Analysing AWN-specifications using mCRL2 (extended abstract)

Rob van Glabbeek[1,2], Peter Höfner[1,2] & Djurre van der Wal[1,3]

[1] Data61, CSIRO, Sydney, Australia
[2] Comput. Sci. and Engineering, University of New South Wales, Sydney, Australia
[3] Formal Methods and Tools, University of Twente, The Netherlands

Abstract. We develop and implement a translation from the process Algebra for Wireless Networks (AWN) into the milli Common Representation Language (mCRL2). As a consequence of the translation, the sophisticated toolset of mCRL2 is now available for AWN-specifications. We show that the translation respects strong bisimilarity; hence all safety properties can be automatically checked using the toolset. To show usability of our translation we report on a case study.

1 Introduction

The Algebra for Wireless Networks (AWN) [11] is a variant of classical process algebras that has been particularly tailored to model and analyse protocols for Mobile Ad hoc Networks (MANETs) and Wireless Mesh Networks (WMNs). Among others it has been successfully used to model and analyse the Ad hoc On-Demand Distance Vector (AODV) routing protocol [30], one of the most popular protocols widely used in WMNs. [12,16]

AWN provides the right level of abstraction to model key features of protocols for (wireless) networks such as unicast and broadcast for message sending, while abstracting from implementation-related details. It is equipped with a (completely unambiguous) formal semantics, which is given in form of structural operational semantics rules. These rules generate a transition system that can be used to describe the behaviour of a protocol.

The algebra has been integrated in the interactive proof assistant Isabelle/HOL [6]. This enabled, amongst others, the machine-checked verification of key correctness properties of AODV. However, apart from that there is only little tool-support for AWN. To provide automatic analysis for protocols written in AWN, the algebra has been used in combination with the model checker Uppaal. [10] The input model for Uppaal [2,1]—a network of timed automata—was created manually and the correctness of this model needed to be established manually as well.

In sum, AWN falls short when it comes to automated analysis of specifications. The development of special-purpose tools for AWN is cumbersome, error-prone and time consuming. Hence we follow the general approach to make

use of highly sophisticated off-the-shelf tools that offer high-performance analysis. In this paper we present and implement an automatic translation from AWN into the milli Common Representation Language (mCRL2) [18].

mCRL2 is a formal specification language with an associated collection of tools offering support for model checking, simulation, state-space generation, as well as for the optimisation and analysis of specifications. [8] The toolset has been used in countless case studies, including the analysis software for the CERN's Large Hadron Collider [21], and the IEEE 1394 link layer [25].

We do not only develop an automatic translation from AWN to mCRL2, which allows us to use the mCRL2 toolset for any protocol specification written in AWN, we also show that the transition system induced by an AWN-specification and the transition system in mCRL2 that stems from our translation are strongly bisimilar. As a consequence, any safety property that has been (dis)proven in the mCRL2 setting also holds/does not hold for the original specification, written in AWN. To illustrate the usefulness of our translation, we report on a case study that analyses the Ad hoc On-Demand Distance Vector (AODV) routing protocol [30], which we formalised in AWN before [12,16].

2 The Algebra for Wireless Networks

The Algebra for Wireless Networks (AWN) [10,12] is a variant of standard process algebras (e.g. [26,20,3,5]) particularly tailored for (wireless) protocols: it defines the protocol in a pseudo-code that is easily readable, and provides the right level of abstraction to model key protocol features.

The algebra offers a local broadcast mechanism and a conditional unicast operator—allowing error handling in response to failed communications while abstracting from link layer implementations of the communication handling—and incorporates data structures with assignments. As a consequence it allows to describe the interaction between nodes in a network with a dynamic or static network topology, and hence is ideal to describe all kinds of protocols.

AWN comprises five layers: (a) *sequential processes* for encoding the protocol as a recursive specification; (b) *parallel composition* of sequential processes for running multiple processes simultaneously on a single (network) node; (c) *node expressions* for running (parallel) processes on a node while tracking the node's address and all nodes within transmission range; (d) *partial network expressions* for describing networks as parallel compositions of nodes and allowing changes in the network topology, and (e) *complete network expressions* for closing partial networks to further interactions with the environment.

Due to lack of space we cannot present the full syntax and semantics of AWN, which can be found in [11]. Table 1 summarises the syntax. In AWN a network is modelled as an encapsulated parallel composition of network nodes (Lines 14 and 15 in Table 1). An individual node has the form $i : P : R$, where i is the unique identifier of the node, P characterises the process running on the node, and the set R contains all identifiers of nodes currently in transmission range of i—the nodes that can receive messages sent by i. On each node several processes

Table 1. process expressions

$X(exp_1, \ldots, exp_n)$	process name with arguments
$P + Q$	choice between processes P and Q
$[\varphi]P$	conditional process
$[\![\mathtt{v} := exp]\!]P$	assignment followed by process P
broadcast$(ms).P$	broadcast *ms* followed by P
groupcast$(dests, ms).P$	iterative unicast or multicast to all destinations *dests*
unicast$(dest, ms).P \blacktriangleright Q$	unicast *ms* to *dest*; if successful proceed with P; otherwise Q
send$(ms).P$	synchronously transmit *ms* to parallel process on same node
deliver$(data).P$	deliver data to client (application layer)
receive$(\mathtt{m}).P$	receive a message
ξ, P	process with valuation
$P \lll Q$	parallel processes on the same node
$i : P : R$	node i running P with range R
$N \| M$	parallel composition of nodes
$[N]$	encapsulation

may be running in parallel (Line 12 in Table 1). A sequential process is given by a sequential process expression P, together with a *valuation* ξ associating values $\xi(\mathtt{v})$ to variables $\mathtt{v}$ maintained by this process (Line 11).

AWN uses an underlying data structure with several types, variables ranging over these types, operators and predicates. Predicate logic yields terms (or *data expressions*) and formulas to denote data values and statements about them. The choice of this data structure is tailored to any particular application of AWN. It must contain the types `DATA`, `MSG`, `IP` and $\mathcal{P}(\mathtt{IP})$ of application layer data, messages, IP addresses—or other node identifiers—and sets of IP addresses.

In addition, AWN employs a collection of *process names*, each carrying parameters of various types. Every process name X comes with a *defining equation* $X(\mathtt{v}_1, \ldots, \mathtt{v}_n) \stackrel{def}{=} P$, in which each $\mathtt{v}_i$ is a variable of the appropriate type and P a sequential process expression.

Lines 1 to 10 describe sequential process expressions. $X(exp_1, \ldots, exp_n)$ is a call to the process defined by the process name X, with expressions of the appropriate types substituted for the parameters. $P + Q$ may act either as P or as Q, depending on which of the two processes is able to act. If both are able to act, a non-deterministic choice is made. Given a valuation of the data variables by concrete data values, the sequential process $[\varphi]P$ acts as P if φ evaluates to true, and deadlocks otherwise. In case φ contains free variables, values are assigned to these variables in any way that satisfies φ, if possible. The process $[\![\mathtt{v} := exp]\!]P$ acts as P, but under an updated valuation of the data variable $\mathtt{v}$. The process **broadcast**(ms) broadcasts *ms* to the other network nodes within transmission range, and subsequently acts as P; **unicast**$(dest, ms).P \blacktriangleright Q$ is a process that tries to unicast the message *ms* to the destination *dest*; if successful it continues to act as P and otherwise as Q. It models an abstraction of an acknowledgment-of-receipt mechanism. The process **groupcast**$(dests, ms).P$ tries to transmit *ms* to all destinations *dests*, and proceeds as P regardless of whether any of the transmissions is successful. The action **send**(ms) (synchronously) transmits a message to another process running on the same node. The sequential process

Process 1 Voting

```
Voting(lip,lno,voted,ip,no) ≝
1. (receive(m) . [m = B(sip, sn)]    /* receive ballot */
2.    Eval(sip,sn,lip,lno,voted,ip,no))
3. + [ !voted ]       /* cast a ballot */
4.    ((broadcast(B(ip, no)) .
5.        Eval(ip,no,lip,lno,true,ip,no))
6.     + (receive(m) . [ m = B(sip, sn) ]
7.        Eval(sip,sn,lip,lno,voted,ip,no)) )
```

Process 2 Vote Evaluation

```
Eval(sip,sn , lip,lno,voted,ip,no) ≝
1. [ sn ≥ lno ]       /* vote better */
2.    Voting(sip,sn,voted,ip,no)
3. + [ sn < lno ]       /* vote worse */
4.    Voting(lip,lno,voted,ip,no)
```

receive(m).P receives any message m (a data value of type MSG) either from another node, from another sequential process running on the same node or from the client hooked up to the local node. It then proceeds as P, but with the data variable m bound to the value m. The submission of data from a client is modelled by the receipt of a message **newpkt**(d, dip), where the function **newpkt** generates a message containing the data d and the intended destination dip. Data is delivered to the client by **deliver**$(data)$.

The layers of sequential and parallel processes usually define the behaviour of a protocol up to a point where it can be implemented; the other layers are used for reasoning, and include primitives for modelling dynamic topologies.

Processes 1 and 2, for example, describe a simple leader election protocol: each node in the network, which is assumed to be fully connected, holds a unique node identifier ip and a natural number n. Each node is initialised by $(\xi,$Voting(lip,lno,voted,ip,no)$)$, with $\xi($lip$) = \xi($ip$) = ip$, $\xi($lno$) = \xi($no$) = n$, and $\xi($voted$) =$ false. The local variables lip and lno hold the identifier and the number of the current leader; the Boolean flag voted indicates whether the process partook in the election.

Process 1 allows the node to receive a ballot (message) B from another node (Lines 1 and 6, resp.). The message contains the sender's address ip, as well as its number no; these are stored in the local variables sip and sn. In case a message is received, the evaluation process Eval is called (Line 2). Once during the protocol (Line 3) each node can partake in the election and send its ballot, containing the node's own information (Line 4). After the message is sent, the flag voted is set to true (Line 5), and the node acts as if it had received this message.

Process 2 evaluates the information received. If the received number sn is greater than or equal to the number of the current leader lno, the current leader is set to sip and the current leader's number to sn, and the process returns to the main process; otherwise the information of the received message is disregarded.

When the protocol terminates—all nodes voted and all messages have been handled—all nodes have agreed on a leader, one holding the highest number no.

Once a model has been described in AWN, its behaviour is governed by the transitions allowed by the algebra's semantics. The formal semantics of AWN is given as structural operational semantics (sos) in the style of Plotkin [31] and describes how states evolve into another by performing *actions*. [12,11]

Table 2 presents four sos-rules of AWN (out of 44), all describing behaviour w.r.t. broadcast. The first rule describes the behaviour of the sequential process **broadcast**$(ms).P$, which performs the action **broadcast**$(\xi(ms))$ without synchronisation. Here $\xi(ms)$ is the data value denoted by the expression ms when

Table 2. Structural operational semantics (AWN) for **broadcast**

$$\xi, \mathbf{broadcast}(ms).P \xrightarrow{\mathbf{broadcast}(\xi(ms))} \xi, P \qquad \frac{P \xrightarrow{\mathbf{broadcast}(m)} P'}{ip : P : R \xrightarrow{R:\mathbf{*cast}(m)} ip : P' : R}$$

$$\frac{M \xrightarrow{R:\mathbf{*cast}(m)} M' \quad N \xrightarrow{H\neg K:\mathbf{arrive}(m)} N'}{M \| N \xrightarrow{R:\mathbf{*cast}(m)} M' \| N'} \begin{pmatrix} H \subseteq R \\ K \cap R = \emptyset \end{pmatrix} \qquad \frac{M \xrightarrow{R:\mathbf{*cast}(m)} M'}{[M] \xrightarrow{\tau} [M']}$$

the variables occurring in ms are evaluated according to ξ. The second rule describes the **broadcast**-action on the node level: as the nodes in transmission range of node ip are known (stored in set R), this set is part of the new label and is used for synchronisation on the network layer. The third rule illustrates this partly. The action $R:\mathbf{*cast}(m)$ casts a message m that can be received by the set R of network nodes. AWN does not distinguish whether this message stems from a **broadcast**-, a **groupcast**- or a **unicast** action—the differences show up merely in the value of R. The action $H\neg K:\mathbf{arrive}(m)$ models that m simultaneously arrives at all addresses $ip \in H$, and fails to arrive at all addresses $ip \in K$. The third rule of Table 2 synchronises a $R:\mathbf{*cast}(m)$-action of one node with an $\mathbf{arrive}(m)$ of all other nodes. To finalise this synchronisation AWN features another two sos-rules: a symmetric form of the third rule, and a rule synchronising two $H\neg K:\mathbf{arrive}(m)$-actions. The side conditions ensure arrival of message m at all the nodes in the transmission range R of the $\mathbf{*cast}(m)$, and non-arrival at the other nodes. The fourth rule of Table 2 closes the network by the encapsulation operator $[\_]$, and transforms the $R:\mathbf{*cast}(m)$-action into an internal action τ. The encapsulation guarantees that no messages will be received that have never been sent.

3 The Algebra mCRL2 and its Associated Toolset

The milli Common Representation Language (mCRL2) [18] is a formal specification language with an associated toolset [8]. Similar to AWN, mCRL2 is a variant of standard process algebras with a formal semantics given as structural operational semantics in the style of Plotkin.

For our translation from AWN to mCRL2 we use only a fragment of mCRL2. In this section we briefly explain the syntax and semantics of those constructs of mCRL2 needed for our translation. As before, we can only show parts of the semantics, and refer to [18] for details.

Similar to AWN, mCRL2 comes with *defining equations*, called process equations in [18], having the form $X(\mathtt{d}_1:\mathtt{D}_1,\ldots,\mathtt{d}_n:\mathtt{D}_n) \stackrel{\text{def}}{=} p$, where $\mathtt{d}_i$ are variables of sorts $\mathtt{D}_i$ and p a process expression defined by the following grammar.

$$p ::= \alpha \mid p{\cdot}p \mid p+p \mid c \to p \mid \sum_{\mathtt{d}:\mathtt{D}} p \mid p \| p \mid X(\mathtt{u}_1,\ldots,\mathtt{u}_n) \mid \Gamma_C(p) \mid \nabla_V(p) \mid \rho_R(p) \mid \tau_I(p)$$
$$\alpha ::= \tau \mid a(\mathtt{u}_1,\ldots,\mathtt{u}_n) \mid \alpha|\alpha$$

Here α denotes a multi-action, c a Boolean, and the $\mathtt{u}_i$ are data expressions.

Actions form the basic building blocks of mCRL2. They consists of a name (taken from a given set) and some parameters, which are expressions denoting

Process 3 Voting in mCRL2

$\mathbf{proc}\ \mathit{Voting}(\mathtt{lip}{:}\mathrm{IP}, \mathtt{lno}{:}\mathbb{N}, \mathtt{voted}{:}\mathbb{B}, \mathtt{ip}{:}\mathrm{IP}, \mathtt{no}{:}\mathbb{N}) \stackrel{\text{def}}{=}$

1. $\sum_{\mathtt{D},\mathtt{D}':\mathrm{Set(IP)},\mathtt{m}:\mathrm{MSG}} \mathbf{receive}(\mathtt{D},\mathtt{D}',\mathtt{m})\cdot$
2. $\quad \sum_{\mathtt{sip}:\mathrm{IP},\mathtt{sn}:\mathbb{N}}((\mathtt{m} \approx \mathtt{MSG}(\mathtt{B},\mathtt{sip},\mathtt{sn})) \rightarrow (\mathbf{t}\cdot \mathit{Eval}(\mathtt{sip},\mathtt{sn},\mathtt{lip},\mathtt{lno},\mathtt{voted},\mathtt{ip},\mathtt{no})))$
3. $+\ (!\mathtt{voted}) \rightarrow \mathbf{t}\cdot($
4. $\quad \sum_{\mathtt{D}:\mathrm{Set(IP)}} \mathbf{cast}(\mathrm{IP},\mathtt{D},\mathtt{MSG}(\mathtt{B},\mathtt{ip},\mathtt{no}))\cdot \mathit{Eval}(\mathtt{ip},\mathtt{sn},\mathtt{lip},\mathtt{lno},\mathtt{voted},\mathtt{ip},\mathtt{no})$
5. $\quad +\ \sum_{\mathtt{D},\mathtt{D}':\mathrm{Set(IP)},\mathtt{m}:\mathrm{MSG}} \mathbf{receive}(\mathtt{D},\mathtt{D}',\mathtt{m})\cdot$
6. $\quad\quad \sum_{\mathtt{sip}:\mathrm{IP},\mathtt{sn}:\mathbb{N}}((\mathtt{m} \approx \mathtt{MSG}(\mathtt{B},\mathtt{sip},\mathtt{sn})) \rightarrow (\mathbf{t}\cdot \mathit{Eval}(\mathtt{sip},\mathtt{sn},\mathtt{lip},\mathtt{lno},\mathtt{voted},\mathtt{ip},\mathtt{no}))))$

data values. Multi-actions are collections of actions that occur at the same time. A multi-action can be empty, denoted by τ; it is used as internal, non-observable action. $a(\vec{\mathtt{u}})$ denotes an action with name a and data parameters $\mathtt{u}_i$. Last, the multi-action $\alpha|\beta$ consists of the actions from both multi-actions α and β.

The process $p{\cdot}q$ behaves like p until p terminates, and then continues to behave as q. The process $p+q$ may act either as p or as q, depending on which of the processes can perform an action. If both are able to act, a non-deterministic choice is made. For a Boolean expression c, the process $c \rightarrow p$ acts like p if c evaluates to true, and deadlocks otherwise. The process $\sum_{\mathtt{d}:\mathtt{D}} p$ allows for a choice of p for any value d from $\mathtt{D}$ substituted for the variable $\mathtt{d}$—of course $\mathtt{d}$ can occur in p. The process $p\|q$ is a parallel composition of p and q. X is a process name, and $\mathtt{u}_i$ are data expressions of type $\mathtt{D}_i$, as declared in the defining equation; $X(\mathtt{u}_1,\ldots,\mathtt{u}_n)$ denotes a process call.

The communication operator $\Gamma_C(p)$ takes some actions out of a multi-action and replaces them with a single action, provided their data parts are equal. The set C describes the replacement by rules of the form $a_1|\cdots|a_1 \rightarrow c$. To enforce communication the allow operator $\nabla_V(p)$ only allows multi-actions listed in the set V to occur. The renaming operator $\rho_R(p)$ renames action names within p, where the set R lists rename rules of the form $a \rightarrow b$. Finally, the hiding operator $\tau_I(p)$ conceals all action names listed in I from the process p, replacing them by the internal action τ.

Process 3 models the same behaviour as Process 1, but written in mCRL2. In fact the presented specification has been translated by our tool (see Section 7); we have only changed minor issues such as variable names and line breaks to ease readability. An interesting issue when looking at the translation is that Process 3 features multiple sum-operators, where the AWN-specification shows none. While it may be understandable why the **receive**-actions (Lines 1 and 5) need to sum over all possible messages that could be received, the argument for sending messages (Line 4) is not straightforward. The reason is that we have to encode all possible transmission ranges $\mathtt{D}$; we elaborate on this in more detail in Section 5. Moreover, the AWN guard $[\mathtt{m} = \mathtt{B}(\mathtt{sip},\mathtt{sn})]$ assigns values to $\mathtt{sip}$ and $\mathtt{sn}$ such that $\mathtt{m} = \mathtt{B}(\mathtt{sip},\mathtt{sn})$; in mCRL2 this involves summing over all values $\mathtt{sip}$ and $\mathtt{sn}$ can take, in combination with the equality check $\mathtt{m} \approx \mathtt{MSG}(\mathtt{B},\mathtt{sip},\mathtt{sn})$.

Table 3 shows some rules of the structural operational semantics of mCRL2. Here ✓ indicates successful termination, and $[\![\_]\!]$ is an interpretation function, sending syntactic expressions to semantic values. We have $[\![\tau]\!]=\tau$, $[\![a(\mathtt{u}_1,\ldots,\mathtt{u}_n)]\!] = a([\![\mathtt{u}_1]\!]\ldots,[\![\mathtt{u}_n]\!])$, and $[\![\alpha|\beta]\!] = [\![\alpha]\!]|[\![\beta]\!]$, where at the right-hand (semantic) side τ denotes the empty multiset, $[\![\alpha]\!]|[\![\beta]\!]$ the union of multisets $[\![\alpha]\!]$ and $[\![\beta]\!]$, and

Table 3. Structural operational semantics (mCRL2)

$$\alpha \xrightarrow{[\![\alpha]\!]} \checkmark \qquad \frac{p \xrightarrow{\omega} \checkmark}{p \cdot q \xrightarrow{\omega} q} \qquad \frac{p \xrightarrow{\omega} p'}{p + q \xrightarrow{\omega} p'} \qquad \frac{p \xrightarrow{\omega} p' \quad q \xrightarrow{\varpi} q'}{p \| q \xrightarrow{\omega|\varpi} p' \| q'}$$

$$\frac{p[\mathtt{d}{:=}t_e] \xrightarrow{\omega} p'}{\sum_{\mathtt{d}:\mathtt{D}} p \xrightarrow{\omega} p'}\ e \in M_{\mathtt{D}} \qquad \frac{p \xrightarrow{\omega} p'}{c \to p \xrightarrow{\omega} p'}\ [\![c]\!] = \text{true} \qquad \frac{q[\mathtt{d}_1{:=}\mathtt{u}_1, \ldots, \mathtt{d}_n{:=}\mathtt{u}_n] \xrightarrow{\omega} q'}{X(\mathtt{u}_1, \ldots, \mathtt{u}_n) \xrightarrow{\omega} q'}$$

$a(e_1, \ldots, e_n)$ (the singleton multiset containing) the action a, whose parameters are now data values rather than data expressions. The first four rules are standard process algebra and (partly) characterise execution of an action, sequential composition (under successful termination), left choice and synchronisation, respectively. The first rule in the second line describes the sum operator. Here $M_{\mathtt{D}}$ is the set of data values of type `D` and t is a function—assumed to exist in mCRL2—that for each data value e returns a closed term t_e denoting e, i.e., $[\![t_e]\!] = e$. The second rule models a guard c; only if it evaluates to true, the process can proceed. The last rule of Table 3 defines recursion, where we assume a process $X(\mathtt{d}_1{:}\mathtt{D}_1 \ldots, \mathtt{d}_n{:}\mathtt{D}_n) \stackrel{\text{def}}{=} q$. mCRL2 also provides rules for the communication, the allow, and the restriction operator:

$$\frac{p \xrightarrow{\omega} p'}{\Gamma_C(p) \xrightarrow{\gamma_C(\omega)} \Gamma_C(p')} \qquad \frac{p \xrightarrow{\omega} p'}{\rho_R(p) \xrightarrow{R\bullet\omega} \rho_R(p')} \qquad \frac{p \xrightarrow{\omega} p'}{\nabla_V(p) \xrightarrow{\omega} \nabla_V(p')}\ \underline{\omega} \in V \cup \{\tau\}$$

Here the functions γ_C, and $R\bullet$ are the counterparts of Γ_C and ρ_R, resp., working on actions rather than processes. For example, $\gamma_{\{a|b\to c\}}(a|a|b|c) = a|c|c$. The stripped multi-action $\underline{\omega}$ is the result of removing all data from the multi-action ω.

Although mCRL2 works on top of an underlying data structure, it does not provide any syntactic construct for assignment.

mCRL2 comes with an associated toolset, consisting of about 50 different tools (see `www.mcrl2.org`). The toolset includes a user interface, which provides an easy way to read and analyse any mCRL2-process. Other tools help in manipulating and visualising state spaces, or provide support for automatic analysis. This includes classical model checking as well as checking properties by parameterised Boolean equation systems. The toolset also includes an interface allowing system analysis by the LTL/CTL/μ-calculus model checker LTSmin [23].

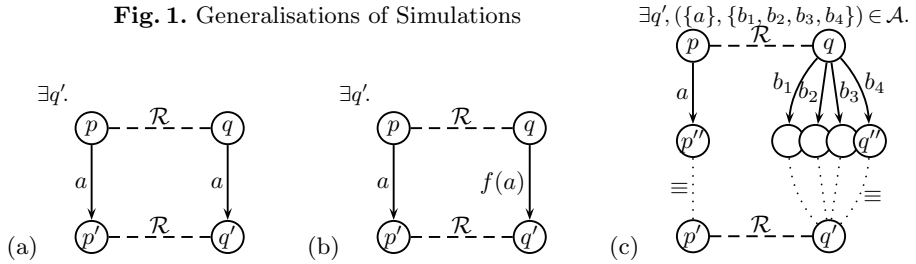
4 Comparing Transition Systems

One goal of this paper is to translate a given specification written in AWN into an mCRL2-specification. Of course the generated specification should be related to the original one, so that we know which properties that should hold for the original specification can be checked in the translated specification.

The process algebras AWN and mCRL2 generate each a labelled transition system $(S, A, \to)$, where S is the set of all closed process algebra expressions, A is the set of possible actions, and $\to \subseteq S \times A \times S$ is the labelled transition relation where the transitions $P \xrightarrow{\alpha} Q$ are derived from the sos rules.

A standard technique to compare two transitions systems is (bi)simulation (e.g. [26]). A binary relation $\mathcal{R} \subseteq S_1 \times S_2$ is a *(strong) simulation*[1] [29] between

Fig. 1. Generalisations of Simulations



transition systems $\mathcal{L}_1 = (S_1, A, \rightarrow_1)$ and $\mathcal{L}_2 = (S_2, A, \rightarrow_2)$ if it satisfies, for $a \in A$,

$$\text{if } p \,\mathcal{R}\, q \text{ and } p \xrightarrow{a}_1 p' \text{ then } \exists q'.\ q \xrightarrow{a}_2 q' \text{ and } p' \,\mathcal{R}\, q' \ .$$

Here $p \xrightarrow{a}_1 p'$ is a short-hand for $(p, a, p') \in \rightarrow_1$. A *bisimulation* is a symmetric simulation. If a bisimulation $\mathcal{R}$ with $p \,\mathcal{R}\, q$ exists then p and q are *bisimilar*.

Figure 1(a) illustrates the situation. Our definition slightly differs from the literature as it builds on two transition systems; the common definition presupposes $\mathcal{L}_1 = \mathcal{L}_2$. The definition requires an exact match of action labels. AWN and mCRL2 do not feature the same labels. For example, $R : \mathbf{*cast}(m)$, which is an action label of AWN, does not follow the syntax of mCRL2-actions.

We relax the definition of simulation and say that $\mathcal{R} \subseteq S_1 \times S_2$ is a *simulation modulo renaming* between $\mathcal{L}_1 = (S_1, A_1, \rightarrow_1)$ and $\mathcal{L}_2 = (S_2, A_2, \rightarrow_2)$ for a bijective *renaming function* $f : A_1 \to A_2$ if it satisfies, for $a \in A_1$,

$$\text{if } p \,\mathcal{R}\, q \text{ and } p \xrightarrow{a}_1 p' \text{ then } \exists q'.\ q \xrightarrow{f(a)}_2 q' \text{ and } p' \,\mathcal{R}\, q' \,;$$

see Figure 1(b). A *bisimulation modulo renaming* is a symmetric simulation modulo renaming, using f and f^{-1}, respectively. Processes $p \in S_1$, $q \in S_2$ are *bisimilar modulo renaming* if a bisimulation modulo renaming $\mathcal{R}$ with $p \,\mathcal{R}\, q$ exists.

It is well known that all safety properties are preserved under bisimilarity; and therefore also under bisimilarity modulo renaming, when the renaming function is applied to the safety property as well.[2]

Two mCRL2 processes p and q are *data congruent*, notation $p \equiv q$, if q can be obtained by replacing data expressions t occurring in p by expressions t' with $[\![t]\!] = [\![t']\!]$, i.e. evaluating to the same data value. For example $a(1{+}2) \equiv a(4{-}1)$. On AWN, we take $\equiv$ to be the identity. A (bi)simulation (modulo renaming) *up to* $\equiv$ is defined as above, but with $p' \equiv \mathcal{R} \equiv q'$ (using relational composition, denoted by juxtaposition) instead of $p' \,\mathcal{R}\, q'$. Using that $\equiv$ is a bisimulation [17], it follows from [26] that constructing a bisimulation $\mathcal{R}$ (modulo renaming) up to $\equiv$ with $p \,\mathcal{R}\, q$ suffices to show that p and q are bisimilar (modulo renaming).

In Section 5 we develop a translation between AWN and mCRL2, and we show that the translation is a bisimulation modulo renaming up to $\equiv$. However, this result only holds for encapsulated networks. When considering the other layers of AWN, a bisimulation cannot be established, not even modulo renaming.

[1] This paper does not treat weak simulations, etc.; therefore we omit the word 'strong'.
[2] See [13] for a formal definition of safety property for labelled transition systems.

The reason is the layered design of AWN. While the set R of recipients of a **broadcast** is added only on the layer of node expressions, we need to introduce this set straightaway in mCRL2. On the process layer we do not have knowledge about nodes in transmission range. To include all possibilities, we require an entire collection of mCRL2-actions. We elaborate on this in the next section.

We call a relation $\mathcal{R} \subseteq S_1 \times S_2$ an *$\mathcal{A}$-warped simulation up to $\equiv$* between transition systems $\mathcal{L}_1$ and $\mathcal{L}_2$ for a relation $\mathcal{A} \subseteq \mathcal{P}(A_1) \times \mathcal{P}(A_2)$ if it satisfies

$$\begin{array}{l}\text{if } p\,\mathcal{R}\,q \text{ and } p \xrightarrow{a}_1 p'' \text{ then } \exists \mathrm{A}_1, \mathrm{A}_2, p', q'.\\ (a \in \mathrm{A}_1,\ p'' \equiv p',\ \mathrm{A}_1\,\mathcal{A}\,\mathrm{A}_2,\ p \xrightarrow{\mathrm{A}_1}_1 \equiv p',\ q \xrightarrow{\mathrm{A}_2}_2 \equiv q' \text{ and } p'\,\mathcal{R}\,q')\,,\end{array}$$

where $p \xrightarrow{\mathrm{A}}_1 \equiv p' \Leftrightarrow_{df} \forall a \in \mathrm{A}. \exists p''.\, p \xrightarrow{a}_1 p'' \wedge p'' \equiv p'$. The definition requires a state q' such that all actions $a \in \mathrm{A}_2$ yield a transition to q', as illustrated in Figure 1(c).

An *$\mathcal{A}$-warped bisimulation up to $\equiv$* is a symmetric $\mathcal{A}$-warped simulation up to $\equiv$, using $\mathcal{A}$ and $\mathcal{A}^\smile =_{df} \{(x,y) \mid (y,x) \in \mathcal{A}\}$, respectively.

Each (bi)simulation (up to $\equiv$) is also a (bi)simulation modulo renaming (up to $\equiv$)—using the identity as renaming; and each (bi)simulation modulo renaming up to $\equiv$ is an $\mathcal{A}$-warped (bi)simulation up to $\equiv$—with $\mathcal{A} = \{(\{a\}, \{f(a)\}) \mid a \in A_1\}$.

5 From AWN to mCRL2

This section presents the formal translation from AWN-to mCRL2-processes.

Both process algebras are parameterised by the choice of an underlying data structure/abstract data type, and neither puts many restrictions on it; only the toolset associated to mCRL2 makes it more specific by predefining the most common concepts, such as integers, sets, lists, structs, etc. To ease readability, our presented translation assumes the same data structure underlying both process algebras. In particular, the translation maintains sorts—integers are mapped to integers etc. We also assume that variable names are the same. In the full version of this paper [17] we use translation functions that follow the detailed restrictions imposed on the respective data structures.

Tables 4–6 define the full translation, in recursive fashion.

Table 4 lists the translation rules for sequential processes. On this level, our translation function operates on sequential process expressions P and additionally carries two parameters: the set V of data variables maintained by P, and a valuation ζ of *some* of these variables. So $\mathrm{DOM}(\zeta) \subseteq V$. ζ evaluates all variables that in the translation to mCRL2 are turned into constants, or other closed data expressions; the variables in $V \backslash \mathrm{DOM}(\zeta)$ remain variables upon translation. exp^ζ denotes the mCRL2-expression exp with $t_{\zeta(x)}$ substituted for each $x \in \mathrm{DOM}(\zeta)$. The set V is always the domain of the valuation ξ of a sequential process (ξ, P); hence the ζ used as a parameter in the translation is only a part of ξ.

The first two equations translate **broadcast** and **groupcast**-actions in a similar fashion. Since mCRL2 does not allow to alter the number nor the type of arguments of an action, we have to add all parameters from the beginning. As a consequence the action **cast** carries three arguments: the intended destinations

Table 4. Translation function T (Sequential Processes)

$$
\begin{aligned}
\mathrm{T}_V(\zeta, \mathbf{broadcast}(ms).P) &= \textstyle\sum_{\mathtt{D}:\mathrm{Set(IP)}} \mathbf{cast}(\mathrm{IP}, \mathtt{D}, ms^\zeta)\cdot \mathrm{T}_V(\zeta, P)\\
\mathrm{T}_V(\zeta, \mathbf{groupcast}(dests, ms).P) &= \textstyle\sum_{\mathtt{D}:\mathrm{Set(IP)}} \mathbf{cast}(dests^\zeta, \mathtt{D}, ms^\zeta)\cdot \mathrm{T}_V(\zeta, P)\\
\mathrm{T}_V(\zeta, \mathbf{unicast}(dest, ms).P \blacktriangleright Q) &= \mathbf{cast}(\{dest^\zeta\}, \{dest^\zeta\}, ms^\zeta)\cdot \mathrm{T}_V(\zeta, P)\\
&\quad + \neg\mathbf{uni}(\{dest^\zeta\}, \emptyset, ms^\zeta)\cdot \mathrm{T}_V(\zeta, Q)\\
\mathrm{T}_V(\zeta, \mathbf{send}(ms).P) &= \mathbf{send}(\emptyset, \emptyset, ms^\zeta)\cdot \mathrm{T}_V(\zeta, P)\\
\mathrm{T}_V(\zeta, \mathbf{deliver}(data).P) &= \textstyle\sum_{\mathtt{ip}:\mathrm{IP}} \mathbf{del}(\mathtt{ip}, data^\zeta)\cdot \mathrm{T}_V(\zeta, P)\\
\mathrm{T}_V(\zeta, \mathbf{receive}(\mathtt{m}).P) &= \textstyle\sum_{\substack{\mathtt{D},\mathtt{D}':\mathrm{Set(IP)}\\ \mathtt{m}:\mathrm{MSG}}} \mathbf{receive}(\mathtt{D}, \mathtt{D}', \mathtt{m})\cdot \mathrm{T}_{V\cup\{\mathtt{m}\}}(\zeta^{\setminus\mathtt{m}}, P)\\
\mathrm{T}_V(\zeta, [\![\mathtt{v} := exp]\!]\ P) &= \textstyle\sum_{\mathtt{y}:\mathrm{sort(v)}}(\mathtt{y} = exp^\zeta) \rightarrow\\
&\quad (\textstyle\sum_{\mathtt{v}:\mathrm{sort(y)}}(\mathtt{v} = \mathtt{y}) \rightarrow \mathbf{t}\cdot \mathrm{T}_{V\cup\{\mathtt{v}\}}(\zeta^{\setminus\mathtt{v}}, P))\\
\mathrm{T}_V(\zeta, X(exp_1, \cdots, exp_n)) &= X(exp_1^\zeta, \cdots, exp_n^\zeta)\\
\mathrm{T}_V(\zeta, P + Q) &= \mathrm{T}_V(\zeta, P) + \mathrm{T}_V(\zeta, Q)\\
\mathrm{T}_V(\zeta, [\varphi]P) &= \textstyle\sum_{\mathrm{Fv}(\varphi)\setminus V} \varphi^\zeta \rightarrow \mathbf{t}\cdot \mathrm{T}_{V\cup\mathrm{Fv}(\varphi)}(\zeta, P)
\end{aligned}
$$

of a message (a set of addresses), the actual destinations, and the message itself. For **broadcast** the set of intended addresses is the set of all IP addresses; for **groupcast** this set is determined by the AWN-primitive. The second argument hinges on the set of reachable destinations (destinations in transmission range), which is only specified on the level of node expressions—see e.g. Rule 2 of Table 2. To allow arbitrary sets of destinations these rules use the sum operator of mCRL2 ($\sum$)—the correct set of destinations is chosen later, by using the parallel operator $\|$. For the translation we have to assume that $\mathtt{D}$ and $\mathtt{D}'$ are fresh variables; in [17] we list all required side conditions, which we skip here to ease readability. After the **broadcast**-action has been translated, the remaining process P is handled by the same translation function. The **unicast** primitive uses a similar translation in case of successful transmission, but also allows the possibility of failure, which is handled by the action $\neg$**uni**.

The translation of the **send**-primitive is straightforward; the only subtlety is that the translation has to have as many arguments as the **cast**-action, since both synchronise with **receive**—we use the empty set $\emptyset$ as dummy parameter. The **deliver**-action delivers *data* to the client; as this can happen at any network node, we sum over all possible recipients $\mathtt{ip}$. The translation of **receive** follows the style of **broadcast** and **groupcast**, and synchronises with the **cast**-action later on. Hence it needs the same number of arguments as that action; as all parameters are unknown, we sum over all of them. After the **receive**-action, the variable $\mathtt{m}$ is added to the set V of variables maintained by the AWN-process P. However, since in the mCRL2 translation it occurs under the scope of a sum operator, it is not instantiated with a concrete message in the translation of P, and hence is removed from the domain of ζ—notation $\zeta^{\setminus\mathtt{m}}$.

Since mCRL2 does not provide a primitive for assignment, the translation of $[\![\mathtt{v} := exp]\!]P$ is non-trivial. The idea behind our translation is to sum over all possible values of $\mathtt{v}$, and use a guard to pick the right value. A first rendering

Table 5. Translation function T (Defining Equation and Parallel Processes)

$$\begin{aligned}
\mathrm{T}(X(\mathtt{v}_1,\cdots,\mathtt{v}_n) \stackrel{def}{=} P) &= \big(X(\mathtt{v}_1{:}\mathrm{sort}(\mathtt{v}_1),\cdots,\mathtt{v}_n{:}\mathrm{sort}(\mathtt{v}_n)) \stackrel{\text{def}}{=} \mathrm{T}_{\{\mathtt{v}_1,\cdots,\mathtt{v}_n\}}(\emptyset,P)\big)\\
\mathrm{T}((\xi,P)) &= \mathrm{T}_{\mathrm{DOM}(\xi)}(\xi,P)\\
\mathrm{T}(P \between Q) &= \nabla_V \Gamma_{\{\mathbf{r}|\mathbf{s}\to\mathbf{t}\}}(\rho_{\{\mathbf{receive}\to\mathbf{r}\}}\mathrm{T}(P) \| \rho_{\{\mathbf{send}\to\mathbf{s}\}}\mathrm{T}(Q))\\
&\quad \text{where } V = \{\mathbf{t}, \mathbf{cast}, \neg\mathbf{uni}, \mathbf{send}, \mathbf{del}, \mathbf{receive}\}
\end{aligned}$$

of the translation rule would be $\sum_{\mathtt{v}:\mathrm{sort}(\mathtt{v})}(\mathtt{v} = exp^{\zeta}) \to X$, where X is a process to be determined. This sum-guard combination works for many cases; it fails when the expression contains the variable itself. An example is the increment of a variable: $[\![\mathtt{x} = \mathtt{x}{+}1]\!]$. To resolve this problem we use a standard technique of programming and introduce a fresh variable $\mathtt{y}$. We then split the assignment and calculate $[\![\mathtt{y} = \mathtt{x}{+}1]\!][\![\mathtt{x} = \mathtt{y}]\!]$. Both assignments are transformed into sum-guard form. Since we aim at strong bisimilarity and the assignment rule of AWN produces a silent action τ, we do something similar for mCRL2. For technical reasons[3] we cannot use a τ-action, and use an action named $\mathbf{t}$ instead.

Both AWN and mCRL2 feature process calls and an operator for (binary) choice with the same semantics; their obvious translation is given by the next two lines of Table 4. The guard of AWN translates to a guard in mCRL2. However, AWN assigns variables that occur free in φ and that are not maintained by the current process in a non-deterministic manner such that φ evaluates to true. We model the same behaviour by a sum over those variables that can be chosen freely; here the set $\mathrm{FV}(\varphi)$ contains all free variables of the Boolean formula φ. This is the only place in the translation where the parameter V is used at all. The mCRL2-expression of this rule simplifies to $\varphi^{\zeta} \to \mathbf{t}{\cdot}\mathrm{T}_V(\zeta, P)$ in case all free variables of φ occur in V.

Table 5 first presents the translation of defining equations, which is straightforward. The set V of variables maintained by P consists of the parameters $\mathtt{v}_i$ of the process name X. The table also lists the translation rules for parallel processes. The rule for (ξ, P) merely needs to initialise the set V as $\mathrm{DOM}(\xi)$. The last rule handles the (asymmetric) parallel operator of parallel processes. This operator allows and enforces synchronisation of a **send**-action on the right with a **receive**-action on the left only. For example, in the expression $(P \between Q) \between R$ the **send** and **receive**-actions of Q can communicate only with P and R, respectively, but the **receive**-actions of R, as well as the **send**-actions of P, remain available for communication with the environment. Since mCRL2 only offers a standard, symmetric parallel operator, we model the behaviour by combining renaming, communication and allow operators. By renaming **receive** to **r** in the left process and **send** to **s** in the right process we guarantee synchronisation of the corresponding actions; the communication operator renames the synchronised action into **t**, which later becomes an internal action τ. To enforce

[3] Using that $\tau|\tau = \tau$, the fourth rule of Table 3 allows any two parallel τ-transitions in mCRL2 to synchronise, which is not possible in AWN. For this reason, τ-actions in AWN are translated in an action **t** of mCRL2, which is turned into a τ only at the outermost layer, where no further parallel compositions are encountered.

Table 6. Translation function T (Network Nodes and Networks)

$$
\begin{aligned}
&\mathrm{T}(ip : P : R) = \nabla_V \Gamma_C(\mathrm{T}(P) \| G(ip, R)) \\
&\quad \text{where } V = \{\mathbf{t}, \mathbf{starcast}, \mathbf{arrive}, \mathbf{deliver}, \mathbf{connect}, \mathbf{disconnect}\} \\
&\quad \text{where } C = \{\mathbf{cast}|\overline{\mathbf{cast}} \to \mathbf{starcast}, \neg\mathbf{uni}|\overline{\neg\mathbf{uni}} \to \mathbf{t}, \\
&\qquad\qquad \mathbf{del}|\overline{\mathbf{del}} \to \mathbf{deliver}, \mathbf{receive}|\overline{\mathbf{receive}} \to \mathbf{arrive}\} \\
&\quad \text{where } G(\mathtt{ip}, \mathtt{R}) \stackrel{\text{def}}{=} \sum_{\substack{\mathtt{D},\mathtt{D}':\mathrm{Set(IP)} \\ \mathtt{m}:\mathrm{MSG}}} (\mathtt{R} \cap \mathtt{D} = \mathtt{D}') \to \overline{\mathbf{cast}}(\mathtt{D}, \mathtt{D}', \mathtt{m}) \cdot G(\mathtt{ip}, \mathtt{R}) \\
&\qquad\qquad + \sum_{\substack{\mathtt{d}:\mathrm{IP} \\ \mathtt{m}:\mathrm{MSG}}} (\mathtt{d} \notin \mathtt{R}) \to \overline{\neg\mathbf{uni}}(\{\mathtt{d}\}, \emptyset, \mathtt{m}) \cdot G(\mathtt{ip}, \mathtt{R}) \\
&\qquad\qquad + \sum_{\mathtt{data}:\mathrm{DATA}} \overline{\mathbf{del}}(\mathtt{ip}, \mathtt{data}) \cdot G(\mathtt{ip}, \mathtt{R}) \\
&\qquad\qquad + \sum_{\mathtt{ip}':\mathrm{IP}} \mathbf{connect}(\mathtt{ip}, \mathtt{ip}') \cdot G(\mathtt{ip}, \mathtt{R} \cup \{\mathtt{ip}'\}) \\
&\qquad\qquad + \sum_{\mathtt{ip}':\mathrm{IP}} \mathbf{connect}(\mathtt{ip}', \mathtt{ip}) \cdot G(\mathtt{ip}, \mathtt{R} \cup \{\mathtt{ip}'\}) \\
&\qquad\qquad + \sum_{\mathtt{ip}',\mathtt{ip}'':\mathrm{IP}} (\mathtt{ip} \notin \{\mathtt{ip}', \mathtt{ip}''\}) \to \mathbf{connect}(\mathtt{ip}', \mathtt{ip}'') \cdot G(\mathtt{ip}, \mathtt{R}) \\
&\qquad\qquad + \sum_{\mathtt{ip}':\mathrm{IP}} \mathbf{disconnect}(\mathtt{ip}, \mathtt{ip}') \cdot G(\mathtt{ip}, \mathtt{R} \backslash \{\mathtt{ip}'\}) \\
&\qquad\qquad + \sum_{\mathtt{ip}':\mathrm{IP}} \mathbf{disconnect}(\mathtt{ip}', \mathtt{ip}) \cdot G(\mathtt{ip}, \mathtt{R} \backslash \{\mathtt{ip}'\}) \\
&\qquad\qquad + \sum_{\mathtt{ip}',\mathtt{ip}'':\mathrm{IP}} (\mathtt{ip} \notin \{\mathtt{ip}', \mathtt{ip}''\}) \to \mathbf{disconnect}(\mathtt{ip}', \mathtt{ip}'') \cdot G(\mathtt{ip}, \mathtt{R}) \\
&\qquad\qquad + \sum_{\substack{\mathtt{D},\mathtt{D}':\mathrm{Set(IP)} \\ \mathtt{m}:\mathrm{MSG}}} (\mathtt{ip} \in \mathtt{D}') \to \overline{\mathbf{receive}}(\mathtt{D}, \mathtt{D}', \mathtt{m}) \cdot G(\mathtt{ip}, \mathtt{R}) \\
&\qquad\qquad + \sum_{\substack{\mathtt{D},\mathtt{D}':\mathrm{Set(IP)} \\ \mathtt{m}:\mathrm{MSG}}} (\mathtt{ip} \notin \mathtt{D}') \to \mathbf{arrive}(\mathtt{D}, \mathtt{D}', \mathtt{m}) \cdot G(\mathtt{ip}, \mathtt{R}) \\
&\mathrm{T}(M \| N) = \rho_R \nabla_V \Gamma_{\{\mathbf{arrive}|\mathbf{arrive} \to \mathbf{a}\}} \Gamma_C(\mathrm{T}(M) \| \mathrm{T}(N)) \\
&\quad \text{where } R = \{\mathbf{a} \to \mathbf{arrive}, \mathbf{c} \to \mathbf{connect}, \mathbf{d} \to \mathbf{disconnect}, \mathbf{s} \to \mathbf{starcast}\} \\
&\quad \text{where } V = \{\mathbf{a}, \mathbf{c}, \mathbf{d}, \mathbf{deliver}, \mathbf{s}, \mathbf{t}\} \\
&\quad \text{where } C = \{\mathbf{starcast}|\mathbf{arrive} \to \mathbf{s}, \mathbf{connect}|\mathbf{connect} \to \mathbf{c}, \\
&\qquad\qquad \mathbf{disconnect}|\mathbf{disconnect} \to \mathbf{d}\} \\
&\mathrm{T}([M]) = \tau_{\{\mathbf{t}\}} \nabla_V \rho_{\{\mathbf{starcast} \to \mathbf{t}\}} \Gamma_C(\mathrm{T}(M) \| H) \\
&\quad \text{where } V = \{\mathbf{t}, \mathbf{newpkt}, \mathbf{deliver}, \mathbf{connect}, \mathbf{disconnect}\} \\
&\quad \text{where } C = \{\overline{\mathbf{newpkt}}|\mathbf{arrive} \to \mathbf{newpkt}\} \\
&\quad \text{where } H \stackrel{\text{def}}{=} \sum_{\mathtt{ip}:\mathrm{IP},\mathtt{data}:\mathrm{DATA},\mathtt{dest}:\mathrm{IP}} \overline{\mathbf{newpkt}}(\{\mathtt{ip}\}, \{\mathtt{ip}\}, \mathtt{newpkt}(\mathtt{data}, \mathtt{dest})) \cdot H
\end{aligned}
$$

synchronisation, we apply the allow-operator ∇, and restrict the set of actions to those possible. Among others this disallows all proper multi-actions.

Table 6 shows the translation rules for network nodes, networks and encapsulated networks. All rules use combinations of the mCRL2-operators ∇ and Γ, similar to the last rule of Table 5. The process G is used to select the correct set of nodes receiving a message—remember that we sum over all possible sets on the level of sequential processes (see Table 4). It also introduces the primitives for changing network topologies, such as **connect**ing and **disconnect**ing two nodes. The rule for $\|$ features two Γ-operators, as mCRL2 forbids a single one to have overlapping redexes. The encapsulation allows only actions with name **newpkt**, **deliver**, **connect**, **disconnect**, as well as the 'to-be' silent action **t**. The process H handles the injection of a new data packet, where all parameters (point of injection `ip`, the destination `dest` as well the content `data` of the message) are unknown; we sum over these values.

This concludes the formal definition and explanation of the translation from the process algebra AWN into the process algebra mCRL2.

Table 7. Action relation $\mathcal{A}$

$$
\begin{aligned}
\mathcal{A} =_{df} \Big\{\ & (\{\tau\},\{\mathbf{t}\}),\ (\{\mathbf{broadcast}(\xi(ms))\},\{\mathbf{cast}([\![\mathrm{IP}]\!],[\![\hat{\mathtt{D}}]\!],[\![\xi(ms)]\!]) \mid \hat{\mathtt{D}}{:}\mathrm{Set(IP)}\}),\\
& (\{\mathbf{groupcast}(\xi(dests),\xi(ms))\},\{\mathbf{cast}([\![\xi(dests)]\!],[\![\hat{\mathtt{D}}]\!],[\![\xi(ms)]\!]) \mid \hat{\mathtt{D}}{:}\mathrm{Set(IP)}\}),\\
& (\{\mathbf{unicast}(\xi(dest),\xi(ms))\},\{\mathbf{cast}([\![\xi(\{dest\})]\!],[\![\xi(\{dest\})]\!],[\![\xi(ms)]\!])\}),\\
& (\{\neg\mathbf{unicast}(\xi(dest),\xi(ms))\},\{\neg\mathbf{uni}([\![\xi(\{dest\})]\!],[\![\emptyset]\!],[\![\xi(ms)]\!])\}),\\
& (\{\mathbf{send}(\xi(ms))\},\{\mathbf{send}([\![\emptyset]\!],[\![\emptyset]\!],[\![\xi(ms)]\!])\}),\\
& (\{\mathbf{deliver}(\xi(data))\},\{\mathbf{del}([\![\hat{\mathtt{ip}}]\!],[\![\xi(data)]\!]) \mid \hat{\mathtt{ip}}{:}\mathrm{IP}\}),\\
& (\{\mathbf{receive}(m)\},\{\mathbf{receive}([\![\hat{\mathtt{D}}]\!],[\![\hat{\mathtt{D}}']\!],m) \mid \hat{\mathtt{D}},\hat{\mathtt{D}}'{:}\mathrm{Set(IP)}\}),\\
& (\{R{:}\mathbf{*cast}(m)\},\{\mathbf{starcast}(D,R,m)\}),\ (\{ip{:}\mathbf{deliver}(d)\},\{\mathbf{deliver}(ip,d)\}),\\
& (\{H\neg K{:}\mathbf{arrive}(m)\},\{\mathbf{arrive}([\![\hat{\mathtt{D}}]\!],[\![\hat{\mathtt{D}}']\!],m) \mid {}^{\hat{\mathtt{D}},\hat{\mathtt{D}}'{:}\mathrm{Set(IP)},\ \hat{\mathtt{D}}'\subseteq\hat{\mathtt{D}}}_{H\subseteq\hat{\mathtt{D}}',\ K\cap\hat{\mathtt{D}}'=\emptyset}\}),\\
& (\{\mathbf{connect}(ip',ip'')\},\{\mathbf{connect}(ip',ip'')\}),\\
& (\{\mathbf{disconnect}(ip',ip'')\},\{\mathbf{disconnect}(ip',ip'')\}),\\
& (\{ip{:}\mathbf{newpkt}(d,dip)\},\{\mathbf{newpkt}(\{ip\},\{ip\},\mathtt{newpkt}(d,dip))\}),(\{\tau\},\{\tau\})\\
& \Big|\ m,\xi(ms){:}\mathrm{MSG};\ ip,ip',ip'',dip,dest{:}\mathrm{IP};\ d,\xi(data){:}\mathrm{DATA};\\
& \quad dests,R,D,H,K{:}\mathrm{Set(IP)};\ R\subseteq D\Big\}
\end{aligned}
$$

6 Correctness of the Translation

This section describes the relationship between AWN-specifications and their counterparts in mCRL2. We establish that our translation forms a warped bisimulation up to $\equiv$ on all layers of AWN; and a bisimulation modulo renaming up to $\equiv$ for encapsulated networks.

Theorem 6.1. The relation $\{(P,\mathrm{T}(P)) \mid P \text{ is an AWN-process}\}$ is an $\mathcal{A}$-warped simulation up to $\equiv$, where $\mathcal{A}$ is the action relation of Table 7.

Proof Sketch. We need to show that

$$\text{if } P \xrightarrow{a} P' \text{ then } \exists \mathrm{A}_1,\mathrm{A}_2.\ (P \xrightarrow{\mathrm{A}_1} P',\ \mathrm{T}(P) \xrightarrow{\mathrm{A}_2}\equiv \mathrm{T}(P'),\ \mathrm{A}_1\,\mathcal{A}\,\mathrm{A}_2 \text{ and } a\in\mathrm{A}_1)\,,$$

for all AWN action labels a. We prove this implication by structural induction on the derivation of $P \xrightarrow{a} P'$ from the inference rules of AWN.

The base cases consider all sos-rules of AWN without premises, such as the first rule of Table 2. Out of the 14 bases cases we only present the proof for this rule, and prove that there are sets A_1 and A_2 satisfying the above properties.

Since $\mathbf{broadcast}(\xi(ms)) \in \mathrm{A}_1$, Table 7 implies $\mathrm{A}_1{=}\mathbf{broadcast}(\xi(ms))$ and $\mathrm{A}_2{=}\{\mathbf{cast}([\![\xi(dests)]\!],[\![\hat{\mathtt{D}}]\!],[\![\xi(ms)]\!]) \mid \hat{\mathtt{D}}{:}\mathrm{Set(IP)}\}$. It suffices to find a derivation in mCRL2 such that $\mathrm{T}(\xi,\mathbf{broadcast}(ms).p) \xrightarrow{a} \mathrm{T}(\xi,p)$, for all $a\in\mathrm{A}_2$. For arbitrary $\hat{\mathtt{D}}$ we have

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{cast}(\mathrm{IP},\hat{\mathtt{D}},\xi(ms)) \xrightarrow{[\![\mathbf{cast}(\mathrm{IP},\hat{\mathtt{D}},\xi(ms))]\!]} \checkmark}}{\mathbf{cast}(\mathrm{IP},\hat{\mathtt{D}},\xi(ms))\cdot\mathrm{T}_{\mathrm{DOM}(\xi)}(\xi,P) \xrightarrow{\mathbf{cast}([\![\mathrm{IP}]\!],[\![\hat{\mathtt{D}}]\!],[\![\xi(ms)]\!])} \mathrm{T}_{\mathrm{DOM}(\xi)}(\xi,P)}}{\big(\mathbf{cast}(\mathrm{IP},\mathtt{D},\xi(ms))\cdot\mathrm{T}_{\mathrm{DOM}(\xi)}(\xi,P)\big)[\mathtt{D}:=\hat{\mathtt{D}}] \xrightarrow{\mathbf{cast}([\![\mathrm{IP}]\!],[\![\hat{\mathtt{D}}]\!],[\![\xi(ms)]\!])} \mathrm{T}_{\mathrm{DOM}(\xi)}(\xi,P)}}{\textstyle\sum_{\mathtt{D}:\mathrm{Set(IP)}} \mathbf{cast}(\mathrm{IP},\mathtt{D},\xi(ms))\cdot\mathrm{T}_{\mathrm{DOM}(\xi)}(\xi,P) \xrightarrow{\mathbf{cast}([\![\mathrm{IP}]\!],[\![\hat{\mathtt{D}}]\!],[\![\xi(ms)]\!])} \mathrm{T}_{\mathrm{DOM}(\xi)}(\xi,P)}}{\mathrm{T}_{\mathrm{DOM}(\xi)}(\xi,\mathbf{broadcast}(ms).P) \xrightarrow{\mathbf{cast}([\![\mathrm{IP}]\!],[\![\hat{\mathtt{D}}]\!],[\![\xi(ms)]\!])} \mathrm{T}_{\mathrm{DOM}(\xi)}(\xi,P)}}{\mathrm{T}(\xi,\mathbf{broadcast}(ms).P) \xrightarrow{\mathbf{cast}([\![\mathrm{IP}]\!],[\![\hat{\mathtt{D}}]\!],[\![\xi(ms)]\!])} \mathrm{T}(\xi,P)}$$

The validity of the first transition follows from the first rule of Table 3. The second one follows from the second rule, and a distributivity property of the interpretation function $[\![\_]\!]$ (see Section 3). To use the sos-rule for sum of Table 3 in Step 4, we rewrite the process on the left-hand side using substitution. The remaining two steps use the presented translation function (Line 1 of Table 4 and Line 2 of Table 5).

The induction step covers all rules that have at least one premise. Out of the 30 cases we present only the proof of

$$\frac{P \xrightarrow{\mathbf{broadcast}(m)} P'}{ip : P : R \xrightarrow{R:\mathbf{*cast}(m)} ip : P' : R}$$

Table 7 allows $\mathrm{A}_1 = \{R:\mathbf{*cast}(m)\}$ and $\mathrm{A}_2 = \{\mathbf{starcast}(\mathrm{IP}, R, m)\}$, choosing $D = \mathrm{IP}$. The induction step is proven by providing a derivation in mCRL2 for $\mathrm{T}(ip:P:R) \xrightarrow{a} \mathrm{T}(ip:P':R)$, for all $a \in A_2$. First, we analyse the process G.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\overline{\mathbf{cast}}(\mathrm{IP}, R, t_m) \xrightarrow{[\![\overline{\mathbf{cast}}(\mathrm{IP},R,t_m)]\!]} \surd}}{\overline{\mathbf{cast}}(\mathrm{IP}, R, t_m)\cdot G(ip, R) \xrightarrow{\overline{\mathbf{cast}}(\mathrm{IP},R,m)} G(ip, R)}}{(R \cap \mathrm{IP} = R) \to \overline{\mathbf{cast}}(\mathrm{IP}, R, t_m)\cdot G(ip, R) \xrightarrow{\overline{\mathbf{cast}}(\mathrm{IP},R,m)} G(ip, R)}}{(c' \to \overline{\mathbf{cast}}(\mathtt{D}, \mathtt{D}', \mathtt{m})\cdot G(ip, R))[\mathtt{D}{:=}\mathrm{IP}, \mathtt{D}'{:=}R, \mathtt{m}{:=}t_m] \xrightarrow{\overline{\mathbf{cast}}(\mathrm{IP},R,m)} G(ip, R)}}{\sum_{\substack{\mathtt{D},\mathtt{D}':\mathrm{Set(IP)}\\ \mathtt{m}:\mathrm{MSG}}} c' \to \overline{\mathbf{cast}}(\mathtt{D}, \mathtt{D}', \mathtt{m})\cdot G(ip, R) \xrightarrow{\overline{\mathbf{cast}}(\mathrm{IP},R,m)} G(ip, R)}}{\sum_{\substack{\mathtt{D},\mathtt{D}':\mathrm{Set(IP)}\\ \mathtt{m}:\mathrm{MSG}}} c' \to \overline{\mathbf{cast}}(\mathtt{D}, \mathtt{D}', \mathtt{m})\cdot G(ip, R) + S' \xrightarrow{\overline{\mathbf{cast}}(\mathrm{IP},R,m)} G(ip, R)}}{(\sum_{\substack{\mathtt{D},\mathtt{D}':\mathrm{Set(IP)}\\ \mathtt{m}:\mathrm{MSG}}} c \to \overline{\mathbf{cast}}(\mathtt{D}, \mathtt{D}', \mathtt{m})\cdot G(\mathtt{ip}, \mathtt{R}) + S)[\mathtt{ip}{:=}ip, \mathtt{R}{:=}R] \xrightarrow{\overline{\mathbf{cast}}(\mathrm{IP},R,m)} G(ip, R)}}{G(ip, R) \xrightarrow{\overline{\mathbf{cast}}(\mathrm{IP},R,m)} G(ip, R)}$$

where S is an expression equal to all summands of G except the first one, and $S' = S[\mathtt{ip}{:=}ip, \mathtt{R}{:=}R]$. Moreover, $c = (\mathtt{R} \cap \mathtt{D} = \mathtt{D}')$ and $c' = (R \cap \mathtt{D} = \mathtt{D}')$. We use '$R$' and 'IP' as data values as well as expressions denoting these, so $[\![R]\!] = R$ and $[\![\mathrm{IP}]\!] = \mathrm{IP}$.

As for the previous derivation the first two steps follow from the first two rules of Table 3, using $[\![t_m]\!] = m$. The following step applies the rule for guard (sixth rule in Table 3), using $[\![R \cap \mathrm{IP} = R]\!] = \mathrm{true}$ as side condition. As before we use substitution such that we can apply the sum operator. We then use the rule of binary choice (third one in Table 3) and substitution again. The final step applies the recursion rule of mCRL2 (last one in the table).

Since there is only one pair $(\mathrm{B}_1, \mathrm{B}_2) \in \mathcal{A}$ with $\mathbf{broadcast}(m) \in \mathrm{B}_1$ (see Table 7), $\mathrm{T}(P) \xrightarrow{\mathbf{cast}(\mathrm{IP},R,m)} \mathrm{T}(P')$, using the induction hypothesis. We combine this fact with the conclusion of the derivation above.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overset{\text{Induction hypothesis}}{\mathrm{T}(P) \xrightarrow{\mathbf{cast}(\mathrm{IP},R,m)} \mathrm{T}(P')} \quad G(ip,R) \xrightarrow{\overline{\mathbf{cast}}(\mathrm{IP},R,m)} G(ip,R)}{\mathrm{T}(P)\|G(ip,R) \xrightarrow{\mathbf{cast}(\mathrm{IP},R,m)|\overline{\mathbf{cast}}(\mathrm{IP},R,m)} \mathrm{T}(P')\|G(ip,R)}}{\Gamma_C(\mathrm{T}(P)\|G(ip,R)) \xrightarrow{\gamma_C(\mathbf{cast}(\mathrm{IP},R,m)|\overline{\mathbf{cast}}(\mathrm{IP},R,m))} \Gamma_C(\mathrm{T}(P')\|G(ip,R))}}{\Gamma_C(\mathrm{T}(P)\|G(ip,R)) \xrightarrow{\mathbf{starcast}(\mathrm{IP},R,m)} \Gamma_C(\mathrm{T}(P')\|G(ip,R))}}{\nabla_V\Gamma_C(\mathrm{T}(P)\|G(ip,R)) \xrightarrow{\mathbf{starcast}(\mathrm{IP},R,m)} \nabla_V\Gamma_C(\mathrm{T}(P')\|G(ip,R))}}{\mathrm{T}(ip:P:R) \xrightarrow{\mathbf{starcast}(\mathrm{IP},R,m)} \mathrm{T}(ip:P':R)}$$

The derivation is straightforward, using the synchronisation rule of Table 3 and the rules for mCRL2-operators listed on Page 7. This finishes the induction step for the **broadcast**-rule. □

A full and detailed proof can be found in [17].

We have shown that translated processes simulate original processes. We now turn to the opposite direction.

Theorem 6.2. The relation $\{(\mathrm{T}(P), P) \mid P \text{ is an AWN-process}\}$ is an $\mathcal{A}^{\smile}$-warped simulation up to $\equiv$, where $\mathcal{A}^{\smile}$ is the converse action relation of Table 7.

Similar to Theorem 6.1, the proof is by structural induction. In contrast to the above proof, the proof of Theorem 6.2 is more complicated. The reason is that the relation $\mathcal{A}$ is a function, whereas $\mathcal{A}^{\smile}$ is not. As a consequence the individual cases (base cases and induction steps) contain several case distinctions. For example, an action labelled $\mathbf{cast}(\llbracket \mathtt{D} \rrbracket, \llbracket \mathtt{D}' \rrbracket, \llbracket \mathtt{m} \rrbracket)$ could stem from a **broadcast**, a **groupcast** or a **unicast**-action in AWN. The action labelled **t** is even worse: it can stem from an internal action τ, the action **starcast**, from a synchronisation $\mathbf{uni}|\neg\mathbf{uni}$, etc. Again, the full proof can be found in [17].

Corollary 6.3. The relation $\{(P, \mathrm{T}(P)) \mid P \text{ is an AWN-process}\}$ is an $\mathcal{A}^{\smile}$-warped bisimulation up to $\equiv$.

Using this result we are now ready to prove the main theorem.

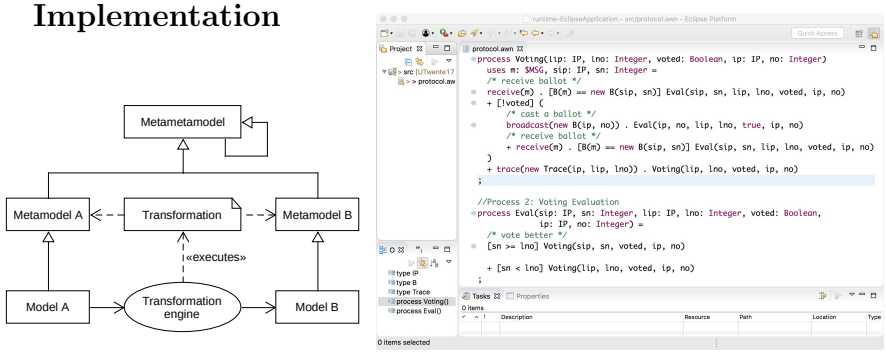
Theorem 6.4. The relation $\{(P, \mathrm{T}(P)) \mid P$ is an encapsulated network expression in AWN$\}$[4] is a bisimulation modulo renaming up to $\equiv$ w.r.t. to the bijective renaming function f, defined as

$$\mathrm{f}(a) =_{df} \begin{cases} \tau & \text{if } a = \tau \\ \mathbf{deliver}(ip, d) & \text{if } a = ip{:}\mathbf{deliver}(d) \\ \mathbf{connect}(ip', ip'') & \text{if } a = \mathbf{connect}(ip', ip'') \\ \mathbf{disconnect}(ip', ip'') & \text{if } a = \mathbf{disconnect}(ip', ip'') \\ \mathbf{newpkt}(\{ip\}, \{ip\}, \mathtt{newpkt}(d, dip)) & \text{if } a = ip{:}\mathbf{newpkt}(d, dip)\ . \end{cases}$$

Given Corollary 6.3, the proof is fairly straightforward. As we have established a bisimulation between specifications written in AWN and their translated counterparts in mCRL2 we can now use the mCRL2 toolset to analyse safety properties, because such properties are preserved under bisimulation.

[4] P is an encapsulated network expression when it has the form $[M]$.

7 Implementation



(a) MDE Basics (b) Eclipse Plugin

Fig. 2. Implementing AWN to mCRL2

We have implemented our translation as an Eclipse-plugin, available at `http://hoefner-online.de/ifm18/` with a description in [36]. The project, written in Java, is based on the principles of *Model-Driven Engineering* (MDE) [24,33].

MDE efficiently combines domain-specific languages with transformation engines and generators. The MDE approach aims to increase productivity by maximising compatibility between systems, via reuse of standardised models; its basic design principle is sketched in Figure 2(a). Based on the idea of "everything is a model", the overall goal is to transform a model A into another model B. The syntax of a model is usually a domain-specific language (DSL), or in terms of MDE a metamodel. All metamodels have to conform to the syntax of a metametamodel—the syntax of a metametamodel can be expressed by the metametamodel itself. Using a commonly available metametamodel, such as the one introduced by the Object Management Group [34], makes metamodels compatible. Abstract transformation rules defined between metamodels are used to transform models.

In our setting Model A is an AWN-specification, and Model B its translated counterpart in mCRL2. The metamodels A and B correspond to the syntax of the two process algebras.

To ensure usability and compatibility, our implementation builds on existing MDE frameworks and techniques. We use the *Eclipse Modeling Framework* (EMF) [35], which includes a metametamodel (Ecore) for describing metamodels. The open-source framework *Xtext* [4] provides infrastructure to create parsers, linkers, and typecheckers. By using Eclipse and Xtext, we are able to provide a user-friendly GUI; see Figure 2(b). To define our model transformation we use *QVT* (Query/View/Transformation) [28,27]; in particular the imperative model transformation language QVTo.

For development purposes and as a sanity check we implement and translate the leader election protocol, presented in Sections 2 and 3. Both the input and output are small enough to be manually inspected and analysed.

We use the mCRL2 toolset—in particular the provided model checker—to determine whether the nodes of a 5-node network eventually agree on a leader. In terms of CTL [9], we want to check

$$\mathrm{A}\Diamond\ (\varphi_{\texttt{lips}}) \qquad \text{and} \qquad \mathrm{A}\Diamond\ \mathrm{A}\Box\ (\varphi_{\texttt{lips}})\ ,$$

where $\varphi_{\texttt{lips}}$ is a propositional (state) formula checking the equality of the values assigned to the nodes' variables `lip`. The former equation states that at some point in time all nodes agree on a common leader; the latter strengthens the statement and requires that the nodes agree on a leader permanently. The mCRL2 toolset only checks formulas written in the modal μ-calculus [32], or (generalised) Hennessy-Milner logic [19]; so we have to translate the above formula. Moreover, because state variables are hidden from direct analysis by mCRL2, we modify the protocol by including an extra, otherwise inert, action **trace**(`ip`, `lip`). It reveals the current choice of leader of a particular node, when added as a 'self-loop' to Process 1. The property can then be specified by requiring that all traces that exclude **trace**-actions eventually can only do exactly one **trace** action for each `ip` with matching values for the `lip` argument. As this is the case, the protocol is correct.

We now analyse a variant of the leader election protocol in which the operator $\geq$ in Process 2 (Line 1) is replaced by $>$, and $<$ by $\leq$. Interestingly, the property under consideration does *not* hold. We leave it to the reader to find the reason.

8 Case Study: The AODV Routing Protocol

To further test our framework, we translate the Ad hoc On-Demand Distance Vector (AODV) routing protocol [30], which two of the authors together with colleagues formalised in AWN before [12,16].

AODV is a reactive protocol, i.e., routes are established on demand, only when needed. It is a widely-used routing protocol designed for Mobile Ad-hoc Networks (MANETs) and Wireless Mesh Networks (WMNs). The protocol is one of the four protocols standardised by the IETF MANET working group, and forms the basis of new WMN protocols, including the Hybrid Wireless Mesh Protocol (HWMP) in the IEEE 802.11s wireless mesh network standard [22].

The AODV routing protocol is specified in the form of an RFC [30], which is the de facto standard. However, it has been shown that the standard contains several ambiguities, contradictions, and cases of underspecification. [12]

To overcome these deficiencies, two of the authors, together with other colleagues, obtained the first rigorous formalisation of the AODV routing protocol [12,16], using the process algebra AWN. The specification consists of about 150 lines of AWN-code, split over seven processes, and around 35 functions working on a customised data structure, including routing tables.

The specification, which is available online, is translated into mCRL2, using our framework. It is not the purpose of this paper to perform a proper analysis of this protocol; we merely illustrate the potential of our framework, namely that it can be used to analysis protocols used in modern networks.

Using the translated specification, we analyse a very weak form of the *packet delivery* property [12], which, in generalised Hennessy-Milner logic, is described as

$$[\mathrm{true}^*\cdot\mathbf{trace}(\texttt{newpkt}(\texttt{dip},\texttt{data}))]\ [\neg(\mathbf{deliver}(\texttt{dip},\texttt{data})^*]\ \langle\mathrm{true}^*\rangle \\ \langle\mathbf{deliver}(\mathit{dip},\mathit{data})\rangle\ \mathrm{true}\ .$$

The property states that whenever a new packet intended for `dip` is injected to the system, modelled by the action labelled **trace**(`newpkt(dip, data)`), then, as long as it has not been delivered yet, it remains possible that this very packet will be delivered in the future. AODV uses a series of control messages to establish a route between source and destination before actually sending the `data`-packet. Similar to the leader election protocol, the AODV specification is modified by inserting a **trace**-action that makes the detection of a `newpkt`-submission to the AODV process visible to mCRL2.

We model a static linear network of three nodes and manually insert two new packets. The mCRL2 toolset checks the packet-delivery property against the given network, and detects a counter example showing that AODV control messages can interfere. Thus, the packet-delivery property does not hold, confirming an analysis done by pen-and-paper [12].

9 Conclusion

In this paper we have developed and implemented a translation from the process algebra AWN into the process algebra mCRL2. The translation allows an automatic analysis of AWN-specifications, using the sophisticated toolset of mCRL2. In contrast to many approaches that transform one formalism into another, we have proven that the translation respects strong bisimilarity (modulo renaming). Therefore we guarantee that all safety properties can be checked on the translated specification and that the (positive/negative) outcome carries over to the AWN-specification. Besides, establishing the relationship in a formal way helped us in finding problems in our translation that we otherwise would have missed. For example, in an early version of the translation function we missed the introduction of a fresh variable `y` when translating an assignment (Line 7 of Table 4).

We have used our framework, which is available online, to analyse a simple leader election protocol, as well as the packet-delivery property of the AODV routing protocol.

Directions for future work are manifold. (a) Having tools for automated analysis available, we can now analyse further protocols, such as a fragmentation and reassembly protocol running on top of a CAN-bus [15]. (b) T-AWN is an extension of AWN by timing constructs. [7] Since mCRL2 supports time as well, it would be interesting to extend our translation of Section 5. Since the timing constructs of T-AWN are fairly complex, this may be a challenging task; in particular if the result on strong bisimulation should be maintained. (c) We want to make use of more highly sophisticated off-the-shelf tools, such as Isabelle/HOL and Uppaal. As our framework follows the methodology of Model-Driven Engineering, implementing translations into other formalisms should be straightforward—proving a bisimulation result is a different story, though. (d) It has been shown that bisimulations do not preserve all liveness properties. [14] We want to come up with a concept that preserves both safety and liveness properties, followed by an adaptation of our translation.

References

1. G. Behrmann, A. David, K.G. Larsen, P. Pettersson & Wang Yi (2011): *Developing* Uppaal *over 15 years*. *Software - Practice and Experience* 41(2), pp. 133–142, doi:10.1002/spe.1006.
2. G. Behrmann, A. David & K.G. Larsen (2004): *A Tutorial on* Uppaal. In M. Bernardo & F. Corradini, eds.: *Formal Methods for the Design of Real-Time Systems*, LNCS 3185, Springer, pp. 200–236, doi:10.1007/978-3-540-30080-9_7.
3. J.A. Bergstra & J.W. Klop (1986): *Algebra of Communicating Processes*. In J.W. de Bakker, M. Hazewinkel & J.K. Lenstra, eds.: *Mathematics and Computer Science*, CWI Monograph 1, North-Holland, pp. 89–138.
4. L. Bettini (2016): *Implementing Domain-Specific Languages with Xtext and Xtend*, 2nd edition. Packt Publishing.
5. T. Bolognesi & E. Brinksma (1987): *Introduction to the ISO Specification Language LOTOS*. *Computer Networks* 14, pp. 25–59, doi:10.1016/0169-7552(87)90085-7.
6. T. Bourke, R.J. van Glabbeek & P. Höfner (2016): *Mechanizing a Process Algebra for Network Protocols*. *Journal of Automated Reasoning* 56(3), pp. 309–341, doi:10.1007/s10817-015-9358-9.
7. E. Bres, R.J. van Glabbeek & P. Höfner (2016): *A Timed Process Algebra for Wireless Networks with an Application in Routing (extended abstract)*. In P. Thiemann, ed.: *European Symposium on Programming (*ESOP 2016*)*, LNCS 9632, Springer, pp. 95–122, doi:10.1007/978-3-662-49498-1_5.
8. S. Cranen, J.F. Groote, J.J.A. Keiren, F.P.M. Stappers, E.P. de Vink, W. Wesselink & T.A.C. Willemse (2013): *An Overview of the mCRL2 Toolset and Its Recent Advances*. In N. Piterman & S.A. Smolka, eds.: TACAS '13, LNCS 7795, Springer, pp. 199–213, doi:10.1007/978-3-642-36742-7_15.
9. E.A. Emerson & E.M. Clarke (1982): *Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons*. *Science of Computer Programming* 2(3), pp. 241–266, doi:10.1016/0167-6423(83)90017-5.
10. A. Fehnker, R.J. van Glabbeek, P. Höfner, A.K. McIver, M. Portmann & W.L. Tan (2012): *Automated Analysis of AODV using UPPAAL*. In C. Flanagan & B. König, eds.: TACAS'12, LNCS 7214, pp. 173–187, doi:10.1007/978-3-642-28756-5_13.
11. A. Fehnker, R.J. van Glabbeek, P. Höfner, A.K. McIver, M. Portmann & W.L. Tan (2012): *A Process Algebra for Wireless Mesh Networks*. In H. Seidl, ed.: *European Symposium on Programming* (ESOP '12), LNCS 7211, Springer, pp. 295–315, doi:10.1007/978-3-642-28869-2_15.
12. A. Fehnker, R.J. van Glabbeek, P. Höfner, A.K. McIver, M. Portmann & W.L. Tan (2013): *A Process Algebra for Wireless Mesh Networks used for Modelling, Verifying and Analysing AODV*. At `http://arxiv.org/abs/1312.7645`.
13. R.J. van Glabbeek (2010): *The Coarsest Precongruences Respecting Safety and Liveness Properties*. In C.S. Calude & V. Sassone, eds.: *Theoretical Computer Science* (TCS '10), 323, Springer, pp. 32–52, doi:10.1007/978-3-642-15240-5_3.
14. R.J. van Glabbeek (2016): *Ensuring Liveness Properties of Distributed Systems (A Research Agenda)*. Position paper. At `https://arxiv.org/abs/1711.04240`.
15. R.J. van Glabbeek & P. Höfner (2017): *Split, Send, Reassemble: A Formal Specification of a CAN Bus Protocol Stack*. In H. Hermanns & P. Höfner, eds.: *Models for Formal Analysis of Real Systems* (MARS'17), EPTCS 244, Open Publishing Association, pp. 14–52, doi:10.4204/EPTCS.244.2.
16. R.J. van Glabbeek, P. Höfner, M. Portmann & W.L. Tan (2016): *Modelling and Verifying the AODV Routing Protocol*. *Distributed Computing* 29(4), pp. 279–315, doi:10.1007/s00446-015-0262-7.

17. R.J. van Glabbeek, P. Höfner & D. van der Wal (2018): *Analysing AWN-specifications using mCRL2*. Technical Report, Data61, CSIRO. To appear.
18. J.F. Groote & M.R. Mousavi (2014): *Modeling and analysis of communicating systems*. MIT Press.
19. M. Hennessy & R. Milner (1985): *Algebraic laws for nondeterminism and concurrency*. *Journal of the ACM* 32(1), pp. 137–161, doi:10.1145/2455.2460.
20. C.A.R. Hoare (1985): *Communicating Sequential Processes*. Prentice Hall.
21. Y.-L. Hwong, J.A. Keiren, V.J.J. Kusters, S.J.J. Leemans & T.A.C. Willemse (2013): *Formalising and analysing the control software of the Compact Muon Solenoid Experiment at the Large Hadron Collider*. *Science of Computer Programming* 78(12), pp. 2435–2452, doi:10.1016/j.scico.2012.11.009.
22. IEEE (2011): *IEEE Standard for Information Technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 10: Mesh Networking*. At `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6018236`.
23. G. Kant, A. Laarman, J. Meijer, J. van de Pol, S. Blom & T. van Dijk (2015): *LTSmin: High-Performance Language-Independent Model Checking*. In C. Baier & C. Tinelli, eds.: *Tools and Algorithms for the Construction and Analysis of Systems* (TACAS'15), LNCS 9035, pp. 692–707, doi:10.1007/978-3-662-46681-0_61.
24. S. Kent (2002): *Model driven engineering*. In M. Butler, L. Petre & K. Sere, eds.: *Integrated Formal Methods* (iFM'02), LNCS 2335, Springer, pp. 286–298, doi:10.1007/3-540-47884-1_16.
25. S.P. Luttik (1997): *Description and Formal Specification of the Link Layer of P1394*. In I. Lovrek, ed.: *2nd International Workshop on Applied Formal Methods in System Design*, pp. 43–56.
26. R. Milner (1989): *Communication and Concurrency*. Prentice Hall.
27. S. Nolte (2010): *QVT – Operational Mappings: Modellierung mit der Query Views Transformation*. Springer, doi:10.1007/978-3-540-92293-3.
28. Object Management Group (2011): *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*. `http://www.omg.org/spec/QVT/`.
29. D. Park (1981): *Concurrency and automata on infinite sequences*. In P. Deussen, ed.: 5^{th} GI Conference on *Theoretical Computer Science*, LNCS 104, Springer, pp. 167–183, doi:10.1007/BFb0017309.
30. C.E. Perkins, E.M. Belding-Royer & S. Das (2003): *Ad hoc On-Demand Distance Vector (AODV) Routing*. RFC 3561 (Experimental), Network Working Group. At `http://www.ietf.org/rfc/rfc3561.txt`.
31. G.D. Plotkin (2004): *A Structural Approach to Operational Semantics*. *Journal of Logic and Algebraic Programming* 60–61, pp. 17–139. At `http://dx.doi.org/10.1016/j.jlap.2004.05.001`. Originally appeared in 1981.
32. V.R. Pratt (1981): *A Decidable mu-Calculus*. In: *Foundations of Computer Science* (FOCS'81), IEEE Computer Society, pp. 421–427, doi:10.1109/SFCS.1981.4.
33. D.C. Schmidt (2006): *Model-Driven Engineering*. *Computer* 39(2), pp. 25–31, doi:10.1109/MC.2006.58.
34. R. Soley & the OMG Staff Strategy Group (2000): *Model Driven Architecture*. `http://www.omg.org/~soley/mda.html`.
35. D. Steinberg, F. Budinsky, M. Paternostro & E. Merks (2009): *EMF: Eclipse Modeling Framework 2.0*, 2nd edition. Addison-Wesley.
36. D. van der Wal (2018): *Modeling AWN networks with an mCRL2 back end*. Master's thesis, University of Twente.