The CSI Journal on Computer Science & Engineering

The CSI Journal on
Computer Science and Engineering
Vol. 15, No. 2, 2018
Pages 44-52
Regular Paper

Verification of Mobile Ad hoc Network Processes with Data

Fatemeh Ghassemi

School of Electrical and Computer Engineering, University of Tehran

Abstract

Topology-dependent behavior of wireless communication makes the modeling and verification of Mobile Ad hoc Networks (MANETs) more complicated. Reliable Restricted Broadcast Process Theory (RRBPT) was introduced to specify and analyze MANETs in an algebraic approach. Constrained Action Computation Tree Logic (CACLT), interpreted over the semantics of RRBPT, allows to specify topology-dependent properties of MANETs. However, model checking of CACTL formulae is restricted to MANETs with a small number of nodes or finite datatypes. Having an algebraic specification, the problem of model checking of CACTL properties can be reduced to solving Boolean equations, as an intermediate formalism. This technique has been followed in the mCRL2 toolset to verify μ-calculus properties extended with data. Having a sound translation from RRBPT to mCRL2, we can use its toolset to verify data-dependent properties of MANET processes with infinite state, data-dependent behaviors. By treating the topology-dependent behavior of CACTL modals as data, we can also verify topology-dependent behavior of MANETs. To this aim, we provide a sound translation from CACTL formulae to first order modal μ-calculus expressions. Our translation takes advantage of the formal treatment of data and parametrized propositional variables in the mCRL2 process and property specifications respectively to address the topology-formulae part of CACTL expressing multi-hop constrains over the topology.
Keywords: Model checking, Ad hoc networks, Topology-dependent property, State-space explosion.

1. Introduction

Model checking is a well-known formal technique to automatically verify systems, and it has been successfully applied to many application areas. There are lots of tools such as CADP [1], Spin [2], UPPAAL [3], etc. that support this technique to analyze systems with one-click without the user intervention. These tools mostly inspect the given properties specified by a temporal logic over Labelled Transition System (LTS)-based models, generated by compilers which transform a given specification expressed by a high-level modeling language into its formal semantics.

Application of this technique is restricted by the size of the underlying semantic model. The semantic model of complex systems is subject to explosion. The occurrence of this so called *state-space explosion problem* is obvious when the size of data domains are infinite or the number of communicating components is large. To minimize this problem, many approaches have been provided. Some approaches focus on reducing the resulting semantic model such that the properties under consideration are preserved, namely partial-order reduction [4], symmetric reduction [5], etc. These approaches can be automatically applied. Some dedicated techniques like regular expressions [6] and queue representations [7] for communications protocols, Presburger arithmetic [8] for networks and counter abstraction [9] for parameterized systems can be followed by the modeler to result in a smaller model by using special data structures [15]. However, such models cannot be used uniquely to verify different properties or systems with arbitrary data structures.

Process algebra frameworks are used to model systems in a compositional way by providing a set of algebraic operators. They provide textual representations of LTSs and hence, make it possible to represent processes with infinite states and even more automatically manipulate LTSs at the textual level. Algebraic frameworks can be orthogonally extended with abstract datatypes [26] to formally treat data in a unified framework. The mCRL2 toolset [10] is an algebraic

framework that enables modeling systems and data in a unified way. The process algebraic language of this framework is an extension of ACP [11]. This tool supports model checking of μ-calculus [12] properties by taking advantage of the textual representations of processes.

The problem of model checking of μ-calculus properties has been reduced in [13] to solving Boolean equations, as an intermediate formalism, derived from the semantic model and the property. Each Boolean variable has one of the values true or false, in contrast to the variables of the original property, and so Boolean equations are decidable to be solved. This technique was further developed in [14] by considering data in properties and using the textural representation of processes to derive the so-called Parametrized Boolean Equations (PBEs). The Boolean variables are parametrized by data sorts to handle parametrized variables of μ-calculus properties. The algorithm which solves PBEs has been introduced in [15,24] and implemented in the mCRL2 toolset.

The topology-dependent behavior of wireless communications makes the properties of Mobile Ad hoc Networks (MANETs) protocols depend on the underlying topology. For instance, the important property of packet delivery in routing or information dissemination protocols in the context of MANETs becomes: "if there exists an end-to-end route between two nodes A and B for a sufficiently long period of time, then packets sent by A will eventually be received by B" [16]. We introduced the temporal Constrained Action Computation Tree Logic (CACTL) in [17,18] to specify properties of MANETs taking constraints over the underlying topology into account. The CACTL operators are interpreted over Constrained Labeled Transition Systems (CLTSs) [19] while its path quantifier *all* is parameterized by a multi-hop constraint over the topology. The CLTS semantic model captures the topology-dependent behavior of wireless communication by annotating transitions by *network constraints* to restrict each behavior for a set of topologies, those that satisfy the given network constraint. The process algebra framework Reliable Restricted Broadcast Process Theory (RRBPT) [18], tailored for the specification of MANET protocols, derives CLTSs from RRBPT terms.

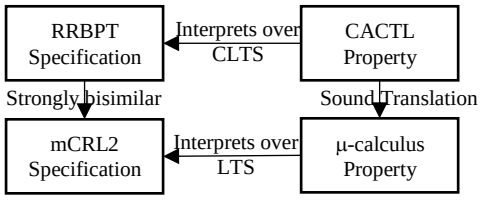

Figure 1: Verification Tool chain

The model checking algorithm of CACTL has been introduced in [18]. However, its application is restricted to MANETs in which infinite datatypes are absent or the size of the network is small. To take advantage of algebraic framework, we provide a sound translation from CACTL formulae to first order modal μ-calculus expressions by adjusting propositional variable parameters to appropriately capture the topology-dependent behavior of the path quantifier *all*. We have previously defined in [19] how a RRBPT specification can be transformed into its behavioral equivalent (strongly bisimilar modulo renaming [20]) mCRL2 specification. Therefore, by providing the translation of CACTL, the chain of verification tools gets complete as illustrated in Figure 1. We prove that our translation is sound.

2. Preliminaries

As CACTL are interpreted over CLTSs, we first introduce CLTSs before explaining CACTL. Furthermore, we briefly explain modal μ-calculus and how the model checking problem of its formulae is reduced to solving PBEs. We also explain how data is formally specified in mCRL2 as it is required in our translation.

2.1. Constrained Labelled Transition Systems (CLTSs)

Let Loc be the set of locations, denoting the network addresses of nodes in a network. A topology γ is defined as a function $Loc \rightarrow \mathbb{P}(Loc)$ which assigns a set of locations as the neighbors of each location.

Constrained labeled transition systems were introduced as the semantic model for the operational behavior of MANETs [19]. A transition label is a pair of an action and a network constraint. A *network constraint* expresses a set of on-hop link constraints over the underlying topology. In other words, it restricts the behavior to the set of topologies which satisfy the one-hop constraints. A network constraint c is specified by a set of connectivity pairs $\rightarrow: Loc \times Loc$ and disconnectivity pairs $\nrightarrow: Loc \times Loc$ such that it does not contain two inconsistent pairs like $l \rightarrow l'$ and $l \nrightarrow l'$. Let $\mathbb{C}(Loc)$ denote the set of network constraints that can be defined over the network addresses in Loc and $constraint(\gamma) = \cup l \in Loc(\{l \rightarrow l' | l' \in \gamma(l)\} \cup \{l \nrightarrow l' | l' \notin \gamma(l)\})$ expresses a topology in terms of its (dis)connectivity relations. A network constraints represents a set of topologies satisfying the constraints of c, formally $topology(c)=\{\gamma | c \subseteq constraint(\gamma)\}$. For instance, considering Loc={A,B,C,D}, the network constrain {A→C, A↛B} defines all the topologies that C is connected to A and hence, can receive data from A (the direction of the pair represents the direction of communication) while B is disconnected from A. The negation $\neg c$ of network constraint c is trivially defined by negating all its (dis)connectivity pairs. A network constraint c *conforms* to the network constraint c′ if they do not have any (dis)connectivity relation inconsistent with each other, i.e., $c \cap \neg c' = \emptyset$. Due to the pair A↛B, {A→C, A↛B} does not conform to {A→B, A→D}.

Let Act_τ be the set of actions, including the silent action τ, ranged over by η. Formally a CLTS is defined by the quadruple $\langle S, \Lambda, \rightarrow, s_0 \rangle$, where S is the set of states, $\Lambda \subseteq \mathbb{C}(Loc) \times Act\tau$, $\rightarrow \subseteq S \times \Lambda \times S$ a transition relation, and $s_0 \in S$ the initial state. A transition $(s,(c,\eta),s') \in \rightarrow$ indicates that a MANET protocol in state s with an underlying topology $\gamma \in topology(c)$ can perform the action η to evolve to state s′.

A path π from a state $t_0 \in S$ in a CLTS is a sequence of transitions $t_0\ (c_1,\eta_1)\ t_1\ (c_2, \eta_2)\ t_2 \ldots$ where $\forall i \geq 1((t_{i-1}, (c_i,\eta_i), t_i) \in \rightarrow)$. A path is said to be *maximal* if it either is infinite or ends in a deadlock state, meaning that there are no outgoing transitions. We use π_i^s to denote i^{th}-state and π_i^η and π_i^C to denote the action and network constraint of (c_i,η_i)-transition, respectively.

2.2. Constraint Action Computation Tree Logic (CACTL)

CACTL is an extension of Action Computation Tree Logic (ACTL) [21], a general framework for verifying properties in process algebra, enriched with unless operator following the approach of [22]. The temporal logic for MANETs includes *until* and *unless* operators, preceded by either the path quantifier *exists* **E** or *all* **A** decorated with multi-hop constraints. The grammar of CACTL has four levels: *state formula* φ, *path formula* ψ, *action formula* χ, and *topology formula* Γ, where the temporal operators *until* **U** and *unless* **W** have been parametrized by action formula and should be preceded by a path quantifier, and the path quantifier *all* has been parametrized by a topology formula:

$$\varphi ::= true \mid \varphi \wedge \varphi' \mid \mathbf{E}\ \psi \mid \mathbf{A}^{\Gamma}\varphi$$
$$\psi ::= \varphi_{\chi}\mathbf{U}_{\chi'}\varphi' \mid \varphi_{\chi}\mathbf{W}_{\chi'}\varphi'$$
$$\chi ::= true \mid\ \eta \in Act_{\tau} \mid \neg\chi \mid \chi \wedge \chi'$$
$$\Gamma ::= true \mid \mapsto l' \mid \Gamma \wedge \Gamma'$$

Each topology formula expresses constraints made of multi-hop connections over the topology. Each topology formula identifies a set of topologies in which the multi-hop connections hold. Therefore, we lift the notion of *topology*(Γ) to represent the topologies identified by a topology formula. A network constraint $c' \in \mathbb{C}(Loc)$ is said to be invalid for (or violate) Γ, if $\forall \gamma \in topology(\Gamma)(constraint(\gamma) \cap \neg c' \neq \emptyset)$. We say a path violates Γ in state t_i, if the accumulated network constraints before reaching to t_i violates Γ, i.e., $\forall \gamma \in topology(\Gamma)(constraint(\gamma) \cap \neg \cup_{1 \le j \le i} c_j' \neq \emptyset)$. A path is said to be invalid for Γ (or violates Γ) if it is invalid in some state over the path. Topology formulae are used in the path quantifier *all* to restrict the inspection of its path formula to the paths that do not violate Γ. Intuitively, these paths contain some permanent multi-hop communication links leading to the given topology formula. For example, the topology formula $A \dashrightarrow C$ restricts paths to those over which at least either the single-hop communication link A→C or the multi-hop communication links A→B and B→C permanently hold.

To verify a property with regard to a smaller set of topologies, state and path formulae in CACTL are interpreted with regard to a network constraint $\zeta \in \mathbb{C}(Loc)$, and so only transitions conforming to ζ are considered. This parameter indicates to the permanent (non-)existence of one-hop communication links. So for a given CLTS $T = \langle S, \Lambda, \rightarrow, s_0 \rangle$, only its maximal ζ-paths are considered, i.e., paths that all the network constrains over the path conforms to ζ, i.e., $\forall i \ge 1\ (\neg c_i \cap \zeta = \emptyset)$.

A state t that satisfies a CACTL formula φ is called a φ-state, and a transition with an action from χ is called a χ-transition. A χ-transition that ends in a φ-state is called a (χ,φ)-transition. A path with an initial φ-state and only (χ,φ)-transitions is called a (χ,φ)-path. The maximal subpath of a path with an initial φ-state and only (χ,φ)-transitions is called (χ,φ)-subpath of the path. The until operator $\varphi_{\chi}\mathbf{U}_{\chi'}\varphi'$ is satisfied by maximal ζ-paths that start at an initial φ-state and perform a finite sequence of (χ,φ)-transitions, until a (χ',φ')-transition is performed. The unless (or weak until) operator $\varphi_{\chi}\mathbf{W}_{\chi'}\varphi'$ extends $\varphi_{\chi}\mathbf{U}_{\chi'}\varphi'$ by moreover allowing infinite ζ-paths which are a (χ,φ)-path. The path quantifier **E** requires that the given path formula is satisfied for at least one maximal ζ-path starting in the given state, while $\mathbf{A}^{\Gamma}$ requires the property holds for all ζ-paths that their (χ,φ)-subpath are valid for the topology formula Γ.

For instance, for the set of locations Loc={A,B,C,D}, the formula $\mathbf{A}^{A \dashrightarrow B \wedge B \dashrightarrow A}\ true_{\{init \wedge \tau\}}\mathbf{U}_{\{succ\}}true$ expresses that after the action "init" or a sequence of communications, abstracted by τ, eventually the action "succ" is performed if A and B are connected (not essentially directly) to each other. The maximal path t_0 ({},init) t_1 ({A→B},τ) t_2 ({B↠A,B→C},τ) t_3 ({},succ) t_4 of the CLTS in Figure 2 satisfies the given formula due to the occurrence of "succ". It should be noted that the accumulated network constraints before the occurrence of "succ" satisfy the topology formula $A \dashrightarrow B \wedge B \dashrightarrow A$, as the single-hop links A→B, B→C,C→A are not invalidated by any network constraint. Furthermore, the maximal path t_0 ({},init) t_1 ({A→B},τ) t_2 ({B↠A,B→C},τ) t_3 ({C↠A,C→B},τ) t_1 ({A→B},τ) t_2 … violates $A \dashrightarrow B \wedge B \dashrightarrow A$, and so it is not considered in the inspection of the path formula ($true_{\{init \wedge \tau\}}\mathbf{U}_{\{succ\}}true$). In contrast, the path t_0 ({},init) t_1 ({A→B},τ) t_2 ({A→B},τ) t_2 … is valid, but no "succ" action is occurred. Therefore, t_0 does not satisfies the given **AU** formula.

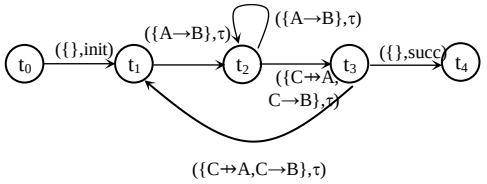

Figure 2: A simple CLTS

2.3. Verification of Temporal Properties of Processes in a Setting with Data

As algebraic framework can be orthogonally extended with data, the modal logic of μ-calculus was extended with data in [14] to support verification of processes with data. The logic contains first order formula, modalities, and fixed point operators parametrized by data variables to specify temporal properties depending on data. This logic is interpreted over the textual representation of process terms in a special format, and thus, the satisfaction of a temporal formula by a process term is reduced to the satisfaction of a first-order Boolean formula containing parametrized fixed point operators. In the following, we briefly explain first-order modal μ-calculus and its verification using Boolean formula as an intermediate formalism.

Modal μ-calculus logic has a two level grammar: *state formula* φ and *action formula* χ:

$$\varphi ::= b \mid Y(e) \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \langle\chi\rangle\varphi \mid \exists y{:}D.\varphi \mid (\mu Y(y{:}D).\varphi)(e)$$
$$\chi ::= true \mid\ \eta(a) \in Act_{\tau} \mid \neg\chi \mid \chi \wedge \chi' \mid \exists y{:}D.\ \eta$$

where b is a Boolean expression, Y is a parametrized propositional variable, y is a data variable of type D. The diamond modal operator $\langle\chi\rangle\varphi$ denotes states with actions satisfying χ. The box model operator $[\chi]\varphi = \neg\langle\chi\rangle\neg\varphi$ and the

maximal fixed point operator $(\nu Y(y{:}D).\varphi)(e) = \neg(\mu Y(y{:}D).\neg\varphi[\neg Y/Y])(e)$ can be defined, where $\varphi[\neg Y/Y]$ denotes the syntactic substitution of $\neg Y$ for Y in φ. Furthermore as CACTL, χ represents an action formula, where $\eta(a)$ is a parametrized action.

First-order Boolean formulae are used as an intermediate formulism for the verification purpose. The grammar of these formulae is shown below:

$$\varphi ::= b \mid Z(e) \mid \neg\psi \mid \psi \wedge \psi' \mid \exists z{:}D.\ \psi \mid (\mu Z(z{:}D).\ \psi)(e)$$

where b is a Boolean expression, Z is a parametrized Boolean variable. The derived Boolean, first-order, modal, and fixed point operator are defined as usual.

To explain how a logic formula in μ-calculus is transformed into a first-order Boolean formula, consider the formula $\mu X.\nu Y.\langle b\rangle X \vee \langle a\rangle Y$ expressing that an infinite sequence of "a" and "b" actions exists, where the total number of "b" is finite. Assume that the property is examined for the given LTS in Figure 3.

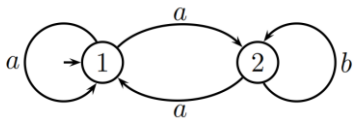

Figure 3: A simple LTS with Act={a,b}[23]

A pair of Boolean variables X_i and Y_i, where $i \in \{1,2\}$, is defined for each state of LTS: X_i represents that the formula $\mu X.\nu Y.\langle b\rangle X \vee \langle a\rangle Y$ is valid in state i while Y_i represents that the formula $\nu Y.\langle b\rangle X \vee \langle a\rangle Y$ is valid in state i. For the formula $\nu Y.\langle b\rangle X \vee \langle a\rangle Y$, we must find the appropriate formulae for the state 1 and state 2. Since state 1 has two a-transitions one leading to the state 1 again and another leading to the state 2, so the subformula $\langle a\rangle Y$ holds if Y_1 and Y_2 hold, respectively. With the same discussion, the state 2 has one b-transition entering the state 2 again. Thus, subformula $\langle b\rangle X$ holds if X_2 holds. The state 2 has another a-transition leading to the state 1. The subformula $\langle a\rangle Y$ holds if Y_1 holds. Thus, the corresponding first-order Boolean formulae for $\nu Y.\langle b\rangle X \vee \langle a\rangle Y$ consist of $\nu Y_1.Y_1 \vee Y_2$ and $\nu Y_2.X_2 \vee Y_1$. For the formula $\mu X.\nu Y.\langle b\rangle X \vee \langle a\rangle Y$, the appropriate formulae for X_1 and X_2 are trivially defined by $\mu X_1.\ Y_1$ and $\mu X_2.Y_2$, respectively. The algorithm that approximately solves the Boolean equation was introduced in [13] which is based on Gauβ elimination. In this algorithm, noting to the fact that $\sigma X_i.\psi_i$ represents $X_i = \psi_i$, where $\sigma \in \{\mu,\nu\}$, X_i is replaced by "true" in ψ_i if $\sigma = \nu$ and otherwise to "false". Then, X_i is replaced by $\psi_i' = \psi_i[\text{true}/X_i]$ in other equations. Reducing the number of equations, makes solving the PBEs much simpler. Giving priority to outer fixed point symbols in model formulae, the total Boolean equation system becomes:

$$\mu X_1.\ Y_1$$
$$\mu X_2.Y_2$$
$$\nu Y_1.Y_1 \vee Y_2$$
$$\nu Y_2.X_2 \vee Y_1$$

First Y_2 is assigned to "true". Thus, $Y_2 = X_2 \vee Y_1$ by substituting $X_2 \vee Y_1$ for Y_2 in the equation of Y_1. Again by assigning Y_1 to "true", $Y_1 = \text{true} \vee X_2 = \text{true}$. Hence, "true" substitutes for Y_1 in the remaining equations. As X_1=true as shown in Table 1, the property indeed is satisfied by the LTS given in Figure 3.

Table 1: Solving the equations

Substitutions	$X_2 \vee Y_1 / Y_2$	true/Y_1
$\nu Y_2.X_2 \vee Y_1$ $\nu Y_1.Y_1 \vee Y_2$ $\mu X_2.Y_2$ $\mu X_1.\ Y_1$	$\nu Y_1.Y_1 \vee X_2$ $\mu X_2.\ Y_1 \vee X_2$ $\mu X_1.Y_1$	$\mu X_2.$ true $\mu X_1.$true

2.4. mCRL2 Framework

Data is an integral part of the process algebra mCRL2 language, which makes it more expressive than other algebra. The model of a system is broken into two specification parts: data part and process part. Data part consists of the declaration of datatypes used as parameter of processes or actions and the required operations to manipulate data values. The process part specifies the behavior of entities. As we take advantage of the formal treatment of data in mCRL2, we briefly explain how a datatype is defined.

A datatype is defined by the keyword "sort" with a set of constructors by which the elements of a domain can be created using the "struct" keyword. Required operation can be defined by the "map" keyword. The behavior of operations are defined by a set of equations using "eqn" keyword. Each equation defines the behavior of a function in terms of its parameter structures. The variables used in equation should be defined using "var" keyword. For instance, Loc and network constraints are defined as datatypes called "Loc" and "NC", respectively in Figure 4.

```
sort Loc = struct A | B| C| D ;
sort Pair = struct conn(Loc#Loc) |
                   disconn(Loc#Loc);
sort NC= struct empty | nx(Pair# NC);
map
pinclude : Pair# NC ->Bool;
union:NC#NC->NC;
neg : NC -> NC;
hasComm, conform: NC # NC ->Bool;

var
l1,l2: Loc; n1,n2: NC;p,p1,p2:Pair;
eqn
neg(empty) = empty;
neg(nx(conn(l1,l2),n1))=
    nx(disconn(l1,l2),neg(n1));
neg(nx(disconn(l1,l2),r1)=
    nx(conn(l1,l2),neg(n1));
pinclude(p,empty)=false;
pinclude(p1,nx(p2,n2))=
    pr1==pr2||pinclude(pr1,nr2);
hasComm(empty,n1)=false;
hasComm(nx(p,n1),n2)=
   pinclude(p,n2)||hasComm(n1,n2);
conform(nr1,nr2)=!hasComm(n1,neg(n2));
```

Figure 4. Declaration of datatypes in mCRL2

The datatype Loc has four constants A, B, C, D. The elements of network constrains are constructed by "empty" and "nx" which adds a (dis)connectivity pair to a network constraint. Connectivity pairs are defined as "Pair" datatype. The network

constraint {A→B,B↛C} can be represented by the data term nx(conn(A,B),nx(disconn(B,C),empty)). The function "neg(c)" computes ¬c while "union(c_1,c_2)" integrates the (dis)connectivity pairs of c_1 and c_2. The inclusion of a (dis)connectivity pair p in a network constraint c is examined by the function "pinclude(p,c)" while "hasComm(c_1,c_2)" inspects if the network constraints c_1 and c_2 has a (dis)connectivity pair in common. The conformance of a network constraint to another one is examined by the function "conform" using the "neg" and "hasComm" functions.

3. Translation of CACTL to First-Order μ-Calculus

We formally define the operators of CACTL in terms of μ-calculus expressions. We first explain the semantic of each formula interpreted over CLTSs and then express its translation interpreted over LTSs.

The RRBPT specifications can be encoded into mCRL2 such that the resulting LTSs have an η(c)-transition for each (c,η)-transition of CLTS, as explained in [18]. As the approach of encoding RRBPT specifications into mCRL2 is out of the scope of this paper, we assume that for each MANET specification, specified by RRBPT which derives a CLTS, there is a specification in mCRL2 which derives LTS, denoted by TR(CLTS).

We remark that CLTS and TR(CLTS) are *strongly bisimilar modulo renaming* [20]. A binary relation over the states is a strong bisimulation modulo renaming for a bijective renaming function f:Act_1→Act_2 if it satisfies:

- ∀a∈Act_1, if s R t and $s\xrightarrow{a}s'$, then $\exists t'.\ t\xrightarrow{f(a)}t'$ and s′Rt′;
- ∀a∈Act_2, if s R t and $t\xrightarrow{a}t'$, then $\exists t'.\ t\xrightarrow{f^{-1}(a)}t'$ and s′Rt′;

In this encoding, network constraint is treated as a datatype as specified in Figure 4 with a set of operations to manipulate its values, and to each action of CLTS, a parameter of network constraint has been added. We also model topology formulae as a datatype, as explained in following, and define the required operations on network constraint and topology formula datatypes to implement the topology-dependent behavior of CACTL operators.

We explain our translation in terms of the possible structure of state formulae. To this aim, we define a mapping TR from the state formulae of CACTL to μ-calculus expressions. Trivially, the state formula "true" is translated to "true", i. e., TR(true)=true. Furthermore, TR(φ₁∧φ₂)=TR(φ₁)∧TR(φ₂) and TR(¬φ)=¬TR(φ).

3.1. Translation of $A^{\Gamma}\varphi_{\chi}U_{\chi'}\varphi'$

The state formula $\mathbf{A}^{\Gamma}\varphi_{\chi}\mathbf{U}_{\chi'}\varphi'$ defines φ-states that all of their ζ-paths after a sequence of (χ,φ)-transitions either violate the topology formula Γ or, a (χ′,φ′)-transition is met. Formally the state s satisfies $\mathbf{A}^{\Gamma}\varphi_{\chi}\mathbf{U}_{\chi'}\varphi'$ under ζ, denoted by $s\vDash_{\zeta}\mathbf{A}^{\Gamma}\varphi_{\chi}\mathbf{U}_{\chi'}\varphi'$, iff for all the ζ-path π such that π_0^s=s, one of the following cases holds:

1. $\pi_0^s\vDash\varphi \wedge \exists i(\forall 1\le j\le i(\pi_j^s\vDash\varphi\wedge\pi_j^{\eta}\vDash\chi) \wedge \forall\gamma\in topology(\Gamma)(constraint(\gamma)\cap\neg\cup_{1\le j\le i}\pi_j^C\neq\varnothing))$.
2. $\pi_0^s\vDash\varphi \wedge \exists i(\forall 1\le j<i(\pi_j^s\vDash\varphi\wedge\pi_j^{\eta}\vDash\chi)\wedge(\pi_i^s\vDash\varphi'\wedge\pi_i^{\eta}\vDash\chi'))$.

We remark that $\eta(a)\vDash\eta(a)$, $\eta(a)\vDash\chi_1\wedge\chi_2$ iff $\eta(a)\vDash\chi_1\wedge\eta(a)\vDash\chi_2$, and $\eta(a)\vDash\neg\chi$ iff $\neg(\eta(a)\vDash\chi)$. We exploit the specification of states violating the formula to intuitively provide the translation. A state immediately violates $\mathbf{A}^{\Gamma}\varphi_{\chi}\mathbf{U}_{\chi'}\varphi'$ under ζ iff it has either (1) an outgoing transition (which conforms to ζ) that is neither a (χ,φ)- nor a (χ′,φ′)-transition, or (2) a (χ,φ)-path that does not violate Γ (and conform to ζ). All preceding states on backward (χ,φ)-paths (with transitions that conform to ζ) that do not violate Γ, also violate the formula. Following the approach of [18], we assume *strong fairness*, meaning that on any infinite path, each infinitely often enabled transition is infinitely often executed. Consequently, in a *terminal* strongly connected components (SCCs) (i.e., SCCs from which no other SCC can be reached) made of only (χ,φ)-transitions that conform to ζ, all the transitions are traversed with this assumption, and all infinite (χ,φ)-paths are guaranteed to end up in such terminal SCCs. If the union of the network constraints of such terminal-SCCs do not violate Γ, then all preceding states over (χ,φ)-transitions as long as they conform to ζ and do not violate Γ, also violate **AU**.

Minimal fixed point operator is used to find the occurrences of none (χ,φ)- and (χ′,φ′)-transitions and terminal-SCCs (made of only (χ,φ)-transitions conforming to ζ and their accumulated network constraints do not violate Γ), as specified by ψ_1. As in terminal SCCs, (χ,φ)-transitions always happen, maximal fixed point is used to find them in formula ψ_2.

$$\begin{aligned}\psi_1\equiv\ & \mu Z(c{:}NC{=}\ empty).\ (\exists c_1{:}NC.\ conform(c_1,\zeta)\wedge\\ & \langle\chi(c_1)\rangle(Z(union(c,c_1))\wedge!violate(c,\Gamma)\wedge\varphi)\ \vee\\ & \langle\chi'(c_1)\wedge\neg\chi(c_1)\rangle\ \neg TR(\varphi'))\ \vee\\ & \langle\chi(c_1)\wedge\neg\chi'(c_1)\rangle\ \neg TR(\varphi))\vee\\ & \langle\neg\chi(c_1)\wedge\neg\chi'(c_1)\rangle true\\ & \langle\chi(c_1)\wedge\chi'(c_1)\rangle\ \neg(\ TR(\varphi'))\wedge TR(\varphi)))\vee\\ & \psi_2(c)\end{aligned}$$

$$\begin{aligned}\psi_2\equiv\ & \nu Y(c{:}NC).\ \forall c_1{:}CN.\ conform(c_1,\zeta)\Rightarrow\\ & [\chi(c_1)]\ Y(union(c,c_1))\wedge\ TR(\varphi)\ \wedge\\ & !violate(c,\Gamma)\wedge[\neg\chi(c_1)]false\end{aligned}$$

The disjunctions of ψ_1 (except $\psi_2(c)$) characterize states with an outgoing transition (which conforms to ζ) that is neither a (χ,φ)- nor a (χ′,φ′)-transition. The first conjunction of this formula finds all possible preceding states on backward (χ,φ)-paths (with transitions that conform to ζ) that do not violate Γ by using the diamond modal operator. The propositional variables are parametrized with a variable of the network constraint datatype to accumulate the network constraints over paths. The conjunctions in formula ψ_2 ensure that only (χ,φ)-transitions exist in a terminal-SCC and their accumulated

network constraints do not violate Γ. The states satisfying **AU** are defined by $\neg(\psi_1) \wedge TR(\varphi)$, simplified in Equation 1:

$$\begin{aligned} TR(\varphi) \wedge \nu Y(c{:}NC).\ \forall c_1{:}NC.\ \mathit{conform}(c_1,\zeta) \Rightarrow \\ [\chi(c_1)]\ Y(\mathrm{union}(c,c_1) \vee \mathrm{violate}(c,\Gamma)) \wedge \\ (([\chi(c_1)]TR(\varphi) \wedge [\chi'(c_1) \wedge \neg\chi(c_1)]\ TR(\varphi')) \vee \\ ([\chi'(c_1)]TR(\varphi') \wedge [\chi(c_1) \wedge \neg\chi'(c_1)]\ TR(\varphi))) \wedge \\ [\neg\chi(c_1) \wedge \neg\chi'(c_1)]\mathrm{false} \wedge \\ \mu Z(c'{:}NC{=}c).\ (\exists c_2{:}NC.\ \mathit{conform}(c_2,\zeta) \wedge \\ (\langle\chi(c_2)\rangle(Z(\mathrm{union}(c',c_2)) \vee \mathrm{violate}(c',\Gamma) \vee \\ \langle\neg\chi(c_2)\rangle \mathrm{true}) \end{aligned} \quad (1)$$

We remark that the parameter of the outer fixed point operator is initially set to "empty" to compute the accumulated network constraints from the initial state. However, the parameter of the inner fixed point operator is initialized to the value of outer one, denoted by $\psi_2(c)$, which indicates the accumulated network constraints before entering into the terminal SCC.

To restrict the formula in Equation 1 to ζ-paths, we use the function "conform" defined over network constraint (NC) datatype as defined in Figure 4. The function *violate*(c,Γ) evaluates to "true" in case the accumulated network constraint c violates Γ. To implement this function, topology formulae are also treated as datatype in mCRL2 specifications as shown in Figure 5. For instance, the topology formula $A \rightarrowtail B \wedge B \rightarrowtail A$ is specified as pconn(A,B,pconn(B,A,none)) by using the constructors "none" and "pconn". The specification of "violate" function has been given in Figure 5. We have used the function "topology(t)" to define "violate" which computes the set of possible topologies for the given topology formula t. This function efficiently computes the minimal connections required that a topology formula holds. For each multi-hop constraint $l \rightarrowtail l'$, it finds one-hop, two-hops, and three-hops, etc (depending on the size of Loc) connections using "conn" constructor of Pair datatype, "twoHop", "threeHop" functions, respectively (in this case |Loc|=4). Each topology is represented as a List of pairs by using the predefined datatype List in mCRL2.

Example. For the set of locations Loc={A,B,C,D}, the translation of formula $\mathbf{A}^{A \rightarrowtail B \wedge B \rightarrowtail A}\ \mathrm{true}_{\{init \wedge \tau\}}\mathbf{U}_{\{succ\}}\mathrm{true}$ under ζ=empty is expressed by the simplified μ-calculus formula in mCRL2:

```
nu Y(c1:NC=empty).(forall c2:NC.
  (conform(c2,empty)) =>
   ([init(c2)+tau(c2)]Y(union(c2,c1))||
(violate(c1,pconn(A,B,pconn(B,A,none))))&&
(mu X(c3:NC=c1).(exists c4:NC.
  (conform(c4,empty))&&
   (<init(c4)+tau(c4)>> X(union(c4,c3)) ||
(violate(c3,pconn(A,B,pconn(B,A,none))))||
    <succ(c4)> true))
```

where "nu", "mu", "exists" and "forall" denote the maximal ν, minimal μ, ∃, and ∀, respectively, and Act={τ,init,succ}. We have implemented this formula, datatypes in Figure 4 and 5 in mCRL2 to validate our implementations [1].

[1] Codes are available at "http:/fghassemi.adhoc.ir/shared/CACTLTrans.zip."

```
sort topologyFormula = struct none |
       pconn(Loc#Loc# topologyFormula);
map
violate:NC# topologyFormula ->Bool;
topology:topologyFormula->List(List(Pair));
conform :NC # List(Pair) -> Bool;
twoHop,threeHop:Loc#Loc#List(Loc) ->
                  List(List(Pair));
add:Pair#List(List(Pair))->
         List(List(Pair));
merge:List(List(Pair))#List(List(Pair)) ->
          List(List(Pair));
mergeh:List(Pair)#List(List(Pair)) ->
        List(List(Pair));
var
   t: topologyFormula;
   g,lp,lp1,lp2:List(Pair);
   llp,llp1,llp2:List(List(Pair));
   l1,l2,l3:Loc; ll:List(Loc);
   nr1:NC;p: Pair;
eqn
violate(nr1,t)= forall g:List(Pair).
     ((g in topology(t))=> !conform(nr1,g));

conform(empty,g)=true;
conform(nx(disconn(l1,l2),nr1),g)=
  (not(conn(l1,l2) in g) && conform(nr1,g));
conform(nx(conn(l1,l2),nr1),g)=
  conform(nr1,g);

topology(none) = [];
topology(pconn(l1,l2,t))=
 merge([[conn(l1,l2)]]++twoHop(l1,l2,loc)++
    threeHop(l1,l2,loc),topology(t));
merge([],llp)=llp;
merge(llp,[])=llp;
llp2!=[] -> merge(lp|>llp1,llp2) =
     mergeh(lp,llp2) ++merge(llp1,llp2);
mergeh(lp,[]) = [];
mergeh(lp1,lp2|>llp)=
     (lp1++lp2)|>mergeh(lp1,llp);

twoHop(l1,l2,[]) =[];
(l1!=l3 && l2!=l3) -> twoHop(l1,l2,l3|>ll)=
[conn(l1,l3),conn(l3,l2)]|>twoHop(l1,l2,ll);
(l1==l3 || l2==l3) -> twoHop(l1,l2,l3|>ll) =
   twoHop(l1,l2,ll);
threeHop(l1,l2,[]) =[];
(l1!=l3 &&l2!=l3) -> threeHop(l1,l2,l3|>ll)=
   add(conn(l1,l3),twoHop(l3,l2,ll))++
   threeHop(l1,l2,ll);
(l1==l3|| l2==l3) -> threeHop(l1,l2,l3|>ll)=
   threeHop(l1,l2,ll);
```

Figure 5. Treating Topology Formula as datatype

3.2. Translation of $E\varphi_\chi U_{\chi'}\varphi'$

A φ-state satisfies $\mathbf{E}\varphi_\chi\mathbf{U}_{\chi'}\varphi'$ under ζ if it has a ζ-path that consists of (χ,φ)-transitions until a (χ′,φ′)-transition is performed. Formally the state s satisfies $\mathbf{E}\varphi_\chi\mathbf{U}_{\chi'}\varphi'$ under ζ, denoted by $s \vDash_\zeta \mathbf{E}\varphi_\chi\mathbf{U}_{\chi'}\varphi'$ iff there exists a ζ-path π such that $\pi_0^s{=}s \wedge \pi_0^s \vDash \varphi \wedge \exists i(\forall 1 \le j < i(\pi_j^s \vDash \varphi \wedge \pi_j^\eta \vDash \chi) \wedge (\pi_i^s \vDash \varphi' \wedge \pi_i^\eta \vDash \chi'))$.

As the (χ′,φ′)-transition must happen over a path, the minimal fixed point operator should be used. To find such a path for a state, the modal diamond operator is used. Equation 2 shows $TR(\mathbf{E}\varphi_\chi\mathbf{U}_{\chi'}\varphi')$:

$$\mu X(c{:}NC{=}\ empty).\ (\forall c_1{:}NC.\ \mathit{conform}(c_1,\zeta)\Rightarrow (\langle\chi(c_1)\rangle X(union(c,c_1))\wedge TR(\varphi))\vee (\exists c_2{:}NC\ \langle\chi'(c_2)\rangle\ TR(\varphi')) \quad (2)$$

The first disjunction looks for a path over (χ,φ)-transitions while the second disjunction handles the occurrence of a (χ',φ')-transition.

3.3. Translation of $A^{\Gamma}\varphi_{\chi}W_{\chi'}\varphi'$

The state formula $\mathbf{A}\varphi_{\chi}\mathbf{W}_{\chi'}\varphi'$ specifies φ-states that all of their ζ-paths consist of (χ,φ)-transitions as long as a (χ',φ')-transition is met or the topology formula Γ is violated. Formally the state s satisfies $\mathbf{A}^{\Gamma}\varphi_{\chi}\mathbf{W}_{\chi'}\varphi'$ under ζ, denoted by $s\models_{\zeta}\mathbf{A}^{\Gamma}\varphi_{\chi}\mathbf{W}_{\chi'}\varphi'$ iff for all ζ-path π such that π_0^s=s, and one of the following cases holds:

1. $\pi_0^s\models\varphi\ \wedge\ \exists i(\forall 1\le j\le i(\pi_j^s\models\varphi\wedge\pi_j^{\eta}\models\chi)\ \wedge\ \forall\gamma\in \mathit{topology}(\Gamma)(\mathit{constraint}(\gamma)\cap\neg\cup_{1\le j\le i}\pi_j^{C}\neq\emptyset))$.
2. $\pi_0^s\models\varphi\wedge\exists i(\forall 1\le j<i(\pi_j^s\models\varphi\wedge\pi_j^{\eta}\models\chi)\wedge(\pi_i^s\models\varphi'\wedge\pi_i^{\eta}\models\chi'))$.
3. $\pi_0^s\models\varphi\wedge\forall 1\le j<len(\pi)\ (\pi_j^s\models\varphi\wedge\pi_j^{\eta}\models\chi)$

where $len(\pi)$ denotes the length of the path. We remark that deadlock states satisfy $\mathbf{A}^{\Gamma}\varphi_{\chi}\mathbf{W}_{\chi'}\varphi'$ as a consequence of the third case. Regarding to the translation of **AU** formula, we again focus on specification of violating states. In contrast to **AU**, a state immediately violates $\mathbf{A}^{\Gamma}\varphi_{\chi}\mathbf{W}_{\chi'}\varphi'$ only iff it has an outgoing transition (which conforms to ζ) that is neither a (χ,φ)- nor a (χ',φ')-transition. So we can consider ψ_1 without the disjunction ψ_2. The translation $TR(\mathbf{A}^{\Gamma}\varphi_{\chi}\mathbf{W}_{\chi'}\varphi')$ is given in Equation 3:

$$TR(\varphi)\wedge\nu Y(c{:}NC).\ \forall c_1{:}NC.\ \mathit{conform}(c_1,\zeta)\Rightarrow [\chi(c_1)]\ Y(union(c,c_1)\vee violate(c,\Gamma))\wedge (([\chi(c_1)]TR(\varphi)\wedge[\chi'(c_1)\wedge\neg\chi(c_1)]\ TR(\varphi'))\vee ([\chi'(c_1)]TR(\varphi')\wedge[\chi(c_1)\wedge\neg\chi'(c_1)]\ TR(\varphi)))\wedge [\neg\chi(c_1)\wedge\neg\chi'(c_1)]false \quad (3)$$

The last three disjunctions prohibit occurrences of none (χ,φ)- and (χ',φ')-transitions, so-called *invalid* transitions. The first conjunction of Equation 3 looks for all (χ,φ)-transitions as long as Γ is not violated and invalid transitions are absent.

3.4. Translation of $E\varphi_{\chi}W_{\chi'}\varphi'$

Formula $\mathbf{E}\varphi_{\chi}\mathbf{W}_{\chi'}\varphi'$ is satisfied by φ-states that either have a ζ-path of (χ,φ)-transitions that end with a (χ',φ')-transition, or have a (χ,φ) ζ-path. Formally the state s satisfies $\mathbf{E}\varphi_{\chi}\mathbf{W}_{\chi'}\varphi'$ under ζ, denoted by $s\models_{\zeta}\mathbf{E}\varphi_{\chi}\mathbf{W}_{\chi'}\varphi'$ iff there exists a ζ-path π such that π_0^s=s, and one of the following cases holds:

1. π_0^s=s $\wedge$ $\pi_0^s\models\varphi$ $\wedge$ $\exists i(\forall 1\le j<i(\pi_j^s\models\varphi\wedge\pi_j^{\eta}\models\chi)\wedge(\pi_i^s\models\varphi'\wedge\pi_i^{\eta}\models\chi'))$.
2. $\pi_0^s\models\varphi\wedge\forall 1\le j<len(\pi)\ (\pi_j^s\models\varphi\wedge\pi_j^{\eta}\models\chi)$.

We remark that deadlock states satisfy $\mathbf{E}\varphi_{\chi}\mathbf{W}_{\chi'}\varphi'$ as a consequence of the second case. As (χ',φ')-transition may not occur over a path and always (χ,φ)-transition can occur (second case), the maximal fixed point operator is used. Equation 4 provides $TR(\mathbf{E}\varphi_{\chi}\mathbf{W}_{\chi'}\varphi')$:

$$\nu X(c{:}NC{=}empty).(\forall c_1{:}NC.\ \mathit{conform}(c_1,\zeta)\Rightarrow \langle\chi(c_1)\rangle\ X(union(c,c_1))\wedge\ TR(\varphi))\vee[true]false\vee (\exists c_2{:}NC\ \langle\chi'(c_2)\rangle\ TR(\varphi')) \quad (4)$$

The first disjunction of Equation 4 looks for a sequence of (χ,φ)-transitions until either the second disjunction, i.e., deadlock, or the third disjunction, i.e., occurrence of a (χ',φ')-transition, holds.

4. Soundness of the Translation

To prove the soundness of our translation, we show that a state of CLTS satisfies the given property φ iff this state in its corresponding strongly bisimilar modulo renaming LTS satisfies the corresponding μ-calculus expression, i.e., $TR(\varphi)$.

Theorem 1. Assume $CLTS\equiv\langle S,\ \Lambda,\ \rightarrow,s_0\rangle$ and $TR(CLTS)\equiv\langle S,\ \Lambda',\ \rightarrow',s_0\rangle$, where $\Lambda'=\{\eta(c)|(c,\eta)\in\Lambda\}$ and $\rightarrow'=\{(s,\eta(c),t)|(s,(c,\eta),t)\in\rightarrow\}$. Then $s,CLTS\models\psi$ iff $s,TR(CLTS)\models TR(\psi)$.

Proof. By induction on the structure of φ.

$\psi\equiv$**true**: any state of CLTS satisfies true, and trivially any state of TR(CLTS) satisfies true.

$\psi\equiv\neg\varphi$: $s,CLTS\models\neg\varphi$ iff $s,CLTS\not\models\varphi$. By induction, $TR(CLTS)\not\models TR(\varphi)$, and thus, $TR(CLTS)\models\neg TR(\varphi)=TR(\neg\varphi)$.

$\psi\equiv\varphi_1\wedge\varphi_2$: $s,CLTS\models\varphi_1\wedge\varphi_2$ iff $s,CLTS\models\varphi_1\wedge s,CLTS\models\varphi_2$. By induction, $s,TR(CLTS)\models TR(\varphi_1)\wedge s,TR(CLTS)\models TR(\varphi_2)$. Therefore, $s,TR(CLTS)\models TR(\varphi_1)\wedge TR(\varphi_2)$ by which we conclude that $s,TR(CLTS)\models TR(\varphi_1\wedge\varphi_2)$. The reverse is proved with a same discussion.

$\psi\equiv\mathbf{A}^{\Gamma}\varphi_{\chi}\mathbf{U}_{\chi'}\varphi'$: We first show that if $s,CLTS\models\psi$, then $s,TR(CLTS)\models TR(\psi)$.We manage the proof by induction on the length of the (χ,φ)-transitions over the paths of the states that satisfy the property.

For the base, assume that state s of CLTS satisfies φ for zero (χ,φ)-transition. Thus, according to the semantics of **AU**, one case can only be considered: states with only (χ',φ')-transitions which conform to ζ and no (χ,φ)-transitions. By application of Tarski Theorem [25], the equation of maximal operator is solved by substituting $\{empty\mapsto S\}$ for Y in $TR(\psi)$, and hence such states are included by the conjunction $([\chi'(c_1)]TR(\varphi')\wedge[\chi(c_1)\wedge\neg\chi'(c_1)]\ TR(\varphi)))$.We assume the assumption holds for the states with paths of (χ,φ)-transitions with a length of less than n. These paths are either followed by (χ',φ')-transitions after at most n-1 (χ,φ)-transitions, or violate Γ. Now consider the state s of CLTS that satisfies φ while all of its paths of (χ,φ)-transitions making the property to be hold, is equal or less than n. By the disjunction $(\forall c_1{:}NC.\mathit{conform}(c_1,\zeta)\Rightarrow\ [\chi(c_1)]\ Y(union(c,c_1))\ \vee\ violate(c_1,\Gamma))$, such states will be added after solving the equation after at most n iterations, because at the last iteration $Y(union(c,c_1))$ is replaced by the set of states whose paths of

(χ,φ)-transitions making the property to be hold, is less than n. Thus by assumption, s will be added to Y.

We now prove that if TR(CLTS) $\models$TR(ψ), then s,CLTS$\models\psi$. We manage the proof by contradiction. Assume that s,CLTS$\not\models\psi$, so according to semantics of **AU**, three cases can be hold for s: 1)it has a transition which is neither a (χ,φ)- nor a (χ',φ')-transition, 2)it has a (χ,φ)-path that its accumulated network constraint does not violate Γ ending in a deadlock or a terminal SCC. The three disjunctions $(([\chi(c_1)]TR(\varphi)\wedge[\chi'(c_1)\wedge\neg\chi(c_1)]\ TR(\varphi')) \vee ([\chi'(c_1)]TR(\varphi')\wedge[\chi\ (c_1)\wedge\neg\chi'(c_1)]\ TR(\varphi)))\wedge[\neg\chi(c_1)\wedge\neg\chi'(c_1)]$false excludes the state s if the first case holds. The minimal fixed point operator excludes the occurrence of the second case by its subformula $))\vee$violate$(c',\Gamma)\vee\ \langle\neg\chi(c_2)\rangle$true); the use of modal diamond excludes deadlocks and states of terminal SCCs that their accumulated network constraint do not violate Γ, and thus states with such (χ,φ)-path are not included, and consequently TR(CLTS) $\not\models$TR(ψ).

The cases $\psi\equiv\mathbf{A}^{\Gamma}\varphi_{\chi}\mathbf{W}_{\chi'}\varphi'$, $\psi\equiv\mathbf{E}\varphi_{\chi}\mathbf{U}_{\chi'}\varphi'$, and $\psi\equiv\mathbf{E}\varphi_{\chi}\mathbf{W}_{\chi'}\varphi'$ can be proved with the same discussion for $\mathbf{A}^{\Gamma}\varphi_{\chi}\mathbf{U}_{\chi'}\varphi'$.

5. Conclusion and Future Work

To verify the topology-dependent properties of MANET processes with data, we provided a sound translation from the temporal CACTL formulae to first-order modal μ-calculus expressions extended with data. The resulting μ-calculus formulae can be verified on the algebraic specification of MANETs by exploiting mCRL2 toolset. This allows to formulate and prove a substantially wider range of properties on much larger and even infinite state MANETs. Our translation takes advantage of parametrized propositional variables and formal treatment of data to specify the topology-dependent behavior of CACTL operators. By treating network constraint as a datatype, the parameter of network constraint in propositional variables accumulates the network constraints of actions over the paths. Furthermore, by specifying topology formulae as datatype, violation of a topology formula by accumulated network constraints can be examined by appropriate operations. We proved that our translation is sound and thus is applicable for larger MANETs.

We aim to provide a compiler which translates automatically MANET processes, specified in RRBPT, and CACTL formulae to mCRL2 and μ-calculus expressions, respectively. Then we can elaborate on more complex MANET.

References

[1] Construction and Analysis of Distributed Processes (CADP), cadp.inria.fr, last visited June 2018.

[2] Verifying Multi-Threaded Software with Spin, spinroot.com/spin/whatispin.html, last visited June 2018.

[3] UPAAL, http://www.uppaal.org/, last visited June 2018.

[4] D. Peled, "All from one, one for all: on model checking using representatives," In Proc. 5th Workshop on Comput.-Aided Verification, pp. 409–423, 1993.

[5] E.M Clarke, E.A Emerson, S. Jha, A.P Sistla, "Symmetry reductions in model checking," In Proc. Computer Aided Verification, pp. 147–158. Springer, 1998.

[6] P. Abdulla, A. Bouajjani, and B. Jonsson, "On-the-fly analysis of systems with unbounded, lossy FIFO channels," In Proc. Computer Aided Verification, Vol. 1427 of LNCS, pp. 305–318. Springer, 1998.

[7] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper, "The power of QDDs. In P. van Hentenryck, editor, Static Analysis," In Proc. 4th International Symposium, Vol. 1302 of LNCS, pp. 172–186. Springer, 1997.

[8] T. Bultan, R. Gerber, and W. Pugh, "Symbolic model checking of infinite state systems using Presburger arithmetic," In Proc. Computer Aided Verification, Vol. 1254 of LNCS, pp. 400–411, Springer, 1997

[9] A. Pnueli, J. Xu, and L. Zuck, "Liveness with (0,1,infinity)-counter abstraction," In Proc. Computer Aided Verification, Vol. 2404 of LNCS, pp. 107– 122. Springer, 2002.

[10] mCRL2: Analyzing System Behaviors, mcrl2.org, last visited at June 2018.

[11] J.C.M. Baeten and W.P. Weijland, Process Algebra, Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[12] D. Kozen, "Results on the propositional mu-calculus," *Theoretical Computer Science*, Vol. 27, pp. 333–354, 1983.

[13] A. Mader, "Modal mu-calculus, model checking and GauB elimination," pp. 72-88, Springer, 1995.

[14] J.F. Groote and R. Mateescu, "Verification of temporal properties of processes in a setting with data," In Proc. 7th International Conference on Algebraic Methodology and Software Technology, Vol. 1548 of LNCS, pp. 74–90. Springer, 1999.

[15] J.F. Groote, Willemse and T. A. C., "Model-checking processes with data," *Science of Computer Programming*, Vol. 56, pp. 251-273, 2005.

[16] A. Fehnker, R. van Glabbeek, P. Höfner, A. McIver, M. Portmann M, and W. Tan W, "A process algebra for wireless mesh networks," In Proc. of the 21st European symposium on programming, Vol. 7211 of LNCS, pp. 295–315. Springer, 2012.

[17] F. Ghassemi, S. Ahmadi, W. Fokkink, and A. Movaghar, "Model checking MANETs with arbitrary mobility," In Proc. of the 5th conference on fundamentals of software engineering, Vol 8161 of LNCS, pp 217–232. Springer, 2013.

[18] F. Ghassemi, W. Fokkink, "Model checking mobile ad hoc networks," *Formal Methods in System Design*, Vol. 49, No. 3, pp.159-189, 2016.

[19] F. Ghassemi, W. Fokkink, and A. Movaghar, "Verification of mobile ad hoc networks: an algebraic approach," *Theoretical Computer Science*, Vol. 412, No. 28, pp. 3262–3282, 2011.

[20] R. Glabbeek, P. Höfner, and D. van der Wal, "Analysing AWN-specifications using mCRL2," In Proc. of Integrated Formal Methods, 2018.

[21] R. De Nicola R, and F. Vaandrager, "Action versus state based logics for transition systems," In Proc. Semantics of

systems of concurrent processes, Vol. 469 of LNCS, pp. 407–419. Springer, 1990.
[22] R. Meolic R, T. Kapus T, and Z. Brezocnik, "ACTLW-an action-based computation tree logic with unless operator," *Information Science*, Vol. 178, No. 6, pp. 1542–1557, 2008.
[23] J.F. Groote and M.R. Mousavi, Modeling and Analysis of Communicating Systems, MIT Press, 2014.
[24] J.F. Groote, and T. Willemse, "Parameterised boolean equation systems," *Theoretical Computer Science*, Vol. 343, No. 3, pp. 332-369, 2005.
[25] A. Tarski et al., "A lattice-theoretical fixpoint theorem and its applications," *Pacific journal of Mathematics*, Vol. 5, pp. 285-309, 1955.
[26] H. Ehrich, J. Loeckx, and M Wolf, Specification of Abstract Data Types, John Wiley, 1996.

Fatemeh Ghassemi has received her Ph.D. in Software Engineering from Sharif university of technology in 2011, and in Computer Science from Vrije Universiteit of Amsterdam in 2018. She is an assistant professor at University of Tehran since 2012, supervising the Formal Methods laboratory. Her research interests includes formal methods in software engineering, protocol verification, model checking, process algebra, software testing, and wireless systems.

Email: fghassemi@ut.ac.ir

Paper Handling Data:

Submitted: 06.24.2018
Received in revised form: 08.02.2018
Accepted: 08.04.2018
Corresponding author: Fatemeh Ghassemi
Affiliation of the corresponding author: University of Tehran