Topology-dependent Abstractions of Broadcast Networks

Sebastian Nanz, Flemming Nielson, and Hanne Riis Nielson

Informatics and Mathematical Modelling
Technical University of Denmark
{nanz,nielson,riis}@imm.dtu.dk

Abstract

Broadcast semantics poses significant challenges over point-to-point communication when it comes to formal modelling and analysis. Current approaches to analysing broadcast networks have focused on fixed connectivities, but this is unsuitable in the case of wireless networks where the dynamically changing network topology is a crucial ingredient. In this paper we develop a static analysis that automatically constructs an abstract transition system, labelled by actions and connectivity information, to yield a mobility-preserving finite abstraction of the behaviour of a network expressed in a process calculus with asynchronous local broadcast. Furthermore, we use model checking based on a 3-valued temporal logic to distinguish network behaviour which differs under changing connectivity patterns.

1 Introduction

Broadcast communication, in contrast to point-to-point message passing, is employed in a wide range of networking paradigms such as Ethernet and wireless LAN, mobile telephony, or mobile ad-hoc networks. These can be further distinguished into approaches where broadcast is taken to be global, i.e. all nodes of the network receive a broadcast message, or local, such that only neighbours of the broadcasting node are able to receive. In order to obtain a formal model for the latter case, the network topology has to be encoded by the chosen modelling formalism to express the notion of a neighbourhood. Furthermore, the connectivity may change over time, caused by node mobility or similar changes in environment conditions which are not controlled by the nodes' protocol actions.

This mix of broadcast behaviour and mobility has turned out to be a challenge for automated verification and analysis techniques. For instance, model checking of mobile ad-hoc networks, in a line of work started by [2], has remained limited to fixed connectivities. In our previous work on static analysis of mobile ad-hoc networks [15], topology changes are considered in the modelling, but abstracted into a fixed representation for the sake of the analysis, hence achieving a safe description of the network, but losing the ability to expose network behaviour related to connectivity change.

In this paper we address these deficiencies by defining *abstract transition systems* which provide finite abstractions of the behaviour of broadcast networks specified in the broadcast calculus bKlaim, which is also introduced in this paper. The abstractions preserve mobility in the sense that their transitions depend on connectivity information, and hence reflect changes in connectivity. We present a 3-valued interpretation of formulae of Action Computation Tree Logic (ACTL) [17] on abstract transition systems, which correctly captures the nature of the abstraction by evaluating to "unknown" whenever the abstraction prevents definite conclusions about the concrete behaviour of the related bKlaim network.

We also show how abstract transition systems can be algorithmically constructed from networks specified in bKlaim. This is done using a static analysis, based on the idea of Monotone Frameworks [18], which also gives us fine-grained control over the coarseness of the abstraction. This analysis has been implemented, and we show how the complete framework enables us expose the influence of the network dynamics on the resulting network state.

The conference publication [16] contains part of the material of this paper in preliminary form. The remainder of the paper is structured as follows. In §2 we present the syntax and operational semantics of bKlaim. In §3 we introduce abstract transition systems and describe 3-valued ACTL and its relation to the concrete transition system of bKlaim. In §4 we define a Control Flow Analysis to describe the name bindings arising from message passing. The result of this analysis is passed as a parameter to a Monotone Framework, defined in §5, which allows us to approximate how analysis information evolves as a result of network evolution steps. In §6 we develop a worklist algorithm that uses the Monotone Framework to construct abstract transition systems for bKlaim networks. We conclude with a discussion of related and future work in §7.

2 bKlaim

Process calculi of the Klaim family [1] are centred around the *tuple space* paradigm in which a system is comprised by a distributed set of nodes that communicate by placing tuples into and getting tuples from one or more shared tuple spaces. In this paper we use this basic paradigm to model systems communicating via *local broadcast*, i.e. only nodes within the neighbourhood of the broadcasting node may receive a sent message tuple; this distinguishes bKlaim from the broadcast calculus CBS [25], where all broadcast is global. In contrast to the standard Klaim semantics, where tuple spaces are shared resources among all nodes, we instrument this approach for the modelling of local broadcast: broadcast messages are output into the tuple spaces of neighbouring nodes to the sending node, where they can be picked up only by the processes residing at the respective locations; this yields an *asynchronous* version of local broadcast, in contrast to the calculi CBS$^\sharp$ [15] and CMN [13] which both feature synchronous behaviour. The notion of neighbourhood is expressed by *connectivity graphs*, which specify the locations currently connected with a sender and may change during the evolution of the network.

2.1 Syntax

$N ::=$	$l :: P$	located node	$a^\ell ::=$	$\mathsf{bcst}^\ell(t)$	broadcast output
$\mid$	$l :: S$	located tuple space	$\mid$	$\mathsf{out}^\ell(t)$	output
$\mid$	$N_1 \parallel N_2$	net composition	$\mid$	$\mathsf{b\text{-}eval}^\ell(P)$	broadcast migration
$P ::=$	nil	null process	$\mid$	$\mathsf{in}^\ell(T)$	input
$\mid$	$a^\ell.P$	action prefixing	$\mid$	$\mathsf{read}^\ell(T)$	read
$\mid$	$P_1 \mid P_2$	parallel composition	$\mid$	$\mathsf{abs}^\ell(T)$	absent
$\mid$	A	process invocation	$T ::=$	$F \mid F, T$	templates
			$F ::=$	$f \mid !x$	template fields
			$t ::=$	$f \mid f, t$	tuples
			$f ::=$	$v \mid l \mid x$	tuple fields

Table 1: Syntax of bKlaim

The bKlaim calculus comprises three parts: networks, processes, and actions. Networks give the overall structure in which processes and tuple spaces are located, and processes execute by performing actions. An overview of the syntax is shown in Table 1.

Tuples are finite lists of tuple fields, which comprise values $v \in \mathbf{Val}$, locations $l \in \mathbf{Loc}$, and variables $x \in \mathbf{Var}$. We assume in general that locations are just distinguished values, i.e. $\mathbf{Loc} \subseteq \mathbf{Val}$. *Templates* are used as patterns to select tuples in a tuple space. They are finite lists of tuple fields and formal fields $!x$ which are used to bind variables to values ($x \in \mathbf{Var}$); within a template, if $x \in \mathbf{Var}$ occurs in a formal field, it must not occur in another formal field or as a variable as well. The sets $fv(t)$ and $fv(T)$ containing the free variables of tuple t and template T are defined as usual, and the definition of *fv* can be extended to actions and processes. In contrast, all values are free as there are no binding statements for them.

Networks consist of located processes and tuple spaces. In contrast to Klaim, a tuple space S is taken to be a multiset (rather than a set) of tuples, i.e. a total map from the set of tuples into $\mathbb{N}_0$. We say that a tuple t is in the domain $dom(S)$ of S if $S(t) > 0$, and use the following notation to express that a copy of tuple t is added to or removed from a multiset S:

$$S[t]^{\uparrow} = \lambda u. \begin{cases} S(u) + 1 & \text{if } u = t \\ S(u) & \text{otherwise} \end{cases}$$

$$S[t]^{\downarrow} = \lambda u. \begin{cases} S(u) - 1 & \text{if } u = t \wedge S(u) > 0 \\ S(u) & \text{otherwise} \end{cases}$$

We also introduce below a well-formedness condition which ensures that there is exactly one tuple space per location. This is because tuple spaces in bKlaim are not seen as freely shared among nodes, but as private components (stores) associated with the processes residing at the same location. Furthermore, having only one tuple space per location enables us to introduce the $\mathsf{abs}^{\ell}(T)$-action, which executes only if there is *no* tuple matching T available at the location.

A *process* is either the terminated process nil, a process prefixed with an action to be executed, a parallel composition, or a process invocation to express recursive behavior. Process definitions are of the form $A \triangleq P$, where P is closed, i.e. contains no free variables. As an abbreviation, we may sometimes use the notation $A(t) \triangleq P$ and have P parameterized in the free variables of t.

Actions are equipped with labels $\ell \in \mathbf{Lab}$ which are necessary for the analysis of §5. The action $\mathsf{bcst}^{\ell}(t)$ places a tuple t into the set of tuple spaces belonging to the current neighbors of the sending node, thus describing local broadcast. Neighborhoods are defined at the semantic level via the notion of connectivity graphs. The action $\mathsf{out}^{\ell}(t)$ models the output of a tuple to the private tuple space of the node performing this action. The action $\mathsf{b\text{-}eval}^{\ell}(P)$ remotely evaluates a process P at all nodes in the current neighborhood. Using $\mathsf{in}^{\ell}(T)$ and $\mathsf{read}^{\ell}(T)$, processes can retrieve tuples which match the template T from their private tuple space, either removing it or leaving it in place respectively. Action $\mathsf{abs}^{\ell}(T)$ describes the absence of any tuple matching the template T at the private tuple space; for the process $\mathsf{abs}^{\ell}(T).P$ we require $fv(T) \cap fv(P) = \emptyset$ because if the continuation P is executed, no tuple t will have been matched against T. Note that there is no statement corresponding to Klaim's creation of new locations $\mathsf{newloc}(l)$ because we want to deal with a given set of located nodes which cannot spawn themselves by process actions.

Example 2.1 We describe a simple protocol for information retrieval in mobile ad-hoc networks. A mobile ad-hoc network is a special kind of wireless network, where participating nodes form temporary multi-hop connections and may act as both host and router, i.e. both sending own requests and relaying messages for others. The protocol is specified in bKlaim as follows:

$$\begin{aligned}
&\mathit{Snd}(x) &&\triangleq \mathsf{bcst}^{1}(\mathtt{ask}, x).\mathit{Rec}(x)\\
&\mathit{Rec}(x) &&\triangleq \mathsf{in}^{2}(\mathtt{has}, !l, x, !y).\mathit{Rec}(x)\\
&\mathit{Prc}(l) &&\triangleq \mathsf{in}^{3}(\mathtt{ask}, !x).(\mathsf{in}^{4}(x, !y).\mathsf{bcst}^{5}(\mathtt{has}, l, x, y) \mid \mathsf{bcst}^{6}(\mathtt{ask}, x).\mathit{Prc}(l))\\
&\mathit{Rel} &&\triangleq \mathsf{in}^{7}(\mathtt{has}, !l, !x, !y).\mathsf{bcst}^{8}(\mathtt{has}, !l, x, y).\mathit{Rel}
\end{aligned}$$

$$\begin{aligned}
\mathit{Net} \quad &\triangleq \mathtt{l}_1 :: \mathit{Snd}(\mathtt{t}) \parallel \mathtt{l}_2 :: (\mathit{Prc}(\mathtt{l}_2) \mid \mathit{Rel}) \parallel \mathtt{l}_2 :: [[\mathtt{t}, \mathtt{i}_2] \mapsto 1]\\
&\parallel \mathtt{l}_3 :: (\mathit{Prc}(\mathtt{l}_3) \mid \mathit{Rel}) \parallel \mathtt{l}_3 :: [[\mathtt{t}, \mathtt{i}_3] \mapsto 1]
\end{aligned}$$

The protocol is initiated on network *Net* when node $\mathtt{l}_1$ executes the process *Snd* to search for information on topic $\mathtt{t}$. Node $\mathtt{l}_1$ then enters a state where it waits for (possibly multiple) answers of the form $(\mathtt{has}, l, x, y)$, meaning that the node at location l sent content y concerning topic x.

Nodes $\mathtt{l}_2$ and $\mathtt{l}_3$ can process `ask`-messages using *Prc*. Upon reception, each of the nodes check whether they have content available in their tuple spaces which match topic x. If so, they broadcast a `has`-message containing this content. In order to make sure that the `ask`-message is propagated across the whole of the network, they also rebroadcast this message, and restart process *Prc* to be ready to receive other requests.

Rel is a simple relay process for `has`-messages. Note further that $\mathtt{l}_2$ and $\mathtt{l}_3$ have tuple spaces with contents $\mathtt{i}_2$ and $\mathtt{i}_3$ associated with topic $\mathtt{t}$.

2.2 Operational Semantics

As a prerequisite for defining the operational semantics of bKlaim, we have to give a notion of connectivity between nodes. A *connectivity graph* as in [15,14] is a directed graph G on a subset of the set of locations **Loc**. As usual, $V(G)$ denotes the set of vertices of G and $E(G)$ its set of edges. Given a graph G, we write

$$G(l) = \{l' \ : \ (l, l') \in E(G)\}$$

to denote the *neighborhood* of a location l.

In this way, a connectivity graph G gives a straightforward notion of connectivity to a network N: a node at location l' may receive a message sent by a node at location l if and only if $(l, l') \in E(G)$. Because the graph is directed, both unidirectional and bidirectional links can be expressed. Note that by separating connectivity from process actions (which most readily distinguishes bKlaim from the bπ-calculus [9] for example) we are able to express the behavior of a variety of networks in which the connectivity may change through changes in the environment conditions, which are not expressed by process actions. Wireless networks are one example, where node movements (which should be clearly separated from the actions of their protocol processes) trigger both link failures and the establishment of new links.

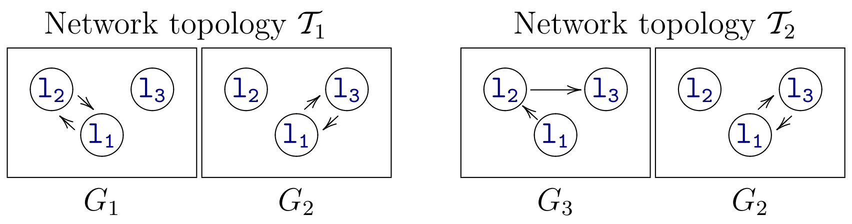
Connectivity graphs provide a snapshot of the network connectivity. In contrast, a *network topology* $\mathcal{T}$ is a set of connectivity graphs which share the same set of vertices. We use network topologies to express the set of possible configurations a particular network may be in.

In order to ensure that a network topology and a network agree, we introduce a well-formedness condition. We first extend the definition of the vertex function V from graphs to networks:

$$V(l :: P) = V(l :: S) = \{l\} \text{ and } V(N_1 \parallel N_2) = V(N_1) \cup V(N_2)$$

We say that the pair $(N, \mathcal{T})$ of a network N and network topology $\mathcal{T}$ is *well-formed* if there is exactly one located tuple space $l :: S$ for each $l \in V(N)$, and if furthermore $\mathcal{T}$ contains only connectivity graphs G with $V(G) = V(N)$.

Example 2.2 Continuing Example 2.1, we define the following network topologies over $V(\mathit{Net})$:


Network topology $\mathcal{T}_1$
Network topology $\mathcal{T}_2$
l_1
l_2
l_3
G_1
G_2
G_3
G_2

We give the operational semantics of bKlaim by a *reduction relation* of the form $\mathcal{T} \vdash M \xrightarrow{\mathbb{l}}_G N$, defined in Table 2, together with a *structural congruence* $M \equiv N$ in Table 3. Derivations of a network N via the reduction relation are with respect to a network topology $\mathcal{T}$ where $(N, \mathcal{T})$ are well-formed; the operational semantics ensures that well-formedness is preserved over all derivations. A derivation is parametrized with a connectivity graph $G \in \mathcal{T}$ to express that the derivation holds under the connectivity expressed by G. We may drop the parameter G and write $\mathcal{T} \vdash M \xrightarrow{\mathbb{l}} N$ when a transition does not depend on the actual choice of $G \in \mathcal{T}$. For the sake of the analysis in §5, transitions are labelled with labels $\mathbb{l}$ of the form (l, ℓ) and $(l, \ell[t])$, to express that the action labelled ℓ has executed at location l, and – in the case of the in^{ℓ}-action only – that the tuple t has been input at location l.

$$\frac{G \in \mathcal{T}}{\mathcal{T} \vdash l :: \mathsf{bcst}^{\ell}(t).P \parallel \prod_{l' \in G(l)} l' :: S_{l'} \xrightarrow{(l,\ell)}_{G} l :: P \parallel \prod_{l' \in G(l)} l' :: S_{l'}(t)^{\uparrow}}$$

$$\frac{}{\mathcal{T} \vdash l :: \mathsf{out}^{\ell}(t).P \parallel l :: S \xrightarrow{(l,\ell)} l :: P \parallel l :: S(t)^{\uparrow}}$$

$$\frac{G \in \mathcal{T}}{\mathcal{T} \vdash l :: \mathsf{b\text{-}eval}^{\ell}(Q).P \xrightarrow{(l,\ell)}_{G} l :: P \parallel \prod_{l' \in G(l)} l' :: Q}$$

$$\frac{S(t) > 0 \qquad match(T,t) = \sigma}{\mathcal{T} \vdash l :: \mathsf{in}^{\ell}(T).P \parallel l :: S \xrightarrow{(l,\ell[t])} l :: P\sigma \parallel l :: S(t)^{\downarrow}}$$

$$\frac{S(t) > 0 \qquad match(T,t) = \sigma}{\mathcal{T} \vdash l :: \mathsf{read}^{\ell}(T).P \parallel l :: S \xrightarrow{(l,\ell)} l :: P\sigma \parallel l :: S}$$

$$\frac{\forall\, t.\ match(T,t) \Rightarrow S(t) = 0}{\mathcal{T} \vdash l :: \mathsf{abs}^{\ell}(T).P \parallel l :: S \xrightarrow{(l,\ell)} l :: P \parallel l :: S}$$

$$\frac{\mathcal{T} \vdash M \xrightarrow{\mathbb{l}} M'}{\mathcal{T} \vdash M \parallel N \xrightarrow{\mathbb{l}} M' \parallel N}$$

$$\frac{N \equiv M \qquad \mathcal{T} \vdash M \xrightarrow{\mathbb{l}} M' \qquad M' \equiv N'}{\mathcal{T} \vdash N \xrightarrow{\mathbb{l}} N'}$$

Table 2: Reduction relation of bKlaim

The bcst-rule puts a tuple t into all tuple spaces in the current neighborhood $G(l)$ of the sender location l, where the current neighborhood is nondeterministically chosen from the network topology $\mathcal{T}$. Rule out puts a tuple t into the private tuple space at location l. Rule b-eval puts a process Q into all nodes in the current neighborhood $G(l)$ of the sender location l, where it can be evaluated. The in-rule inputs (deletes) a tuple contained in the private tuple space S if it matches to the template T, and continues with the process $P\sigma$, where σ captures the bindings introduced by the template matching. Rule read works in the same fashion, but leaves the contents of S unchanged. The rule for abs executes if there is no tuple in the private tuple space S that would match the template T.

The structural congruence provides rules for reordering networks and pro-

$$\begin{aligned}
N_1 \parallel N_2 &\equiv N_2 \parallel N_1 \\
(N_1 \parallel N_2) \parallel N_3 &\equiv N_1 \parallel (N_2 \parallel N_3) \\
l :: P &\equiv l :: P \mid \mathsf{nil} \\
l :: A &\equiv l :: P \text{ if } A \triangleq P \\
l :: P_1 \mid P_2 &\equiv l :: P_1 \parallel l :: P_2
\end{aligned}$$

Table 3: Structural congruence of bKlaim

$$match(v, v) = \epsilon \qquad match(!x, v) = [v/x]$$

$$\frac{match(F, f) = \sigma_1 \qquad match(T, t) = \sigma_2}{match((F, T), (f, t)) = \sigma_1 \circ \sigma_2}$$

Table 4: Template matching

cesses. It is defined as the least equivalence relation satisfying the rules given in Table 3. The first two rules state commutativity and associativity of parallel composition of networks. Furthermore, the empty sum nil is a neutral element for parallel composition of processes, process invocations can be expanded, and parallel composition of processes naturally corresponds to parallel composition of networks.

The semantics for *template matching* is given in Table 4. As in original Klaim, a template matches against a tuple if both have the same number of fields and corresponding fields match; two values match if they are identical while the formal field $!x$ matches against any value. On success, the function *match* returns a substitution associating the variables of the formal fields of the template with the corresponding values in the tuple.

3 Abstract Transition Systems

For a given network, the operational semantics of bKlaim gives rise to a (possibly infinite) transition system where the transitions are determined by the actions performed at each step and the connectivity the network has to abide by when performing a step. For the sake of analysis, we are interested in transforming this transition system into a *finite* one which still preserves the influence of the network topology on the resulting network states. For this purpose this section introduces *abstract transition systems*, and a version of

Action Computation Tree Logic (ACTL) [17] to describe their properties. In order to accommodate the notion of abstraction in the logic, we use a 3-valued interpretation of formulae on abstract transition systems. The use of 3-valued logic for this purpose has first been recognised by [26], and we adapt it to our setting by having a formula evaluate to “unknown” whenever the abstraction prevents us from obtaining a definite result; if a formula evaluates to “true” or “false” however, an embedding theorem ensures that the same formula holds (resp. fails) in its 2-valued interpretation on the concrete transition system.

3.1 Exposed Actions

This section introduces the notion of exposed actions which is used to express abstract network configurations; abstract transition systems, introduced in the following section, will then describe transitions between such abstract configurations, which are related to transitions between concrete networks.

An *exposed action* is an action (or tuple) that *may* participate in the next interaction. In general, a process may contain many, even infinitely many, occurrences of the same action (all identified by the same label) and it may be that several of them are ready to participate in the next interaction.

To capture this, we define an *extended multiset* M as an element of:

$$\mathfrak{M} = \mathbf{Loc} \times (\mathbf{Lab} \cup \mathbf{Val}^*) \to \mathbb{N} \cup \{\infty\}$$

The idea is that $M(l, \ell)$ records the number of occurrences of the label ℓ, and analogously $M(l, t)$ the number of occurrences of the tuple t, at a location l; there may be a finite number, in which case $M(\mathbb{l}) \in \mathbb{N}$, or an infinite number, in which case $M(\mathbb{l}) = \infty$ (where $\mathbb{l}$ ranges over (l, ℓ) or (l, t)). The set $\mathfrak{M}$ is equipped with a partial ordering $\leq_{\mathfrak{M}}$ defined by:

$$M \leq_{\mathfrak{M}} M' \text{ iff } \forall\, \mathbb{l}.\ M(\mathbb{l}) \leq M'(\mathbb{l}) \vee M'(\mathbb{l}) = \infty$$

The domain $(\mathfrak{M}, \leq_{\mathfrak{M}})$ is a complete lattice, and in addition to least and greatest upper bound operators, we shall need operations $+_{\mathfrak{M}}$ and $-_{\mathfrak{M}}$ for addition and subtraction, which can be defined straightforwardly.

To calculate exposed actions, we shall introduce the function

$$\mathcal{E} : \mathbf{Net} \to \mathfrak{M}$$

which takes a network and calculates its extended multiset of exposed actions; this function is defined in Table 5. In the case for tuple spaces, every tuple $t \in S$ is recorded with according multiplicity $S(t)$ at location l. Processes

$$\begin{aligned}
\mathcal{E}[\![N_1 \parallel N_2]\!] &= \mathcal{E}[\![N_1]\!] +_{\mathfrak{M}} \mathcal{E}[\![N_2]\!] \\
\mathcal{E}[\![l :: P]\!] &= \mathcal{E}_l[\![P]\!]\, env_{\mathcal{E}_l} \\
\mathcal{E}[\![l :: S]\!] &= \textstyle\sum_{\mathfrak{M},t} \perp_{\mathfrak{M}}[(l,t) \mapsto S(t)]
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}_l[\![\mathsf{nil}]\!]\, env &= \perp_{\mathfrak{M}} \\
\mathcal{E}_l[\![a^{\ell}.P]\!]\, env &= \perp_{\mathfrak{M}}[(l,\ell) \mapsto 1] \\
\mathcal{E}_l[\![P_1 \mid P_2]\!]\, env &= \mathcal{E}_l[\![P_1]\!]\, env +_{\mathfrak{M}} \mathcal{E}_l[\![P_2]\!]\, env \\
\mathcal{E}_l[\![A]\!]\, env &= env(A)
\end{aligned}$$

$$\begin{aligned}
\text{where } \mathcal{F}_{\mathcal{E}_l}(env) &= [A_1 \mapsto \mathcal{E}_l[\![P_1]\!]\, env, \ldots, A_k \mapsto \mathcal{E}_l[\![P_k]\!]\, env] \\
\text{and } env_{\perp_{\mathfrak{M}}} &= [A_1 \mapsto \perp_{\mathfrak{M}}, \ldots, A_k \mapsto \perp_{\mathfrak{M}}] \\
\text{and } env_{\mathcal{E}_l} &= \textstyle\bigsqcup_{j \geq 0} \mathcal{F}^{j}_{\mathcal{E}_l}(env_{\perp_{\mathfrak{M}}})
\end{aligned}$$

Table 5: Exposed actions for $\mathsf{let}\ A_1 \triangleq P_1; \ldots; A_k \triangleq P_k\ \mathsf{in}\ N_0$

invoke a local function

$$\mathcal{E}_l : \mathbf{Net} \to (\mathbf{PNam} \to \mathfrak{M}) \to \mathfrak{M}$$

which takes as an additional parameter an environment $env \in \mathbf{PNam} \to \mathfrak{M}$ holding the required information for the process names. In the case of actions $a^{\ell}.P$, the label ℓ is recorded at location l with multiplicity 1. For process names we simply consult the environment env. The remaining cases are straightforward.

As shown in Table 5, this defines a family of functionals $\mathcal{F}_{\mathcal{E}_l} : (\mathbf{PNam} \to \mathfrak{M}) \to (\mathbf{PNam} \to \mathfrak{M})$. Since the operations involved in the definition of each $\mathcal{F}_{\mathcal{E}_l}$ are all monotonic, we have monotonic functional on a complete lattice and Tarski's fixed point theorem ensures they it has a fixed point which is denoted $env_{\mathcal{E}_l}$. Since all processes are finite, it follows that all $\mathcal{F}_{\mathcal{E}_l}$ are continuous and hence that the Kleene formulation of the fixed point is permissible.

Example 3.1 Continuing Example 2.1, it is easy to check that

$$\begin{aligned}
\mathcal{E}[\![Net]\!] = [&(\mathtt{l}_1, 1) \mapsto 1, (\mathtt{l}_2, 3) \mapsto 1, (\mathtt{l}_2, 7) \mapsto 1, (\mathtt{l}_3, 3) \mapsto 1, \\
&(\mathtt{l}_3, 7) \mapsto 1, (\mathtt{l}_2, [\mathtt{t}, \mathtt{i}_2]) \mapsto 1, (\mathtt{l}_3, [\mathtt{t}, \mathtt{i}_3]) \mapsto 1].
\end{aligned}$$

We can show that the exposed actions are invariant under the structural congruence and that they correctly capture the actions that may be involved in the first reduction step.

Lemma 3.2 *If $M \equiv N$, then $\mathcal{E}[\![M]\!] = \mathcal{E}[\![N]\!]$. Furthermore, if $\mathcal{T} \vdash M \xrightarrow{\mathbb{l}}_G N$ and $\mathbb{l} = (l, \ell)$, then $\mathbb{l} \in dom(\mathcal{E}[\![M]\!])$; and if $\mathbb{l} = (l, \ell[t])$, then $(l, \ell), (l, t) \in dom(\mathcal{E}[\![M]\!])$.*

Proof. The first result is shown by induction on the rules of structural congruence in Table 3, using the definitions for exposed actions in Table 5. In the rule for recursion unfolding, we have to show that $env_{\mathcal{E}_l}(A) = \mathcal{E}_l[\![P]\!]env_{\mathcal{E}_l}$, which follows from $env_{\mathcal{E}_l} = \bigsqcup_{j \geq 0} \mathcal{F}^j_{\mathcal{E}_l}(env_{\perp_\mathfrak{M}})$ and $\mathcal{F}_{\mathcal{E}_l}(env)(A) = \mathcal{E}_l[\![P]\!]env$. The remaining cases are straightforward.

For the second part, we proceed by induction on the rules of the transition system in Table 2. In the case for input it suffices to show that $(l, \ell), (l, t) \in dom(\mathcal{E}[\![l :: \mathsf{in}^\ell(T).P \parallel l :: S]\!])$ where $S(t) > 0$. We have $(l, \ell) \in dom(\mathcal{E}_l[\![\mathsf{in}^\ell(T).P]\!]env_{\mathcal{E}_l})$, and $(l, u) \in dom(\mathcal{E}[\![l :: S]\!])$ for all $u \in dom(S)$, by the definitions for exposed actions in Table 5. The cases for the other axioms are simpler. For the rule involving the congruence use Lemma 3.2. Then these two cases and the case for the parallel rule can be solved by application of the induction hypothesis. □

3.2 Abstract Transition Systems

An *abstract transition system* is a quadruple $(\mathsf{Q}, q_0, \delta, \mathsf{E})$ with the following components:

- A finite set of states Q where each state q is associated with an extended multiset $\mathsf{E}[q]$ and the idea is that q represents all networks N with $\mathcal{E}[\![N]\!] \leq_\mathfrak{M} \mathsf{E}[q]$;
- an initial state q_0, representing the initial network N_0; and,
- a finite transition relation δ, where $(q_s, (G, \mathbb{l}), q_t) \in \delta$ reflects that starting in state q_s, under connectivity G, the action $\mathbb{l}$ may execute and give rise to q_t.

Definition 3.3 We say that a state denoting the multiset E *represents* a network N, written $N \triangleright E$, iff $\mathcal{E}[\![N]\!] \leq_\mathfrak{M} E$.

Definition 3.4 We say that an abstract transition system $(\mathsf{Q}, q_0, \delta, \mathsf{E})$ *faithfully describes* the evolution of a network N_0 if:

$$M \triangleright \mathsf{E}[q_s] \text{ and } \mathcal{T} \vdash N_0 \to^* M \xrightarrow{\mathbb{l}}_G N,$$

imply that there exists a unique $q_t \in \mathsf{Q}$ such that

$$N \triangleright \mathsf{E}[q_t] \text{ and } (q_s, (G, \mathbb{l}), q_t) \in \delta.$$

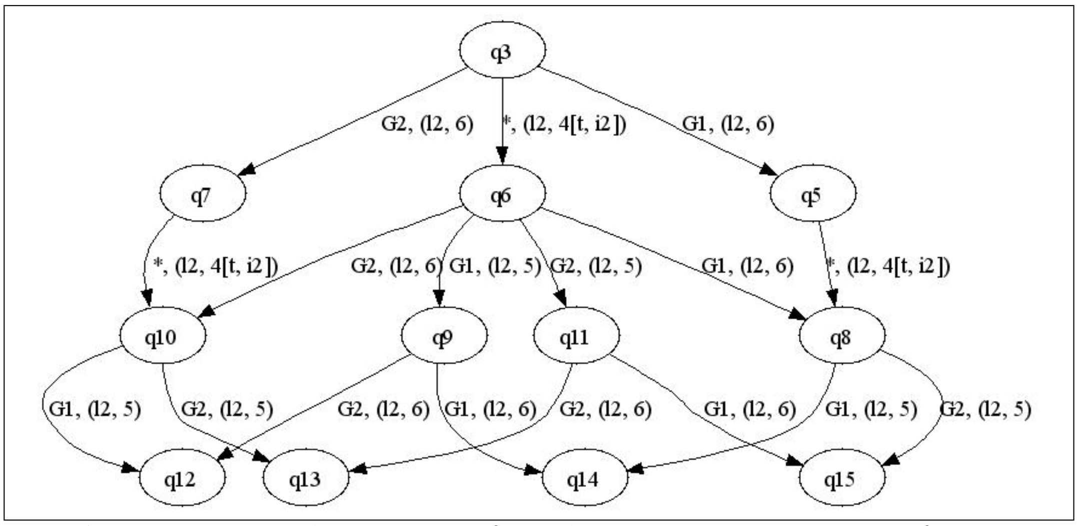


Figure 1: Example 3.5: Part of an abstract transition system for *Net*

In §5 we shall show how to construct an abstract transition system that faithfully describes the evolution of a given network N.

Example 3.5 For the network $(Net, \mathcal{T}_1)$ of Example 2.1, the static analysis of §5 generates an abstract transition system with 27 states and 46 transitions; Figure 1 depicts a part of this transition system. We look at one of its transitions in detail, namely $(q_3, (*, (\mathtt{l}_2, 4[\mathtt{t}, \mathtt{i}_2])), q_6) \in \delta$; the star $*$ stands for any connectivity graph from $\mathcal{T}_1$, as label 4 denotes a (local) input action which thus does not depend on connectivity. For the states q_3 and q_6 involved in this transition, it holds that

$$\begin{aligned} dom(\mathsf{E}[q_3]) &= \{(\mathtt{l}_1, 2), (\mathtt{l}_2, 4), (\mathtt{l}_2, 6), (\mathtt{l}_2, 7), (\mathtt{l}_3, 3), (\mathtt{l}_3, 7), (\mathtt{l}_2, [\mathtt{t}, \mathtt{i}_2]), (\mathtt{l}_3, [\mathtt{t}, \mathtt{i}_3])\} \\ dom(\mathsf{E}[q_6]) &= \{(\mathtt{l}_1, 2), (\mathtt{l}_2, 5), (\mathtt{l}_2, 6), (\mathtt{l}_2, 7), (\mathtt{l}_3, 3), (\mathtt{l}_3, 7), (\mathtt{l}_3, [\mathtt{t}, \mathtt{i}_3])\} \end{aligned}$$

and therefore state q_3 represents a network of the form

$$\mathtt{l}_1 :: \mathsf{in}^2(\ldots).Rec(\mathtt{t}) \parallel \mathtt{l}_2 :: (\mathsf{in}^4(\ldots).\mathsf{bcst}^5(\ldots)\ldots \mid \mathsf{bcst}^6(\ldots).Prc(\mathtt{l}_2)) \parallel \mathtt{l}_2 :: [(\mathtt{t}, \mathtt{i}_2) \mapsto 1] \parallel \ldots$$

and after a transition with action $(\mathtt{l}_2, 4[\mathtt{t}, \mathtt{i}_2])$, we end up in state q_6 that represents

$$\mathtt{l}_1 :: \mathsf{in}^2(\ldots).Rec(\mathtt{t}) \parallel \mathtt{l}_2 :: (\mathsf{bcst}^5(\ldots).\ldots \mid \mathsf{bcst}^6(\ldots).Prc(\mathtt{l}_2)) \parallel \mathtt{l}_2 :: [(\mathtt{t}, \mathtt{i}_2) \mapsto 0] \parallel \ldots.$$

3.3 Interpretation of ACTL Properties

In order to express properties about a network, we propose to use a model checking approach which allows us to describe properties in some temporal logic. We are using a variant of Action Computation Tree Logic (ACTL) [17], which allows us (in contrast to other branching time logics) to utilise the labels $(G, \mathbb{1})$ on the edges of an abstract transition system to constrain the set of paths we are interested in; in this way we may for example determine which properties hold if only node movements specified by a subset $\mathcal{T}' \subseteq \mathcal{T}$ of the original topology are considered. The syntax is defined by the following grammar describing *path formulae* ϕ and *state formulae* γ:

$$\begin{aligned} \phi &::= \mathtt{tt} \mid \mathbb{l} \mid \neg\phi \mid \phi \wedge \phi \mid \exists\gamma \mid \forall\gamma \\ \gamma &::= \mathbf{X}_\Omega\, \phi \mid \phi\, \mathbf{U}_\Omega\, \phi \end{aligned}$$

Here, $\mathbb{l}$ denotes (l, ℓ) or (l, t), $\exists$ and $\forall$ are path quantifiers, Ω is a set of transition labels $(G, \mathbb{1})$ and will be used to constrain the paths a formula is evaluated on, and $\mathbf{X}_\Omega$ and $\mathbf{U}_\Omega$ are *next* and *until* operators, respectively. We shall give two interpretations of this logic; the first relates to the concrete semantics of §2.

We define two judgements $N \vDash \phi$ and $\Pi \vDash \gamma$ for satisfaction of ϕ by a network N, and γ by a path Π. A path Π is of the form $(N_0, (G_0, \mathbb{1}_0), N_1, (G_1, \mathbb{1}_1), \ldots)$ where $\Pi(i) \xrightarrow{\mathbb{1}_i}_{G_i} \Pi(i+1)$ for $i \geq 0$ (we write $\Pi(i)$ for N_i, and $\Pi[i]$ for $(G_i, \mathbb{1}_i)$). The judgements are displayed in Table 6.

$N \vDash \mathtt{tt}$		
$N \vDash \mathbb{l}$	iff	$\mathbb{l} \in \mathcal{E}[\![N]\!]$
$N \vDash \neg\phi$	iff	$N \nvDash \phi$
$N \vDash \phi_1 \wedge \phi_2$	iff	$N \vDash \phi_1 \wedge N \vDash \phi_2$
$N \vDash \exists\gamma$	iff	there exists a path Π such that $\Pi(0) = N$ and $\Pi \vDash \gamma$
$N \vDash \forall\gamma$	iff	$\Pi \vDash \gamma$ holds for all paths Π with $\Pi(0) = N$
$\Pi \vDash \mathbf{X}_\Omega\, \phi$	iff	$\Pi(1) \vDash \phi$ and $\Pi[0] \in \Omega$
$\Pi \vDash \phi_1\, \mathbf{U}_\Omega\, \phi_2$	iff	there exists $k \geq 0$ such that $\Pi(k) \vDash \phi_2$ and
		for all $0 \leq i < k : \Pi(i) \vDash \phi_1$ and $\Pi[i] \in \Omega$

Table 6: Satisfaction relation for networks

Thus the semantics of formulae closely resembles that of ACTL, with the exception that for the novel clause $\mathbb{l}$ to evaluate to satisfy network N, $\mathbb{l}$ must be exposed in N.

Clearly, we cannot directly establish satisfaction of a formula on a network because the related transition system might be infinite. We therefore propose to check formulae on the basis of abstract transition systems, and formally relate the results obtained to the concrete network evolution.

The important question is how to represent the nature of the abstraction. A natural way to model the uncertainty of whether an abstract edge is present in the concrete transition system is to use a 3-valued logic. Here the classical set of truth values $\{0, 1\}$ is extended with a value 1/2 for expressing the uncertainty; 0 and 1 are called *definite* truth values, and 1/2 an *indefinite* truth value. Several choices of 3-valued logics exist and we choose here to use Kleene's strongest regular 3-valued logic [11]; this is in line with the developments of [5,26]. Formulae defined over the abstraction may make use of all three truth values, but unlike e.g. [26,19], the abstraction itself will only make use of the value 0 and 1/2.

A simple way to define conjunction (resp. disjunction) in this logic is as the minimum (resp. maximum) of its arguments, under the order $0 < 1/2 < 1$. We write *min* and *max* for these functions, and extend them to sets in the obvious way, with $\mathit{min}\,\emptyset = 1$ and $\mathit{max}\,\emptyset = 0$. Negation $\neg^3$ maps 0 to 1, 1 to 0, and 1/2 to 1/2. Other operations can be lifted from the classical setting to the 3-valued setting using the method of [20].

Let $L(q, \mathbb{l}) = 0$ if $\mathbb{l} \notin \mathsf{E}[q]$, and 1/2 otherwise. Furthermore, let $D_\Omega(G, \mathbb{1}) = 0$ if $(G, \mathbb{1}) \notin \Omega$, and 1/2 otherwise. A path π is of the form $(q_0, (G_0, \mathbb{1}_0), q_1, (G_1, \mathbb{1}_1), \dots)$ where $(\pi(i), \pi[i], \pi(i+1)) \in \delta$ for $i \geq 0$. The satisfaction relations $[q \models^3 \phi]$ and $[\pi \models^3 \gamma]$ for states q and paths π are defined in Table 7.

$$\begin{aligned}
[q \models^3 \mathtt{tt}] &= 1\\
[q \models^3 \mathbb{l}] &= L(q, \mathbb{l})\\
[q \models^3 \neg\phi] &= \neg^3([q \models^3 \phi])\\
[q \models^3 \phi_1 \wedge \phi_2] &= \mathit{min}([q \models^3 \phi_1], [q \models^3 \phi_2])\\
[q \models^3 \exists\gamma] &= \mathit{max}\,\{[\pi \models^3 \gamma] \;:\; \pi(0) = q\}\\
[q \models^3 \forall\gamma] &= \mathit{max}\,\{\mathit{min}\,\{[\pi \models^3 \gamma] \;:\; \pi(0) = q\}, 1/2\}\\
[\pi \models^3 \mathbf{X}_\Omega\, \phi] &= \mathit{min}([\pi(1) \models^3 \phi], D_\Omega(\pi[0]))\\
[\pi \models^3 \phi_1\, \mathbf{U}_\Omega\, \phi_2] &= \mathit{max}\,\{[\pi \models^3 \phi_1\, \mathbf{U}^k_\Omega\, \phi_2] \;:\; k \geq 0\}\\
[\pi \models^3 \phi_1\, \mathbf{U}^k_\Omega\, \phi_2] &= \mathit{min}(\mathit{min}(\{[\pi(k) \models^3 \phi_2]\} \cup \{[\pi(i) \models^3 \phi_1] \;:\; i < k\}),\\
&\qquad\qquad \mathit{min}\,\{D_\Omega(\pi[i]) \;:\; i < k\})
\end{aligned}$$

Table 7: Satisfaction relation for states

Recall that our abstract transition systems constitute an *overapproximation* of the concrete transition relations, and that we therefore expect to be able to decide *universal* properties only. In the case of the $\exists$ path quantifier, we therefore evaluate to a definite value only if there *does not* exist a path such that a property γ holds, and for $\forall$ only if for all paths γ indeed holds. This is expressed by the following definitions:

$$[q \models^3 \exists\gamma] = \mathit{min}\,\{\mathit{max}\,\{[\pi \models^3 \gamma] \;:\; \pi(0) = q\}, 1/2\}$$
$$[q \models^3 \forall\gamma] = \mathit{max}\,\{\mathit{min}\,\{[\pi \models^3 \gamma] \;:\; \pi(0) = q\}, 1/2\}$$

However, it turns out that we can do better in the case for $\exists$, which leads to a simplification of this case, and the asymmetry in Table 7. The following lemma enables us to do this:

Lemma 3.6 *If* $[\pi \models^3 \gamma] = 1$ *then* $[\pi' \models^3 \gamma] = 1$ *for all* π' *with* $\pi'(0) = \pi(0)$.

Proof. It is easy to see that $[\pi \models^3 \mathbf{X}_\Omega\ \phi]$ cannot evaluate to 1 because $D_\Omega(\pi[0])$ never evaluates to 1. If $[\pi \models^3 \phi_1\ \mathbf{U}_\Omega\ \phi_2] = 1$, then $[\pi \models^3 \phi_1\ \mathbf{U}^k_\Omega\ \phi_2]$ must evaluate to 1 for some k, and this is only possible for $k = 0$ where $\{D_\Omega(\pi[i]) : i < k\}$ is the empty set and $[\pi(0) \models^3 \phi_2] = 1$. Hence for all π' with $\pi'(0) = \pi(0)$ we have $[\pi'(0) \models^3 \phi_2] = 1$ and thus $[\pi' \models^3 \phi_1\ \mathbf{U}^k_\Omega\ \phi_2] = 1$ for $k = 0$ which establishes the claim. □

We therefore know that if a path formula γ holds on one path starting from a state q, then it holds in all such paths. Therefore the property would hold as well in any concrete transition path, and we do not have to evaluate to 1/2 in this case.

In [5], a stronger result in the $\forall$-case can be achieved as well, because there the Egli-Milner powerdomain ordering (over- *and* underapproximation) is assumed to produce the abstract transition system, where we use the Hoare ordering (overapproximation). Our approach is justified by the fact that we are actually providing a practical method (see §6) which can *generate* our abstractions for concrete systems. Using two transition relations as in [12,8] – one representing the Hoare ordering, the other the Smyth ordering (underapproximation) – could likewise be used to strengthen the result for the $\forall$-case.

We lift the notion of representation $\rhd$ from states to paths by defining:

$$\Pi \blacktriangleright \mathsf{E}[\pi] \text{ iff } \forall\, i \geq 0.\ \Pi(i) \rhd \mathsf{E}[\pi(i)] \wedge \Pi[i] = \pi[i]$$

Furthermore, we define an *information order* $\sqsubseteq$ on truth values by $1/2 \sqsubseteq 0$, $1/2 \sqsubseteq 1$, and $x \sqsubseteq x$ for all $x \in \{0, 1/2, 1\}$. Using this, we can formulate an embedding theorem, which allows us to relate the 2- and 3-valued interpretations of ACTL:

Theorem 3.7 *Suppose* $(\mathsf{Q}, q_0, \delta, \mathsf{E})$ *faithfully describes the evolution of network* N_0*, and* $\mathcal{T} \vdash N_0 \to^* N$*. Then:*

(1) *If* $N \rhd \mathsf{E}[q]$ *then* $[q \models^3 \phi] \sqsubseteq [N \models \phi]$.
(2) *If* $\Pi \blacktriangleright \mathsf{E}[\pi]$ *then* $[\pi \models^3 \gamma] \sqsubseteq [\Pi \models \gamma]$.

Proof. By induction on the length of the formula, simultaneously over both parts of the theorem. By the definition of the information ordering, there is nothing to show for $[q \models^3 \phi] = 1/2$ or $[\pi \models^3 \gamma] = 1/2$, we therefore distinguish only the cases where these judgements evaluate to definite truth values.

Case $\phi = \mathtt{tt}$. Clearly, $[q \models^3 \mathtt{tt}] \sqsubseteq [N \models \mathtt{tt}]$.

Case $\phi = \mathbb{l}$. If $[q \models^3 \mathbb{l}] = 0$ then $\mathbb{l} \notin \mathsf{E}[q]$. Because $N \rhd \mathsf{E}[q]$, we also have $\mathbb{l} \notin \mathcal{E}[\![N]\!]$, and hence $N \not\models \mathbb{l}$. Furthermore, $[q \models^3 \mathbb{l}]$ can never evaluate to 1. Thus, $[q \models^3 \mathbb{l}] \sqsubseteq [N \models \mathbb{l}]$.

Case $\phi = \neg\phi$. If $[q \models^3 \neg\phi] = 0$ then $[q \models^3 \phi] = 1$ because of the semantics of $\neg^3$. We can apply the induction hypothesis to have $q \models \phi$ which is equivalent to $q \not\models \neg\phi$. The case $[q \models^3 \neg\phi] = 1$ is analogous.

Case $\phi = \phi_1 \wedge \phi_2$. If $[q \models^3 \phi_1 \wedge \phi_2] = 0$ then $[q \models^3 \phi_1] = 0$ or $[q \models^3 \phi_2] = 0$. By the induction hypothesis we thus have $N \not\models \phi_1$ or $N \not\models \phi_2$, hence $N \not\models \phi_1 \wedge \phi_2$.

If $[q \models^3 \phi_1 \wedge \phi_2] = 1$ then $[q \models^3 \phi_1] = 1$ and $[q \models^3 \phi_2] = 1$. By the induction hypothesis we thus have $N \models \phi_1$ and $N \models \phi_2$, hence $N \models \phi_1 \wedge \phi_2$.

Case $\phi = \exists\gamma$. If $[q \models^3 \exists\gamma] = 0$ then $[\pi \models^3 \gamma] = 0$ for all π with $\pi(0) = q$. Suppose there exists a path Π such that $\Pi(0) = N$ and $\Pi \models \gamma$. Then this path would be faithfully described by the abstract transition system, and hence $\Pi \blacktriangleright \mathsf{E}[\pi']$ would hold for some π' with $\pi'(0) = q$. By the induction hypothesis we have $[\pi' \models^3 \gamma] \sqsubseteq [\Pi \models \gamma]$, where we know that $[\pi' \models^3 \gamma] = 0$. Hence $\Pi \not\models \gamma$, a contradiction. Therefore we have $\Pi \not\models \gamma$ for all Π with $\Pi(0) = N$, which establishes $N \not\models \exists\gamma$.

If $[q \models^3 \exists\gamma] = 1$ then there exists a path π with $\pi(0) = q$ such that $[\pi \models^3 \gamma] = 1$. Because of Lemma 3.6, $[\pi \models^3 \gamma] = 1$ holds for all π with $\pi(0) = q$. Suppose for all paths Π with $\Pi(0) = N$ we have $\Pi \not\models \gamma$. Then all these Π would be faithfully described by the abstract transition system, and hence $\Pi \blacktriangleright \mathsf{E}[\pi']$ would hold for some π' with $\pi'(0) = q$. By the induction hypothesis we have $[\pi' \models^3 \gamma] \sqsubseteq [\Pi \models \gamma]$, where we know that $[\pi' \models^3 \gamma] = 1$. Hence $\Pi \models \gamma$, a contradiction. Therefore we have that there exists a Π with $\Pi(0) = N$ such that $\Pi \models \gamma$, which establishes $N \models \forall\gamma$.

Case $\phi = \forall\gamma$. Because of Definition 7, $[q \models^3 \forall\gamma]$ can never evaluate to 0.

If $[q \models^3 \forall\gamma] = 1$ then, by Definition 7, $[\pi \models^3 \gamma] = 1$ for all π with $\pi(0) = q$. Suppose there exists a path Π such that $\Pi(0) = N$ and $\Pi \not\models \gamma$. Then this path would be faithfully described by the abstract transition system, and hence $\Pi \blacktriangleright \mathsf{E}[\pi']$ would hold for some π' with $\pi'(0) = q$. By the induction hypothesis we have $[\pi' \models^3 \gamma] \sqsubseteq [\Pi \models \gamma]$, where we know that $[\pi' \models^3 \gamma] = 1$. Hence $\Pi \models \gamma$, a contradiction. Therefore we have $\Pi \models \gamma$ for all Π with $\Pi(0) = N$, which establishes $N \models \exists\gamma$.

Case $\gamma = \mathbf{X}_\Omega\ \phi$. If $[\pi \models^3 \mathbf{X}_\Omega\ \phi] = 0$ then $[\pi(1) \models^3 \phi] = 0$ or $D_\Omega(\pi[0]) = 0$. Because $\Pi \blacktriangleright \mathsf{E}[\pi]$ gives $\pi[0] = \Pi[0]$ and because of the definition of D_Ω, whenever $D_\Omega(\pi[0]) = 0$ also $\Pi[0] \notin \Omega$. If $[\pi(1) \models^3 \phi] = 0$, then $\Pi(1) \not\models \phi$ by the induction hypothesis. In both cases we can conclude $\Pi \not\models \mathbf{X}_\Omega\ \phi$ as required.

Because $min([\pi(1) \models^3 \phi], D_\Omega(\pi[0]))$ depends on $D_\Omega(\pi[0])$ which cannot evaluate to 1, $[\pi \models^3 \mathbf{X}_\Omega\ \phi]$ cannot evaluate to 1 either.

Case $\phi = \phi_1\ \mathbf{U}_\Omega\ \phi_2$. If $[\pi \models^3 \phi_1\ \mathbf{U}_\Omega\ \phi_2] = 0$ then $[\pi \models^3 \phi_1\ \mathbf{U}^k_\Omega\ \phi_2] = 0$ for all $k \geq 0$. Hence either $min(\{[\pi(k) \models^3 \phi_2]\} \cup \{[\pi(i) \models^3 \phi_1]\ :\ i < k\}) = 0$ or $min\,\{D_\Omega(\pi(i), \pi(i+1))\ :\ i < k\} = 0$. If the latter holds, then there exists an $i < k$ such that $D_\Omega(\pi[i]) = 0$; because $\Pi \blacktriangleright \mathsf{E}[\pi]$ gives $\pi[i] = \Pi[i]$ for all $i \geq 0$ and because of the definition of D_Ω, whenever $D_\Omega(\pi[i]) = 0$ also $\Pi[i] \notin \Omega$. If the former holds, either $\{[\pi(k) \models^3 \phi_2]\} = 0$ or $\{[\pi(i) \models^3 \phi_1]\ :\ i < k\} = 0$, and thus by the induction hypothesis either $\Pi(k) \not\models \phi_2$ or $\Pi(i) \not\models \phi_1$ for some $i < k$; hence $\Pi \not\models \phi_1\ \mathbf{U}_\Omega\ \phi_2$.

If $[\pi \models^3 \phi_1\ \mathbf{U}_\Omega\ \phi_2] = 1$, then $[\pi \models^3 \phi_1\ \mathbf{U}^k_\Omega\ \phi_2]$ must evaluate to 1 for some k, and this is only possible for $k = 0$ where $\{D_\Omega(\pi[i])\ :\ i < k\}$ is the empty set and $[\pi(0) \models^3 \phi_2] = 1$. By the induction hypothesis we thus have $\Pi(0) \models \phi_2$ and hence $\Pi \models \phi_1\ \mathbf{U}_\Omega\ \phi_2$. □

Example 3.8 For the abstract transition system for $(Net, \mathcal{T}_1)$ of Example 2.1 and 2.2, and an Ω containing all possible transition labels, we have

$$[q_0 \models^3 \neg\exists[\mathtt{tt}\ \mathbf{U}_\Omega\ ((\mathtt{l}_1, [\mathtt{has}, \mathtt{l}_2, \mathtt{t}, \mathtt{i}_2]) \wedge (\mathtt{l}_1, [\mathtt{has}, \mathtt{l}_3, \mathtt{t}, \mathtt{i}_3]))]] = 1$$

while on $(Net, \mathcal{T}_2)$ we get the result 1/2. Using Theorem 3.7, this means that $(Net, \mathcal{T}_1)$ has no evolution such that both $[\mathtt{has}, \mathtt{l}_2, \mathtt{t}, \mathtt{i}_2]$ and $[\mathtt{has}, \mathtt{l}_3, \mathtt{t}, \mathtt{i}_3]$ are exposed tuples at location $\mathtt{l}_1$. In other words, under topology $\mathcal{T}_1$, the node $\mathtt{l}_1$ requesting information on topic $\mathtt{t}$ cannot get replies from both $\mathtt{l}_2$ and $\mathtt{l}_3$. For $(Net, \mathcal{T}_2)$ the analysis states that the abstraction prevents a definite answer.

4 Control Flow Analysis

Control Flow Analyses have been used in order to analyze a variety of process calculi, e.g. [4,3], and we have used it in particular in [15,14] to establish security properties of broadcast networks. In this earlier work we have however abstracted away the dynamics of the system, i.e. the network topology $\mathcal{T}$ was replaced by a single connectivity graph which contains all possible edges, i.e. any edges that might occur in a $G \in \mathcal{T}$. While this is a safe view (as it yields an overapproximation of the messages that may be sent in the network), it prevents the analysis result from exposing the influence of topology changes.

In this paper, our main analysis is based on a Monotone Framework (see §5) and an worklist algorithm (see §6), which enables us to construct abstract transition systems as described in §3. However, the variable bindings for a network have to be supplied to the Monotone Framework. Therefore we still define a Control Flow Analysis for bKlaim in order to deal with this aspect of the analysis, the results of which become a parameter in the Monotone Framework.

The Control Flow Analysis uses the following abstract domains:

$\hat{\rho} : \mathbf{Var} \to \wp(\mathbf{Val})$	Variable environment
$\hat{S} : \mathbf{Loc} \to \wp(\mathbf{Val}^*)$	Store environment

The *variable environment* $\hat{\rho}$ records for every variable occurring in a network N the set of values it may be bound to during the evolution of N. The variable environment can be extended to tuples by defining:

$$\hat{\rho}[\![v]\!] = \{v\} \text{ and } \hat{\rho}[\![x]\!] = \hat{\rho}(x) \text{ and } \hat{\rho}[\![f, t]\!] = \hat{\rho}[\![f]\!] \times \hat{\rho}[\![t]\!]$$

The *store environment* $\hat{S}$ records for every location the set of tuples that may reside at the tuple space belonging to that location during the evolution of N.

We define the analysis using the Flow Logic framework [22], that takes a specification oriented approach to determining whether or not a given analysis estimate correctly describes all configurations reachable from a given initial network. The correctness result is given by a subject reduction result, which means that analysis estimates can be “too large”. The next step therefore is to use standard techniques (not covered here, but see e.g. [21]) to turn this specification into a form where “the least” acceptable analysis estimate can be computed in polynomial time.

The flow logic uses three main judgments:

$(\hat{\rho}, \hat{S}) \models^G N$	Judgment for networks
$(\hat{\rho}, \hat{S}) \models^G_l P$	Judgment for processes
$(\hat{\rho}, \hat{S}) \models^G_l a$	Judgment for actions

Note that the judgments for processes and actions are parametrized with the location at which they are executing. Furthermore, the three main judgments are parametrized with a connectivity graph G. In order to achieve an overapproximation of all possible variable bindings that may occur in $(N, \mathcal{T})$, this G must be chosen to contain all possible edges that might arise during compu-

$$\begin{array}{lll}
(\hat{\rho},\hat{S}) \vDash^G N_1 \parallel N_2 & \text{iff} & (\hat{\rho},\hat{S}) \vDash^G N_1 \wedge (\hat{\rho},\hat{S}) \vDash^G N_2 \\
(\hat{\rho},\hat{S}) \vDash^G l :: P & \text{iff} & (\hat{\rho},\hat{S}) \vDash^G_l P \\
(\hat{\rho},\hat{S}) \vDash^G l :: S & \text{iff} & \forall\, u \in dom(S).\ u \in \hat{S}(l) \\
 & & \\
(\hat{\rho},\hat{S}) \vDash^G_l \mathsf{nil} & \text{iff} & \mathit{true} \\
(\hat{\rho},\hat{S}) \vDash^G_l a^{\ell}.P & \text{iff} & (\hat{\rho},\hat{S}) \vDash^G_l a^{\ell} \wedge (\hat{\rho},\hat{S}) \vDash^G_l P \\
(\hat{\rho},\hat{S}) \vDash^G_l P_1 \mid P_2 & \text{iff} & (\hat{\rho},\hat{S}) \vDash^G_l P_1 \wedge (\hat{\rho},\hat{S}) \vDash^G_l P_2 \\
(\hat{\rho},\hat{S}) \vDash^G_l A & \text{iff} & (\hat{\rho},\hat{S}) \vDash^G_l P \text{ where } A \triangleq P \\
 & & \\
(\hat{\rho},\hat{S}) \vDash^G_l \mathsf{bcst}^{\ell}(t) & \text{iff} & \forall\, l' \in G(l).\ \hat{\rho}[\![t]\!] \subseteq \hat{S}(l') \\
(\hat{\rho},\hat{S}) \vDash^G_l \mathsf{out}^{\ell}(t) & \text{iff} & \hat{\rho}[\![t]\!] \subseteq \hat{S}(l) \\
(\hat{\rho},\hat{S}) \vDash^G_l \mathsf{b\text{-}eval}^{\ell}(Q) & \text{iff} & \forall\, l' \in G(l).\ (\hat{\rho},\hat{S}) \vDash^G_{l'} Q \\
(\hat{\rho},\hat{S}) \vDash^G_l \mathsf{in}^{\ell}(T) & \text{iff} & \exists\, \hat{T}.\ \hat{\rho} \vDash_1 T : \hat{S}(l) \triangleright \hat{T} \\
(\hat{\rho},\hat{S}) \vDash^G_l \mathsf{read}^{\ell}(T) & \text{iff} & \exists\, \hat{T}.\ \hat{\rho} \vDash_1 T : \hat{S}(l) \triangleright \hat{T} \\
(\hat{\rho},\hat{S}) \vDash^G_l \mathsf{abs}^{\ell}(T) & \text{iff} & \mathit{true}
\end{array}$$

Table 8: Control Flow Analysis for bKlaim

tation:

$$(\exists\, G' \in \mathcal{T}.\ (m,n) \in E(G')) \text{ iff } (m,n) \in E(G)$$

We write $G = \bigsqcup \mathcal{T}$ for a connectivity graph constructed in this manner, and call it the *abstract connectivity graph* corresponding to $\mathcal{T}$.

The main judgments are defined in Table 8. The judgment for networks proceeds in a syntax-directed manner and is straightforward. Note that in the case for tuple spaces all tuples t which are in the domain of the multiset S (i.e. where $S(t) > 0$, see §2.1) are taken to be in the store environment at location l.

Also the judgment for processes proceeds in a mainly syntax directed manner, except for the need to unfold recursive processes. This does not invalidate our axiomatization, as in general we take a co-inductive rather than inductive interpretation of a Flow Logic [22].

The rule for summation invokes the judgment for actions. In the case for bcst, it is made sure that the estimation for the tuple t according to $\hat{\rho}$ is included in the estimation for all tuple stores in the neighborhood $G(l)$ of location l. For the local out-action, only the estimation for the tuple space at l is affected. For action b-eval, the judgment to evaluate the migrating process Q is invoked at all locations in the neighborhood of l. The two rules for in and read update the variable environment $\hat{\rho}$ with the new possible bindings calculated by an auxiliary judgment for pattern matching $\hat{\rho} \models_1 T : \hat{S}(l) \triangleright \hat{T}$. This auxiliary judgment expresses informally that $\hat{T}$ is a safe estimate to the tuples contained in $\hat{S}(l)$ that match with template T under bindings $\hat{\rho}$ (new bindings can be introduced by the matching); we formally define the judgment below. To achieve safety, the rule for `abs` always holds.

The main judgments use the following auxiliary judgment

$$\hat{\rho} \models_i T : \hat{S}_\circ \triangleright \hat{T}_\bullet \qquad \text{Auxiliary judgment for pattern matching}$$

which is defined in Table 9. This judgment traverses the template in a forward direction (starting at index i that is supposed not to exceed the length of T) and then in a backward direction (stopping at index i). In the forward direction the tuples in $\hat{S}_\circ$ are tested against the relevant component of the template T and only tuples satisfying the requirements are carried forward. In the backward direction the tuples in $\hat{T}_\bullet$ are those that passed all requirements and the values in the relevant component are used for defining the names (of the form $!x$) to be matched in that component.

$\hat{\rho} \models_i \epsilon : \hat{S}_\circ \triangleright \hat{S}_\bullet$	iff	$\{t \in \hat{S}_\circ \ : \ \lvert t\rvert = i-1\} \sqsubseteq \hat{S}_\bullet$
$\hat{\rho} \models_i v, T : \hat{S}_\circ \triangleright \hat{T}_\bullet$	iff	$\hat{\rho} \models_{i+1} T : \hat{S}_\bullet \triangleright \hat{T}_\bullet \wedge \{t \in \hat{S}_\circ \ : \ \mathrm{prj}_i(t) = v\} \sqsubseteq \hat{S}_\bullet$
$\hat{\rho} \models_i x, T : \hat{S}_\circ \triangleright \hat{T}_\bullet$	iff	$\hat{\rho} \models_{i+1} T : \hat{S}_\bullet \triangleright \hat{T}_\bullet \wedge \{t \in \hat{S}_\circ \ : \ \mathrm{prj}_i(t) \in \hat{\rho}(x)\} \sqsubseteq \hat{S}_\bullet$
$\hat{\rho} \models_i !x, T : \hat{S}_\circ \triangleright \hat{T}_\bullet$	iff	$\hat{\rho} \models_{i+1} T : \hat{S}_\bullet \triangleright \hat{T}_\bullet \wedge \hat{S}_\circ \sqsubseteq \hat{S}_\bullet \wedge \mathrm{prj}_i(\hat{T}_\bullet) \sqsubseteq \hat{\rho}(x)$

Table 9: Abstract matching

The correctness of the main judgment $(\hat{\rho}, \hat{S}) \models^G N$ is formulated as a subject reduction result which is proved below. Two auxiliary lemmas are required, the first one stating a property of the judgment for matching.

Lemma 4.1 *Suppose match$(T,t) = \sigma$ and $t \in \hat{S}_\circ$ for a ground tuple t and closed template T. If $\hat{\rho} \models_1 T : \hat{S}_\circ \triangleright \hat{T}_\bullet$, then $t \in \hat{T}_\bullet$ and $\sigma(x) \in \hat{\rho}(x)$ for all $x \in dom(\sigma)$.*

Proof. Let T^i denote the template obtained from T by dropping the first $i-1$ fields (analogously t^i). We prove the following stronger result:

Let $i \leq length(T) + 1$ and suppose match$(T^i, t^i) = \sigma_i$ and $t \in \hat{S}_\circ$. If $\hat{\rho} \models_i T^i : \hat{S}_\circ \triangleright \hat{T}_\bullet$, then $t \in \hat{T}_\bullet$ and $\sigma_i(x) \in \hat{\rho}(x)$ for all $x \in dom(\sigma_i)$.

We proceed by structural induction on T^i.

Case $T^i = \epsilon$**.** This means that T and t have length $i-1$. Hence $t \in \hat{T}_\bullet$ by the rule for ϵ in Table 9. Furthermore, σ_i has an empty domain and there is nothing to show for the second part.

Case $T^i = v, T^{i+1}$**.** Thus $\mathrm{prj}_i(t) = v$ and therefore $t \in \hat{S}_\bullet$ on the right-hand side of the rule for values in Table 9, and also $\mathit{match}(T^{i+1}, t^{i+1}) = \sigma_i \ (= \sigma_{i+1})$ by the definition of matching. Hence we can apply the induction hypothesis to $\hat{\rho} \vDash_{i+1} T^{i+1} : \hat{S}_\bullet \triangleright \hat{T}_\bullet$ and have $t \in \hat{T}_\bullet$ and $\forall\ x \in \mathit{dom}(\sigma_i).\ \sigma_i(x) \in \hat{\rho}(x)$ as required.

Case $T^i = x, T^{i+1}$**.** Does not apply because T is closed.

Case $T^i = !x, T^{i+1}$**.** We have $t \in \hat{S}_\bullet$ by the rule for formal fields in Table 9, where $\hat{S}_\circ \sqsubseteq \hat{S}_\bullet$. Also $\mathit{match}(T^{i+1}, t^{i+1}) = \sigma_{i+1}$ where $[\mathrm{prj}_i(t)/x] \circ \sigma_{i+1} = \sigma_i$ by the definition of matching. Hence we can apply the induction hypothesis to $\hat{\rho} \vDash_{i+1} T^{i+1} : \hat{S}_\bullet \triangleright \hat{T}_\bullet$ and have $t \in \hat{T}_\bullet$ and $\forall\ x \in \mathit{dom}(\sigma_{i+1}).\ \sigma_{i+1}(x) \in \hat{\rho}(x)$. Because $\mathrm{prj}_i(\hat{T}_\bullet) \sqsubseteq \hat{\rho}(x)$ and $t \in \hat{T}_\bullet$ we have $\mathrm{prj}_i(t) \in \hat{\rho}(x)$, and thus $\forall\ x \in \mathit{dom}(\sigma_i).\ \sigma_i(x) \in \hat{\rho}(x)$. □

The next lemma says that the judgments for processes, actions, and matching are invariant under a substitution σ, if the variable environment $\hat{\rho}$ expresses all bindings of σ.

Lemma 4.2 (Substitution) *Suppose $\sigma(x) \in \hat{\rho}(x)$ for all $x \in \mathit{dom}(\sigma)$. Then the following implications hold:*

(1) *If $\hat{\rho} \vDash_1 T : \hat{S}_\circ \triangleright \hat{T}_\bullet$ then $\hat{\rho} \vDash_1 T\sigma : \hat{S}_\circ \triangleright \hat{T}_\bullet$.*
(2) *If $(\hat{\rho}, \hat{S}) \vDash^G_l a^\ell$ then $(\hat{\rho}, \hat{S}) \vDash^G_l a^\ell\sigma$.*
(3) *If $(\hat{\rho}, \hat{S}) \vDash^G_l P$ then $(\hat{\rho}, \hat{S}) \vDash^G_l P\sigma$.*

Proof. **Ad** (1)**.** By structural induction on T. The only interesting case (where something is actually substituted) is $T = x, U$. Then we have $\hat{\rho} \vDash_{i+1} U : \hat{S}_\bullet \triangleright \hat{T}_\bullet$ and $\{t \in \hat{S}_\circ\ :\ \mathrm{prj}_i(t) \in \hat{\rho}(x)\} \sqsubseteq \hat{S}_\bullet$ by the rule for variables in Table 9. Because $\sigma(x) \in \hat{\rho}(x)$ and $\hat{\rho}(\sigma(x)) = \{\sigma(x)\}$, we have $v \in \hat{\rho}(\sigma(x)) \Rightarrow v \in \hat{\rho}(x)$ for all values v. Therefore

$$\{t \in \hat{S}_\circ\ :\ \mathrm{prj}_i(t) \in \hat{\rho}(\sigma(x))\} \sqsubseteq \{t \in \hat{S}_\circ\ :\ \mathrm{prj}_i(t) \in \hat{\rho}(x)\} \sqsubseteq \hat{S}_\bullet$$

By the induction hypothesis we obtain $\hat{\rho} \vDash_{i+1} U\sigma : \hat{S}_\bullet \triangleright \hat{T}_\bullet$, and thus we can use the rule for variables again to prove the case.

Ad (2)**.** We proceed by structural induction on a^ℓ. For all cases the respective rules in Table 8 are used. The cases bcst and out follow from the fact $\hat{\rho}[\![t\sigma]\!] \subseteq \hat{\rho}[\![t]\!]$. Case b-eval is proved by applying the induction hypothesis. Cases in and read follow from part (1) of the lemma. There is nothing to show for abs.

Ad (3)**.** By a straightforward induction on the rules used to obtain $(\hat{\rho}, \hat{S}) \vDash^G_l P$, where part (2) of the lemma is used in the case for actions. □

The main theorem states the invariance of the analysis estimate for networks under the rules of the structural congruence and the reduction relation.

Theorem 4.3 (Subject Reduction)

(1) *If* $M \equiv N$ *then* $(\hat\rho, \hat S) \models^{\bigsqcup \mathcal{T}} M \iff (\hat\rho, \hat S) \models^{\bigsqcup \mathcal{T}} N$.
(2) *If* $\mathcal{T} \vdash M \xrightarrow{\mathbb{1}}_G N$ *and* $(\hat\rho, \hat S) \models^{\bigsqcup \mathcal{T}} M$, *then* $(\hat\rho, \hat S) \models^{\bigsqcup \mathcal{T}} N$.

Proof. **Ad** (1)**.** By a straightforward induction on the rules of the structural congruence in Table 3.

Ad (2)**.** By induction on the inference of $\mathcal{T} \vdash M \xrightarrow{\mathbb{1}}_G N$. For abbreviation purposes, let $\hat G = \bigsqcup \mathcal{T}$.

Case bcst**.** Then we know that

$$\begin{aligned} M &= l :: \mathsf{bcst}^{\ell}(t).P \parallel \textstyle\prod_{l' \in G(l)} l' :: S_{l'} \\ N &= l :: P \parallel \textstyle\prod_{l' \in G(l)} l' :: S_{l'}(t)^{\uparrow}. \end{aligned}$$

We have $(\hat\rho, \hat S) \models^{\hat G} l :: \mathsf{bcst}^{\ell}(t).P \parallel \prod_{l' \in G(l)} l' :: S_{l'}$ by assumption. Using the rules of Table 8 for parallel composition, nodes, tuple spaces, and bcst, we have

$$(\forall\, l' \in \hat G(l).\ \hat\rho[\![t]\!] \subseteq \hat S(l')) \wedge (\hat\rho, \hat S) \models^{\hat G}_{l} P \wedge \textstyle\bigwedge_{l' \in G(l)} (\forall\, u \in dom(S).\ u \in \hat S(l')).$$

We know $t \in \hat\rho[\![t]\!]$ for all ground t, and $G(l) \subseteq \hat G(l)$, hence

$$(\hat\rho, \hat S) \models^{\hat G}_{l} P \wedge \textstyle\bigwedge_{l' \in G(l)} (\forall\, u \in dom(S_{l'}(t)^{\uparrow}).\ u \in \hat S(l'))$$

which is equivalent to $(\hat\rho, \hat S) \models^{\hat G} l :: P \parallel \prod_{l' \in G(l)} l' :: S_{l'}(t)^{\uparrow}$.

Case out**.** Analogous to case bcst (simpler).

Case b-eval**.** Then we know that

$$\begin{aligned} M &= l :: \mathsf{b\text{-}eval}^{\ell}(Q).P \\ N &= l :: P \parallel \textstyle\prod_{l' \in G(l)} l' :: Q. \end{aligned}$$

We have $(\hat\rho, \hat S) \models^{\hat G} l :: \mathsf{b\text{-}eval}^{\ell}(Q).P$ by assumption which corresponds to

$$\textstyle\bigwedge_{l' \in G(l)} (\hat\rho, \hat S) \models^{\hat G}_{l'} Q \wedge (\hat\rho, \hat S) \models^{\hat G}_{l} P$$

and this implies $(\hat\rho, \hat S) \models^{\hat G} l :: P \parallel \prod_{l' \in G(l)} l' :: Q$.

Case in. Then we know that $S(t) > 0$, $\mathit{match}(T,t) = \sigma$ and

$$\begin{aligned} M &= l :: \mathsf{in}^{\ell}(T).P \parallel l :: S \\ N &= l :: P\sigma \parallel l :: S(t)^{\downarrow}. \end{aligned}$$

We have $(\hat{\rho}, \hat{S}) \models^{\hat{G}} l :: \mathsf{in}^{\ell}(T).P \parallel l :: S$ by assumption. Using the rules of Table 8 for parallel composition, nodes, tuple spaces, and in, we have

$$\hat{\rho} \models_1 T : \hat{S}(l) \triangleright \hat{T}_{\bullet} \wedge (\hat{\rho}, \hat{S}) \models_l^{\hat{G}} P \wedge (\forall\, u \in \mathit{dom}(S).\ u \in \hat{S}(l))$$

Because $S(t) > 0$, we know that $t \in \hat{S}(l)$. Together with $\mathit{match}(T,t) = \sigma$, this allows us to apply Lemma 4.1 on $\hat{\rho} \models_1 T : \hat{S}(l) \triangleright \hat{T}_{\bullet}$, thus obtaining $\sigma(x) \in \hat{\rho}(x)$ for all $x \in \mathit{dom}(\sigma)$. Hence we can apply Lemma 4.2 (3) to have $(\hat{\rho}, \hat{S}) \models_l^{\hat{G}} P\sigma$. Note that $\mathit{dom}(S(t)^{\downarrow}) \subseteq \mathit{dom}(S)$, and thus:

$$(\hat{\rho}, \hat{S}) \models_l^{\hat{G}} P\sigma \wedge (\forall\, u \in \mathit{dom}(S(t)^{\downarrow}).\ u \in \hat{S}(l))$$

which is equivalent to $(\hat{\rho}, \hat{S}) \models^{\hat{G}} l :: P\sigma \parallel l :: S(t)^{\downarrow}$

Case read. Analogous to case in.

Case abs. Nothing to show.

Case Parallel Composition**.** By a straightforward application of the induction hypothesis.

Case Structural Congruence**.** By a straightforward application of the induction hypothesis, and use of part (1) of the theorem. □

5 Monotone Framework

The abstraction function $\mathcal{E}$ only gives us the information of interest for the initial process. Once an action has participated in an interaction, some new actions may become exposed and some may cease to be exposed. We shall now present auxiliary functions $\mathcal{G}_{\hat{\rho}}^{G}$ and $\mathcal{K}$ allowing us to approximate how the information evolves during the execution of the process. These correspond to a classical approach in Data Flow Analysis, namely the *gen* and *kill* components of Monotone Frameworks, which have been generalised similarly [24] in the setting of CCS. The relevant information will be an element of:

$$\mathfrak{T} = \mathbf{Loc} \times (\mathbf{Lab} \cup \mathbf{Val}^{*}) \rightarrow \mathfrak{M}$$

As for exposed actions it is not sufficient to use sets: there may be more than one occurrence of an action that is either generated or killed by another action.

The ordering $\leq_{\mathfrak{T}}$ is defined as the pointwise extension of $\leq_{\mathfrak{M}}$:

$$T_1 \leq_{\mathfrak{T}} T_2 \text{ iff } \forall\, \mathbb{l}.\ T_1(\mathbb{l}) \leq_{\mathfrak{M}} T_2(\mathbb{l})$$

5.1 Generated Actions

To calculate generated actions, we shall introduce the function

$$\mathcal{G}^G_{\hat\rho} : \mathbf{Net} \to \mathfrak{T}$$

which takes a network N and computes an *over*-approximation of which actions might be generated in N; this function is defined in Table 10. Note that the function carries two more parameters, namely a connectivity graph G and a variable environment $\hat\rho$. The connectivity graph G is needed because it determines at which locations tuples are generated when using broadcast. Likewise, we need $\hat\rho$ to correctly determine which tuples might be output; it is therefore assumed in the following that $(\hat\rho, \hat S) \models^{\bigsqcup \mathcal{T}} N_0$ holds.

$$
\begin{aligned}
\mathcal{G}^G_{\hat\rho}[\![N_1 \parallel N_2]\!] &= \mathcal{G}^G_{\hat\rho}[\![N_1]\!] \sqcup_{\mathfrak{T}} \mathcal{G}^G_{\hat\rho}[\![N_2]\!] \\
\mathcal{G}^G_{\hat\rho}[\![l :: P]\!] &= \mathcal{G}^G_{\hat\rho,l}[\![P]\!]\, env_{\mathcal{G}^G_l} \\
\mathcal{G}^G_{\hat\rho}[\![l :: S]\!] &= \bot_{\mathfrak{T}} \\[1ex]
\mathcal{G}^G_{\hat\rho,l}[\![\mathsf{nil}]\!]\, env &= \bot_{\mathfrak{T}} \\
\mathcal{G}^G_{\hat\rho,l}[\![a^\ell.P]\!]\, env &= \tilde{\mathcal{G}}^G_{\hat\rho,l}[\![a^\ell.P]\!]\, env \sqcup_{\mathfrak{T}} \mathcal{G}^G_{\hat\rho,l}[\![P]\!]\, env \\
\mathcal{G}^G_{\hat\rho,l}[\![P_1 \mid P_2]\!]\, env &= \mathcal{G}^G_{\hat\rho,l}[\![P_1]\!]\, env \sqcup_{\mathfrak{T}} \mathcal{G}^G_{\hat\rho,l}[\![P_2]\!]\, env \\
\mathcal{G}^G_{\hat\rho,l}[\![A]\!]\, env &= env(A) \\[1ex]
\tilde{\mathcal{G}}^G_{\hat\rho,l}[\![\mathsf{bcst}^\ell(t).P]\!]\, env &= \bot_{\mathfrak{T}}[(l,\ell) \mapsto \mathcal{E}_l[\![P]\!]\, env_{\mathcal{E}_l} +_{\mathfrak{M}} (\textstyle\sum_{\mathfrak{M}, l' \in G(l), u \in \hat\rho[\![t]\!]} \bot_{\mathfrak{M}}[(l',u) \mapsto 1])] \\
\tilde{\mathcal{G}}^G_{\hat\rho,l}[\![\mathsf{out}^\ell(t).P]\!]\, env &= \bot_{\mathfrak{T}}[(l,\ell) \mapsto \mathcal{E}_l[\![P]\!]\, env_{\mathcal{E}_l} +_{\mathfrak{M}} (\textstyle\sum_{\mathfrak{M}, u \in \hat\rho[\![t]\!]} \bot_{\mathfrak{M}}[(l,u) \mapsto 1])] \\
\tilde{\mathcal{G}}^G_{\hat\rho,l}[\![\mathsf{b\text{-}eval}^\ell(Q).P]\!]\, env &= \bot_{\mathfrak{T}}[(l,\ell) \mapsto \mathcal{E}_l[\![P]\!]\, env_{\mathcal{E}_l} +_{\mathfrak{M}} \textstyle\sum_{\mathfrak{M}, l' \in G(l)} \mathcal{E}_{l'}[\![Q]\!]\, env_{\mathcal{E}_{l'}}] \\
\tilde{\mathcal{G}}^G_{\hat\rho,l}[\![a^\ell.P]\!]\, env &= \bot_{\mathfrak{T}}[(l,\ell) \mapsto \mathcal{E}_l[\![P]\!]\, env_{\mathcal{E}_l}],\ \text{ for } a^\ell = \mathsf{in}^\ell(T), \mathsf{read}^\ell(T), \mathsf{abs}^\ell(T) \\[1ex]
\text{where } \mathcal{F}_{\mathcal{G}^G_l}(env) &= [A_1 \mapsto \mathcal{G}^G_{\hat\rho,l}[\![P_1]\!]\, env, \ldots, A_k \mapsto \mathcal{G}^G_{\hat\rho,l}[\![P_k]\!]\, env] \\
\text{and } env_{\bot_{\mathfrak{T}}} &= [A_1 \mapsto \bot_{\mathfrak{T}}, \ldots, A_k \mapsto \bot_{\mathfrak{T}}] \\
\text{and } env_{\mathcal{G}^G_l} &= \textstyle\bigsqcup_{j \geq 0} \mathcal{F}^j_{\mathcal{G}^G_l}(env_{\bot_{\mathfrak{T}}})
\end{aligned}
$$

Table 10: Generated actions for $\mathsf{let}\ A_1 \triangleq P_1; \ldots; A_k \triangleq P_k\ \mathsf{in}\ N_0$

As in the case for exposed actions, we need a local function

$$\mathcal{G}^G_{\hat{\rho},l} : \mathbf{Net} \to (\mathbf{PNam} \to \mathfrak{T}) \to \mathfrak{T}$$

which is invoked by processes. Furthermore note that there is an auxiliary function $\tilde{\mathcal{G}}^G_{\hat{\rho},l}$ for actions. All actions $a^\ell.P$ then expose $\mathcal{E}_l[\![P]\!]env_{\mathcal{E}_l}$, i.e. the actions of the continuation process. Furthermore, bcst exposes the tuples $u \in \hat{\rho}[\![t]\!]$ for all locations $l' \in G(l)$ in the neighborhood of the sending process. Simpler, the action out exposes all $u \in \hat{\rho}[\![t]\!]$ only at location l. The migration $\mathsf{b\text{-}eval}$ exposes $\mathcal{E}_{l'}[\![Q]\!]env_{\mathcal{E}_{l'}}$, the exposed actions of the migrating process Q, at all neighboring locations.

Analogously to the argumentation used for exposed actions, this defines a family of functionals $\mathcal{F}_{\mathcal{G}^G_l} : (\mathbf{PNam} \to \mathfrak{T}) \to (\mathbf{PNam} \to \mathfrak{T})$, and each $\mathcal{F}_{\mathcal{G}^G_l}$ has a fixed point which can be written in the Kleene formulation.

We can show that the the information computed by $\mathcal{G}^G_{\hat{\rho}}$ is invariant under the structural congruence and that it potentially *decreases* with the reduction of the process:

Lemma 5.1 *Suppose* $(\hat{\rho}, \hat{S}) \models^{\bigsqcup \mathcal{T}} M$ *holds. If* $M \equiv N$*, then* $\mathcal{G}^G_{\hat{\rho}}[\![M]\!] = \mathcal{G}^G_{\hat{\rho}}[\![N]\!]$*. Furthermore, if* $\mathcal{T} \vdash M \xrightarrow{\mathbb{l}}_G N$*, then* $\mathcal{G}^G_{\hat{\rho}}[\![N]\!] \leq_{\mathfrak{T}} \mathcal{G}^G_{\hat{\rho}}[\![M]\!]$*.*

Proof. The first result is shown by induction on the rules of structural congruence in Table 3, using the definitions for generated actions in Table 10. In the rule for recursion unfolding, we have to show that $env_{\mathcal{G}^G_l}(A) = \mathcal{G}^G_{\hat{\rho},l}[\![P]\!]env_{\mathcal{G}^G_l}$, which follows from $env_{\mathcal{G}^G_l} = \bigsqcup_{j\geq 0} \mathcal{F}^j_{\mathcal{G}^G_l}(env_{\perp_{\mathfrak{T}}})$ and $\mathcal{F}_{\mathcal{G}^G_l}(env)(A) = \mathcal{G}^G_{\hat{\rho},l}[\![P]\!]env$. The remaining cases are straightforward.

For the second part we proceed by induction on the inference of $\mathcal{T} \vdash M \xrightarrow{\mathbb{l}}_G N$ as defined in Table 2. The inequality $\mathcal{G}^G_{\hat{\rho}}[\![N]\!] \leq_{\mathfrak{T}} \mathcal{G}^G_{\hat{\rho}}[\![M]\!]$ is straightforward to show for all outputting actions and for abs. For actions in and read we require the auxiliary lemma $\mathcal{G}^G_{\hat{\rho},l}[\![P\sigma]\!]env \leq_{\mathfrak{T}} \mathcal{G}^G_{\hat{\rho},l}[\![P]\!]env$, which is straightforwardly proved by induction, using the assumption $(\hat{\rho}, \hat{S}) \models^{\bigsqcup \mathcal{T}} M$ and Lemma 4.1. The rules for parallel composition and structural congruence are proved by applications of the induction hypothesis, where in the latter case we also have to use the first part of the lemma. □

Note that the function $\mathcal{G}^G_{\hat{\rho}}$ is defined on pairs of locations and actions only. It can be trivially extended to the general label $\mathbb{l} = (l, \ell[t])$ which is used in the reduction rule for in by defining:

$$\mathcal{G}^G_{\hat{\rho}}[\![N]\!](l, \ell[t]) = \mathcal{G}^G_{\hat{\rho}}[\![N]\!](l, \ell)$$

To calculate killed actions, we shall introduce the function

$$\mathcal{K} : \mathbf{Net} \to \mathfrak{T}$$

which takes a network N and computes an *under*-approximation of which actions might be killed in N; this function is defined in Table 11. When actions $a^{\ell}.P$ execute at location l, it is clear that one occurrence (l, ℓ) can be killed.

Analogously to the argumentation used for exposed actions, this defines a family of functionals $\mathcal{F}_{\mathcal{K}_l} : (\mathbf{PNam} \to \mathfrak{T}) \to (\mathbf{PNam} \to \mathfrak{T})$, and each $\mathcal{F}_{\mathcal{K}_l}$ has a fixed point which can be written in the Kleene formulation.

$$\begin{aligned}
\mathcal{K}[\![N_1 \parallel N_2]\!] &= \mathcal{K}[\![N_1]\!] \sqcap_{\mathfrak{T}} \mathcal{K}[\![N_2]\!] \\
\mathcal{K}[\![l :: P]\!] &= \mathcal{K}_l[\![P]\!]\, env_{\mathcal{K}_l} \\
\mathcal{K}[\![l :: S]\!] &= \top_{\mathfrak{T}} \\
\\
\mathcal{K}_l[\![\mathsf{nil}]\!]\, env &= \top_{\mathfrak{T}} \\
\mathcal{K}_l[\![a^{\ell}.P]\!]\, env &= \top_{\mathfrak{T}}[(l,\ell) \mapsto \bot_{\mathfrak{M}}[(l,\ell) \mapsto 1]] \sqcap_{\mathfrak{T}} \mathcal{K}_l[\![P]\!]\, env \\
\mathcal{K}_l[\![P_1 \mid P_2]\!]\, env &= \mathcal{K}_l[\![P_1]\!]\, env \sqcap_{\mathfrak{T}} \mathcal{K}_l[\![P_2]\!]\, env \\
\mathcal{K}_l[\![A]\!]\, env &= env(A) \\
\\
\text{where } \mathcal{F}_{\mathcal{K}_l}(env) &= [A_1 \mapsto \mathcal{K}_l[\![P_1]\!]\, env, \ldots, A_k \mapsto \mathcal{K}_l[\![P_k]\!]\, env] \\
\text{and } env_{\top_{\mathfrak{T}}} &= [A_1 \mapsto \top_{\mathfrak{T}}, \ldots, A_k \mapsto \top_{\mathfrak{T}}] \\
\text{and } env_{\mathcal{K}_l} &= \textstyle\bigsqcap_{j \geq 0} \mathcal{F}^{j}_{\mathcal{K}_l}(env_{\top_{\mathfrak{T}}})
\end{aligned}$$

Table 11: Killed actions for $\mathsf{let}\ A_1 \triangleq P_1; \ldots; A_k \triangleq P_k\ \mathsf{in}\ N_0$

We can show that the the information computed by $\mathcal{K}$ is invariant under the structural congruence and that it potentially *increases* with the reduction of the process:

Lemma 5.2 *If $M \equiv N$, then $\mathcal{K}[\![M]\!] = \mathcal{K}[\![N]\!]$. Furthermore, if $\mathcal{T} \vdash M \xrightarrow{\mathbb{I}}_G N$ then $\mathcal{K}[\![M]\!] \leq_{\mathfrak{T}} \mathcal{K}[\![N]\!]$.*

Proof. The first result is shown by induction on the rules of structural congruence in Table 3, using the definitions for killed actions in Table 11. In the rule for recursion unfolding, we have to show that $env_{\mathcal{K}_l}(A) = \mathcal{K}_l[\![P]\!]\, env_{\mathcal{K}_l}$, which follows from $env_{\mathcal{K}_l} = \bigsqcap_{j \geq 0} \mathcal{F}^{j}_{\mathcal{K}_l}(env_{\top_{\mathfrak{T}}})$ and $\mathcal{F}_{\mathcal{K}_l}(env)(A) = \mathcal{K}_l[\![P]\!]\, env$. The remaining cases are straightforward.

For the second part we proceed by induction on the inference of $\mathcal{T} \vdash M \xrightarrow{\mathbb{l}}_G N$ as defined in Table 2. The inequality $\mathcal{K}[\![M]\!] \leq_{\mathfrak{T}} \mathcal{K}[\![N]\!]$ is straightforward to show for all outputting actions and abs. For actions in and read we require the result $\mathcal{K}_l[\![P\sigma]\!]\, env = \mathcal{K}_l[\![P]\!]\, env$, which is immediate since $\mathcal{K}$ does not take tuples into account. The rules for parallel composition and structural congruence are proved by applications of the induction hypothesis, where in the latter case we also have to use the first part of the lemma. □

Analogously to the case of $\mathcal{G}^G_{\hat{\rho}}$ we can define an extension of $\mathcal{K}$ by

$$\mathcal{K}[\![N]\!](l, \ell[t]) = \mathcal{K}[\![N]\!](l, \ell) +_{\mathfrak{M}} \perp_{\mathfrak{M}} [(l, t) \mapsto 1]$$

i.e. an input action additionally removes a tuple t from the tuple space.

5.3 Transfer Function

In this setting the transfer function from classical Monotone Frameworks takes the following form

$$(E -_{\mathfrak{M}} \mathcal{K}[\![M]\!](\mathbb{l})) +_{\mathfrak{M}} \mathcal{G}^G_{\hat{\rho}}[\![M]\!](\mathbb{l}),$$

which corresponds to a transition $\mathcal{T} \vdash M \xrightarrow{\mathbb{l}}_G N$.

Correctness. The following result states that the transfer function defined above provides safe approximations to the exposed actions of the resulting network:

Theorem 5.3 *Suppose* $(\hat{\rho}, \hat{S}) \models^{\bigsqcup \mathcal{T}} M$ *holds for a network* M *and a network topology* $\mathcal{T}$. *If* $\mathcal{T} \vdash M \xrightarrow{\mathbb{l}}_G N$, *then*

$$\mathcal{E}[\![N]\!] \leq_{\mathfrak{M}} (\mathcal{E}[\![M]\!] -_{\mathfrak{M}} \mathcal{K}[\![M]\!](\mathbb{l})) +_{\mathfrak{M}} \mathcal{G}^G_{\hat{\rho}}[\![M]\!](\mathbb{l}).$$

Proof. By induction of the inference of $\mathcal{T} \vdash M \xrightarrow{\mathbb{l}}_G N$.

Case bcst. Then we know that $\mathbb{l} = (l, \ell)$ and

$$\begin{aligned} M &= l :: \mathsf{bcst}^{\ell}(t).P \parallel \textstyle\prod_{l' \in G(l)} l' :: S_{l'} \\ N &= l :: P \parallel \textstyle\prod_{l' \in G(l)} l' :: S_{l'}(t)^{\uparrow}, \end{aligned}$$

and we can calculate:

$$\begin{array}{ll}
\mathcal{E}[\![M]\!] & = \bot_{\mathfrak{M}}[(l,\ell)\mapsto 1] +_{\mathfrak{M}} \sum_{\mathfrak{M},l'\in G(l)} \sum_{\mathfrak{M},u} \bot_{\mathfrak{M}}[(l',u)\mapsto S_{l'}(u)] \\
\mathcal{K}[\![M]\!](l,\ell) & = \bot_{\mathfrak{M}}[(l,\ell)\mapsto 1] \sqcap_{\mathfrak{M}} (\mathcal{K}_l[\![P]\!]\,env_{\mathcal{K}_l})(l,\ell) \\
\mathcal{G}^G_{\hat\rho}[\![M]\!](l,\ell) & = (\mathcal{E}_l[\![P]\!]\,env_{\mathcal{E}_l} +_{\mathfrak{M}} (\sum_{\mathfrak{M},l'\in G(l)} \sum_{\mathfrak{M},u\in\hat\rho[\![t]\!]} \bot_{\mathfrak{M}}[(l',u)\mapsto 1])) \\
& \quad \sqcup_{\mathfrak{M}} (\mathcal{G}^G_{\hat\rho,l}[\![P]\!]\,env_{\mathcal{G}^G_l})(l,\ell) \\
\mathcal{E}[\![N]\!] & = \mathcal{E}_l[\![P]\!]\,env_{\mathcal{E}_l} +_{\mathfrak{M}} \sum_{\mathfrak{M},l'\in G(l)} \sum_{\mathfrak{M},u\neq t} \bot_{\mathfrak{M}}[(l',u)\mapsto S_{l'}(u)] \\
& \quad +_{\mathfrak{M}} \sum_{\mathfrak{M},l'\in G(l)} \bot_{\mathfrak{M}}[(l',t)\mapsto S_{l'}(t)^{\uparrow}]
\end{array}$$

Since $t\in\hat\rho[\![t]\!]$ (a consequence of Theorem 4.3), we have

$$\begin{array}{ll}
& (\mathcal{E}[\![M]\!] -_{\mathfrak{M}} \mathcal{K}[\![M]\!](l,\ell)) +_{\mathfrak{M}} \mathcal{G}^G_{\hat\rho}[\![M]\!](l,\ell) \\
\geq_{\mathfrak{M}} & \sum_{\mathfrak{M},l'\in G(l)} \sum_{\mathfrak{M},u} \bot_{\mathfrak{M}}[(l',u)\mapsto S_{l'}(u)] \\
& +_{\mathfrak{M}}\mathcal{E}_l[\![P]\!]\,env_{\mathcal{E}_l} +_{\mathfrak{M}} (\sum_{\mathfrak{M},l'\in G(l)} \sum_{\mathfrak{M},u\in\hat\rho[\![t]\!]} \bot_{\mathfrak{M}}[(l',u)\mapsto 1]) \\
\geq_{\mathfrak{M}} & \mathcal{E}[\![N]\!].
\end{array}$$

Case out. Analogous to case bcst (simpler).

Case b-eval. Then we know that $\mathbb{l}=(l,\ell)$ and

$$\begin{array}{ll}
M & = l :: \mathsf{b\text{-}eval}^{\ell}(Q).P \\
N & = l :: P \parallel \prod_{l'\in G(l)} l' :: Q,
\end{array}$$

and it suffices to calculate

$$\begin{array}{ll}
\mathcal{G}^G_{\hat\rho}[\![M]\!](l,\ell) & = (\mathcal{E}_l[\![P]\!]\,env_{\mathcal{E}_l} +_{\mathfrak{M}} \sum_{\mathfrak{M},l'\in G(l)} \mathcal{E}_{l'}[\![Q]\!]\,env_{\mathcal{E}_{l'}}) \sqcup_{\mathfrak{M}} (\mathcal{G}^G_{\hat\rho,l}[\![P]\!]\,env_{\mathcal{G}^G_l})(l,\ell) \\
\mathcal{E}[\![N]\!] & = \mathcal{E}_l[\![P]\!]\,env_{\mathcal{E}_l} +_{\mathfrak{M}} \sum_{\mathfrak{M},l'\in G(l)} \mathcal{E}_{l'}[\![Q]\!]\,env_{\mathcal{E}_{l'}}
\end{array}$$

to have $\mathcal{E}[\![N]\!] \leq_{\mathfrak{M}} \mathcal{G}^G_{\hat\rho}[\![M]\!](l,\ell)$.

Case in. Then we know $\mathbb{l}=(l,\ell[t])$, $S(t)>0$, $match(T,t)=\sigma$, and

$$\begin{array}{ll}
M & = l :: \mathsf{in}^{\ell}(T).P \parallel l :: S \\
N & = l :: P\sigma \parallel l :: S(t)^{\downarrow},
\end{array}$$

and we can calculate:

$$\begin{array}{ll}
\mathcal{E}[\![M]\!] & = \bot_{\mathfrak{M}}[(l,\ell)\mapsto 1] +_{\mathfrak{M}} \sum_{\mathfrak{M},u} \bot_{\mathfrak{M}}[(l',u)\mapsto S_{l'}(u)] \\
\mathcal{K}[\![M]\!](l,\ell[t]) & = (\bot_{\mathfrak{M}}[(l,\ell)\mapsto 1] \sqcap_{\mathfrak{M}} (\mathcal{K}_l[\![P]\!]\,env_{\mathcal{K}_l})(l,\ell)) +_{\mathfrak{M}} \bot_{\mathfrak{M}}\,[(l,t)\mapsto 1] \\
\mathcal{G}^G_{\hat\rho}[\![M]\!](l,\ell[t]) & = \mathcal{E}_l[\![P]\!]\,env_{\mathcal{E}_l} \sqcup_{\mathfrak{M}} (\mathcal{G}^G_{\hat\rho,l}[\![P]\!]\,env_{\mathcal{G}^G_l})(l,\ell) \\
\mathcal{E}[\![N]\!] & = \mathcal{E}_l[\![P\sigma]\!]\,env_{\mathcal{E}_l} +_{\mathfrak{M}} \sum_{\mathfrak{M},u\neq t} \bot_{\mathfrak{M}}[(l',u)\mapsto S_{l'}(u)] \\
& \quad +_{\mathfrak{M}} \bot_{\mathfrak{M}}[(l',t)\mapsto S_{l'}(t)^{\downarrow}]
\end{array}$$

Because $\mathcal{E}_l[\![P\sigma]\!]env_{\mathcal{E}_l} = \mathcal{E}_l[\![P]\!]env_{\mathcal{E}_l}$ (exposed actions do not depend on tuples), it remains to check that

$$\begin{aligned} &\textstyle\sum_{\mathfrak{M},u\neq t} \perp_{\mathfrak{M}}[(l',u) \mapsto S_{l'}(u)] +_{\mathfrak{M}} \perp_{\mathfrak{M}}[(l',t) \mapsto S_{l'}(t)^{\downarrow}] \\ \leq_{\mathfrak{M}} &\textstyle(\sum_{\mathfrak{M},u} \perp_{\mathfrak{M}}[(l',u) \mapsto S_{l'}(u)]) -_{\mathfrak{M}} \perp_{\mathfrak{M}} [(l,t) \mapsto 1] \end{aligned}$$

This holds because the count of t in $S_{l'}$ is decreased on both sides of the inequation.

Case read and abs**.** Analogous to in (simpler).

Case Parallel Composition**.** Then we know that

$$\begin{aligned} M &= M_0 \parallel N_0 \\ N &= M_0' \parallel N_0, \end{aligned}$$

and can calculate:

$$\begin{aligned} \mathcal{E}[\![M]\!] &= \mathcal{E}[\![M_0]\!] +_{\mathfrak{M}} \mathcal{E}[\![N_0]\!] \\ \mathcal{K}[\![M]\!](\mathbb{1}) &= \mathcal{K}[\![M_0]\!](\mathbb{1}) \sqcap_{\mathfrak{M}} \mathcal{K}[\![N_0]\!](\mathbb{1}) \\ \mathcal{G}_{\hat\rho}^{G}[\![M]\!](\mathbb{1}) &= \mathcal{G}_{\hat\rho}^{G}[\![M_0]\!](\mathbb{1}) \sqcup_{\mathfrak{M}} \mathcal{G}_{\hat\rho}^{G}[\![N_0]\!](\mathbb{1}) \\ \mathcal{E}[\![N]\!] &= \mathcal{E}[\![M_0']\!] +_{\mathfrak{M}} \mathcal{E}[\![N_0]\!] \end{aligned}$$

By the induction hypothesis we have $\mathcal{E}[\![M_0']\!] \leq_{\mathfrak{M}} (\mathcal{E}[\![M_0]\!] -_{\mathfrak{M}} \mathcal{K}[\![M_0]\!](\mathbb{1})) +_{\mathfrak{M}} \mathcal{G}_{\hat\rho}^{G}[\![M_0]\!](\mathbb{1})$. Therefore

$$\begin{aligned} \mathcal{E}[\![N]\!] &\leq_{\mathfrak{M}} (\mathcal{E}[\![M_0]\!] -_{\mathfrak{M}} \mathcal{K}[\![M_0]\!](\mathbb{1})) +_{\mathfrak{M}} \mathcal{G}_{\hat\rho}^{G}[\![M_0]\!](\mathbb{1}) +_{\mathfrak{M}} \mathcal{E}[\![N_0]\!] \\ &\leq_{\mathfrak{M}} ((\mathcal{E}[\![M_0]\!] +_{\mathfrak{M}} \mathcal{E}[\![N_0]\!]) -_{\mathfrak{M}} \mathcal{K}[\![M_0]\!](\mathbb{1}) +_{\mathfrak{M}} \mathcal{G}_{\hat\rho}^{G}[\![M_0]\!](\mathbb{1}) \end{aligned}$$

Case Structural Congruence**.** This case is proved by a straightforward application of the induction hypothesis, where we note that $\mathcal{E}$, $\mathcal{K}$, and $\mathcal{G}_{\hat\rho}^{G}$ are all invariant under the structural congruence as stated in Lemmas 3.2, 5.1, and 5.2. □

As a corollary of the previous theorem, we can state that for an initial network let $A_1 \triangleq P_1; \ldots; A_k \triangleq P_k$ in N_0 the transfer function

$$\mathsf{transfer}_{(G,\mathbb{1}),\hat\rho}(E) = E -_{\mathfrak{M}} \mathcal{K}[\![N_0]\!](\mathbb{1}) +_{\mathfrak{M}} \mathcal{G}_{\hat\rho}^{G}[\![N_0]\!](\mathbb{1})$$

which requires the functions for killed and generated actions only to be computed once for N_0, gives a safe description of the transfer of exposed actions under execution of an action with semantic label $\mathbb{1}$.

Corollary 5.4 *Consider the network* let $A_1 \triangleq P_1; \ldots; A_k \triangleq P_k$ in N_0 *and* *suppose* $(\hat{\rho}, \hat{S}) \models^{\bigsqcup \mathcal{T}} N_0$. *If* $\mathcal{T} \vdash N_0 \rightarrow^* M \xrightarrow{\mathbb{l}}_G N$ *then*

$$\mathcal{E}[\![N]\!] \leq_{\mathfrak{M}} \mathsf{transfer}_{(G,\mathbb{l}),\hat{\rho}}(\mathcal{E}[\![M]\!])$$

Proof. A direct consequence of Theorem 5.3, Lemma 5.1, Lemma 5.2, and Theorem 4.3. □

Example 5.5 Continuing Example 3.5, we can calculate that

$$\begin{aligned} \mathcal{K}[\![Net]\!](\mathtt{l}_2, 4[\mathtt{t}, \mathtt{i}_2]) &= [(\mathtt{l}_2, 4) \mapsto 1, (\mathtt{l}_2, [\mathtt{t}, \mathtt{i}_2]) \mapsto 1] \\ \mathcal{G}^G_{\hat{\rho}}[\![Net]\!](\mathtt{l}_2, 4[\mathtt{t}, \mathtt{i}_2]) &= [(\mathtt{l}_2, 5) \mapsto 1] \end{aligned}$$

and hence that $\mathsf{E}[q_6] = (\mathsf{E}[q_3] -_{\mathfrak{M}} \mathcal{K}[\![Net]\!](\mathtt{l}_2, 4[\mathtt{t}, \mathtt{i}_2])) +_{\mathfrak{M}} \mathcal{G}^G_{\hat{\rho}}[\![Net]\!](\mathtt{l}_2, 4[\mathtt{t}, \mathtt{i}_2])$.

6 Worklist Algorithm

We are interested in analyzing networks of the form

$$\mathsf{let}\ A_1 \triangleq P_1; \ldots; A_k \triangleq P_k\ \mathsf{in}\ N_0$$

where we assume in the following that $(\hat{\rho}, \hat{S}) \models^{\bigsqcup \mathcal{T}} N_0$ holds. We shall now construct a abstract transition system which faithfully describes the evolution of N_0 as specified in §3.2.

The key algorithm is a *worklist algorithm*, which is described in §6.1. It starts out from the initial state and constructs the automaton by adding more and more states and transitions. The algorithm makes use of several auxiliary operations which are further developed in the subsequent sections:

- Given a state q_s representing some exposed actions, we need to select those labels $\mathbb{l}$ that represent actions that may interact in the next step; this is done using the procedure $\mathsf{enabled}_{(\hat{\rho},\hat{S})}$ described in §6.3.
- Once the labels $\mathbb{l}$ have been selected, we can use the function $\mathsf{transfer}_{(G,\mathbb{l}),\hat{\rho}}$, which has been introduced already in §5.3.
- Finally, an appropriate target state q_t has to be constructed and the transition $(q_s, (G, \mathbb{l}), q_t)$ must be recorded; this is done using the procedure update developed in §6.2.

6.1 Worklist Algorithm

The main data structures of the algorithm are:

- A set Q of the current states.

- A worklist W being a subset of Q and containing those states that have yet to be processed.
- A set δ of the current transitions.

```
1 $\mathsf{Q} := \{q_0\};\ \ \mathsf{E}[q_0] := \mathcal{E}[\![N_0]\!];\ \ \mathsf{W} := \{q_0\};\ \ \delta := \emptyset;$
2 while $W \neq \emptyset$ do
3     select $q_s$ from $\mathsf{W}$;  $\mathsf{W} := \mathsf{W} \backslash \{q_s\}$;
4     foreach $G \in \mathcal{T}$ do
5         foreach $\mathbb{I} \in \mathsf{enabled}_{(\hat{\rho},\hat{S})}(\mathsf{E}[q_s])$ do
6             let $E = \mathsf{transfer}_{(G,\mathbb{I}),\hat{\rho}}(\mathsf{E}[q_s])$ in $\mathsf{update}(q_s, (G,\mathbb{I}), E)$
```

Table 12: Worklist algorithm

The algorithm has the form displayed in Table 12. The initializations are performed in line 1. Both the set of states and the worklist are initialized to contain the initial state q_0, and q_0 is associated with the set of the exposed actions of the initial network $\mathcal{E}[\![N_0]\!]$. The transition relation δ is empty.

The algorithm then loops over the contents of the worklist W by selecting a q_s it contains, and removing it from W (line 3). For each $G \in \mathcal{T}$ and enabled action $\mathbb{I} \in \mathsf{enabled}_{(\hat{\rho},\hat{S})}(\mathsf{E}[q_s])$ (lines 4–5) the procedure $\mathsf{transfer}_{(G,\mathbb{I}),\hat{\rho}}(\mathsf{E}[q_s])$ returns an extended multiset describing the denotation of the target state. The last step is to update the automaton to include the new transition step, and this is done in line 6 by the procedure call $\mathsf{update}(q_s, (G,\mathbb{I}), E)$.

6.2 *Procedure* update

The procedure $\mathsf{update}(q_s, (G,\mathbb{I}), E)$ is specified in Table 13. Recall that E is the extended multiset describing the denotation of the target state (to be called q_t) to which there should be a transition labeled $(G,\mathbb{I})$ that emerges from q_s.

First, the state q_t is determined in lines 2–6, where it is checked whether one of the existing states can be used and if not, a new state is created and the corresponding entry in E is set to $\bot_{\mathfrak{M}}$. To determine the reusability of a state, we make use of a *granularity function* H, which is described below.

In lines 7–8 it is checked whether the description $\mathsf{E}[q_t]$ includes the required information E, and if not it is updated and the state is put on the worklist for future processing. The widening operator $\nabla_{\mathfrak{M}}$ makes sure to combine the old

```
1 procedure update($q_s, (G, \mathbb{1}), E$)
2   if there exists $q \in \mathsf{Q}$ with $H(\mathsf{E}[q]) = H(E)$ then
3       $q_t := q$
4   else
5       select $q_t$ from outside $\mathsf{Q}$;
6       $\mathsf{Q} := \mathsf{Q} \cup \{q_t\};\quad \mathsf{E}[q_t] := \perp_{\mathfrak{M}};$
7   if $\neg(E \leq_{\mathfrak{M}} \mathsf{E}[q_t])$ then
8       $\mathsf{E}[q_t] := \mathsf{E}[q_t] \nabla_{\mathfrak{M}} E;\quad \mathsf{W} := \mathsf{W} \cup \{q_t\};$
9   $\delta := \delta \backslash \{(q_s, (G, \mathbb{1}), q) \ : \ q \in Q\} \cup \{(q_s, (G, \mathbb{1}), q_t)\};$
```

Table 13: Procedure update

and the new extended multiset in such a way that termination of the overall algorithm is ensured. We shall return to the definition of $\nabla_{\mathfrak{M}}$ below.

The transition relation is updated in line 9. The triple $(q_s, (G, \mathbb{1}), q_t)$ is added, but we also have to remove any previous transitions from q_s with label $(G, \mathbb{1})$, as its target states may be no longer correct. As a consequence, the automaton may contain unreachable parts, which can be removed at this point or after the completion of the algorithm by a simple clean-up procedure for Q, W, and δ.

Widening Operator. The widening operator $\nabla_{\mathfrak{M}} : \mathfrak{M} \times \mathfrak{M} \to \mathfrak{M}$ is defined by:

$$(M_1 \nabla_{\mathfrak{M}} M_2)(\mathbb{l}) = \begin{cases} M_1(\mathbb{l}) \text{ if } M_2(\mathbb{l}) \leq M_1(\mathbb{l}) \\ M_2(\mathbb{l}) \text{ if } M_1(\mathbb{l}) = 0 \wedge M_2(\mathbb{l}) > 0 \\ \infty \qquad \text{otherwise} \end{cases}$$

It will ensure that the chain of values taken by $\mathsf{E}[q_t]$ in line 8 always stabilizes after a finite number of steps. We refer to [7,18] for a formal definition of widening and merely note that $M_1 \sqcup_{\mathfrak{M}} M_2 \leq_{\mathfrak{M}} M_1 \nabla_{\mathfrak{M}} M_2$.

Granularity Function. Granularity functions have been introduced in [23] in order to have control over the coarseness of the abstraction and to enforce termination of the worklist algorithm; we can adapt them to this setting. The most obvious choice for a granularity function $H : \mathfrak{M} \to \mathcal{H}$ might be the identity function, but it turns out that this choice may lead to nontermination of the algorithm. A more interesting choice is $H(E) = \mathit{dom}(E)$, meaning that only the domain of the extended multiset is of interest; we have used this choice to compute our examples. In general, in order to ensure termination of the algorithm, we will require that a granularity function H is *finitary*, i.e. for all choices of finite sets $\mathbf{LL}_{\text{fin}} \subseteq \mathbf{Loc} \times (\mathbf{Lab} \cup \mathbf{Val}^*)$, H specializes to

$$H : (\mathbf{LL}_{\text{fin}} \to \mathbb{N} \cup \{\infty\}) \to \mathcal{H}_{\text{fin}}$$

for some finite subset $\mathcal{H}_{\text{fin}} \subseteq \mathcal{H}$.

We are now able to state a general termination result for the construction of the finite automaton.

Theorem 6.1 *If the granularity function H is finitary, then the worklist algorithm always terminates.*

Proof. This is proved by contradiction. So let us fix a finite set $\mathbf{LL}_{\text{fin}}$ as appropriate for the program considered and let us consider a non-terminating execution of the worklist algorithm. It is immediate that line 3 of Table 12 must execute infinitely often. It is also clear that Q and E grow in a non-decreasing manner.

Also the set $\{H(\mathsf{E}[q]) \ : \ q \in \mathsf{Q}\}$ grows in a non-decreasing manner and since H is finitary, the value of the set will stabilize. Subsequently, the test in line 2 of Table 13 must always succeed and hence lines 4–6 cannot be executed any more. This shows that also Q stabilizes.

Next consider the vector $(\mathsf{E}[q])_{q\in\mathsf{Q}}$ which is known to grow in a non-decreasing manner. It follows from the properties of the widening operator $\nabla_{\mathfrak{M}}$ that $(\mathsf{E}[q])_{q\in\mathsf{Q}}$ must eventually stabilize and therefore W does not grow from this point onwards.

Each subsequent execution of lines 4–6 of Table 12 will remove an element from the finite set W. It follows that at some point the test in line 3 of Table 12 yields false and that the algorithm terminates. This constitutes our desired contradiction. □

6.3 Procedure enabled

We now return to the definition of the procedure $\mathsf{enabled}_{(\hat{\rho},\hat{S})}(E)$ used in the worklist algorithm; it is shown in Table 14. Recall that E is the extended multiset of exposed actions in the state of interest, and remember that $(\hat{\rho}, \hat{S}) \models^{\bigsqcup\mathcal{T}} N_0$ holds.

First of all, $\mathsf{enabled}_{(\hat{\rho},\hat{S})}(E)$ shall only contain labels $\mathbb{l}$ which are exposed in E, hence $\mathbb{l} \in dom(E)$. Then we have to distinguish three cases:

- If ℓ is the label of an outputting action or $\mathsf{b\text{-}eval}$, then $(l, \ell) \in \mathsf{enabled}_{(\hat{\rho},\hat{S})}(E)$, because these actions can always execute.

$$
\begin{aligned}
&\mathsf{enabled}_{(\hat\rho,\hat S)}(E) = dom(E) \cap \\
&\quad (\{(l,\ell) \quad : \ell \text{ is the label of an } \mathsf{bcst}\text{-}, \mathsf{out}\text{-}, \text{ or } \mathsf{b\text{-}eval}\text{-action}\} \cup \\
&\quad \{(l,\ell[t]) : \hat\rho \vDash_1 T : \hat S(l) \triangleright \hat T_\bullet \wedge \\
&\qquad\qquad \ell \text{ is the label of an } \mathsf{in}(T)\text{-action and } t \in \hat T_\bullet \text{ and } E(l,t) > 0\} \cup \\
&\quad \{(l,\ell) \quad : \hat\rho \vDash_1 T : \hat S(l) \triangleright \hat T_\bullet \wedge \\
&\qquad\qquad ((\ell \text{ is the label of an } \mathsf{read}(T)\text{-action and } \exists\, t \in \hat T_\bullet.\ E(l,t) > 0) \vee \\
&\qquad\qquad (\ell \text{ is the label of an } \mathsf{abs}(T)\text{-action and } \forall\, t \in \hat T_\bullet.\ E(l,t) = 0))\})
\end{aligned}
$$

Table 14: Procedure enabled

- If ℓ is the label of an $\mathsf{in}(T)$-action, we have to check which tuples t contained in E match the template T and can be input, and record $(l, \ell[t]) \in \mathsf{enabled}_{(\hat\rho,\hat S)}(E)$. To find the matching tuples we invoke the judgment $\hat\rho \vDash_1 T : \hat S(l) \triangleright \hat T_\bullet$ such that by Lemma 4.1 $\hat T_\bullet$ contains all matching tuples of $\hat S(l)$.
- If ℓ is the label of an $\mathsf{read}(T)$- or $\mathsf{abs}(T)$-action, we also invoke the judgment for matching. We record $(l, \ell) \in \mathsf{enabled}_{(\hat\rho,\hat S)}(E)$ if there is one matching tuple in $\hat T_\bullet$ in the case of read, or if there are no matching tuples in the case of abs.

The correctness of the definition of $\mathsf{enabled}_{(\hat\rho,\hat S)}$ amounts to strengthening Lemma 3.2:

Lemma 6.2 *Suppose* $(\hat\rho, \hat S) \vDash^{\bigsqcup \mathcal{T}} M$ *holds. If* $\mathcal{T} \vdash M \xrightarrow{\mathbb{l}}_G N$*, then* $\mathbb{l} \in \mathsf{enabled}_{(\hat\rho,\hat S)}(\mathcal{E}[\![M]\!])$.

Proof. We proceed by induction on the rules of the transition system in Table 2. For rules bcst, out, and b-eval the result follows directly from Lemma 3.2. In case in, we know $\mathbb{l} = (l, \ell[t])$, $S(t) > 0$, $match(T, t) = \sigma$. Using the assumption $(\hat\rho, \hat S) \vDash^{\bigsqcup \mathcal{T}} M$ we can therefore establish $\hat\rho \vDash_1 T : \hat S(l) \triangleright \hat T_\bullet$, and use Lemma 4.1 to have $t \in \hat T_\bullet$. Lemma 3.2 gives $(l, t) \in \mathcal{E}[\![M]\!]$, which establishes $(l, \ell[t]) \in \mathsf{enabled}_{(\hat\rho,\hat S)}(\mathcal{E}[\![M]\!])$. The cases for read and abs are proved analogously to the one for in. □

6.4 Correctness

We can now establish the main result which implies that we can use the worklist algorithm to produce abstract transition systems for which the embedding theorem (Theorem 3.7) is applicable. This result is independent of the choice of the granularity function H:

Theorem 6.3 *Suppose* $(\hat{\rho}, \hat{S}) \models^{\mathcal{T}}_{\bigsqcup} N_0$ *holds for a network* $\mathsf{let}\ A_1 \triangleq P_1; \ldots; A_k \triangleq P_k\ \mathsf{in}\ N_0$ *and a network topology* $\mathcal{T}$*, and furthermore that the worklist algorithm terminates and produces an abstract transition system* $\mathcal{A}$*. Then* $\mathcal{A}$ *faithfully describes the evolution of* N_0*.*

Proof. Consider the last time where the state q was removed from the worklist W in line 4 of the worklist algorithm in Table 12. Letting δ_0 and E_0 denote the corresponding values of the data structures we have $\mathsf{E}_0[q] = \mathsf{E}[q]$ and hence $M \rhd \mathsf{E}_0[q]$.

Since $\mathcal{T} \vdash M \xrightarrow{\mathbb{l}}_G N$ it follows that $G \in \mathcal{T}$ and also $\mathbb{l} \in \mathsf{enabled}_{(\hat{\rho},\hat{S})}(\mathcal{E}[\![M]\!])$ by Lemma 6.2 and Theorem 4.3, and hence G and $\mathbb{l}$ are selected for consideration in lines 5 and 6 of the algorithm, respectively. By Corollary 5.4 it follows that E in line 7 of the algorithm is an extended multiset with $N \rhd E$.

By line 7 and the definition of update in Table 13 it is immediate that we identify a state q' in lines 2–5 of Table 13 and that after execution of lines 6–8 of Table 13 we have $(q, (G, \mathbb{l}), q') \in \delta_1$ and $E \leq_{\mathfrak{M}} \mathsf{E}_1[q']$, where δ_1 and E_1 denote the corresponding values of the data structures at this time.

It is immediate that the values of $\mathsf{E}[.]$ grow in a non-decreasing manner. Writing δ and E for the final values of the data structures, we have $(q, (G, \mathbb{l}), q') \in \delta$ and $E \leq_{\mathfrak{M}} \mathsf{E}_1[q'] \leq_{\mathfrak{M}} \mathsf{E}[q']$, which establishes the claim. $\square$

7 Discussion

In this paper, we have dealt with the problem of analysing the behaviour of broadcast networks under changing network connectivity. For networks modelled in the calculus bKlaim, we have defined an algorithm which constructs a finite automaton such that all transition sequences obtained by the evolution of a network correspond to paths in this automaton. We captured the nature of our abstraction by defining a 3-valued interpretation of a temporal logic such that a formula evaluating to a definite truth value on the automaton would imply the truth or falsity of that formula on the transition system of the concrete network. In the following, we conclude this paper by discussing related and possible future work.

7.1 Related Work

Prasad [25] has introduced the *Calculus of Broadcasting Systems (CBS)* as the first process calculus with broadcast as communication primitive; broadcast is taken to be global, inspired by local area networks in which nodes overhear all messages. Ene and Muntean [9] describe the *bπ-calculus* which builds on the ideas of CBS, but introduces a notion of channels inherited from the π-calculus. Nanz and Hankin [15] have introduced *CBS*$^\sharp$ which uses a local version of broadcast in order to be able to model wireless networks. They express the notion of neighborhoods of nodes by connectivity graphs, an idea we have adapted for bKlaim. Merro [13] has defined a *Calculus of Mobile Ad-Hoc Networks (CMN)* which employs local broadcast as well, but expresses the neighborhood by a distance function on locations. In contrast to these works, bKlaim does not strive to be a definitive model for a specific networking paradigm such as LANs or mobile ad-hoc networks, although we use the idea of wireless networks throughout the paper in order to provide intuition. Instead we were looking for a rather simple calculus for a more general study of broadcast. This is supported by the asynchronous nature of bKlaim which contains as traces the behaviour of the synchronous models.

A number of works is concerned with analysing wireless networks, in particular mobile ad-hoc networks. Bhargavan, Obradovic, and Gunter [2] have studied verification of routing protocols for mobile ad-hoc networks. For a loop-freedom property expressed in temporal logic they can use the model checker SPIN to expose flaws on a fixed network setup. Chiyangwa and Kwiatkowska [6] also use model checking in order to check timing properties of a protocol for mobile ad-hoc networks; they also employ a fixed topology. In the work of Zakiuddin et al. [27] CSP and a refinement checker have been applied to model and analyse a self-configuration protocol. They succeed in integrating the mobility aspect by modelling links as individual processes which can be either up or down, but soon experience space explosion. Nanz and Hankin [15] have used static analysis to establish security properties for mobile ad-hoc networks. For these properties they can safely abstracts away the mobility aspect, and thus define the analysis again over fixed connectivities only.

A multitude of works have addressed the use of abstraction in the realm of model checking, most of which are based on property-preserving simulation relations for state transition systems (see e.g. references in [8]). More recently, the topic of using the theory of abstract interpretation to compute the abstraction (an approach that is more closely related to ours) is receiving considerable attention. Bruns and Godefroid [5] show how to use 3-valued interpretation of modal logic formulae over *partial Kripke structures*, which provide a 3-valued interpretation of each atomic proposition associated with a state. Dams, Gerth, and Grumberg [8] define *mixed transtion systems*, which use two separate transition systems to express an over- and an under-approximation and are thus able to accommodate for the preservation of universal as well as existential temporal properties. Larsen and Thomson [12] introduce *modal transition systems* which also combine two transition relations, referred to as "may" and "must", where the must-relation is required to be a subset of the may-relation (in contrast to mixed transition systems). Huth, Jagadeesan, and Schmidt [10] use a generalisation of modal transition systems and a 3-valued logic for model checking of partial state spaces. Note that the mentioned approaches focus much on the general definitions of the abstractions and leave open the choice of an appropriate abstract domain as well as the algorithmic construction of the abstraction. We have instead focused on providing a concrete, implemented algorithm that provides these choices for the analysis of a specific language.

7.2 Future Work

As a main direction for future work, we would like to investigate adapting our approach to construct the abstract transition system as a 3-valued structure itself [12], in order to model the cases where we can show that progress is enforced. It would also be interesting to investigate the possibility of constructing a model checker in this setting, which would give us – together with the Standard ML implementation of the Monotone Framework we already have – a complete automation of the framework. Using such a framework, we could then analyse concrete application scenarios, for example in a security setting, where one might show that certain attacks on networks are enabled or prevented by a given series of topology changes.

References

[1] L. Bettini, V. Bono, R. De Nicola, G. Ferrari, D. Gorla, M. Loreti, E. Moggi, R. Pugliese, E. Tuosto, and B. Venneri. The Klaim project: theory and practice. In *Proceedings of the IST/FET International Workshop on Global Computing: Programming Environments, Languages, Security and Analysis of Systems (GC'03)*, volume 2874 of *Lecture Notes in Computer Science*. Springer, 2003.

[2] Karthikeyan Bhargavan, Davor Obradovic, and Carl A. Gunter. Formal verification of standards for distance vector routing protocols. *Journal of the ACM*, 49(4):538–576, 2002.

[3] C. Bodei, M. Buchholtz, P. Degano, H. Riis Nielson, and F. Nielson. Static validation of security protocols. *Journal of Computer Security*, 13(3):347–390, 2005.

[4] Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Control flow analysis for the pi-calculus. In *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 1998.

[5] Glenn Bruns and Patrice Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *Proceedings of the 11th International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 274–287. Springer, 1999.

[6] Sibusisiwe Chiyangwa and Marta Kwiatkowska. A timing analysis of AODV. In *Proceedings of the 7th IFIP WG 6.1 International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'05)*, volume 3535 of *Lecture Notes in Computer Science*, pages 306–321. Springer, 2005.

[7] Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'79)*, pages 269–282. ACM, 1979.

[8] Dennis Dams, Rob Gerth, and Orna Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems*, 19(2):253–291, 1997.

[9] Christian Ene and Traian Muntean. A broadcast-based calculus for communicating systems. In *Proceedings of the 6th International Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA'03)*, 2001.

[10] Michael Huth, Radha Jagadeesan, and David A. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In *Proceedings of the 10th European Symposium on Programming Languages and Systems (ESOP'01)*, pages 155–169. Springer, 2001.

[11] Steven Cole Kleene. *Introduction to Metamathematics*, volume 1 of *Biblioteca Mathematica*. North-Holland, 1952.

[12] Kim Guldstrand Larsen and Bent Thomsen. A modal process logic. In *Proceedings of the Third Annual Symposium on Logic in Computer Science (LICS'88)*, pages 203–210. IEEE Computer Society, 1988.

[13] Massimo Merro. An observational theory for mobile ad hoc networks. In *Proceedings of the 23rd International Conference on the Mathematical Foundations of Programming Semantics (MFPS'07)*, volume 173 of *Electronic Notes in Theoretical Computer Science*, pages 275–293, 2007.

[14] Sebastian Nanz. *Specification and Security Analysis of Mobile Ad-Hoc Networks.* PhD thesis, Imperial College London, 2006.

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

[16] Sebastian Nanz, Flemming Nielson, and Hanne Riis Nielson. Topology-dependent abstractions of broadcast networks. In *Proceedings of the 18th International Conference on Concurrency Theory (CONCUR'07)*, 2007. To appear.

[17] Rocco De Nicola and Frits W. Vaandrager. Action versus state based logics for transition systems. In *Proceedings of the LITP Spring School on Semantics of Systems of Concurrent Processes*, volume 469 of *Lecture Notes in Computer Science*, pages 407–419, 1990.

[18] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer, 1999.

[19] Flemming Nielson, Hanne Riis Nielson, and Mooly Sagiv. A Kleene analysis of mobile ambients. In *European Symposium on Programming (ESOP'00)*, volume 1782 of *Lecture Notes in Computer Science*, pages 305–319. Springer, 2000.

[20] Flemming Nielson, Hanne Riis Nielson, and Mooly Sagiv. Kleene's logic with equality. *Information Processing Letters*, 80:131–137, 2001.

[21] Flemming Nielson, Hanne Riis Nielson, and Helmut Seidl. A succinct solver for ALFP. *Nordic Journal of Computing*, 9(4):335–372, 2002.

[22] Hanne Riis Nielson and Flemming Nielson. Flow logic: A multi-paradigmatic approach to static analysis. 2566:223–244, 2002.

[23] Hanne Riis Nielson and Flemming Nielson. A monotone framework for CCS. Submitted for publication, 2006.

[24] Hanne Riis Nielson and Flemming Nielson. Data flow analysis for CCS. In *Program Analysis and Compilation. Theory and Practice*, volume 4444 of *Lecture Notes in Computer Science*. Springer, 2007.

[25] K. V. S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25(2-3):285–327, 1995.

[26] Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'99)*, pages 105–118. ACM, 1999.

[27] Irfan Zakiuddin, Michael Goldsmith, Paul Whittaker, and Paul Gardiner. A methodology for model-checking ad-hoc networks. In *Proceedings of the 10th International SPIN Workshop on Model Checking Software (SPIN'03)*, volume 2648 of *Lecture Notes in Computer Science*, pages 181–196. Springer, 2003.