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 [11], 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) [13] on abstract transition systems, which 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 [14], 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 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. We develop a Monotone Framework and worklist algorithm to construct abstract transition systems for bKlaim networks in §4. We conclude in §5. A technical report [12] contains full proofs.

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 [19], 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$^\sharp$ [11] and CMN [9] 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

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.

$N ::= l :: P$		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	$\mid\ \mathsf{in}^\ell(T)$		input
$P ::= \mathsf{nil}$		null process	$T ::= F \mid F, T$		templates
$\mid\ a^\ell.P$		action prefixing	$F ::= f \mid\ !x$		template fields
$\mid\ P_1 \mid P_2$		parallel composition	$t ::= f \mid f, t$		tuples
$\mid\ A$		process invocation	$f ::= v \mid l \mid x$		tuple fields

Table 1. Syntax of a fragment of bKlaim

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. $\mathbf{Loc} \subseteq \mathbf{Val}$. A ground tuple t is an element of $\mathbf{Val}^*$. *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; within a template, x must not occur both as a variable and a formal field, or in more than one formal field. The set $fv(t)$ containing the free variables of tuple t are defined as usual, and the definition of fv can be extended to templates, actions, and processes. 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:

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

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.

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 behaviour. 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 parameterised in the free variables of t.

Actions are equipped with labels $\ell \in \mathbf{Lab}$ which facilitate the analysis in §4. The action $\mathsf{bcst}^\ell(t)$ places a tuple t into the set of tuple spaces belonging to the current neighbours of the sending node, thus describing local broadcast. Neighbourhoods 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. Using $\mathsf{in}^\ell(T)$, processes retrieve tuples which match the template T from their private tuple space and remove it. 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. Because of space constraints, some actions contained in the full version of bKlaim (such as process migration) are omitted in this description. We refer the reader to the companion technical report [12].

Example 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:

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

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

The protocol is initiated on network *Net* when node $\mathtt{l}_1$ executes the process *Snd* to search for information on topic `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 `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 [11,10] 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 *neighbourhood* 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 [5] for example) we are able to express the behaviour 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 trigger both link failures and the establishment of new links.

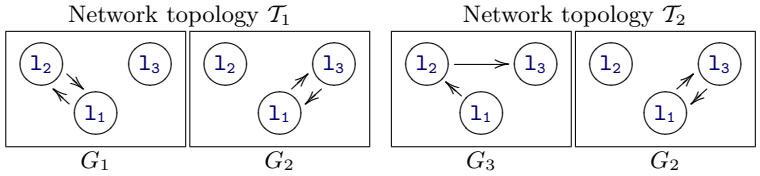
Connectivity graphs provide a snapshot of the network connectivity. In contrast, a *network topology* $\mathcal{T}$ is a set of connectivity graphs which share the same set of vertices. We use network topologies to express the set of possible configurations a particular network may be in.

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:

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

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. Continuing Example 1, we define the following network topologies over $V(Net)$:



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 straightforward *structural congruence* $M \equiv N$ and *template matching* semantics 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 parametrised 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 §4, 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.

The bcst-rule puts a tuple t into all tuple spaces in the current neighbourhood $G(l)$ of the sender location l, where the current neighbourhood is nondeterministically chosen from the network topology $\mathcal{T}$. Rule out puts a tuple t into the private tuple space at location l. 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.

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

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

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

$$\frac{\mathcal{T} \vdash M \xrightarrow{\mathbb{1}} M'}{\mathcal{T} \vdash M \parallel N \xrightarrow{\mathbb{1}} M' \parallel N} \qquad \frac{N \equiv M \qquad \mathcal{T} \vdash M \xrightarrow{\mathbb{1}} M' \qquad M' \equiv N'}{\mathcal{T} \vdash N \xrightarrow{\mathbb{1}} N'}$$

Table 2. Reduction relation of bKlaim

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

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

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

Table 3. Structural congruence and template matching of bKlaim

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) [13] to describe their properties. In order to accommodate the notion of abstraction in the logic, we use a 3-valued interpretation on abstract transition systems so that a formula evaluates to "unknown" whenever the abstraction prevents us from obtaining a definite result; when evaluating 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, occur-

rences 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 $M(\ell\!\!\ell) \in \mathbb{N}$, or an infinite number, in which case $M(\ell\!\!\ell) = \infty$ (where $\ell\!\!\ell$ ranges over (l, ℓ) or (l, t)). The set $\mathfrak{M}$ is equipped with a partial ordering $\leq_{\mathfrak{M}}$ defined by:

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

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 as follows:

$$\begin{aligned}
\mathcal{E}[\![N_1 \parallel N_2]\!] &= \mathcal{E}[\![N_1]\!] +_{\mathfrak{M}} \mathcal{E}[\![N_2]\!] & \mathcal{E}_l[\![\mathsf{nil}]\!] &= \bot_{\mathfrak{M}} \\
\mathcal{E}[\![l :: P]\!] &= \mathcal{E}_l[\![P]\!] & \mathcal{E}_l[\![a^\ell.P]\!] &= \bot_{\mathfrak{M}}[(l, \ell) \mapsto 1] \\
\mathcal{E}[\![l :: S]\!] &= \textstyle\sum_{\mathfrak{M}, t} \bot_{\mathfrak{M}}[(l, t) \mapsto S(t)] & \mathcal{E}_l[\![P_1 \mid P_2]\!] &= \mathcal{E}_l[\![P_1]\!] +_{\mathfrak{M}} \mathcal{E}_l[\![P_2]\!] \\
& & \mathcal{E}_l[\![A]\!] &= \mathcal{E}_l[\![P]\!] \text{ if } A \triangleq P
\end{aligned}$$

Note that in the case for tuple spaces, every tuple $t \in S$ is recorded with according multiplicity $S(t)$ at location l. In the case of actions $a^\ell.P$, the label ℓ is recorded at location l with multiplicity 1. The remaining cases are straightforward. The operations involved in the definition of $\mathcal{E}$ are all monotonic such that a least fixed point is ensured by Tarski's fixed point theorem.

Example 3. Continuing Example 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 1. *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 $(l, \ell), (l, t) \in dom(\mathcal{E}[\![M]\!])$.*

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

Definition 2. *We say that an abstract transition system $(\mathsf{Q}, q_0, \delta, \mathsf{E})$* faithfully describes *the evolution of a network N_0 if:*

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

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

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

In §4 we shall show how to construct an abstract transition system that faithfully describes the evolution of a given network N.

Example 4. For the network $(Net, \mathcal{T}_1)$ of Example 1, the static analysis of §4 generates an abstract transition system with 27 states and 46 transitions. We look at one of these transitions in detail, namely $(q_3, (G_1, (\mathtt{l}_2, 4[\mathtt{t}, \mathtt{i}_2])), q_6) \in \delta$. 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

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

and after a transition under connectivity graph G_1 with action $(\mathtt{l}_2, 4[\mathtt{t}, \mathtt{i}_2])$ (and analogously for G_2, as label 4 denotes a (local) input action which thus does not depend on connectivity), we end up in state q_6 that represents

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

3.3 Interpretation of ACTL Properties

In order to express properties about a network, we are using a variant of Action Computation Tree Logic (ACTL) [13], which allows us to utilise the labels $(G, \mathbb{l})$ 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 following grammar describes the syntax of path formulae ϕ and state formulae γ:

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

Here, $\mathbb{l}$ denotes (l, ℓ) or (l, t), $\exists$ is a path quantifier, Ω is a set of transition labels $(G, \mathbb{l})$ 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 $(N_0, (G_0, \mathbb{l}_0), N_1, (G_1, \mathbb{l}_1), \dots)$ where $\Pi(i) \xrightarrow{\mathbb{l}_i}_{G_i} \Pi(i+1)$ for $i \geq 0$ (we write $\Pi(i)$ for N_i, and $\Pi[i]$ for $(G_i, \mathbb{l}_i)$).

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

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. Several choices of 3-valued logics exist and we choose here to use Kleene's strongest regular 3-valued logic [7]; this is in line with the developments of [3,20]. Formulae defined over the abstraction may make use of all three truth values, but unlike e.g. [20,15], 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 [16].

Let $L(q, \mathbb{l}) = 0$ if $\mathbb{l} \notin \mathsf{E}[q]$, and $1/2$ otherwise. Furthermore, let $D_\Omega(q_s, q_t) = 0$ if $(q_s, (G, \mathbb{l}), q_t) \notin \delta$ for all $(G, \mathbb{l}) \in \Omega$, and $1/2$ otherwise. The satisfaction relations $[q \vDash^3 \phi]$ and $[\pi \vDash^3 \gamma]$ for states q and paths $\pi = (q_0, (G_0, \mathbb{l}_0), q_1, \dots)$ is defined as follows:

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

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 1. *Suppose* $(\mathsf{Q}, q_0, \delta, \mathsf{E})$ *faithfully describes the evolution of network* N_0*, and* $\mathcal{T} \vdash N_0 \rightarrow^* N$*. 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* $[\pi \models^3 \gamma] \sqsubseteq [\Pi \models \gamma]$.

Example 5. For the abstract transition system for $(\mathit{Net}, \mathcal{T}_1)$ of Example 1 and 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 $(\mathit{Net}, \mathcal{T}_2)$ we get the result $1/2$. Using Theorem 1, this means that $(\mathit{Net}, \mathcal{T}_1)$ 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 $(\mathit{Net}, \mathcal{T}_2)$ the analysis states that the abstraction prevents a definite answer.

4 Constructing Abstract Transition Systems

In this section we develop an *algorithm* for constructing an abstract transition system. The development amounts to adapting the approach of [17,18] from the synchronous language CCS to the asynchronous message-passing language bKlaim. This involves solving the challenge of how to deal with the name bindings resulting from message passing. We employ a classical Control Flow Analysis like in [6], using judgements of the form

$$(\hat{\rho}, \hat{S}) \models^G N.$$

The analysis states that $\hat{\rho}$ correctly describes the name bindings and $\hat{S}$ the contents of tuple stores that may take place during the execution of the net N using the connectivity graph G. In the subsequent development we shall not need the $\hat{S}$ component as it will be modelled more precisely using the multisets of exposed labels. We leave the details to the technical report [12].

4.1 Transfer Functions

The abstraction function $\mathcal{E}$ only gives us the information of interest for the initial network. We shall now present auxiliary functions allowing us to approximate how the information evolves during the execution of the network.

Once an action has participated in an interaction, some new actions may become exposed and some may cease to be exposed. We shall now introduce two functions $\mathcal{G}^G_{\hat\rho}$ and $\mathcal{K}$ approximating this information. The relevant information will be an element of:

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

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 $\leq_{\mathfrak{M}}$.

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:

$$\begin{array}{rlrl}
\mathcal{G}^G_{\hat\rho}[\![N_1 \parallel N_2]\!] &= \mathcal{G}^G_{\hat\rho}[\![N_1]\!] \sqcup_{\mathfrak{T}} \mathcal{G}^G_{\hat\rho}[\![N_2]\!] & \mathcal{G}^G_{\hat\rho,l}[\![\mathsf{nil}]\!] &= \bot_{\mathfrak{T}} \\
\mathcal{G}^G_{\hat\rho}[\![l :: P]\!] &= \mathcal{G}^G_{\hat\rho,l}[\![P]\!] & \mathcal{G}^G_{\hat\rho,l}[\![a^\ell.P]\!] &= \tilde{\mathcal{G}}^G_{\hat\rho,l}[\![a^\ell.P]\!] \sqcup_{\mathfrak{T}} \mathcal{G}^G_{\hat\rho,l}[\![P]\!] \\
\mathcal{G}^G_{\hat\rho}[\![l :: S]\!] &= \bot_{\mathfrak{T}} & \mathcal{G}^G_{\hat\rho,l}[\![P_1 \mid P_2]\!] &= \mathcal{G}^G_{\hat\rho,l}[\![P_1]\!] \sqcup_{\mathfrak{T}} \mathcal{G}^G_{\hat\rho,l}[\![P_2]\!] \\
& & \mathcal{G}^G_{\hat\rho,l}[\![A]\!] &= \mathcal{G}^G_{\hat\rho,l}[\![P]\!] \text{ if } A \triangleq P
\end{array}$$

$$\begin{array}{l}
\tilde{\mathcal{G}}^G_{\hat\rho,l}[\![\mathsf{bcst}^\ell(t).P]\!] = \bot_{\mathfrak{T}}[(l,\ell) \mapsto \mathcal{E}_l[\![P]\!] +_{\mathfrak{M}} (\sum_{\mathfrak{M}, l' \in G(l), u \in \hat\rho[\![t]\!]} \bot_{\mathfrak{M}}[(l',u) \mapsto 1])] \\
\tilde{\mathcal{G}}^G_{\hat\rho,l}[\![\mathsf{out}^\ell(t).P]\!] = \bot_{\mathfrak{T}}[(l,\ell) \mapsto \mathcal{E}_l[\![P]\!] +_{\mathfrak{M}} (\sum_{\mathfrak{M}, u \in \hat\rho[\![t]\!]} \bot_{\mathfrak{M}}[(l,u) \mapsto 1])] \\
\tilde{\mathcal{G}}^G_{\hat\rho,l}[\![\mathsf{in}^\ell(T).P]\!] = \bot_{\mathfrak{T}}[(l,\ell) \mapsto \mathcal{E}_l[\![P]\!]]
\end{array}$$

Note that the function carries two more parameters, namely a connectivity graph G and the environment $\hat\rho$ which we obtain from the Control Flow Analysis (see above) and which describes the occurring name bindings. 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 (where $\bigsqcup \mathcal{T}$ is the graph which contains the edges of all $G \in \mathcal{T}$), as it can be shown to imply $(\hat\rho, \hat S) \models^G N_0$ for all $G \in \mathcal{T}$.

All actions $a^\ell.P$ then expose $\mathcal{E}_l[\![P]\!]$, i.e. the actions of the continuation process. Furthermore, $\mathsf{bcst}^\ell(t)$ exposes the tuples $u \in \hat\rho[\![t]\!]$ for all locations $l' \in G(l)$ in the neighbourhood of the sending process; note that $\hat\rho[\![t]\!]$ describes an overapproximation of the ground tuples t can evaluate to. The action $\mathsf{out}^\ell(t)$ exposes all $u \in \hat\rho[\![t]\!]$ only at location l. Analogous to the case for exposed actions, a least fixed point of $\mathcal{G}^G_{\hat\rho}$ can be obtained.

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

Lemma 2. *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 $\mathcal{G}^G_{\hat\rho}[\![N]\!] \leq_{\mathfrak{T}} \mathcal{G}^G_{\hat\rho}[\![M]\!]$.*

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

Killed Actions. We define 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:

$$\begin{array}{rlrl} \mathcal{K}[\![N_1 \parallel N_2]\!] &= \mathcal{K}[\![N_1]\!] \sqcap_{\mathfrak{T}} \mathcal{K}[\![N_2]\!] & \mathcal{K}_l[\![\mathsf{nil}]\!] &= \top_{\mathfrak{T}} \\ \mathcal{K}[\![l :: P]\!] &= \mathcal{K}_l[\![P]\!] & \mathcal{K}_l[\![a^\ell.P]\!] &= \top_{\mathfrak{T}}[(l,\ell) \mapsto \bot_{\mathfrak{M}}[(l,\ell) \mapsto 1]] \sqcap_{\mathfrak{T}} \mathcal{K}_l[\![P]\!] \\ \mathcal{K}[\![l :: S]\!] &= \top_{\mathfrak{T}} & \mathcal{K}_l[\![P_1 \mid P_2]\!] &= \mathcal{K}_l[\![P_1]\!] \sqcap_{\mathfrak{T}} \mathcal{K}_l[\![P_2]\!] \\ & & \mathcal{K}_l[\![A]\!] &= \mathcal{K}_l[\![P]\!] \text{ if } A \triangleq P \end{array}$$

Note that when actions $a^\ell.P$ execute at location l, it is clear that one occurrence (l, ℓ) can be killed. A greatest fixed point of $\mathcal{K}$ can be obtained.

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

Lemma 3. *If $M \equiv N$, then $\mathcal{K}[\![M]\!] = \mathcal{K}[\![N]\!]$. Furthermore, if $\mathcal{T} \vdash M \xrightarrow{\mathbb{1}}_G N$ then $\mathcal{K}[\![M]\!] \leq_{\mathfrak{T}} \mathcal{K}[\![N]\!]$.*

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

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

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

We can use $\mathcal{G}^G_{\hat{\rho}}$ and $\mathcal{K}$ to obtain a transfer function as in a classical Monotone Framework, where E represents a set of exposed actions, and $\mathcal{K}[\![N_0]\!](\mathbb{1})$ (resp. $\mathcal{G}^G_{\hat{\rho}}[\![N_0]\!](\mathbb{1})$) represent the actions which are no longer (resp. newly) exposed by a transition with label $\mathbb{1}$:

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

Example 6. Continuing Example 4, we can calculate that

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

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

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

Theorem 2. *Consider the network* $\mathsf{let}\ A_1 \triangleq P_1; \ldots; A_k \triangleq P_k\ \mathsf{in}\ N_0$ *and suppose* $(\hat{\rho}, \hat{S}) \vDash^{\sqcup}_{\mathcal{T}} N_0$. *If $\mathcal{T} \vdash N_0 \to^* M \xrightarrow{\mathbb{1}}_G N$ then $\mathcal{E}[\![N]\!] \leq_{\mathfrak{M}} \mathsf{transfer}_{(G,\mathbb{1}),\hat{\rho}}(\mathcal{E}[\![M]\!])$.*

4.2 Worklist Algorithm

We are interested in analysing networks N_0 for which we assume in the following that $(\hat{\rho}, \hat{S}) \vDash^{\sqcup}_{\mathcal{T}} N_0$ holds. We shall now construct an abstract transition system which faithfully describes the evolution of N_0 as specified in §3.2.

The key algorithm is a *worklist algorithm*, which starts out from the initial state and constructs the abstract transition system by adding more and more states and transitions. The algorithm makes use of several auxiliary operations:

– Given a state q_s representing some exposed actions, $\mathsf{enabled}$ selects those labels $\mathbb{l}$ that represent actions that may interact in the next step.
– Once the labels $\mathbb{l}$ have been selected, we can use the function $\mathsf{transfer}$ of §4.1.
– Finally, update constructs an appropriate target state q_t and records the transition $(q_s, (G, \mathbb{l}), q_t)$.

The algorithm's main data structures 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; and, a set δ of the current transitions. The algorithm has the following form:

```
1 Q := {q0};  E[q0] := ℰ⟦N0⟧;  W := {q0};  δ := ∅;
2 while W ≠ ∅ do
3     select qs from W;  W := W\{qs};
4     foreach G ∈ 𝒯 do
5         foreach 𝕝 ∈ enabled_(ρ̂,Ŝ)(E[qs]) do
6             let E = transfer_((G,𝕝),ρ̂)(E[qs]) in update(qs, (G, 𝕝), E)
```

In line 1 both the set of states and the worklist are initialised 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 $\mathbb{l} \in \mathsf{enabled}_{(\hat{\rho},\hat{S})}(\mathsf{E}[q_s])$ (lines 4–5) the procedure $\mathsf{transfer}_{(G,\mathbb{l}),\hat{\rho}}(\mathsf{E}[q_s])$ 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 $\mathsf{update}(q_s, (G, \mathbb{l}), E)$.

Procedure update. The procedure update is specified as follows:

```
1 procedure update(qs, (G, 𝕝), E)
2     if there exists q ∈ Q with H(E[q]) = H(E) then qt := q
3     else select qt from outside Q;  Q := Q ∪ {qt};  E[qt] :=⊥𝔐;
4     if ¬(E ≤𝔐 E[qt]) then E[qt] := E[qt]∇𝔐 E;  W := W ∪ {qt};
5     δ := δ\{(qs, (G, 𝕝), q) : q ∈ Q} ∪ {(qs, (G, 𝕝), qt)};
```

First, the target state q_t is determined in lines 2–3, where the reusability of a state is checked by using a *granularity function* H, which is described below.

In line 4 it is ensured that the description $\mathsf{E}[q_t]$ includes the required information E by using a widening operator $\nabla_{\mathfrak{M}}$ 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 5. The triple $(q_s, (G, \mathbb{l}), q_t)$ is added, but we also have to remove any previous transitions from q_s with label $(G, \mathbb{l})$, 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.

Granularity Function. 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 $H(E) =$

$dom(E)$, meaning that only the domain of the extended multiset is of interest; we have used this choice to compute our examples. We can show that termination of the algorithm is ensured once H is chosen to be *finitary*, meaning that $\mathcal{H}$ is finite on all finite sets of labels.

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 stabilises after a finite number of steps. We refer to [4,14] for a formal definition of widening and merely note that $M_1 \sqcup_{\mathfrak{M}} M_2 \leq_{\mathfrak{M}} M_1 \nabla_{\mathfrak{M}} M_2$.

Procedure enabled. Recall that E is the extended multiset of exposed actions in the state of interest, and remember that $(\hat{\rho}, \hat{S}) \models^{\sqcup \mathcal{T}} N_0$ holds. Then:

$$\begin{aligned} \mathsf{enabled}_{(\hat{\rho},\hat{S})}(E) = dom(E) \cap (\{(l,\ell) \ : \ \ell \text{ labels an } \mathsf{bcst}\text{- or } \mathsf{out}\text{-action}\} \cup \\ \{(l,\ell[t]) \ : \ \ell \text{ labels an } \mathsf{in}(T)\text{-action}, t \in \hat{\rho}[\![T]\!] \text{ and } E(l,t) > 0\}) \end{aligned}$$

First of all, $\mathsf{enabled}_{(\hat{\rho},\hat{S})}(E)$ shall only contain labels $\mathbb{l}$ which are exposed in E, hence $\mathbb{l} \in dom(E)$. Furthermore, if ℓ is the label of an bcst- or out-action, then $(l,\ell) \in \mathsf{enabled}_{(\hat{\rho},\hat{S})}(E)$, because these actions can always execute; and 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 $(l,\ell[t]) \in \mathsf{enabled}_{(\hat{\rho},\hat{S})}(E)$.

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 1) is applicable.

Theorem 3. *Suppose $(\hat{\rho}, \hat{S}) \models^{\sqcup \mathcal{T}} N_0$ holds for a network 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.*

5 Conclusion

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.

As a main direction for future work, we would like to construct the abstract transition system as a 3-valued structure itself [8], to model the cases where we can show that progress is enforced.

References

1. L. Bettini et al. The Klaim project: theory and practice. In *Global Computing (GC'03)*, volume 2874 of *Lecture Notes in Computer Science*. Springer, 2003.
2. K. Bhargavan, D. Obradovic, and C. A. Gunter. Formal verification of standards for distance vector routing protocols. *Journal of the ACM*, 49(4):538–576, 2002.
3. G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 274–287. Springer, 1999.
4. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Principles of Programming Languages (POPL'79)*, pages 269–282. ACM, 1979.
5. C. Ene and T. Muntean. A broadcast-based calculus for communicating systems. In *Formal Methods for Parallel Programming: Theory and Applications (FMPPTA'03)*, 2001.
6. R. R. Hansen, C. W. Probst, and F. Nielson. Sandboxing in myKlaim. In *Availability, Reliability and Security (ARES'06)*, pages 174–181. IEEE, 2006.
7. S. C. Kleene. *Introduction to Metamathematics*, volume 1 of *Biblioteca Mathematica*. North-Holland, 1952.
8. K. G. Larsen and B. Thomsen. A modal process logic. In *Logic in Computer Science (LICS'88)*, pages 203–210. IEEE Computer Society, 1988.
9. M. Merro. An observational theory for mobile ad hoc networks. In *Mathematical Foundations of Programming Semantics (MFPS'07)*, volume 173 of *Electronic Notes in Theoretical Computer Science*, pages 275–293, 2007.
10. S. Nanz. *Specification and Security Analysis of Mobile Ad-Hoc Networks*. PhD thesis, Imperial College London, 2006.
11. S. Nanz and C. Hankin. A framework for security analysis of mobile wireless networks. *Theoretical Computer Science*, 367(1-2):203–227, 2006.
12. S. Nanz, F. Nielson, and H. R. Nielson. Topology-dependent abstractions of broadcast networks. Technical report IMM-TR-2007-11, Technical University of Denmark, 2007.
13. R. D. Nicola and F. W. Vaandrager. Action versus state based logics for transition systems. In *LITP Spring School on Semantics of Systems of Concurrent Processes*, volume 469 of *Lecture Notes in Computer Science*, pages 407–419, 1990.
14. F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 1999.
15. F. Nielson, H. R. Nielson, and M. 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.
16. F. Nielson, H. R. Nielson, and M. Sagiv. Kleene's logic with equality. *Information Processing Letters*, 80:131–137, 2001.
17. H. R. Nielson and F. Nielson. A monotone framework for CCS. Submitted for publication, 2006.
18. H. R. Nielson and F. Nielson. Data flow analysis for CCS. In *Program Analysis and Compilation. Theory and Practice*, volume 4444 of *Lecture Notes in Computer Science*. Springer, 2007.
19. K. V. S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25(2-3):285–327, 1995.
20. M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. In *Principles of Programming Languages (POPL'99)*, pages 105–118. ACM, 1999.