2 \xrightarrow{\overline{a},\nu_2} t_2'$, and $\nu_1 \asymp \nu_2$. By induction, $t_1 \xrightarrow{a,\nu_1'} t_1'$ and $t_2 \xrightarrow{\overline{a},\nu_2'} t_2'$, where $\nu_1 = \nu_1' \cdot \lambda_?$, $\nu_2 = \nu_2' \cdot \lambda_?$, and $|\nu_1'|, |\nu_2'| \geq max(bi(t))$. Therefore, $\nu_1 \asymp \nu_2$ implies $\nu_1' \asymp \nu_2'$, and by SOS rule *Sync*, we have that $t \xrightarrow{a,\nu_1' \odot \nu_2'} t_1' \parallel t_2'$. The same argument holds when $t \xrightarrow{a,\nu} t_1' \parallel t_2$ or $t \xrightarrow{a,\nu} t_1 \parallel t_2'$ by SOS rule *Par* as the results of $t_1 \xrightarrow{a,\nu'} t_1'$ or $t_2 \xrightarrow{a,\nu'} t_2'$, respectively.

□

We now proceed with the proof of Theorem 6.12. Following Definition 6.4, $s \simeq_{PL} t$ implies that for any full configuration ν_1^f with respect to s, there exists a valid full configuration ν_2^f with respect to t such that $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$ holds. In the remainder of the proof, we assume that $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$ is witnessed by the strong bisimulation relation $\mathcal{R}$.

Case 1. $\nu_1^f \in VFConfig(s)$ and $\nu_2^f \in VFConfig(t)$ and hence, $\nu_1^f \in VFConfig(a.s)$ and $\nu_2^f \in VFConfig(a.t)$. Therefore, we only need to prove that $\Pi(a.s, \nu_1^f) \sim \Pi(a.t, \nu_2^f)$. To this end, we prove that the closure of $\mathcal{R}$ with action prefixing, denoted by $\mathcal{R}'$, is a strong bisimulation relation. We formally define $\mathcal{R}'$ as $\mathcal{R} \cup \{(\Pi(a.s, \nu_1^f), \Pi(a.t, \nu_2^f)) \mid (\Pi(s, \nu_1^f), \Pi(t, \nu_2^f)) \in \mathcal{R}\}$. It remains to show the transfer conditions of Definition 2.1 for each pair in $\mathcal{R}'$. The case for the pairs in $\mathcal{R}$ holds vacuously. We only need to show the transfer conditions for an arbitrary pair $(\Pi(a.s, \nu_1^f), \Pi(a.t, \nu_2^f)) \in \mathcal{R}'$. Assume that $\Pi(a.s, \nu_1^f) \xrightarrow{a} (s', \nu_1^f)$ for some s'. This transition can only be due to SOS rule *Prefix* and s' must be s. Hence, $a.s \xrightarrow{a,\nu_?} s$ and $\nu_? \sqsubseteq \nu_1^f$. Similarly, it follows from SOS rule *Prefix* and the definition of the LTS semantic of product lines that $\Pi(a.t, \nu_2^f) \xrightarrow{a} \Pi(t, \nu_2^f)$. Due to the definition of $\mathcal{R}'$, we have that $(\Pi(s, \nu_1^f), \Pi(t, \nu_2^f)) \in \mathcal{R}$ and hence $(\Pi(s, \nu_1^f), \Pi(t, \nu_2^f)) \in \mathcal{R}'$, which was to be shown.

Case 3. Following the Definition 6.4, we prove that for any valid full configuration ${\nu^f}'_1$ with respect to $s \oplus_i r$, there exists a valid full configuration ${\nu^f}'_2$ with respect to $t \oplus_i r$ such that $\Pi(s \oplus_i r, {\nu^f}'_1) \sim \Pi(t \oplus_i r, {\nu^f}'_2)$ holds. Since $s \oplus_i r$ and $t \oplus_i r$ are fully expanded, i is fresh in s, t, and r. Regarding the value of ${\nu^f}'_1|_i$, two cases can be considered:

1. $\nu^{f'}{}_1|_i = L$: Let $\nu^{f'}{}_1 = \nu_1^f \cdot \lambda_1$. Take $\nu^{f'}{}_2 = \nu_1^f \cdot \lambda_2$ such that $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$ and $\nu^{f'}{}_1|_i = \nu^{f'}{}_2|_i$. Construct $\mathcal{R}_1'$ as:

$$\mathcal{R}_1' = \{(\Pi(s \oplus_i r, \nu^{f'}{}_1), \Pi(t \oplus_i r, \nu^{f'}{}_2))\} \cup \{(\Pi(s', \nu^{f'}{}_1), \Pi(t', \nu^{f'}{}_2)) | (\Pi(s', \nu_1^f), \Pi(t', \nu_2^f)) \in \mathcal{R}\} \ .$$

We prove that $\mathcal{R}_1'$ satisfies the transfer conditions of Definition 2.1. We only examine the transfer conditions for for an arbitrary pair $(\Pi(s \oplus_i r, \nu^{f'}{}_1), \Pi(t \oplus_i r, \nu^{f'}{}_2)) \in \mathcal{R}_1'$ as the pairs in $\mathcal{R}$ trivially satisfy the transfer conditions. Suppose that $\Pi(s \oplus_i r, \nu^{f'}{}_1) \xrightarrow{a} p$. This transition can only be due to SOS rule *Select* and transition $s \xrightarrow{a,\nu} s'$, where $\nu \sqsubseteq \nu^{f'}{}_1$, and $p \equiv \Pi(s', \nu^{f'}{}_1)$. Hence, we have that $s \oplus_i r \xrightarrow{a,\nu|_{i/L}} s'$. By Lemma B.2, $s \xrightarrow{a,\nu_s} s'$, where $\nu = \nu_s \cdot \lambda_?$, $\nu_s \sqsubseteq \nu_1^f$ and $\nu_s|_i = ?$. Therefore, it follows from the definition of the LTS semantic of product lines that $\Pi(s, \nu_1^f) \xrightarrow{a} \Pi(s', \nu_1^f)$. Furthermore, $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$ implies that $\Pi(t, \nu_2^f) \xrightarrow{a} \Pi(t', \nu_2^f)$ and $\Pi(s', \nu_1^f) \ \mathcal{R} \ \Pi(t', \nu_2^f)$. Similarly, it follows from the definition of the LTS semantic of product lines that $t \xrightarrow{a,\nu_t} t'$, where $\nu_t \sqsubseteq \nu_2^f$, $\nu_t|_i = ?$. Thus, by Lemma B.1, $t \xrightarrow{a,\nu'} t'$, where $\nu' = \nu_t \cdot \lambda_?$ and $\nu' \sqsubseteq \nu^{f'}{}_2$. From SOS rule *Select* and the definition of the LTS semantic of product lines, it follows that $\Pi(t \oplus_i r, \nu^{f'}{}_2) \xrightarrow{a} \Pi(t', \nu^{f'}{}_2)$. Due to the definition of $\mathcal{R}'$, we have that $(\Pi(s', \nu_1^f), \Pi(t', \nu_2^f)) \in \mathcal{R}$ and hence $(\Pi(s', \nu_1^f), \Pi(t', \nu_2^f)) \in \mathcal{R}'$, which was to be shown. The same discussion holds when $\Pi(t \oplus_i r, \nu^{f'}{}_2) \xrightarrow{a} p$. Concluding that $\mathcal{R}_1'$ is a strong bisimulation.

2. $\nu^{f'}{}_1|_i = R$: Let $\nu^{f'}{}_1 = \nu^f \cdot \lambda_1$, where ν^f is a valid configured configuration with respect to r, then take $\nu^{f'}{}_2 = \nu^f \cdot \lambda_2$ such that $\nu^{f'}{}_1|_i = \nu^{f'}{}_2|_i$. Construct $\mathcal{R}_2'$ as:

$$\mathcal{R}_2' = \{(\Pi(s \oplus_i r, \nu^{f'}{}_1), \Pi(t \oplus_i r, \nu^{f'}{}_2))\} \cup \{(\Pi(r', \nu^{f'}{}_1), \Pi(r', \nu^{f'}{}_2))\} \ .$$

We prove that $\mathcal{R}_2'$ satisfies the transfer conditions of Definition 2.1. We only examine the transfer conditions for an arbitrary pair $(\Pi(s \oplus_i r, \nu^{f'}{}_1), \Pi(t \oplus_i r, \nu^{f'}{}_2)) \in \mathcal{R}_2'$ as the pairs in $\{(\Pi(r', \nu^{f'}{}_1), \Pi(r', \nu^{f'}{}_2))\}$ trivially satisfy the transfer conditions. Suppose that $\Pi(s \oplus_i r, \nu^{f'}{}_1) \xrightarrow{a} p$. This transition can only be due to SOS rule *Select* and transition $r \xrightarrow{a,\nu} r'$, where $\nu \sqsubseteq \nu^{f'}{}_1$, and $p \equiv \Pi(r', \nu^{f'}{}_1)$. Hence, $s \oplus_i r \xrightarrow{a,\nu|_{i/R}} r'$. By Lemma B.2, $r \xrightarrow{a,\nu_r} r'$, where $\nu = \nu_r \cdot \lambda_?$ and $\nu_r \sqsubseteq \nu^f$. Hence, by Lemma B.1, $r \xrightarrow{a,\nu'} r'$, where $\nu' = \nu_r \cdot \lambda_?$, and $\nu' \sqsubseteq \nu^{f'}{}_2$. From SOS rule *Select* and the definition of the LTS semantic of product lines, it follows that $\Pi(t \oplus_i r, \nu^{f'}{}_2) \xrightarrow{a} \Pi(r', \nu^{f'}{}_2)$, and by the definition of $\mathcal{R}_2'$ we have that $\Pi(r', \nu^{f'}{}_1) \ \mathcal{R}_2' \ \Pi(r', \nu^{f'}{}_2)$, which was to be shown. The same discussion holds when $\Pi(t \oplus_i r, \nu^{f'}{}_2) \xrightarrow{a} p$. Concluding, $\mathcal{R}_2'$ is a strong bisimulation.

The same discussion holds for any valid full configuration ${\nu^f}'_2$ with respect to $t \oplus_i r$. We conclude that $s \oplus_i r \simeq_{PL} t \oplus_i r$.

Case 6. Since $s \parallel r$ and $t \parallel r$ are fully expanded and the root of their parse tree is parallel composition, s and t cannot have any binary variant in common with r, i.e., $bi(s) \cap bi(r) = \emptyset$ and $bi(t) \cap bi(r) = \emptyset$. Consequently, for any $\nu^f_1 \in VFConfig(s)$, there exists a $\nu^f_2 \in VFConfig(t)$ such that $\forall i \in bi(r)(\nu^f_1|_i = \nu^f_2|_i)$. Following the Definition 6.4, we prove that for any ${\nu^f}'_1 \in VFConfig(s \parallel r)$, there exists ${\nu^f}'_2 \in VFConfig(t \parallel r)$ such that $\Pi(s \parallel r, {\nu^f}'_1) \sim \Pi(t \parallel r, {\nu^f}'_2)$ holds. To this aim, for any ${\nu^f}'_1 = \nu^f_1 \cdot \lambda_1$, take ${\nu^f}'_2 = \nu^f_2 \cdot \lambda_2$ such that $\Pi(s, \nu^f_1) \sim \Pi(t, \nu^f_1)$ and $\forall i \in bi(r)(\nu^f_1|_i = \nu^f_2|_i)$. Construct $\mathcal{R}'$ as

$$\mathcal{R}' = \{(\Pi(s' \parallel r', {\nu^f}'_1), \Pi(t' \parallel r', {\nu^f}'_2)) | (\Pi(s', \nu^f_1), \Pi(t', \nu^f_2)) \in \mathcal{R}\}$$

We show that it satisfies the transfer conditions of Definition 2.1.

For an arbitrary pair $\Pi(s' \parallel r', {\nu^f}'_1)\ \mathcal{R}'\ \Pi(t' \parallel r', {\nu^f}'_1)$, suppose that $\Pi(s' \parallel r', {\nu^f}'_1) \xrightarrow{a} p$. Using the SOS rules Par and $Sync$, three cases can be considered:

1. This transition can be due to SOS rule Par and $s' \xrightarrow{a,\nu'_1} s''$, where $\nu'_1 \sqsubseteq {\nu^f}'_1$. Hence, $s' \parallel r' \xrightarrow{a,\nu'_1} s'' \parallel r'$, and $p \equiv \Pi(s'' \parallel r', {\nu^f}'_1)$. By Lemma B.2, $s' \xrightarrow{a,\nu_s} s''$, where $\nu'_1 = \nu_s \cdot \lambda_?$ and $\nu_s \sqsubseteq \nu^f_1$. Thus, it follows from the definition of the LTS semantic of product lines that $\Pi(s', \nu^f_1) \xrightarrow{a} \Pi(s'', \nu^f_1)$. Furthermore, $\Pi(s', \nu^f_1)\ \mathcal{R}\ \Pi(t', \nu^f_2)$ implies that $\Pi(t', \nu^f_2) \xrightarrow{a} \Pi(t'', \nu^f_2)$ and $\Pi(s'', \nu^f_1)\ \mathcal{R}\ \Pi(t'', \nu^f_2)$. Similarly, it follows from the definition of the LTS semantic of product lines that $t' \xrightarrow{a,\nu_t} t''$, where $\nu_t \sqsubseteq \nu^f_2$ and consequently by Lemma B.1, $t' \xrightarrow{a,\nu'_2} t''$, where $\nu'_2 = \nu_t \cdot \lambda_?$ and $\nu'_2 \sqsubseteq {\nu^f}'_2$. By SOS rule Par, $t' \parallel r' \xrightarrow{a,\nu'_2} t'' \parallel r'$ and consequently, $\Pi(t' \parallel r', {\nu^f}'_2) \xrightarrow{a} \Pi(t'' \parallel r', {\nu^f}'_2)$. Due to the definition of $\mathcal{R}'$ we have that $\Pi(s'', \nu^f_1)\ \mathcal{R}\ \Pi(t'', \nu^f_2)$, and hence $\Pi(s'' \parallel r', {\nu^f}'_1)\ \mathcal{R}'\ \Pi(t'' \parallel r', {\nu^f}'_2)$, which was to be shown.

2. This transition can be due to SOS rule Par and $r' \xrightarrow{a,\nu'} r''$, where $\nu' \sqsubseteq {\nu^f}'_1$. Hence, $s' \parallel r' \xrightarrow{a,\nu'} s' \parallel r''$ and $p \equiv \Pi(s' \parallel r'', {\nu^f}'_1)$. Therefore, by Lemmas B.2 and B.1, $r' \xrightarrow{a,\nu''} r''$, where $\nu'' \sqsubseteq {\nu^f}'_2$. Thus, by SOS rule Par, $t' \parallel r' \xrightarrow{a,\nu''} t' \parallel r''$, where $\nu'' \sqsubseteq {\nu^f}'_2$, and consequently, $\Pi(t' \parallel r', {\nu^f}'_2) \xrightarrow{a} \Pi(t' \parallel r'', {\nu^f}'_2)$. Due to the definition of $\mathcal{R}'$ we have that $\Pi(s', \nu^f_1)\ \mathcal{R}\ \Pi(t', \nu^f_2)$, and hence $\Pi(s' \parallel r'', {\nu^f}'_1)\ \mathcal{R}'\ \Pi(t' \parallel r'', {\nu^f}'_2)$, which was to be shown.

3. This transition can be due to SOS rule $Sync$ and transitions $s' \xrightarrow{a,\nu'_1} s''$ and $r' \xrightarrow{\overline{a},\nu'} r''$, where $\nu'_1 \sqsubseteq {\nu^f}'_1$, $\nu' \sqsubseteq {\nu^f}'_1$, and $\nu'_1 \asymp \nu'$, and consequently $p \equiv \Pi(s'' \parallel r'', {\nu^f}'_1)$. Hence, $s' \parallel r' \xrightarrow{\tau,\nu'_1 \odot \nu'} s'' \parallel r''$. By Lemma B.2,

$s' \xrightarrow{a,\nu_s} s''$, where $\nu'_1 = \nu_s \cdot \lambda_?$ and $\nu_s \sqsubseteq \nu_1^f$. It follows from the definition of the LTS semantic of product lines that $\Pi(s', \nu_1^f) \xrightarrow{a} \Pi(s'', \nu_1^f)$. Furthermore, $\Pi(s', \nu_1^f)\ \mathcal{R}\ \Pi(t', \nu_2^f)$ implies that $\Pi(t', \nu_2^f) \xrightarrow{a} \Pi(t'', \nu_2^f)$ and $\Pi(s'', \nu_1^f)\ \mathcal{R}\ \Pi(t'', \nu_2^f)$. It follows that $t' \xrightarrow{a,\nu_t} t''$, where $\nu_t \sqsubseteq \nu_2^f$, and consequently, by Lemma B.1, $t' \xrightarrow{a,\nu'_2} t''$, where $\nu'_2 = \nu_t \cdot \lambda_?$ and $\nu'_2 \sqsubseteq {\nu^f}'_2$. Since $\nu_s \sqsubseteq \nu_1^f$ and $\nu_t \sqsubseteq \nu_2^f$, then $\nu'_1 \asymp \nu'$ and $\forall i \in bi(r)(\nu_1^f|_i = \nu_2^f|_i)$ imply that $\nu'_2 \asymp \nu'$ and $\nu'_2 \odot \nu' \sqsubseteq {\nu^f}'_2$. By SOS rule *Sync*, we have that $t' \parallel r' \xrightarrow{\tau,\nu'_2 \odot \nu'} t'' \parallel r''$ and hence, $\Pi(t' \parallel r', {\nu^f}'_2) \xrightarrow{\tau} \Pi(t'' \parallel r'', {\nu^f}'_2)$. Due to the Definition of $\mathcal{R}'$, we have that $\Pi(s'', \nu_1^f)\ \mathcal{R}\ \Pi(t'', \nu_2^f)$ and hence, $\Pi(s'' \parallel r'', {\nu^f}'_1)\ \mathcal{R}'\ \Pi(t'' \parallel r'', {\nu^f}'_2)$.

The same discussion holds when $\Pi(t' \parallel r', {\nu^f}'_2) \xrightarrow{a} p$. Concluding, $\mathcal{R}'$ is a strong bisimulation, and consequently $s \parallel r \simeq_{PL} t \parallel r$.

B.1.3 Proof of Theorem 6.10

Theorem 6.10 - Part 1. Strict strong bisimilarity is an equivalence relation.
Proof. To show that strict strong bisimilarity is an equivalence, we must show that it is reflexive, symmetric, and transitive. Reflexivity and symmetry follow immediately from the reflexivity and symmetry properties of strong bisimilarity. Hence, it only remains to prove transitivity.

Consider PL-CCS terms s, t, and r such that $s \approx_{PL} t$, and $t \approx_{PL} r$; following Definition 6.3 for any valid full configuration $\nu_1^f \in \mathit{VFConfig}(s) \cap \mathit{VFConfig}(t)$ and $\nu_2^f \in \mathit{VFConfig}(t) \cap \mathit{VFConfig}(r)$, it holds that $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_1^f)$, and $\Pi(t, \nu_2^f) \sim \Pi(r, \nu_2^f)$. Without loss of generality, we assume that $max(bi(s)) \geq max(bi(t)) \geq max(bi(r))$ (the proof of all other cases is almost identical). Therefore, $\nu_1^f = \nu_r \cdot \lambda_1 \cdot \lambda_2$, $\nu_2^f = \nu_r \cdot \lambda_2$, where $\nu_r \in \mathit{VFConfig}(r)$. We prove that $\Pi(s, \nu_1^f) \sim \Pi(r, \nu_1^f)$. Construct $\mathcal{R}$ as

$$\mathcal{R} = \{(\Pi(s', \nu_1^f), \Pi(r', \nu_1^f)) \mid \Pi(s', \nu_1^f) \sim \Pi(t', \nu_1^f) \wedge \Pi(t', \nu_2^f) \sim \Pi(r', \nu_2^f)\}$$

We prove that $\mathcal{R}$ satisfies the transfer conditions of Definition 2.1.

For an arbitrary pair $(\Pi(s', \nu_1^f), \Pi(r', \nu_1^f)) \in \mathcal{R}$, consider that $\Pi(s', \nu_1^f) \xrightarrow{\alpha} \Pi(s'', \nu_1^f)$ since $s' \xrightarrow{\alpha,\nu'_1} s''$, where $\nu'_1 \sqsubseteq \nu_1^f$. Hence, $\Pi(s', \nu_1^f) \sim \Pi(t', \nu_1^f)$ implies that $\Pi(t', \nu_1^f) \xrightarrow{\alpha} \Pi(t'', \nu_1^f)$ and $\Pi(s'', \nu_1^f) \sim \Pi(t'', \nu_1^f)$. It follows that $t' \xrightarrow{\alpha,\nu_2} t''$, where $\nu_2 \sqsubseteq \nu_1^f$. By Lemma B.2, it holds that $t' \xrightarrow{\alpha,\nu'_2} t''$, where $\nu_2 = \nu'_2 \cdot \lambda_?$ and $\nu'_2 \sqsubseteq \nu_2^f$ and consequently $\Pi(t', \nu_2^f) \xrightarrow{\alpha} \Pi(t'', \nu_2^f)$. Therefore, $\Pi(t', \nu_2^f) \sim \Pi(r', \nu_2^f)$ implies that $\Pi(r', \nu_2^f) \xrightarrow{\alpha} \Pi(r'', \nu_2^f)$ and $\Pi(t'', \nu_2^f) \sim \Pi(r'', \nu_2^f)$. It follows that $r' \xrightarrow{\alpha,\nu'_3} r''$, where $\nu'_3 \sqsubseteq \nu_2^f$. Consequently, by Lemma B.1, $r' \xrightarrow{\alpha,\nu_3} r''$, where $\nu_3 = \nu'_3 \cdot \lambda_?$ and $\nu_3 \sqsubseteq \nu_1^f$. Hence, $\Pi(r', \nu_1^f) \xrightarrow{\alpha} \Pi(r'', \nu_1^f)$. Due to the Definition of $\mathcal{R}$, we have that $\Pi(s'', \nu_1^f) \sim \Pi(t'', \nu_1^f)$ and $\Pi(t'', \nu_2^f) \sim \Pi(r'', \nu_2^f)$ and hence,

$(\Pi(s'', \nu_1^f), \Pi(r'', \nu_1^f)) \in \mathcal{R}$, which was to be shown. The same discussion holds when $r' \xrightarrow{\alpha,\nu_3} r''$, where $\nu_3 \sqsubseteq \nu_1^f$. Concluding, $\mathcal{R}$ is a strong bisimulation, and consequently $s \approx_{PL} r$.

Theorem 6.10 - Part 2. Strict strong bisimilarity is congruence on PL-CCS terms.

Proof. To show that strict strong bisimilarity is a congruence on PL-CCS terms with respect to the PL-CCS operators, we need to prove that for any arbitrary product line r, renaming function ϕ, and $L \subseteq Act$, $s \approx_{PL} t$ implies following cases:

1. $\alpha.s \approx_{PL} \alpha.t$;
2. $s + r \approx_{PL} t + r$;
3. $s \oplus_i r \approx_{PL} t \oplus_i r$;
4. $s \setminus L \approx_{PL} t \setminus L$;
5. $s[\phi] \approx_{PL} t[\phi]$;
6. $s \parallel r \approx_{PL} t \parallel r$;

We only prove cases 1, 3, and 6; the proof of remaining cases is almost identical. Note that $\forall \nu^{f'} \in \mathit{VFConfig}(s \odot r) \cap \mathit{VFConfig}(t \odot r) \Rightarrow \exists \nu^f \in \mathit{VFConfig}(s) \cap \mathit{VFConfig}(t)(\nu^{f'} = \nu^f \cdot \lambda)$, where $\odot \in \{+, \oplus, \parallel, \between, |\}$. For any valid full configuration ν^f with respect to s and t, we assume $\Pi(s, \nu^f) \sim \Pi(t, \nu^f)$ is witnessed by strong bisimulation $\mathcal{R}$.

Case 1. We have that $\nu^f \in \mathit{VFConfig}(s) \cap \mathit{VFConfig}(t)$, and hence, $\nu^f \in \mathit{VFConfig}(a.s) \cap \mathit{VFConfig}(a.t)$. Therefore, we only need to prove that $\Pi(a.s, \nu^f) \sim \Pi(a.t, \nu^f)$. To this end, we prove that the closure of $\mathcal{R}$ with action prefixing, denoted by $\mathcal{R}'$ is a strong bisimulation relation. It remains to show the transfer conditions of Definition 2.1 for each pair in $\mathcal{R}'$. The case for the pairs in $\mathcal{R}$ holds vacuously. We only need to show the transfer conditions for an arbitrary pair $(\Pi(a.s, \nu_1^f), \Pi(a.t, \nu_2^f)) \in \mathcal{R}'$. Assume that $\Pi(a.s, \nu^f) \xrightarrow{a} (s', \nu^f)$ for some s'. This transition can only be due to SOS rule *Prefix* and s' must be s. Hence, we have that $a.s \xrightarrow{a,\nu_?} s$ and $\nu_? \sqsubseteq \nu^f$. Similarly, it follows from SOS rule *Prefix* and the definition of the LTS semantic of product lines that $\Pi(a.t, \nu^f) \xrightarrow{a} \Pi(t, \nu^f)$. Due to the definition of $\mathcal{R}'$, we have that $(\Pi(s, \nu^f), \Pi(t, \nu^f)) \in \mathcal{R}$ and hence $(\Pi(s, \nu^f), \Pi(t, \nu^f)) \in \mathcal{R}'$, which was to be shown.

Case 3. Following Definition 6.3, we prove that for any valid full configuration $\nu^{f'}$ with respect to $s \oplus_i r$ and $t \oplus_i r$, $\Pi(s \oplus_i r, \nu^{f'}) \sim \Pi(t \oplus_i r, \nu^{f'})$ holds. Construct $\mathcal{R}'$, as:

$$\mathcal{R}' = \{(\Pi(s \oplus_i r, \nu^{f'}), \Pi(t \oplus_i r, \nu^{f'}))\} \cup \{(\Pi(t', \nu^{f'}), \Pi(t', \nu^{f'}))\} \cup \{(\Pi(s', \nu^{f'}), \Pi(t', \nu^{f'})) | (\Pi(s', \nu^f), \Pi(t', \nu^f)) \in \mathcal{R}\}$$

We prove that $\mathcal{R}'$ satisfies the transfer conditions of Definition 2.1. We only prove the transfer conditions for the pair $(\Pi(s \oplus_i r, \nu^{f'}), \Pi(t \oplus_i r, \nu^{f'})) \in \mathcal{R}'$ as for others,

the proof is trivial (the proof for pairs such as $(\Pi(s', \nu^{f'}), \Pi(t', \nu^{f'})) \in \mathcal{R}'$ follows from Lemmas B.1 and B.2).

Suppose that $\Pi(s \oplus_i r, \nu^{f'}) \xrightarrow{a} p$. Regarding the value of $\nu^{f'}|_i$ two cases can be considered:

1. $\nu^{f'}|_i = L$: This transition can be due to SOS rule *Select* and $s \xrightarrow{a,\nu} s'$, where $\nu \sqsubseteq \nu^{f'}$, and $p \equiv \Pi(s', \nu^{f'})$. Hence, $s \oplus_i r \xrightarrow{a,\nu|_{i/L}} s'$. By Lemma B.2, $s \xrightarrow{a,\nu_s} s'$, where $\nu = \nu_s \cdot \lambda_?$, and $\nu_s \sqsubseteq \nu^f$. Therefore, from the definition of the LTS semantic of product lines it follows that $\Pi(s, \nu^f) \xrightarrow{a} \Pi(s', \nu^f)$. Furthermore, $\Pi(s, \nu^f) \sim \Pi(t, \nu^f)$ implies that $\Pi(t, \nu^f) \xrightarrow{a} \Pi(t', \nu^f)$ and $\Pi(s', \nu^f) \sim \Pi(t', \nu^f)$. Similarly, it follows that $t \xrightarrow{a,\nu_t} t'$, where $\nu_t \sqsubseteq \nu^f$. Thus, by Lemma B.1, $t \xrightarrow{a,\nu'} t'$, where $\nu' = \nu_t \cdot \lambda_?$, and $\nu' \sqsubseteq \nu^{f'}$. By *Select*, $t \oplus_i r \xrightarrow{a,\nu'|_{i/L}} t'$ and consequently $\Pi(t \oplus_i r, \nu^{f'}) \xrightarrow{a} \Pi(t', \nu^{f'})$. Due to the definition of $\mathcal{R}'$, we have that $\Pi(s', \nu^f) \sim \Pi(t', \nu^f)$ and hence, $\Pi(s', \nu^{f'}) \; \mathcal{R}' \; \Pi(t', \nu^{f'})$, which was to be shown.

2. $\nu^f|_i = R$: This transition can be due to SOS rule *Select* and $r \xrightarrow{a,\nu} r'$, where $\nu \sqsubseteq \nu^{f'}$, and $p \equiv \Pi(r', \nu^{f'})$. Hence, $s \oplus_i r \xrightarrow{a,\nu|_{i/R}} r'$ and similarly, $t \oplus_i r \xrightarrow{a,\nu|_{i/R}} t'$. It follows from the definition of the LTS semantic of product lines that $\Pi(t \oplus_i r, \nu^{f'}) \xrightarrow{a} \Pi(r', \nu^{f'})$. By the definition of $\mathcal{R}'$, $\Pi(r', \nu^{f'}) \; \mathcal{R}' \; \Pi(r', \nu^{f'})$.

The same discussion holds when $\Pi(t \oplus_i r, \nu^{f'}) \xrightarrow{a} p$. Concluding, $\mathcal{R}'$ is a strong bisimulation. Consequently $s \oplus_i r \approx_{PL} t \oplus_i r$.

Case 6. Following the Definition 6.3, we prove that for any valid full configuration $\nu^{f'}$ with respect to $s \parallel r$ and $t \parallel r$, $\Pi(s \parallel r, \nu^{f'}) \sim \Pi(t \parallel r, \nu^{f'})$ holds. Construct $\mathcal{R}'$ as

$$\mathcal{R}' = \{(\Pi(s' \parallel r', \nu^{f'}), \Pi(t' \parallel r', \nu^{f'}))|(\Pi(s', \nu^f), \Pi(t', \nu^f)) \in \mathcal{R}\}$$

We show that it satisfies the transfer conditions of Definition 2.1.

For an arbitrary pair $\Pi(s' \parallel r', \nu^{f'}) \; \mathcal{R}' \; \Pi(t' \parallel r', \nu^{f'})$, suppose that $\Pi(s' \parallel r', \nu^{f'}) \xrightarrow{a} p$. Using the SOS rules *Par* and *Sync*, three cases can be considered:

1. This transition can be due to SOS rule *Par* and $s' \xrightarrow{a,\nu'_1} s''$, where $\nu'_1 \sqsubseteq \nu^{f'}$, and $p \equiv \Pi(s'' \parallel r', \nu^{f'})$. Hence, $s' \parallel r' \xrightarrow{a,\nu'_1} s'' \parallel r'$. By Lemma B.2, $s' \xrightarrow{a,\nu_s} s''$, where $\nu'_1 = \nu_s \cdot \lambda_?$ and $\nu_s \sqsubseteq \nu^f$. It follows from the definition of the LTS semantic of product lines that $\Pi(s', \nu^f) \xrightarrow{a} \Pi(s'', \nu^f)$. Furthermore, $\Pi(s', \nu^f) \; \mathcal{R} \; \Pi(t', \nu^f)$ implies that $\Pi(t', \nu^f) \xrightarrow{a} \Pi(t'', \nu^f)$ and $\Pi(s'', \nu^f) \; \mathcal{R} \; \Pi(t'', \nu^f)$. It follows that $t' \xrightarrow{a,\nu_t} t''$, where $\nu_t \sqsubseteq \nu^f$, and consequently by Lemma B.1, $t' \xrightarrow{a,\nu'_2} t''$, where $\nu'_2 = \nu_t \cdot \lambda_?$ and $\nu'_2 \sqsubseteq \nu^{f'}$.

By SOS rule Par, $t' \parallel r' \xrightarrow{a,\nu'_2} t'' \parallel r'$ and hence, $\Pi(t' \parallel r', \nu^{f'}) \xrightarrow{a} \Pi(t'' \parallel r', \nu^{f'})$. Due to the Definition of $\mathcal{R}'$, we have that $\Pi(s'', \nu^f)\ \mathcal{R}\ \Pi(t'', \nu^f)$ and hence $\Pi(s'' \parallel r', \nu^{f'})\ \mathcal{R}'\ \Pi(t'' \parallel r', \nu^{f'})$, which was to be shown.

2. This transition can be due to SOS rule Par and $r' \xrightarrow{a,\nu'} r''$, where $\nu' \sqsubseteq \nu^{f'}$, and $p \equiv \Pi(s' \parallel r'', \nu^{f'})$. Hence, $s' \parallel r' \xrightarrow{a,\nu'} s' \parallel r''$ and similarly, $t' \parallel r' \xrightarrow{a,\nu'} t' \parallel r''$, where $\nu' \sqsubseteq \nu^{f'}$. Therefore, it follows that $\Pi(t' \parallel r', \nu^{f'}) \xrightarrow{a} \Pi(t' \parallel r'', \nu^{f'})$. Due to the Definition of $\mathcal{R}'$, we have that $\Pi(s' \parallel r'', \nu^{f'})\ \mathcal{R}'\ \Pi(t' \parallel r', \nu^{f''})$, which was to be shown.

3. This transition can be due to SOS rule $Sync$ and transitions $s' \xrightarrow{a,\nu'_1} s''$ and $r' \xrightarrow{\overline{a},\nu'} r''$, where $\nu'_1 \sqsubseteq \nu^{f'}$, $\nu' \sqsubseteq \nu^{f'}$, and $\nu'_1 \asymp \nu'$, and consequently $p \equiv \Pi(s'' \parallel r'', \nu^{f'})$. Hence, $s' \parallel r' \xrightarrow{\tau,\nu'_1 \odot \nu'} s'' \parallel r''$. By Lemma B.2, $s' \xrightarrow{a,\nu_s} s''$, where $\nu'_1 = \nu_s \cdot \lambda_?$ and $\nu_s \sqsubseteq \nu^f$. It follows from the definition of the LTS semantic of product lines that $\Pi(s', \nu^f) \xrightarrow{a} \Pi(s'', \nu^f)$. Furthermore, $\Pi(s', \nu^f)\ \mathcal{R}\ \Pi(t', \nu^f)$ implies that $\Pi(t', \nu^f) \xrightarrow{a} \Pi(t'', \nu^f)$ and $\Pi(s'', \nu^f)\ \mathcal{R}\ \Pi(t'', \nu^f)$. It follows that $t' \xrightarrow{a,\nu_t} t''$, where $\nu_t \sqsubseteq \nu^f$, and consequently, by Lemma B.1, $t' \xrightarrow{a,\nu'_2} t''$, where $\nu'_2 = \nu_t \cdot \lambda_?$ and $\nu'_2 \sqsubseteq \nu^{f'}$. Since $\nu_s \sqsubseteq \nu^f$ and $\nu_t \sqsubseteq \nu^f$, then $\nu_s \asymp \nu_t$. Therefore, $\nu'_1 \asymp \nu'$ implies $\nu'_2 \asymp \nu'$ and $\nu'_2 \odot \nu' \sqsubseteq \nu^{f'}$. By SOS rule $Sync$, we have that $t' \parallel r' \xrightarrow{\tau,\nu'_2 \odot \nu'} t'' \parallel r''$ and hence, $\Pi(t' \parallel r', \nu^{f'}) \xrightarrow{\tau} \Pi(t'' \parallel r'', \nu^{f'})$. Due to the Definition of $\mathcal{R}'$, we have that $\Pi(s'', \nu^f)\ \mathcal{R}\ \Pi(t'', \nu^f)$ and hence, $\Pi(s'' \parallel r'', \nu^{f'})\ \mathcal{R}'\ \Pi(t'' \parallel r'', \nu^{f'})$.

The same discussion holds when $\Pi(t' \parallel r', \nu^f)' \xrightarrow{a} p$. Concluding, $\mathcal{R}'$ is a strong bisimulation.

B.2 Proof of Theorem 6.13 and 6.20

In this Section, we first prove Theorem 6.13 and then we provide a proof for Theorem 6.20. To this aim, we prove that each PL-CCS term can be rewritten by axioms A_{4-6}, $D_{1,2}$, $P_{4,5}$, $S_{1,3}$, R_3, E_5, Dri, and N_{1-5} into a form where no binary variant occurs in the scope of a CCS-operator.

We prove this by structural induction on the syntax of term t. The base case of the induction, where $t \equiv 0$, holds trivially. We distinguish the following cases based on the structure of t:

- if $t \equiv a.t'$, then t can be rewritten into the required form by applying the induction hypothesis on t', and subsequently applying axiom A_4.

- if $t \equiv t'_1 + t'_2$, then t can be rewritten into the required format by applying the induction hypothesis on t'_1 and t'_2, and then applying axioms $A_{5,6}$. Assume that $\vdash t'_1 = p_1 \oplus_i p_2$ and $\vdash t'_2 = q_1 \oplus_j q_2$. By axioms $A_{5,6} \vdash t = ((p_1 + q_1) \oplus_j (p_1 + q_2)) \oplus_i ((p_2 + q_1) \oplus_j (p_2 + q_2))$. These two axioms are applied to each $p_i + q_j$, where $i, j \in \{1, 2\}$, repeatedly until the operands of $+$ become CCS terms.

- if $t \equiv t'_1 \oplus t'_2$, t can be rewritten into the required format by applying the induction hypothesis on t'_1 and t'_2.

- if $t \equiv t'_1 \parallel t'_2$, then t can be rewritten into the required format by applying the induction hypothesis on t'_1 and t'_2, and then applying axioms $D_{1,2}$. Assume $\vdash t'_1 = p_1 \oplus_i p_2$ and $\vdash t'_2 = q_1 \oplus_j q_2$. By axioms $D_{1,2} \vdash t = ((p_1 \parallel q_1) \oplus_j (p_1 \parallel q_2)) \oplus_i ((p_2 \parallel q_1) \oplus_j (p_2 \parallel q_2))$. These two axioms are applied to each $p_i \parallel q_j$, where $i, j \in \{1, 2\}$, repeatedly until the operands of $\parallel$ become CCS terms.

- if $t \equiv t'_1 \between t'_2$, then t can be rewritten into the required format by applying the induction hypothesis on t'_1 and t'_2, and then applying axioms $P_{4,5}$. Assume $\vdash t'_1 = p_1 \oplus_i p_2$ and $\vdash t'_2 = q_1 \oplus_j q_2$. By axioms $P_{4,5} \vdash t = ((p_1 \between q_1) \oplus_j (p_1 \between q_2)) \oplus_i ((p_2 \between q_1) \oplus_j (p_2 \between q_2))$. These two axioms are applied to each $p_i \between q_j$, where $i, j \in \{1, 2\}$, repeatedly until the operands of $\between$ become CCS terms.

- if $t \equiv t'_1 \mid t'_2$, then t can be rewritten into the required format by applying the induction hypothesis on t'_1 and t'_2, and then applying axioms $S_{1,3}$. Assume $\vdash t'_1 = p_1 \oplus_i p_2$ and $\vdash t'_2 = q_1 \oplus_j q_2$. By axioms $S_{1,3} \vdash t = ((p_1 \mid q_1) \oplus_j (p_1 \mid q_2)) \oplus_i ((p_2 \mid q_1) \oplus_j (p_2 \mid q_2))$. These two axioms are applied to each $p_i \mid q_j$, where $i, j \in \{1, 2\}$, repeatedly until the operands of $\mid$ become CCS terms.

- if $t \equiv \langle A | \{A \stackrel{def}{=} t'\}\rangle$, then by applying the induction hypothesis, we obtain $\vdash t' = p_1 \oplus_i p_2$. By axiom $Dri \vdash t = \langle A | \{A \stackrel{def}{=} p_1\}\rangle \oplus_i \langle A | \{A \stackrel{def}{=} p_2\}\rangle$. We apply this axiom repeatedly to p_1 and p_2, until no binary variant operator occurs in the equations defining A.

Hence, t is rewritten into a form where no binary variant is in the scope of CCS operator, i.e., $\vdash t = p_1 \oplus_i p_2$, where p_1 and p_2 are also in this form. For all $j \in bi(p_1 \oplus_i p_2)$, first apply N_5 to any subterm $p' \oplus_j q'$, i.e. $\vdash p' \oplus_j q' = L(p', j) \oplus_j L(q', j)$ and then N_{1-4} to make j free in p' and q'. We claim that the resulting term $t'' \equiv p'_1 \oplus_i p'_2$ is fully expanded, otherwise there is a simple path from $\oplus_j$ to another $\oplus_j$. This is impossible due to application of N_{1-5}.

For all $j \in bi(p'_1) \cap bi(p'_2)$, apply N_6 to p'_2, and replace all occurrences of j by a fresh index $k \notin bi(p'_1) \cup bi(p'_2)$ to derive t'''. Hence, all indices are unique, and the term is fully expanded. Theorem 6.20 is proved by applying axioms $A_{1,2}$ to t'''.

B.3 Soundness of Axiomatization

To prove the soundness of axioms, we show that all axioms except $A_{1,2}$ and N_6 are sound with respect to strict strong bisimilarity, while $A_{1,2}$ and N_6 are sound with respect to product line bisimilarity.

To show the soundness of A_1, for any $\nu_1^f \in \mathit{VFConfig}(p \oplus_i q)$, we define $\nu_2^f \in \mathit{VFConfig}(q \oplus_i p)$ such that $|\nu_1^f| = |\nu_2^f|$ and $\forall_{j \neq i}(\nu_1^f|_j = \nu_2^f|_j)$, $(\nu_1^f|_i = R \Rightarrow \nu_2^f|_i = L)$, and $(\nu_1^f|_i = L \Rightarrow \nu_1^f|_i = R)$. We show that $\Pi(p \oplus_i q, \nu_1^f) \sim \Pi(q \oplus_i p, \nu_2^f)$. To this aim, we show that $\mathcal{R} = \{(\Pi(p \oplus_i q, \nu_1^f), \Pi(q \oplus_i p, \nu_2^f))\} \cup \{(\Pi(t, \nu_1^f), \Pi(t, \nu_2^f))\}$ is a strong bisimulation. Regarding $\nu_1^f|_i$, two cases can be distinguished. We only discuss the case where $\nu_1^f|_i = L$, as the other can be dealt with in the same fashion.

We show that the transfer conditions hold for the pair $(\Pi(p \oplus_i q, \nu_1^f), \Pi(q \oplus_i p, \nu_2^f)) \in \mathcal{R}$, as for others, the proof is straightforward. Suppose $\Pi(p \oplus_i q, \nu_1^f) \xrightarrow{a} \Pi(p', \nu_1^f)$, since $p \xrightarrow{a,\nu_1} p'$, where $\nu_1 \sqsubseteq \nu_1^f$ and $\nu_1|_i =?$. Therefore, $\nu_1 \sqsubseteq \nu_2^f$. By SOS rule *Select*, $q \oplus_i p \xrightarrow{a,\nu_2} p'$, where $\nu_2 = \nu_1|_{i/R}$, $\nu_2 \sqsubseteq \nu_2^f$, and $\Pi(p', \nu_1^f)\ \mathcal{R}\ \Pi(p', \nu_2^f)$. The same discussion holds when $\Pi(q \oplus_i p, \nu_2^f) \xrightarrow{a} \Pi(p', \nu_2^f)$. Similarly, for any $\nu_2^f \in \mathit{VFConfig}(q \oplus_i p)$, we define $\nu_1^f \in \mathit{VFConfig}(q \oplus_i p)$ as discussed above. Concluding that $p \oplus_i q \simeq_{PL} q \oplus_i p$. Axiom A_2 is proved similar to A_1: for any $\nu_1^f \in \mathit{VFConfig}((p \oplus_i q) \oplus_j r)$, we define $\nu_2^f \in \mathit{VFConfig}(p \oplus_i (q \oplus_j r))$ such that $|\nu_1^f| = |\nu_2^f|$ and $\forall_{k \neq i, k \neq j}(\nu_1^f|_k = \nu_2^f|_k)$, $(\nu_1^f|_i = R \wedge \nu_1^f|_j = R) \Rightarrow (\nu_2^f|_i = R \wedge \nu_2^f|_j = R)$, $(\nu_1^f|_i = L \wedge \nu_1^f|_j = R) \Rightarrow (\nu_2^f|_i = R \wedge \nu_2^f|_j = R)$, $(\nu_1^f|_i = R \wedge \nu_1^f|_j = L) \Rightarrow (\nu_2^f|_i = R \wedge \nu_2^f|_j = L)$, and $(\nu_1^f|_i = L \wedge \nu_1^f|_j = L) \Rightarrow (\nu_2^f|_i = L \wedge \nu_2^f|_j = L)$. It can be easily shown that $\Pi((p \oplus_i q) \oplus_j r, \nu_1^f) \sim \Pi(p \oplus_i (q \oplus_j r), \nu_2^f)$. Axiom N_6 is proved similar to $A_{1,2}$: for any $\nu_1^f \in \mathit{VFConfig}(p)$, we define $\nu_2^f \in \mathit{VFConfig}(p[k/i])$ such that $\nu_2^f = \nu_1^f \cdot \lambda$ and $\nu_1^f|_i = \nu_2^f|_k$.

For axiom A_3, we show that for any $\nu^f \in \mathit{VFConfig}(p \oplus_i p)$ (which implicitly implies $\nu^f \in \mathit{VFConfig}(p)$), $\Pi((p \oplus_i p), \nu^f) \sim \Pi(p, \nu^f)$. It is trivial that $\mathcal{R} = \{(\Pi((p \oplus_i p), \nu^f)\} \cup \{(\Pi(t, \nu^f), \Pi(t, \nu^f))\}$ is a strong bisimulation. Axioms $N_{1,3}$ are proved similarly.

For axiom A_4, for any $\nu^f \in \mathit{VFConfig}(a.(p \oplus_i q))$ (which implicitly implies $\nu^f \in \mathit{VFConfig}(a.p \oplus_i a.q)$), $\Pi(a.(p \oplus_i q), \nu^f) \sim \Pi(a.p \oplus_i a.q, \nu^f)$. Regarding the value of $\nu^f|_i$, two cases can be distinguished. We only discuss the case that $\nu^f|_i = L$, as the other cases can be proven identically. Therefore, we show that

$$\mathcal{R} = \{(\Pi(a.(p \oplus_i q), \nu^f), \Pi(a.p \oplus_i a.q, \nu^f)), (\Pi((p \oplus_i q), \nu^f), \Pi(p, \nu^f))\} \cup \{(\Pi(t, \nu^f), \Pi(t, \nu^f))\}$$

is a strong bisimulation and satisfies the transfer conditions of Definition 2.1. For the pair $(\Pi(a.(p \oplus_i q), \nu^f), \Pi(a.p \oplus_i a.q, \nu^f)) \in \mathcal{R}$, suppose $\Pi(a.(p \oplus_i q), \nu^f) \xrightarrow{a} \Pi(p \oplus q_i, \nu^f)$ since by SOS rule *Prefix* $a.(p \oplus_i q) \xrightarrow{a,\nu_?} p \oplus_i q$, where $\nu_? \sqsubseteq \nu^f$. By SOS rules *Prefix* and *Select*, $a.p \oplus_i a.q \xrightarrow{a,\nu_?|_{i/L}} p$, $\nu_?|_{i/L} \sqsubseteq \nu^f$. There-

fore, $\Pi(a.p \oplus_i a.q, \nu^f) \xrightarrow{a} \Pi(p, \nu^f)$, and $\Pi((p \oplus_i q), \nu^f)\ \mathcal{R}\ \Pi(p, \nu^f)$. The same discussion holds when $\Pi(a.p \oplus a.q, \nu^f) \xrightarrow{a} \Pi(p, \nu^f)$. Furthermore, for the pair $(\Pi((p \oplus_i q), \nu^f), \Pi(p, \nu^f)) \in \mathcal{R}$, assume that $\Pi((p \oplus_i q), \nu^f) \xrightarrow{a} \Pi(p', \nu^f)$ since $p \oplus_i q \xrightarrow{a,\nu} p'$ and $\nu \sqsubseteq \nu^f$. Therefore, $p \xrightarrow{a,\nu'} p'$, where $\nu'|_{i/L} = \nu$, and $\nu' \sqsubseteq \nu^f$. Consequently $\Pi(p, \nu^f) \xrightarrow{a} \Pi(p', \nu^f)$ and $\Pi(p', \nu^f)\ \mathcal{R}\ \Pi(p', \nu^f)$. The same discussion holds when $\Pi(p, \nu^f) \xrightarrow{a} \Pi(p', \nu^f)$. Transfer conditions trivially hold for pairs like $(\Pi(t, \nu^f), \Pi(t, \nu^f)) \in \mathcal{R}$. Consequently $\mathcal{R}$ is a strong bisimulation, concluding that $a.(p \oplus_i q) \approx_{PL} a.p \oplus_i a.q$.

To prove axiom A_5, for any $\nu^f \in \mathit{VFConfig}((p \oplus_i q) + r)$ (which implicitly implies $\nu^f \in \mathit{VFConfig}((p + r) \oplus_i (q + r))$), it is straightforward to show that $\Pi((p \oplus_i q) + r, \nu^f) \sim \Pi((p + r) \oplus_i (q + r), \nu^f)$ witnessed by the strong bisimulation relation $\mathcal{R} = \{(\Pi((p \oplus_i q) + r, \nu^f), \Pi((p + r) \oplus_i (q + r), \nu^f))\} \cup \{(\Pi(t, \nu^f), \Pi(t, \nu^f))\}$. Axioms A_6, $P_{4,5}$ R_3, E_5, $N_{2,4,5}$, S_3 are proved similar to A_5.

For axiom C_1, it can be proved that for any $\nu^f \in \mathit{VFConfig}(p + q)$ (which implicitly implies $\nu^f \in \mathit{VFConfig}(q + p)$) $\Pi(p + q, \nu^f) \sim \Pi(q + p, \nu^f)$. It is easy to show that $\mathcal{R} = \{(\Pi(p + q, \nu^f), \Pi(q + p, \nu^f))\} \cup \{(\Pi(t, \nu^f), \Pi(t, \nu^f))\}$ is a strong bisimulation. Axioms C_{2-4}, $P_{1-3,6}$, $S_{1,2,4,5,6}$, $R_{1,2,4}$, and E_{1-4} are proved similarly.

Axioms Dec, UnFold, Fold, and Ung are standard [9]. We provide a full proof for the new axiom Dri. To prove axiom Dri, we show that for any $\nu^f \in \mathit{VFConfig}(\langle A|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle)$ (which implicitly implies $\nu^f \in \mathit{VFConfig}(\langle A|\{A \stackrel{def}{=} t_1\}\rangle \oplus_i \langle A|\{A \stackrel{def}{=} t_2\}\rangle)$), $\Pi(\langle A|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f) \sim \Pi(\langle A|\{A \stackrel{def}{=} t_1\}\rangle \oplus_i \langle A|\{A \stackrel{def}{=} t_2\}\rangle, \nu^f)$. We only discuss the case when $\nu^f|_i = L$, as the other can be dealt with in the same way. To this aim, we prove that $\mathcal{R} = \{(\Pi(\langle t|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle), \Pi(\langle t|\{A \stackrel{def}{=} t_1\}\rangle \oplus_i \langle t|\{A \stackrel{def}{=} t_2\}\rangle, \nu^f))\} \cup \{(\Pi(\langle t|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle), \Pi(\langle t|\{A \stackrel{def}{=} t_1\}\rangle, \nu^f))\}$ is a strong bisimulation.

The transfer conditions of Definition 2.1 for the pair $(\Pi(\langle t|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f), \Pi(\langle t|\{A \stackrel{def}{=} t_1\}\rangle \oplus_i \langle t|\{A \stackrel{def}{=} t_2\}\rangle, \nu^f))$ can be examined by structural induction over the syntax of t. The base case of induction for $t \equiv 0$ is trivial.

- if $t \equiv a.t'$, then by rule Prefix, $\langle a.t'|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle \xrightarrow{a,\nu_?} \langle t'|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle$ and $\nu_? \sqsubseteq \nu^f$. Similarly, $\langle a.t'|\{A \stackrel{def}{=} t_1\}\rangle \oplus_i \langle a.t'|\{A \stackrel{def}{=} t_2\}\rangle \xrightarrow{a,\nu_?|_{i/L}} \langle t'|\{A \stackrel{def}{=} t_1\}\rangle$. Hence, $\Pi(\langle a.t'|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t'|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f)$, $\Pi(\langle a.t'|\{A \stackrel{def}{=} t_1\}\rangle \oplus_i \langle a.t'|\{A \stackrel{def}{=} t_2\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t'|\{A \stackrel{def}{=} t_1\}\rangle, \nu^f)$, and $\Pi(\langle t'|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f)\ \mathcal{R}\ \Pi(\langle t'|\{A \stackrel{def}{=} t_1\}\rangle, \nu^f)$.

- if $t \equiv t_1 + t_2$, then by rule Choice, either $\langle t_1 + t_2|\{A \stackrel{def}{=} p \oplus_i q\}\rangle \xrightarrow{a,\nu_1} \langle t_1'|\{A \stackrel{def}{=} p \oplus_i q\}\rangle$ or $\langle t_1 + t_2|\{A \stackrel{def}{=} p \oplus_i q\}\rangle \xrightarrow{b,\nu_2} \langle t_2'|\{A \stackrel{def}{=} p \oplus_i q\}\rangle$, and $\nu_1 \sqsubseteq \nu^f$ or $\nu_2 \sqsubseteq \nu^f$, since $\langle t_1|\{A \stackrel{def}{=} p \oplus_i q\}\rangle \xrightarrow{a,\nu_1} \langle t_1'|\{A \stackrel{def}{=} p \oplus_i q\}\rangle$

or $\langle t_2|\{A \stackrel{def}{=} p \oplus_i q\}\rangle \xrightarrow{b,\nu_2} \langle t_2'|\{A \stackrel{def}{=} p \oplus_i q\}\rangle$. Hence, $\Pi(\langle t_1 + t_2|\{A \stackrel{def}{=} p \oplus_i q\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t_1'|\{A \stackrel{def}{=} p \oplus_i q\}\rangle, \nu^f)$ or $\Pi(\langle t_1 + t_2|\{A \stackrel{def}{=} p \oplus_i q\}\rangle, \nu^f) \xrightarrow{b} \Pi(\langle t_2'|\{A \stackrel{def}{=} p \oplus_i q\}\rangle, \nu^f)$. By induction, $\Pi(\langle t_1|\{A \stackrel{def}{=} p \oplus_i q\}\rangle, \nu^f) \sim \Pi(\langle t_1|\{A \stackrel{def}{=} p\}\rangle \oplus_i \langle t_1|\{A \stackrel{def}{=} q\}\rangle, \nu^f)$ and $\Pi(\langle t_2|\{A \stackrel{def}{=} p \oplus_i q\}\rangle, \nu^f) \sim \Pi(\langle t_2|\{A \stackrel{def}{=} p\}\rangle \oplus_i \langle t_2|\{A \stackrel{def}{=} q\}\rangle, \nu^f)$. Therefore, $\Pi(\langle t_1|\{A \stackrel{def}{=} p\}\rangle \oplus_i \langle t_1|\{A \stackrel{def}{=} q\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t_1'|\{A \stackrel{def}{=} p\}\rangle, \nu^f)$, or $\Pi(\langle t_2|\{A \stackrel{def}{=} p\}\rangle \oplus_i \langle t_2|\{A \stackrel{def}{=} q\}\rangle, \nu^f) \xrightarrow{b} \Pi(\langle t_2'|\{A \stackrel{def}{=} p\}\rangle, \nu^f)$, and $\Pi(\langle t_1'|\{A \stackrel{def}{=} p \oplus_i q\}\rangle, \nu^f) \mathcal{R} \Pi(\langle t_1'|\{A \stackrel{def}{=} p\}\rangle, \nu^f)$, and $\Pi(\langle t_2'|\{A \stackrel{def}{=} p \oplus_i q\}\rangle, \nu^f) \mathcal{R} \Pi(\langle t_2'|\{A \stackrel{def}{=} p\}\rangle, \nu^f)$. Consequently, $\Pi(\langle t_1 + t_2|\{A \stackrel{def}{=} p\}\rangle \oplus_i \langle t_1 + t_2|\{A \stackrel{def}{=} q\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t_1'|\{A \stackrel{def}{=} p\}\rangle, \nu^f)$ or $\Pi(\langle t_1 + t_2|\{A \stackrel{def}{=} p\}\rangle \oplus_i \langle t_1 + t_2|\{A \stackrel{def}{=} q\}\rangle \xrightarrow{b} \Pi(\langle t_2'|\{A \stackrel{def}{=} p\}\rangle, \nu^f)$. The same discussion holds when $\langle t_1+t_2|\{A \stackrel{def}{=} p\}\rangle \oplus_i \langle t_1+t_2|\{A \stackrel{def}{=} q\}\rangle \xrightarrow{a,\nu_1} \langle t_1'|\{A \stackrel{def}{=} p\}\rangle$ or $\langle t_1 + t_2|\{A \stackrel{def}{=} p\}\rangle \oplus_i \langle t_1 + t_2|\{A \stackrel{def}{=} q\}\rangle \xrightarrow{b,\nu_2} \langle t_2'|\{A \stackrel{def}{=} q\}\rangle$.

- if $t \equiv t_1 \oplus_j t_2$, $t \equiv t_1 \parallel t_2$, $t \equiv t_1 \between t_2$, $t \equiv t_1 \mid t_2$, $t \equiv t' \setminus L$, or $t \equiv t'[f]$, then the proof is almost identical to the previous case.

- if $t \equiv \langle A|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle$, then $\Pi(\langle A|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t_1'|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f)$ since $\langle A|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle \xrightarrow{a,\nu} \langle t_1'|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle$, where $\nu \sqsubseteq \nu_f$, and by SOS rules *call* and *Select*, $\langle t_1|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle \xrightarrow{a,\nu'} \langle t_1'|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle$, where $\nu'|_{i/L} = \nu$. Therefore, by induction $\Pi(\langle t_1|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f) \sim \Pi(\langle t_1|\{A \stackrel{def}{=} t_1\}\rangle, \nu^f)$, and hence $\Pi(\langle t_1|\{A \stackrel{def}{=} t_1\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t_1'|\{A \stackrel{def}{=} t_1\}\rangle, \nu^f)$, and $\Pi(\langle t_1'|\{A \stackrel{def}{=} t_1\}\rangle, \nu^f) \sim \Pi(\langle t_1'|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f)$. Thus, by SOS rules *Select* and *Call*, $\Pi(\langle A|\{A \stackrel{def}{=} t_1\}\rangle \oplus_i \langle A|\{A \stackrel{def}{=} t_2\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t_1'|\{A \stackrel{def}{=} t_1\}\rangle, \nu^f)$ and $\Pi(\langle t_1'|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f) \mathcal{R} \Pi(\langle t_1'|\{A \stackrel{def}{=} t_1\}\rangle, \nu^f)$. The same discussion holds when $\Pi(\langle A|\{A \stackrel{def}{=} t_1\}\rangle \oplus_i \langle A|\{A \stackrel{def}{=} t_2\}\rangle \xrightarrow{a} \langle t_1'|\{A \stackrel{def}{=} t_1\}\rangle$.

The transfer conditions of Definition 2.1 for the pair $(\Pi(\langle t|\{A \stackrel{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f), \Pi(\langle t|\{A \stackrel{def}{=} t_1\}\rangle))$ can be examined by structural induction over the syntax of t as discussed above.

B.4 Ground-Completeness of Axiomatization

We are going to prove Theorem 6.17, that the axiomatization of PL-CCS is ground-complete for closed, finite-state terms modulo product line bisimilarity. The idea behind the ground-completeness proof, following the approach of [118], is to

show that $t = s$ via the intermediate results $t = t \oplus_i s$ and $t \oplus_i s = s$, which imply that $t = t \oplus_i s = s$. The following lemmata are required before explaining the proof.

Lemma B.3. *Let t, s, and r be fully expanded. If $(t \oplus_i s) \oplus_j r \simeq_{PL} r$, where $i \neq j$, $i \notin bi(t) \cup bi(s)$ and $j \notin bi(r)$, then $t \oplus_j r \simeq_{PL} r$ and $s \oplus_j r \simeq_{PL} r$.*

Proof. We show that $t \oplus_j r \simeq_{PL} r$ as the other case is symmetric. Regarding Definition 6.4, we must show that

- for any ${\nu^f}'_1 \in \mathit{VFConfig}(t \oplus_j r)$, there exists ${\nu^f}'_2 \in \mathit{VFConfig}(r)$ such that $\Pi(t \oplus_j r, {\nu^f}'_1) \sim \Pi(r, {\nu^f}'_2)$.

  Assume ${\nu^f}'_1|_j = L$; hence, ${\nu^f}'_1$ can be written as $\nu_t \cdot \lambda$, where $\nu_t \in \mathit{VFConfig}(t)$. Consider ${\nu^f}_1 \in \mathit{VFConfig}((t \oplus_i s) \oplus_j r)$ such that ${\nu^f}_1|_i = L$, ${\nu^f}_1|_j = L$, and $\forall k \leq |\nu_t| \wedge (k \neq i)({\nu^f}_1|_k = {\nu^f}'_1|_k)$. Hence, by Definition 6.4, $(t \oplus_i s) \oplus_j r \simeq_{PL} r$ implies that there exists $\nu^f_2 \in \mathit{VFConfig}(r)$ such that $\Pi((t \oplus_i s) \oplus_j r, \nu^f_1) \sim \Pi(r, \nu^f_2)$. Define ${\nu^f}'_2 = {\nu^f}_2$. It immediately follows that $\Pi(t \oplus_j r, {\nu^f}'_1) \sim \Pi(r, {\nu^f}'_2)$.

  Assume ${\nu^f}'_1|_j = R$, so it can be written as ${\nu^f}'_1 = \nu_r \cdot \lambda$, where $\nu_r \in \mathit{VFConfig}(r)$. Consider ${\nu^f}_1 \in \mathit{VFConfig}((t \oplus_i s) \oplus_j r)$ such that ${\nu^f}_1|_j = R$, and $\forall k \leq |\nu_r|({\nu^f}_1|_k = {\nu^f}'_1|_k)$. Define ${\nu^f}'_2 = \nu_r$. It immediately follows that $\Pi(t \oplus_j r, {\nu^f}'_1) \sim \Pi(r, {\nu^f}'_2)$.

- for any ${\nu^f}'_2 \in \mathit{VFConfig}(r)$, there exists ${\nu^f}'_1 \in \mathit{VFConfig}(t \oplus_j r)$ such that $\Pi(t \oplus_j r, {\nu^f}'_1) \sim \Pi(r, {\nu^f}'_2)$. Take ${\nu^f}'_1$ such that ${\nu^f}'_1|_j = R$ and $\forall k \leq |\nu_r| \wedge (k \neq j)({\nu^f}'_1|_k = {\nu^f}'_2|_k)$. It follows immediately that $\Pi(t \oplus_j r, {\nu^f}'_1) \sim \Pi(r, {\nu^f}'_2)$.

□

Lemma B.4. *Let $\bigoplus_{i \leq n} p_i$, where $n > 0$, be a PL-CCS term such that p_is are CCS terms. Then $\Pi(\bigoplus_{i \leq n} p_i, \nu_f) \sim p_j$, where $\nu_f|_j = L$ and $\forall k < j(\nu_f|_k = R)$.*

Proof. It is straightforward to check that for a given CCS term p, $\Pi(p, \nu_f) \sim p$ since ν_f has no effect on deriving transitions of p (ν_f is only considered in rules *Select*, *RSelect*, and *LSelect*). Therefore, $\Pi(\bigoplus_{i \leq n} p_i, \nu_f) \xrightarrow{a} \Pi(p', \nu_f)$, since $\bigoplus_{i \leq n} p_i \xrightarrow{a,\nu} p'$, where $\nu \sqsubseteq \nu_f$. Hence, by SOS rule *Select*, $p_j \xrightarrow{a,\nu_?} p'$ and $\forall k < j(\nu|_k) = R$ and $\nu|_j = L$. □

Theorem. For all closed finite-state PL-CCS$_f$ terms t_1 and t_2, $t_1 \simeq_{PL} t_2$ implies $t_1 = t_2$.

Proof. By Theorem 6.20, PL-CCS$_f$ terms t and s can be derived by our axiomatization into fully expanded terms $t' \equiv \bigoplus_{i \leq n} p_i$ and $s' \equiv \bigoplus_{j \leq m} q_j$, where p_is and q_is are CCS term such that every recursive specification E included in p_is or

q_is is essentially finite state. Since our axiomatization subsumes CCS axiomatization, completeness of CCS axiomatization for closed finite-state CCS terms [9] implies $p \sim q \Leftrightarrow p = q$. The soundness of our axiomatization yields $t \simeq_{PL} t'$ and $s \simeq_{PL} s'$. By transitivity of product line bisimilarity, $t' \simeq_{PL} s'$. Therefore, it is enough to prove that $t' \oplus_i s' \simeq_{PL} s' \Rightarrow t' \oplus_i s' = s'$ and $t' \simeq_{PL} t' \oplus_i s' \Rightarrow t' = t' \oplus_i s'$, where i is free in s' and t'. These properties are sufficient to prove the theorem as follows: If $t' \simeq_{PL} s'$, then, by the fact that product line bisimilarity is reflexive (i.e., $t' \simeq_{PL} t'$ and $s' \simeq_{PL} s'$) and the fact that product line bisimilarity is a congruence on fully expanded PL-CCS terms, we have $t' \oplus_i t' \simeq_{PL} s' \oplus_i t'$ and $t' \oplus_i s' \simeq_{PL} s' \oplus_i s'$ (note that $t' \oplus_i s'$ is still fully expanded). The soundness of the axiomatization, more in particular the validity of Axiom A_3, implies that $t' \oplus_i t' \simeq_{PL} t'$ and $s' \oplus_i s' \simeq_{PL} s'$. Using symmetry and transitivity of product line bisimilarity, $t' \simeq_{PL} t' \oplus_i s'$ and $t' \oplus_i s' \simeq_{PL} s'$ are obtained. Thus, the above properties yield that $t' = t' \oplus_i s'$ and $t' \oplus_i s' = s'$. These last results can be combined to show that $t' = t' \oplus_i s' = s'$. Consequently, $t = t'$ and $s = s'$ together with $t' = s'$ result $t = s$.

For all fully expanded PL-CCS$_f$ terms $t' \oplus_i s'$, t' and s', where $t' \equiv \bigoplus_{i \leq n} p_i$ and $s' \equiv \bigoplus_{j \leq m} q_j$, $t' \oplus_i s' \simeq_{PL} s'$ implies $t' \oplus_i s' = s'$, is proven by induction on number of CCS terms of t', i.e. n. The proof that $t' \simeq_{PL} t' \oplus_i s'$ implies $t' = t' \oplus_i s'$ is similar and therefore omitted.

The base case of the induction corresponds to the case that $n = 0$. Therefore, $p_1 \oplus_i s' \simeq_{PL} s$ implies for ${\nu^f}_1$, where ${\nu^f}_1|_1 = L$, there exists ${\nu^f}_2$ such that $\Pi(p_1 \oplus_i s', {\nu^f}_1) \sim \Pi(s', {\nu^f}_2)$. By Lemma B.4, $\Pi(p_1 \oplus_i s', {\nu^f}_1) \sim p_1$ and $\Pi(s', {\nu^f}_2) \sim q_k$, where ${\nu^f}_2|_k = L$ and $\forall j < k({\nu^f}_2|_j = R)$. Therefore, $p_1 \sim q_k$ and consequently by completeness of the axiomatization of CCS, $p_1 = q_k$. Thus, $p_1 \oplus_i s' = q_k \oplus_i \bigoplus_{j \leq m} q_j =^{A_1, A_2, A_3} s'$.

Assume that for all $0 < w < n$ such that $t' \equiv \bigoplus_{i \leq w} p_i$, $t' \oplus_i s' \simeq_{PL} s'$ implies $t' \oplus_i s' = s'$. We prove that $t' \oplus_i s' \simeq_{PL} s'$, where $t' \equiv \bigoplus_{i \leq n} p_i$, implies $t' \oplus_i s' = s'$. By Lemma B.3, $t' \oplus_i s' \simeq_{PL} s'$ implies $p_1 \oplus_i s' \simeq_{PL} s'$ and $(\bigoplus_{1 < i \leq n} p_i) \oplus_i s' \simeq_{PL} s'$. By application of axiom N_6, $\bigoplus_{1 < i \leq n} p_i$ can be derived into $t'' \equiv \bigoplus_{k < n} p_k$. Therefore, by soundness of axiomatization and transitivity of product line bisimilarity, $(\bigoplus_{k < n} p_k) \oplus_i s' \simeq_{PL} s'$. By induction $p_1 \oplus_i s' = s'$, $t'' \oplus_i s' = s'$. Therefore, $t' \oplus_i s' = (p_1 \oplus_1 (\bigoplus_{1 < i \leq n} p_i)) \oplus_i s' =^{N_6} (p_1 \oplus_z (\bigoplus_{1 < i \leq n} p_i)) \oplus_i s' =^{A_1, A_2} (\bigoplus_{1 < i \leq n} p_i) \oplus_z (p_1 \oplus_i s') = (\bigoplus_{1 < i \leq n} p_i) \oplus_z s' = t'' \oplus_i s' = s'$, where z is a fresh index such that $z > m$, $z > n$, and $z \neq i$.

B.5 Proof of Theorem 6.24

We use the following so-called *reduction* lemma taken from [6] to complete the proof. The right-hand sides of the bi-implications express the unfolding of the fixed points: in case of minimum a single element p is removed, while for the maximum it is added.

Lemma B.5. *For ψ a monotonic function on a powerset $Pow(D)$ with $p \in D$, we have:*

$$p \in \mu V.\psi(V) \Leftrightarrow p \in \psi(\mu V.(\psi(V) \setminus \{p\})),$$
$$p \in \nu V.\psi(V) \Leftrightarrow p \in \psi(\nu V.(\psi(V) \cup \{p\})).$$

The proof Theorem of 6.24 follows:

"$\Rightarrow$" Suppose $r \simeq_{PL} s$ and let $\varphi \in mv - \mathfrak{L}_\mu$. We prove that for all valid full configurations $\nu_1^f \in VFConfig(r)$ and $\nu_2^f \in VFConfig(s)$ that $\Pi(r, \nu_1^f) \sim \Pi(s, \nu_2^f)$, $\nu_1^f \in [\![\varphi]\!](r)$ iff $\nu_2^f \in [\![\varphi]\!](s)$. The proof is managed by induction on the structure of φ.

1. If $\varphi = true$, then obviously $\nu_1^f \in [\![\varphi]\!](r)$ and $\nu_2^f \in [\![\varphi]\!](s)$.

2. If $\varphi = \varphi_1 \wedge \varphi_2$, then by definition $\nu_1^f \in [\![\varphi_1]\!](r)$ and $\nu_1^f \in [\![\varphi_2]\!](r)$, and the claim follows by straightforward induction.

3. If $\varphi = \varphi_1 \vee \varphi_2$, it is proved similar to the previous case.

4. If $\varphi = \langle a \rangle \varphi'$, then by definition $\nu_1^f \in [\![\varphi]\!](r)$, if $\nu_1^f \in \mathcal{R}_a(r, r')$ and $\nu_1^f \in [\![\varphi']\!](r')$ for some r'. Hence $\nu_1^f \in \mathcal{R}_a(r, r')$ implies that $r \xrightarrow{a,\nu_1} r'$ for some ν_1 such that $\nu_1 \sqsubseteq \nu_1^f$. Consequently $\Pi(r, \nu_1^f) \xrightarrow{a} \Pi(r', \nu_1^f)$. By assumption, $\Pi(r, \nu_1^f) \sim \Pi(s, \nu_2^f)$ implies $\Pi(s, \nu_2^f) \xrightarrow{a} \Pi(s', \nu_2^f)$ and $\Pi(r', \nu_1^f) \sim \Pi(s', \nu_2^f)$. Thus $s \xrightarrow{a,\nu_2} s'$ for some ν_2 such that $\nu_2 \sqsubseteq \nu_2^f$ and $\nu_2^f \in \mathcal{R}_a(s, s')$. Concluding, by induction $\nu_2^f \in [\![\varphi']\!](s')$, so the claim follows.

5. If $\varphi = [a]\varphi'$, then by definition $\nu_1^f \in [\![\varphi]\!](r)$ implies that for all s':

   - either $\nu_1^f \in \mathcal{R}_a(r, r')$ and $\nu_1^f \in [\![\varphi']\!](r')$: Hence $\nu_1^f \in \mathcal{R}_a(r, r')$ implies that $r \xrightarrow{a,\nu_1} r'$ for some ν_1 such that $\nu_1 \sqsubseteq \nu_1^f$. Consequently $\Pi(r, \nu_1^f) \xrightarrow{a} \Pi(r', \nu_1^f)$. By assumption, $\Pi(r, \nu_1^f) \sim \Pi(s, \nu_2^f)$ implies $\Pi(s, \nu_2^f) \xrightarrow{a} \Pi(s', \nu_2^f)$ and $\Pi(r', \nu_1^f) \sim \Pi(s', \nu_2^f)$. Thus $s \xrightarrow{a,\nu_2} s'$ for some ν_2 such that $\nu_2 \sqsubseteq \nu_2^f$ and $\nu_2^f \in \mathcal{R}_a(s, s')$. Concluding, by induction $\nu_2^f \in [\![\varphi']\!](s')$, so the claim follows.
   - or $\nu_1^f \notin \mathcal{R}_a(r, r')$: Hence there exists no transition that $r \xrightarrow{a,\nu_1} r'$ for some ν_1 such that $\nu_1 \sqsubseteq \nu_1^f$. Consequently $\Pi(r, \nu_1^f) \xrightarrow{a} \not\!\!\!\!\checkmark$. By assumption, $\Pi(r, \nu_1^f) \sim \Pi(s, \nu_2^f)$ implies $\Pi(r, \nu_1^f) \xrightarrow{a} \not\!\!\!\!\checkmark$, and hence $\nu_2^f \notin \mathcal{R}_a(s, s')$ for any s'.

6. If $\varphi = \mu Z.\phi$, then by Lemma B.5, $\nu_1^f \in [\![\varphi]\!](r)$ implies $\nu_1^f \in [\![\phi]\!]_{\rho[Z \mapsto \mu Z.\phi \setminus \{\nu_1^f\}]}(r)$. By induction $\nu_2^f \in [\![\phi]\!]_{\rho[Z \mapsto \mu Z.\phi \setminus \{\nu_2^f\}]}(s)$, and hence $\nu_2^f \in [\![\varphi]\!](s)$.

7. If $\varphi = \nu Z.\phi$, then by Lemma B.5, $\nu_1^f \in [\![\varphi]\!](r)$ implies $\nu_1^f \in [\![\phi]\!]_{\rho[Z \mapsto \nu Z.\phi \cup \{\nu_1^f\}]}(r)$. By induction $\nu_2^f \in [\![\phi]\!]_{\rho[Z \mapsto \nu Z.\phi \cup \{\nu_2^f\}]}(s)$, and consequently $\nu_2^f \in [\![\varphi]\!](s)$.

"$\Leftarrow$" Suppose $r \sim_L s$. We prove that for any valid full configuration $\nu_1^f \in VFConfig(r)$, there exists $\nu_2^f \in VFConfig(s)$ such that $\Pi(r, \nu_1^f) \sim \Pi(s, \nu_2^f)$. As r and s have finite-state behaviors, it is trivial that $\Pi(r, \nu_1^f)$ is finitely branching. Hence for $\nu_1^f \in VFConfig(r)$, we can find $\varphi \in mv - \mathfrak{L}_\mu$ such that it characterizes the strong bisimulation class for $\Pi(r, \nu_1^f)$, called *characteristic formula*. Following the approach of [3], each process t is characterized by the formula $\nu X_t. \bigwedge_{a,t',t \xrightarrow{a} t'} \langle a \rangle X_{t'} \wedge \bigwedge_a [a](\bigvee_{t',t \xrightarrow{a} t'} X_{t'})$, where $X_{t'}$ denotes the solution of the characteristic formula of t'. Intuitively, this formula defines the possible actions of the process by the diamond sub-formula and disables others by the square sub-formula. Since $\nu_1^f \in [\![\varphi]\!](r)$, by Definition 6.23, there exists $\nu_2^f \in [\![\varphi]\!](s)$. Therefore, $\Pi(r, \nu_1^f) \sim \Pi(s, \nu_2^f)$.

Similarly for any valid full configuration $\nu_2^f \in VFConfig(s)$, we can find $\nu_1^f \in VFConfig(r)$ such that $\Pi(r, \nu_1^f) \sim \Pi(s, \nu_2^f)$.

Summary

Formal Modeling and Analysis of Mobile Ad hoc Networks

Wireless ad hoc networks, in particular mobile ad hoc networks (MANETs), make communication easier and more flexible. However, their protocols tend to be difficult to design, due to the topology-dependent nature of wireless communication, and their distributed and adaptive operations to cope with a dynamic topology. Therefore it is desirable to model and verify such protocols using formal methods. To this aim, we introduce constrained labeled transition systems (CLTS) as a semantic model, to compactly capture the behavior of MANETs while mobility is implicitly specified. The transitions are constrained so as to indicate conditions over the underlying topology for which the behavior is valid. We present two modeling languages with different computation models to address main challenges of modeling MANETs, such as local broadcast, underlying topology, and its changes.

The first approach, presented in Chapter 3, is an algebraic framework for modeling and analysis of MANET protocols. The framework abstracts away from data link layer services by assuming reliable synchronous (local) broadcast. We discuss how the non-blocking property of local broadcast is guaranteed by making network processes input-enabled. We define the appropriate equivalence relation in this setting and provide a process theory to verify MANET protocols using equational reasoning. To utilize our complete axiomatization for analyzing the correctness of protocols at the syntactic level, we introduce a precongruence relation which abstracts away from a sequence of multi-hop communications, leading to an application-level action preconditioned by a multi-hop constraint over the topology. We illustrate the applicability of our framework through a simple routing protocol. To prove its correctness, we introduce a novel proof process, based on our precongruence relation.

The second approach, presented in Chapter 4, is an actor-based modeling language, with the aim to reduce the labor of modeling and generating the state space. We discuss how MANET protocols can be modeled efficiently at the semantic level, to make their verification feasible. This framework abstracts away from data link layer services by providing asynchronous (local) broadcast, unicast, and multicast communications, while assuming that message delivery is in order and is guaranteed for connected receivers. We illustrate the applicability

of our framework through two routing protocols, flooding and AODVv2-11, and show how their state spaces are reduced efficiently by the proposed techniques implemented in a tool. Furthermore, we demonstrate a loop formation scenario in AODV, found by our analysis tool. This has led to an adaptation of the AODV standard.

To model check MANET protocols with respect to the underlying topology and connectivity changes, we introduce a branching-time temporal logic which is interpreted over CLTSs in Chapter 5. The path quantifiers are parameterized by multi-hop constraints over topologies, to discriminate the paths over which the temporal behavior should be investigated; paths that violate the multi-hop constraints are not considered. A model checking algorithm is presented to verify MANETs, under the assumption of reliable communication. It is applied to analyze a leader election protocol and the packet-delivery property of AODVv2-11.

Software product lines (SPLs) facilitate reuse and customization in software development by genuinely addressing the concept of variability. Product Line Calculus of Communicating Systems (PL-CCS) is a process calculus for behavioral modeling of SPLs, in which variability can be explicitly modeled by a binary variant operator. The transitions of underlying semantic models are constrained to indicate the set of products (families) for which the behavior is valid. Hence, to extend the results over CLTSs, we study different notions of behavioral equivalence for PL-CCS in Chapter 6, based on Park and Milner's strong bisimilarity. These notions enable reasoning about the behavior of SPLs at different levels of abstraction. We study the compositionality property and the mutual relationship among these notions. We further show how the strengths of these notions can be consolidated in an equational reasoning method. Finally, we designate the notions of behavioral equivalence that are characterized by the property specification language for PL-CCS, called multi-valued modal μ-calculus.

Samenvatting

Formal Modeling and Analysis of Mobile Ad hoc Networks

Draadloze ad hoc networken, in het bijzonder mobiele ad hoc netwerken (MANETs), maken communicatie gemakkelijker en flexibeler. Het is echter lastig om correcte protocollen voor MANETs te ontwerpen, doordat draadloze communicatie afhankelijk is van de dynamische posities van de entiteiten in het netwerk, Ook kunnen entiteiten vaak niet direct met elkaar communiceren, door de beperkte reikwijdte van communicatie. Daarom is het belangrijk dergelijke protocollen te modelleren en verifiëren door middel van formele methoden.

Dit proefschrift introduceert een semantisch model, genaamd CLTS (constrained labeled transition system), waarin mobiliteit impliciet wordt gespecificeerd. Zoals de Engelse naam aangeeft wordt elke transitie in een CLTS beperkt, door expliciet aan te geven voor welke netwerktopologieën de transitie mogelijk is. Dit zorgt voor een compacte representatie van het gedrag van MANET-protocollen.

Het proefschrift presenteert twee verschillende modelleertalen. De eerste aanpak, in Hoofdstuk 3, is een algebraïsch raamwerk voor het modelleren en analyzeren van MANET-protocollen. Het raamwerk abstraheert van de dataverbindingslaag, door de aanname van betrouwbare synchrone (locale) communicatie. Er wordt bediscussieerd hoe kan worden gegarandeerd dat locale communicatie nooit blokkeert, door netwerkprocessen altijd in staat te stellen berichten te ontvangen. Ook wordt een equivalentierelatie geïntroduceerd die geschikt is voor deze aanpak. Een nieuwe procescalculus maakt het mogelijk de correctheid van MANET-protocollen aan te tonen met behulp van een equationele axiomatizering. Er wordt een precongruentie-relatie gedefinieerd die abstraheert van sequenties van communicaties, waardoor de axiomatizering toegepast kan worden op het syntactische niveau. De toepasbaarheid van het raamwerk wordt getoond aan de hand van een simpel routeerprotocol voor MANETs. In het correctheidsbewijs wordt een nieuwe bewijsmethode gebruikt, gebaseerd op de precongruentierelatie.

De tweede aanpak is een actor-gebaseerde modelleertaal die als doel heeft toestandsruimtes te vereenvoudigen. In Hoofdstuk 4 wordt aangetoond hoe MANET-protocollen aldus efficiënt gemodelleerd kunnen worden op het semantische niveau, waardoor hun verificatie haalbaar wordt. De aanpak is geïmple-

menteerd in een tool. De toepasbaarheid van het raamwerk wordt geïllustreerd op basis van twee routeerprotocollen voor MANETs, namelijk flooding and AODVv2-11. Een scenario waarin ongewenst een cyclisch pad wordt geformeerd door het AODV protocol is gevonden met behulp van het tool. Dit heeft geleid tot een aanpassing van de wereldwijde AODV standaard.

Hoofdstuk 5 introduceert een zogeheten branching-time temporele logica die wordt geïnterpreteerd over CLTSen, ten opzichte van de onderliggende dynamische netwerktopologie. Quantoren over paden worden geparametrizeerd door beperkingen op topologieën; alleen paden die deze beperkingen behouden worden in ogenschouw genomen. Er wordt een zogeheten model checking algoritme gegeven dat efficiënt verifieert in welke toestanden in een CLTS een gegeven temporele formule geldig is. Een implementatie van dit algoritme is toegepast om een protocol te analyzeren waarmee een leider in een MANET gekozen wordt, alsmede om te verifiëren of pakketten altijd hun bestemming bereiken in het AODV protocol.

Een software product lijn (SPL) faciliteert hergebruik en maatwerk in de ontwikkeling van software, door rekening te houden met variabiliteit. De Product Line Calculus of Communicating Systems (PL-CCS) is een procescalculus voor het formeel modelleren van SPLen, waarin variabiliteit expliciet wordt gemodelleerd door een binaire variant-operator. The transities van de onderliggende semantische modellen worden beperkt op basis van de verzameling producten (families) waarvoor een transitie geldig is. Hoofdstuk 6 oppert verschillende noties van equivalentie voor PL-CCS, gebaseerd op sterke bisimulariteit. Deze noties maken het mogelijk om op verschillende niveaus van abstractie te redeneren over het gedrag van SPLen. De compositionaliteitseigenschap voor deze noties van equivalentie wordt bestudeerd, alsmede de onderlinge relaties tussen de verschillende noties. Ook wordt aangetoond hoe deze noties ten grondslag liggen aan een equationeel raamwerk voor PL-CCS. Tenslotte wordt een specificatietaal, een meerwaardige modale μ-calculus, gedefinieerd om eigenschappen uit te drukken voor PL-CCS.