Modeling and Verification Techniques for Ad Hoc Network Protocols

A DISSERTATION PRESENTED

BY

Anu Singh

TO

The Graduate School

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

Doctor of Philosophy

IN

COMPUTER SCIENCE

Stony Brook University

August 2009

Stony Brook University

The Graduate School

Anu Singh

We, the dissertation committee for the above candidate for
the degree of Doctor of Philosophy,
hereby recommend acceptance of this dissertation.

This dissertation is accepted by the Graduate School.

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

Modeling and Verification Techniques for Ad Hoc Network Protocols

by

Anu Singh

Doctor of Philosophy

in

Computer Science

Stony Brook University

2009

Ad hoc networks are widely used for unmanaged and decentralized operations. Mobile ad hoc networks (MANETs) and wireless sensor networks are examples of ad hoc networks used for supporting self-configured and non-monitored services such as network routing and surveillance.

We developed the ω-calculus, a process-algebraic framework for formally modeling and reasoning about ad hoc networks and their protocols. The ω-calculus naturally captures essential characteristics of MANETs, including the ability of a MANET node to broadcast a message to any other node within its physical transmission range (and no others), and to move in and out of the transmission range of other nodes in the network. A key feature of the ω-calculus is the separation of a node's communication and computational behavior, described by an ω-process, from the description of its physical transmission range, referred to as an ω-process *interface*. The ω-calculus uses the notion of groups to model local broadcast-based communication, and separates the description of the actions of a protocol (called processes) from that of the network topology (specified by sets of groups, called interfaces of processes). As a result, the problems of verifying reachability properties and bisimilarity are decidable for a large class of omega calculus specifications (even in the presence of arbitrary node movement).

Ad hoc network protocols pose a unique verification problem called *instance explosion* because an AHN, with a fixed number of nodes, can assume exponential number of topologies. We have developed an automata-theoretic framework, based on key features of the omega-calculus, for the verification of ad hoc network protocols over unknown network topologies. Instance explosion is mitigated by using constraints to represent *sets* of topologies. A corresponding symbolic verification algorithm can efficiently infer the set of topologies for which an AHN protocol possesses a given correctness property.

We have also developed a partial model checker for parameterized systems of omega-calculus nodes. For a node M in an n-node system and a given formula φ, the partial model checker treats M as a *property transformer*, inferring the formula $\Pi(M)(\varphi)$ that must hold in the (n-1)-node system with M removed. Our technique is such that n may be infinite, thereby supporting the verification of infinite families of processes.

To My Family

Contents

List of Figures

List of Tables

Acknowledgements

I would like to sincerely thank all people who have supported and encouraged me to complete this thesis.

Foremost, I would like to express my sincere gratitude to my advisor Prof. C. R. Ramakrishnan for his continuous support, guidance and patience during my doctoral study and research. His willingness to help his students and closely work with them made my research experience highly rewarding. Through his immense knowledge and breadth of expertise he offered me opportunities and led me to work on diverse exciting projects.

I would like to express my sincere thanks to my co-advisor Prof. Scott A. Smolka, an eminent researcher in the field of formal methods and verification. His rigor and passion of research helped me develop insights into formal methods. His guidance and constructive comments on research, writing and presentation have been very helpful.

I am grateful to the rest of my thesis committee : Prof. Radu Grosu, Prof. Scott D. Stoller and Prof. Rance Cleaveland for their encouragement and insightful comments. Their suggestions helped in improving the quality of this thesis.

I would like to extend my thanks to Prof. I. V. Ramakrishnan for his support and encouragement all these years. He provided a motivating environment to work in the Applied Logic Lab and conduct exciting research.

I had the opportunity to work with Prof. David Warren. Interaction with him helped me realize the potential for research through smallest observations. I also had the opportunity to interact with Prof. Jennifer Wong. Her cordial nature and eagerness to help with minute details helped me quickly acquire knowledge on new topics and expand my research interests.

I thank all my lab-mates in the Applied Logic Lab who made it a convivial place to work. Their efforts and accomplishments inspired me in research. I treasure all precious moments we shared in the lab. I am also thankful to the departmental staff for providing computing and administrative facilities.

My deepest gratitude goes to my entire family for their constant love and support throughout my life; this thesis would have not been possible without them. I am deeply indebted to my parents for their love and inspiration. My husband provided me continuous support as a friend and as a guide. His belief in me to accomplish this thesis kept me motivated all these years. I feel fortunate to get immense support and encouragement from my family. I dedicate my thesis to my them.

Chapter 1

Introduction

An ad hoc network (AHN) is a collection of autonomous nodes connected by wireless links. Each node N has a physical transmission range within which it can directly transmit data to other nodes. Any node that falls within N's transmission range is considered a *neighbor* of N. Mobile ad hoc networks (MANETs) are a special case of AHNs, where nodes can move freely, leading to rapid change in the network's communication topology.

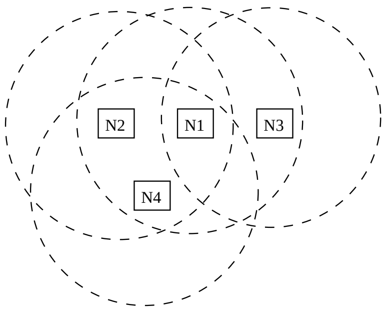


Figure 1: Ad Hoc Network Example

An example of an AHN is shown in Figure 1. There are four nodes labeled N_1, N_2, N_3 and N_4 in the network. The dotted circle centered around a node represents its transmission range. Thus, N_1 is within the transmission range of N_2, N_3, and N_4 and vice versa, and N_2 and N_4 are in each other's transmission range. The figure shows the topology of the network at an instant of time as nodes are free to move in and out of each other's transmission range.

AHNs are used in a variety of application domains including unmanaged and decentralized operations. Mobile ad hoc networks (MANETs) and wireless sensor networks (WSNs) are examples of AHNs used for supporting self-configured and non-monitored services such as network routing, surveillance, etc. Minimal configuration and quick deployment make them suitable for ad hoc applications, including emergency situations like natural disasters or military operations, health and environment monitoring. Such applications are safety-critical and demand reliability. Thus, it is important to verify the behavior of an AHN protocol before its use in a real network. In the computer networks community, system verification is mainly done through simulation. Though simulation can provide performance guarantees for a system, it cannot guarantee absence of errors, due to some unseen erroneous system behavior during simulation. To this end, formal methods serve as a suitable tool for verifying concurrent system behavior. In this thesis, we focus on formalisms for modeling and verifying AHNs and their associated protocols.

Two aspects of AHNs make them especially difficult to model using existing formal specification languages such as process algebras. First, AHNs use wireless links for local broadcast communication: an AHN node can transmit a message simultaneously to all nodes within its transmission range, but the message cannot be received by any node outside that range. Secondly, the neighborhood of nodes that lie within the transmission range of a node can change unpredictably due to node movement, thereby altering the set of nodes that can receive a transmitted message.

Ideally, the specification of an AHN node's control behavior should be independent of its neighborhood information. Since, however, the eventual recipients of a local broadcast message depend on this information, a model of an AHN or MANET protocol given in a traditional process calculus must intermix the computation of neighborhood information with the protocol's control behavior. This tends to render such models unnatural and unnecessarily complex. In order to directly and succinctly model these features, a formal framework is required that enables the concise expression of node movement and local broadcast communication particular to AHNs.

1.1 Problem Addressed in this Thesis

The environments in which AHN applications operate are generally unreliable due to node failure and mobility and intermittent wireless communication links. AHNs are resource-constrained and have arbitrary topology. These problems require AHN applications to be robust, and necessitate behavioral assurances for AHN applications before their actual deployment. AHNs introduce a rich domain of networking applications with complex control structures and optimizations for resource usage. The complexity of AHN applications make them difficult to model and verify.

Another prominent source of complexity in the verification of AHN protocols is the fact that the number of network topologies grows exponentially with the square of the number of nodes. Due to the vast space of possible network topologies, the verification of AHN protocols is a computationally intensive if not intractable task. Consider, for example, the verification of the LMAC medium access control [vHH04] protocol performed in [FvHM07]. The approach taken in there was to separately model check each of the possible network topologies (modulo isomorphism) for a fixed number of nodes in order to detect those that might lead to *unresolved collisions*. The problem with this approach is that as the number of nodes in the network grows, the number of possible topologies grows exponentially, posing an *instance explosion* problem for verification.

In this thesis, we present a formal framework for modeling and analysis of AHNs and their protocols. The framework consists of a process-algebraic modeling language called the ω-calculus, which separates the description of a node's behavior from the description of the network topology. The modeling language provides abstraction for node locations in contrast to using real numbers to represent node locations as in [Mer07], thus making verification decidable for finite-control models. The framework also consists of a procedure to check if two AHNs are bisimilar, i.e. they exhibit the same behavior. The largest bisimulation is defined to be a congruence, i.e. two AHNs that are bisimilar can replace each other in any network context. We also extend this basic framework with data operations for the modeling and verification of complex AHN protocols.

This thesis also presents an efficient symbolic verification algorithm to combat instance explosion. The symbolic algorithm uses constraints to represent sets of topologies. The algorithm can be used to verify AHN systems for all possible instances of topologies in a single verification run. It can also infer the set of topologies for which a property holds in the system.

This thesis also addresses the problem of parameterized verification of AHN protocols that enables the verification of infinite families of ω-calculus nodes. In comparison to prior works on parameterized verification [And95, EN96, BR06, YBR06], the new concerns that arise in case of (mobile) ad hoc network protocols are broadcast-based communication and node mobility.

1.2 Overview of Our Approach

In this thesis, we present a unified framework for the following:

- Modeling and abstraction of dynamic network topology and local broadcast communication in an AHN, and verification of AHN protocols.
- Symbolic representation of network topologies, enabling verification of AHN protocols over multiple network topologies in a single run.
- Parameterized verification of AHN protocols.

1.2.1 Modeling and Verification Framework

We developed the ω-calculus, a conservative extension of the π-calculus [MPW92] that has been designed expressly to address the MANET modeling problems outlined in Section 1.1. A key feature of the ω-calculus is the separation of a node's communication and computational behavior, described by an ω-process, from the description of its physical transmission range, referred to as an ω-process *interface*. This separation allows one to model the control behavior of a MANET protocol, using ω-processes, independently from the protocol's underlying communication topology, using process interfaces. (A similar separation of concerns has been achieved in several recently introduced process calculi for wireless and mobile networks [NH06, MS06, Mer07, God07], but not as simply and naturally as in the ω-calculus.) An ω-process interface is a set of *groups*, each of which operationally functions as a local broadcast port. Mobility

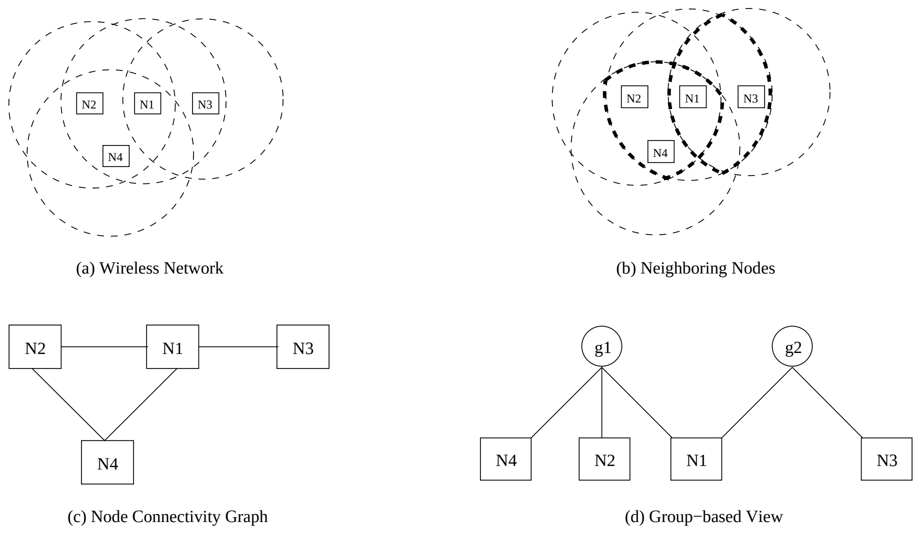


Figure 2: Multiple views of a MANET network.

is captured in the ω-calculus by the dynamic creation of new groups and dynamically changing process interfaces.

As an illustrative example of the ω-calculus, consider the MANET of Fig. 2(a) (same as in Fig. 1). We assume that the transmission ranges of all nodes are identical, and hence connectivity is symmetric. The assumption of symmetry makes the notation cleaner, although the assumption can be readily removed, as discussed later in this section.

Fig. 2(b) highlights the *maximal sets of neighboring nodes* in the network, one covering N_1, N_2, and N_4, and the other covering N_1 and N_3. A maximal set of neighboring nodes corresponds to a *maximal clique* in the network's node connectivity graph (Fig. 2(c)), and, equivalently, to an ω-calculus *group*, as illustrated in Fig. 2(d). The set of groups to which a node is connected is specified by the *interface* of the underlying process; i.e. the process executing at the node. Thus, the ω-calculus expression for the network is the parallel composition $N_1|N_2|N_3|N_4$, where $N_1 = P_1\!:\!\{g_1, g_2\}$, $N_2 = P_2\!:\!\{g_1\}$, $N_3 = P_3\!:\!\{g_2\}$, $N_4 = P_4\!:\!\{g_1\}$, for process expressions P_1, P_2, P_3 and P_4.

Note that process interfaces may contain groups that do not correspond to maximal cliques. Groups that do not represent any additional connectivity information are redundant. Group g_2 of Fig. 3 is an example of a redundant group. A *canonical* form for ω-calculus expressions can be defined in which redundant groups are elided.

Fig. 2 provides multiple views of the MANET's topology at a particular moment in time. As discussed below, the network topology may change over time due to node movement, a feature of MANETs captured operationally in the ω-calculus via dynamic updates of process interfaces.

Local Broadcast in the ω-calculus. The ω-calculus action to locally broadcast a value x is $\overline{\mathbf{b}}x$, while $\mathbf{r}(y)$ is the action for receiving a value y. Thus, when a process transmits a message, only the message x to be sent is included in the specification. The set of possible recipients depends on the process's current interface: only those processes that share a common group with the sender can receive the message and this information is not part of the syntax of local broadcast actions. In the example of Fig. 2, if P_2 can broadcast a message and P_1, P_3, P_4 are willing to receive it, then the expression

$$N = \mathbf{r}(x).P_1'\!:\!\{g_1, g_2\} \;\mid\; \overline{\mathbf{b}}u.P_2'\!:\!\{g_1\} \;\mid\; \mathbf{r}(y).P_3'\!:\!\{g_2\} \;\mid\; \mathbf{r}(z).P_4'\!:\!\{g_1\}$$

may evolve to

$$N = P_1'\{u/x\}\!:\!\{g_1, g_2\} \;\mid\; P_2'\!:\!\{g_1\} \;\mid\; \mathbf{r}(y).P_3'\!:\!\{g_2\} \;\mid\; P_4'\{u/z\}\!:\!\{g_1\}$$

Observe that P_3 does not receive the message since N_3 is not in N_2's neighborhood. It should be noted that communication is assumed to be lossy, and hence even nodes that are within a sender's transmission range may not receive a message.

When the interfaces of two nodes share a group name, the nodes are in each others' transmission ranges. We can remove the assumption of symmetric connections by partitioning the interface into transmission and reception parts. Then a node N_1 can send a message that can be received by node N_2 if the transmission interface of N_1 overlaps with the reception interface of N_2. Note that N_2's transmission interface and N_1's reception interface may be disjoint. This captures the scenario where N_2 is in N_1's transmission range, but N_1 is not in N_2's transmission range. While asymmetric connections can be handled in principle, this introduces notational clutter. Consequently, our technical development for the ω-calculus assumes symmetric connections.

Node mobility in the ω-calculus. Node mobility is captured through the dynamic creation of new groups and dynamically changing process interfaces. Fig. 3

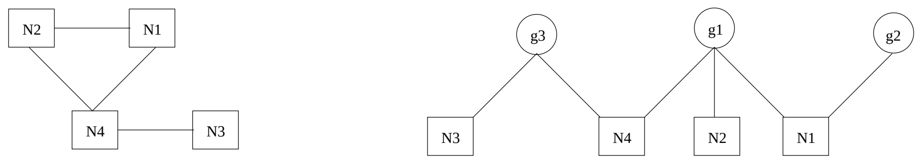


Figure 3: (a) Node Connectivity Graph after N_3's movement and (b) View in ω-calculus.

shows the topology of the network of Fig. 2 after N_3 moves away from N_1's transmission range and into N_4's transmission range. N_3's movement means that the ω-calculus expression

$$(\nu g_1)(\nu g_2)(P_1 \!:\! \{g_1, g_2\} \;|\; P_2 \!:\! \{g_1\} \;|\; P_3 \!:\! \{g_2\} \;|\; P_4 \!:\! \{g_1\})$$

evolves to

$$(\nu g_1)(\nu g_2)(P_1 \!:\! \{g_1, g_2\} \;|\; P_2 \!:\! \{g_1\} \;|\; (\nu g_3)(P_3 \!:\! \{g_3\} \;|\; P_4 \!:\! \{g_1, g_3\}))$$

The new group g_3 in the above expression represents the new maximal set of neighboring nodes N_3 and N_4 that arises post-movement. We use the familiar νg notation for group-name scoping.

When process interfaces are allowed to change arbitrarily, the network topology may change without any restriction. Correctness properties of many MANET algorithms and protocols may hold only in certain restricted class of topologies. We restrict node movement in the ω-calculus by imposing an invariant over a network's topology, called the connectivity invariant. The connectivity invariant must be preserved whenever the topology changes. Note that a connectivity invariant of "true" will allow arbitrary node movement.

Nodes vs. Processes. In an ω-calculus specification, nodes typically represent physical devices; as such, the calculus does not provide a primitive for node creation. Process creation, however, is supported, as processes model programs and other executables that execute within the confines of a device.

The ω-calculus framework [SRS08, SRS09a] and its properties are described in Chapters 3 and 4. The practical utility of the framework is demonstrated through case studies involving several AHN protocols in Chapter 5.

1.2.2 Constraint-Based Verification

Due to the ad-hoc nature of AHNs, an n-node AHN can assume any one of the possible $O(2^{n^2})$ topologies. This vast space of possible network topologies makes the verification of AHN protocols a computationally intensive if not intractable task. Consider, for example, the verification of the LMAC medium access control [vHH04] protocol performed in [FvHM07]. The approach taken in [FvHM07] was to separately model check each of the possible network topologies (modulo isomorphism) for a fixed number of nodes in order to detect those that might lead to *unresolved collisions*. An unresolved collision occurs when neighboring nodes (connected by at most two links) without a common neighbor attempt to transmit within the same time slot; due to the lack of a common neighbor, the collision remains undetected. The problem with this approach is that as the number of nodes in the network grows, the number of possible topologies grows exponentially, posing an *instance explosion* problem for verification.

To combat instance explosion, we developed a new, constraint-based *symbolic verification technique for ad-hoc network protocols*. The basic idea behind our approach is as follows. We use the key features of the ω-calculus as the basis for the development of an automata-theoretic framework. As in the ω-calculus, AHNs are represented as a collection of nodes of the form $P : I$, where P is a sequential process and I is an *interface*. An interface is a set of *groups*, with each group corresponding to a clique in the network topology. Dually, a group is used as a local-broadcast (multicast) communication port. Two nodes in the network can communicate (are within each other's transmission range) only if there respective interfaces have a non-null intersection (share a common group).

To achieve a *symbolic* representation of an AHN, we treat process interfaces as variables and introduce a constraint language for representing topologies. Terms of the language are simply conjunctions of *connection* and *disconnection constraints* of the form $conn(\mathcal{J}_i, \mathcal{J}_j)$ and $dconn(\mathcal{J}_i, \mathcal{J}_j)$, respectively. Here, $\mathcal{J}_i$ and $\mathcal{J}_j$ are interface variables, and $conn(\mathcal{J}_i, \mathcal{J}_j)$ signifies that $\mathcal{J}_i$ and $\mathcal{J}_j$ are connected ($\mathcal{J}_i \cap \mathcal{J}_j \neq \emptyset$), while $dconn(\mathcal{J}_i, \mathcal{J}_j)$ means that $\mathcal{J}_i$ and $\mathcal{J}_j$ are disconnected ($\mathcal{J}_i \cap \mathcal{J}_j = \emptyset$). As such, each term of the language symbolically represents a *set* of possible topologies.

Given this symbolic representation of AHNs, one can now ask *model-checking queries* of the form: under what evaluations (i.e. topologies) of the symbolic interface variables does the reachability property in question hold? Our *symbolic reachability algorithm* explores the symbolic state space of an AHN. A *symbolic state* is a pair of the form (s, γ), where s is a network state comprising both the locations of the component processes and valuations of their local variables, and γ is a term from our topology constraint language. A symbolic transition from (s, γ) to (s', γ') is constructed by adding constraints to γ to obtain γ' whenever a communication (local broadcast) occurs. Assuming the communication involves process P_i as the broadcaster, the following constraints will be added: those of the form $conn(\mathcal{J}_i, \mathcal{J}_j)$, where P_j is a process capable of performing a corresponding receive action and deemed to fall within the transmission range of P_i; and those of the form $dconn(\mathcal{J}_i, \mathcal{J}_k)$, where P_k is also a process capable of performing a receive action and deemed *not* to fall within P_i's transmission range.

We present an efficient symbolic reachability algorithm to verify reachability properties of symbolic AHNs, and demonstrate its practical utility by applying it to the problem of detecting unresolved collisions in the LMAC protocol [vHH04]. Our results show that our symbolic approach to query-based model checking is highly effective: in the case of a 6-node network, our symbolic reachability algorithm explored only 2,082 symbolic topologies, compared to a possible 32,768 actual topologies. Moreover, all 2,082 symbolic topologies were considered *in a single verification run*. In contrast, for the same property, the authors of [FvHM07] considered no more than a 5-node network, using 61 separate verification runs, one for each unique (modulo isomorphism) concrete topology. The constraint-based verification algorithm [SRS09b] and its demonstration through the case study of LMAC protocol [vHH04] is presented in Chapter 6.

1.2.3 Parameterized Verification

Communication networks describe an infinite family of systems, parameterized by the number of concurrent components in the system. Traditional model-checking techniques are restricted to verifying properties of a given instance of a parameterized system. A number of techniques have been developed to verify parameterized systems for all instances of their parameter space. We address the problem of parameterized verification of AHNs.

A compositional model checking technique for automatic verification of infinite families of systems specified in CCS [Mil89] process algebra has been presented in [BR06]. CCS expressions for system specifications are interpreted over a domain of modal mu-calculus [JB01] formulas. Processes are treated as property transformers using partial model checking [And95]. A similar approach has been adopted in [YBR06] for verifying infinite-families of mobile systems specified in the π-calculus [MPW92]. The partial-model-checking technique used in [BR06, YBR06] treats each process in a system as a property transformer. For a property φ that is to be verified of an N-process system, a partial model checker is used to infer property φ' that must hold of an (N-1)-process system. If the sequence of formulas φ, φ', $\ldots$ thus constructed converges, and the limit is satisfied by the deadlocked process, it is concluded that the N-process system satisfies φ.

The partial model checking technique has been described more formally in [YBR06] as follows: given a property (formula) φ and a system containing a process P (specified using some process algebra), compute the property φ' that should hold in P's *environment* (say, Q) if φ holds in $P|Q$. The property *transformer* of a process P, denoted by $\Pi(P)$, is such that: $\forall Q.(P|Q \models \varphi) \Leftrightarrow (Q \models \Pi(P)(\varphi))$.

Following the approach of [BR06, YBR06], we developed a partial-model-checking technique for the verification of parameterized AHNs that allow us to verify infinite families of AHN protocols. The key difference between this work and prior work [BR06, YBR06] is the broadcast-based communication vs. binary communication. Broadcast raises issues for partial model checking because it cannot be known in advance how many concurrent processes in the environment of a broadcasting process will receive the broadcast. This is not the case in binary communication where there can be only one receiver process that can synchronize with the send action of a process. The partial-model-checking technique for AHN protocols is described in Chapter 7.

Chapter 2

Background and Related Work

2.1 Formal Methods and Process Calculi

Formal Methods. Formal methods are mathematically based techniques used to specify, develop and verify systems [Cla96]. Specification involves describing the behavior of a system in a language with a mathematically defined syntax and semantics. The properties desired of a system are also specified formally. Verification involves checking whether the system description meets the specified properties.

Process Calculi. Process calculi are formal techniques for specifying and reasoning about the behavior of concurrent and communicating systems. Pioneering process calculi are the Calculus of Communicating Systems (CCS) [Mil89] and Communicating Sequential Processes (CSP) [Hoa85]. CCS is based on point-to-point (binary) communication. Process actions are either input/output on ports or the internal action. Input on a port α is represented by the port itself (α) and output on α is represented by the complementary port $\overline{\alpha}$. Two agents that can perform actions involving complementary ports can synchronize and evolve simultaneously. CSP is based on multi-way communication. All the processes that can perform an action involving a port name can synchronize. The calculus of broadcasting systems (CBS) [Pra93a, Pra93b, Pra94, Pra95] is a calculus with broadcast as the basic communication primitive. CBS does not use a notion of channel or port for transmitting values. CCS, CSP and CBS do not provide mechanism to model process mobility.

Process Calculi for Mobile Processes. The π-calculus [MPW92] is one of the first process calculi developed for mobile processes. Names are used to represent communication links as well as communicable data across links. Links are considered as references to processes. Communication is binary, i.e. only two processes can communicate at a time. The concept of using link names as data, together with the ability to generate fresh and unique names, is the basis for the π-calculus and gives it the expressive power to model dynamically reconfigurable systems.

Though change in the interconnection of processes can be modeled using the π-calculus, one-to-many (synchronous) communication cannot be expressed. It has been shown in [EM99] that it is difficult to encode broadcast in a calculus based on point-to-point communication. This motivated the development of $b\pi$-calculus [EM01] which is based on the π-calculus and CBS. The basic communication primitive in $b\pi$-calculus is broadcast. The $b\pi$-calculus can be used to model a notion of process mobility similar to that in the π-calculus.

Many variants of the π-calculus have been proposed: asynchronous π-calculus [HT91], fusion calculus [PV98] (a simplification of π-calculus with a more symmetric form of communication), the spi-calculus [AG97] (designed for the specification and analysis of cryptographic protocols), the join-calculus [FG96], and the typed π-calculus [DZG00] which develops a notion of groups for typing channels.

The π-calculus does not directly model the distribution of processes in different localities, or migrations of processes from one location to another. To overcome this limitation of π-calculus, process calculi with an explicit notion of locality and explicit primitives for process migration have been developed. Such process calculi are categorized as *distributed process calculi*. The addition of an explicit notion of location to a process calculus was introduced in [AP94]. Subsequently, a distributed version of the π-calculus, the $D\pi$-calculus [HR98, FH05], a language for specifying dynamically evolving networks of distributed processes, was proposed. The $D\pi$-calculus extends the π-calculus with notions of remote execution and migration. The Ambient calculus [CG98] is a process calculus for modeling mobile computing (computation carried out in mobile devices) and mobile computation (mobile code that moves between devices) in a single framework.

There are two notions of mobile processes addressed in the process calculi literature:

- Labile processes: Processes which undergo dynamic changes in their interaction structure.
- Motile processes: Processes which can exhibit motion by changing their physical locations.

The π-calculus is suitable for modeling labile processes. The ambient calculus and distributed process calculi can be used to model motile processes.

2.2 Related Work

Several process calculi have recently been developed for wireless and mobile ad hoc networks. The closest to our work are CBS# [NH06], CWS [MS06], CMN [Mer07], and CMAN [God07]. These calculi provide local broadcast and separate control behavior from neighborhood information. However, there are significant differences between these calculi and ours, which we now discuss. CBS# [NH06], based on the CBS process algebra of [Pra95], supports a notion of located processes. Node connectivity information is given independently of a system specification in terms of node connectivity graphs. The effect of mobility is achieved by nondeterministically choosing a node connectivity graph from a family of such graphs when a transition is derived. In contrast, the ω-calculus offers a single, integrated language for specifying control behavior and connectivity information, and permits reasoning about changes to connectivity information within the calculus itself.

In CWS [MS06], node location and transmission range are a part of the node syntax. Node movement is not supported, although the authors suggest the addition of primitives for this feature. CWS is well-suited for modeling device-level behaviors (e.g., interference due to simultaneous transmissions) in wireless systems.

In CMN [Mer07], a MANET node is a named, located sequential process that can broadcast within a specific transmission radius. Both the location and transmission radius are values in a physical coordinate system. Nodes are designated as mobile or stationary, and those of the former kind can move to an arbitrary location (resulting in a tau-transition). Bisimulation as defined for CMN is based on a notion of physically located observers. A calculus based on physical locations may pose problems for model checking as a model's state space would be infinite if locations are drawn from a real coordinate system.

In CMAN [God07], each node is associated with a specific *location*. Furthermore, each node n is annotated by a *connection set*: the set of locations of nodes to which n is connected. Connections sets thus determine the network topology. Synchronous local broadcast is the sole communication primitive. The connection set of a node explicitly identifies the node's neighbors. Consequently, when a node moves, its neighbors actively participate by removing from (or adding to) their connection sets the location of the moving node. This explicit handling of connection information affects the modularity of the calculus's semantics (the definition of bisimulation, in particular), and may preclude reasoning about open systems. In contrast, in the ω-calculus, neighborhood information is implicitly maintained using groups, thereby permitting us to define bisimulation relations in a natural way.

Other calculi for mobile processes that have been proposed in the literature include the π-calculus [MPW92], HOBS [OPT02], distributed process calculus $D\pi$ [HR98], and the ambient calculus [CG98]. These calculi do not support broadcast. Some calculi that support broadcast as a primitive are the $b\pi$-calculus [EM01] and PRISMA [BL08]. The $b\pi$-calculus adds broadcast communication as a primitive to the π-calculus and provides the same mechanism for mobility as in the π-calculus. PRISMA is a parametric calculus that can be instantiated with different interaction policies, and provides a uniform framework for expressing different synchronization models such as unicast and broadcast. Mobility in PRISMA is provided by name-passing as in the π-calculus. These calculi could be used to model MANETs but not as in a concise and natural fashion as with the ω-calculus because they intermix the specification of network structure with the specification of the control behavior of a protocol.

Chapter 3

Syntax and Transitional Semantics of the ω-Calculus

We begin this section by presenting the syntax and semantics of ω_0, our core calculus for MANETs. We then introduce the extensions to ω_0 that result in the more expressive ω_1- and ω_2-calculi.

3.1 Syntax of the ω_0-Calculus

A system description in the ω_0-calculus comprises a set of *nodes*, each of which runs a sequential *process* annotated by its *interface*. We use **N** and **P** to denote the sets of all nodes and all processes, respectively, with M, N ranging over nodes and P, Q ranging over processes. We also use names drawn from two disjoint sets: **Pn** and **Gn**. The names in **Pn**, called *pnames* for *process names*, are used for data values. The names in **Gn**, called *gnames* for *group names*, are used for process interfaces. We use x, y, z to range over **Pn** and g (possibly subscripted) to range over **Gn**. The ω_0-calculus has a two-level syntax describing nodes and processes, respectively.

The syntax of ω_0-calculus processes is defined by the following grammar:

$$\begin{array}{rcl} P & ::= & nil \mid Act.P \mid P + P \mid [x = y]P \mid A(\overrightarrow{x}) \\ Act & ::= & \overline{\mathbf{b}}x \mid \mathbf{r}(x) \mid \tau \end{array}$$

Action $\overline{\mathbf{b}}x$ represents the local broadcast of a value x, while the reception of a locally broadcasted value is denoted by $\mathbf{r}(x)$. Internal (silent) actions are denoted by

τ. Process *nil* is the deadlocked process; $Act.P$ is the process that can perform action Act and then behave as P; and $+$ is the operator for nondeterministic choice. Process $[x = y]P$ (where x and y are pnames) behaves as P if names x and y match, and as *nil* otherwise. $A(\vec{x})$ denotes *process invocation*, where A is a process identifier (having a corresponding definition) and $\vec{x}$ is a comma-separated list of actual parameters (pnames) of the invocation. A process definition is of the form $A(\vec{x}) \overset{\text{def}}{=} P$, and associates a process identifier A and a list of formal parameters $\vec{x}$ (i.e. distinct pnames) with process expression P. Process definitions may be recursive.

The following grammar defines the syntax of ω_0-calculus node expressions:

$$M \quad ::= \quad \mathbf{0} \;\mid\; P\!:\!G \;\mid\; (\nu g)M \;\mid\; M|M$$

$\mathbf{0}$ is the inactive node, while $P\!:\!G$, where $G \subseteq \mathbf{Gn}$, is a node with process P having interface G. The operator (νg) is used to restrict the scopes of gnames. $M|N$ represents the parallel composition of node expressions M and N. Node expressions of the form $P\!:\!G$ are called *basic node expressions*, while those containing the restriction or parallel operator are called *structured node expressions*. Note that gnames occur only at the node level, capturing the intuition that, in an ad hoc network, the behavioral specification of a (basic) node (represented by its process) is independent of its underlying interface.

Free and Bound Names. For a process expression P, the set of free names and bound names of P, denoted as *fn(P)* and *bn(P)*, respectively, are defined as follows:

$$\begin{array}{rcl@{\qquad}rcl}
fn(nil) & = & \emptyset & bn(nil) & = & \emptyset \\
fn(\overline{\mathbf{b}}x.P) & = & fn(P) \cup \{x\} & bn(\overline{\mathbf{b}}x.P) & = & bn(P) \\
fn(\mathbf{r}(x).P) & = & fn(P) \setminus \{x\} & bn(\mathbf{r}(x).P) & = & bn(P) \cup \{x\} \\
fn(\tau.P) & = & fn(P) & bn(\tau.P) & = & bn(P) \\
fn(P+Q) & = & fn(P) \cup fn(Q) & bn(P+Q) & = & bn(P) \cup bn(Q) \\
fn([x=y]P) & = & fn(P) \cup \{x,y\} & bn([x=y]P) & = & bn(P) \\
fn(A(x_1,\ldots,x_n)) & = & \{x_1,\ldots,x_n\} & bn(A(x_1,\ldots,x_n)) & = & \emptyset
\end{array}$$

In a process definition of the form $A(\vec{x}) \overset{\text{def}}{=} P$, $\vec{x}$ are the only names that may occur free in P. The set of all names in a process expression P is given by $n(P)$, where $n(P) = fn(P) \cup bn(P)$. Similarly, the set of all pnames and gnames in a node expression M are denoted by $pn(M)$ and $gn(M)$, and those that occur free are

P1.	$P + Q \equiv Q + P$
P2.	$(P + Q) + R \equiv P + (Q + R)$
P3.	$P \equiv Q,$ if $P \equiv_\alpha Q$
N1.	$M \equiv M \mid \mathbf{0}$
N2.	$M_1 \mid M_2 \equiv M_2 \mid M_1$
N3.	$(M_1 \mid M_2) \mid M_3 \equiv M_1 \mid (M_2 \mid M_3)$
N4.	$(\nu g)M \equiv M,$ if $g \notin fgn(M)$
N5.	$(\nu g)M \mid N \equiv (\nu g)(M \mid N),$ if $g \notin fgn(N)$
N6.	$(\nu g_1)(\nu g_2)M \equiv (\nu g_2)(\nu g_1)M$
N7.	$M \equiv N,$ if $M \equiv_\alpha N$
N8.	$P{:}G \equiv Q{:}G,$ if $P \equiv Q$
N9.	$P{:}G \equiv (\nu g)(P{:}G \cup \{g\}),$ if $g \notin G$

Table 1: Structural congruence relation for the ω_0-calculus.

denoted by $fpn(M)$ and $fgn(M)$, respectively. Gname g is bound in $(\nu g)M$, and all gnames in G are free in $P : G$. The set of all free names in a node expression M is given by $fn(M) = fpn(M) \cup fgn(M)$. An expression without free names is called *closed*. An expression that is not *closed* is said to be *open*. The theory developed in the following sections is applicable to both *open* and *closed* systems (expressions).

3.2 Transitional Semantics of the ω_0-Calculus

The transitional semantics of the ω_0-calculus is defined in terms of a structural congruence relation $\equiv$ (Table 1) and a labeled transition relation $\longrightarrow \subseteq \mathbf{N} \times L \times \mathbf{N}$, where $L = \{\overline{G}x, G(x), \tau, \mu \mid G \subseteq \mathbf{Gn}, x \in \mathbf{Pn}\}$ is a set of transition labels. A labeled transition $(M, \alpha, M') \in \longrightarrow$, is also represented as $M \xrightarrow{\alpha} M'$. As such, only node expressions have transitions. When a node of the form $P{:}G$ broadcasts a value x, it generates a transition labeled by $\overline{G}x$. When $P : G$ receives a broadcast value x, the corresponding transition label is $G(x)$. Actions μ and τ also serve as transition labels, with μ, as explained below, indicating node movement, and τ representing internal (silent) actions.

For transition label α, the sets of bound names and gnames of α are denoted $bn(\alpha)$ and $gn(\alpha)$, respectively, and defined as follows:

Rule Name	Rule	Side Condition
MCAST	$\dfrac{}{(\overline{\mathbf{b}}x.P){:}G \xrightarrow{\overline{G}x} P{:}G}$	$G \neq \emptyset$
RECV	$\dfrac{}{(\mathbf{r}(x).P){:}G \xrightarrow{G(x)} P{:}G}$	$G \neq \emptyset$
CHOICE	$\dfrac{P{:}G \xrightarrow{\alpha} P'{:}G}{(P+Q){:}G \xrightarrow{\alpha} P'{:}G}$	
MATCH	$\dfrac{P{:}G \xrightarrow{\alpha} P'{:}G}{([x{=}x]P){:}G \xrightarrow{\alpha} P'{:}G}$	
DEF	$\dfrac{P\{\overrightarrow{y}/\overrightarrow{x}\}{:}G \xrightarrow{\alpha} P'{:}G}{A(\overrightarrow{y}){:}G \xrightarrow{\alpha} P'{:}G}$	$A(\overrightarrow{x}) \overset{def}{=} P$

Table 2: Transition rules for ω_0-calculus basic node expressions.

$bn(\overline{G}x) = \emptyset$, $bn(G(x)) = \{x\}$, $bn(\mu) = \emptyset$, $bn(\tau) = \emptyset$.

$gn(\overline{G}x) = G$, $gn(G(x)) = G$, $gn(\mu) = \emptyset$, $gn(\tau) = \emptyset$.

We define a label restriction operation $\alpha \setminus G$ that makes visible only those group names in α that are not in set G as follows:

$$\begin{aligned} \tau \setminus G &= \tau \\ \mu \setminus G &= \mu \\ \overline{G_1}x \setminus G_2 &= \overline{G_1 - G_2}\,x \\ G_1(x) \setminus G_2 &= (G_1 - G_2)(x) \end{aligned}$$

where we use $G_1 - G_2$ to denote the set $\{g \mid g \in G_1, g \notin G_2\}$.

We use the standard notion of substitution for names, viz. a mapping $\sigma : \mathbf{Pn} \times \mathbf{Pn}$. We also use the standard notation for application of substitution to terms. The expression $M\{y/x\}$ denotes the node expression in which all free occurrences of x are replaced by y in M, with a change of bound names if necessary to avoid any of the new name y from becoming bound in M.

Process interfaces provide an abstract specification of network topology in terms of node connectivity graphs. Formally, the *node connectivity graph* of a node expression M, denoted by $\chi(M)$, is an undirected graph (V, E) such that V, the set of vertices,

Rule Name	**Rule**	**Side Condition**
STRUCT	$\dfrac{N \equiv M \quad M \xrightarrow{\alpha} M' \quad M' \equiv N'}{N \xrightarrow{\alpha} N'}$	
MOBILITY(I)	$\dfrac{}{M \mid P{:}G \xrightarrow{\mu} M \mid P{:}G'}$	$G' \neq G$, $G' \subseteq G \cup fgn(M)$, $I(M \mid P{:}G) \implies I(M \mid P{:}G')$
PAR(I)	$\dfrac{M \xrightarrow{\alpha} M'}{M \mid N \xrightarrow{\alpha} M' \mid N}$	$bn(\alpha) \cap fn(N) = \emptyset$ $I(M \mid N) \Longrightarrow I(M' \mid N)$
COM	$\dfrac{M \xrightarrow{\overline{G}x} M' \qquad N \xrightarrow{G'(y)} N'}{M \mid N \xrightarrow{\overline{G}x} M' \mid N'\{x/y\}}$	$G \cap G' \neq \emptyset$
GNAME-RES1	$\dfrac{M \xrightarrow{\alpha} M'}{(\nu\, g)M \xrightarrow{\alpha\backslash\{g\}} (\nu\, g)M'}$	$\alpha \in \{\tau, \mu\}$, or $gn(\alpha) \setminus \{g\} \neq \emptyset$
GNAME-RES2	$\dfrac{M \xrightarrow{\overline{G}x} M'}{(\nu\, g)M \xrightarrow{\tau} (\nu\, g)M'}$	$G = \{g\}$

Table 3: Transition rules for ω_0-calculus structured node expressions.

are the basic nodes of M (i.e. subexpressions of M of the form $P : G$) and E, the set of edges, is defined as follows. There is an edge between two vertices $P_1 : G_1$ and $P_2 : G_2$ of $\chi(M)$ only if P_1 and P_2's interfaces overlap; i.e. $G_1 \cap G_2 \not\equiv \emptyset$ (assuming bound names of M are unique and distinct from its free names). The node connectivity graph for the ω_0 node expression of Fig. 2(d) is given in Fig. 2(c).

We use the notion of *connectivity invariant*, to impose different models of node movement on the calculus. A connectivity invariant is a decidable property over undirected graphs. For example, k-connectedness, for a given k, is a candidate connectivity invariant, as is `true`, indicating no constraints on node movement. We write $I(U)$ to indicate that undirected graph U possesses property I. We also use $I(M)$, thus overloading I, to denote $I(\chi(M))$ which means that the connectivity graph of node expression M satisifies connectivity invariant I.

The transitional semantics of the ω_0-calculus is given by the inference rules of Tables 2 and 3, with the former supplying the inference rules for basic node expressions and the latter for structured node expressions. Rules CHOICE, MATCH, and DEF of Table 2 are standard. Rules MCAST and RECV of Table 2, together with COM of Table 3, define a notion of *local broadcast* communication. RECV states that a basic node with process interface G can receive a local broadcast on any gname in G. This, together with COM, means that a local-broadcast sender can synchronize with any local-broadcast receiver with whom it shares a gname (i.e. the receiver is in the transmission range of the sender). Note that a node with an empty in interface cannot perform send or receive actions. Note also that the above definition corresponds to *late* semantics due to the late instantiation of received names.

Local-broadcast synchronization results in a local-broadcast transition label of the form $\overline{G}x$, thereby enabling other receivers to synchronize with the original send action. PAR rule indicates the interleaving semantics for actions of nodes in parallel. The first side condition is standard and is used to avoid name capture. The second side condition permits only those node movements that preserve a connectivity invariant I in a larger network context.

GNAME-RES1 and GNAME-RES2 define the effect of closing the scope of a gname. GNAME-RES1 states that a restricted gname cannot occur in a transition label. GNAME-RES2 states that when all gnames of a local-broadcast-send action are restricted, it becomes a τ-action. MCAST, GNAME-RES1 and GNAME-RES2 together mean that a local-broadcast send is non-blocking; i.e., it can be performed on a set of restricted groups even when there are no corresponding receive actions. In contrast, other actions containing gnames, such as local-broadcast receive, are not covered by GNAME-RES2, and hence have blocking semantics: a system cannot perform actions involving restricted gnames unless there is a corresponding synchronizing action.

In contrast to the broadcast calculi of [EM01, NH06], a node that is capable of receiving a local broadcast is not forced to synchronize with the sender. The semantics of local broadcast in the ω-calculus allows a receiver to ignore a local-broadcast event even if this node is in the transmission range of the broadcasting node. A semantics of this nature captures the lossy transmission inherent in MANETs. The semantics of local broadcast can be modified to force all potential receivers to receive a local broadcast, as done in other broadcast calculi [EM01, NH06]. This would require the addition of a side-condition to the PAR rule, allowing autonomous broadcast/receive actions only when the context (node expression N in the PAR rule) is incapable of synchronizing with that action.

The notion of structural congruence (Table 1) considered in rule STRUCT is defined for processes (rules P1-P3) in the standard way—P and Q are structurally congruent if they are alpha-equivalent or congruent under the associativity and commutativity of the choice ('+') operator—and then lifted to nodes (rules N1-N9). Two basic node expressions are structurally congruent if they have identical process interfaces and run structurally congruent processes (rule N8). Rules N4-N6 are for restriction on gnames. Rule N9 allows basic nodes to create and acquire a new group name or drop a local group name. Structural congruence of nodes includes alpha-equivalence (rule N7) and the associativity and commutativity of the parallel ('|') operator (rules N2 and N3).

Semantics of mobility. The semantics of node movement is defined by the MOBILITY rule, which states that the process interface of node $P\!:\!G$ can change from G to G' whenever the node is in parallel with another node M. In particular, the side condition $G' \subseteq G \cup \mathit{fgn}(M)$ stipulates that P may drop gnames from its interface or acquire free gnames from M.

The MOBILITY rule reflects the fact that P's interface may change when node $P\!:\!G$, or the nodes around it, are in motion. A change in P's interface may further result in a corresponding change in the overall network topology. Note that the rule does not specify which nodes moved, only that the topology has been updated as the result of movement of one or more nodes. The third side condition to the MOBILITY rule, decrees that whenever $M \xrightarrow{\mu} M'$ is derived using the MOBILITY rule, the resulting transition must preserve a connectivity invariant.

We thus have that the MOBILITY and PAR rules in particular, and the calculus's semantics in general, are parameterized by the connectivity invariant, thus taking into account the constraints on node movement.

An example derivation of node movement is shown in Fig. 4. This derivation was obtained using the structural congruence and transition rules defining the semantics

$$\Box$$

MOBILITY

$$(P_1\!:\!\{g_1,g_2\} \mid P_2\!:\!\{g_1\} \mid P_4\!:\!\{g_1,g_3\}) \mid P_3\!:\!\{g_2\} \xrightarrow{\mu} (P_1\!:\!\{g_1,g_2\} \mid P_2\!:\!\{g_1\} \mid P_4\!:\!\{g_1,g_3\}) \mid P_3\!:\!\{g_3\}$$

STRUCT

$$P_1\!:\!\{g_1,g_2\} \mid P_2\!:\!\{g_1\} \mid P_3\!:\!\{g_2\} \mid P_4\!:\!\{g_1,g_3\} \xrightarrow{\mu} P_1\!:\!\{g_1,g_2\} \mid P_2\!:\!\{g_1\} \mid P_3\!:\!\{g_3\} \mid P_4\!:\!\{g_1,g_3\}$$

GNAME-RES1 (thrice)

$$(\nu g_1)(\nu g_2)(\nu g_3)(P_1\!:\!\{g_1,g_2\} \mid P_2\!:\!\{g_1\} \mid P_3\!:\!\{g_2\} \mid P_4\!:\!\{g_1,g_3\}) \xrightarrow{\mu} (\nu g_1)(\nu g_2)(\nu g_3)(P_1\!:\!\{g_1,g_2\} \mid P_2\!:\!\{g_1\} \mid P_3\!:\!\{g_3\} \mid P_4\!:\!\{g_1,g_3\})$$

STRUCT

$$(\nu g_1)(\nu g_2)(P_1\!:\!\{g_1,g_2\} \mid P_2\!:\!\{g_1\} \mid P_3\!:\!\{g_2\} \mid (\nu g_3)(P_4\!:\!\{g_1,g_3\})) \xrightarrow{\mu} (\nu g_1)(\nu g_2)(P_1\!:\!\{g_1,g_2\} \mid P_2\!:\!\{g_1\} \mid (\nu g_3)(P_3\!:\!\{g_3\} \mid P_4\!:\!\{g_1,g_3\}))$$

STRUCT

$$(\nu g_1)(\nu g_2)(P_1\!:\!\{g_1,g_2\} \mid P_2\!:\!\{g_1\} \mid P_3\!:\!\{g_2\} \mid P_4\!:\!\{g_1\}) \xrightarrow{\mu} (\nu g_1)(\nu g_2)(P_1\!:\!\{g_1,g_2\} \mid P_2\!:\!\{g_1\} \mid (\nu g_3)(P_3\!:\!\{g_3\} \mid P_4\!:\!\{g_1,g_3\}))$$

Figure 4: Derivation for movement of N_3 from its position in Fig. 2 to that in Fig. 3.

of the ω-calculus, and "connectedness" as the connectivity invariant.

3.3 The ω_1-Calculus

The ω_1- and ω_2-calculi are defined in a modular fashion by adding new syntactic constructs, and associated inference rules for their semantics, to the ω_0-calculus. In this section, we consider the extension ω_1.

Extending ω_0 to ω_1. Syntactically, we obtain ω_1 from ω_0 as follows:

- We add restriction operators for *pnames* for both process-level and node-level expressions. We use the standard notation of $(\nu x)P$ for a pname x restricted to a process expression P, and $(\nu x)N$ for a pname x restricted to a node expression N. As usual, x is bound in $(\nu x)P$ and $(\nu x)N$.

- We introduce unicast communication as a prefix operator for process expressions. Although unicast in principle can be implemented on top of broadcast, we prefer to give it first-class status, as it is a frequent action in MANET protocols. Doing so also facilitates concise modeling and deterministic reasoning (only the intended recipient can receive a unicast message). We use the standard notation of $\overline{x}y$ to denote the sending of name y along x, and $x(y)$ to denote the reception of a name along x that will bind to y. As usual, x and y are free in the expression $\overline{x}y.P$, and x is free and y is bound in $x(y).P$.

Unlike in ω_0 where *pnames* are used strictly as data values, in ω_1, *pnames* (the set **Pn**) can be used as communicable data as well as communication (unicast) channels.

Semantically, the introduction of scoped pnames needs new inference rules to handle scope extrusion. We add OPEN and CLOSE rules (as in the π-calculus [MPW92]) and, in addition to the broadcast communication rule (COM) of ω_0, a rule for communication of bound names. We also add RES rules at the process and node levels to disallow communication over a restricted name. These additional rules follow closely the standard rules for handling scopes and scope extrusion in the π-calculus; details are omitted. New structural congruence rules are added to take the restriction of pnames into account. For instance, restriction of pnames and gnames commute (i.e. $(\nu x)(\nu g)N \equiv (\nu g)(\nu x)N$), and the restriction operator can be pushed into or pulled out of node and process expressions as long as free names are not captured. At first glance, it may appear that the structural congruence rules for scope extension of pnames are redundant in the presence of the scope-extrusion rules (OPEN/CLOSE). However, the OPEN/CLOSE rules are essential for reasoning about open systems, and the scope extension rules are essential for defining normal forms (see Definition 3).

The addition of unicast communication raises certain interesting issues with respect to mobility. Recall that *groups* encapsulate the locality of a process. When two processes share a private name, they can use that name as a channel of communication. However, after establishing that link, if the processes move away from each other, they may no longer be able to use that name as a channel. In summary, unicast channels should also respect the locality of communication. We enforce this in the ω_1-calculus by annotating unicast action labels with the interfaces of the participating processes, and allowing synchronization between actions only when their interfaces

Rule Name	Rule	Side Condition
UNI-SEND	$\dfrac{}{(\overline{z}x.P){:}G \xrightarrow{\overline{z{:}G}x} P{:}G}$	$G \neq \emptyset$
UNI-RECV	$\dfrac{}{(z(x).P){:}G \xrightarrow{z{:}G(x)} P{:}G}$	$G \neq \emptyset$
UNI-COM	$\dfrac{M \xrightarrow{\overline{z{:}G}x} M' \quad N \xrightarrow{z{:}G'(y)} N'}{M \mid N \xrightarrow{\tau} M' \mid N'\{x/y\}}$	$G \cap G' \neq \emptyset$

Table 4: Transition rules for unicast communication in ω_1-calculus.

overlap (meaning that the processes are in each other's transmission range). Hence, the execution of a unicast send action of value x on channel z by a basic node with process interface G is represented by action label $\overline{z{:}G}x$; the corresponding receive action is labeled $z{:}G(x)$.

The semantic rules for unicast send (UNI-SEND), receive (UNI-RECV), and synchronization (UNI-COM) are given in Table 4. Scope extrusion via unicast communication is accomplished by naturally extending their π-calculus counterparts (OPEN/CLOSE) rules as follows. Bound-output actions (due to OPEN) are annotated with the interface of the participating process, and the CLOSE rule applies only when the interfaces overlap. These extensions are straightforward, and the details are omitted.

The set of bound names and gnames for the transition labels introduced by the ω_1-calculus are given below:

$$bn(\overline{z : G}x) = \emptyset,\;\; bn(z : G(x)) = \{x\},\;\; bn((\nu x)\overline{z : G}x) = \{x\},\;\; bn((\nu x)\overline{G}x) = \{x\}.$$
$$gn(\overline{z : G}x) = G,\;\; gn(z : G(x)) = G,\;\; gn((\nu x)\overline{z : G}x) = G,\;\; gn((\nu x)\overline{G}x) = G.$$

Note that the scope of a name may encompass different processes regardless of their interfaces, and hence two processes may share a secret even when they are outside each others transmission ranges. The restriction we impose is that shared names can be used as unicast channels only when the processes are within each others transmission ranges.

P4.	$(\nu x)P \equiv P$, if $x \notin fn(P)$
P5.	$(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$
P6.	$P \mid Q \equiv Q \mid P$
P7.	$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$
P8.	$(\nu x)P_1 \mid P_2 \equiv (\nu x)(P_1 \mid P_2)$ if $x \notin fn(P_2)$
N10.	$(\nu x)M \equiv M$, if $x \notin fpn(M)$
N11.	$(\nu x)M_1 \mid M_2 \equiv (\nu x)(M_1 \mid M_2)$, if $x \notin fpn(M_2)$
N12.	$(\nu x)(\nu y)M \equiv (\nu y)(\nu x)M$
N13.	$(\nu g)(\nu x)M \equiv (\nu x)(\nu g)M$
N14.	$((\nu x)P){:}G \equiv (\nu x)(P{:}G)$

Table 5: Additional structural congruence rules for the ω-node expressions.

3.4 The full ω-calculus: ω_2-calculus.

We obtain the ω_2-calculus by adding the parallel composition ('|') operator at the process level, thereby allowing concurrent processes within a node. This addition facilitates e.g. the modeling of communication between layers of a protocol stack running at a single node; it also renders the π-calculus a subcalculus of the ω_2-calculus. In ω_2, the actions of two processes within a node may be interleaved. Moreover, two processes within a node can synchronize using unicast (binary) communication. We add PAR, COM and CLOSE rules corresponding to intra-node interleaving, synchronization and scope extrusion, respectively; these rules are straightforward extensions of the corresponding rules in the π-calculus.

The syntax of processes in the ω-calculus is defined by the following grammar:

$$\begin{array}{rcl} P & ::= & nil \mid Act.P \mid P+P \mid (\nu x)P \mid [x=y]P \mid P|P \mid A(\vec{x}) \\ Act & ::= & \overline{x}y \mid x(y) \mid \overline{\mathbf{b}}x \mid \mathbf{r}(x) \mid \tau \end{array}$$

The following grammar defines the syntax of node expressions in the ω-calculus:

$$M \quad ::= \quad \mathbf{0} \mid P{:}G \mid (\nu g)M \mid (\nu x)M \mid M|M$$

The structural congruence rules for the ω-calculus are given in Tables 1 and 5, and the transitional semantics rules are given in Tables 2, 3, 4, 6, and 7.

Rule Name	Rule	Side Condition
PROC-PAR	$\dfrac{P{:}G \xrightarrow{\alpha} P'{:}G}{(P \mid Q){:}G \xrightarrow{\alpha} (P' \mid Q){:}G}$	$bn(\alpha) \cap fn(Q) = \emptyset$
PROC-COM	$\dfrac{P{:}G \xrightarrow{\overline{z{:}G}x} P'{:}G \qquad Q{:}G \xrightarrow{z{:}G(y)} Q'{:}G}{(P \mid Q){:}G \xrightarrow{\tau} (P' \mid Q'\{x/y\}){:}G}$	
PROC-CLOSE	$\dfrac{P{:}G \xrightarrow{(\nu x)\overline{z{:}G}x} P'{:}G \qquad Q{:}G \xrightarrow{z{:}G(x)} Q'{:}G}{(P \mid Q){:}G \xrightarrow{\tau} ((\nu x)(P' \mid Q')){:}G}$	

Table 6: Additional transitional semantics rules for basic ω-node expressions.

Rule Name	Rule	Side Condition
UNI-OPEN	$\dfrac{M \xrightarrow{\overline{z{:}G}x} M'}{(\nu x)M \xrightarrow{(\nu x)\overline{z{:}G}x} M'}$	$x \neq z$
UNI-CLOSE	$\dfrac{M \xrightarrow{(\nu x)\overline{z{:}G}x} M' \qquad N \xrightarrow{z{:}G'(x)} N'}{M \mid N \xrightarrow{\tau} (\nu x)(M' \mid N')}$	$G \cap G' \neq \emptyset$
OPEN	$\dfrac{M \xrightarrow{\overline{G}x} M'}{(\nu x)M \xrightarrow{(\nu x)\overline{G}x} M'}$	
COM-RES	$\dfrac{M \xrightarrow{(\nu x)\overline{G}x} M' \qquad N \xrightarrow{G'(x)} N'}{M \mid N \xrightarrow{(\nu x)\overline{G}x} M' \mid N'}$	$G \cap G' \neq \emptyset$
CLOSE	$\dfrac{M \xrightarrow{(\nu x)\overline{G}x} M'}{(\nu\, g)M \xrightarrow{\tau} (\nu\, g)(\nu x)M'}$	$G = \{g\}$
PNAME-RES	$\dfrac{M \xrightarrow{\alpha} M'}{(\nu x)M \xrightarrow{\alpha} (\nu x)M'}$	$x \notin n(\alpha)$

Table 7: Additional transition semantics rules for structured ω-node expressions.

3.5 Symbolic Semantics for the ω-Calculus

We now define a symbolic transitional semantics for the ω-calculus. The symbolic semantics binds names lazily and enables more efficient construction of transition systems for verification. In particular, transition system for a node expression is given using the traditional semantics (Section 3) assuming that the free names in the expression are all distinct. Consequently, transition system for an expression needs to be generated for each context in which the expression is used. In contrast, the symbolic semantics can be used to generate a symbolic transition system for each expression, and the transition system can then be applied to each context of the expression. The symbolic semantics also permits us to introduce useful operators, such as a π-calculus-like *mismatch* operator to the ω-calculus, and provide concise semantics to such operators.

Rule Name	**Rule**	**Side Condition**
MCAST	$\dfrac{}{(\overline{\mathbf{b}}x.P){:}G \xrightarrow{\mathbf{true},\overline{G}x} P{:}G}$	$G \neq \emptyset$
RECV	$\dfrac{}{(\mathbf{r}(x).P){:}G \xrightarrow{\mathbf{true},G(x)} P{:}G}$	$G \neq \emptyset$
CHOICE	$\dfrac{P{:}G \xrightarrow{\lambda} P'{:}G}{(P+Q){:}G \xrightarrow{\lambda} P'{:}G}$	
MATCH	$\dfrac{P{:}G \xrightarrow{C_1,\alpha} P'{:}G}{([x{=}y]P){:}G \xrightarrow{C_1\wedge[x=y],\alpha} P'{:}G}$	$x, y \notin bn(\alpha)$
MISMATCH	$\dfrac{P{:}G \xrightarrow{C_1,\alpha} P'{:}G}{([x{\neq}y]P){:}G \xrightarrow{C_1\wedge[x\neq y],\alpha} P'{:}G}$	$x, y \notin bn(\alpha)$
DEF	$\dfrac{P\{\overrightarrow{y}/\overrightarrow{x}\}{:}G \xrightarrow{\lambda} P'{:}G}{A(\overrightarrow{y}){:}G \xrightarrow{\lambda} P'{:}G}$	$A(\overrightarrow{x}) \overset{def}{=} P$

Table 8: Symbolic semantics for basic ω-node expressions.

Rule Name	Rule	Side Condition
UNI-SEND	$\dfrac{}{(\overline{z}x.P){:}G \xrightarrow{\mathbf{true},\overline{z{:}G}x} P{:}G}$	$G \neq \emptyset$
UNI-RECV	$\dfrac{}{(z(x).P){:}G \xrightarrow{\mathbf{true},z{:}G(x)} P{:}G}$	$G \neq \emptyset$
PROC-PAR	$\dfrac{P{:}G \xrightarrow{C,\alpha} P'{:}G}{(P \mid Q){:}G \xrightarrow{C,\alpha} (P' \mid Q){:}G}$	$bn(\alpha) \cap fn(Q) = \emptyset$
PROC-COM	$\dfrac{P{:}G \xrightarrow{C_1,\overline{w{:}G}x} P'{:}G \qquad Q{:}G \xrightarrow{C_2,z{:}G(y)} Q'{:}G}{(P \mid Q){:}G \xrightarrow{C_1 \wedge C_2 \wedge [w=z],\tau} (P' \mid Q'\{x/y\}){:}G}$	
PROC-CLOSE	$\dfrac{P{:}G \xrightarrow{C_1,(\nu x)\overline{w{:}G}x} P'{:}G \qquad Q{:}G \xrightarrow{C_2,z{:}G(x)} Q'{:}G}{(P \mid Q){:}G \xrightarrow{C_1 \wedge C_2 \wedge [w=z],\tau} ((\nu x)(P' \mid Q')){:}G}$	

Table 9: Symbolic semantics for basic ω-node expressions.

We define a symbolic semantics for the ω-calculus in a manner similar to that for the π-calculus [Par01]. The symbolic semantics is given in terms of a symbolic labeled transition relation. Each symbolic transition has an associated constraint representing the conditions under which that transition is enabled. More specifically, symbolic transitions are of the form $M \xrightarrow{C,\alpha} M'$, where C is a constraint on the free pnames of M. Constraints are conjunctions of zero or more atomic constraints, which are of the form **true**, **false**, and for pnames x and y, $x = y$ and $x \neq y$. An empty constraint is equivalent to **true** as are constraints of the form $x = x$. Constraints of the form $x \neq x$ are equivalent to **false**. A conjunction containing **false** is equivalent to **false**. In the following, we assume that constraints are maintained, using these equivalences, in simplified form.

The inference rules for the symbolic semantics of the ω-calculus are given in Tables 8-11. In the tables, we use λ to represent transition labels, i.e., pairs (C, α). The rules also use a constraint expression of the form $C - x$, which represents the constraint obtained from C by replacing all occurrences of $x = y$ by **false** and $x \neq y$

Rule Name	Rule	Side Condition
STRUCT	$\dfrac{N \equiv M \quad M \xrightarrow{\lambda} M' \quad M' \equiv N'}{N \xrightarrow{\lambda} N'}$	
MOBILITY(I)	$\dfrac{}{M \mid P{:}G \xrightarrow{\mathbf{true},\mu} M \mid P{:}G'}$	$G' \neq G$, $G' \subseteq G \cup fgn(M)$, $I(M \mid P{:}G) \implies I(M \mid P{:}G')$
PAR(I)	$\dfrac{M \xrightarrow{C,\alpha} M'}{M \mid N \xrightarrow{C,\alpha} M' \mid N}$	$bn(\alpha) \cap fn(N) = \emptyset$ $I(M \mid N) \Longrightarrow I(M' \mid N)$
COM	$\dfrac{M \xrightarrow{C_1,\overline{G}x} M' \quad N \xrightarrow{C_2,G'(y)} N'}{M \mid N \xrightarrow{C_1 \wedge C_2,\overline{G}x} M' \mid N'\{x/y\}}$	$G \cap G' \neq \emptyset$
GNAME-RES1	$\dfrac{M \xrightarrow{C,\alpha} M'}{(\nu\, g)M \xrightarrow{C,\alpha\setminus\{g\}} (\nu\, g)M'}$	$\alpha \in \{\tau, \mu\}$, or $gn(\alpha) \setminus \{g\} \neq \emptyset$
GNAME-RES2	$\dfrac{M \xrightarrow{C,\overline{G}x} M'}{(\nu\, g)M \xrightarrow{C,\tau} (\nu\, g)M'}$	$G = \{g\}$

Table 10: Symbolic semantics for structured ω-node expressions.

by **true**. We do not need to consider cases such as $x = x$ since we assume that C is in simplified form.

Rule MATCH and the rules corresponding to binary (unicast) synchronization (PROC-COM, UNI-COM, and UNI-CLOSE) generate equality constraints over pnames. In the non-symbolic case, we say that two nodes performing unicast send and receive operations can synchronize only if they use identical channel names. In contrast, in the symbolic case, we permit any two such operations to synchronize and generate a condition that the two channels should be the same (represented by the equality constraint between the two names). Inequality constraints are introduced by the MISMATCH rule.

Consider the PNAME-RES rule, which says that $(\nu x)M \xrightarrow{C',\alpha} (\nu x)M'$ can be

Rule Name	Rule	Side Condition
UNI-COM	$\dfrac{M \xrightarrow{C_1,\overline{w{:}G}x} M' \quad N \xrightarrow{C_2,z{:}G'(y)} N'}{M \mid N \xrightarrow{C_1 \wedge C_2 \wedge [w=z],\tau} M' \mid N'\{x/y\}}$	$G \cap G' \neq \emptyset$
UNI-OPEN	$\dfrac{M \xrightarrow{C,\overline{z{:}G}x} M'}{(\nu x)M \xrightarrow{C-x,(\nu x)\overline{z{:}G}x} M'}$	$x \neq z$
UNI-CLOSE	$\dfrac{M \xrightarrow{C_1,(\nu x)\overline{w{:}G}x} M' \quad N \xrightarrow{C_2,z{:}G'(x)} N'}{M \mid N \xrightarrow{C_1 \wedge C_2 \wedge [w=z],\tau} (\nu x)(M' \mid N')}$	$G \cap G' \neq \emptyset$
OPEN	$\dfrac{M \xrightarrow{C,\overline{G}x} M'}{(\nu x)M \xrightarrow{C-x,(\nu x)\overline{G}x} M'}$	
COM-RES	$\dfrac{M \xrightarrow{C_1,(\nu x)\overline{G}x} M' \quad N \xrightarrow{C_2,G'(x)} N'}{M \mid N \xrightarrow{C_1 \wedge C_2,(\nu x)\overline{G}x} M' \mid N'}$	$G \cap G' \neq \emptyset$
CLOSE	$\dfrac{M \xrightarrow{C,(\nu x)\overline{G}x} M'}{(\nu\, g)M \xrightarrow{C,\tau} (\nu\, g)(\nu x)M'}$	$G = \{g\}$
PNAME-RES	$\dfrac{M \xrightarrow{C,\alpha} M'}{(\nu x)M \xrightarrow{C-x,\alpha} (\nu x)M'}$	$x \notin n(\alpha)$

Table 11: Symbolic semantics for structured ω-node expressions.

inferred from $M \xrightarrow{C,\alpha} M'$ if x is not a name in α. Note that while C is a constraint on the free pnames of M, C' is a constraint on the free pnames of $(\nu x)M$; i.e., C' does not contain x. Consider an equality constraint of the form $x = y$ in C. This constraint will be unsatisfiable in the context of (νx) since x is a restricted pname. Now consider a disequality constraint of the form $x \neq y$ in C. This constraint will be always satisfiable in the context of (νx) since x is a restricted name and y is a free name. Hence we obtain C' as $C - x$: derived from C by replacing all occurrences of $x = y$ by **false** and $x \neq y$ by **true**.

The equality constraints in a conjunction of constraints induce an equivalence relation on the names appearing in the constraints. For a given constraint C, we use σ_C to denote a substitution that maps all names in the same equivalence class to a representative name (chosen arbitrarily) of the class. We use $C_1 \triangleright C_2$ to indicate that C_1 implies C_2.

We can establish a correspondence between the symbolic semantics and the transitional semantics presented in Section 3, formalized as follows.

Theorem 1 (Correspondence) *For all M in the mismatch-free fragment of the ω-calculus: $M \xrightarrow{C,\alpha} M'$ iff $M\sigma_C \xrightarrow{\alpha\sigma_C} M'\sigma_C$.*

This theorem can be proved by induction on the derivation length.

Chapter 4

Properties of the ω-Calculus

In this chapter, we prove some fundamental properties of the ω-calculus, including congruence results for strong bisimulation equivalence and a weak version of bisimulation equivalence that treats τ- and μ-actions as unobservable.

Embedding of the π-Calculus. The ω-calculus is a conservative extension of the π-calculus [MPW92]. That is, every process expression P in the π-calculus can be translated to an ω-node expression $P : G$, for $G \subseteq \mathbf{Gn}$ and $G \neq \emptyset$, such that the transition system generated by $P : G$ is isomorphic to the one generated by P. We impose the condition $G \neq \emptyset$ since a basic node with an empty interface ($P : \{\}$) cannot perform any action. This property is formally stated by the following theorem, which is readily proved by induction on the length of derivations.

Theorem 2 *For any process expression P in the π-calculus, $P : G$ is a node expression in the ω-calculus, where $G \subseteq \boldsymbol{Gn}$ and $G \neq \emptyset$. Moreover, $P \xrightarrow{\alpha} P'$ is a transition derivable from the operational semantics of the π-calculus if and only if $P : G \xrightarrow{\alpha'} P' : G$ is derivable from the operational semantics of the ω-calculus, and one of the following conditions hold: (i) $\alpha = \alpha' = \tau$; (ii) $\alpha = x(y)$ and $\alpha' = x : G(y)$; (iii) $\alpha = \overline{x}y$ and $\alpha' = \overline{x : G}y$; or (iv) $\alpha = (\nu y)\overline{x}y$ and $\alpha' = (\nu y)\overline{x : G}y$, for some names x, y.*

Decidability of the Finite-Control Fragment. The *finite-control* fragment of the ω-calculus, as in the case of the π-calculus, is the subcalculus where recursive definitions do not contain the parallel operator (‘$|$’) and every occurrence of process identifiers is guarded. Reachability properties are decidable for closed process expressions (i.e. those without free names) specified in the finite-control fragment [Dam97]. We extend the notion of finite control to the ω-calculus, and show that reachability remains decidable for closed node expressions. This result, formally stated in Theorem 3, is of practical importance in verifying MANET system specifications. Formally, we say that an ω-calculus expression N is *reachable* from M (denoted by $M \longrightarrow^* N$) if there is a finite sequence of transitions $M \xrightarrow{\alpha_1} M_1 \xrightarrow{\alpha_2} M_2 \cdots \xrightarrow{\alpha_n} N$.

Theorem 3 *Let M be a closed finite-control ω-calculus expression and let $R_M = \{N \mid M \longrightarrow^* N\}_{\equiv}$ be the set of node expressions reachable from M modulo the structural congruence relation $\equiv$. Then, R_M is finite.*

The following proof uses the finite reachability result for the finite-control π-calculus given in [Dam97].

Proof Sketch: Consider the fragment ω_π of the ω-calculus without broadcast actions and the MOBILITY rule. For a node expression M in ω_π, the corresponding π-calculus process expression, denoted by M_π, is obtained from M by deleting process interfaces and gname restrictions. Let M be a ω_π-expression such that all process expressions have the same process interface. Then M’s transition system is isomorphic to that of M_π.

Now further assume that M is closed and finite-control. Then the set of expressions reachable from M, R_M, is similar (except for occurrences of process interfaces and gnames) to that for M_π. Since only finitely many expressions are reachable from M_π, R_M is also finite.

Next, extend ω_π to $\omega_{b\pi}$ by including broadcast actions. Let M_1 be such a $\omega_{b\pi}$-expression that is both closed and finite-control. The process contexts due to broadcast action prefixes are analyzed in a similar manner as the binary-synchronization action prefixes. Using an argument similar to the one used above for ω_π, it can be concluded that R_{M_1} is finite.

Finally, we include the MOBILITY rule in $\omega_{b\pi}$, extending $\omega_{b\pi}$ to the ω-calculus. Let M_2 be a closed finite-control ω-expression. The MOBILITY rule affects only the gnames (including process-interfaces) appearing in expressions reachable from M_2. It can be observed that the set R_{M_2} is a variant of the set R_{M_1}, with different combinations of process-interfaces (permitted by the MOBILITY rule) attached to the process expressions appearing in the elements of R_{M_1}. The different combinations of process interfaces possible for n basic nodes in an ω-expression (modulo $\equiv$) is finite and bounded by the number of topologies that a network of n nodes can assume. This implies that R_{M_2} is finite.

Hence, reachability for the finite-control fragment of the ω-calculus is decidable.

$\square$

Bisimulation for the ω-Calculus. The definition of strong (late) bisimulation for the π-calculus [MPW92] can be extended to the ω-calculus.

Definition 1 *A relation $S \subseteq \mathbf{N} \times \mathbf{N}$ is a* strong simulation *if $M\,S\,N$ implies:*

- *fgn(M) = fgn(N), and*
- *whenever $M \xrightarrow{\alpha} M'$ where $bn(\alpha)$ is fresh then:*
  - *if $\alpha \in \{G(x),\, z\!:\!G(x)\}$, there exists an N' s.t. $N \xrightarrow{\alpha} N'$ and for all $y \in$ **Pn**, $M'\{y/x\}\,S\,N'\{y/x\}$,*
  - *if $\alpha \notin \{G(x),\, z\!:\!G(x)\}$, there exists an N' s.t. $N \xrightarrow{\alpha} N'$ and $M'\,S\,N'$.*

S is a strong bisimulation *if both S and S^{-1} are strong simulations. Nodes M and N are* strong bisimilar*, written $M \sim N$, if $M\,S\,N$ for some* strong bisimulation *S.*

Proposition 4 *(i) $\sim$ is an equivalence; and (ii) $\sim$ is the largest strong bisimulation.*

Strong bisimulation equivalence is a congruence for the ω-calculus, as formally stated in Theorem 10. We use the *bisimulation up to* $\equiv$ technique [San98] to establish this result. The following definitions and lemmas are also needed.
Notation: For a given relation R, the relation $\equiv\!R\!\equiv$ is given by: $\{(x,y)\,|\,(x',y') \in R, x \equiv x', y \equiv y'\}$.

Definition 2 *A symmetric relation $S \subseteq \mathbf{N} \times \mathbf{N}$ is a* strong bisimulation up to $\equiv$ *if $M\,S\,N$ implies*

- *fgn(M) = fgn(N), and*

- *whenever* $M \xrightarrow{\alpha} M'$ *where* $bn(\alpha)$ *is fresh then:*
  - *if* $\alpha \in \{G(x),\, z\!:\!G(x)\}$*, there exists an* N' *s.t.* $N \xrightarrow{\alpha} N'$ *and for all* $y \in \boldsymbol{Pn}$, $M'\{y/x\} \equiv S \equiv N'\{y/x\}$,
  - *if* $\alpha \notin \{G(x),\, z\!:\!G(x)\}$*, there exists an* N' *s.t.* $N \xrightarrow{\alpha} N'$ *and* $M' \equiv S \equiv N'$.

Lemma 5 *If* S *is a strong bisimulation up to* $\equiv$*, then for any* $M, N \in \mathbf{N}$*,* $M\,S\,N$ *implies* $M \sim N$*.*

Proof: For any $M, N \in \mathbf{N}$, $M\,S\,N$ implies $M \equiv S \equiv N$. It is sufficient to show that $\equiv S \equiv$ is a strong bisimulation because then $M \equiv S \equiv N$ would imply that M and N are strong bisimilar. $M \equiv S \equiv N$ implies there exist some M_1 and N_1 s.t. $M \equiv M_1$, $N \equiv N_1$, and $M_1\,S\,N_1$. From STRUCT rule, $M \xrightarrow{\alpha} M'$ implies that there exists M_1' s.t. $M_1 \xrightarrow{\alpha} M_1'$ and $M' \equiv M_1'$.

For the case $\alpha \notin \{G(x), z\!:\!G(x)\}$, using Def. 2 it can be inferred that $M_1\,S\,N_1$ and $M_1 \xrightarrow{\alpha} M_1'$ imply that there exists N_1' s.t. $N_1 \xrightarrow{\alpha} N_1'$ and $M_1' \equiv S \equiv N_1'$. $M\,S\,N$ and $M \xrightarrow{\alpha} M'$ imply that there exists N' s.t. $N \xrightarrow{\alpha} N'$, and $N_1' \equiv N'$ because $N \equiv N_1$. $M_1' \equiv S \equiv N_1'$ holds since $M_1\,S\,N_1$ and S is a strong bisimulation up to $\equiv$. By transitivity of $\equiv$, $M' \equiv S \equiv N'$.

Similarly, it can be shown that for $\alpha \in \{G(x), z\!:\!G(x)\}$, and for each $y \in \mathbf{Pn}$, $M'\{y/x\} \equiv S \equiv N'\{y/x\}$.

From Def. 2 and $M\,S\,N$, it holds that $fgn(M) = fgn(N)$.

Hence, $\equiv S \equiv$ is a strong bisimulation. Therefore, $M \equiv S \equiv N$ implies $M \sim N$.

□

An intermediate step in establishing that strong bisimulation equivalence is a congruence for the ω-calculus is to prove it for ω-expressions in a normal form, defined below. We use the term "guarded restrictions" in the context of ω-expressions to mean restrictions that are preceded by an action prefix.

Definition 3 (Normal Form) *An* ω*-expression is in* normal form *if all bound names are distinct and all unguarded restrictions are at the top level with all gnames preceding pnames.*

We use $\mathbf{N_{nf}}$ to denote the set of ω-node expressions in normal form. The structural congruence rules are extended by the following rules (as in [Par01]).

$$\begin{array}{lr}
(\nu x)\mathbf{0} \equiv \mathbf{0} & \\
(\nu x)P + Q \equiv (\nu x)(P + Q) & \textit{if}\, x \notin \mathit{fn}(Q) \\
[y = z](\nu x)P \equiv (\nu x)[y = z]P & \textit{if}\, x \neq y \text{ and } x \neq z \\
A(\overrightarrow{y}) \equiv P\{\overrightarrow{y}/\overrightarrow{x}\} & A(\overrightarrow{x}) \overset{def}{=} P
\end{array}$$

Proposition 6 *Every ω-expression is structurally congruent to an ω-expression in normal form.*

Every ω-expression can be converted into a structurally congruent ω-expression in normal form by renaming all bound names so that they are distinct and using structural congruence rules to pull out all unguarded restrictions to the outermost level. The following lemma originally appeared in [Par01] and is lifted here to the ω-calculus.

Lemma 7 *If $M \xrightarrow{\alpha} M'$ and $M \equiv N$ where $N \in \mathbf{N_{nf}}$, then there exists $N' \in \mathbf{N}$ such that by inference of no greater depth, $N \xrightarrow{\alpha} N'$ and $M' \equiv N'$.*

Proof Idea: The proof is by induction on the inference of $M \equiv N$ and involves examination of all the structural congruence rules. □

Lemma 8 *For every $M, M' \in \mathbf{N}$, if $M \xrightarrow{\alpha} M'$, then there exists an $N \in \mathbf{N_{nf}}$ such that $N \equiv M$ and*

(i) N is of the form $(\nu\tilde{g})(\nu\tilde{x})N'$ where $\tilde{g}$ and $\tilde{x}$ are nonempty sets, and

(ii) there exists $M'' \in \mathbf{N}$ such that $N \xrightarrow{\alpha} M''$, $M'' \equiv M'$, and $N \xrightarrow{\alpha} M''$ can be derived without using STRUCT rules in the last two steps of the derivation.

Proof Sketch: Clearly, we can always find an N in normal form obeying condition (i) that is equivalent to any given $M \in \mathbf{N}$. Since N has an outermost (non-empty) gname restriction and a non-empty pname restriction at the next level, any derivation for a transition from N will involve at least two steps.

Consider the shortest derivation for $N \xrightarrow{\alpha} M''$ (shortest among all N equivalent to M). For such a derivation, the last step cannot be an application of STRUCT rule. To the contrary, assume that the last step in the derivation is an application of the STRUCT rule. Then the last step is of the form:

$$\frac{N \equiv M_1 \quad M_1 \xrightarrow{\alpha} M_2 \quad M_2 \equiv M}{N \xrightarrow{\alpha} M}$$

M_1 cannot be in normal form; otherwise there is a shorter derivation. However, by Lemma 7, there is a normal form equivalent to M_1 that has at least as short a derivation. Thus, there is a shorter derivation for $N'' \xrightarrow{\alpha} M''$ for some normal form $N'' \equiv M$, which is a contradiction. Hence the last step in the shortest derivation cannot be an application of the STRUCT rule.

This means that the last step in the shortest derivation must be due to the outermost gname restriction. We can similarly argue that in the shortest derivation, the next-to-last step is not an application of STRUCT rule. □

Lemma 9 *For all $M_1, M_2 \in \mathbf{N_{nf}}$, i.e., M_1, M_2 are in normal form, the following hold:*

(i) $M_1 \sim M_2$ implies $\forall x \in \mathbf{Pn} : (\nu x)M_1 \sim (\nu x)M_2$;

(ii) $M_1 \sim M_2$ implies $\forall g \in \mathbf{Gn} : (\nu g)M_1 \sim (\nu g)M_2$; and

(iii) $M_1 \sim M_2$ implies $\forall N \in \mathbf{N_{nf}}$: $M_1|N \sim M_2|N$.

Proof Sketch. We give a sketch of the proof in what follows. The complete proof is given in Appendix A.

We show parts *(i–iii)* of the lemma simultaneously by considering the set $S = \{((\nu\tilde{g})(\nu\tilde{x})(M_1|N), (\nu\tilde{g})(\nu\tilde{x})(M_2|N)) \mid M_1 \sim M_2, \tilde{g} \subseteq \mathbf{Gn}, \tilde{x} \subseteq \mathbf{Pn}, M_1, M_2, N \in \mathbf{N_{nf}}\}$. Following Lemma 5, it is sufficient to show that S is a strong bisimulation upto $\equiv$.

Note that if $M_1 \sim M_2$ then $fgn(M_1) = fgn(M_2)$, and hence $fgn((\nu\tilde{g})(\nu\tilde{x})(M_1|N)) = fgn((\nu\tilde{g})(\nu\tilde{x})(M_2|N))$ for all $\tilde{g}, \tilde{x}$ and N. We then show that every transition from $(\nu\tilde{g})(\nu\tilde{x})(M_1|N)$ can be matched by $(\nu\tilde{g})(\nu\tilde{x})(M_2|N)$ by considering the derivations of transitions. Transitions for $(\nu\tilde{g})(\nu\tilde{x})(M_1|N)$ can be derived by use of rules CLOSE, GNAME-RES1, GNAME-RES2, MOBILITY, PAR, UNI-COM, UNI-CLOSE, COM, COM-RES, UNI-OPEN, OPEN and PNAME-RES. Only the last three steps of each transition derivation are considered in the proof. Most importantly, following Lemma 8, we do not need to consider derivations that use STRUCT rules in the last two steps. From the structural operational semantics, the last step of a derivation will be due to the outermost $(\nu\tilde{g})$ in the expression, the next-to-last step will be due to the $(\nu\tilde{x})$ following the outermost $(\nu\tilde{g})$, and the step before that will be due to the parallel composition $(M_1|N)$. We omit in the proof the

symmetric cases arising due to the commutativity of the parallel operator '|'. This gives rise to 15 cases (owing to the combinations of rules applied during the last three steps in a derivation). For illustration, we show here one such case:
Case CLOSE, OPEN, COM:
$(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N'\{x'/y\})$ given $M_1 \xrightarrow{\overline{G}x'} M_1'$ and $N \xrightarrow{G'(y)} N'$. The derivation is as follows, where $\tilde{x}_1 = \tilde{x} \setminus \{x'\}$.

$$\begin{array}{lcr}
\text{COM:} & \dfrac{M_1 \xrightarrow{\overline{G}x'} M_1' \quad N \xrightarrow{G'(y)} N'}{M_1|N \xrightarrow{\overline{G}x'} M_1'|N'\{x'/y\}} & G \cap G' \neq \emptyset \\
\text{OPEN:} & \dfrac{}{(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{G}x'} (\nu\tilde{x}_1)(M_1'|N'\{x'/y\})} & \\
\text{CLOSE:} & \dfrac{}{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu x')(\nu\tilde{x}_1)(M_1'|N'\{x'/y\})} & G \setminus \tilde{g} = \emptyset
\end{array}$$

Since $M_1 \sim M_2$, $M_1 \xrightarrow{\overline{G}x'} M_1'$ means that there is an M_2' such that $M_2 \xrightarrow{\overline{G}x'} M_2'$ and $M_1' \sim M_2'$. Moreover, there exist expressions M_{N1}', M_{N2}' and N_N' in normal form such that $M_1' \equiv M_{N1}'$, $M_2' \equiv M_{N2}'$ and $N'\{x'/y\} \equiv N_N'$. Now, since $M_1' \sim M_2'$, we know $M_{N1}' \sim M_{N2}'$. Hence by construction of S, we can conclude that the pair $(\ (\nu\tilde{g})(\nu x')(\nu\tilde{x}_1)(M_{N1}'|N_N'), (\nu\tilde{g})(\nu x')(\nu\tilde{x}_1)(M_{N2}'|N_N')\ ) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x})(M_1'|N'\{x'/y\}), (\nu\tilde{g})(\nu\tilde{x})(M_2'|N'\{x'/y\})) \in \equiv S \equiv$.
By considering the 15 cases that cover all possible derivations, we conclude that for every transition from $(\nu\tilde{g})(\nu\tilde{x})(M_1|N)$, there is a transition from $(\nu\tilde{g})(\nu\tilde{x})(M_2|N)$ such that the destinations of the two transitions are related by $\equiv S \equiv$. Thus we establish that S is a strong bisimulation upto $\equiv$. □

Theorem 10 (Congruence) *$\sim$ is a congruence for the ω-calculus; i.e., for all $M_1, M_2 \in \mathbf{N}$, the following hold:*
(i) $M_1 \sim M_2$ implies $\forall x \in \mathbf{Pn} : (\nu x)M_1 \sim (\nu x)M_2$;
(ii) $M_1 \sim M_2$ implies $\forall g \in \mathbf{Gn} : (\nu g)M_1 \sim (\nu g)M_2$; and
(iii) $M_1 \sim M_2$ implies $\forall N \in \mathbf{N} : M_1|N \sim M_2|N$.

Proof: Let $M_1 \equiv M_{N_1}$ and $M_2 \equiv M_{N_2}$, where $M_1, M_2 \in \mathbf{N}$ and $M_{N_1}, M_{N_2} \in \mathbf{N_{nf}}$. Then the following hold:

- $M_1 \sim M_2$ implies $M_{N_1} \sim M_{N_2}$ (from Definition 2 and Lemma 5). $M_{N_1} \sim M_{N_2}$ implies $\forall x \in \mathbf{P_n}$: $(\nu x)M_{N_1} \sim (\nu x)M_{N_2}$ (by Lemma 9), which in turn implies

$(\nu x)M_1 \sim (\nu x)M_2$ (from Definition 2 and Lemma 5). Therefore, whenever $M_1 \sim M_2$ then $(\nu x)M_1 \sim (\nu x)M_2$.

- $M_1 \sim M_2$ implies $M_{N_1} \sim M_{N_2}$ (from Definition 2 and Lemma 5). $M_{N_1} \sim M_{N_2}$ implies $\forall g \in \mathbf{G_n}$: $(\nu g)M_{N_1} \sim (\nu g)M_{N_2}$ (by Lemma 9), which in turn implies $(\nu g)M_1 \sim (\nu g)M_2$ (from Definition 2 and Lemma 5). Therefore, whenever $M_1 \sim M_2$ then $(\nu g)M_1 \sim (\nu g)M_2$.

- $M_1 \sim M_2$ implies $M_{N_1} \sim M_{N_2}$ (from Definition 2 and Lemma 5). $M_{N_1} \sim M_{N_2}$ implies for any $N \in \mathbf{N}$, and $N \equiv N_N$ where, $N_N \in \mathbf{N_{nf}}$: $(M_{N_1}|N_N) \sim (M_{N_2}|N_N)$ (by Lemma 9), which in turn implies $(M_1|N) \sim (M_2|N)$ (from Definition 2 and Lemma 5). Therefore, whenever $M_1 \sim M_2$ then $(M_1|N) \sim (M_2|N)$.

$\sim$ is preserved by all node contexts. Hence, $\sim$ is a congruence . $\square$

Recall that we defined a late semantics for transition systems in the ω-calculus. Late semantics yields a more deterministic proof system for deriving transitions (as compared to that of early semantics) because of the late instantiation of an input name. Using the late semantics we defined a strong late bisimulation. Late bisimulation is more natural for automated verification tools because the late instantiation of names leads to more efficient checking of bisimulation.

Early bisimulation can also be defined for the ω-calculus using the late semantics. It turns out that early bisimulation equivalence is also a congruence for the ω-calculus. The fact that nodes in the ω-calculus cannot be placed in the context of an input or output prefix (only processes can be) makes the congruence result hold for both late and early bisimulation equivalences. In contrast for the π-calculus neither late nor early bisimulation equivalence is a congruence due to input prefix. The congruence results for early bisimulation equivalence in the ω-calculus can be established similar to that for the late bisimulation. A lemma similar to Lemma 9 can be used to prove congrunce for early bisimulation equivalence. In the proof of Lemma 9 given in Appendix A, all the cases except case 9 will be identical to those for late bisimulation equivalence. Case 9 which includes derivations for input transition labels, will have a more elaborate proof for early bisimulation equivalence to consider early instantiation of input names.

Weak Bisimulation for the ω-Calculus. We can also define a notion of *weak bisimulation* for the ω-calculus, in which τ- and μ-actions are treated as unobservable. Its definition is similar to that for strong bisimulation (Definition 1) and is given in Definition 4. We also establish that weak bisimulation equivalence, like its strong counterpart, is a congruence for the ω-calculus.

We use $\Longrightarrow$ to denote $\xrightarrow{(\tau|\mu)^*}$, i.e., zero or more τ- or μ-transitions, and $\overset{\widehat{\alpha}}{\Longrightarrow}$ to denote $\xrightarrow{(\tau|\mu)^*}\xrightarrow{\alpha}\xrightarrow{(\tau|\mu)^*}$ if $\alpha \notin \{\tau, \mu\}$ and $\Longrightarrow$ otherwise.

Definition 4 *A relation $S \subseteq \mathbf{N} \times \mathbf{N}$ is a* weak simulation *if $M\,S\,N$ implies:*

- *$fgn(M) = fgn(N)$, and*
- *whenever $M \xrightarrow{\alpha} M'$ where $bn(\alpha)$ is fresh then:*
  - *if $\alpha \in \{G(x),\, z\!:\!G(x)\}$, there exists an N'' s.t. $N\Longrightarrow\xrightarrow{\alpha} N''$ and for all $y \in \boldsymbol{Pn}$, there exists an N' s.t. $N''\{y/x\}\Longrightarrow N'$ and $M'\{y/x\}\,S\,N'$,*
  - *if $\alpha \notin \{G(x),\, z\!:\!G(x)\}$, there exists an N' s.t. $N \overset{\widehat{\alpha}}{\Longrightarrow} N'$ and $M'\,S\,N'$.*

S is a weak bisimulation *if both S and S^{-1} are weak simulations. Nodes M and N are* weak bisimilar, *written $M \approx N$, if $M\,S\,N$, for some* weak bisimulation *S.*

Proposition 11 *(i) $\approx$ is an equivalence relation; and (ii) $\approx$ is the largest weak bisimulation.*

Weak bisimulation equivalence is a congruence for the ω-calculus as formally stated below.

Theorem 12 (Congruence) *$\approx$ is a congruence for the ω-calculus; i.e., for all $M_1, M_2 \in \mathbf{N}$, the following hold:*
(i) $M_1 \approx M_2$ implies $\forall x \in \mathbf{Pn} : (\nu x)M_1 \approx (\nu x)M_2$;
(ii) $M_1 \approx M_2$ implies $\forall g \in \mathbf{Gn} : (\nu g)M_1 \approx (\nu g)M_2$; and
(iii) $M_1 \approx M_2$ implies $\forall N \in \mathbf{N} : M_1|N \approx M_2|N$.

Proof Sketch. It suffices to show that $S = \{((\nu\tilde{g})(\nu\tilde{x})(M_1|N),\, (\nu\tilde{g})(\nu\tilde{x})(M_2|N)) \mid M_1 \approx M_2, \tilde{g} \subseteq \mathbf{Gn}, \tilde{x} \subseteq \mathbf{Pn}, M_1, M_2, N \in \mathbf{N}\}$ is a weak bisimulation.

$M_1 \approx M_2$ implies that if $M_1 \xrightarrow{\alpha} M_1'$, where $\alpha \notin \{G(x),\, z:G(x)\}$, then there exists an M_2' such that $M_2 \overset{\widehat{\alpha}}{\Longrightarrow} M_2'$ and $M_1' \approx M_2'$. $M_2 \overset{\widehat{\alpha}}{\Longrightarrow} M_2'$ implies that there exist M_{2a} and M_{2b} such that $M_2 \Longrightarrow M_{2a}$, $M_{2a} \xrightarrow{\alpha} M_{2b}$, and $M_{2b} \Longrightarrow M_2'$.

It can be shown that if $(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\alpha} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N')$, then also $(\nu\tilde{g})(\nu\tilde{x})(M_{2a}|N) \xrightarrow{\alpha} (\nu\tilde{g})(\nu\tilde{x})(M_{2b}|N')$ which implies $(\nu\tilde{g})(M_2|N) \overset{\widehat{\alpha}}{\Longrightarrow} (\nu\tilde{g})(M_2'|N')$. We can similarly reason about the case when $\alpha \in \{G(x),\, z:G(x)\}$. Using these arguments along with the ideas used in the proof of congruence for strong bisimulation equivalence, it can be shown that weak bisimulation equivalence for ω-calculus is a congruence. □

Symbolic Bisimulation for the ω-Calculus. We now proceed to define symbolic bisimulation for the ω-calculus. As desired, the congruence results of Section 4 can be established for this extension as well.

Definition 5 *A relation $S \subseteq \mathbf{N} \times \mathbf{N}$ on nodes is a* symbolic simulation *if $M\,S\,N$ implies:*

- *fgn(M) = fgn(N), and*
- *whenever $M \xrightarrow{C_1,\alpha} M'$, where $bn(\alpha)$ is fresh, there exist N', β, and C_2 s.t. $N \xrightarrow{C_2,\beta} N'$ and*
  - $C_1 \triangleright C_2$
  - $\alpha\sigma_{C_1} \equiv \beta\sigma_{C_1}$
  - $M'\sigma_{C_1}\ S\ N'\sigma_{C_1}$.

S is a symbolic bisimulation *if both S and S^{-1} are symbolic simulations. Nodes M and N are* symbolic bisimilar, *written $M \asymp N$, if $M\,S\,N$ for some* symbolic bisimulation *S.*

Proposition 13 *(i) $\asymp$ is an equivalence relation; and (ii) $\asymp$ is the largest symbolic bisimulation.*

Symbolic bisimulation equivalence is a congruence for the ω-calculus, as formally stated below.

Theorem 14 (Congruence for Symbolic Bisimulation for the ω-Calculus) *$\asymp$ is a congruence for the ω-calculus; i.e., for all $M_1, M_2 \in \mathbf{N}$, the following hold:*

(i) $M_1 \asymp M_2$ implies $\forall x \in \mathbf{Pn} : (\nu x)M_1 \asymp (\nu x)M_2$;

(ii) $M_1 \asymp M_2$ implies $\forall g \in \mathbf{Gn} : (\nu g)M_1 \asymp (\nu g)M_2$; and

(iii) $M_1 \asymp M_2$ implies $\forall N \in \mathbf{N} : M_1|N \asymp M_2|N$.

See Appendix B for a proof for the ω_0-calculus; proofs for the extended calculi follow along the same lines.

For mismatch-free fragment of the ω-calculus, the symbolic bisimulation and the strong (late) bisimulation coincide. The notion of weak transitions used in defining the weak bisimulation for the ω-calculus, can be lifted to the symbolic semantics to define symbolic weak transitions. A weak version of symbolic bisimulation can be defined over the symbolic weak transitions.

Chapter 5

Towards Verification of ω-Calculus Specifications

5.1 Prototype Verifier for the ω-Calculus

In this section, we present two developments that yield a prototype system for the specification and verification of ω-calculus specifications. First, we extend the calculus with constructs that simplify the specification of practical MANET protocols. Secondly, we show how the transitional semantics of the ω-calculus can be directly encoded in Prolog, in order to generate the transition system corresponding to a given specification.

Syntactic extensions to the ω-calculus. The ω-calculus provides the basic mechanisms needed to model MANETs. In order to make specifications more concise, we extend the calculus to a polyadic version (along the lines of the polyadic π-calculus [Mil93]) and add support for data types such as bounded integers and structured terms. The matching prefix is extended to include equality over these types. Terms composed of these types can be used as values in a unicast or local broadcast transmission, or as actual parameters in a process invocation. We also introduce set-valued types and permit the use of a membership operation (denoted by '$\in$') in a match. Note that finite sets can be represented by finite terms, and the test for set membership can be implemented by a sum of equality tests. Hence the addition of set-valued data can be regarded as syntactic sugar and does not affect the proofs of the properties of the calculus given in Section 4. The modifications to the theory developed in the preceding sections to account for these syntactic extensions to the calculus are straightforward.

Encoding the transitional semantics in Prolog. Following the Prolog encoding of the semantics of value-passing CCS and the π-calculus [RRR$^+$97, YRS04] using the XSB tabled logic-programming system [XSB], we encoded the symbolic transitional semantics of the ω-calculus using Prolog rules. Each inference rule of the semantics is represented as a rule for the predicate `trans`, which evaluates the transition relation of an ω-calculus model.

In our encoding, each ω-calculus expression is represented as a Prolog term. For instance, a basic node expression of the form $P : G$ is represented by the term `basic(P, G)`, where `P` and `G` are the Prolog terms representing the process expression P and set of group names G, respectively. The key aspect of our encoding is to represent names in the ω calculus—pnames as well as gnames—as Prolog variables. This representation was used for the π-calculus in our earlier work [YRS04].

Using this representation, several operations such as renaming of bound names need not be implemented by our encoding explicitly; the way the deductive engine of Prolog handles logical variables implements all the necessary name manipulation. For instance, our encoding of the transitional semantics does not have to handle substitutions to names explicitly (which arise in the application of process names). Moreover, it is not necessary to encode alpha-renaming; bound names are implicitly renamed when clause instances are picked by Prolog's resolution step (which renames all variables in the selected program clause). Finally, when encoding the symbolic semantics of ω-calculus, we explicitly represent only the disequality constraints on transitions (i.e., those of the form $x \neq y$); the equality constraints are processed implicitly using Prolog's unification mechanism.

In our encoding, each symbolic transition is represented by an instance of a 4-ary prolog predicate `trans`. In particular, a symbolic transition of the form $M_1 \xrightarrow{C,\alpha} M_2$ is represented by `trans(`M_1`, `C`, `α`, `M_2`)`. The derivation of a symbolic transition from the semantic rules is realized by encoding the rules as clauses defining `trans`, and using query evaluation in Prolog.

Figure 5.1 shows the Prolog encoding of selected symbolic transition rules of the

```
% MCAST
trans(basic(pref(bcast(X),PExp),Gs), [], bcast(Gs,X), basic(PExp,Gs)).

% RECV
trans(basic(pref(recv(X),PExp),Gs), [], brecv(Gs,X), basic(PExp,Gs)).

% UNI-SEND
trans(basic(pref(out(Z,X),PExp),Gs), [], unisend(Z,Gs,X), basic(PExp,Gs)).

% COM
trans(par(M1, N1), C, bcast(Gs1, X), par(M2, N2)) :-
  trans(M1, C1, bcast(Gs1, X), M2),
  trans(N1, C2, brecv(Gs2, X), N2),
  non_disjoint(Gs1, Gs2), % sets Gs1 and Gs2 have common gname(s)
  and(C1, C2, C).

% GNAME-RES2
trans(nu_g(G, M1), C, tau, nu_g(G, M2)) :-
  trans(M1, C, bcast(Gs, X), M2),
  Gs == [G].

% UNI-OPEN
trans(nu(X, M1), C, unires(Z, Gs, X), M2) :-
  trans(M1, C1, unisend(Z, Gs, Y), M2),
  X == Y, Z \== X,
  remove_from_constraint(C1, X, C).  % C = C1-X
```

Figure 5: Encoding of selected symbolic transition rules in Prolog.

ω-calculus. Observe that the encoding of the MCAST, RECV and UNI-SEND rules is straightforward. For these rules, the constraint **true** is represented by the empty list '`[]`'. The encoding of COM rule is also direct. Predicate `non_disjoint` is used to test for non-disjointness of two sets (represented as lists) and predicate `and` is used to compute the conjunction of two constraints. Note that in a broadcast-receive transition of the form $M \xrightarrow{C,G(y)} M'$, the name y is a bound name of M. In the symbolic semantics, we assume that before applying the COM rule, y is renamed to the name received in COM. Such alpha-renaming corresponds to an application of the STRUCT rule. In our encoding, we first ensure that all bound names are mapped to distinct Prolog variables and are also distinct from free names. We then ensure that the receiving and sent names are identical by unifying the corresponding Prolog variables.

The GNAME-RES2 rule is applicable only when the set of group names involved in the broadcast action (`Gs` in our encoding) is a singleton set containing only `G`, the restricted group name. Since group names are encoded as variables, this check has to be performed by testing if `Gs` is *identical* to `[G]`: i.e. whether for all substitutions θ the two terms `Gs`θ and `[G]`θ are the same. This test is accomplished using the "`==`" operator in Prolog. Note that if we had used "`=`" instead, we would have incorrectly unified `Gs` with `[G]`, thereby possibly treating two distinct group names as the same.

Finally, consider the encoding of UNI-OPEN. This rule is applicable when the name sent by unicast is a restricted name. In the encoding, we apply this rule by first generating transitions from `M1`, and then checking if the name `Y` sent by unicast is same as the restricted name `X`. As in the case of GNAME-RES2, we use "`==`" to test whether two names are identical. This rule also uses "`\==`" to test whether two names are not identical (i.e. distinct).

The other symbolic transition rules of the ω-calculus are encoded similarly. As remarked earlier, the key aspect of the encoding is the representation of pnames and gnames by Prolog variables, following [YRS04]. The soundness and completeness of our encoding can be established along the same lines as in [YRS04].

5.2 Modeling and Verifying MANET Protocols using the ω-Calculus

We used our Prolog encoding of the semantics of the ω-calculus to develop and analyze formal ω-calculus models of a leader-election algorithm for MANETs [VKT04] and the AODV routing protocol [PBRD03]. The main purpose of these case studies is to show that models of realistic MANET protocols can be constructed in the ω-calculus, and the semantics of these encodings, in terms of labeled transition systems, can be effectively computed. We use the derived transition systems to verify reachability properties of these protocols.

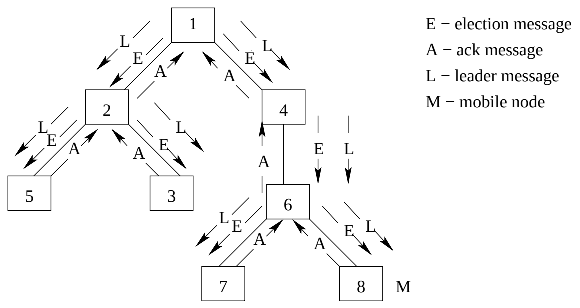


Figure 6: Message flow in leader election protocol

5.2.1 Case Study 1: A Leader Election Protocol for MANETs

The algorithm of [VKT04] elects the node with the maximum id among a set of connected nodes as the leader of the connected component. A node that initiates a leader election sends an *election* message to its neighboring nodes. The recipients of the *election* message mark the node from which they received the message as their parent and send the *election* message to their neighbors, thereby building a spanning tree with the initiator as the root. After sending an *election* message, a node awaits acknowledgements from its children in the spanning tree. A child node n sends its parent an acknowledgement *ack* with the maximum id in the spanning tree rooted at n. The maximum id in the spanning tree is propagated up the tree to the root. The root node then announces the leader to all the nodes in its spanning tree by sending a *leader* message. To keep track of the neighbors of a node, *probe* and *reply* messages are used periodically. When a node discovers that it is disconnected from its *leader*, it initiates an election process. The flow of *election*, *ack*, and *leader* messages is depicted in Fig. 6, where the node with id 1 is the initiator.

Specification of the leader election protocol in the ω-calculus. We model a network as the parallel composition of basic ω-nodes, whose process interfaces reflect the initial topology of the network. Each node runs an instance of process $node(id, chan, init, elec, lid, pChan)$ defined in Fig. 7. The meaning of this process's parameters is the following: id is the node identifier; $chan$ is an input channel; $init$ indicates whether the node initiates the election process; $elec$ indicates whether the node is part of the election process; lid represents the node's knowledge of the leader id; and $pChan$ is the parent's input channel. These parameters are represented by pnames and integers.

A node may receive $election, ack$, and $leader$ messages, representing an election message, an acknowledgement to the election process, and a leader message, respectively. We need not consider *probe* and *reply* messages in our model because a node can broadcast to its neighbors without knowing its neighbors, and the effect of disconnection between nodes can be modeled using the choice operator. The ω-calculus model of the protocol is given in Fig. 7. The messages, their parameters, and the parameters used in the definitions appearing in Fig. 7 are explained below:

Messages: $election(sndrChan)$; $ack(maxid)$; $leader(maxid)$.

Message parameters: $sndrChan$: input channel of the sender of the message; $maxid$: maximum id seen so far by the sender of the message.

Definition parameters: *id*: id of the node, *chan*: input channel of the node; *init*: 1 if node initiated the election process, 0 otherwise; *elec*: 1 if node is participating in the election process, 0 otherwise; *lid*: node's knowledge of the leader id; $pChan$: input channel of the node's parent in the spanning tree; $sndrChan$: input channel of the sender node of the message; $maxid$: maximum id seen so far by the node.

An example specification of an eight-node network running the leader election protocol of Fig. 7 is given in Fig. 8. The initial network topology is the same as that of the network of Fig. 6. The node with id 1 (*initElection*) is designated to be the initiator of the leader-election process. The last parameter *none* in the process invocations indicates that the parent channel is initially not known to the processes.

Verifying the leader election protocol model. Using our implementation of the transitional semantics of the ω-calculus, we verified the following correctness property for the leader election protocol for MANETs: *On some computation path in the transition system, eventually a node with the maximum id in a connected component is elected as the leader of the component, and every node connected to it (via one or more hops) learns about it.*

Note that the reachability property stated above does not guarantee that a leader will be always computed. In fact, due to lossy communication, there will be paths in

/* A node may receive an *election* or a *leader* message. */

$node(id, chan, init, elec, lid, pChan) \overset{def}{=}$
$\quad \mathbf{r}(election(sndrChan)).\ processElection(id, chan, init, 1, lid, pChan, sndrChan)$
$\quad + \mathbf{r}(leader(maxid)).\ processLeader(id, chan, init, elec, lid, pChan, maxid)$

/* Node that initiates election process broadcasts *election* msg and awaits *ack* in state *awaitAck*. */

$initElection(id, chan, init, elec, lid, pChan) \overset{def}{=}$
$\quad \overline{\mathbf{b}}\, election(chan).\ awaitAck(id, chan, init, 1, id, none)$

/* When a node receives an *election* message it reaches the *processElection* state where it broadcasts the *election* message and goes to state *awaitAck*. */

$processElection(id, chan, init, elec, lid, pChan, sndrChan) \overset{def}{=}$
$\quad \overline{\mathbf{b}}\, election(chan).\ awaitAck(id, chan, init, elec, lid, sndrChan)$

/* A node in *awaitAck* state may receive an *ack* and reach *processAck* state or it may nondeterministically conclude that it has received *ack* from all its children in the spanning tree. In the latter case, it declares the leader by broadcasting a *leader* message if it is the initiator. Otherwise, it sends (unicast) an *ack* to its parent node (*pChan*) with the maximum id in the spanning tree rooted at this node. */

$awaitAck(id, chan, init, elec, lid, pChan) \overset{def}{=}$
$\quad chan(ack(maxid)).\ processAck(id, chan, init, elec, lid, pChan, maxid)$
$\quad + [init = 1]\, \overline{\mathbf{b}}\, leader(lid).\ node(id, chan, init, 0, lid, pChan)$
$\quad + [init = 0]\, \overline{pChan}\, ack(id, lid).\ node(id, chan, init, elec, lid, pChan)$

/* On receiving an *ack*, a node stores the maximum of the ids received in *ack* messages. */

$processAck(id, chan, init, elec, lid, pChan, maxid) \overset{def}{=}$
$\quad [maxid >= lid]\, awaitAck(id, chan, init, elec, maxid, pChan)$
$\quad + [maxid < lid]\, awaitAck(id, chan, init, elec, lid, pChan)$

/* On receiving a *leader* message, a node sets its *lid* parameter to the *maxid* in the *leader* message. If *maxid* is less than *lid*, then either the node was not part of the election process or did not report *ack* to its parent node (probably because it moved away from its parent). In either case, it broadcasts its *lid* as the maximum id. */

$processLeader(id, chan, init, elec, lid, pChan, sndrChan, maxid) \overset{def}{=}$
$\quad [maxid = lid]($
$\qquad [elec = 1]\ \overline{\mathbf{b}}\, leader(maxid).\ node(id, chan, init, 0, lid, pChan)$
$\qquad + [elec = 0]\ node(id, chan, init, 0, lid, pChan)$
$\quad)$
$\quad + [maxid > lid]\ \overline{\mathbf{b}}\, leader(maxid).\ node(id, chan, init, 0, maxid, pChan)$
$\quad + [maxid < lid]\ \overline{\mathbf{b}}\, leader(lid).\ node(id, chan, init, 0, lid, pChan)$

Figure 7: ω-calculus encoding of the leader election protocol for MANETs.

$$\begin{aligned} M = (\nu a)(\nu b)(\nu c)(\nu d)(\nu e)(\nu h)(\nu i)(\nu j)(\nu g_1)(\nu g_2)(\nu g_3)(\nu g_4)(\nu g_5)(\nu g_6)(\nu g_7) \\ (\mathit{initElection}(1, a, 1, 0, 1, \mathit{none}) : \{g_1, g_2\} \\ \mid \mathit{node}(2, b, 0, 0, 2, \mathit{none}) : \{g_1, g_3, g_4\} \\ \mid \mathit{node}(3, c, 0, 0, 3, \mathit{none}) : \{g_4\} \\ \mid \mathit{node}(4, d, 0, 0, 4, \mathit{none}) : \{g_2, g_5\} \\ \mid \mathit{node}(5, e, 0, 0, 5, \mathit{none}) : \{g_3\} \\ \mid \mathit{node}(6, h, 0, 0, 6, \mathit{none}) : \{g_5, g_6, g_7\} \\ \mid \mathit{node}(7, i, 0, 0, 7, \mathit{none}) : \{g_6\} \\ \mid \mathit{node}(8, j, 0, 0, 8, \mathit{none}) : \{g_7\}) \end{aligned}$$

Figure 8: ω-calculus specification of leader election protocol for an 8-node tree-structured network.

Nodes	Tree			Ring		
	States	Transitions	Time(sec)	States	Transitions	Time(sec)
5	77	96	0.97	98	118	1.22
6	168	223	3.35	212	281	4.45
7	300	455	11.55	453	664	17.58
8	663	1073	45.85	952	1560	71.22

Table 12: Verification statistics for ω-calculus model of leader election protocol.

the transition system where a leader may never be elected; hence the correctness condition can be shown only using fairness assumptions, e.g. that message loss does not happen infinitely often. Our implementation verifies reachability properties without fairness conditions, and hence we only verify the weaker property stated above.

The verification was performed on models having *tree-* and *ring*-structured initial topologies. A distinguished node (with maximum id, for example, node 8 marked ‘M’ for “mobile” in Fig. 6) was free to move as long as the network remained connected. A mobility invariant was used to constrain the other nodes to remain connected to their neighbors. For verification purposes, we added a node *final* to the model that remains connected to all other nodes. A node, upon learning its leader, forwards this information to node *final*. After *final* receives messages from every other node with their leader ids equal to the maximum id in the network, it performs the observable action $\mathit{action}(\mathit{leader}(MaxId))$. The closed ω-specification of the protocol was checked for weak bisimilarity with an ω-specification that emits $\mathit{action}(\mathit{leader}(MaxId))$ as the only observable action. Weak bisimilarity between these two specifications indicates that the correctness property is true of the system.

Our Prolog encoding of the weak bisimulation checker for the ω-calculus includes the weak version of the transition relation, abstracting τ- and μ-transitions, encoded as the `dtrans` predicate. The predicate `nb(S1, S2)` checks if two ω-specifications $S1$ and $S2$ are weak bisimilar.

We verified the correctness property for networks containing 5 through 8 nodes. Table 12 lists the states, transitions and time (in seconds) it took our Prolog implementation of the calculus and weak bisimulation checker to verify the property for networks with initial tree and ring topologies.

5.2.2 Case Study 2: The AODV Routing Protocol

The *Ad Hoc On-Demand Distance Vector* (AODV) protocol [PBRD03] is a routing protocol that discovers and maintains point-to-point routes in a MANET. Route discovery is initiated by a node on demand. If a node (source) does not know a route to a destination node to which it wants to route a packet, it initiates route discovery by locally broadcasting a *route request*. On receiving a *route request*, if a node knows a route to the destination node or is itself the destination, it responds to the sender node with a *route reply*, otherwise it forwards (locally broadcasts) the *route request*. Each route request is marked with a broadcast-id, assigned by the originator node of the request. The broadcast-id and the originator node's id uniquely define a route request, and are used to avoid processing of duplicate requests. The broadcast-id is incremented by a node every time it originates a route request. Sequence numbers are used with route requests and route replies to maintain freshness of routes. Route error messages are used to convey invalidation of routes due to staleness of routes, indicated by a lower sequence number.

Specification of the AODV Protocol in the ω-calculus. We model a MANET as the parallel composition of basic ω-nodes. The interfaces of all nodes are initialized in accordance with the initial topology of the network. Each node in the network runs an instance of process *aodv* defined in Fig. 9. Process *aodv* has the following parameters: process identifier *id* (a pname), broadcast id *bid*, sequence number *sqn* (for messages and route requests), route table *rt* (a list of tuples), set of previously seen route requests *rreqs* (a list of tuples), and set of known destinations *kD* (a list of pnames). These parameters record the state of a node which may change as the protocol runs and the network evolves. An *aodv* process can receive a message either destined for it, or a message locally broadcasted by a neighboring node. A node may receive *data*, *rreq*, *rrep*, *rerr* messages representing data packet, route request, route reply and route invalidation, respectively. On receiving a message, the protocol may modify its state and/or broadcast a message. The *aodv* process invokes message handlers, defined using ω-process definitions, to process the received messages. Reception of *data*, *rreq*, *rrep*, and *rerr* (parameterized) messages is handled by processes defined by $pktP$, $rreqP$, $rrepP$, and $rerrP$, respectively (See Fig. 9). A route table *rt* is a set of tuples with each tuple containing id, sequence number, hop count, next hop, active neighbors, and route validity for each known destination node. Data manipulation code for updating route table (*rt* to *nrt*), extracting next hop (y), sequence number (*dsqn*), and active neighbors (*dactn*) for a destination, from the route table, and incrementing sequence number, broadcast id, and hop count is omitted from the encoding given in Fig. 9.

On receiving a data packet, a node accepts it if the packet is destined for it, otherwise if it knows the route to the destination, it sends the packet to the next hop towards the destination node, else it initiates a route discovery for the destination node. On receiving a route request *rreq*, a node replies with *rrep*, if it knows a route to the destination, otherwise it forwards the *rreq* via local broadcast. Each such request is associated with a broadcast id (*mbid*) set by the originator (identified by *srcid*) of the message. A route request *rreq* is discarded if it had been received previously ($(srcid, mbid) \in rreqs$). Otherwise, the route table is updated (to *nrt*) with a route to node *srcid*. If the node itself is the destination node (identified by *did*) to which the route is sought, or if the node knows a route to the destination ($did \in kD$), a route-reply message (*rrep*) is sent. Otherwise, the node locally broadcasts the *rreq* message (via action $\overline{\mathbf{b}}$) with the hop count (*hops*) incremented by one (to *nhops*). On receiving a route reply *rrep*, a node updates its route table accordingly. If the node itself is not the initiator of the corresponding *rreq*, it forwards the *rrep* to the next hop towards the initiator node. Detection of a change in network topology is modeled using non-determinism. On detection of a change in network topology, a node invalidates the route table entry for the disconnected neighbor node, and sends a route error *rerr* to the affected nodes.

```
$aodv(id, bid, sqn, rt, rreqs, kD) \stackrel{\text{def}}{=}$
  $\mathbf{r}(msg).\ ([msg = pkt(data, did, sndrid)]$
      $pktP(data, did, sndrid, id, bid, sqn, rt, rreqs, kD)$
    $+\ [msg = rreq(hops, mbid, did, dsqn, srcid, ssqn, sndrid)]$
      $rreqP(hops, mbid, did, dsqn, srcid, ssqn, sndrid, id, bid, sqn, rt, rreqs, kD)$
    $+\ [msg = rrep(hops, did, dsqn, srcid, sndrid)]$
      $rrepP(hops, did, dsqn, srcid, sndrid, id, bid, sqn, rt, rreqs, kD)$
    $+\ [msg = rerr(did, dsqn, sndrid)]$
      $rerrP(did, dsqn, sndrid, id, bid, sqn, rt, rreqs, kD)\ )$

$pktP(data, did, sndrid, id, bid, sqn, rt, rreqs, kD) \stackrel{\text{def}}{=}$
    $[did = id]\ aodv(id, bid, sqn, rt, rreqs, kD)$
    $+\ [did \neq id]$  /* $y$ is the next hop node towards $did$. $nrt$ is obtained by
    adding $sndrid$ to $actn$ of $did$ in $rt$ */
    $(\ [did \in kD]\ \ \overline{y}\, pkt(data, did, id).\ aodv(id, bid, sqn, nrt, rreqs, kD)$
      + /* $newbid$ is $bid + 1$ and $rdsqn$ is the sequence number for $did$ in $rt$ */
      $[did \notin kD]\ \ \overline{\mathbf{b}}\, rreq(0, newbid, did, rdsqn, id, sqn, id).$
        $aodv(id, newbid, sqn, rt, rreqs, kD)\ )$

$rreqP(hops, mbid, did, dsqn, srcid, ssqn, sndrid, id, bid, sqn, rt, rreqs, kD) \stackrel{\text{def}}{=}$
    $[(srcid, mbid) \in rreqs]\ aodv(id, bid, sqn, rt, rreqs, kD)$
    $+\ [(srcid, mbid) \notin rreqs]\ ($
      /* $y$ is the next hop node towards $srcid$. maxsqn is the maximum of
      $sqn$ and $dsqn$. $nrt$ is obtained by updating the route to $srcid$ in $rt$. */
      $[did = id]\ \ \overline{\mathbf{b}}\, rrep(y, 0, id, maxsqn, srcid, id).$
        $aodv(id, bid, maxsqn, nrt, rreqs, kD)$
      + /* $dhops$ is the number of hops towards $did$ in $rt$. $rsqn$ is the
      sequence number for $did$. $nhops$ is $hops + 1$. */
      $[did \neq id]\ \ (\ [did \in kD]\ \ \overline{\mathbf{b}}\, rrep(y, dhops, did, rsqn, srcid, id).$
          $aodv(id, bid, sqn, nrt, rreqs, kD)$
        $+\ [did \notin kD]\ \ \overline{\mathbf{b}}\, rreq(nhops, mbid, did, dsqn, srcid, ssqn).$
          $aodv(id, bid, sqn, nrt, rreqs, kD)\ \ )\ )$

$rrepP(hops, did, dsqn, srcid, sndrid, id, bid, sqn, rt, rreqs, kD) \stackrel{\text{def}}{=}$
    /* $nrt$ is obtained by updating the route to $did$ in $rt$. */
    $[srcid = id]\ \ aodv(id, bid, sqn, nrt, rreqs, kD)$
    + /* $y$ is the next hop node towards $srcid$. $nhops$ is $hops + 1$. */
    $[srcid \neq id]\ \ \overline{y}\, rrep(nhops, did, dsqn, srcid, id).\ \ aodv(id, bid, sqn, nrt, rreqs, kD)$

$rerrP(did, dsqn, sndrid, id, bid, sqn, rt, rreqs, kD) \stackrel{\text{def}}{=}$
    $[did \in kD]$ /* $dactn$ are active neighbors for $did$ in $rt$.
      $nrt$ is obtained by invalidating the route to $did$. */
      $notifyAllRErr(dactn, rerr(did, dqsn, id), id, bid, sqn, nrt, rreqs, kD)$
    $+\ [did \notin kD]\ \ aodv(id, bid, sqn, rt, rreqs, kD)$

$notifyAllRErr(actn, msg, id, bid, sqn, rt, rreqs, kD) \stackrel{\text{def}}{=}$
    $[actn = []]\ \ aodv(id, bid, sqn, rt, rreqs, kD)$
    $+\ [actn \neq []]\ \ notifyRErr(actn, msg, id, bid, sqn, rt, rreqs, kD)$

$notifyRErr(actn, msg, id, bid, sqn, rt, rreqs, kD) \stackrel{\text{def}}{=}$
    /* $x$ is an element in $actn$ and $remactn$ are remaining elements of $actn$ */
    $\overline{x}\, msg.\ notifyAllRErr(remactn, msg, id, bid, sqn, rt, rreqs, kD)$
```

Figure 9: Encoding of the AODV protocol in the ω-calculus.

Verifying the AODV protocol model. Using our Prolog encoding of the transitional semantics of the ω-calculus, we verified a simplified version of the AODV routing protocol. The simplified version ignores sequence numbers and uses two distinguished nodes as the source and destination nodes for the route discovery process. Broadcast id (*bid*) and hop count (*hops*) are modeled as bounded integers. Routes get invalidated due to node movement or link failures along the route. We verified following properties:

- *deadlock-freedom:* There is no state in the model without an outgoing transition.
- *route-found:* As long as there exists a path from a source node to a destination node during route detection, on some computation path in the transition system it will eventually be detected.

It should be noted that, similar to the leader-election property, the *route-found* property is a weaker form of the correctness condition that a route is always found provided node mobility and message loss do not occur infinitely often.

The verification was performed on models with initial *line* topologies, with the destination node being 1-, 2-, 3-hops away from the source node in networks with 2, 3, and 4 nodes, respectively. The network topology was allowed to change freely during verification. The *deadlock-freedom* property involved searching for states with no transitions. In the model, when a node has found a route it performs an external action *action(routeFound).* The *route-found* property was verified by checking for reachability of a transition labeled *action(routeFound)* from the start state of the model. Table 13 lists the number of states and transitions generated using our XSB-based implementation of the ω-calculus for network models containing 2, 3 and 4 nodes, as well as the time (in seconds) it took to verify deadlock-freedom and route-found properties.

5.3 Discussion

We consider our current implementation of the calculus to be a prototype. Its main purpose is to demonstrate the feasibility and straightforwardness of implementing the calculus in a tabled logic-programming system. As future work, we plan to develop

Nodes	Deadlock Freedom			Route Found		
	States	Transitions	Time(sec)	States	Transitions	Time(sec)
2	8	16	0.07	5	10	0.06
3	30	78	0.25	15	39	0.16
4	380	1440	4.56	191	732	2.74

Table 13: Verification statistics for ω-calculus model of AODV protocol.

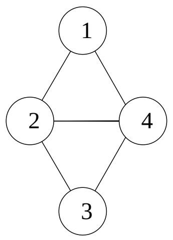
an optimizing compiler for the ω-calculus, along the lines of one for the π-calculus implemented in the MMC model checker [YDRS05]. As these prior results demonstrate, this should significantly improve the performance of our implementation.

We observed a number of benefits in using the ω-calculus to model the leader election protocol for MANETs and the AODV routing protocol. (1) Neither of these protocols assumes reliable communication. This fits well with the ω-calculus semantics which models lossy broadcast. (2) The concise and modular nature of our specifications is a direct consequence of the calculus's basic features, including separation of control behavior (processes) from neighborhood information (interfaces), and modeling support for unicast, local broadcast, and mobility. (3) The mobility constraints imposed on the leader election protocol model (Section 5.2.1) are specified independently of the control logic using a mobility invariant. For the case at hand, the invariant dictates that all nodes other than a distinguished node (node 8 in Fig. 6) remain connected to their initial neighbors. Thus, during protocol execution, process interfaces may change at will as long as the mobility invariant is maintained. (4) Our specifications of the leader-election protocol and the AODV protocol are given in the finite-control sub-calculus of the ω-calculus, thereby rendering them amenable to automatic verification; see also Theorem 3.

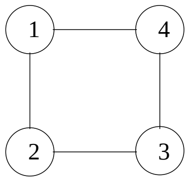
Chapter 6

Query-Based Model Checking of Ad Hoc Network Protocols

A prominent source of complexity in the verification of ad hoc network (AHN) protocols is the fact that the number of network topologies grows exponentially with the square of the number of nodes. To combat this *instance explosion* problem, we developed a query-based verification framework for AHN protocols that utilizes symbolic reachability analysis. Specifically we consider AHN nodes of the form $P : I$, where P is a process and I is an interface: a set of groups, where each group represents a multicast port. Two processes can communicate if their interfaces share a common group. To achieve a *symbolic* representation of network topologies, we treat process interfaces as variables and introduce a constraint language for representing topologies. Terms of the language are simply conjunctions of *connection* and *disconnection* constraints of the form $conn(\mathcal{J}_i, \mathcal{J}_j)$ and $dconn(\mathcal{J}_i, \mathcal{J}_j)$, where $\mathcal{J}_i$ and $\mathcal{J}_j$ are interface variables. Our *symbolic reachability algorithm* explores the symbolic state space of an AHN in breadth-first order, accumulating topology constraints as multicast-transmit and multicast-receive transitions are encountered. We demonstrate the practical utility of our framework by applying it to the problem of detecting unresolved collisions in the LMAC protocol for sensor networks.



(a) Topology with detected collision



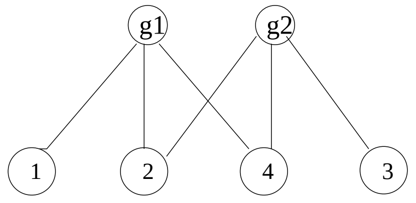
(b) Topology with undetected collision

Figure 10: Example topologies for collision and collision-detection in the LMAC protocol.

6.1 An Example of Topologies and Topology Constraints

Below we illustrate the use of a constraint language for representing sets of network topologies. In the LMAC protocol of [vHH04], which is used to allocate transmission slots in a sensor network MAC layer, collision, i.e. simultaneous transmission between two nodes with overlapping ranges, is detected by neighbors common to both nodes. Fig. 10(a) shows a network topology for which a collision between nodes 1 and 2 can be detected due to the presence of a common neighbor (node 4). Fig. 10(b) shows a topology for which a collision between 1 and 2 remains undetected since they do not share a neighbor.

As described earlier, we consider AHN nodes of the form $P : I$, where P is a process and I is an *interface*. Further, an interface is a set of *groups*, with each group g representing a shared communication channel and dually corresponding to a clique in the network topology. Figs. 11(a) and 11(b) provide a group-based view and concrete representation based on process interfaces of the network topology of Fig. 10(a). A symbolic representation of the same topology is given in Fig. 11(c) using connection (*conn*) and disconnection (*dconn*) constraints over interface variables $\mathcal{J}_1$–$\mathcal{J}_4$. The language in which symbolic topology constraints is expressed is formally described in Section 6.3. The symbolic representation permits us to compactly represent *sets* of topologies. For instance, consider the constraint



(a) Group-Based View

AHN
$\Pi_{1 \leq i \leq 4} P_i : I_i$
$I_1 = \{g1\}$
$I_2 = \{g1, g2\}$
$I_3 = \{g2\}$
$I_4 = \{g1, g2\}$

(b) Concrete Representation of Interfaces

AHN
$\Pi_{1 \leq i \leq 4} P_i : \mathcal{J}_i$
$conn(\mathcal{J}_1, \mathcal{J}_2)$
$conn(\mathcal{J}_1, \mathcal{J}_4)$
$conn(\mathcal{J}_2, \mathcal{J}_3)$
$conn(\mathcal{J}_2, \mathcal{J}_4)$
$conn(\mathcal{J}_3, \mathcal{J}_4)$
$dconn(\mathcal{J}_1, \mathcal{J}_3)$

(c) Symbolic Representation of Interfaces

Figure 11: Concrete and symbolic views of network topology of Fig. 10(a).

$conn(\mathcal{J}_1, \mathcal{J}_2) \wedge conn(\mathcal{J}_1, \mathcal{J}_4) \wedge conn(\mathcal{J}_2, \mathcal{J}_3) \wedge conn(\mathcal{J}_3, \mathcal{J}_4)$. This represents topologies that contain edges $(1, 2), (1, 4), (2, 3)$ and $(3, 4)$. The topologies in this set may or may not contain edges $(1, 3)$ and/or $(2, 4)$. Hence the above constraint represents four 4-node topologies, including the ones in Fig. 10. We use topology constraints when constructing a symbolic verification proof (by reachability or model checking) to consider a set of topologies simultaneously. These constraints may get refined as needed as we progress in the proof, corresponding to case splits among the set of topologies. The constraint representation and lazy case-splitting enable us to consider a large number of topologies simultaneously within a single verification run.

6.2 Related Work

Our symbolic approach to query-based model checking of AHN protocols can be considered a form of *constraint-based model checking*. Traditionally this technique has been used for the verification of infinite-state systems [DP99, Pod00], data-independent systems [SR03], systems with non-linear arithmetic constraints [CABN97], timed automata [Fri99], and imperative infinite-state programs [Fla04]. In these works, constraints were used to compactly represent sets of states of a system being verified. In contrast to these, our approach uses variables in the system specification (to represent interconnections) and finds their valuations (in this case, topologies) for which a property holds. In this sense, our approach is closely related to temporal logic query checking, introduced in [Cha00], which addresses the following problem: given a Kripke structure and a temporal logic formula with a placeholder, determine all propositional formulas ϕ such that when ϕ is inserted in the placeholder, the resulting temporal logic formula is satisfied by the Kripke structure. Query checking has been extended in a number of ways, including query checking of a wide range of temporal logics using a new class of alternating automata [BG01]; the application of query checking to a variety of model exploration tasks, ranging from invariant computation to test case generation [GCD03]; and its adaptation to solving temporal queries in which formulas may contain integer variables [ZC05].

Recently, symbolic representation of the set topologies has been used in [GFM09] to analyze ad hoc networks. The constraint language in that work can only express the presence of connections between nodes, and not the absence of connections, in contrast to our work. It should be noted that the undetected collision problem in the LMAC protocol (see Section 6.5) is due to absence of connections, and cannot be detected using the constraint language of [GFM09].

The correctness of the 4-node and 5-node LMAC protocol [vHH04] has been previously established in [FvHM07] using the UPPAAL model checker for timed automata. By systematically considering all 11 topologies for the 4-node case and all 61 topologies for the 5-node case (modulo isomorphism), they report all network topologies for which collisions may remain undetected in the LMAC protocol. They also iteratively improve the protocol model so that the number of topologies for which the protocol may fail is reduced. In contrast, our query-based approach verifies a property related to unresolved collisions using a single symbolic reachability run, thereby allowing us to additionally consider the 6-node case.

6.3 Modeling Framework

6.3.1 Syntax

We formally define the syntax and semantics of our framework. Systems in our framework are modeled as composition of *nodes*. Following the notion of separation of a node's communication and computation behavior presented in the ω-calculus, we consider a node to consist of a *process* (computational behavior) and an *interface* (communication capability). We present the notations used in defining our framework, followed by formal definitions of the components of our framework, namely a process, an interface, a node, and a system.

Let $\mathcal{D}$ be a non-empty *domain* with a set of *operations* F and *relations* R defined over it, and $\mathcal{V}ar$ be a countable set of *variables* over domain $\mathcal{D}$. For instance, $\mathcal{D}$ may be a set of finite integers, with F containing arithmetic operations, and R comprising equality, dis-equality and relational operations over integers. Symbols x, y (possibly subscripted) range over elements of $\mathcal{V}ar$. An *environment* $\theta : X \mapsto \mathcal{D}$, where $X \subseteq \mathcal{V}ar$ is a mapping from variables in $\mathcal{V}ar$ to values in domain $\mathcal{D}$. Symbol Θ is used to denote the set of all environments over $\mathcal{V}ar$ and $\mathcal{D}$. We use $\mathcal{E}$ to denote the set of *expressions*, which are terms over elements of $\mathcal{D} \cup \mathcal{V}ar \cup F$. Expressions are represented by symbol e (possibly subscripted). A *primitive condition* is a term with a symbol from R whose arguments are elements of $\mathcal{E}$. A *condition* is a conjunction of primitive conditions. An *assignment* is of the form $x := e$, where $x \in \mathcal{V}ar$ and $e \in \mathcal{E}$. Following traditional programming language semantics, we use $[\![.]\!]$ to represent semantics for expressions, conditions and assignments. For an expression e, condition *cond*, and assignment *asgn*, $[\![e]\!] : \Theta \mapsto \mathcal{D}$, $[\![cond]\!] : \Theta \mapsto Bool$, and $[\![asgn]\!] : \Theta \mapsto \Theta$ are mappings from an environment to domain $\mathcal{D}$, $Bool = \{\ \textbf{true}\ ,\ \textbf{false}\ \}$, and an environment, respectively. Semantics of a single assignment can be extended to a set of simultaneous assignments in the standard way.

The syntactic definition of a process is as follows.

Definition 6 (Process) *A* process $= \langle L, X, \Sigma, \delta, l_0, \eta_0 \rangle$, *is an extended finite state automaton over domain* $\mathcal{D}$*, where:*

- L *is a finite set of* locations.
- $X \subseteq \mathcal{V}ar$ *is a set of* local variables *for the process.*
- Σ *is a finite set of* action labels *containing*
  - $\mathbf{b}\ e$*,* $e \in \mathcal{E}$ *(*broadcast *action).*
  - $\mathbf{r}\ (x)$*,* $x \in X$ *(*receive *action).*

- δ *is a finite set of* transitions. *A transition is a tuple* $(l, \alpha, l', \langle \rho, \eta \rangle)$, *where*
  - $l, l' \in L$ *are* source *and* target *locations, respectively.*
  - $\alpha \in \Sigma$ *is an* action label.
  - ρ, *a condition, is a* transition guard.
  - η *is a set of* simultaneous assignments *of the form* $x_1 := e_1, \ldots, x_n := e_n$, *where the* x_i *are pairwise distinct.*
- $l_0 \in L$ *is the* start location.
- η_0 *is the set of* initial assignments *of the form* $x := c$, $\forall x \in X$, *and* $c \in \mathcal{D}$.

In the above definition of a process, we require that a variable that is used in a receive transition should not be assigned in the same transition.

An *interface*, represented by symbol I (possibly subscripted), is a finite set of names called *group names*. Group names are denoted by symbol g (possibly subscripted). We use $\mathcal{I}$ to denote the set of all interfaces. A *node* $P\!:\!I$ denotes a process P with interface I. Henceforth we use $\mathbf{n}$ to denote the set $\{1, \ldots, n\}$, and $P_i, i \in \mathbf{n}$, to denote the process $\langle L_i, X_i, \Sigma, \delta_i, l_{0,i}, \eta_{0,i} \rangle$ over domain $\mathcal{D}$.

Definition 7 (Ad Hoc Network, AHN) *For* $i \in \mathbf{n}$, $P_i = \langle L_i, X_i, \Sigma, \delta_i, l_{0,i}, \eta_{0,i} \rangle$ *s.t.* $X_i \subseteq \mathcal{V}ar$ *are pairwise disjoint, then* $\Pi_{i \in \mathbf{n}} P_i\!:\!I_i$ *is an* AHN.

6.3.2 Concrete Semantics

We provide a labeled transition system (LTS) based semantics for AHNs. An LTS is a 4-tuple $(S, \mathit{Act}, \longrightarrow, s_0)$, where S is a set of states, Act is a set of labels, $\longrightarrow \subseteq S \times \mathit{Act} \times S$ is a ternary relation of labeled transitions, and $s_0 \in S$ is the initial state. A labeled transition $(s, \alpha, t) \in \longrightarrow$, is also represented as $s \xrightarrow{\alpha} t$.

Definition 8 (Semantics of an AHN) *The semantics of an AHN* $\Pi_{i \in \mathbf{n}} P_i : I_i$, *denoted as* $[\![\Pi_{i \in \mathbf{n}} P_i : I_i]\!]$, *is the LTS* $(S, \mathit{Act}, T, s_0)$ *such that:*

- $S = \overline{L} \times \Theta$, *where* $\overline{L} = L_1 \times \ldots \times L_n$, Θ *is the set of all possible environments* $X \mapsto \mathcal{D}$, $X = X_1 \uplus \cdots \uplus X_n$.

- *$Act = \{\mathbf{b}\ v \mid v \in \mathcal{D}\}$.*
- *$\longrightarrow$ is such that $(\bar{l}, \theta) \xrightarrow{\mathbf{b}\ v} (\bar{l}', \theta')$, where $\bar{l} = (l_1, \ldots, l_n)$, $\bar{l}' = (l'_1, \ldots, l'_n)$, $\theta' = [\![\eta]\!]\theta$, $v = [\![e]\!]\theta$ if:*
  - *$\exists i \in \mathbf{n}$: $(l_i, \mathbf{b}\ e, l'_i, \langle \rho_i, \eta_i \rangle) \in \delta_i$, and*
  - *$\mathbf{k} = \{k | (l_k, \mathbf{r}\ (x_k), l'_k, \langle \rho_k, \eta_k \rangle) \in \delta_k, k \in \mathbf{n}, k \neq i, I_i \cap I_k \neq \emptyset\}$, such that :*
    - *$\forall j \in \mathbf{n} \setminus (\mathbf{k} \cup \{i\})$: $l'_j = l_j$*
    - *$\rho = \rho_i \wedge \bigwedge_{k \in \mathbf{k}} \rho_k$, $[\![\rho]\!]\theta$ is* **true**
    - *$\eta = \eta_i \cup \bigcup_{k \in \mathbf{k}} \eta_k[v/x_k] \cup \{x_k := v\}$*
- *$s_0 = (\bar{l}_0, \theta_0)$, where $\bar{l}_0 = \langle l_{0,1}, \ldots, l_{0,n} \rangle$, $\theta_0 = [\![\bigcup_{i \in \mathbf{n}} \eta_{0,i}]\!]\theta_\epsilon$, and θ_ϵ is the empty environment.*

In the description of the transition relation ($\longrightarrow$) in Definition 8, i denotes the index of a process capable of performing a broadcast ($\mathbf{b}\ e$) action, and $\mathbf{k}$ denotes the set of indices of processes that are able to receive a value broadcast by process P_i. Note that processes not participating in the synchronization remain in the same location. For a transition to be enabled, the guards of synchronizing processes must be true. When a transition is taken, the value transmitted by the broadcaster is propagated to all receivers, and the assignments of the participating processes are performed.

6.3.3 Symbolic System Specification

We define a symbolic semantics for AHNs in which process interfaces are treated as variables. For example, for a node $P : I$, I is treated as a variable in contrast to the concrete semantics, where I represents a set of group names. We use $\mathbf{J}$ to denote the set of interface variables and $\mathcal{J}$ (possibly subscripted) to denote elements of $\mathbf{J}$.

Topology Constraint Language. Constraints on process interface variables are given by the following grammar. Symbol Γ represents the constraint language and γ (possibly subscripted) represents elements of Γ.

$$\Gamma \quad ::= \quad \mathbf{true} \ \mid \ \mathbf{false} \ \mid \ \mathit{conn}(\mathbf{J}, \mathbf{J}) \ \mid \ \mathit{dconn}(\mathbf{J}, \mathbf{J}) \ \mid \ \Gamma \wedge \Gamma$$

A valuation $\vartheta : \mathbf{J} \rightarrow \mathcal{I}$ maps an interface variable $\mathcal{J}$ to an interface I. A valuation ϑ is a model of a constraint γ, written as $\vartheta \models \gamma$, defined as follows:

$\vartheta \models$ **true**

$\vartheta \not\models$ **false**

$\vartheta \models conn(\mathcal{J}_1, \mathcal{J}_2) \quad \text{if} \quad \vartheta(\mathcal{J}_1) \cap \vartheta(\mathcal{J}_2) \neq \emptyset$

$\vartheta \models dconn(\mathcal{J}_1, \mathcal{J}_2) \quad \text{if} \quad \vartheta(\mathcal{J}_1) \cap \vartheta(\mathcal{J}_2) = \emptyset$

$\vartheta \models \Gamma_1 \wedge \Gamma_2 \quad \text{if} \quad \vartheta \models \Gamma_1 \ \wedge \ \vartheta \models \Gamma_2$

A constraint of the form $conn(\mathcal{J}_1, \mathcal{J}_2)$ requires that nodes with interface variables $\mathcal{J}_1$ and $\mathcal{J}_2$ be connected, enabling them to communicate with each other. Constraint $dconn(\mathcal{J}_1, \mathcal{J}_2)$ requires nodes with interface variables $\mathcal{J}_1$ and $\mathcal{J}_2$ to be disconnected. A constraint γ is *satisfiable*, if there exists an interface valuation ϑ that assigns each interface variable in γ a value (set of group names) such that $\vartheta \models \gamma$. Two constraints γ_1 and γ_2 are *equivalent ($\equiv$)* if for every valuation ϑ s.t. $\vartheta \models \gamma_1$, it holds that $\vartheta \models \gamma_2$, and vice-versa.

Proposition 15 *Satisfiability of topology constraints is decidable.*

Proof Sketch: The following procedure determines the satisfiability of conjunction of primitive constraints over interface variables, and returns a satisfying assignment if there exists one.

Consider a constraint γ over interface variables $\mathcal{J}_1, \ldots, \mathcal{J}_n$.

- Step 1: For every constraint of the form $conn(\mathcal{J}_i, \mathcal{J}_j)$, add a fresh name g_{ij} to $\mathcal{J}_i$ and $\mathcal{J}_j$ (so that $\mathcal{J}_i \cap \mathcal{J}_j \neq \emptyset$).
- Step 2: For every $\mathcal{J}_i$ that is not assigned a value in Step 1, initialize $\mathcal{J}_i$ to singleton set $\{g_i\}$, such that g_i has not been assigned to any interface variable in Step 1.
- Step 3: For every constraint of the form $dconn(\mathcal{J}_i, \mathcal{J}_j)$, if $\mathcal{J}_i \cap \mathcal{J}_j = \emptyset$, then constraint γ is satisfiable, otherwise γ is unsatisfiable.

This procedure terminates and if γ is satisfiable, returns one satisfying assignment.

□

For example, solution to the constraint $conn(\mathcal{J}_1, \mathcal{J}_2) \wedge conn(\mathcal{J}_1, \mathcal{J}_4) \wedge conn(\mathcal{J}_2, \mathcal{J}_3) \wedge conn(\mathcal{J}_3, \mathcal{J}_4)$, is $\mathcal{J}_1 = \{g_{1,2}, g_{1,4}\}, \mathcal{J}_2 = \{g_{1,2}, g_{2,3}\}, \mathcal{J}_3 = \{g_{2,3}, g_{3,4}\}, \mathcal{J}_4 = \{g_{1,4}, g_{3,4}\}$.

A symbolic AHN is an AHN for which topology is represented using interface variables.

Definition 9 (Symbolic AHN) *For $i \in \mathbf{n}$, $P_i = \langle L_i, X_i, \Sigma, \delta_i, l_{0,i}, \eta_{0,i} \rangle$ s.t. $X_i \subseteq \mathcal{V}ar$ are pairwise disjoint, then $\Pi_{i\in\mathbf{n}} P_i : \mathcal{J}_i$ is a* symbolic AHN.

Definition 10 (Semantics of a symbolic AHN) *The semantics of a symbolic AHN $\Pi_{i\in\mathbf{n}} P_i : \mathcal{J}_i$, denoted as $[\![\Pi_{i\in\mathbf{n}} P_i : \mathcal{J}_i]\!]$, is the symbolic LTS (S, Act, T, s_0), such that:*

- $S = \overline{L} \times \Theta \times \Gamma$, *where* $\overline{L} = L_1 \times \ldots \times L_n$, Θ *is the set of all possible environments* $X \mapsto \mathcal{D}$, $X = X_1 \uplus \cdots \uplus X_n$.
- $Act = \{\mathbf{b}\ v \mid v \in \mathcal{D}\}$.
- $\rightsquigarrow$ *is such that* $(\bar{l}, \theta, \gamma) \overset{\mathbf{b}\ v}{\rightsquigarrow} (\bar{l}', \theta', \gamma')$, *where* $\bar{l} = (l_1, \ldots, l_n)$, $\bar{l}' = (l'_1, \ldots, l'_n)$, $\theta' = [\![\eta]\!]\theta$, $v = [\![e]\!]\theta$ *if:*
  - $\exists i \in \mathbf{n}$: $(l_i, \mathbf{b}\ e, l'_i, \langle \rho_i, \eta_i \rangle) \in \delta_i$, *and*
  - $\mathbf{k} = \{k | (l_k, \mathbf{r}\ (x_k), l'_k, \langle \rho_k, \eta_k \rangle) \in \delta_k, k \in \mathbf{n}, k \neq i\}$, $\exists \mathbf{k_c}, \mathbf{k_d} : \mathbf{k} = \mathbf{k_c} \uplus \mathbf{k_d}$ *such that:*
    - $\forall j \in \mathbf{n} \setminus (\mathbf{k_c} \cup \{i\})$: $l'_j = l_j$
    - $\rho = \rho_i \wedge \bigwedge_{k\in\mathbf{k_c}} \rho_k$, $[\![\rho]\!]\theta$ *is* **true**
    - $\eta = \eta_i \cup \bigcup_{k\in\mathbf{k_c}} \eta_k[v/x_k] \cup \{x_k := v\}$
    - $\gamma' = \gamma \wedge \bigwedge_{k\in\mathbf{k_c}} conn(\mathcal{J}_i, \mathcal{J}_k) \wedge \bigwedge_{k\in\mathbf{k_d}} dconn(\mathcal{J}_i, \mathcal{J}_k)$ *is satisfiable*
- $s_0 = (\bar{l}_0, \theta_0, \mathbf{true})$, *where* $\bar{l}_0 = \langle l_{0,1}, \ldots, l_{0,n} \rangle$, $\theta_0 = [\![\bigcup_{i\in\mathbf{n}} \eta_{0,i}]\!]\theta_\epsilon$, *and* θ_ϵ *is the empty environment.*

In the clause for transition relation ($\rightsquigarrow$) in Definition 10, i denotes the index of a process enabled to do a broadcast ($\mathbf{b}\ e$) action, and $\mathbf{k}$ denotes the set of indices of processes that are enabled to perform a receive action. $\mathbf{k_c}$ and $\mathbf{k_d}$ form a partition of $\mathbf{k}$ such that $\mathbf{k_c}$ is the set of indices of processes that synchronize with the P_i; thus *conn* constraint is generated for processes in $\mathbf{k_c}$. Processes with indices in $\mathbf{k_d}$ do not synchronize with broadcast action of P_i, and thus are not connected to P_i, and *dconn* constraint is generated for the transition. Note that, as in the concrete semantics, processes not involved in the synchronization remain in their locations. The guards and assignments are treated exactly as in the concrete semantics, considering only the synchronizing processes.

Theorem 16 (Correspondence) *The symbolic semantics is sound and complete w.r.t. the concrete semantics; i.e.* $(s, \gamma) \overset{\alpha}{\rightsquigarrow} (s', \gamma')$ *in* $[\![\Pi_{i\in\mathbf{n}} P_i : \mathcal{J}_i]\!]$ *iff* $\forall$ *interface valuations* ϑ *s.t.* $\vartheta \models \gamma'$, $s \xrightarrow{\alpha} s'$ *in* $[\![\Pi_{i\in\mathbf{n}} P_i : \vartheta(\mathcal{J}_i)]\!]$.

Proof Sketch:

- *Soundness:* Consider a symbolic transition $(s, \gamma) \overset{\alpha}{\rightsquigarrow} (s', \gamma')$ in $\Pi_{i\in\mathbf{n}} P_i : \mathcal{J}_i$. From the semantics of the symbolic transitions, $\gamma' \implies \gamma$. For all ϑ s.t. $\vartheta \models \gamma'$ (also $\vartheta \models \gamma$), there exists a concrete transition $s \xrightarrow{\alpha} s'$ in $\Pi_{i\in\mathbf{n}} P_i : \vartheta(\mathcal{J}_i)$.

- *Completeness:* Consider a concrete transition $s \xrightarrow{\alpha} s'$ in $\Pi_{i\in\mathbf{n}} P_i : I_i$. Let ϑ be an interface valuation, γ' be a constraint, and for $i \in \mathbf{n}$, $\mathcal{J}_i$ be interface variables, such that $\vartheta(\mathcal{J}_i) = I_i$, and $\vartheta \models \gamma'$. Then $\exists \gamma : \gamma \implies \gamma'$, and $(s, \gamma) \overset{\alpha}{\rightsquigarrow} (s', \gamma')$ in $\Pi_{i\in\mathbf{n}} P_i : \mathcal{J}_i$. $\square$

6.4 Constraint-Based Verification

6.4.1 Verification of Reachability Properties

We first consider verification of symbolic AHNs for reachability properties, which is done by constructing and traversing the symbolic transition system.

Definition 11 (Reachability) *For an AHN* $A_C = \Pi_{i\in\mathbf{n}} P_i : I_i$, *the set of states reachable from a state* s *in* $[\![A_C]\!]$, *denoted by* $Reach_C(s, A_C)$, *is the smallest set such that* $s \in Reach_C(s, A_C)$ *and for every* $s' \in Reach_C(s, A_C)$ *and for every* $\alpha \in Act$ *if* $s' \xrightarrow{\alpha} s'' \in [\![A_C]\!]$ *then* $s'' \in Reach_C(s, A_C)$

For a symbolic AHN $A_S = \Pi_{i\in\mathbf{n}} P_i : \mathcal{J}_i$, *the set of states reachable from a symbolic state* (s, γ) *in the* $[\![A_S]\!]$, *denoted by* $Reach_S((s, \gamma), A_S)$, *is the smallest set such that* $(s, \gamma) \in Reach_S((s, \gamma), A_S)$, *and for every* $(s', \gamma') \in Reach_S((s, \gamma), A_S)$ *and for every* $\alpha \in Act$ *if* $(s', \gamma') \overset{\alpha}{\rightsquigarrow} (s'', \gamma'')$ *then* $(s'', \gamma'') \in Reach_S((s, \gamma), A_S)$.

```
      Algorithm **SymReach**
      **Input :** predicate $p$ ; symbolic AHN $A_S$; initial symbolic state $(s_0, \gamma_0)$
      **Output :** $CS$ the set of most general constraints in states that satisfy $p$ and
              are reachable from initial state $(s_0, \gamma_0)$
1.    $R := \{(s_0, \gamma_0)\}$
2.    $CS := \left\{ \begin{array}{ll} \{\gamma_0\} & if (s_0, \gamma_0) \models p \\ \emptyset & otherwise \end{array} \right\}$
3.    $WS := \{(s_0, \gamma_0)\}$ // working set (FIFO queue)
4.      while ( $WS \neq \emptyset$)
5.        let $(s, \gamma) \in WS$
6.        $WS := WS \setminus (s, \gamma)$
7.        for each transition $(s, \gamma) \overset{\alpha}{\rightsquigarrow} (s', \gamma')$ in $[\![A_S]\!]$
8.          if $\gamma'$ not subsumed by any constraint in $CS$
9.            if there exists no $(s', \gamma'') \in R$ such that $\gamma' \implies \gamma''$
10.             $WS := WS \cup \{(s', \gamma')\}$
11.             $R := R \cup \{(s', \gamma')\}$
12.             if $(s', \gamma') \models p$
13.               $CS := mg(CS \cup \{\gamma'\})$
14.       return $CS$
```

Figure 12: Symbolic Reachability Algorithm

Satisfaction of a Property. A property over a concrete AHN A_C, denoted by ϕ is either a proposition, defined over the states of A_C, or of the form EFp, where p is a proposition. We use $s \models \phi$ to denote satisfaction of property ϕ in state s. We say that $s \models EFp$ if there is some state s' reachable from s such that $s' \models p$. The notion of satisfaction of a property is lifted to symbolic states, denoted as $(s, \gamma) \models \phi$, if γ is satisfiable, and ϕ is true in s in every topology ϑ such that $\vartheta \models \gamma$. The following proposition establishes that when verifying a reachability property for a symbolic AHN, it is sufficient to examine a subset of symbolic states. In particular, once (s, γ) is visited and $(s, \gamma) \models \phi$, all states (s, γ') such that $\gamma' \implies \gamma$ can be discarded from consideration.

Proposition 17 *For a given symbolic state (s_0, γ_0), symbolic AHN A_S, and property ϕ, if $\exists (s, \gamma) \in Reach_S((s_0, \gamma_0), A_S)$ s.t. $(s, \gamma) \models \phi$, then $\forall (s, \gamma') \in Reach_S((s_0, \gamma_0), A_S)$ s.t. $\gamma' \implies \gamma$, $(s, \gamma') \models \phi$.*

Algorithm **SymReach** (Fig. 12) uses Prop. 17 to prune the search space for proving reachability properties. For a given predicate p, a symbolic AHN and a start state

(s_0, γ_0) in the AHN, Algorithm **SymReach** returns the set of most general constraints CS such that for all $\gamma \in CS$ $(s_0, \gamma) \models EFp$. The set of reachable states are stored in R and a working set WS is used to store unvisited states (Line 3) during a breadth-first traversal of the transition system. At the beginning of each iteration (Line 4) states in $R - WS$ have been completely explored. Since each transition only adds to the topology constraints, we discard symbolic states whose topologies are already known to satisfy the reachability property (Line 8). Line 9 uses Prop. 17 to prune the search space. In Line 13, mg chooses the most general set of constraints from a given set of constraints. Algorithm **SymReach** returns the CS set upon termination. It is easily shown that for a finite-state AHN Algorithm **SymReach** terminates.

The following theorem formally states the correctness of the algorithm: that the set of topology constraints computed by **SymReach** exactly covers the topology constraints in $Reach_S$ (Def. 11).

Theorem 18 (Correctness) *Let* $CS' = \{\gamma \mid (s, \gamma) \in Reach_S((s_0, \gamma_0), A_S),\ (s, \gamma) \models \phi\}$ *be the set of all constraints that are part of the reachable symbolic states* (s, γ) *for which* ϕ *holds. Let* CS *be the set returned by Algorithm* **SymReach** *(Figure 12). Then* $\forall \gamma' \in CS'\ \exists \gamma \in CS : \gamma' \Longrightarrow \gamma$*, and* $\forall \gamma \in CS\ \exists \gamma' \in CS' : \gamma \equiv \gamma'$*.*

The choice of breadth-first search (BFS) in Algorithm **SymReach** is important for the following two reasons. First, subsumption-based pruning of search space is more effective with BFS because general constraints are visited before more specific constraints. Secondly, the use of BFS makes it easy to show the tight bound on the total number of symbolic transitions, used in the complexity analysis.

6.4.2 Complexity Analysis for the SymReach Algorithm

Consider a concrete AHN A_C with n nodes. Let the total number of states in A_C be $|S|$, and the total number of transitions in A_C be $|T| = O(|S|^2)$. The time for reachability analysis from a given initial state in A_C is bounded by the number of transitions and is equal to $|T| = O(|S|^2)$. The total number of topologies for an n-node AHN is $O(2^{n^2})$. Therefore, the time complexity for exploring states reachable from a given state in all n-node AHNs (all possible topologies) is $O(2^{n^2}) \times |T| = O(2^{n^2}|S|^2)$.

Let $A_S = \Pi_{i \in \mathbf{n}} P_i : \mathcal{J}_i$ be a symbolic AHN and $\mathcal{A_C}$ the set of all concrete AHNs $A_{C_j} = \Pi_{i \in \mathbf{n}} P_i : I_{i,j}$, where index j indicates one of the $O(2^{n^2})$ possible topologies for an n-node network. Recall that each state of A_S is of the form (s, γ), where s is a location-environment pair, and γ is a topology constraint. Let $|S|$ be the largest number of states of any concrete AHN $A_C \in \mathcal{A_C}$. Since the number of distinct γ's is $O(2^{n^2})$, the total number of symbolic states is bounded by $O(2^{n^2}|S|)$.

The number of symbolic transitions is bounded by the total number of concrete transitions for all possible topologies. We can establish this bound by defining a 1-1 mapping between symbolic transitions from a symbolic state (s, γ) in A_S to a transition from concrete state s in $\mathcal{A_C}$. Consider associating each state in R and WS with an index which is the length of the shortest path from the initial state to (s, γ). Now, let (s, γ) be the selected state with index i at some iteration of the algorithm. There is no state (s, γ') in $R - WS$ (i.e. visited state) such that $\gamma \implies \gamma'$ (due to the use of subsumption, line 9 of the algorithm). First consider the case when there is no other state (s, γ') in R with index i. It follows from Theorem 16 that for every concrete topology that satisfies γ, state s is reachable in i or fewer steps. In fact, there is a concrete topology $\vartheta \models \gamma$ for which the shortest path to reach s is of length i. The symbolic transition that placed (s, γ) in WS can then be mapped to the corresponding concrete transition in the topology given by ϑ. Now consider the case when there is another state (s, γ') in R with index i. If (s, γ) and (s, γ') can be reached using a single transition from a common state, say (s'', γ''), then the symbolic transition that placed (s, γ) in WS can then be mapped to the corresponding concrete transition in a topology that satisfies $\gamma \wedge \neg\gamma'$. Otherwise, (s, γ) and (s, γ') descend from two distinct states, both of which have the same index. We can then associate with the symbolic transition to (s, γ) the same concrete instance ϑ used to map the transition to its parent (and similarly with (s, γ')).

We now show that reachability computation over symbolic state space takes no additional time, in the asymptotic sense, than reachability over concrete state spaces. The main additional cost of symbolic reachability algorithm is constraint subsumption (line 9 of the algorithm). We can do this operation in amortized constant time, as follows. First, consider computing and storing the subsumption lattice for the constraints *a priori*. The construction cost of this lattice is $O(2^{n^2})$ but is paid only once. We can associate a set, initially empty, with each constraint in the lattice. To determine whether (s, γ) should be added to R, we check if s is in the set associated with γ in the lattice. This check can be done in constant time. When (s, γ) is added to R, we add s to the sets associated with constraints more specific than γ. This operation may take $O(2^{n^2})$ in the worst case, but note that an element s may be added to the set associated with γ at most once, and hence maintaining this data structure incurs a total cost of $O(2^{n^2}|S|)$ over the entire run of the algorithm. Hence symbolic reachability can be done in $O(2^{n^2}|S|^2)$, the same complexity as that of the concrete algorithm.

The space complexity is bounded by the size of the set of reachable states, R. The number of elements of this set is $2^{n^2}|S|$. The size of each element is $O(n^2)$ due to the size of the topology constraint, but this factor gets down-played in the asymptotic case. Hence the asymptotic space complexity for the symbolic algorithm is $O(2^{n^2}|S|)$.

6.4.3 Model Checking Symbolic AHNs

The symbolic transition system can be readily used for checking LTL properties of AHNs. We can use the standard procedure of constructing the product between a Büchi automaton (corresponding to the negation of a given LTL property) and the symbolic transition system and look for reachable accepting cycles in the product graph. Note that for every symbolic transition of the form $(s, \gamma) \rightsquigarrow (s', \gamma')$, it holds that $\gamma' \implies \gamma$. Hence it follows that if (s, γ) and (s, γ') are two states in a cycle, then $\gamma \equiv \gamma'$. Hence the constraint component of states in a cycle are all equivalent. Let $(s_1, \gamma), (s_2, \gamma), \ldots, (s_n, \gamma)$ be states in an accepting cycle such that $(s_i, \gamma) \rightsquigarrow (s_{i+1}, \gamma)$ for $1 \leq i < n$, and $(s_n, \gamma) \rightsquigarrow (s_1, \gamma)$. It follows from Theorem 16 that for every concrete topology ϑ such that $\vartheta \models \gamma$, the states $s_1, s_2, \ldots, s_n$ will be in an accepting cycle. Hence reachable good cycles in the symbolic case mean that there are reachable good cycles in the concrete case. This forms the basis for LTL model checking of symbolic AHNs.

Model checking of other temporal logics such as CTL and CTL* can be performed over symbolic AHNs by using the standard algorithms over the symbolic transition system. From the complexity results for reachability checking, it follows that model checking for symbolic AHNs can be done in time and space comparable to the total time and space for model checking of concrete AHNs for all topologies.

6.5 Verification of the LMAC Protocol

We built a prototype implementation of **SymReach** in the XSB logic programming system [XSB]. XSB adds the capability of memoizing inferences to a traditional Prolog-based system, which simplifies the implementation of fixed point algorithms such as **SymReach**. Below we present the results of verifying the LMAC protocol [vHH04], a medium access control protocol for wireless sensor networks, using this prototype.

LMAC protocol for Wireless Sensor Networks
The LMAC protocol aims to allocate each node in the sensor network a time slot during which the node can transmit without collisions. Note that for collision freedom, direct (one-hop) neighbors as well as two-hop neighbors must have pairwise different slots. The protocol works by nondeterministically assigning slots, and resolving any collisions that result from this assignment. We apply our query-based verification technique to this protocol to compute the set of topologies for which there are undetected and hence unresolved collisions.

Protocol Description [vHH04]. In schedule-based MAC protocols, time is divided into slots, which are grouped into fixed length frames. Every node is allocated one time slot in which it can carry out its transmission in a frame without causing collision or interference with other transmissions. Each node broadcasts a set of time slots occupied by its (one-hop) neighbors and itself. When a node receives a message from a neighbor it marks the respective time slot as occupied. The four phases of the LMAC protocol involved in allocating time slots to nodes are as follows. ***Initialization phase:*** a node listens on the wireless medium to detect other nodes. On listening from a neighboring node, the node synchronizes by learning the current slot number and transitions to the wait phase. ***Wait phase:*** a node waits for a random period of time and then continues with the discover phase. ***Discover phase:*** a node listens to its one-hop neighbors during one entire frame and records the time slots occupied by them and their neighbors. On gathering information regarding the occupied time slots, the node randomly chooses a time slot from the available ones (time slots that do not interfere in its one-hop and two-hop neighborhood), and advances to the active phase. ***Active phase:*** a node transmits a message in its own time slot and listens during other time slots. When a neighboring node informs that there was a collision in the time slot of the node, the node transitions to the wait phase to discover a new time slot for itself. Collisions can occur when two or more one-hop or two-hop neighboring nodes choose the same time slot for transmission. Nodes causing a collision cannot detect the collision themselves, they need to be informed by their neighboring nodes about the collision. When a node detects a collision it transmits information about the collision in its time slot.

Modeling the LMAC protocol in our framework. Our encoding of the LMAC protocol in our framework follows the encoding used in [FvHM07]. We carry over the underlying assumption in the LMAC protocol, that the local clocks of nodes are synchronous. Since there is no support for modeling time in our prototype framework, we define a special *timer* node that informs other nodes about the end of a time slot by broadcasting an *end of slot* message. Nodes update their local information at the end of every time slot.

An encoding of a process in an AHN model of LMAC is presented in Fig. 13. At the beginning, we assume that one distinguished node is "active" (i.e. in `active` location) and the rest are "passive" (i.e. in `init` location). Note that the figure gives the definition of a passive node; the definition of the active node is identical except for its initial state. The (symbolic) system specification for a 3-node network is shown below.

$$\mathcal{A} = \mathit{timer} : \mathcal{J}_1 \mid \mathit{active\_node} : \mathcal{J}_2 \mid \mathit{passive\_node} : \mathcal{J}_3 \mid \mathit{passive\_node} : \mathcal{J}_4$$

Transitions in Fig. 13 are specified in the form [*label*] l & $\rho \rightarrow l'$ & η, where *label* is the label of the transition, l and l' are the source and destination locations, ρ is the (optional) guard and η is the set of simultaneous assignments. We use the standard notation of *primed* variables to denote variables in the destination state. We use "epsilon" transitions (denoted by action label `[ ]` in the figure) to simplify the encoding. We can derive the epsilon-free description (as in the formal definition

Passive LMAC Process : $< L, X, \Sigma, \delta, l_0, \eta_0 >$
$L = \{init, init1, init2, listening0, recOne0, done0, choice0, choice, active, sent, listening, recOne, recTwo, collision\_detected\}$
$X = \{Current, RecVec, Counter, SlotNo, First, Second, Col, Collision\}$
$\Sigma = \{\mathbf{r}\ (msg(Sslot, Scollision, Sfirst)), \mathbf{r}\ (eos), \mathbf{b}\ msg(slot, collision, first)\}$
$l_0 = init$
$\eta_0 = \{Current := -1, RecVec := \emptyset, Counter := 0, SlotNo := -1, First := \emptyset, Second := \emptyset, Col := -1, Collision := -1\}$
Transitions $(l, \alpha, l', \langle \rho, \eta \rangle) \in \delta$ are given below:

Init
$[\mathbf{r}\ (msg(Sslot, \_, \_))]\ \ init \ \rightarrow\ init1\ \ \&\ \ Current' := Sslot$
$[\mathbf{r}\ (eos)]\ \ init1\ \rightarrow\ listening0\ \ \&\ \ Current' := (Current\ +\ 1)\%frame,\ Counter' := 0$
$[\mathbf{r}\ (msg(\_, \_, \_))]\ \ init1\ \rightarrow\ init2$
$[\mathbf{r}\ (eos)]\ \ init2\ \rightarrow\ init$

Discover
$[\mathbf{r}\ (msg(\_, \_, Sfirst))]\ \ listening0\ \rightarrow\ recOne0\ \ \&\ \ RecVec' := Sfirst,$
$First' := \{Current\} \cup First$
$[\mathbf{r}\ (msg(\_, \_, \_))]\ \ recOne0\ \rightarrow\ done0\ \ \&\ \ if\ Collision < 0\ then\ Collision' := Current,$
$RecVec' := \emptyset$
$[\mathbf{r}\ (eos)]\ \ done0\ \rightarrow\ choice0\ \ \&\ \ Current' := (Current + 1)\%frame$
$[\mathbf{r}\ (eos)]\ \ recOne0\ \rightarrow\ choice0\ \ \&\ \ Current' := (Current + 1)\%frame,$
$Second' := RecVec \cup Second,\ RecVec' := \emptyset$
$[\mathbf{r}\ (eos)]\ \ listening0\ \rightarrow\ choice0\ \ \&\ \ Current' := (Current + 1)\%frame$
$[\ ]\ \ choice0\ \ \&\ \ Counter < frame - 1\ \rightarrow\ listening0\ \ \&\ \ Counter' := Counter + 1$
$[\ ]\ \ choice0\ \ \&\ \ Counter >= frame - 1\ \rightarrow\ choice\ \ \&\ \ Second' := First \cup Second$

Choice
$[\ ]\ \ choice\ \ \&\ \ Second \neq AllSlots\ \rightarrow\ active\ \ \&\ \ SlotNo' \in AllSlots \setminus Second,$
$Second' := \emptyset$
$[\ ]\ \ choice\ \ \&\ \ Second = AllSlots\ \rightarrow\ listening0\ \ \&\ \ Counter' := -1,\ Collision' := -1,$
$First' := \emptyset,\ Second' := \emptyset$

Active
$[\mathbf{b}\ msg(SlotNo, Collision, First)]\ active\ \&\ Current{=}SlotNo \rightarrow sent\ \&\ Collision' := -1$
$[\ ]\ \ active\ \ \&\ \ Current \neq SlotNo\ \rightarrow\ listening$

Send
$[\mathbf{r}\ (eos)]\ \ sent\ \rightarrow\ active\ \ \&\ \ Current' := (Current + 1)\%frame$

Listen
$[\mathbf{r}\ (msg(\_, Scollision, \_))]\ \ listening\ \rightarrow\ recOne\ \ \&\ \ Col' := Scollision,$
$First' := Current \cup First$
$[\mathbf{r}\ (eos)]\ \ listening\ \rightarrow\ active\ \ \&\ \ Current' := (Current + 1)\%frame$
$[\mathbf{r}\ (msg(\_, \_, \_))]\ \ recOne\ \rightarrow\ recTwo\ \ \&\ \ if\ Collision' < 0\ then\ Collision' := Current$
$[\mathbf{r}\ (eos)]\ \ recTwo\ \rightarrow\ active\ \ \&\ \ Current' := (Current + 1)\%frame$
$[\mathbf{r}\ (eos)]\ \ recOne\ \ \&\ \ Col \neq SlotNo\ \rightarrow\ active\ \ \&\ \ Current' := (Current + 1)\%frame$

Collision Reported
$[\mathbf{r}\ (eos)]\ recOne\ \&\ Col = SlotNo \rightarrow collision\_detected\ \&\ First' := \emptyset,\ RecVec' := \emptyset$
$Current' := (Current + 1)\%frame,$
$Counter' := 0,\ SlotNo' := -1,$
$Col' := -1,\ Collision' := -1$
$[\ ]\ \ collision\_detected\ \rightarrow\ listening0$

Figure 13: LMAC protocol model.

of AHNs, Defn. 6) using standard automata construction techniques. In our model of LMAC, locations *init*, *init1* and *init2* correspond to the `initialization` phase; locations *listening0*, *recOne0*, *done0*, *choice0* and *choice* to the `discover` phase; and locations *active*, *sent*, *listening*, *recOne*, *recTwo*, and *collision_detected* to the `active` phase. It should be noted that the *wait* phase of the protocol is not modeled, since its function is to only separate the initialization and discover phases by an arbitrary period of time.

The length of a time frame i.e. number of slots (= 5 for 5-node network) is represented by *frame*, and *AllSlots* denotes the set of all time slots. The state variables of a node are: *Current* (the current slot number w.r.t. the beginning of a frame), *RecVec* (auxiliary set to record the slots occupied by one-hop and two-hop neighbors), *Counter* (used to count the number of slots seen by the node in a frame), *SlotNo* (slot number of the node), *First* (set of slots occupied by one-hop neighbors of the node), *Second* (set of slots occupied by two-hop neighbors of the node), *Col* (collision slot reported by another node), *Collision* (slot in which the node detects a collision). The parameters of messages (*msg*) exchanged between nodes are: *Slot*, *Collision*, and *First* variables of the sender node.

Analysis of the LMAC protocol. The property "every collision is eventually detected" can be encoded in LTL as $G(\mathit{collision} \Rightarrow F\,\mathit{collision\_detected})$, where *collision* and *collision_detected* are propositions that are true in states where collision and collision detection occur, respectively. Although LTL model checking of symbolic AHNs can be done as outlined in 6.4, our current prototype implementation supports only reachability checking. We hence checked a related property "there is a detected collision" ($EF\,\mathit{collision\_detected}$). Let CS be the set of all topology constraints computed using algorithm **SymReach** when checking for reachability of proposition *collision_detected*. Let ϑ be a valuation such that $\vartheta \not\models \gamma$ for any $\gamma \in CS$. Note that in the LMAC protocol, there may be a collision between any two neighboring nodes. If γ does not represent a fully disconnected topology, then we can conclude that there is an undetected collision in γ. Hence, by checking for reachability of proposition *collision_detected*, we can compute (a subset of) topologies which have undetected collision. Moreover, using this method is sound: if there is an undetected collision in some topology, we will find at least one representative.

Nodes	# Topologies Symbolic/Concrete	# States	# Transitions	CPU Time	Memory (MB)
2	1/2	36	36	0.08 sec	2.42
3	5/8	110	123	0.24 sec	2.46
4	25/64	458	667	3.38 sec	3.05
5	181/1024	2204	5223	69.51 sec	5.09
6	2082/32768	29012	110194	2 hr 51 min 46 sec	49.79

Table 14: Verification statistics for the LMAC protocol for *detected collisions*.

Verification Statistics and Results. We did symbolic reachability checking for 2- to 6-node networks. The performance results are shown in Table 14. The results were obtained on a machine with Intel Xeon 1.7GHz processor and 2Gb memory running Linux 2.6.18, and with XSB Prolog version 3.1. For 2- and 3-node cases there were no collisions. For 4-, 5- and 6-node cases, topologies containing one-hop neighboring (directly connected) node pairs that appeared in a ring in the topology and did not have a common direct neighbor were found to be in collision that remained undetected.

The second column in the table gives two numbers ξ_s/ξ_c, where ξ_s is the number of symbolic topology constraints explored in a reachability run, i.e. the number of distinct γ such that $(s, \gamma) \in R$ as per the algorithm in Fig. 12; and ξ_c is the total number of possible concrete topologies. Observe that for the 6-node case the number of symbolic topology constraints examined is smaller than the number of concrete topologies by a factor of more than 5. It should also be noted that the same property was verified for a 5-node network in [FvHM07] by using 61 separate verification runs, one for each unique (modulo isomorphism) concrete topology. In contrast, we verified a related property using a single symbolic reachability run.

The third and fourth columns in Table 14 give the number of symbolic states and transitions explored, respectively; and the last two columns give the CPU time and total memory used. Observe that the performance of our prototype implementation is efficient enough to be used for topologies of reasonable size (e.g. 6 nodes). It should be noted that our technique and its implementation does not exploit the symmetry inherent in the problem by identifying isomorphic topologies. At a high level, symmetry reduction can be incorporated by using a check in line 9 of **SymReach** that recognizes constraints representing the same set of topologies modulo isomorphism. Doing so will enable the technique to scale to large network sizes.

6.6 Discussion

We presented an efficient query-based verification technique for ad hoc network protocols. Network topologies are represented symbolically using interface variables, and the model-checking process generates constraints on the topology under which a system specification satisfies a specified property. As such, a term in our constraint language compactly represents a set of concrete topologies that may lead to the satisfaction of the property in question. We demonstrated the practical utility of our approach by considering the verification of a medium access control protocol for sensor networks (LMAC) [vHH04], identifying topologies under which collision may remain unresolved.

The basic data structure for query-based verification is the symbolic transition system, where each state carries with it a topology constraint. If a symbolic state is reachable, then, for every topology satisfying its constraint, the corresponding concrete state is reachable. This structure makes it possible to infer topologies under which reachability properties hold. As described in this chapter, it is also possible to verify properties specified in temporal logics such as LTL over symbolic transition systems, inferring sets of topologies under which the properties hold. Extending our prototype implementation to handle verification with an expressive temporal logic is a topic of future work. There are several avenues for further improving the efficiency of the symbolic verification technique. Some of these are optimizations to common low-level operations, subsumption checks, while others are high-level state-space reductions, e.g. by exploiting symmetries in systems and topologies.

In this work, the focus is on a verification technique and not on the modeling language. We considered ad hoc networks whose topology does not change with time. We deliberately considered only closed systems and chose a simple language that uses interfaces to separate node behavior from network topology as in the ω-calculus. As part of our future work, we plan to extend this work to open systems specified in the ω-calculus, and consider compositional verification in that setting.

Chapter 7

Towards Parameterized Verification of Ad Hoc Network Protocols

In the previous chapter, we presented a technique for verification of AHN protocols for arbitrary instances of topologies. In this chapter, we describe a verification technique for AHNs parameterized on the number of nodes to facilitate verification of AHN protocols for arbitrary number of nodes. As discussed in Section 1.2.3, compositional analysis facilitates parameterized verification of infinite instances of a system (n^{th} instance having n nodes in the system). In [BR06], a parameterized verification technique has been developed for systems specified in CCS process algebra and interpreted over modal μ-calculus formulas. The work in [YBR06] developed parameterized verification technique for systems specified in the π-calculus and interpreted over an extended version of modal μ-calculus formulas, extensions including constructs for representing names and their scopes, and actions. Both of the techniques [BR06, YBR06] developed partial model checkers considering processes as property transformers. The verification problem is reduced to finding a limit to the transformed formulas and deciding the satisfiability of the limit to the formula. Consider a parameterized system P^n consisting of n instances of a process P. In order to verify whether property φ holds in P^n for all n, construct the sequence of properties $\varphi_0, \varphi_1, \ldots$ such that $\varphi_0 = \varphi$ and $\varphi_{i+1} = \Pi(P)(\varphi_i)$ for all $i \geq 0$, where Π denotes a property transformer introduced in Section 1.2.3. Let the sequence converge after k steps: i.e. $\varphi_{k+1} = \varphi_k$. By the definition of Π, it holds that for $n \geq k$, $P^n \models \varphi$ if $P^{n-k} \models \varphi_k$. Let *0* denote the deadlocked process, the unit of the parallel composition operator. Specifically, P^n is equivalent to $P^n|0$. It then follows that $\forall n \geq k$, $P^n \models \varphi$ if $0 \models \varphi_k$, i.e. the zero process has the property specified by the limit of the sequence of formulas.

In [BR06, YBR06], optimization techniques have been developed to determine equivalence of formulas and accelerate the convergence of formula transformation to facilitate practical utility of their partial model checking techniques. The techniques in [BR06, YBR06] have been developed for systems that communicate using binary synchronization. In comparison to these works, the new concerns that arise in the context of AHNs are broadcast-based communication. Compositional analysis of AHNs requires extending similar techniques devised for point-to-point synchronous communication to broadcast (multi-party) synchronization. Following the approach of [BR06, YBR06], we present a compositional analysis technique based on partial model checking, for verification of parameterized AHN protocols. We demonstrate our compositional analysis technique using a fragment of the ω_0-calculus (referred to as ω_m-calculus) without node mobility.

7.1 Property Specification Logic

We present a property specification logic, referred to as the $\omega\mu$-calculus, for specifying properties of systems specified in the ω_m-calculus. The $\omega\mu$-calculus is similar to the $C\mu$-calculus formula logic of [YBR06]. The $C\mu$-calculus extends value-passing μ-calculus with explicit syntactic structures to specify and manipulate local names, parameterized formula variables, modalities with actions that are closed under complementation. Similarly, the $\omega\mu$-calculus extends value-passing μ-calculus with parameterized formula variables and modalities with actions for local broadcast and receive.

The grammar for formula expressions in the $\omega\mu$-calculus is given below:

$$\begin{array}{lll}
\phi & ::= & tt \mid \mathit{ff} \mid x = y \mid x \neq y \mid \phi \vee \phi \mid \phi \wedge \phi \\
 & & \mid \langle A \rangle \phi \mid [A]\phi \mid \langle G(y) \rangle \exists y.\phi \mid \langle G(y) \rangle \forall y.\phi \mid [G(y)]\exists y.\phi \mid [G(y)]\forall y.\phi \\
 & & \mid X(\vec{e}) \mid (\mu X(\vec{z}).\phi)(\vec{e}) \mid (\nu X(\vec{z}).\phi)(\vec{e}) \\
A & ::= & Gy \mid \overline{G}y \mid \overline{G}\{y\} \mid \tau
\end{array}$$

We use Φ to denote the set of all formula expressions. Symbols ϕ, φ and ψ (possibly subscripted) range over formula expressions. Formulas tt and ff stand for propositional constants true and false, respectively. Equality and disequality of names also constitute atomic formulas. Conjunction, disjunction, diamond (existential) and box (universal) modalities and quantifiers can be used to construct formulas. The modal actions $G(y)$, Gy, and τ represent input (broadcast-receive), free input and internal actions, respectively. $\overline{G}y$ is a free output (broadcast-send) action where y is a free name and $\overline{G}\{y\}$ is an output action that has binding occurrence of variable y. In input and output actions $G(y)$ and $\overline{G}\{y\}$, y is bound; in free input and free output actions, all names are free. $\langle G(y) \rangle \exists y.\phi$ and $\langle G(y) \rangle \forall y.\phi$ represent early and late diamond modalities for input action $G(y)$, respectively. $[G(y)]\exists y.\phi$ and $[G(y)]\forall y.\phi$ represent the early and late box modalities for input action $G(y)$, respectively. The intuition behind the choice of the modal actions is to capture all possible behaviors of nodes. $X(\vec{z})$ represents a parameterized formula variable. The least and greatest fixed point formulas are specified as $(\mu X(\vec{z}).\phi)(\vec{e})$ and $(\nu X(\vec{z}).\phi)(\vec{e})$, respectively, where $\vec{z}$ represents formal parameters and $\vec{e}$ represents actual parameters. A formula is often represented as a sequence of fixed point equations. Any property with formula expressions of the form $\sigma X(\vec{z}).\varphi$ can be converted in linear time to set of equations of the form $X(\vec{z}) =_\sigma \varphi$ similarly as described in [YBR06]. For a given $\omega\mu$-calculus formula φ where each fixed point variable has distinct names, the number of equations in the corresponding equational set is equal to the number of fixed point sub-formulas of φ. The formulas in the equational form follow the nesting of the corresponding (sub-)formulas in the property. Each sub-formula of the form $\sigma X.\phi$ is translated to an equation $X =_\sigma \psi$, where ψ is obtained by replacing every occurrence of its sub-formula with the formula variable. For example, the formula expression $\nu X(u).(\mu Y(v).([\overline{\{g_1\}}u]X(u) \wedge [\overline{\{g_2\}}v]Y(v)))$ is translated to $X(u) =_\nu Y(v)$

1. (a) $[\![x = y]\!]\xi\delta = \begin{cases} \{s\delta | s \in S\} & \texttt{if}\, \delta \models x = y \\ \emptyset & \texttt{otherwise} \end{cases}$

   (b) $[\![x \neq y]\!]\xi\delta = \begin{cases} \{s\delta | s \in S\} & \texttt{if}\, \delta \models x \neq y \\ \emptyset & \texttt{otherwise} \end{cases}$

2. $[\![\varphi \vee \psi]\!]\xi\delta = [\![\varphi]\!]\xi\delta \cup [\![\psi]\!]\xi\delta$

3. $[\![\varphi \wedge \psi]\!]\xi\delta = [\![\varphi]\!]\xi\delta \cap [\![\psi]\!]\xi\delta$

4. $[\![\langle\tau\rangle\varphi]\!]\xi\delta = \{s \mid \exists s'.s \xrightarrow{b,\tau} s' \wedge (\delta \models b) \wedge s' \in [\![\varphi]\!]\xi\delta\}$

5. $[\![\langle\overline{G_1}y\rangle\varphi]\!]\xi\delta = \{s \mid \exists s'.s \xrightarrow{b,\overline{G_2}y} s' \wedge (\delta \models b) \wedge (G_1 = G_2) \wedge s' \in [\![\varphi]\!]\xi\delta\}$

6. $[\![\langle\overline{G_1}\{y\}\rangle\varphi]\!]\xi\delta = \{s \mid \exists s'.s \xrightarrow{b,\overline{G_2}x} s' \wedge (\delta \models b) \wedge (G_1 = G_2) \wedge s' \in [\![\varphi\{x/y\}]\!]\xi\delta\}$

7. $[\![\langle G_1 y\rangle\varphi]\!]\xi\delta = \{s \mid \exists s'.s \xrightarrow{b,G_2(x)} s' \wedge (\delta \models b) \wedge (G_1 \cap G_2 \neq \emptyset) \wedge s'\{y/x\} \in [\![\varphi]\!]\xi\delta\}$

8. $[\![\langle G_1(y)\rangle\exists y.\varphi]\!]\xi\delta = \{s \mid \exists s'.s \xrightarrow{b,G_2(x)} s' \wedge (\delta \models b) \wedge (G_1 \cap G_2 \neq \emptyset) \wedge \exists v.s'\{v/x\} \in [\![\varphi\{v/y\}]\!]\xi\delta\}$

9. $[\![\langle G_1(y)\rangle\forall y.\varphi]\!]\xi\delta = \{s \mid \exists s'.s \xrightarrow{b,G_2(x)} s' \wedge (\delta \models b) \wedge (G_1 \cap G_2 \neq \emptyset) \wedge \forall v.s'\{v/x\} \in [\![\varphi\{v/y\}]\!]\xi\delta\}$

10. $[\![X(\overrightarrow{e})]\!]\xi\delta = \xi(X)(\overrightarrow{e}\delta)$

11. $[\![(\mu X(\overrightarrow{z}).\varphi)(\overrightarrow{e})]\!]\xi\delta = (\cap\{f \mid [\![\varphi]\!](\xi \circ \{X \mapsto f\}) \subseteq f\})\delta[\overrightarrow{e}/\overrightarrow{z}]$

12. $[\![(\nu X(\overrightarrow{z}).\varphi)(\overrightarrow{e})]\!]\xi\delta = (\cup\{f \mid f \subseteq [\![\varphi]\!](\xi \circ \{X \mapsto f\})\})\delta[\overrightarrow{e}/\overrightarrow{z}]$

Figure 14: Semantics of the property specification logic.

and $Y(v) =_\mu [\overline{\{g_1\}}u]X(u) \wedge [\overline{\{g_2\}}v]Y(v)$ where X is the outer-fixed point variable and Y is the inner one. The use of equational form permits the transformation to be done on a per-equation basis and eliminates the need to keep track of all the sub-formulas of a formula in a non-equational form. We assume that all formulas are *closed*, i.e. all free names in a formula appear in the parameters of the definition.

The formulas are interpreted over the symbolic semantics of the ω_0-calculus defined in Section 3.5, excluding the MOBILITY rule. The semantics of the formula logic is presented in Fig. 14. The semantics relation $[\![\varphi]\!]\xi\delta$ represents the set of symbolic states that satisfy the formula φ under substitution δ over pnames, over which equality

(=) and disequality ($\neq$) constraints on pnames are interpreted, and a function ξ that maps formula variables to sets of symbolic states of the symbolic transition system. The set of states of the symbolic transition system are represented by S. The symbolic transition relation is used as an implicit parameter in the definition. All rules are evaluated w.r.t. the same transition system. Rules 1-3 for equality and disequality constraints on pnames, and conjunction and disjunction of formulas are straightforward. Rules 4-9 provide semantics for the diamond modality. The semantics for the box modality can be obtained by considering it as the dual of the diamond modality. The substitution δ is updated in Rules 11 and 12 to capture the mapping of formal parameters (free names) to actual arguments. We use notation $s \models_\delta \varphi$ to denote $s \in [\![\varphi]\!]\xi\delta$.

7.2 Compositional Model Checker for the ω_m-Calculus

We define a compositional model checker for the ω_m-calculus based on a transformation function $\Pi : \mathbf{N} \to \Phi \to \Phi$ in a manner similar to as done for CCS in [BR06] and for the π-calculus in [YBR06]. Given nodes M and N, a formula $\varphi \in \Phi$, and a set of substitutions δ, the transformation function Π is defined such that

$$M \mid N \mid \mathbf{0} \models_\delta \varphi \Leftrightarrow N \mid \mathbf{0} \models_\delta \Pi(M)(\varphi) \Leftrightarrow \mathbf{0} \models_\delta \Pi(N)(\Pi(M)(\varphi))$$

The transformation function Π, similar to as defined for CCS in [BR06] and for the π-calculus in [YBR06], generates formulas which represent the temporal obligation of the environment of the process(node) used for the transformation. This technique is referred to as *partial model checking*.

The transformation function Π is define for each formula expression as shown in Fig. 15. Rule 5 transforms a parameterized formula variable $X(\overrightarrow{e})$ into a new formula $X_N(\overrightarrow{e_1})$ where $\overrightarrow{e}$ is formed by concatenation of $\overrightarrow{e_1}$ and free names of N. Compositionality of property transformers is represented by rule 9. Rule 10 uses functions ϕ_{+G} and ϕ_{-G} (defined in Fig. 16) for considering the effect of restricted gnames appearing in the node expression that does the transformation. For ϕ_{-G}, there may be cases when $G \setminus G'$ is $\emptyset$. We impose conditions that a modal receive

1. (a) $\Pi(N)(tt) = tt$ (b) $\Pi(N)(\mathit{ff}) = \mathit{ff}$

2. (a) $\Pi(N)(x = y) = \begin{cases} tt & \texttt{if}\, x = y \\ x = y & \texttt{otherwise} \end{cases}$ (b) $\Pi(N)(x \neq y) = \begin{cases} \mathit{ff} & \texttt{if}\, x = y \\ x \neq y & \texttt{otherwise} \end{cases}$

3. $\Pi(N)(\varphi_1 \vee \varphi_2) = \Pi(N)(\varphi_1) \vee \Pi(N)(\varphi_2)$

4. $\Pi(N)(\varphi_1 \wedge \varphi_2) = \Pi(N)(\varphi_1) \wedge \Pi(N)(\varphi_2)$

5. $\Pi(N)(X(\overrightarrow{e})) = X_N(\overrightarrow{e_1})$ $\texttt{where}\, \overrightarrow{e_1} = \overrightarrow{e} + \mathit{fn}(N)$

6. (a) $\Pi(N)(\exists x.\varphi) = \exists x.\Pi(N)(\varphi)$ (b) $\Pi(N)(\forall x.\varphi) = \forall x.\Pi(N)(\varphi)$

7. (a) $\Pi(\mathbf{0})(\varphi) = \varphi$ (b) $\Pi(\mathit{nil}\!:\!G)(\varphi) = \varphi$

8. $\Pi(A(\overrightarrow{x}) : G)(\varphi) = \Pi(P : G)(\varphi)$ $\texttt{where}\, A(\overrightarrow{x}) \overset{def}{=} P$

9. $\Pi(N_1 \mid N_2)(\varphi) = \Pi(N_2)(\Pi(N_1)(\varphi))$

10. $\Pi((\nu g)N)(\varphi) = (\Pi(N\{g'/g\})(\varphi_{+\{g'\}}))_{-\{g'\}}$ $\texttt{where}\, g' \notin n(\varphi)$

11. $\Pi(a.P : G)(\langle\alpha\rangle\varphi) = \langle\alpha\rangle\Pi(a.P : G)(\varphi)$ $\texttt{where}\, bn(\alpha) \cap \mathit{fn}(a.P : G) = \emptyset$

$$\vee \left\{ \begin{array}{ll} \Pi(P : G)(\varphi) & \texttt{if}\, a = \tau \wedge \alpha = \tau \\ \Pi(P : G)(\varphi) & \texttt{if}\, a = \overline{\mathbf{b}}x \wedge \alpha = \overline{G'}x \wedge G = G' \\ \Pi(P : G)(\varphi\{x/y\}) & \texttt{if}\, a = \overline{\mathbf{b}}x \wedge \alpha = \overline{G'}\{y\} \wedge G = G' \\ \Pi(P : G)(\varphi\{x/y\}) & \texttt{if}\, a = \mathbf{r}(x) \wedge \alpha = G'(y) \wedge G \cap G' \neq \emptyset \\ \Pi(P\{y/x\} : G)(\varphi) & \texttt{if}\, a = \mathbf{r}(x) \wedge \alpha = G'y \wedge G \cap G' \neq \emptyset \\ \mathit{ff} & \texttt{otherwise} \end{array} \right\}$$

$$\vee \left\{ \begin{array}{ll} \langle Gx\rangle\Pi(P : G)(\varphi) & \texttt{if}\, a = \overline{\mathbf{b}}x \wedge \alpha = \overline{G'}x \wedge G = G' \\ \langle Gx\rangle\Pi(P : G)(\varphi\{x/y\}) & \texttt{if}\, a = \overline{\mathbf{b}}x \wedge \alpha = \overline{G'}\{y\} \wedge G = G' \\ \langle\overline{G'}y\rangle\Pi(P\{y/x\} : G)(\varphi),\ \texttt{where}\, bn(a) \cap n(\varphi) = \emptyset & \texttt{if}\, a = \mathbf{r}(x) \wedge \alpha = \overline{G'}y \wedge G \cap G' \neq \emptyset \\ \langle\overline{G'}\{y\}\rangle\Pi(P\{y/x\} : G)(\varphi),\ \texttt{where}\, bn(a) \cap n(\varphi) = \emptyset & \texttt{if}\, a = \mathbf{r}(x) \wedge \alpha = \overline{G'}\{y\} \wedge G \cap G' \neq \emptyset \\ \langle G'(x)\rangle\Pi(P : G)(\varphi\{x/y\}),\ \texttt{where}\, bn(a) \cap n(\varphi) = \emptyset & \texttt{if}\, a = \mathbf{r}(x) \wedge \alpha = G'(y) \wedge G \cap G' \neq \emptyset \\ \langle G'y\rangle\Pi(P\{y/x\} : G)(\varphi),\ \texttt{where}\, bn(a) \cap n(\varphi) = \emptyset & \texttt{if}\, a = \mathbf{r}(x) \wedge \alpha = G'y \wedge G \cap G' \neq \emptyset \\ \mathit{ff} & \texttt{otherwise} \end{array} \right\}$$

12. $\Pi((P_1 + P_2)\!:\!G)(\langle\alpha\rangle\varphi) = \langle\alpha\rangle\Pi((P_1 + P_2)\!:\!G)(\varphi) \vee \Pi(P_1 : G)(\langle\alpha\rangle\varphi) \vee \Pi(P_2 : G)(\langle\alpha\rangle\varphi)$

13. $\Pi((P_1 + P_2)\!:\!G)([\alpha]\varphi) = [\alpha]\Pi((P_1 + P_2)\!:\!G)(\varphi) \wedge \Pi(P_1 : G)([\alpha]\varphi) \wedge \Pi(P_2 : G)([\alpha]\varphi)$

14. $\Pi([x = y]P : G)(\varphi) = C \wedge \Pi(P : G)(\varphi)$ $\texttt{where}\, C = \begin{cases} tt & \texttt{if}\, x = y \\ x = y & \texttt{otherwise} \end{cases}$

A. $\Pi(N)(X(\overrightarrow{z}) =_\sigma \varphi \cup E) = \{$
$X_N(\overrightarrow{z_1}) =_\sigma \Pi(N)(\varphi)$ where $((n(\varphi) - \overrightarrow{z}) \cap \mathit{fn}(N) = \emptyset)$ and $\overrightarrow{z_1} = \overrightarrow{z} + \mathit{fn}(N)\}$
$\cup\ \Pi(N)(E) \cup \bigcup\{\Pi(N')(X'(\overrightarrow{z_2}) =_{\sigma'} \varphi')$ s.t. $X'_{N'}(\overrightarrow{z_3})$ is a subformula of
$\Pi(N)(\varphi), \overrightarrow{z_3} = \overrightarrow{z_2} + \mathit{fn}(N')$ and $(n(\varphi') - \overrightarrow{z_2}) \cap \mathit{fn}(N') = \emptyset)\}$

B. $\Pi(N)(\{\}) = (\{\})$

Figure 15: Partial model checker for the ω_m-calculus.

ϕ	ϕ_{+G}	ϕ_{-G}
tt	tt	tt
ff	ff	ff
$x = y$	$x = y$	$x = y$
$x \neq y$	$x \neq y$	$x \neq y$
$\varphi \vee \psi$	$\varphi_{+G} \vee \psi_{+G}$	$\varphi_{-G} \vee \psi_{-G}$
$\varphi \wedge \psi$	$\varphi_{+G} \vee \psi_{+G}$	$\varphi_{-G} \wedge \psi_{-G}$
$\langle G'x \rangle \varphi$	$\langle G'x \rangle \varphi_{+G} \vee \langle G' \cup G\, x \rangle \varphi_{+G}$	$\langle G' \backslash G\ x \rangle \varphi$
$\langle \overline{G'}x \rangle \varphi$	$\langle \overline{G'}x \rangle \varphi_{+G} \vee \langle \overline{G' \cup G}\, x \rangle \varphi_{+G}$	$\langle \overline{G' \backslash G}\, x \rangle \varphi$
$\langle \overline{G'}\{x\} \rangle \varphi$	$\langle \overline{G'}\{x\} \rangle \varphi_{+G} \vee \langle \overline{G' \cup G}\, \{x\} \rangle \varphi_{+G}$	$\langle \overline{G' \backslash G}\, \{x\} \rangle \varphi_{-G}$
$\langle \tau \rangle \varphi$	$\langle \tau \rangle \varphi_{+G} \vee \langle \overline{G}\{x\} \rangle \varphi_{+G}$	$\langle \tau \rangle \varphi_{-G}$
$\langle \mu \rangle \varphi$	$\langle \mu \rangle \varphi_{+G}$	$\langle \mu \rangle \varphi_{-G}$
$[G'x]\varphi$	$[G'x]\varphi_{+G} \wedge [G' \cup G\, x]\varphi_{+G}$	$[G' \backslash G\, x]\varphi_{-G}$
$[\overline{G'}x]\varphi$	$[\overline{G'}x]\varphi_{+G} \wedge [\overline{G' \cup G}\, x]\varphi_{+G}$	$[\overline{G' \backslash G}\, x]\varphi_{-G}$
$[\overline{G'}\{x\}]\varphi$	$[\overline{G'}\{x\}]\varphi_{+G} \wedge [\overline{G' \cup G}\, \{x\}]\varphi_{+G}$	$[\overline{G' \backslash G}\, \{x\}]\varphi_{-G}$
$[\tau]\varphi$	$[\tau]\varphi_{+G} \wedge [\overline{G}\{x\}]\varphi_{+G}$	$[\tau]\varphi_{-G}$
$[\mu]\varphi$	$[\mu]\varphi_{+G}$	$[\mu]\varphi_{-G}$
$\langle G'(y) \rangle \exists y.\varphi$	$\langle G'(y) \rangle \exists y.\varphi_{+G} \vee \langle G' \cup G\, (y) \rangle \exists y.\varphi_{+G}$	$\langle G' \backslash G\, (y) \rangle \exists y.\varphi_{-G}$
$\langle G'(y) \rangle \forall y.\varphi$	$\langle G'(y) \rangle \forall y.\varphi_{+G} \vee \langle G' \cup G\, (y) \rangle \forall y.\varphi_{+G}$	$\langle G' \backslash G\, (y) \rangle \forall y.\varphi_{-G}$
$[G'(y)]\exists y.\varphi$	$[G'(y)]\exists y.\varphi_{+G} \wedge [G' \cup G\, (y)]\exists y.\varphi_{+G}$	$[G' \backslash G\, (y)]\exists y.\varphi_{-G}$
$[G'(y)]\forall y.\varphi$	$[G'(y)]\forall y.\varphi_{+G} \wedge [G' \cup G\, (y)]\forall y.\varphi_{+G}$	$[G' \backslash G\, (y)]\forall y.\varphi_{-G}$
$X(\vec{e})$	$X(\vec{e})$	$X(\vec{e})$
$(\mu X(\vec{z}).\varphi)(\vec{e})$	$(\mu X(\vec{z}).\varphi_{+G})(\vec{e})$	$(\mu X(\vec{z}).\varphi_{-G})(\vec{e})$
$(\nu X(\vec{z}).\varphi)(\vec{e})$	$(\nu X(\vec{z}).\varphi_{+G})(\vec{e})$	$(\nu X(\vec{z}).\varphi_{-G})(\vec{e})$

Figure 16: Definition of ϕ_{+G} and ϕ_{-G}.

action on an empty ($\emptyset$) interface is not satisfied by any node, and a modal broadcast action on an empty ($\emptyset$) interface is satisfied by a node if the node satisfies a modal τ action.

Rule 11 represents transformation of $\langle \alpha \rangle \varphi$ by a node with action-prefixed process ($a.P : G$). There are three cases to be considered in which $a.P : G$ when composed with an environment can satisfy $\langle \varphi \rangle$.

1. The environment takes action α satisfying the modal obligation (first disjunct). The side condition demands that the bound names in α do not bind any free names of $a.P : G$. An alpha-conversion (renaming of all bound names in a formula with fresh names) can be used to ensure the satisfaction of this condition.
2. Node $a.P : G$ satisfies the modal obligation (second disjunct). The action a

matches the modal obligation with appropriate substitution of names applied to the generated formula in case of input (free and bound) and bound output.

3. The environment synchronizes with node $a.P:G$ (third disjunct). Synchronization in the ω_0-calculus happens when a broadcasting node synchronizes with a receiver. The third disjunct in rule 11 (in Fig. 15), corresponds to broadcast synchronization. The first two cases in the third disjunct in rule 11 are indicative of when the transformer node $a.P:G$ matches the broadcast action of the formula and the environment has to perform a receive action. In the third and fourth cases node $a.P:G$ performs a receive action synchronizing with the broadcast action that the environment is obligated to perform. The fifth and sixth cases correspond to the scenario when both $a.P:G$ and the environment both must perform a receive action and synchronize.

Similar to rule 11, a dual rule can be defined for actions with box modality. Rule 12 specifies the transformation of a diamond modal formula using a basic node with choice-process expression. The resulting formula contains disjuncts corresponding to the cases when the environment is left with the obligation to satisfy the modal action, and when the first or the second process is selected for subsequent transformation. Rule 13 is similar to rule 12. Rule 14 defines the transformation of a formula using a node with match process expression. The match is converted to an equality formula in the transformation. Rules A and B define transformation of formula expressions in equational form. The nesting of the formula variables is preserved in this transformation.

The correctness of the partial model checker (represented by rules given in Fig. 15) is stated below in Theorem 20. For establishing this theorem, we use the result presented below in Proposition 19 which considers the effect of restriction of gnames on property satisfaction and shifts the restriction from node expression to property and vice-versa.

Proposition 19 *For an ω_0-expression N, group name g, formula φ, and δ a set of substitutions, following holds:*

(a). $N \models_\delta \varphi \implies (\nu g)N \models_\delta \varphi_{-\{g\}}$

(b). $(\nu g)N \models_\delta \varphi \implies N \models_\delta \varphi_{+\{g\}}$

where $\varphi_{-\{g\}}$ and $\varphi_{+\{g\}}$ are interpreted using the definition given in Fig. 16.

Theorem 20 (Correctness) *Let M and N be two node expressions, and δ a set of substitutions. Then for all formulas φ, the following holds:*

$$M \,|\, N \models_\delta \varphi \;\Leftrightarrow\; M \models_\delta \Pi(N)(\varphi)$$

The proof proceeds by induction on the size of the node expression and the formula, and is given in Appendix C.

7.3 An Example

$$\begin{array}{l} P \overset{\text{def}}{=} \overline{\mathbf{b}}\, y.P \\ Q \overset{\text{def}}{=} \mathbf{r}\ (x).Q \\ sys(n) \text{ represents } (\nu g)(P\!:\!\{g\} \mid \underbrace{Q\!:\!\{g\} \mid \ldots \mid Q\!:\!\{g\}}_{\text{n instances}}) \end{array}$$

Figure 17: A simple example of a parameterized system.

We consider a simple example of a parameterized system containing a node that repeatedly does a broadcast, and n receiver nodes. The example is shown in Fig. 17, where the system sys contains a broadcasting node $P\!:\!\{g\}$ and n instances of receiver node $Q\!:\!\{g\}$. The system sys is parameterized on n, the number of receivers. The property to be verified, φ_0, is specified in the $\omega\mu$-calculus and written in the equational form (Fig. 18). The property is a greatest fixed point formula and states that a *τ action is possible after every τ action.* An example of parameterized verification problem is to determine whether $\forall n.sys(n) \models \varphi_0$. The property φ_0 is first transformed by $P\!:\!\{g\}$ leading to property φ_1. Then $Q\!:\!\{g\}$ transforms property φ_1 to property φ_2. Formulas φ_1 and φ_2 differ only in the names of formula variables and hence represent the same property. Thus the sequence of transformed properties converges at φ_2. Due to this convergence, it suffices to check whether inactive node $\mathbf{0}$ satisfies φ_2, to determine $\forall n.sys(n) \models \varphi_0$. The steps involved in the transformation from φ_0 to φ_2 are shown in Fig. 18.

$\varphi_0 \equiv X =_\nu \langle\tau\rangle tt \wedge [\tau]X$

By proposition 19,

$$P\!:\!\{g\} \mid \underbrace{Q\!:\!\{g\} \mid \ldots \mid Q\!:\!\{g\}}_{\text{n instances}} \models_\delta \varphi_{0+\{g\}}$$

$$\implies (\nu g)(P\!:\!\{g\} \mid \underbrace{Q\!:\!\{g\} \mid \ldots \mid Q\!:\!\{g\}}_{\text{n instances}}) \models_\delta \varphi_0 \textit{ where } g \notin n(\varphi)$$

$$\begin{aligned}
\varphi_1 &\equiv \Pi(P\!:\!\{g'\})(\varphi_{0+\{g'\}}) \\
&\equiv X_1 =_\nu \Pi(P\!:\!\{g'\})((\langle\tau\rangle tt \vee \langle\overline{\{g'\}}\{z\}\rangle tt) \wedge ([\tau]X \wedge [\overline{\{g'\}}\{z\}]X)) \\
&\equiv X_1 =_\nu (((\langle\tau\rangle tt \vee tt \vee \ldots) \wedge ([\tau]X_1 \wedge X_1 \wedge [\overline{\{g'\}}\{z\}]X_1 \wedge [\{g'\}y]X_1)) \\
&\equiv X_1 =_\nu ([\tau]X_1 \wedge X_1 \wedge [\overline{\{g'\}}\{z\}]X_1 \wedge [\{g'\}y]X_1) \\
\varphi_2 &\equiv \Pi(Q\!:\!\{g'\})(\varphi_1) \\
&\equiv X_2 =_\nu ([\tau]X_2 \wedge X_2 \wedge [\overline{\{g'\}}\{z\}]X_2 \wedge [\{g'\}y]X2)
\end{aligned}$$

Figure 18: Property transformation for example in Fig. 17

7.4 Discussion

We presented an automatable parameterized verification technique for AHN protocols specified using the ω_m-calculus (ω_0 without mobility). Properties are specified in an expressive logic, $\omega\mu$-calculus. The technique is based on a compositional model checker for the ω_m-calculus, verifying each instance of a node in an unknown environment. This technique leads to generation of a large number of formulas at each step of property transformation. A complete automation of the technique will require development of optimization techniques to reduce the potential blow-up due to generated formulas and techniques for checking equivalence of formulas. Using this technique AHN protocols such as LMAC [vHH04] protocol can be verified for infinite instances of nodes for static topologies. Currently, this technique does not consider mobility of nodes. Future extensions to this technique will include consideration of mobility of nodes which will require symbolic representation of process interfaces as discussed in the previous chapter, to handle all possibilities of changes to process interfaces in an unknown environment. With the above optimizations and extensions to mobility it will be feasible to verify AHN protocols with node mobility such as AODV [PBRD03] routing protocol, leader election [VKT04] protocol, for infinite instances of nodes.

Chapter 8

Conclusion

In this thesis, we addressed the problem of modeling and verifying ad hoc network (AHN) protocols. Below we summarize our major results followed by a brief discussion of our work, and directions for future work.

8.1 Summary of Major Results

- **Modeling and Verification Framework.** We developed a process-algebraic framework, the ω-calculus, for modeling and verifying AHNs. The ω-calculus enables concise modeling of AHNs and provides abstraction for broadcast-based communication over dynamically changing network topologies. The ω-calculus provides a separation between the description of the control behavior of nodes and the network topology, thus permitting a natural and succinct modeling of AHNs. Bisimulation equivalence is defined for the ω-calculus and is shown to be a congruence.

- **Constraint-Based Verification.** We developed a constraint-based verification technique for AHNs to mitigate *instance explosion*. Symbolic verification of AHN protocols over a constraint-based representation of the network topology enables verification of a protocol for all possible topologies in a single verification run and permits inference of the topologies for which a property holds for the protocol under consideration.

- **Parameterized Verification.** Partial model checking of AHNs enables parameterized verification of AHN protocols. We developed a compositional analysis technique to analyze infinite instances of AHN protocols (in the absence of node mobility).

- **Implementation and Case Studies.** Prototype implementations of modeling and verification framework for AHNs presented in this thesis have been developed using the XSB tabled logic-programming system [XSB]. These implementations have allowed us to demonstrate the utility of our framework by verifying key properties of distributed algorithms for routing, leader election, and medium access control in AHNs.

8.2 Discussion

In this thesis, we developed the ω-calculus, a conservative extension of the π-calculus that permits succinct and high-level encodings of AHN protocols. The salient aspect of the calculus is its group-based support for local broadcast communication over dynamically changing network topologies. We have shown that reachability is decidable for the finite-control fragment of the calculus, and late strong, weak, and symbolic bisimulation equivalences are a congruence. We also showed how the calculus's operational semantics can be readily encoded in the XSB tabled logic-programming system, thereby permitting the generation of transition systems from ω-calculus specifications. We used this feature to implement a prototype verifier for the ω-calculus, which we then used to verify certain key properties of our encodings of the leader election algorithm of [VKT04] and the AODV routing protocol [PBRD03].

Using the ω-calculus framework as a basis, we devised a constraint-based verification technique for AHNs. Network topologies are specified symbolically using interface variables. Models with multiple possible topologies can be verified efficiently by generating interface constraints that may lead to the satisfaction of the system property under consideration. We demonstrated the utility of this technique through the verification of a medium access control protocol (LMAC) for sensor networks [vHH04].

Lastly, we developed a partial model checking technique for the parameterized verification of AHNs. The technique is developed for the ω-calculus in the absence of node mobility and extends the parameterized verification techniques developed for CCS in [BR06] and for the π-calculus in [YBR06].

We now discuss some limitations of the ω-calculus. Due to the intermittent nature of communication links and the difference in transmission powers of nodes, asymmetric connections are possible in AHNs. The ω-calculus models bidirectional (symmetric) connections only. In order to model asymmetric connections, the ω-calculus will require extensions as discussed in Section 1.2.1. Use of AHNs for safety- and mission-critical operations requires models of AHNs to be able to represent time realistically. Presently, ω-calculus models abstract away time using non-determinism. In order to facilitate analysis of time-critical AHN protocols, it will be useful to extend the ω-calculus with a notion of time. Similar concerns have been addressed in [GN09], where time is added to a process calculus for mobile processes. A global notion of time is used, and time advances simultaneously for all nodes. Node mobility in the ω-calculus is restricted by a class of invariants that are decidable properties over network topologies. More realistic mobility models (taking direction and speed into account) such as those considered in [GN09] cannot be represented in the ω-calculus. The prototype verifiers developed in this thesis check only for reachability properties. Extending these implementations to full-fledged model checkers will be useful for verifying non-trivial properties of AHN protocols.

8.3 Future Work

We present some avenues for future work including verification techniques and formal modeling features for AHN protocols.

- **Reduction Techniques.** Several symmetry reduction techniques [MDC06] have been developed to reduce the explored state-space for verification of communication-network protocols. Traditional reduction techniques exploit symmetry for specific network topologies, e.g. ring and star. These techniques cannot be readily used in the context of AHNs since the network topologies may be arbitrary. The dynamic network structure raises difficulty in recognizing and exploiting symmetry in AHNs. Our constraint-based representation of topologies, and AC (associative-commutative) unification technique [Sti81] for symmetry reduction as used for the model checking of mobile processes [YDRS05], can be leveraged in the context of AHNs.

- **Parameterized Verification.** Partial model checking facilitates parameterized verification [And95, EN96, BR06, YBR06] of infinite families of processes. The new concerns that arise in the context of parameterized verification of AHNs are broadcast-based communication and dynamic network structure. We have developed compositional analysis for broadcast feature of AHNs by extending similar techniques devised for point-to-point synchronous communication to broadcast (multi-party) synchronization. Dynamic network structure poses a problem for the partial model checking of AHNs because the effect of the dynamic neighborhood on a node's communication capabilities is not known. Constraint-based representation and processing of topologies can be used to address this problem. Symmetry reduction techniques for AHNs as presented above can be applied in addition to partial model checking. In this thesis, we developed a technique for verification of AHN protocols for arbitrary instances of topologies; parameterized verification of AHNs will facilitate verification of AHN protocols for an arbitrary number of nodes.

- **Model Generation.** The effort required in modeling applications limits the use of formal verification to experienced users. Ideally, a user should be able to leverage formal analysis techniques directly for the implementation of an application rather than having to specify a formal model for it. This motivates the plan to develop an integrated framework for model generation and verification from implementations of AHN applications. There are two approaches for verifying implementations: verify the model generated from an implementation, or verify the implementation directly. It will be useful to develop a framework for verifying implementations specified in nesC, an event-driven language widely used for programming sensor networks. nesC programs are structured which makes it easier to translate a nesC specification, as compared to translating a similar specification in other programming languages (such as C), into a formal model. The difficulty in verifying nesC implementations is that they may contain arbitrary data structures e.g. process identifiers, route tables, increasing sequence numbers. Classical counterexample-guided abstraction-refinement techniques and symmetry reduction techniques as stated above will be useful in efficiently handling complex data structures for verification of nesC implementations.

- **Modeling Language.** The scenarios in which AHN protocols operate, such as intermittent links, unreliable communication, timing issues, motivate the need for a rich modeling language for AHNs. Such a language will have capabilities to model real-time aspects and probabilistic behavior of nodes and communication links. Development of such an enhanced modeling language for AHNs is orthogonal to the development of the verification techniques proposed above.

8.4 Final Notes

AHNs are used in many applications including health and environment monitoring, and military operations. Such applications are safety-critical and demand reliability. The work presented in this thesis develops formal frameworks for assessing the reliability of AHN applications, and could therefore lead to more widespread use of AHNs. We also believe that the fundamental theory and techniques developed in this thesis for the analysis of AHNs is useful for the formal analysis of safety-critical and distributed applications in general.

Bibliography

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[And95] Henrik Reif Andersen. Partial model checking. In *Proceedings of Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 398–407. IEEE Computer Society Press, 1995.

[AP94] Roberto M. Amadio and Sanjiva Prasad. Localities and failures. In *FSTTCS*, pages 205–216. Springer-Verlag, 1994.

[BG01] Glenn Bruns and Patrice Godefroid. Temporal logic query checking. In *LICS*, pages 409–417, 2001.

[BL08] Roberto Bruni and Ivan Lanese. Parametric synchronizations in mobile nominal calculi. *Theor. Comput. Sci.*, 402(2-3):102–119, 2008.

[BR06] Samik Basu and C. R. Ramakrishnan. Compositional analysis for verification of parameterized systems. *Theor. Comput. Sci.*, 354(2):211–229, 2006.

[CABN97] William Chan, Richard Anderson, Paul Beame, and David Notkin. Combining constraint solving and symbolic model checking for a class of a systems with non-linear constraints. In *CAV*, pages 316–327. Springer-Verlag, 1997.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *FOSSACS*. Springer-Verlag, 1998.

[Cha00] William Chan. Temporal-logic queries. In *CAV*, volume 1855, pages 450–463. Springer, 2000.

[Cla96] Edmund M. Clarke, et al. Formal methods: state of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[Dam97] M. Dam. On the decidability of process equivalences for the π-calculus. *Theoretical Computer Science*, 183:215–228, 1997.

[DP99] Giorgio Delzanno and Andreas Podelski. Model checking in CLP. In *TACAS*, pages 223–239. Springer-Verlag, 1999.

[DZG00] Silvano Dal-Zilio and Andrew D. Gordon. Region analysis and a pi-calculus with groups. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, pages 1–20. Springer-Verlag, 2000.

[EM99] Cristian Ene and Traian Muntean. Expressiveness of point-to-point versus broadcast communications. In *Fundamentals of Computation Theory*, pages 258–268, 1999.

[EM01] Cristian Ene and Traian Muntean. A broadcast-based calculus for communicating systems. In *Intl. Workshop on Formal Methods for Parallel Programming: Theory and Applications*, 2001.

[EN96] E. Allen Emerson and Kedar S. Namjoshi. Automatic verification of parameterized synchronous systems. In *CAV*, pages 87–98, 1996.

[FG96] Cedric Fournet and Georges Gonthier. The reflexive cham and the join-calculus. In *POPL '96: Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 372–385, New York, NY, USA, 1996. ACM Press.

[FH05] Adrian Francalanza and Matthew Hennessy. A theory of system behaviour in the presence of node and link failures. In *CONCUR*, pages 368–382. Springer, 2005.

[Fla04] Cormac Flanagan. Automatic software model checking via constraint logic. *Sci. Comput. Program.*, 50(1-3):253–270, 2004.

[Fri99] Laurent Fribourg. Constraint logic programming applied to model checking. In *In Proc. 9th Int. Workshop on Logic-based Program Synthesis and Transformation (LOPSTR'99), LNCS 1817*, pages 30–41. Springer-Verlag, 1999.

[FvHM07] Ansgar Fehnker, Lodewijk van Hoesel, and Angelika Mader. Modelling and verification of the LMAC protocol for wireless sensor networks. In *IFM*, pages 253–272, 2007.

[GCD03] Arie Gurfinkel, Marsha Chechik, and Benet Devereux. Temporal logic query checking: A tool for model exploration. *IEEE Trans. Software Eng.*, 29(10):898–914, 2003.

[GFM09] F. Ghassemi, W.J. Fokkink, and A. Movaghar. Equational reasoning on ad hoc networks. In *Proceedings of the Third International Conference on Fundamentals of Software Engineering (FSEN)*, 2009.

[GN09] Jens Chr. Godskesen and Sebastian Nanz. Mobility models and behavioural equivalence for wireless networks. In *Coordination Models and Languages, 11th International Conference (COORDINATION)*, volume 5521 of *Lecture Notes in Computer Science*, pages 106–122. Springer, 2009.

[God07] Jens Chr. Godskesen. A calculus for mobile ad hoc networks. In *COORDINATION*, volume 4467 of *Lecture Notes in Computer Science*, pages 132–150. Springer, 2007.

[Hoa85] C. A. R. Hoare. *Communicating sequential processes.* Prentice-Hall, Inc., NJ, USA, 1985.

[HR98] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. In *High-Level Concurrent Languages*, volume 16.3 of *Electr. Notes Theor. Comput. Sci.*, pages 3–17, 1998.

[HT91] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In *ECOOP '91: Proceedings of the European Conference on Object-Oriented Programming*, pages 133–147, London, UK, 1991. Springer-Verlag.

[JB01] C. Stirling J. Bradfield. Modal logics and mu-calculi: an introduction. In *Handbook of Process Algebra*. Elsevier, 2001.

[MDC06] Alice Miller, Alastair F. Donaldson, and Muffy Calder. Symmetry in temporal logic model checking. *ACM Comput. Surv.*, 38(3), 2006.

[Mer07] Massimo Merro. An observational theory for mobile ad hoc networks. In *International Conference on the Mathematical Foundations of Programming Semantics (MFPS'07)*, volume 173 of *Electr. Notes Theor. Comput. Sci.*, pages 275–293. Elsevier, 2007.

[Mil89] Robin Milner. *Communication and concurrency.* Prentice-Hall, Inc., NJ, USA, 1989.

[Mil93] Robin Milner. The polyadic pi-calculus: a tutorial. In *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1993.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Parts I and II. *Information and Computation*, 100(1):1–77, 1992.

[MS06] N. Mezzetti and D. Sangiorgi. Towards a calculus for wireless systems. In *Proc. MFPS '06*, volume 158 of *Electr. Notes Theor. Comput. Sci.*, pages 331–354. Elsevier, 2006.

[NH06] Sebastian Nanz and Chris Hankin. A framework for security analysis of mobile wireless networks. *Theoretical Computer Science*, 367(1-2):203–227, 2006.

[OPT02] Karol Ostrovsky, K. V. S. Prasad, and Walid Taha. Towards a primitive higher order calculus of broadcasting systems. In *PPDP*, pages 2–13. ACM, 2002.

[Par01] Joachim Parrow. An introduction to the pi-calculus. In Bergstra, Ponse, and Smolka, editors, *Handbook of Process Algebra*, pages 479–543. Elsevier, 2001.

[PBRD03] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir R. Das. Ad hoc on-demand distance vector routing protocol. Internet-draft, IETF MANET Working Group, 2003. Available at `http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-13.txt`.

[Pod00] Andreas Podelski. Model checking as constraint solving. In *Proceedings of the 7th International Symposium on Static Analysis (SAS)*, pages 22–37. Springer-Verlag, 2000.

[Pra93a] K. V. S. Prasad. A calculus of value broadcasts. In *Parallel Architectures and Languages Europe*, pages 391–402, 1993.

[Pra93b] K. V. S. Prasad. Programming with broadcasts. In *International Conference on Concurrency Theory*, pages 173–187, 1993.

[Pra94] K. V. S. Prasad. Broadcasting with priority. In *European Symposium on Programming*, pages 469–484, 1994.

[Pra95] K. V. S. Prasad. A calculus of broadcasting systems. *Sci. Comput. Program.*, 25(2-3):285–327, 1995.

[PV98] Joachim Parrow and Bjorn Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Logic in Computer Science*, pages 176–185, 1998.

[RRR$^+$97] Y. S. Ramakrishna, C. R. Ramakrishnan, I. V. Ramakrishnan, Scott A. Smolka, Terrance Swift, and David Scott Warren. Efficient model checking using tabled resolution. In *CAV*, volume 1254 of *LNCS*, pages 143–154. Springer, 1997.

[San98] Davide Sangiorgi. On the bisimulation proof method. *Mathematical. Structures in Comp. Sci.*, 8(5):447–479, 1998.

[SR03] Beata Sarna Starosta and C. R. Ramakrishnan. Constraint-based model checking of data-independent systems. In *International Conference on Formal Engineering Methods (ICFEM)*, volume 2885 of *Lecture Notes in Computer Science*, pages 579–598. Springer, 2003.

[SRS08] Anu Singh, C. R. Ramakrishnan, and Scott A. Smolka. A process calculus for mobile ad hoc networks. In *Proceedings of the 10th International Conference on Coordination Models and Languag es (COORDINATION)*, volume 5052 of *Lecture Notes in Computer Science*, pages 296–314. Springer, 2008.

[SRS09a] Anu Singh, C. R. Ramakrishnan, and Scott A. Smolka. A process calculus for mobile ad hoc networks. *Science of Computer Programming Journal (To Appear)*, 2009. `http://www.cs.sunysb.edu/~anusingh/research/scp.pdf`.

[SRS09b] Anu Singh, C. R. Ramakrishnan, and Scott A. Smolka. Query-based model checking of ad hoc network protocols. In *Proceedings of the 20th International Conference on Concurrency Theory (CONCUR) (To Appear)*, 2009. `http://www.cs.sunysb.edu/~anusingh/research/mcq.pdf`.

[Sti81] Mark E. Stickel. A unification algorithm for associative-commutative functions. *J. ACM*, 28(3):423–434, 1981.

[vHH04] L.F.W. van Hoesel and P.J.M. Havinga. A lightweight medium access protocol (LMAC) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. In *1st International Workshop on Networked Sensing Systems (INSS)*, pages 205–208, 2004.

[VKT04] Sudarshan Vasudevan, James F. Kurose, and Donald F. Towsley. Design and analysis of a leader election algorithm for mobile ad hoc networks. In *ICNP*, pages 350–360. IEEE Computer Society, 2004.

[XSB] XSB. The XSB logic programming system. `http://xsb.sourceforge.net`.

[YBR06] Ping Yang, Samik Basu, and C. R. Ramakrishnan. Parameterized verification of pi-calculus systems. In *TACAS*, volume 3920 of *LNCS*, pages 42–57. Springer, 2006.

[YDRS05] Ping Yang, Yifei Dong, C. R. Ramakrishnan, and Scott A. Smolka. A provably correct compiler for efficient model checking of mobile processes. In *PADL*, volume 3350 of *LNCS*, pages 113–127. Springer, 2005.

[YRS04] Ping Yang, C. R. Ramakrishnan, and Scott A. Smolka. A logical encoding of the pi-calculus: Model checking mobile processes using tabled resolution. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(1):38–66, 2004.

[ZC05] Dezhuang Zhang and Rance Cleaveland. Efficient temporal-logic query checking for presburger systems. In *ASE*, pages 24–33. ACM, 2005.

Appendix A

Proof of Lemma 9

Lemma 9. For all nodes $M_1, M_2 \in \mathbf{N_{nf}}$, i.e., M_1, M_2 are in normal form, the following hold:

(i) $M_1 \sim M_2$ implies $\forall x \in \mathbf{Pn} : (\nu x)M_1 \sim (\nu x)M_2$;

(ii) $M_1 \sim M_2$ implies $\forall g \in \mathbf{Gn} : (\nu g)M_1 \sim (\nu g)M_2$; and

(iii) $M_1 \sim M_2$ implies $\forall N \in \mathbf{N_{nf}}$: $M_1|N \sim M_2|N$.

Proof. We show parts *(i–iii)* of the lemma simultaneously by considering the set $S = \{((\nu\tilde{g})(\nu\tilde{x})(M_1|N), (\nu\tilde{g})(\nu\tilde{x})(M_2|N)) \mid M_1 \sim M_2, \tilde{g} \subseteq \mathbf{Gn}, \tilde{x} \subseteq \mathbf{Pn}, M_1, M_2, N \in \mathbf{N_{nf}}\}$. Following Lemma 5 it is sufficient to show that S is a strong bisimulation upto $\equiv$ to establish this lemma.

Note that if $M_1 \sim M_2$ then $\mathit{fgn}(M_1) = \mathit{fgn}(M_2)$, and hence $\mathit{fgn}((\nu\tilde{g})(\nu\tilde{x})(M_1|N)) = \mathit{fgn}((\nu\tilde{g})(\nu\tilde{x})(M_2|N))$ for all $\tilde{g}, \tilde{x}$ and N. We then show that every transition from $(\nu\tilde{g})(\nu\tilde{x})(M_1|N)$ can be matched by $(\nu\tilde{g})(\nu\tilde{x})(M_2|N)$ by considering the derivations of transitions. Transitions for $(\nu\tilde{g})(\nu\tilde{x})(M_1|N)$ can be derived by use of rules CLOSE, GNAME-RES1, GNAME-RES2, MOBILITY, PAR, UNI-COM, UNI-CLOSE, COM, COM-RES, UNI-OPEN, OPEN and PNAME-RES. Only the last three steps of each transition derivation are considered in the proof. Most importantly, following Lemma 8, we do not need to consider derivations that use STRUCT rules in the last two steps. From the structural operational semantics, the last step of a derivation will be due to the outermost $(\nu\tilde{g})$ in the expression, the next-to-last step due to the $(\nu\tilde{x})$ following the outermost $(\nu\tilde{g})$, and the earliest of the three steps due to the parallel composition $(M_1|N)$.

We omit in the proof the symmetric cases arising due to the commutativity of the parallel operator '$|$'. This gives rise to 15 cases (combinations of rules in the last three steps in a derivation).

1. Case CLOSE, OPEN, COM: $(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N'\{x'/y\})$ given $M_1 \xrightarrow{\overline{G}x'} M_1'$ and $N \xrightarrow{G'(y)} N'$. The derivation is as follows, where $\tilde{x}_1 = \tilde{x} \setminus \{x'\}$.

$$\begin{array}{lcl}
\text{COM:} & \dfrac{M_1 \xrightarrow{\overline{G}x'} M_1' \quad N \xrightarrow{G'(y)} N'}{M_1|N \xrightarrow{\overline{G}x'} M_1'|N'\{x'/y\}} & G \cap G' \neq \emptyset \\
\text{OPEN:} & \overline{(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{G}x'} (\nu\tilde{x_1})(M_1'|N'\{x'/y\})} & \\
\text{CLOSE:} & \overline{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu x')(\nu\tilde{x}_1)(M_1'|N'\{x'/y\})} & G \setminus \tilde{g} = \emptyset
\end{array}$$

Since $M_1 \sim M_2$, $M_1 \xrightarrow{\overline{G}x'} M_1'$ means that there is an M_2' such that $M_2 \xrightarrow{\overline{G}x'} M_2'$ and $M_1' \sim M_2'$. Moreover, there exist expressions M_{N1}', M_{N2}' and N_N' in normal form such that $M_1' \equiv M_{N1}'$, $M_2' \equiv M_{N2}'$ and $N'\{x'/y\} \equiv N_N'$. Now, since $M_1' \sim M_2'$, we know $M_{N1}' \sim M_{N2}'$. Hence by construction of S, we can conclude that the pair $(\ (\nu\tilde{g})(\nu x')(\nu\tilde{x}_1)(M_{N1}'|N_N'), (\nu\tilde{g})(\nu x')(\nu\tilde{x}_1)(M_{N2}'|N_N')\ ) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x})(M_1'|N'\{x'/y\}), (\nu\tilde{g})(\nu\tilde{x})(M_2'|N'\{x'/y\})) \in {\equiv}S{\equiv}$.

2. Case CLOSE, OPEN, PAR:

$(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N)$ given $M_1 \xrightarrow{\overline{G}x'} M_1'$, where $\tilde{x}_1 = \tilde{x} \setminus \{x'\}$. The derivation is given below:

$$\begin{array}{lcl}
\text{PAR:} & \dfrac{M_1 \xrightarrow{\overline{G}x'} M_1'}{M_1|N \xrightarrow{\overline{G}x'} M_1'|N} & \\
\text{OPEN:} & \overline{(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{G}x'} (\nu\tilde{x_1})(M_1'|N)} & \\
\text{CLOSE:} & \overline{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu x')(\nu\tilde{x}_1)(M_1'|N)} & G \setminus \tilde{g} = \emptyset
\end{array}$$

Since $M_1 \sim M_2$, $M_1 \xrightarrow{\overline{G}x'} M_1'$ means that there is an M_2' such that $M_2 \xrightarrow{\overline{G}x'} M_2'$ and $M_1' \sim M_2'$. Moreover, there exist expressions M_{N1}' and

M'_{N2} in normal form such that $M'_1 \equiv M'_{N1}$ and $M'_2 \equiv M'_{N2}$. Now, since $M'_1 \sim M'_2$, we know $M'_{N1} \sim M'_{N2}$. Hence by construction of S, we can conclude that the pair $((\nu\tilde{g})(\nu\tilde{x})(M'_{N_1}|N), (\nu\tilde{g})(\nu\tilde{x})(M'_{N_2}|N)) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x})(M'_1|N), (\nu\tilde{g})(\nu\tilde{x})(M'_2|N)) \in \equiv S \equiv$.

3. Case CLOSE, PNAME-RES, COM-RES:

   $(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu\tilde{x}_1)(M'_1|N')$ given $M_1 \xrightarrow{(\nu x')\overline{G}x'} M'_1$ and $N \xrightarrow{G'(x')} N'$ where $\tilde{x}_1 = \tilde{x} \cup \{x'\}$. The derivation is given below:

$$\begin{array}{lcr}
\text{COM-RES:} & \dfrac{M_1 \xrightarrow{(\nu x')\overline{G}x'} M'_1 \quad N \xrightarrow{G'(x')} N'}{M_1|N \xrightarrow{(\nu x')\overline{G}x'} M'_1|N'} & G \cap G' \neq \emptyset \\
\text{PNAME-RES:} & \dfrac{M_1|N \xrightarrow{(\nu x')\overline{G}x'} M'_1|N'}{(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{G}x'} (\nu\tilde{x})(M'_1|N')} & x' \notin \tilde{x} \\
\text{CLOSE:} & \dfrac{(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{G}x'} (\nu\tilde{x})(M'_1|N')}{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu x')(\nu\tilde{x})(M'_1|N')} & G \setminus \tilde{g} = \emptyset
\end{array}$$

   Since $M_1 \sim M_2$, $M_1 \xrightarrow{(\nu x')\overline{G}x'} M'_1$ means that there is an M'_2 such that $M_2 \xrightarrow{(\nu x')\overline{G}x'} M'_2$ and $M'_1 \sim M'_2$. Moreover, there exist expressions M'_{N1}, M'_{N2} and N'_N in normal form such that $M'_1 \equiv M'_{N1}$, $M'_2 \equiv M'_{N2}$ and $N' \equiv N'_N$. Now, since $M'_1 \sim M'_2$, we know $M'_{N_1} \sim M'_{N_2}$. Hence by construction of S, we can conclude that the pair $((\nu\tilde{g})(\nu\tilde{x}_1)(M'_{N_1}|N'_N), (\nu\tilde{g})(\nu\tilde{x}_1)(M'_{N_2}|N'_N)) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x}_1)(M'_1|N'), (\nu\tilde{g})(\nu\tilde{x}_1)(M'_2|N')) \in \equiv S \equiv$.

4. Case CLOSE, PNAME-RES, PAR:

   $(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu\tilde{x}_1)(M'_1|N)$ given $M_1 \xrightarrow{(\nu x')\overline{G}x'} M'_1$, where $\tilde{x}_1 = \tilde{x} \cup \{x'\}$. The derivation is given below:

$$\begin{array}{lcr}
\text{PAR:} & \dfrac{M_1 \xrightarrow{(\nu x')\overline{G}x'} M_1'}{M_1|N \xrightarrow{(\nu x')\overline{G}x'} M_1'|N} & x' \cap fn(N) = \emptyset \\
\text{PNAME-RES:} & \dfrac{}{(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{G}x'} (\nu\tilde{x})(M_1'|N)} & x' \notin \tilde{x} \\
\text{CLOSE:} & \dfrac{}{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu x')(\nu\tilde{x})(M_1'|N)} & G \setminus \tilde{g} = \emptyset
\end{array}$$

Since $M_1 \sim M_2$, $M_1 \xrightarrow{(\nu x')\overline{G}x'} M_1'$ means that there is an M_2' such that $M_2 \xrightarrow{(\nu x')\overline{G}x'} M_2'$ and $M_1' \sim M_2'$. Moreover, there exist expressions M_{N_1}' and M_{N_2}' in normal form such that $M_1' \equiv M_{N_1}'$ and $M_2' \equiv M_{N_2}'$. Now, since $M_1' \sim M_2'$, we know $M_{N_1}' \sim M_{N_2}'$. Hence by construction of S, we can conclude that the pair $((\nu\tilde{g})(\nu\tilde{x_1})(M_{N_1}'|N), (\nu\tilde{g})(\nu\tilde{x_1})(M_{N_2}'|N)) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x_1})(M_1'|N), (\nu\tilde{g})(\nu\tilde{x_1})(M_2'|N)) \in \equiv S \equiv$.

5. Case GNAME-RES1, UNI-OPEN, PAR:

   $(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{z{:}G''}x'} (\nu\tilde{g})(\nu\tilde{x_1})(M_1'|N)$ given $M_1 \xrightarrow{\overline{z{:}G}x'} M_1'$, where $\tilde{x_1} = \tilde{x} \setminus \{x'\}$ and $G'' = G \setminus \tilde{g}$. The derivation is given below:

$$\begin{array}{lcr}
\text{PAR:} & \dfrac{M_1 \xrightarrow{\overline{z{:}G}x'} M_1'}{M_1|N \xrightarrow{\overline{z{:}G}x'} M_1'|N} & \\
\text{UNI-OPEN:} & \dfrac{}{(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{z{:}G}x'} (\nu\tilde{x_1})(M_1'|N)} & x' \neq z, z \notin \tilde{x} \\
\text{GNAME-RES1:} & \dfrac{}{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{z{:}G''}x'} (\nu\tilde{g})(\nu\tilde{x_1})(M_1'|N)} & G'' \neq \emptyset
\end{array}$$

   Since $M_1 \sim M_2$, $M_1 \xrightarrow{\overline{z{:}G}x'} M_1'$ means that there is an M_2' such that $M_2 \xrightarrow{\overline{z{:}G}x'} M_2'$ and $M_1' \sim M_2'$. Moreover, there exist expressions M_{N_1}' and M_{N_2}' in normal form such that $M_1' \equiv M_{N_1}'$ and $M_2' \equiv M_{N_2}'$. Now, since $M_1' \sim M_2'$, we know $M_{N_1}' \sim M_{N_2}'$. Hence, by construction of S, we can conclude that the pair $((\nu\tilde{g})(\nu\tilde{x_1})(M_{N_1}'|N), (\nu\tilde{g})(\nu\tilde{x_1})(M_{N_2}'|N)) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x_1})(M_1'|N), (\nu\tilde{g})(\nu\tilde{x_1})(M_2'|N)) \in \equiv S \equiv$.

6. Case GNAME-RES1, OPEN, COM:

   $(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{G''}x'} (\nu\tilde{g})(\nu\tilde{x_1})(M_1'|N'\{x'/y\})$ given $M_1 \xrightarrow{\overline{G}x'} M_1'$ and $N \xrightarrow{G'(y)} N'$, where $\tilde{x_1} = \tilde{x} \setminus \{x'\}$ and $G'' = G \setminus \tilde{g}$. The derivation is given below:

$$\begin{array}{ll}
\text{COM:} & \dfrac{M_1 \xrightarrow{\overline{G}x'} M_1' \quad N \xrightarrow{G'(y)} N'}{M_1|N \xrightarrow{\overline{G}x'} M_1'|N'\{x'/y\}} \quad G \cap G' \neq \emptyset \\
\text{OPEN:} & \dfrac{}{(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{G}x'} (\nu\tilde{x_1})(M_1'|N'\{x'/y\})} \\
\text{GNAME-RES1:} & \dfrac{}{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{G''}x'} (\nu\tilde{g})(\nu\tilde{x_1})(M_1'|N'\{x'/y\})} \quad G'' \neq \emptyset
\end{array}$$

   Since $M_1 \sim M_2$, $M_1 \xrightarrow{\overline{G}x'} M_1'$ means that there is an M_2' such that $M_2 \xrightarrow{\overline{G}x'} M_2'$ and $M_1' \sim M_2'$. Moreover, there exist expressions M'_{N_1}, M'_{N_2} and N'_N in normal form such that $M_1' \equiv M'_{N_1}$, $M_2' \equiv M'_{N_2}$ and $N'\{x'/y\} \equiv N'_N$. Now, since $M_1' \sim M_2'$, we know $M'_{N_1} \sim M'_{N_2}$. Hence, by construction of S, we can conclude that the pair $((\nu\tilde{g})(\nu\tilde{x_1})(M'_{N_1}|N'_N), (\nu\tilde{g})(\nu\tilde{x_1})(M'_{N_2}|N'_N)) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x_1})(M_1'|N'\{x'/y\}), (\nu\tilde{g})(\nu\tilde{x_1})(M_2'|N'\{x'/y\})) \in \equiv S \equiv$.

7. Case GNAME-RES1, OPEN, PAR:

   $(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{G''}x'} (\nu\tilde{g})(\nu\tilde{x_1})(M_1'|N)$ given $M_1 \xrightarrow{\overline{G}x'} M_1'$, where $\tilde{x_1} = \tilde{x} \setminus \{x'\}$ and $G'' = G \setminus \tilde{g}$. The derivation is given below:

$$\begin{array}{ll}
\text{PAR:} & \dfrac{M_1 \xrightarrow{\overline{G}x'} M_1'}{M_1|N \xrightarrow{\overline{G}x'} M_1'|N} \\
\text{OPEN:} & \dfrac{}{(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{G}x'} (\nu\tilde{x_1})(M_1'|N)} \\
\text{GNAME-RES1:} & \dfrac{}{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{G''}x'} (\nu\tilde{g})(\nu\tilde{x_1})(M_1'|N)} \quad G'' \neq \emptyset
\end{array}$$

   Since $M_1 \sim M_2$, $M_1 \xrightarrow{\overline{G}x'} M_1'$ means that there is an M_2' such that $M_2 \xrightarrow{\overline{G}x'} M_2'$ and $M_1' \sim M_2'$. Moreover, there exist expressions M'_{N_1} and M'_{N_2} in normal form such that $M_1' \equiv M'_{N_1}$ and $M_2' \equiv M'_{N_2}$. Now, since

$M_1' \sim M_2'$, we know $M'_{N_1} \sim M'_{N_2}$. Hence, by construction of S, we can conclude that the pair $((\nu\tilde{g})(\nu\tilde{x_1})(M'_{N_1}|N), (\nu\tilde{g})(\nu\tilde{x_1})(M'_{N_2}|N)) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x_1})(M_1'|N), (\nu\tilde{g})(\nu\tilde{x_1})(M_2'|N)) \in \equiv S \equiv$.

8. Case GNAME-RES1, PNAME-RES, MOBILITY:

   $(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\mu} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N')$. The derivation is given below:

   $$\begin{array}{ll}
   \text{MOBILITY:} & \dfrac{}{M_1|N \xrightarrow{\mu} M_1'|N'} \\
   \text{PNAME-RES:} & \dfrac{}{(\nu\tilde{x})(M_1|N) \xrightarrow{\mu} (\nu\tilde{x})(M_1'|N')} \\
   \text{GNAME-RES1:} & \dfrac{}{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\mu} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N')}
   \end{array}$$

   and $I(M_1|N) \implies I(M_1'|N')$ for a connectivity invariant I.

   Now consider the following cases for M_1' and N':

   (a) $M_1' = M_1$ and N' differs from N only in one of its basic node's interface, i.e. N' is obtained by replacing one basic node $P : G$ in N by $P : G'$, where $G' \subseteq \mathit{fgn}(M_1) \cup \mathit{fgn}(N)$.

   Since $M_1 \sim M_2$, $\mathit{fgn}(M_1) = \mathit{fgn}(M_2)$ and $M_1|N \xrightarrow{\mu} M_1|N'$ imply that $M_2|N \xrightarrow{\mu} M_2|N'$ such that $I(M_2|N) \implies I(M_2|N')$, and it can be derived that $(\nu\tilde{g})(\nu\tilde{x})(M_2|N) \xrightarrow{\mu} (\nu\tilde{g})(\nu\tilde{x})(M_2|N')$. Moreover, there exist N'_N in normal form such that $N' \equiv N'_N$. Hence, by construction of S, we can conclude that pair $((\nu\tilde{g})(\nu\tilde{x})(M_1|N'_N), (\nu\tilde{g})(\nu\tilde{x})(M_2|N'_N)) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x})(M_1|N'), (\nu\tilde{g})(\nu\tilde{x})(M_2|N')) \in \equiv S \equiv$.

   (b) $N' = N$ and M_1' is obtained from M_1 by replacing one of its basic node $P : G_P$ in M_1 by $P : G'_P$ in M_1', where $G'_P \subseteq \mathit{fgn}(M_1) \cup \mathit{fgn}(N)$.

   Let M_2 contain a basic node $Q : G_Q$ and M_2' differ from M_2 only due to $Q : G_Q$ replaced by $Q : G'_Q$, where $G'_Q \subseteq \mathit{fgn}(M_2) \cup \mathit{fgn}(N)$.

   Consider the following two cases:

   (i) G'_P and G'_Q contain gnames only in $\mathit{fgn}(M_1)$ and $\mathit{fgn}(M_2)$, respectively, then M_1' and M_2' can be derived using MOBILITY rule from M_1 and M_2, respectively. Since $M_1 \sim M_2$, $fgn(M_1) = fgn(M_2)$ and $M_1 \xrightarrow{\mu} M_1'$ implies that $M_2 \xrightarrow{\mu} M_2'$, and $M_1' \sim M_2'$.

(ii) G_P' and G_Q' also contain gnames in $fgn(N)$. Since the possible new free gnames (other than $fgn(M_1)$ and $fgn(M_2)$), added to basic nodes $P : G_P$ in M_1 and $Q : G_Q$ in M_2 leading to M_1' and M_2', respectively, are drawn from the same set of gnames $fgn(N)$, similarity in behavior (transitions) of M_1' and M_2' is preserved i.e. $M_1' \sim M_2'$.

$M_2|N \xrightarrow{\mu} M_2'|N$ and it can be derived that $(\nu\tilde{g})(\nu\tilde{x})(M_2|N) \xrightarrow{\mu} (\nu\tilde{g})(\nu\tilde{x})(M_2'|N)$ such that $I(M_2|N) \implies I(M_2'|N)$. Moreover, there exist expressions M_{N_1}' and M_{N_2}' in normal form such that $M_1' \equiv M_{N_1}'$ and $M_2' \equiv M_{N_2}'$. Now, since $M_1' \sim M_2'$, we know $M_{N_1}' \sim M_{N_2}'$. Hence, by construction of S, we can conclude that the pair $((\nu\tilde{g})(\nu\tilde{x})(M_{N_1}'|N), (\nu\tilde{g})(\nu\tilde{x})(M_{N_2}'|N)) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x})(M_1'|N), (\nu\tilde{g})(\nu\tilde{x})(M_2'|N)) \in \equiv S \equiv$.

9. Case GNAME-RES1, PNAME-RES, PAR:

$(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\alpha\backslash\tilde{g}} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N)$ given $M_1 \xrightarrow{\alpha} M_1'$. The derivation is given below:

$$\begin{array}{lcr}
\text{PAR:} & \dfrac{M_1 \xrightarrow{\alpha} M_1'}{M_1|N \xrightarrow{\alpha} M_1'|N} & bn(\alpha) \cap fn(N) = \emptyset \\
\text{PNAME-RES:} & \dfrac{}{(\nu\tilde{x})(M_1|N) \xrightarrow{\alpha} (\nu\tilde{x})(M_1'|N)} & \tilde{x} \cap n(\alpha) = \emptyset \\
\text{GNAME-RES1:} & \dfrac{}{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\alpha\backslash\tilde{g}} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N)} &
\end{array}$$

Since $M_1 \sim M_2$, $M_1 \xrightarrow{\alpha} M_1'$ means that there is an M_2' such that $M_2 \xrightarrow{\alpha} M_2'$ and $M_1' \sim M_2'$. Moreover, there exist expressions M_{N_1}' and M_{N_2}' in normal form such that $M_1' \equiv M_{N_1}'$ and $M_2' \equiv M_{N_2}'$. Now, since $M_1' \sim M_2'$, we know $M_{N_1}' \sim M_{N_2}'$. Hence, by construction of S, we can conclude that the pair $((\nu\tilde{g})(\nu\tilde{x})(M_{N_1}'|N), (\nu\tilde{g})(\nu\tilde{x})(M_{N_2}'|N)) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x})(M_1'|N), (\nu\tilde{g})(\nu\tilde{x})(M_2'|N)) \in \equiv S \equiv$.

For the case $\alpha = \mu$, the conditions $I(M_1|N) \implies I(M_1'|N)$ and

$I(M_2|N) \implies I(M_2'|N)$, for a connectivity invariant I, also come into effect.

Note that if $\alpha \setminus \tilde{g}$ is of the form $G(x')$ or $z : G(x')$, where $x' \in \mathbf{Pn}$, the proof involves following reasoning:

$M_1 \sim M_2$ implies for all $y \in \mathbf{Pn}$, $M_1'\{y/x'\} \sim M_2'\{y/x'\}$. Moreover, there exist expressions M_{N_1}' and M_{N_2}' in normal form such that $M_1' \equiv M_{N_1}'$ and $M_2' \equiv M_{N_2}'$. We infer that for all $y \in \mathbf{Pn}$, $M_1'\{y/x'\} \sim M_2'\{y/x'\}$ implies $M_{N_1}'\{y/x'\} \sim M_{N_2}'\{y/x'\}$. Therefore, for all $y \in \mathbf{Pn}$, $((\nu\tilde{g})(\nu\tilde{x})(M_{N_1}'\{y/x'\}|N), (\nu\tilde{g})(\nu\tilde{x})(M_{N_2}'\{y/x'\}|N)) \in S$. Since $bn(\alpha) \cap fn(N) = \emptyset$, we know $x' \notin fn(N)$. Hence, for all pname $y \in \mathbf{Pn}$, $(\nu\tilde{g})(\nu\tilde{x})(M_{N_1}'\{y/x'\}|N) = ((\nu\tilde{g})(\nu\tilde{x})(M_{N_1}'|N))\{y/x'\}$ and $(\nu\tilde{g})(\nu\tilde{x})(M_{N_2}'\{y/x'\}|N) = ((\nu\tilde{g})(\nu\tilde{x})(M_{N_2}'|N))\{y/x'\}$. Hence, by construction of S, we can conclude that for all $y \in \mathbf{Pn}$, the pair $(((\nu\tilde{g})(\nu\tilde{x})(M_{N_1}'|N))\{y/x'\}, ((\nu\tilde{g})(\nu\tilde{x})(M_{N_2}'|N))\{y/x'\}) \in S$, and hence for all $y \in \mathbf{Pn}$, $(((\nu\tilde{g})(\nu\tilde{x})(M_1'|N)\{y/x'\}), ((\nu\tilde{g})(\nu\tilde{x})(M_2'|N))\{y/x'\}) \in {\equiv}S{\equiv}$.

10. Case GNAME-RES1, PNAME-RES, UNI-COM:

$(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N'\{x'/y\})$ given $M_1 \xrightarrow{\overline{z:G}x'} M_1'$ and $N \xrightarrow{z:G'(y)} N'$. The derivation is given below:

$$\begin{array}{ll}
\text{UNI-COM:} & \dfrac{M_1 \xrightarrow{\overline{z:G}x'} M_1' \quad N \xrightarrow{z:G'(y)} N'}{M_1|N \xrightarrow{\tau} M_1'|N'\{x'/y\}} \quad G \cap G' \neq \emptyset \\
\text{PNAME-RES:} & \dfrac{}{(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{x})(M_1'|N'\{x'/y\})} \\
\text{GNAME-RES1:} & \dfrac{}{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N'\{x'/y\})}
\end{array}$$

Since $M_1 \sim M_2$, $M_1 \xrightarrow{\overline{z:G}x'} M_1'$ means that there is an M_2' s.t. $M_2 \xrightarrow{\overline{z:G}x'} M_2'$ and $M_1' \sim M_2'$. Moreover, there exist expressions M_{N_1}', M_{N_2}' and N_N' in normal form such that $M_1' \equiv M_{N_1}'$, $M_2' \equiv M_{N_2}'$ and $N'\{x'/y\} \equiv N_N'$. Now, since $M_1' \sim M_2'$, we know $M_{N_1}' \sim M_{N_2}'$. Hence, by construction of S, we can conclude that the pair $((\nu\tilde{g})(\nu\tilde{x})(M'_{N_1}|N'_N), (\nu\tilde{g})(\nu\tilde{x})(M'_{N_2}|N'_N)) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x})(M'_1|N'\{x'/y\}), (\nu\tilde{g})(\nu\tilde{x})(M'_2|N'\{x'/y\})) \in \equiv S \equiv$.

11. Case GNAME-RES1, PNAME-RES, UNI-CLOSE:

$(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu\tilde{x_1})(M'_1|N')$ given $M_1 \xrightarrow{(\nu x')\overline{z{:}G}x'} M'_1$ and $N \xrightarrow{z{:}G'(x')} N'$, where $\tilde{x_1} = \tilde{x} \cup \{x'\}$. The derivation is given below:

$$\begin{array}{ll}
\text{UNI-CLOSE:} & \dfrac{M_1 \xrightarrow{(\nu x')\overline{z{:}G}x'} M'_1 \quad N \xrightarrow{z{:}G'(x')} N'}{M_1|N \xrightarrow{\tau} (\nu x')(M'_1|N')} \quad G \cap G' \neq \emptyset \\
\text{PNAME-RES:} & \dfrac{}{(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{x})(\nu x')(M'_1|N')} \\
\text{GNAME-RES1:} & \dfrac{}{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu\tilde{x})(\nu x')(M'_1|N')}
\end{array}$$

Since $M_1 \sim M_2$, $M_1 \xrightarrow{(\nu x')\overline{z{:}G}x'} M'_1$ means that there exists an M'_2 such that $M_2 \xrightarrow{(\nu x')\overline{z{:}G}x'} M'_2$ and $M'_1 \sim M'_2$. Moreover, there exist expressions M'_{N_1}, M'_{N_2} and N'_N in normal form such that $M'_1 \equiv M'_{N_1}$, $M'_2 \equiv M'_{N_2}$ and $N' \equiv N'_N$. Now, since $M'_1 \sim M'_2$, we know $M'_{N_1} \sim M'_{N_2}$. Hence, by construction of S, we can conclude that the pair $((\nu\tilde{g})(\nu\tilde{x_1})(M'_{N_1}|N'_N), (\nu\tilde{g})(\nu\tilde{x_1})(M'_{N_2}|N'_N)) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x_1})(M'_1|N'), (\nu\tilde{g})(\nu\tilde{x_1})(M'_2|N')) \in \equiv S \equiv$.

12. Case GNAME-RES1, PNAME-RES, COM:

$(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\overline{G''}x'} (\nu\tilde{g})(\nu\tilde{x})(M'_1|N'\{x'/y\})$ given $M_1 \xrightarrow{\overline{G}x'} M'_1$ and $N \xrightarrow{G'(y)} N'$, where $G'' = G \setminus \tilde{g}$. The derivation is given below:

$$\begin{array}{ll}
\text{COM:} & \dfrac{M_1 \xrightarrow{\overline{G}x'} M'_1 \quad N \xrightarrow{G'(y)} N'}{M_1|N \xrightarrow{\overline{G}x'} M'_1|N'\{x'/y\}} \quad G \cap G' \neq \emptyset \\
\text{PNAME-RES:} & \dfrac{}{(\nu\tilde{x})(M_1|N) \xrightarrow{\overline{G}x'} (\nu\tilde{x})(M'_1|N'\{x'/y\})} \quad x' \notin \tilde{x} \\
\text{GNAME-RES1:} & \dfrac{}{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\overline{G''}x'} (\nu\tilde{g})(\nu\tilde{x})(M'_1|N'\{x'/y\})} \quad G'' \neq \emptyset
\end{array}$$

Since $M_1 \sim M_2$, $M_1 \xrightarrow{\overline{G}x'} M_1'$ means that there exists an M_2' such that $M_2 \xrightarrow{\overline{G}x'} M_2'$ and $M_1' \sim M_2'$. Moreover, there exist expressions M'_{N_1}, M'_{N_2} and N'_N in normal form such that $M_1' \equiv M'_{N_1}$, $M_2' \equiv M'_{N_2}$ and $N'\{x'/y\} \equiv N'_N$. Now, since $M_1' \sim M_2'$, we know $M'_{N_1} \sim M'_{N_2}$. Hence, by construction of S, we can conclude that the pair $((\nu\tilde{g})(\nu\tilde{x})(M'_{N_1}|N'_N), (\nu\tilde{g})(\nu\tilde{x})(M'_{N_2}|N'_N)) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x})(M_1'|N'\{x'/y\}), (\nu\tilde{g})(\nu\tilde{x})(M_2'|N'\{x'/y\})) \in \equiv S \equiv$.

13. Case GNAME-RES1, PNAME-RES, COM-RES:

    $(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{G''}x'} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N')$ given $M_1 \xrightarrow{(\nu x')\overline{G}x'} M_1'$ and $N \xrightarrow{G'(x')} N'$, where $G'' = G \setminus \tilde{g}$. The derivation is given below:

$$\begin{array}{lcr}
\text{COM-RES:} & \dfrac{M_1 \xrightarrow{(\nu x')\overline{G}x'} M_1' \quad N \xrightarrow{G'(x')} N'}{M_1|N \xrightarrow{(\nu x')\overline{G}x'} M_1'|N'} & G \cap G' \neq \emptyset \\
\text{PNAME-RES:} & \dfrac{}{(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{G}x'} (\nu\tilde{x})(M_1'|N')} & x' \notin \tilde{x} \\
\text{GNAME-RES1:} & \dfrac{}{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{(\nu x')\overline{G''}x'} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N')} & G'' \neq \emptyset
\end{array}$$

    Since $M_1 \sim M_2$, $M_1 \xrightarrow{(\nu x')\overline{G}x'} M_1'$ means that there is an M_2' such that $M_2 \xrightarrow{(\nu x')\overline{G}x'} M_2'$ and $M_1' \sim M_2'$. Moreover, there exist expressions M'_{N_1}, M'_{N_2} and N'_N in normal form such that $M_1' \equiv M'_{N_1}$, $M_2' \equiv M'_{N_2}$ and $N' \equiv N'_N$. Now, since $M_1' \sim M_2'$, we know $M'_{N_1} \sim M'_{N_2}$. Hence, by construction of S, we can conclude that the pair $((\nu\tilde{g})(\nu\tilde{x})(M'_{N_1}|N'_N), (\nu\tilde{g})(\nu\tilde{x})(M'_{N_2}|N'_N)) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x})(M_1'|N'), (\nu\tilde{g})(\nu\tilde{x})(M_2'|N')) \in \equiv S \equiv$.

14. Case GNAME-RES2, PNAME-RES, COM:

    $(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N'\{x'/y\})$ given $M_1 \xrightarrow{\overline{G}x'} M_1'$ and $N \xrightarrow{G'(y)} N'$. The derivation is given below:

$$
\begin{array}{ll}
\text{COM:} & \dfrac{M_1 \xrightarrow{\overline{G}x'} M_1' \quad N \xrightarrow{G'(y)} N'}{M_1|N \xrightarrow{\overline{G}x'} M_1'|N'\{x'/y\}} \quad G \cap G' \neq \emptyset \\
\text{PNAME-RES:} & \dfrac{}{(\nu\tilde{x})(M_1|N) \xrightarrow{\overline{G}x'} (\nu\tilde{x})(M_1'|N'\{x'/y\})} \quad x' \notin \tilde{x} \\
\text{GNAME-RES2:} & \dfrac{}{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N'\{x'/y\})} \quad G \setminus \tilde{g} = \emptyset
\end{array}
$$

Since $M_1 \sim M_2$, $M_1 \xrightarrow{\overline{G}x'} M_1'$ means that there is an M_2' such that $M_2 \xrightarrow{\overline{G}x'} M_2'$ and $M_1' \sim M_2'$. Moreover, there exist expressions M'_{N_1}, M'_{N_2} and N'_N in normal form such that $M_1' \equiv M'_{N_1}$, $M_2' \equiv M'_{N_2}$ and $N'\{x'/y\} \equiv N'_N$. Now, since $M_1' \sim M_2'$, we know $M'_{N_1} \sim M'_{N_2}$. Hence, by construction of S, we can conclude that the pair $((\nu\tilde{g})(\nu\tilde{x})(M'_{N_1}|N'_N), (\nu\tilde{g})(\nu\tilde{x})(M'_{N_2}|N'_N)) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x})(M_1'|N'\{x'/y\}), (\nu\tilde{g})(\nu\tilde{x})(M_2'|N'\{x'/y\})) \in \equiv S \equiv$.

15. Case GNAME-RES2, PNAME-RES, PAR:
$(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N)$ given $M_1 \xrightarrow{\overline{G}x'} M_1'$.

$$
\begin{array}{ll}
\text{PAR:} & \dfrac{M_1 \xrightarrow{\overline{G}x'} M_1'}{M_1|N \xrightarrow{\overline{G}x'} M_1'|N} \\
\text{PNAME-RES:} & \dfrac{}{(\nu\tilde{x})(M_1|N) \xrightarrow{\overline{G}x'} (\nu\tilde{x})(M_1'|N)} \quad x' \notin \tilde{x} \\
\text{GNAME-RES2:} & \dfrac{}{(\nu\tilde{g})(\nu\tilde{x})(M_1|N) \xrightarrow{\tau} (\nu\tilde{g})(\nu\tilde{x})(M_1'|N)} \quad G \setminus \tilde{g} = \emptyset
\end{array}
$$

Since $M_1 \sim M_2$, $M_1 \xrightarrow{\overline{G}x'} M_1'$ means that there exists an M_2' such that $M_2 \xrightarrow{\overline{G}x'} M_2'$ and $M_1' \sim M_2'$. Moreover, there exist expression M'_{N_1} and M'_{N_2} in normal form such that $M_1' \equiv M'_{N_1}$ and $M_2' \equiv M'_{N_2}$. Now, since $M_1' \sim M_2'$, we know $M'_{N_1} \sim M'_{N_2}$. Hence, by construction of S, we can conclude that the pair $((\nu\tilde{g})(\nu\tilde{x})(M'_{N_1}|N), (\nu\tilde{g})(\nu\tilde{x})(M'_{N_2}|N)) \in S$, and hence $((\nu\tilde{g})(\nu\tilde{x})(M_1'|N), (\nu\tilde{g})(\nu\tilde{x})(M_2'|N)) \in \equiv S \equiv$.

By considering the 15 cases and their symmetric counterparts due to commutativity of '|' operator, all possible derivations are covered and we conclude that for every transition from $(\nu\tilde{g})(\nu\tilde{x})(M_1|N)$, there is a transition from $(\nu\tilde{g})(\nu\tilde{x})(M_2|N)$ such that the destinations of the two transitions are related by $\equiv S \equiv$. Thus we establish that S is a strong bisimulation upto $\equiv$. Following Lemma 5, we conclude that S is a strong bisimulation. Therefore, $\sim$ is preserved by restriction of pnames and gnames, and the parallel operator for ω-expressions in normal form.

This proof is *complete* because at each proof step all possible transitions from an expression are considered to find its derivatives. The fifteen cases along with their symmetric counterparts for the parallel operator cover all the derivation possibilities. All the possible transitions at the node level (pertaining to broadcast, unicast, silent action, and mobility) are taken into account through the derivations given in the proof. $\square$

Appendix B

Symbolic Bisimulation for the ω_0-Calculus

We prove that the symbolic bisimulation equivalence for the ω_0-calculus is a congruence. The proof for the extended calculi follow along the same lines.

Lemma 21 *For all $M_1, M_2 \in \mathbf{N_{nf}}$, i.e., M_1, M_2 are in normal form, the following hold:*

(i) $M_1 \asymp M_2$ implies $\forall g \in \mathbf{Gn} : (\nu g)M_1 \asymp (\nu g)M_2$; and

(ii) $M_1 \asymp M_2$ implies $\forall N \in \mathbf{N_{nf}}$: $M_1|N \asymp M_2|N$.

Proof. We show parts *(i–ii)* of the lemma simultaneously by considering the set $S = \{((\nu\tilde{g})(M_1|N), (\nu\tilde{g})(M_2|N)) \mid M_1 \asymp M_2, \tilde{g} \subseteq \mathbf{Gn}, M_1, M_2, N \in \mathbf{N_{nf}}\}$. Following Lemma 5 it is sufficient to show that S is a strong bisimulation upto $\equiv$ to establish this lemma.

Note that if $M_1 \asymp M_2$ then $fgn(M_1) = fgn(M_2)$, and hence $fgn((\nu\tilde{g})(M_1|N)) = fgn((\nu\tilde{g})(M_2|N))$ for all $\tilde{g}$ and N. We then show that every transition from $(\nu\tilde{g})(M_1|N)$ can be matched by $(\nu\tilde{g})(M_2|N)$ by considering the derivations of transitions. Transitions for $(\nu\tilde{g})(M_1|N)$ can be derived by the use of rules GNAME-RES1, GNAME-RES2, MOBILITY, PAR and COM. Only the last two steps of each transition derivation are considered in the proof. Most importantly, following Lemma 8, we do not need to consider derivations that use STRUCT rules in the last step. From the structural operational semantics, the last step of a derivation will be due to the outermost $(\nu\tilde{g})$ in the expression, and the first step due to the parallel composition $(M_1|N)$. We omit in the proof the symmetric cases arising due to the commutativity of the parallel operator '$|$'. This gives rise to 5 cases (combinations of rules in the last two steps in a derivation).

1. Case GNAME-RES1, COM:

   $(\nu\tilde{g})(M_1|N) \overset{C_1\wedge C,\overline{G''}x}{\longrightarrow} (\nu\tilde{g})(M_1'|N'\{x/y\})$ given $M_1 \overset{C_1,\overline{G}x}{\longrightarrow} M_1'$ and $N \overset{C,G'(y)}{\longrightarrow} N'$, where $G'' = G \setminus \tilde{g}$. The derivation is given below:

   $$\begin{array}{lcr} \text{COM:} & \dfrac{M_1 \overset{C_1,\overline{G}x}{\longrightarrow} M_1' \quad N \overset{C,G'(y)}{\longrightarrow} N'}{M_1|N \overset{C_1\wedge C,\overline{G}x}{\longrightarrow} M_1'|N'\{x/y\}} & G\cap G' \neq \emptyset \\ \text{GNAME-RES1:} & \overline{(\nu\tilde{g})(M_1|N) \overset{C_1\wedge C,\overline{G''}x}{\longrightarrow} (\nu\tilde{g})(M_1'|N'\{x/y\})} & G'' \neq \emptyset \end{array}$$

   Since $M_1 \asymp M_2$, $M_1 \overset{C_1,\overline{G}x}{\longrightarrow} M_1'$ implies $\exists M_2'$, β, and C_2 such that $M_2 \overset{C_2,\beta}{\longrightarrow} M_2'$ and $C_1 \rhd C_2$, $\overline{G}x\sigma_{C_1} \equiv \beta\sigma_{C_1}$, $M_1'\sigma_{C_1} \asymp M_2'\sigma_{C_1}$. Moreover, there exist expressions M_{N_1}', M_{N_2}' and N_N' in normal form such that $M_1' \equiv M_{N_1}'$, $M_2' \equiv M_{N_2}'$ and $N'\{x/y\} \equiv N_N'$. Now, since $M_1'\sigma_{C_1} \asymp M_2'\sigma_{C_1}$, we know $M_{N_1}'\sigma_{C_1} \asymp M_{N_2}\sigma_{C_1}'$. Hence, by construction of S, we can conclude that the pair $((\nu\tilde{g})(M_{N_1}'\sigma_{C_1}|N_N'\sigma_{C_1\wedge C}), (\nu\tilde{g})(M_{N_2}'\sigma_{C_1}|N_N'\sigma_{C_1\wedge C})) \in S$, and hence $((\nu\tilde{g})(M_1'|N'\{x/y\})\sigma_{C_1\wedge C}, (\nu\tilde{g})(M_2'|N'\{x/y\})\sigma_{C_1\wedge C}) \in \equiv S \equiv$.

2. Case GNAME-RES1, MOBILITY:

   $(\nu\tilde{g})(M_1|N) \overset{\mathbf{true},\mu}{\longrightarrow} (\nu\tilde{g})(M_1'|N')$. The derivation is given below:

   $$\begin{array}{lc} \text{MOBILITY:} & \dfrac{}{M_1|N \overset{\mathbf{true},\mu}{\longrightarrow} M_1'|N'} \\ \text{GNAME-RES1:} & \overline{(\nu\tilde{g})(M_1|N) \overset{\mathbf{true},\mu}{\longrightarrow} (\nu\tilde{g})(M_1'|N')} \end{array}$$

   and $I(M_1|N) \implies I(M_1'|N')$ for a connectivity invariant I.

   A case analysis of M_1' and N', similar to as in Case 8 (GNAME-RES1, PNAME-RES, MOBILITY) for proof of Lemma 9 given in Appendix A, can be used to conclude that $((\nu\tilde{g})(M_1'|N'), (\nu\tilde{g})(M_2'|N')) \in \equiv S \equiv$.

3. Case GNAME-RES1, PAR:

   $(\nu\tilde{g})(M_1|N) \xrightarrow{C_1,\alpha\backslash\tilde{g}} (\nu\tilde{g})(M_1'|N)$ given $M_1 \xrightarrow{C_1,\alpha} M_1'$. The derivation is given below:

$$\begin{array}{lcr} \text{PAR:} & \dfrac{M_1 \xrightarrow{C_1,\alpha} M_1'}{M_1|N \xrightarrow{C_1,\alpha} M_1'|N} & bn(\alpha) \cap fn(N) = \emptyset \\ \text{GNAME-RES1:} & \overline{(\nu\tilde{g})(M_1|N) \xrightarrow{C_1,\alpha\backslash\tilde{g}} (\nu\tilde{g})(M_1'|N)} & \end{array}$$

   Since $M_1 \asymp M_2$, $M_1 \xrightarrow{C_1,\alpha} M_1'$ implies $\exists M_2'$, β, and C_2 such that $M_2 \xrightarrow{C_2,\beta} M_2'$ and $C_1 \triangleright C_2$, $\alpha\sigma_{C_1} \equiv \beta\sigma_{C_1}$, $M_1'\sigma_{C_1} \asymp M_2'\sigma_{C_1}$. Moreover, there exist expressions M_{N_1}' and M_{N_2}' in normal form such that $M_1' \equiv M_{N_1}'$ and $M_2' \equiv M_{N_2}'$. Now, since $M_1'\sigma_{C_1} \asymp M_2'\sigma_{C_1}$, we know $M_{N_1}'\sigma_{C_1} \asymp M_{N_2}'\sigma_{C_1}$. Hence, by construction of S, we can conclude that the pair $((\nu\tilde{g})(M_{N_1}'\sigma_{C_1}|N\sigma_{C_1}), (\nu\tilde{g})(M_{N_2}'\sigma_{C_1}|N\sigma_{C_1})) \in S$, and hence $((\nu\tilde{g})(M_1'|N)\sigma_{C_1}, (\nu\tilde{g})(M_2'|N)\sigma_{C_1}) \in \equiv S \equiv$.
   For the case $\alpha = \mu$, the conditions $I(M_1|N) \implies I(M_1'|N)$ and $I(M_2|N) \implies I(M_2'|N)$, for a connectivity invariant I, also come into effect in the above derivations.

   For the case when $\alpha \setminus \tilde{g}$ is of the form $G(x')$, we can reason in a manner similar to that for the Case 9 (GNAME-RES1, PNAME-RES, PAR) for proof of Lemma 9 given in Appendix A.

4. Case GNAME-RES2, COM:

   $(\nu\tilde{g})(M_1|N) \xrightarrow{C_1\wedge C,\tau} (\nu\tilde{g})(M_1'|N'\{x/y\})$ given $M_1 \xrightarrow{C_1,\overline{G}x} M_1'$ and $N \xrightarrow{C,G'(y)} N'$. The derivation is given below:

$$\begin{array}{lcc} \text{COM:} & \dfrac{M_1 \overset{C_1,\overline{G}x}{\longrightarrow} M_1' \quad N \overset{C,G'(y)}{\longrightarrow} N'}{M_1|N \overset{C_1\wedge C,\overline{G}x}{\longrightarrow} M_1'|N'\{x/y\}} & G \cap G' \neq \emptyset \\ \text{GNAME-RES2:} & \dfrac{}{(\nu\tilde{g})(M_1|N) \overset{C_1\wedge C,\tau}{\longrightarrow} (\nu\tilde{g})(M_1'|N'\{x/y\})} & G \setminus \tilde{g} = \emptyset \end{array}$$

Since $M_1 \asymp M_2$, $M_1 \overset{C_1,\overline{G}x}{\longrightarrow} M_1'$ implies $\exists M_2'$, β, and C_2 such that $M_2 \overset{C_2,\beta}{\longrightarrow} M_2'$ and $C_1 \triangleright C_2$, $\overline{G}x\sigma_{C_1} \equiv \beta\sigma_{C_1}$, $M_1'\sigma_{C_1} \asymp M_2'\sigma_{C_1}$. Moreover, there exist expressions M'_{N_1}, M'_{N_2} and N'_N in normal form such that $M_1' \equiv M'_{N_1}$, $M_2' \equiv M'_{N_2}$ and $N'\{x/y\} \equiv N'_N$. Now, since $M_1'\sigma_{C_1} \asymp M_2'\sigma_{C_1}$, we know $M'_{N_1}\sigma_{C_1} \asymp M'_{N_2}\sigma_{C_1}$. Hence, by construction of S, we can conclude that $((\nu\tilde{g})(M'_{N_1}\sigma_{C_1}|N'_N\sigma_{C_1\wedge C}), (\nu\tilde{g})(M'_{N_2}\sigma_{C_1}|N'_N\sigma_{C_1\wedge C})) \in S$, and hence $((\nu\tilde{g})(M_1'|N'\{x/y\})\sigma_{C_1\wedge C}, (\nu\tilde{g})(M_2'|N'\{x/y\})\sigma_{C_1\wedge C}) \in \equiv S \equiv$.

5. Case GNAME-RES2, PAR:

   $(\nu\tilde{g})(M_1|N) \overset{C_1,\tau}{\longrightarrow} (\nu\tilde{g})(M_1'|N)$ given $M_1 \overset{C_1,\overline{G}x}{\longrightarrow} M_1'$. The derivation is given below:

$$\begin{array}{lcc} \text{PAR:} & \dfrac{M_1 \overset{C_1,\overline{G}x}{\longrightarrow} M_1'}{M_1|N \overset{C_1,\overline{G}x}{\longrightarrow} M_1'|N} & \\ \text{GNAME-RES2:} & \dfrac{}{(\nu\tilde{g})(M_1|N) \overset{C_1,\tau}{\longrightarrow} (\nu\tilde{g})(M_1'|N)} & G \setminus \tilde{g} = \emptyset \end{array}$$

   Since $M_1 \asymp M_2$, $M_1 \overset{C_1,\overline{G}x}{\longrightarrow} M_1'$ implies $\exists M_2'$, β, and C_2 such that $M_2 \overset{C_2,\beta}{\longrightarrow} M_2'$ and $C_1 \triangleright C_2$, $\overline{G}x\sigma_{C_1} \equiv \beta\sigma_{C_1}$, $M_1'\sigma_{C_1} \asymp M_2'\sigma_{C_1}$. Moreover, there exist expression M'_{N_1} and M'_{N_2} in normal form such that $M_1' \equiv M'_{N_1}$ and $M_2' \equiv M'_{N_2}$. Now, since $M_1'\sigma_{C_1} \asymp M_2'\sigma_{C_1}$, we know $M'_{N_1}\sigma_{C_1} \asymp M'_{N_2}\sigma_{C_1}$. Hence, by construction of S, we can conclude that $((\nu\tilde{g})(M'_{N_1}\sigma_{C_1}|N\sigma_{C_1}), (\nu\tilde{g})(M'_{N_2}\sigma_{C_1}|N\sigma_{C_1})) \in S$, and hence $((\nu\tilde{g})(M_1'|N)\sigma_{C_1}, (\nu\tilde{g})(M_2'|N)\sigma_{C_1}) \in \equiv S \equiv$.

By considering the 5 cases and their symmetric counterparts due to the commutativity of '|' operator, all possible derivations are covered and we conclude that S is a symbolic bisimulation up to $\equiv$. Following Lemma 5 we conclude that S is a symbolic bisimulation. Therefore, $\asymp$ is preserved by restriction of gnames and the parallel operator for ω_0-expressions in normal form.

This proof is *complete* because at each proof step all possible transitions from an expression are considered to find its derivatives. The five cases along with their symmetric counterparts for the parallel operator cover all the derivation possibilities. All the possible transitions at the node level (pertaining to broadcast send/receive, silent action, and mobility) are taken into account through the derivations given in the proof. $\square$

Theorem 22 (Congruence for Symbolic Bisimulation for the ω_0-Calculus) $\asymp$ *is a congruence for the ω_0-calculus; i.e., for all $M_1, M_2 \in \mathbf{N}$, the following hold:*

(i) $M_1 \asymp M_2$ implies $\forall g \in \mathbf{Gn} : (\nu g)M_1 \asymp (\nu g)M_2$; and

(ii) $M_1 \asymp M_2$ implies $\forall N \in \mathbf{N} : M_1|N \asymp M_2|N$.

Proof: Let $M_1 \equiv M_{N_1}$ and $M_2 \equiv M_{N_2}$, where M_{N_1} and M_{N_2} are in normal form. Then the following holds:

- $M_1 \asymp M_2$ implies $M_{N_1} \asymp M_{N_2}$ (from Definition 2 and Lemma 5). $M_{N_1} \asymp M_{N_2}$ implies $\forall g \in \mathbf{G_n}$: $(\nu g)M_{N_1} \asymp (\nu g)M_{N_2}$ (by Lemma 21), which in turn implies $(\nu g)M_1 \asymp (\nu g)M_2$ (by Def. 2 and Lemma 5). Therefore, whenever $M_1 \asymp M_2$ then $(\nu g)M_1 \asymp (\nu g)M_2$.

- $M_1 \asymp M_2$ implies $M_{N_1} \asymp M_{N_2}$ (from Definition 2 and Lemma 5). $M_{N_1} \asymp M_{N_2}$ implies for any $N \in \mathbf{N}$, and $N \equiv N_N$ where, $N_N \in \mathbf{N_{nf}}$: $(M_{N_1}|N_N) \asymp (M_{N_2}|N_N)$ (by Lemma 21), which in turn implies $(M_1|N) \asymp (M_2|N)$ (by Def. 2 and Lemma 5). Therefore, whenever $M_1 \asymp M_2$ then $(M_1|N) \asymp (M_2|N)$.

$\asymp$ is preserved by all the node contexts for the ω_0-calculus. Hence, $\asymp$ is a congruence for the ω_0-calculus. $\square$

Appendix C

Proof of Theorem 20

Proposition 23 *Let M and N be two ω_0 node expressions, δ a set of substitutions on pnames. Then for all formulas φ following holds:*

$$M \equiv N \implies (M \models_\delta \varphi \Leftrightarrow N \models_\delta \varphi)$$

Theorem 20. Let M and N be two node expressions, and δ a set of substitutions. Then for all formulas φ, the following holds:

$$M \,|\, N \models_\delta \varphi \;\Leftrightarrow\; M \models_\delta \Pi(N)(\varphi)$$

Proof: The proof proceeds by induction on the size of the node expression and the formula. We go through each of the rules of the compositional model checker to complete the proof.

- Rule 1: The theorem is trivially true when φ is a propositional constant (*tt* or *ff*).
- Rule 2: $\varphi = (x = y)$ or $\varphi = (x \neq y)$.

  $M \,|\, N \models_\delta (x = y) \;\Leftrightarrow\; \delta \models x = y \;\Leftrightarrow\; M \models_\delta \Pi(N)(\varphi)$
- Rules 3 and 4: $\varphi = \varphi_1 \vee \varphi_2$

  $\quad M \,|\, N \models_\delta \varphi_1 \vee \varphi_2$

  $\Leftrightarrow\; M \,|\, N \models_\delta \varphi_1 \vee M \,|\, N \models_\delta \varphi_2$

$\Leftrightarrow\ M \models_\delta \Pi(N)(\varphi_1) \vee M \models_\delta \Pi(N)(\varphi_2)$
$\Leftrightarrow\ M \models_\delta \Pi(N)(\varphi_1) \vee \Pi(N)(\varphi_2)$
$\Leftrightarrow\ M \models_\delta \Pi(N)(\varphi_1 \vee \varphi_2)$

The case for conjunctive formula (Rule 4) follows along the same lines.

- Rule 6: $\varphi = \exists x.\psi$.

  $M \mid N \models_\delta \exists x.\psi \ \Leftrightarrow\ M \models_\delta \Pi(N)(\exists x.\psi) \ \Leftrightarrow\ M \models_\delta \exists x.\Pi(N)(\psi)$, if $x \notin fn(N)$.

  The case for $\varphi = \forall x.\psi$ is similar.

- Rule 7: Node expression $N = \mathbf{0}$
  $M \mid \mathbf{0} \models_\delta \varphi \ \Leftrightarrow\ M \models_\delta \varphi \ \Leftrightarrow\ M \models_\delta \Pi(\mathbf{0})(\varphi)$

- Rule 8: Process definition $A(\overrightarrow{x}) \overset{def}{=} P$

  $M \mid A(\overrightarrow{x}) : G \models_\delta \varphi$
  $\Leftrightarrow M \mid P : G \models_\delta \varphi$
  $\Leftrightarrow M \models_\delta \Pi(P : G)(\varphi)$
  $\Leftrightarrow M \models_\delta \Pi(A(\overrightarrow{x}) : G)(\varphi)$

- Rule 9: Consider the case $N = N_1 \mid N_2$
  $M \mid (N_1 \mid N_2) \models_\delta \varphi$
  $\Leftrightarrow M \mid N_1 \models_\delta \Pi(N_2)(\varphi)$ induction hypothesis
  $\Leftrightarrow M \models_\delta \Pi(N_1)(\Pi(N_2)(\varphi))$ induction hypothesis
  $\Leftrightarrow M \models_\delta \Pi(N_1 \mid N_2)(\varphi)$

- Rule 10: $N = (\nu g)N'$ where g is a local name.

  $M \mid (\nu g)N' \models_\delta \varphi$
  $\Leftrightarrow (\nu g')(M \mid N'\{g'/g\}) \models_\delta \varphi$ from Proposition 23
  where $g' \notin fgn(M) \cup fgn(N') \cup n(\varphi)$
  $\Leftrightarrow M \mid N'\{g'/g\} \models_\delta \varphi_{+\{g'\}}$ from Proposition 19
  $\Leftrightarrow M \models_\delta \Pi(N'\{g'/g\})(\varphi_{+\{g'\}})$ induction hypothesis
  $\Leftrightarrow (\nu g')M \models_\delta \Pi(N'\{g'/g\})(\varphi_{+\{g'\}})$ from Proposition 23
  $\Leftrightarrow M \models_\delta (\Pi(N'\{g'/g\})(\varphi_{+\{g'\}}))_{-\{g'\}}$ from Proposition 19
  $\Leftrightarrow M \models_\delta \Pi((\nu g)N')(\varphi)$

- Rules 12 and 13: $N = (P_1 + P_2){:}G$

$$\begin{aligned}
&M \,|\, (P_1 + P_2){:}G \models_\delta \langle\alpha\rangle\varphi \\
\Leftrightarrow\ & M' \,|\, (P_1 + P_2){:}G \models_\delta \varphi, \text{ if } M \xrightarrow{b,\alpha} M' \wedge (\delta \models b) \\
&\vee M \,|\, P_1{:}G \models_\delta \langle\alpha\rangle\varphi \\
&\vee M \,|\, P_2{:}G \models_\delta \langle\alpha\rangle\varphi \\
\Leftrightarrow\ & M' \models_\delta \Pi(P_1 + P_2{:}G)(\varphi) \vee M \models_\delta \Pi(P_1{:}G)(\langle\alpha\rangle\varphi) \vee M \models_\delta \Pi(P_2{:}G)(\langle\alpha\rangle\varphi) \\
\Leftrightarrow\ & M \models_\delta \langle\alpha\rangle\Pi((P_1 + P_2){:}G)(\varphi) \vee M \models_\delta \Pi(P_1{:}G)(\langle\alpha\rangle\varphi) \vee M \models_\delta \Pi(P_2{:}G)(\langle\alpha\rangle\varphi) \\
\Leftrightarrow\ & M \models_\delta \langle\alpha\rangle\Pi((P_1 + P_2){:}G)(\varphi) \vee \Pi(P_1{:}G)(\langle\alpha\rangle\varphi) \vee \Pi(P_2{:}G)(\langle\alpha\rangle\varphi)
\end{aligned}$$

The case for $[\alpha]\varphi$ is similar.

- Rule 11: $N = a.P{:}G$ and $\varphi = \langle\alpha\rangle\varphi'$

Consider the case when $\alpha = \overline{G'}\{y\}$.

$$\begin{aligned}
&M \,|\, a.P{:}G \models_\delta \langle\overline{G'}\{y\}\rangle\varphi' \\
\Leftrightarrow\ & M' \,|\, a.P{:}G \models_\delta \varphi'\{x/y\} \text{ if } M \xrightarrow{b,\overline{G''}x} M' \wedge (\delta \models b) \wedge G'' = G' \\
&\vee M \,|\, P{:}G \models_\delta \varphi'\{x/y\} \text{ if } a = \overline{b}x \wedge G = G'
\end{aligned}$$

Considering the first disjunct,

$$\begin{aligned}
&M' \,|\, a.P{:}G \models_\delta \varphi'\{x/y\} \\
\Leftrightarrow\ & M' \models_\delta \Pi(a.P{:}G)(\varphi'\{x/y\}) \\
\Leftrightarrow\ & M \models_\delta \langle\overline{G'}\{y\}\rangle\Pi(a.P{:}G)(\varphi') \text{ where } y \notin fn(a.P{:}G)
\end{aligned}$$

Considering the second disjunct,

$$\begin{aligned}
&M \,|\, P{:}G \models_\delta \varphi'\{x/y\} \\
\Leftrightarrow\ & M \models_\delta \Pi(P{:}G)(\varphi'\{x/y\})
\end{aligned}$$

Finally,

$$\begin{aligned}
&M \models_\delta \Pi(a.P{:}G)(\langle\overline{G'}\{y\}\rangle\varphi') \\
\Leftrightarrow\ & M \models_\delta \langle\overline{G'}\{y\}\rangle\Pi(a.P{:}G)(\varphi') \text{ where } y \notin fn(a.P{:}G) \\
&\vee\ M \models_\delta (\Pi(P{:}G)(\varphi'\{x/y\}) \wedge G = G') \\
\Leftrightarrow\ & M \models_\delta \langle\overline{G'}\{y\}\rangle\Pi(a.P{:}G)(\varphi') \text{ where } y \notin fn(a.P{:}G) \\
&\qquad \vee\ (\Pi(P{:}G)(\varphi'\{x/y\}) \wedge G = G')
\end{aligned}$$

Similarly, we can prove for other cases of α and for actions with box modality, the dual of rule 11.

- Rules 5, A and B: Let $X(\overrightarrow{z}) =_\sigma \varphi$.

  $$\begin{aligned} & M \mid N \models_\delta X(\overrightarrow{z}) =_\sigma \varphi \\ \Leftrightarrow\ & M \models_\delta \Pi(N)(X(\overrightarrow{z})) =_\sigma \Pi(N)(\varphi) \\ \Leftrightarrow\ & M \models_\delta (X_N(\overrightarrow{z})) =_\sigma \Pi(N)(\varphi) \\ \Leftrightarrow\ & M \models_\delta \Pi(N)(X(\overrightarrow{z}) =_\sigma \varphi) \end{aligned}$$

  For finite-control node expressions, the number of different substitutions to be considered for the parameters $\overrightarrow{z}$ are finite, thus leading to termination of the transformation.

$\square$