

# Modal Abstractions of Concurrent Behaviour

FLEMMING NIELSON

Technical University of Denmark

SEBASTIAN NANZ

ETH Zurich

and

HANNE RIIS NIELSON

Technical University of Denmark

---

We present an effective algorithm for the automatic construction of finite modal transition systems as abstractions of potentially infinite concurrent processes. Modal transition systems are recognised as valuable abstractions for model checking because they allow for the validation as well as refutation of safety and liveness properties. However, the algorithmic construction of finite abstractions from potentially infinite concurrent processes is a missing link that prevents their more widespread usage for model checking of concurrent systems. Our algorithm is a worklist algorithm using concepts from abstract interpretation and operating upon mappings from sets to intervals in order to express simultaneous over- and underapproximations of the multisets of process actions available in a particular state. We obtain a finite abstraction that is 3-valued in both states and transitions and that supports the definition of a 3-valued modal logic for validating as well as refuting properties of systems. The construction is illustrated on a few examples, including the Ingemarsson-Tang-Wong key agreement protocol.

Categories and Subject Descriptors: D.1.3 [Programming Techniques]: Concurrent Programming – *Distributed programming*; F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs – *Mechanical verification*; F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages – *Program analysis*; F.4.1 [Mathematical Logic and Formal Languages]: Mathematical Logic – *Temporal logic*

General Terms: Concurrency, Abstraction

Additional Key Words and Phrases: Static analysis, Monotone frameworks, Over- and underapproximations, Abstract Interpretation, Modal transition systems, Labelled transition systems, Process calculi, CCS, 3-valued logic, Temporal logic

---

## 1. INTRODUCTION

Conventional state transition systems consist of states and transitions between them and are intended to describe the behaviour of given systems presented in some syntactic manner. Model checking amounts to validating or refuting given safety

---

Author's addresses: F. Nielson, H. Riis Nielson, DTU Informatics, Technical University of Denmark, Building 321, DK-2800 Kongens Lyngby, Denmark; email: {nielson,riis}@imm.dtu.dk. S. Nanz, Department of Computer Science, ETH Zurich, Clausiusstr. 59, CH-8092 Zurich, Switzerland; email: nanz@inf.ethz.ch.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 1529-3785/20YY/0700-0001 \$5.00

or liveness properties on the state transition systems by means of giving a classical Boolean interpretation of logical formulae over transition systems.

Property-preserving abstractions are the topic of intensive research in model checking, as they provide successful techniques for fighting the state-explosion problem. State-explosion arise because the state transition system is typically exponentially larger than the syntactic presentation of the concurrent system; in the presence of recursion the state transition system may even be infinite despite the finiteness of the syntactic presentation. This problem seems intrinsic in the verification of a concurrent system where the correctness statement is often given by a mix of safety and liveness properties. Abstractions based on overapproximation, i.e. by adding more transition paths [Clarke et al. 1994], have proved useful for the validation of properties. However, while certain error traces can be produced (e.g. in the form of a “lasso”) they may turn out to be spurious and hence abstraction based on overapproximation alone fails to offer a satisfactory treatment of the refutation of properties.

*Modal transition systems* [Larsen and Thomsen 1988] use instead two kinds of transition relations, one representing necessary behaviour (“*must*”) and the other possible behaviour (“*may*”), while the absence of any transition describes behaviour that cannot possibly occur. Using modal transition systems as abstractions therefore allows for the validation as well as refutation of safety and liveness properties. Model checking then amounts to giving a 3-valued interpretation of logical formulae over modal transition systems [Godefroid et al. 2001] using Kleene’s 3-valued logic.

In modal transition systems the presence of necessary behaviour (“*must*”) always implies the presence of the similar possible behaviour (“*may*”). This is not so in the related notion of *mixed transition systems* [Dams et al. 1997; Gurfinkel et al. 2006] where there is the fourth possibility of conflicting behaviour. Correspondingly a property can be both validated and refuted. Model checking then amounts to giving a 4-valued interpretation of logical formulae over modal transition systems [Chechik et al. 2003] using Belnap’s 4-valued logic. For our purposes we find it more useful to stay within the 3-valued approach and to solve (in Section 2) any technical challenges this decision might pose.

The benefits of using modal or mixed abstractions are particularly relevant for the verification of concurrent systems, as these are frequently infinite-state. Existing techniques for automatic generation of modal abstractions [Godefroid et al. 2001; Gurfinkel and Chechik 2006] mainly apply to classical programming languages, and not to process calculi, one of the most widely used specification formalisms for concurrent systems. Developing new techniques for automatic generation of modal abstractions is likely to enable abstraction-based verification frameworks for communication protocols and service-oriented systems, where a large community is working with process calculus specifications. The construction is illustrated on a few examples, including the Ingemarsson-Tang-Wong (ITW) key agreement protocol, in order to show how to adapt the amount of precision produced and to demonstrate our ability to deal precisely with “*must*” transitions.

## 1.1 Contribution

This paper develops an algorithm for constructing finite modal transition systems as abstractions of processes specified in Milner’s *Calculus of Communicating Systems*

(*CCS*) [Milner 1989] which is sufficiently powerful that infinite transition systems can be described. We choose *CCS* as the basis of the description of our algorithm because, as a seminal process calculus, it allows us to focus on the essentials of our technique, and thus may facilitate the technique’s adaption to more expressive formalisms such as the  $\pi$ -calculus [Milner 1999] and other concurrent languages.

Our approach is rooted in a generalisation of *Monotone Frameworks* [Kam and Ullman 1977], which have been used in classical data flow analysis and comprise a transfer function

$$\text{transfer}_\ell(\hat{s}) = (\hat{s} \setminus \text{kill}_\ell) \cup \text{gen}_\ell$$

yielding the analysis information that holds at the next program point, provided that the information  $\hat{s}$  holds for the current program point  $\ell$ , and where  $\text{kill}_\ell$  is the information invalidated, and  $\text{gen}_\ell$  the new information created at  $\ell$ . While the classical analyses require an imperative program’s control flow graph as an input, our algorithm *constructs* a concurrent system’s *abstract* control flow graph besides the associated analysis information. As evidenced by the many papers studying the construction of so-called OCFA analyses (corresponding to the construction of a flow graph) for process calculi this is in itself non-trivial (see for example [Bodei et al. 1998; Nielson and Nielson 2002; Nielson et al. 2000; Buchholtz et al. 2004; Nielson et al. 2004]). Our generalisation of Monotone Frameworks takes account of multisets of actions exposed (rather than just sets of values); this is essential in order for the abstraction to accurately model the differences between parallel composition and non-deterministic choice as in the former case sibling actions are not removed when an action is performed whereas in the latter case they are.

Previous developments along these lines can be found in [Nielson and Nielson 2007a; 2009; 2007b; Pilegaard et al. 2008; Nanz et al. 2007] that only deal with over-approximations of the multiplicities. Indeed, [Nielson and Nielson 2007a] contained the first such development for *CCS* with a full version appearing in [Nielson and Nielson 2009]. In [Nielson and Nielson 2007b] the ideas were shown to be robust for applications to computer security expressed in the  $\pi$ -calculus, in [Pilegaard et al. 2008] the ideas were applied to the analysis of biological pathway systems expressed in BioAmbients, and in [Nanz et al. 2007] the ideas were developed for a distributed broadcast calculus. As already explained above the restriction to over-approximation goes a long way in validating properties of interest but falls short on refuting those that do not hold.

The *main contribution of this paper* (the full version of [Nanz et al. 2008]) is to extend the above developments by using intervals to give simultaneous lower as well as upper bounds on the multiplicities of actions which can execute in a certain abstract state and hence to construct a modal transition system. This development is performed in Sections 3–7. In Section 3 we recapitulate the syntax and operational semantics of *CCS* and define the labelled transition system forming the basis for our subsequent work. In Section 4 we construct the abstract domain to be used for our modal transition systems and we define the representation function used for mapping *CCS* into the modal transition systems. Sections 5 and 6 construct the *kill* and *gen* functions to be used in our adaptation of the Monotone Frameworks; in Section 5 we focus on the construction of “may” transitions and in Section 6 on the construction of “must” transitions. This development is performed directly

on the syntax of processes in CCS; this enables us to construct a finite modal transition system from a CCS process calculus term even though it describes an infinite concrete transition system. We then develop the worklist based algorithm implementing the Monotone Frameworks analysis in Section 7 using concepts from abstract interpretation; in particular, we use widenings and our notion of “granularity functions” to ensure that our algorithm always terminates with a finite modal transition system. We have implemented the algorithm in OCaml, and applied it to a few examples, in order to show how we can adapt the amount of detail produced and that we can provide useful information about “*must*” transitions.

An *additional contribution* is to embed our development into a general description of (concrete and modal) transition systems suitable for our development. This is the topic of Section 2 where we give an overview of our partially ordered approach to labelled transition systems and modal transition systems and relate it to the existing approaches of modal transition systems and mixed transition systems.

Another *additional contribution* is to show how our abstraction techniques link into existing approaches to model checking modal structures, e.g. [Shoham and Grumberg 2007; Grumberg et al. 2007; Bruns and Godefroid 2000]. This is the topic of Section 8 where we show how to use modal transition systems (as constructed in the paper) as the basis for interpreting a fragment of Action Computation Tree Logic (*ACTL\**) over Kleene’s 3-valued logic; in particular, we show in an example how our methods for adapting the amount of detail produced by our algorithm directly influences the evaluation of 3-valued modal formulae.

We conclude in Section 9.

## 1.2 Related Work

Modal transition systems have been introduced in [Larsen and Thomsen 1988] and require *must* actions also to be *may* actions. Mixed transition systems do not impose this constraint and have been considered in [Dams et al. 1997; Gurfinkel et al. 2006]. More general modal structures have been considered in [Larsen and Xinxin 1990].

The work in this paper is based on the original model [Larsen and Thomsen 1988] and is motivated in part by the use of 3-valued abstraction techniques in static analysis [Sagiv et al. 1999; 2002; Nielson et al. 2000; 2001].

Ideas from Abstract Interpretation [Cousot and Cousot 1977] have been used to abstract a concrete transition into a mixed transition system [Dams et al. 1997; Gurfinkel et al. 2006; Schmidt 2007] or modal transition system [Godefroid et al. 2001]. More general modal simulations have been considered in [Dams and Namjoshi 2004] and a proposal for using tree automata as unifying abstractions may be found in [Dams and Namjoshi 2005]. The importance of modal transition systems and related structures has since been emphasised in the context of program analysis and model checking [Bruns and Godefroid 1999; Huth et al. 2001; Chechik et al. 2003]. Besides Abstract Interpretation techniques to construct abstract systems from concrete ones [Dams et al. 1997], also predicate abstraction [Graf and Saïdi 1997] has been used [Ball et al. 2001; Godefroid et al. 2001; Gurfinkel and Chechik 2006; Fecher and Shoham 2007].

Our work (in Sections 3–7) focuses on constructing a finite modal transition system from a necessarily finite term in a process calculus; this is preferable to

working from the concrete transition system as the latter might be infinite. Previous approaches with a similar aim [Dams et al. 1997; Ball et al. 2001; Gurfinkel and Chechik 2006] consider the considerably simpler setting of action systems, flow chart programs or C programs whereas we deal with the challenges arising from synchronisation in the concurrent process calculus CCS [Milner 1989]. To show the relationship with the approaches mentioned above we embed our results into a general description of abstracting labelled transition systems in Section 2. To be concrete, our approach is based on Monotone Frameworks [Nielson et al. 1999], a well-known data flow analysis technique for flow chart programs.

A number of papers address model checking over modal structures. In [Shoham and Grumberg 2007] a game-based CTL model checking over Kripke modal transition systems is defined. This approach is extended to the full  $\mu$ -calculus in [Grumberg et al. 2007] and as in [Bruns and Godefroid 2000] performs a reduction from multi-valued (3 or 4) a reduction to 2-valued problems.

We believe that our abstraction generation technique is suitable for any of these approaches. While a full development of model checking is beyond the scope of the present paper we do define the satisfaction relation for a fragment of the Action Computation Tree Logic ( $ACTL^*$ ) [De Nicola and Vaandrager 1990] in Section 8 in order to substantiate this claim.

We should mention that simultaneous over- and underapproximations are also used in other application areas, as for example in [Baldan et al. 2008] to analyse infinite-state graph transformation systems. In the realm of stochastic systems, interval abstractions for discrete-time Markov chains are used in [Jonsson and Larsen 1991; Huth 2005; Katoen et al. 2008; 2007].

## 2. MODAL TRANSITION SYSTEMS

In this section we provide the main definitions of concrete labelled transition systems and abstract modal transition systems within which our development is performed. This includes defining the notion of representation and refinement between transition systems. We indicate when and why we differ from the standard approaches of modal transition systems [Larsen and Thomsen 1988] and mixed transition systems [Dams et al. 1997; Gurfinkel et al. 2006].

We assume that the concrete model of a system is given by a *labelled transition system* of the form  $T = (S, A, \rightarrow)$ , where  $S$  is a non-empty set of *states*,  $A$  is a non-empty set of *actions*, and  $\rightarrow \subseteq S \times A \times S$  is a *transition relation*. A transition describes the state change that takes place when a certain action is performed. Usually, a labelled transition system is implicitly defined using language-based descriptions. For example, a process specified in a process calculus together with the operational semantics of the process calculus gives rise to a labelled transition system with the process as starting state; this is indeed the approach we shall explore in Section 3.

Concrete models may have an infinite or large finite state space which prevents efficient reasoning about the system. One therefore strives to abstract the concrete transition system in order to finitise it or reduce the size of its state space. Such abstract transition systems consist of abstract states, each of which represents a set of concrete states. In order to obtain an abstract transition relation, we can repre-

sent either *possible* or *definite* transitions in the concrete model, which correspond to *over-* and *underapproximations* of the concrete transition relation. Using the approach of *modal transition systems* [Larsen and Thomsen 1988] we use “*may*” transitions for the *possible* transitions and “*must*” transitions for the *definite* transitions. Indeed, the fact that “*must*” transitions imply “*may*” transitions is very natural for this point of view and the reason why we do not use the approach of mixed transition systems [Dams et al. 1997; Gurfinkel et al. 2006].

A modal transition system is of the form  $\hat{T} = (\hat{S}, A, \dashrightarrow, \longrightarrow)$  where  $\hat{S}$  is a non-empty set of *abstract states* equipped with a partial order  $\sqsubseteq$ , and where  $A$  is a non-empty set of *actions*,  $\dashrightarrow \subseteq \hat{S} \times A \times \hat{S}$  is a transition relation representing possible transitions, called “*may*”, and  $\longrightarrow \subseteq \dashrightarrow$  is a transition relation representing definite transitions, called “*must*”. The condition  $\longrightarrow \subseteq \dashrightarrow$  is imposed as motivated above and as done in [Larsen and Thomsen 1988].

Our definition differs from the one in [Larsen and Thomsen 1988] by having the set of abstract states partially ordered. This is a natural requirement when viewing modal transition systems as abstractions (rather than specifications), a setting in which the set of abstract states can be viewed as a property space where it is useful to have a partial order imposed.

For a single concrete model, we might be interested in several abstractions which raises the question of comparing abstract models. For this it is useful to consider an *abstract domain*  $\mathcal{U}$  having the set of abstract states  $\hat{S}$  as a subset; we shall assume that the partial orders of  $\hat{S}$  and  $\mathcal{U}$  agree on their common elements and write  $(\hat{S}, A, \dashrightarrow, \longrightarrow) : \mathcal{U}$  to indicate this.

Consider next two models  $(\hat{S}_i, A, \dashrightarrow^i, \longrightarrow^i) : \mathcal{U}_i$  for  $i = 1, 2$ . Our first task is to introduce a monotonic function  $h : \mathcal{U}_1 \rightarrow \mathcal{U}_2$  for transforming one abstract domain into another. Clearly  $h : \mathcal{U}_1 \rightarrow \mathcal{U}_2$  specialises to  $h : \hat{S}_1 \rightarrow \hat{S}_2$ , but we do *not* impose that  $h : \hat{S}_1 \rightarrow \hat{S}_2$ , i.e. we do *not* impose that the image  $h(\hat{S}_1) = \{h(\hat{s}) \mid \hat{s} \in \hat{S}_1\}$  is a subset of  $\hat{S}_2$ . We shall write  $h : (\hat{S}_1, A, \dashrightarrow^1, \longrightarrow^1) \rightarrow (\hat{S}_2, A, \dashrightarrow^2, \longrightarrow^2)$  whenever  $h(\hat{S}_1) \subseteq \hat{S}_2$  is the case. The main advantage of having introduced the abstract domain  $\mathcal{U}_2$  is that we can discuss the choice of the subset  $\hat{S}_2$ .

We can then define a notion of refinement under this function, along the lines of refinement as defined in [Larsen and Thomsen 1988].

*Definition 2.1 Refinement.* Let  $\mathcal{U}_1$  and  $\mathcal{U}_2$  be two abstract domains, and let  $h : \mathcal{U}_1 \rightarrow \mathcal{U}_2$  be a monotonic function. Furthermore, let  $\hat{T}_1 = (\hat{S}_1, A, \dashrightarrow^1, \longrightarrow^1) : \mathcal{U}_1$  and  $\hat{T}_2 = (\hat{S}_2, A, \dashrightarrow^2, \longrightarrow^2) : \mathcal{U}_2$  be two modal transition systems sharing a set of actions  $A$ . We say that  $\hat{T}_1$  *h-refines*  $\hat{T}_2$ , written  $\hat{T}_1 \blacktriangleleft_h \hat{T}_2$ , iff for all  $\ell \in A$ , and  $\hat{s}_1 \in \hat{S}_1$ ,  $\hat{s}_2 \in \hat{S}_2$  with  $h(\hat{s}_1) \sqsubseteq \hat{s}_2$  the following implications hold:

- (1) If  $\hat{s}_1 \dashrightarrow_\ell^1 \hat{s}'_1$  then there exists  $\hat{s}'_2 \in \hat{S}_2$  such that  $\hat{s}_2 \dashrightarrow_\ell^2 \hat{s}'_2$  and  $h(\hat{s}'_1) \sqsubseteq \hat{s}'_2$ .
- (2) If  $\hat{s}_2 \longrightarrow_\ell^2 \hat{s}'_2$  then there exists  $\hat{s}'_1 \in \hat{S}_1$  such that  $\hat{s}_1 \longrightarrow_\ell^1 \hat{s}'_1$  and  $h(\hat{s}'_1) \sqsubseteq \hat{s}'_2$ .

For the identity function *id*, it turns out that *id*-refinement is not necessarily reflexive nor transitive (but see Lemma 2.3 and Definition 2.4).

Clearly the most general way to relate two transition systems is by means of a general relation as in [Larsen and Thomsen 1988, Definition 2.1], [Dams et al. 1997, Definition 2.4.2] or [Godefroid et al. 2001, Definition 6]. We have restricted ourselves to consider only relations of the form  $\blacktriangleleft_h$  where indeed the function  $h$  can

be considered as an abstraction function (usually denoted  $\alpha$ ) in the framework of Abstract Interpretation [Cousot and Cousot 1979].

We can now formalise the notion that a labelled transition system  $T$  is abstracted by a modal transition system. The abstraction is induced by a representation function  $\beta : S \rightarrow \mathcal{U}$  which maps a concrete state  $s \in S$  into the abstract state  $\hat{s} \in \mathcal{U}$  chosen for representing it. In general, we say that a concrete state  $s$  is  $\beta$ -represented by an abstract state  $\hat{s}$  iff  $\beta(s) \sqsubseteq \hat{s}$ . This definition can be lifted to transition systems by using the fact that  $T$  can itself be seen as a modal transition system, namely the one where both “*may*”- and “*must*”-relations equal its transition relation, and where the partial order on the states is equality.

*Definition 2.2 Representation.* Let  $T = (S, A, \rightarrow)$  be a labelled transition system and  $\hat{T} = (\hat{S}, A, \dashrightarrow, \longrightarrow) : \mathcal{U}$  be a modal transition system, and let  $\beta : S \rightarrow \mathcal{U}$  be a representation function. We say that  $T$  is  $\beta$ -represented by  $\hat{T}$  iff  $(S, A, \rightarrow, \rightarrow) \triangleleft_{\beta} \hat{T}$ .

It may be useful to write  $\langle (S, A, \rightarrow) \rangle$  for  $(S, A, \rightarrow, \rightarrow)$  so that  $T$  is  $\beta$ -represented by  $\hat{T}$  can be written  $\langle T \rangle \triangleleft_{\beta} \hat{T}$ .

Representation functions are usually chosen so as to formalise the desired degree of approximation. In the main part of the paper, we fix such a suitable representation function  $\beta$  and describe a method for finding an abstraction  $\hat{T}$  that  $\beta$ -represents a given transition system  $T$ . Most importantly, the method works directly on the terms of a process calculus. We next establish some useful properties of refinement.

LEMMA 2.3. *Refinement is transitive, in the sense that  $\hat{T}_1 \triangleleft_h \hat{T}_2 \triangleleft_g \hat{T}_3$  implies that  $\hat{T}_1 \triangleleft_{g \circ h} \hat{T}_3$  provided that  $h : \hat{T}_1 \rightarrow \hat{T}_2$ .*

PROOF. Assume  $\hat{s}_1 \dashrightarrow_{\ell} \hat{s}'_1$  and  $g(h(\hat{s}_1)) \sqsubseteq \hat{s}_3$ . Writing  $\hat{s}_2$  for  $h(\hat{s}_1)$  (which is an element of  $\hat{S}_2$ ), we have  $h(\hat{s}_1) \sqsubseteq \hat{s}_2$  and  $g(\hat{s}_2) \sqsubseteq \hat{s}_3$ . Using the first premise we have that there exists  $\hat{s}_2 \dashrightarrow_{\ell} \hat{s}'_2$  and  $h(\hat{s}'_1) \sqsubseteq \hat{s}'_2$ . Using the second premise we have that there exists  $\hat{s}_3 \dashrightarrow_{\ell} \hat{s}'_3$  and  $g(\hat{s}'_2) \sqsubseteq \hat{s}'_3$ . Since  $g$  is monotonic, we have  $g(h(\hat{s}'_1)) \sqsubseteq g(\hat{s}'_2)$ . Transitivity,  $g(h(\hat{s}'_1)) \sqsubseteq \hat{s}'_3$ .

Assume  $\hat{s}_3 \dashrightarrow_{\ell} \hat{s}'_3$  and  $g(h(\hat{s}_1)) \sqsubseteq \hat{s}_3$ . Writing  $\hat{s}_2$  for  $h(\hat{s}_1)$  (which is an element of  $\hat{S}_2$ ), we have  $h(\hat{s}_1) \sqsubseteq \hat{s}_2$  and  $g(\hat{s}_2) \sqsubseteq \hat{s}_3$ . Using the second premise we have that there exists  $\hat{s}_2 \dashrightarrow_{\ell} \hat{s}'_2$  and  $g(\hat{s}'_2) \sqsubseteq \hat{s}'_3$ . Using the first premise we have that there exists  $\hat{s}_1 \dashrightarrow_{\ell} \hat{s}'_1$  and  $h(\hat{s}'_1) \sqsubseteq \hat{s}'_2$ . Since  $g$  is monotonic, we have  $g(h(\hat{s}'_1)) \sqsubseteq g(\hat{s}'_2)$ . Transitivity,  $g(h(\hat{s}'_1)) \sqsubseteq \hat{s}'_3$ .  $\square$

*Definition 2.4.* A modal transition system  $\hat{T}$  is *well-defined* whenever  $\hat{T} \triangleleft_{id} \hat{T}$ .

This expresses that refinement is reflexive and automatically holds in the special case where the partial order  $\sqsubseteq$  is equality.

### 3. COMMUNICATING SYSTEMS AS CONCRETE PROCESSES

We now embark on the main contribution of this paper: the effective construction of a finite modal transition system even in the case where the concrete transition system is itself infinite.

In this section we give a brief review of the syntax and operational semantics of Milner’s *Calculus of Communicating Systems (CCS)* [Milner 1989] that will serve as basis for defining both the (potentially infinite) concrete transition system (in

$$\begin{array}{c}
\tau^\ell.P + Q \rightarrow_\ell P \quad (\bar{x}^{\ell_1}.P_1 + Q_1) | (x^{\ell_2}.P_2 + Q_2) \rightarrow_{\ell_2 \ell_1} P_1 | P_2 \\
\frac{P \rightarrow_{\bar{\ell}} P'}{P | Q \rightarrow_{\bar{\ell}} P' | Q} \quad \frac{P \rightarrow_{\bar{\ell}} P'}{\text{new } x P \rightarrow_{\bar{\ell}} \text{new } x P'} \quad \frac{P' \rightarrow_{\bar{\ell}} Q'}{P \rightarrow_{\bar{\ell}} Q} \quad \text{if } P \equiv P' \text{ and } Q' \equiv Q
\end{array}$$

Table I. Reaction semantics  $P \rightarrow_{\bar{\ell}} Q$  for  $\text{let } A_1 \triangleq P_1; \dots; A_k \triangleq P_k \text{ in } P_0$ 

this section) as well as for the construction of an abstract modal transition system (in the following sections).

### 3.1 Syntax

The syntax of CCS *processes*  $P \in \mathbf{Proc}$  and *actions*  $\alpha$  is given by the following definition, where we presuppose that *names*  $x$  can be drawn from some countably infinite set.

$$\begin{array}{l}
P ::= \text{new } x P \mid P_1 \mid P_2 \mid \Sigma_{i \in I} \alpha_i^{\ell_i}.P_i \mid A \\
\alpha ::= \bar{x} \mid x \mid \tau
\end{array}$$

Here  $\text{new } x P$  introduces a new name  $x$  with scope  $P$ , and parallel composition is modelled using the construct  $P_1 \mid P_2$ . Non-deterministic choice is expressed using guarded summations of the form  $\Sigma_{i \in I} \alpha_i^{\ell_i}.P_i$  where  $I$  is a finite index set; if  $I = \emptyset$  we write  $\mathbf{0}$  for the summation, if  $I$  is a singleton set we obtain prefixing  $\alpha^\ell.P$ , and when  $I$  is binary we allow to write  $\alpha_1^{\ell_1}.P_1 + \alpha_2^{\ell_2}.P_2$ . Actions  $\alpha$  are annotated with labels  $\ell \in \mathbf{Lab}$  serving as pointers into the process; they will be used in the static analysis but have no semantic significance. Complementary actions take the form  $\bar{x}$  and  $x$  where  $x$  is a name and are used to perform the synchronisation of two processes over the name  $x$ ; internal actions are denoted by  $\tau$  and are also used to hide synchronisations from outside observers as will become clear when defining the semantics. We shall be interested in *programs* of the form

$$\text{let } A_1 \triangleq P_1; \dots; A_k \triangleq P_k \text{ in } P_0 \quad (1)$$

where the processes named  $A_1, \dots, A_k (\in \mathbf{PN})$  are mutually recursively defined and may be used in the main process  $P_0$  as well as in the process bodies  $P_1, \dots, P_k$ .

*Example 3.1.* We introduce a simple running example to illustrate the technical developments throughout the paper. Consider the following program called  $P$ :

$$\text{let } S \triangleq a^1.r^2.S; \quad Q \triangleq \bar{a}^3.(\bar{r}^4.Q + \tau^5.\bar{r}^6.Q + \tau^5.Q) \text{ in } S \mid Q \mid Q$$

The process  $S$  models a semaphore which allows for two actions *acquire* and *release*, and then starts over. The process  $Q$  tries to acquire the lock and may then do one of three actions before recursing: release the lock immediately; perform an internal action and release the lock; and, perform an internal action but not release the lock.

### 3.2 Operational Semantics

Following Chapter 4 of [Milner 1999], we equip the calculus with a reaction semantics and a structural congruence. This style of operational semantics focuses on the interactions between components of a system, and is thus for example well

suitable for describing the behaviour of protocols, one of the prototypical application domains for our abstraction method.<sup>1</sup>

The *reaction relation*  $P \rightarrow_{\tilde{\ell}} Q$  is specified in Table I and expresses that the process  $P$  may evolve in one step into the process  $Q$ ; we have annotated the arrow with the labels  $\tilde{\ell}$  of the actions involved as this will prove useful when expressing the correctness of the analysis. Note that we use  $\tilde{\ell}$  to denote both single labels  $\ell$  and label pairs  $\ell_2\ell_1$ . The reaction relation naturally extends to programs in that only the main process can evolve.

The *structural congruence*  $P \equiv Q$  is defined as the least congruence generated from the axioms of Table II. We require additionally that processes are congruent if they can be obtained from each other by *disciplined alpha-renaming*; this merely amounts to requiring that each name  $x$  has a *canonical name*  $[x]$  that is preserved by alpha-renaming. This canonical name stands as a representative for the equivalence class of its alpha-renamings, and is essential for interpreting the analysis result. Canonical names are only used in the analysis and its related correctness proofs. The condition that  $x \notin \text{fn}(P)$  means that  $x$  does not occur as a free name in  $P$ .

*Example 3.2.* Using the formal semantics we can express steps of the evolution of the main process  $S \mid Q \mid Q$  of Example 3.1 as follows:

$$S \mid Q \mid Q \rightarrow_{13} r^2.S \mid (\bar{r}^4.Q + \tau^5.\bar{r}^6.Q + \tau^5.Q) \mid Q \rightarrow_5 r^2.S \mid \bar{r}^6.Q \mid Q$$

The notion of canonical names is lifted to actions so  $[\alpha]$  will be the *canonical action* corresponding to  $\alpha$ . A program is *consistently labelled* if all occurrences of actions  $\alpha_1^\ell$  and  $\alpha_2^\ell$  with the same label  $\ell$  have the same canonical action, that is  $[\alpha_1] = [\alpha_2]$ ; we shall then write  $\partial(\ell)$  for this canonical action. A program is *uniquely labelled* if all occurrences of actions have distinct labels; clearly, a uniquely labelled program is consistently labelled.

*Example 3.3.* The program of Example 3.1 is not uniquely labelled because there are two distinct syntactic occurrences of  $\tau^5$ . The program is consistently labelled because  $[\tau^5] = [\tau^5]$ .

It is easy to see that the property of being consistently labelled is invariant under the structural congruence and that it is preserved by the reaction semantics; this does not hold for the property of being uniquely labelled. In the following we shall assume to have consistently labelled programs; clearly, such labellings can easily be automatically obtained.

### 3.3 Labelled Transition System

Linking back to the overview of our development in Section 2, the labelled transition system  $T_0 = (S, \mathbf{ALL}, \rightarrow)$  of a program let  $A_1 \triangleq P_1; \dots; A_k \triangleq P_k$  in  $P_0$  can be formally obtained as follows: the set of states  $S$  is given by the set of all processes derivable from the starting state  $P_0$ , i.e.  $\{P \mid P_0 \rightarrow^* P\}$  where  $\rightarrow^*$  is the transitive

<sup>1</sup>However, a labelled transition semantics, where processes can also proceed by independent observable actions, could likewise be used (with straightforward changes in the respective theorems in the following sections) because the abstraction method we will describe works directly on the standard process syntax presented in Subsection 3.1.

$$\begin{array}{l}
P \mid Q \equiv Q \mid P \quad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \quad P \mid 0 \equiv P \\
\text{new } x \, 0 \equiv 0 \quad \text{new } x \, \text{new } y \, P \equiv \text{new } y \, \text{new } x \, P \\
\text{new } x \, (P \mid Q) \equiv P \mid \text{new } x \, Q \text{ if } x \notin \text{fn}(P) \quad A \equiv P \text{ if } A \triangleq P
\end{array}$$

Table II. Axioms generating the structural congruence  $P \equiv Q$  for let  $A_1 \triangleq P_1; \dots; A_k \triangleq P_k$  in  $P_0$ 

closure of  $\rightarrow$ , **ALL** is given by the set  $\mathbf{Lab} \cup (\mathbf{Lab} \times \mathbf{Lab})$  consisting of all single labels and label pairs, and  $\rightarrow$  is defined by Tables I and II.

#### 4. THE ABSTRACT REPRESENTATION OF CONCRETE PROCESSES

In the previous section we have shown how CCS processes give rise to concrete transition systems and in Sections 4–7 we focus on the construction of abstract modal transition systems.

As a preparation we review in Subsection 4.1 the basic information we want for individual processes [Nielson and Nielson 2007a; 2009]. This amounts to obtaining multisets indicating those actions (and their multiplicity) that might have a chance to execute in the next reaction step – consequently these actions are said to be exposed.

In Subsection 4.2 we then determine the abstract domain over which we should like to consider abstract modal transition systems for CCS; basically this is an abstract domain that maps labels to intervals containing lower and upper bounds of the number of exposed actions.

This also involves finding a suitable representation function for mapping CCS processes into sets of abstract states in Subsection 4.3. For the main process  $S \mid Q \mid Q$  of Example 3.1 that has one occurrence of  $a^1$  and two occurrences of  $\bar{a}^3$  as exposed actions we shall want a mapping  $\beta(S \mid Q \mid Q)$  that maps the label **1** to the interval  $[1, 1]$ , the label **3** to the interval  $[2, 2]$  and all other labels to the interval  $[0, 0]$ .

##### 4.1 Exposed Actions

As in [Nielson and Nielson 2007a; 2009] an *exposed action* is an action that *might* participate in the next reaction step. For example, the main process  $S \mid Q \mid Q$  of Example 3.1 has one occurrence of  $a^1$  and two occurrences of  $\bar{a}^3$  as exposed actions. We shall say that an action involving synchronisation (e.g.  $x^i$ ) is exposed regardless of whether its counterpart (e.g.  $\bar{x}^j$ ) is exposed<sup>2</sup> or not. In general, a process may contain many occurrences of the same action (all identified by the same label) and it may be the case that several of them are ready to participate in the next reaction step; in fact due to the semantic treatment of recursion by arbitrary unfolding, even infinitely many occurrences may be exposed.

Let  $\mathbb{N}$  be the set of natural numbers including 0. To capture exposed actions we use *extended multisets*  $M \in \mathfrak{M}$  where

$$\mathfrak{M} = \mathbf{Lab} \rightarrow (\mathbb{N} \cup \{\infty\})$$

and the idea is that  $M(\ell)$  records the number of occurrences of the label  $\ell$ ; there may be a finite number in which case  $M(\ell) \in \mathbb{N}$  or an infinite number so that

<sup>2</sup>This will be taken into account when studying enabled actions in Subsection 6.2.

$$\begin{aligned}
 \mathcal{E}[\text{new } x P]env &= \mathcal{E}[P]env \\
 \mathcal{E}[P \mid Q]env &= \mathcal{E}[P]env +_{\mathfrak{M}} \mathcal{E}[Q]env \\
 \mathcal{E}[\sum_{i \in I} \alpha_i^{\ell_i} . P_i]env &= \sum_{\mathfrak{M} \ i \in I} \perp_{\mathfrak{M}} [\ell_i \mapsto 1] \\
 \mathcal{E}[A]env &= env(A) \\
 \mathcal{E}_*[P] &= \mathcal{E}[P]env_{\mathcal{E}} \\
 \text{where } \mathcal{F}_{\mathcal{E}}(env) &= [A_1 \mapsto \mathcal{E}[P_1]env, \dots, A_k \mapsto \mathcal{E}[P_k]env] \\
 \text{and } env_{\perp_{\mathfrak{M}}} &= [A_1 \mapsto \perp_{\mathfrak{M}}, \dots, A_k \mapsto \perp_{\mathfrak{M}}] \\
 \text{and } env_{\mathcal{E}} &= \bigsqcup_{j \geq 0} \mathcal{F}_{\mathcal{E}}^j(env_{\perp_{\mathfrak{M}}})
 \end{aligned}$$

 Table III. Exposed actions for let  $A_1 \triangleq P_1; \dots; A_k \triangleq P_k$  in  $P_0$ 

$M(\ell) = \infty$ . We say that a label  $\ell$  is in the domain of  $M$ , written  $\ell \in \text{dom}(M)$ , if  $M(\ell) \neq 0$ .

The exposed actions of a process can be computed by an observation function

$$\mathcal{E}_* : \mathbf{Proc} \rightarrow \mathfrak{M}.$$

As an example, let us reconsider the main process  $S \mid Q \mid Q$  of the program of Example 3.1. Here we want to obtain  $\mathcal{E}_*(S \mid Q \mid Q) = \perp_{\mathfrak{M}}[1 \mapsto 1, 3 \mapsto 2]$  which is the function that maps the label 1 to the number 1, the label 3 to the number 2, and all other labels to the number 0.

To prepare for the formal definition of  $\mathcal{E}_*$  we shall equip the set  $\mathfrak{M}$  with a partial ordering  $\leq_{\mathfrak{M}}$  defined by:

$$M \leq_{\mathfrak{M}} M' \quad \text{iff} \quad \forall \ell : M(\ell) \leq M'(\ell) \vee M'(\ell) = \infty$$

The domain  $(\mathfrak{M}, \leq_{\mathfrak{M}})$  is a complete lattice with bottom element  $\perp_{\mathfrak{M}} = \lambda \ell. 0$  and top element  $\top_{\mathfrak{M}} = \lambda \ell. \infty$ , and in addition to least upper and greatest lower bound operators, we shall need the operation  $+_{\mathfrak{M}}$ , which is an extension of ordinary integer addition with the rule that addition of  $\infty$  yields again  $\infty$ .

We are now ready to consider the definition of  $\mathcal{E}_*$ . We begin with the sum of two processes  $\alpha_1^{\ell_1} . P_1 + \alpha_2^{\ell_2} . P_2$ . Here both of the actions  $\alpha_1$  and  $\alpha_2$  are ready to interact but none of those of  $P_1$  and  $P_2$  are so we shall take:

$$\mathcal{E}_*[\alpha_1^{\ell_1} . P_1 + \alpha_2^{\ell_2} . P_2] = \perp_{\mathfrak{M}}[\ell_1 \mapsto 1] +_{\mathfrak{M}} \perp_{\mathfrak{M}}[\ell_2 \mapsto 1]$$

Note that if the two labels happened to be equal, the overall count would become 2 (as desired) since we have used pointwise addition  $+_{\mathfrak{M}}$ .

To handle the general case we shall introduce the function

$$\mathcal{E} : \mathbf{Proc} \rightarrow (\mathbf{PN} \rightarrow \mathfrak{M}) \rightarrow \mathfrak{M}$$

that as an additional parameter takes an environment holding the required information for the process names. The function is defined in Table III for arbitrary processes; in the case of sums it generalises the clause shown above. The clause for the  $\text{new } x P$  construct simply ignores the introduction of the new name and thereby its scope. For process names we simply consult the environment  $env$ .

This yields the functional  $\mathcal{F}_{\mathcal{E}} : (\mathbf{PN} \rightarrow \mathfrak{M}) \rightarrow (\mathbf{PN} \rightarrow \mathfrak{M})$  shown in Table III. Since the operations involved in its definition are all monotonic we have a monotonic functional defined on a complete lattice and Tarski's fixed point theorem ensures that it has a least fixed point which is denoted  $env_{\mathcal{E}}$  in Table III. Since all processes

are finite it follows that  $\mathcal{F}_{\mathcal{E}}$  is continuous and hence that the Kleene formulation of the least fixed point can be used; here  $\mathcal{F}_{\mathcal{E}}^j$  denotes the  $j$ -fold functional iteration of the function  $\mathcal{F}_{\mathcal{E}}$  so that we are taking the least upper bound of the chain  $env_{\perp_{\mathfrak{M}}}, \mathcal{F}_{\mathcal{E}}(env_{\perp_{\mathfrak{M}}}), \mathcal{F}_{\mathcal{E}}(\mathcal{F}_{\mathcal{E}}(env_{\perp_{\mathfrak{M}}})) , \dots , \mathcal{F}_{\mathcal{E}}^j(env_{\perp_{\mathfrak{M}}}), \dots$ . We can now define the function  $\mathcal{E}_{\star}$  simply as  $\mathcal{E}_{\star}[[P]] = \mathcal{E}[[P]]env_{\mathcal{E}}$ .

*Example 4.1.* For the running example of Example 3.1 we get:

$$\mathcal{E}_{\star}[[S \mid Q \mid Q]] = \perp_{\mathfrak{M}}[1 \mapsto 1, 3 \mapsto 2]$$

We may consider another process  $R \triangleq (\bar{a}^3.0 + \tau^5.\bar{r}^4.0) \mid R$  to illustrate that also an infinite number of actions can be exposed:

$$\mathcal{E}_{\star}[[S \mid R]] = \perp_{\mathfrak{M}}[1 \mapsto 1, 3 \mapsto \infty, 5 \mapsto \infty]$$

Note that the process  $S \mid R$  gives rise to an infinite transition system, which can be abstracted into a finite (modal) transition system using the technique to be presented in Sections 5–7.

The implementation of the function  $\mathcal{E}_{\star}$  requires a bit of care. It does not suffice to simply unfold the iterands  $\mathcal{F}_{\mathcal{E}}^j(env_{\perp_{\mathfrak{M}}})$  until stabilisation, i.e.  $\mathcal{F}_{\mathcal{E}}^j(env_{\perp_{\mathfrak{M}}}) = \mathcal{F}_{\mathcal{E}}^{j+1}(env_{\perp_{\mathfrak{M}}})$ , as this process need not terminate due to the existence of infinite ascending chains in  $(\mathfrak{M}, \leq_{\mathfrak{M}})$ . Instead we shall use the method of [Nielson and Nielson 2009]. First let  $k$  be given as the number of recursive definitions, i.e. we are considering a program of the form let  $A_1 \triangleq P_1; \dots; A_k \triangleq P_k$  in  $P_0$ . Next consider the iterands  $M_k = \mathcal{F}_{\mathcal{E}}^k(env_{\perp_{\mathfrak{M}}})$  and  $M_{2k} = \mathcal{F}_{\mathcal{E}}^{2k}(env_{\perp_{\mathfrak{M}}})$ . Then

$$env_{\mathcal{E}}(\ell) = \begin{cases} M_k(\ell) & \text{if } M_k(\ell) = M_{2k}(\ell) \\ \infty & \text{if } M_k(\ell) \neq M_{2k}(\ell) \end{cases}$$

follows from [Nielson and Nielson 2009, Lemma 12].

It is worthwhile stressing that the information provided by  $\mathcal{E}_{\star}$  is exact – there is no amount of approximation going on. The reason that numbers can become  $\infty$  is because of the way the unfolding of recursion is handled in Table II. Indeed, 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 reaction step:

LEMMA 4.2. *If  $P \equiv Q$  then  $\mathcal{E}_{\star}[[P]] = \mathcal{E}_{\star}[[Q]]$ , and if  $P \rightarrow_{\bar{i}} Q$  then  $\bar{i} \in \text{dom}(\mathcal{E}_{\star}[[P]])$ .*

PROOF. We prove  $\mathcal{E}[[P]]env_{\mathcal{E}} = \mathcal{E}[[Q]]env_{\mathcal{E}}$  by induction on how  $P \equiv Q$  was obtained from Table II. When no unfolding of recursion takes place it is straightforward to show that  $\mathcal{E}[[P]]env = \mathcal{E}[[Q]]env$  for all  $env$ . So consider  $A \equiv P$  where  $A$  is defined by  $A \triangleq P$ . We then have to show  $env_{\mathcal{E}}(A) = \mathcal{E}[[P]]env_{\mathcal{E}}$  which follows from  $env_{\mathcal{E}} = \bigsqcup_{j \geq 0} \mathcal{F}_{\mathcal{E}}^j(env_{\perp_{\mathfrak{M}}})$  and  $\mathcal{F}_{\mathcal{E}}(env)(A) = \mathcal{E}[[P]]env$ .

For the second part of the lemma we proceed by induction on the inference of  $P \rightarrow_{\bar{i}} Q$  as defined in Table I. The result is immediate for the two axioms and for the rule using the congruence we make use of the first part of the lemma. The remaining rules are straightforward applications of the induction hypothesis.  $\square$

The information provided by  $\mathcal{E}_{\star}$  may change as processes evolve because once an action has executed, some new actions may become exposed and some may

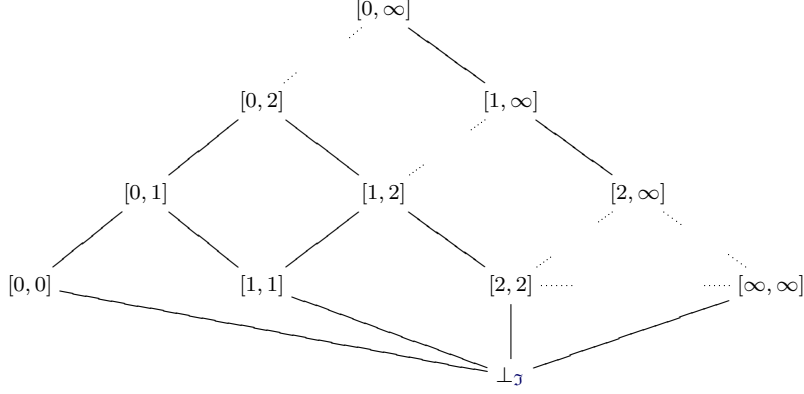


Fig. 1. The interval lattice

cease to be exposed. For example, in process  $S$  of Example 3.1 the action  $a^1$  is initially exposed but once it has been executed it will no longer be exposed (we say that it is *killed*) and instead the action  $r^2$  becomes exposed (we say that it is *generated*). It is not always possible to be precise about the number of actions that are killed or generated; for example, when executing the action  $\tau^5$  of  $Q$ , we cannot be sure whether  $\bar{r}^6$  or  $\bar{a}^3$  becomes exposed. We shall return to these considerations in Section 5 and this development will be facilitated by the abstract domain of Subsection 4.2.

#### 4.2 The Abstract Domain of Interval Lattices

To obtain a *simultaneous* description of over- and under-approximation we are interested in lifting the domain  $\mathbb{N} \cup \{\infty\}$  of the mapping  $M \in \mathfrak{M}$  to a richer domain. While saying earlier that a label  $\ell$  is exposed  $M(\ell) = k$  times in a process, we now want to express that  $\ell$  is exposed *at least*  $i$  times and *at most*  $j$  times, and the idea will be to use a new mapping  $N$  with  $N(\ell) = [i, j]$ . The lattice described below and depicted in Figure 1 provides the appropriate domain.

The elements of the lattice  $(\mathcal{J}, \leq_{\mathcal{J}})$  of intervals over  $\mathbb{N} \cup \{\infty\}$  are

$$\mathcal{J} = \{\perp_{\mathcal{J}}\} \cup \{[i, j] \mid i \leq j, i, j \in \mathbb{N} \cup \{\infty\}\}$$

where  $\perp_{\mathcal{J}}$  denotes the empty interval. The partial ordering  $\leq_{\mathcal{J}}$  is defined as

$$I_1 \leq_{\mathcal{J}} I_2 \text{ iff } \inf_{\mathcal{J}}(I_2) \leq \inf_{\mathcal{J}}(I_1) \wedge \sup_{\mathcal{J}}(I_1) \leq \sup_{\mathcal{J}}(I_2)$$

where the supremum and infimum operators on intervals are given by:

$$\inf_{\mathcal{J}}(I) = \begin{cases} i & \text{if } I = [i, j] \\ \infty & \text{if } I = \perp_{\mathcal{J}} \end{cases} \quad \sup_{\mathcal{J}}(I) = \begin{cases} j & \text{if } I = [i, j] \\ 0 & \text{if } I = \perp_{\mathcal{J}} \end{cases}$$

Indeed,  $(\mathcal{J}, \leq_{\mathcal{J}})$  is then a complete lattice with least element  $\perp_{\mathcal{J}}$  and top element  $\top_{\mathcal{J}} = [0, \infty]$ , and its least upper bound operator is given by

$$I_1 \sqcup_{\mathcal{J}} I_2 = \begin{cases} \perp_{\mathcal{J}} & \text{if } I_1 = I_2 = \perp_{\mathcal{J}} \\ [\inf(\inf_{\mathcal{J}}(I_1), \inf_{\mathcal{J}}(I_2)), \sup(\sup_{\mathcal{J}}(I_1), \sup_{\mathcal{J}}(I_2))] & \text{otherwise.} \end{cases}$$

*Interval Arithmetic.* We would like to define arithmetic operations on elements of the lattice  $(\mathcal{J}, \leq_{\mathcal{J}})$  in order to enable us to add (and subtract) abstract multiplicities of exposed actions in a similar way as we did for  $\mathfrak{M}$ . For this let us consider for a moment an interval  $I$  to be given by a subset  $I \subseteq \mathbb{N} \cup \{\infty\}$  with the convexity property that whenever  $i$  and  $j$  are in  $I$  and  $i \leq k \leq j$  then  $k$  is in  $I$ . An operation  $\star_{\mathcal{J}}$  on intervals can then be defined pointwise

$$I \star_{\mathcal{J}} J = \{i \star_{\mathcal{J}} j \mid i \in I, j \in J\} \cap (\mathbb{N} \cup \{\infty\})$$

where we take  $i + \infty = \infty$  and  $i - \infty = -\infty$  for  $i \in \mathbb{N} \cup \{\infty\}$ . It is easy to establish that, for non-empty intervals, the basic operations addition  $+_{\mathcal{J}}$  and subtraction  $-_{\mathcal{J}}$  of interval arithmetic are equivalently expressed by:

$$\begin{aligned} [i_1, i_2] +_{\mathcal{J}} [j_1, j_2] &= [i_1 + j_1, i_2 + j_2] \\ [i_1, i_2] -_{\mathcal{J}} [j_1, j_2] &= [i_1 - j_2, i_2 - j_1] \cap (\mathbb{N} \cup \{\infty\}) \end{aligned}$$

Note that the neutral element of addition is given by  $\mathbf{0}_{\mathcal{J}} = [0, 0]$ , i.e.  $I +_{\mathcal{J}} \mathbf{0}_{\mathcal{J}} = I$ , whereas the empty interval  $\perp_{\mathcal{J}}$  is an annihilator, i.e.  $I +_{\mathcal{J}} \perp_{\mathcal{J}} = \perp_{\mathcal{J}}$ . Adapting this definition of interval arithmetic to the case where intervals are formally elements of  $\mathcal{J}$  is straightforward.

LEMMA 4.3. *The operations  $+_{\mathcal{J}}$  and  $-_{\mathcal{J}}$  enjoy the following properties:*

- (1)  $+_{\mathcal{J}}$  and  $-_{\mathcal{J}}$  are monotonic in both arguments.
- (2)  $+_{\mathcal{J}}$  is associative and commutative, and for  $-_{\mathcal{J}}$  we have that  $\inf_{\mathcal{J}}(J) \leq \sup_{\mathcal{J}}(I)$  implies  $(I -_{\mathcal{J}} J) +_{\mathcal{J}} K \leq_{\mathcal{J}} (I +_{\mathcal{J}} K) -_{\mathcal{J}} J$  for all  $K$ .

PROOF. Assume in the following that intervals  $I, J, K \neq \perp_{\mathcal{J}}$ , otherwise the results would hold trivially; we can thus write  $I = [i_1, i_2]$ ,  $J = [j_1, j_2]$ , and  $K = [k_1, k_2]$ . The first parts of both (1) and (2) follow straightforwardly from the properties of integer addition.

To prove the second part of (1), we assume  $I \leq_{\mathcal{J}} J$ ; hence  $j_1 \leq i_1$  and  $i_2 \leq j_2$ . Then  $K -_{\mathcal{J}} I = [k_1 - i_2, k_2 - i_1] \cap (\mathbb{N} \cup \{\infty\}) \leq_{\mathcal{J}} [k_1 - j_2, k_2 - j_1] \cap (\mathbb{N} \cup \{\infty\}) = K -_{\mathcal{J}} J$ , because  $k_1 - j_2 \leq k_1 - i_2$  and  $k_2 - i_1 \leq k_2 - j_1$ . Monotonicity in the left argument follows analogously.

For the second part of (2), note that the condition  $\inf_{\mathcal{J}}(J) \leq \sup_{\mathcal{J}}(I)$  spells out as  $j_1 \leq i_2$ . We define  $\underline{n} = \max(n, 0)$  in order to express the intersection with  $(\mathbb{N} \cup \{\infty\})$  in the case of interval subtraction. We have to show

$$\underline{(i_1 + k_1) - j_2} \leq \underline{(i_1 - j_2) + k_1} \quad \text{and} \quad \underline{(i_2 - j_1) + k_2} \leq \underline{(i_2 + k_2) - j_1},$$

where the first inequation holds in general, and the second one holds because of  $j_1 \leq i_2$ .  $\square$

*Approximative Multisets.* Coming back to our original goal of representing over- and under-approximations of the multiplicity of labels, we can now lift  $\mathcal{J}$  to

$$\mathfrak{N} = \mathbf{Lab} \rightarrow \mathcal{J}.$$

and we call a set  $N \in \mathfrak{N}$  an *approximative multiset*. Using pointwise lifting of the interval domain ordering, the domain  $(\mathfrak{N}, \leq_{\mathfrak{N}})$  is again a complete lattice. In particular, the least element  $\perp_{\mathfrak{N}}$  is given by  $\lambda \ell. \perp_{\mathcal{J}}$  and the largest element  $\top_{\mathfrak{N}}$

by  $\lambda\ell.[0, \infty]$ . The neutral element of addition is  $\mathbf{0}_{\mathfrak{N}} = \lambda\ell.\mathbf{0}_{\mathfrak{J}}$  and the results of Lemma 4.3 hold analogously.

The relationship of  $\mathfrak{N}$  with the lattice of extended multisets  $\mathfrak{M} = \mathbf{Lab} \rightarrow \mathbb{N} \cup \{\infty\}$  can be made explicit using the operation  $\lfloor \cdot \rfloor$  defined by

$$\lfloor M \rfloor(\ell) = [M(\ell), M(\ell)]$$

and that intuitively maps integers into the singleton interval containing just that integer. It is easy to show that this operation preserves addition, i.e.  $\lfloor M +_{\mathfrak{M}} N \rfloor = \lfloor M \rfloor +_{\mathfrak{N}} \lfloor N \rfloor$ , and is *non-strict* since  $\lfloor \perp_{\mathfrak{M}} \rfloor = \mathbf{0}_{\mathfrak{N}}$ .

### 4.3 The Representation Function

Connecting extended multisets to our development in Section 2, the representation function  $\beta : S \rightarrow \mathcal{U}$  for processes  $P$  of  $S$  can now be chosen as  $\beta(P) = \lfloor \mathcal{E}_* \llbracket P \rrbracket \rfloor$  as corresponds to choosing  $\mathcal{U}$  to be  $\mathfrak{N}$ .

*Example 4.4.* In comparison with Example 4.1, for the running example of Example 3.1 we get:

$$\beta(S \mid Q \mid Q) = \mathbf{0}_{\mathfrak{N}}[1 \mapsto [1, 1], 3 \mapsto [2, 2]]$$

For the process  $R \triangleq (\bar{a}^3.0 + \tau^5.\bar{r}^4.0) \mid R$  we get:

$$\beta(S \mid R) = \mathbf{0}_{\mathfrak{N}}[1 \mapsto [1, 1], 3 \mapsto [\infty, \infty], 5 \mapsto [\infty, \infty]]$$

It is worthwhile stressing that the information provided by  $\beta$  is exact – there is no amount of approximation going on because we always get singleton intervals. The reason that the interval  $[\infty, \infty]$  can arise is because of the way the unfolding of recursion is handled in Table II as was already discussed in the case of  $\mathcal{E}_*$ .

## 5. MODAL ABSTRACTIONS: “MAY”-TRANSITIONS

Having introduced the abstract domain,  $\mathcal{U}$ , and the representation function,  $\beta = \lfloor \mathcal{E}_* \llbracket \cdot \rrbracket \rfloor$ , in Section 4, we shall embark on the construction of a modal transition system  $(\hat{S}, \mathbf{ALL}, \dashrightarrow, \longrightarrow)$  in Sections 5–7.

We want that  $(\hat{S}, \mathbf{ALL}, \dashrightarrow, \longrightarrow)$  to  $\beta$ -represents the labelled transition system  $(S, \mathbf{ALL}, \rightarrow)$  of let  $A_1 \triangleq P_1; \dots; A_k \triangleq P_k$  in  $P_0$  as defined in Subsection 3.3. We want to perform this development directly from the syntactic presentation of the CCS process since this is finite unlike the labelled transition system that it gives rise to. In this section we focus on the construction of “*may*” transitions that may leave a particular state; this will be used in Section 7 for constructing the  $\dashrightarrow$  relation. In the next section we similarly study the “*must*” transitions that may leave a particular state; this will be used in Section 7 for constructing the  $\longrightarrow$  relation. Finally, in Section 7 we develop a worklist algorithm constructing the set  $\hat{S}$  of abstract states and the relations  $\dashrightarrow$  and  $\longrightarrow$ . As in Section 4, the set  $\mathbf{ALL}$  is the set of all single labels and label pairs

### 5.1 Generated and Killed Actions

The representation function  $\beta$  calculates the information about exposed actions for initial processes. As already discussed this information changes as the process

$$\begin{aligned}
\mathcal{G}[\text{new } x P]env &= \mathcal{G}[P]env \\
\mathcal{G}[P \mid Q]env &= \mathcal{G}[P]env \sqcup_{\mathfrak{T}} \mathcal{G}[Q]env \\
\mathcal{G}[\sum_{i \in I} \alpha_i^{\ell_i} . P_i]env &= \bigsqcup_{\mathfrak{T}, i \in I} (\perp_{\mathfrak{T}}[\ell_i \mapsto N_i] \sqcup_{\mathfrak{T}} \mathcal{G}[P_i]env) \\
&\quad \text{where } N_i = \lfloor \mathcal{E}_* \llbracket P_i \rrbracket \rfloor \\
\mathcal{G}[A]env &= env(A) \\
\mathcal{G}_* \llbracket P \rrbracket &= \mathcal{G}[P]env_{\mathcal{G}} \\
\text{where } \mathcal{F}_{\mathcal{G}}(env) &= [A_1 \mapsto \mathcal{G}[P_1]env, \dots, A_k \mapsto \mathcal{G}[P_k]env] \\
\text{and } env_{\perp_{\mathfrak{T}}} &= [A_1 \mapsto \perp_{\mathfrak{T}}, \dots, A_k \mapsto \perp_{\mathfrak{T}}] \\
\text{and } env_{\mathcal{G}} &= \bigsqcup_{j \geq 0} \mathcal{F}_{\mathcal{G}}^j(env_{\perp_{\mathfrak{T}}})
\end{aligned}$$

Table IV. Approximative multisets of actions *generated* from  $\text{let } A_1 \triangleq P_1; \dots; A_k \triangleq P_k \text{ in } P_0$ 

evolves because once an action has executed, some new actions may become exposed and some may cease to be exposed. In this section, we present auxiliary functions allowing us to approximate this evolution of exposed actions during process execution. While the set of exposed actions computed by  $\beta$  is always precise, indeed we always obtain singleton intervals, this is not the case for generated and killed actions. For example, when executing the action  $\tau^5$  of  $Q$ , we cannot be sure whether  $\bar{r}^6$  or  $\bar{a}^3$  becomes exposed. We therefore let the auxiliary functions give both an over- and an underapproximation of the multiplicities of exposed actions using the abstract domain of Subsection 4.2.

We shall now define the two functions  $\mathcal{G}_*$  and  $\mathcal{K}_*$  approximating generated and killed actions. The relevant information will be an element of

$$\mathfrak{T} = \mathbf{Lab} \rightarrow \mathfrak{N} \quad (= \mathbf{Lab} \rightarrow (\mathbf{Lab} \rightarrow \mathfrak{J}))$$

and  $(\mathfrak{T}, \leq_{\mathfrak{T}})$  is a complete lattice using pointwise lifting of the ordering of  $\mathfrak{N}$ . The idea is that  $\mathcal{G}_* \llbracket P \rrbracket(\ell)$  and  $\mathcal{K}_* \llbracket P \rrbracket(\ell)$  produce approximative multisets of the actions which are newly exposed and no longer exposed, respectively, when the action labelled  $\ell$  executes. In the end we would like to obtain a transfer function of the kind we have hinted at in Section 1, where generated and killed actions will play the role of the  $gen_{\ell}$  and  $kill_{\ell}$  components.

To motivate the definition let us consider prefixing as expressed in the process  $\alpha^{\ell}.P$ . Clearly, once  $\alpha^{\ell}$  has been executed it will no longer be exposed whereas the actions of  $\mathcal{E}_* \llbracket P \rrbracket$  will become exposed. Thus a first suggestion might be to take  $\mathcal{G}_* \llbracket \alpha^{\ell}.P \rrbracket(\ell) = \lfloor \mathcal{E}_* \llbracket P \rrbracket \rfloor$ , where we note that we need to lift  $\mathcal{E}_* \llbracket P \rrbracket$  from  $\mathfrak{M}$  to  $\mathfrak{N}$ ; likewise, we might take  $\mathcal{K}_* \llbracket \alpha^{\ell}.P \rrbracket(\ell) = \mathbf{0}_{\mathfrak{N}}[\ell \mapsto [1, 1]]$  to express that we decrease the count of  $\ell$  by (at least and at most) 1.

However, in the actual definitions in Table IV and Table V we have to cater for the case where the same label may occur several times in a process (as when  $\ell$  is used inside  $P$ ), and thus have to take the least upper bound with  $\mathcal{G}_* \llbracket P \rrbracket$  and  $\mathcal{K}_* \llbracket P \rrbracket$  to ensure that they correctly combine the information available about  $\ell$ . For killed actions it is also worth pointing out that the set  $N$  in the clause for summations in Table V actually equals  $\lfloor \mathcal{E}_* \llbracket \sum_{i \in I} \alpha_i^{\ell_i} . P_i \rrbracket \rfloor$  reflecting that *all* the exposed actions of the summation are indeed killed once one of them has been selected for the reaction

step. The functions

$$\mathcal{G}_\star : \mathbf{Proc} \rightarrow \mathfrak{I} \quad \mathcal{K}_\star : \mathbf{Proc} \rightarrow \mathfrak{I}$$

are defined using auxiliary functions  $\mathcal{G}$  and  $\mathcal{K}$ , following the same pattern and reasoning for well-definedness used when defining  $\mathcal{E}_\star$ .

*Example 5.1.* Continuing Example 3.1, the generated and killed actions for the main process  $S \mid Q \mid Q$  calculate to:

$\ell$	$\mathcal{G}_\star \llbracket S \mid Q \mid Q \rrbracket (\ell)$	$\mathcal{K}_\star \llbracket S \mid Q \mid Q \rrbracket (\ell)$
1	$\mathbf{0}_{\mathfrak{N}}[2 \mapsto [1, 1]]$	$\mathbf{0}_{\mathfrak{N}}[1 \mapsto [1, 1]]$
2	$\mathbf{0}_{\mathfrak{N}}[1 \mapsto [1, 1]]$	$\mathbf{0}_{\mathfrak{N}}[2 \mapsto [1, 1]]$
3	$\mathbf{0}_{\mathfrak{N}}[4 \mapsto [1, 1], 5 \mapsto [2, 2]]$	$\mathbf{0}_{\mathfrak{N}}[3 \mapsto [1, 1]]$
4	$\mathbf{0}_{\mathfrak{N}}[3 \mapsto [1, 1]]$	$\mathbf{0}_{\mathfrak{N}}[4 \mapsto [1, 1], 5 \mapsto [2, 2]]$
5	$\mathbf{0}_{\mathfrak{N}}[3 \mapsto [0, 1], 6 \mapsto [0, 1]]$	$\mathbf{0}_{\mathfrak{N}}[4 \mapsto [1, 1], 5 \mapsto [2, 2]]$
6	$\mathbf{0}_{\mathfrak{N}}[3 \mapsto [1, 1]]$	$\mathbf{0}_{\mathfrak{N}}[6 \mapsto [1, 1]]$

One may note that the intervals are singletons except for two occurrences of  $[0, 1]$ . This expresses an imprecision in our ability to predict whether or not the actions labelled 3 and 6 will be generated when performing an action labelled 5. This is the best we can hope for because there are two actions labelled 5 in Example 3.1.

The next result shows that the information calculated by  $\mathcal{G}_\star$  and  $\mathcal{K}_\star$  is invariant under the structural congruence and that it cannot increase during the evolution of the process:

LEMMA 5.2. *The functions  $\mathcal{G}_\star$  and  $\mathcal{K}_\star$  enjoy the following properties:*

- (1) (a) *If  $P \equiv Q$  then  $\mathcal{G}_\star \llbracket P \rrbracket = \mathcal{G}_\star \llbracket Q \rrbracket$ ; (b) *If  $P \rightarrow_{\bar{\ell}} Q$  then  $\mathcal{G}_\star \llbracket Q \rrbracket \leq_{\mathfrak{I}} \mathcal{G}_\star \llbracket P \rrbracket$ .**
- (2) (a) *If  $P \equiv Q$  then  $\mathcal{K}_\star \llbracket P \rrbracket = \mathcal{K}_\star \llbracket Q \rrbracket$ ; (b) *If  $P \rightarrow_{\bar{\ell}} Q$  then  $\mathcal{K}_\star \llbracket Q \rrbracket \leq_{\mathfrak{I}} \mathcal{K}_\star \llbracket P \rrbracket$ .**

PROOF. We show part (1); the proof for part (2) is completely analogous. For the first part of (1) we show that  $\mathcal{G} \llbracket P \rrbracket \text{env}_{\mathcal{G}} = \mathcal{G} \llbracket Q \rrbracket \text{env}_{\mathcal{G}}$  by induction on how  $P \equiv Q$  is obtained from Table II. When no recursion is involved it is fairly straightforward to show that  $\mathcal{G} \llbracket P \rrbracket \text{env} = \mathcal{G} \llbracket Q \rrbracket \text{env}$  for all  $\text{env}$ . In the case of recursion consider  $A \equiv P$  where  $A$  is defined by  $A \triangleq P$ . To show  $\text{env}_{\mathcal{K}}(A) = \mathcal{G} \llbracket P \rrbracket \text{env}_{\mathcal{K}}$  we observe that  $\text{env}_{\mathcal{K}} = \bigsqcup_{j \geq 0} \mathcal{F}_{\mathcal{G}}^j(\text{env}_{\perp_{\mathfrak{I}}})$  and  $\mathcal{F}_{\mathcal{G}}(\text{env})(A) = \mathcal{G} \llbracket P \rrbracket \text{env}$ .

For the second part of (1) we proceed by induction on the inference of  $P \rightarrow_{\bar{\ell}} Q$  as defined in Table I. First consider one of the two axioms, where it is straightforward to show that  $\mathcal{G}_\star \llbracket Q \rrbracket \leq_{\mathfrak{I}} \mathcal{G}_\star \llbracket P \rrbracket$ . In the case of the three rules the result follows from the induction hypothesis and the first part of the lemma.  $\square$

The next lemma states that the labels definitely killed by  $\mathcal{K}_\star$  never exceed the actions actually exposed by a certain process. To formulate it we need the function  $\text{inf}_{\mathfrak{N}}$  defined by  $\text{inf}_{\mathfrak{N}} N \stackrel{\text{def}}{=} \lambda \ell. \text{inf}_{\mathfrak{I}} N(\ell)$ .

LEMMA 5.3. *If  $P \rightarrow_{\bar{\ell}} Q$  then  $\text{inf}_{\mathfrak{N}} \mathcal{K}_\star \llbracket P \rrbracket(\tilde{\ell}) \leq_{\mathfrak{N}} \mathcal{E}_\star \llbracket P \rrbracket$ .*

PROOF. The result holds for the two axioms because the clause for  $N$  in Table V equals the (lifted) exposed actions of  $P$  and the operator  $\sqcup_{\mathfrak{N}}$  is monotonic. In the case of the three rules the result follows directly from the induction hypothesis.  $\square$

$$\begin{aligned}
\mathcal{K}[\text{new } x P]env &= \mathcal{K}[P]env \\
\mathcal{K}[P \mid Q]env &= \mathcal{K}[P]env \sqcup_{\overline{\mathfrak{T}}} \mathcal{K}[Q]env \\
\mathcal{K}[\sum_{i \in I} \alpha_i^{\ell_i}.P_i]env &= \bigsqcup_{\overline{\mathfrak{T}}} \mathcal{K}[\sum_{i \in I} (\perp_{\overline{\mathfrak{T}}}[\ell_i \mapsto N] \sqcup_{\overline{\mathfrak{T}}} \mathcal{K}[P_i]env)] \\
&\quad \text{where } N = +_{\mathfrak{N}} \mathbf{0}_{\mathfrak{N}}[\ell_j \mapsto [1, 1]] \\
\mathcal{K}[A]env &= env(A) \\
\mathcal{K}_*[P] &= \mathcal{K}[P]env_{\mathcal{K}} \\
\text{where } \mathcal{F}_{\mathcal{K}}(env) &= [A_1 \mapsto \mathcal{K}[P_1]env, \dots, A_k \mapsto \mathcal{K}[P_k]env] \\
\text{and } env_{\perp_{\overline{\mathfrak{T}}}} &= [A_1 \mapsto \perp_{\overline{\mathfrak{T}}}, \dots, A_k \mapsto \perp_{\overline{\mathfrak{T}}}] \\
\text{and } env_{\mathcal{K}} &= \bigsqcup_{j \geq 0} \mathcal{F}_{\mathcal{K}}^j(env_{\perp_{\overline{\mathfrak{T}}}})
\end{aligned}$$

Table V. Approximative multisets of actions *killed* by let  $A_1 \triangleq P_1; \dots; A_k \triangleq P_k$  in  $P_0$ 

We are now ready to establish the following central result stating that generated and killed actions can be combined in order to provide safe (over- and under-) approximations to the exposed actions of the resulting process of a reaction; this is further spelled out in Corollary 5.5 below. Basically this results expresses a *simulation* between the concrete behaviour of processes in CCS and the elements of the abstract domain  $\mathfrak{N}$ .

**THEOREM 5.4.** *If  $P \rightarrow_{\tilde{\ell}} Q$  then  $[\mathcal{E}_*[Q]] \leq_{\mathfrak{N}} ([\mathcal{E}_*[P]] -_{\mathfrak{N}} \mathcal{K}_*[P](\tilde{\ell})) +_{\mathfrak{N}} \mathcal{G}_*[P](\tilde{\ell})$ .*

**PROOF.** We proceed by induction on the inference of  $P \rightarrow_{\tilde{\ell}} Q$  as defined in Table I.

**CASE  $\tau^{\ell}.P + Q \rightarrow_{\ell} P$ .** We first show the following inequality:

$$\mathbf{0}_{\mathfrak{N}} \leq_{\mathfrak{N}} [\mathcal{E}_*[\tau^{\ell}.P + Q]] -_{\mathfrak{N}} \mathcal{K}_*[\tau^{\ell}.P + Q](\ell) \quad (\star)$$

Let  $\sum_{i \in I} \alpha_i^{\ell_i}.P_i$  generalise  $\tau^{\ell}.P + Q$ , hence  $\ell = \ell_k$  for some  $k \in I$ . From the definitions in Tables III and V we have:

$$\begin{aligned}
[\mathcal{E}[\sum_{i \in I} \alpha_i^{\ell_i}.P_i]] &= [\sum_{\mathfrak{M} i \in I} \perp_{\mathfrak{M}}[\ell_i \mapsto 1]] \\
&= +_{\mathfrak{N}} \mathbf{0}_{\mathfrak{N}}[\ell_i \mapsto [1, 1]] \leq_{\mathfrak{N}} \mathcal{K}[\sum_{i \in I} \alpha_i^{\ell_i}.P_i](\ell)
\end{aligned}$$

Hence  $\mathbf{0}_{\mathfrak{N}} \leq_{\mathfrak{N}} [\mathcal{E}_*[\sum_{i \in I} \alpha_i^{\ell_i}.P_i]] -_{\mathfrak{N}} \mathcal{K}_*[\sum_{i \in I} \alpha_i^{\ell_i}.P_i](\ell)$  which proves  $(\star)$ .

We can use Lemma 4.3 (1), i.e. the monotonicity of  $+_{\mathfrak{N}}$ , on  $(\star)$  to establish the first inequation of the following chain, and the definition of generated actions in Table IV for the remaining ones:

$$\begin{aligned}
([\mathcal{E}_*[\tau^{\ell}.P + Q]] -_{\mathfrak{N}} \mathcal{K}_*[\tau^{\ell}.P + Q](\ell)) +_{\mathfrak{N}} \mathcal{G}_*[\tau^{\ell}.P + Q](\ell) \\
\geq_{\mathfrak{N}} \mathbf{0}_{\mathfrak{N}} +_{\mathfrak{N}} \mathcal{G}_*[\tau^{\ell}.P + Q](\ell) &= \mathcal{G}_*[\tau^{\ell}.P + Q](\ell) \\
\geq_{\mathfrak{N}} \mathcal{G}_*[\tau^{\ell}.P](\ell) &\geq_{\mathfrak{N}} [\mathcal{E}_*[P]]
\end{aligned}$$

**CASE  $(\bar{x}^{\ell_1}.P_1 + Q_1) \mid (x^{\ell_2}.P_2 + Q_2) \rightarrow_{\ell_1 \ell_2} P_1 \mid P_2$ .** We use *lhs* as an abbreviation for  $(\bar{x}^{\ell_1}.P_1 + Q_1) \mid (x^{\ell_2}.P_2 + Q_2)$ . As in the previous case, we first show an inequality:

$$\mathbf{0}_{\mathfrak{N}} \leq_{\mathfrak{N}} [\mathcal{E}_*[lhs]] -_{\mathfrak{N}} \mathcal{K}_*[lhs](\ell_1 \ell_2) \quad (\star\star)$$

We generalise *lhs* to summations over index sets  $I$  and  $J$ , where we assume  $\ell_1 = \ell_i$  for  $i \in I$  and  $\ell_2 = \ell_j$  for  $j \in J$ . Then we can prove  $(\star\star)$  using the following

reasoning:

$$\begin{aligned}
 [\mathcal{E}[\Sigma_{i \in I} \alpha_i^{\ell_i} . P_i \mid \Sigma_{j \in J} \beta_j^{\ell_j} . P_j]] &= +_{\mathfrak{N}} \mathbf{0}_{\mathfrak{N}}[\ell_i \mapsto [1, 1]] +_{\mathfrak{N}} +_{\mathfrak{N}} \mathbf{0}_{\mathfrak{N}}[\ell_j \mapsto [1, 1]] \\
 &\leq_{\mathfrak{N}} \mathcal{K}[\Sigma_{i \in I} \alpha_i^{\ell_i} . P_i](\ell_1) +_{\mathfrak{N}} \mathcal{K}[\Sigma_{j \in J} \beta_j^{\ell_j} . P_j](\ell_2) \\
 &= \mathcal{K}[\Sigma_{i \in I} \alpha_i^{\ell_i} . P_i \mid \Sigma_{j \in J} \beta_j^{\ell_j} . P_j](\ell_1 \ell_2)
 \end{aligned}$$

Thus we can use the monotonicity of  $+_{\mathfrak{N}}$  again on  $(\star\star)$  for the first inequation of the following chain, and as before we observe that  $\mathcal{G}_{\star}[\bar{x}^{\ell_1} . P_1 + Q_1](\ell_1) \geq_{\mathfrak{N}} [\mathcal{E}_{\star}[P_1]]$  and  $\mathcal{G}_{\star}[x^{\ell_2} . P_2 + Q_2](\ell_2) \geq_{\mathfrak{N}} [\mathcal{E}_{\star}[P_2]]$ :

$$\begin{aligned}
 (\mathcal{E}_{\star}[\mathit{lhs}] -_{\mathfrak{N}} \mathcal{K}_{\star}[\mathit{lhs}](\ell_1 \ell_2)) +_{\mathfrak{N}} \mathcal{G}_{\star}[\mathit{lhs}](\ell_1 \ell_2) \\
 &\geq_{\mathfrak{N}} \mathcal{G}_{\star}[\mathit{lhs}](\ell_1) +_{\mathfrak{N}} \mathcal{G}_{\star}[\mathit{lhs}](\ell_2) \\
 &\geq_{\mathfrak{N}} \mathcal{G}_{\star}[\bar{x}^{\ell_1} . P_1 + Q_1](\ell_1) +_{\mathfrak{N}} \mathcal{G}_{\star}[x^{\ell_2} . P_2 + Q_2](\ell_2) \\
 &\geq_{\mathfrak{N}} [\mathcal{E}_{\star}[P_1]] +_{\mathfrak{N}} [\mathcal{E}_{\star}[P_2]]
 \end{aligned}$$

Since lifting preserves addition and using the definition of exposed actions in Table III we have  $[\mathcal{E}_{\star}[P_1 \mid P_2]] = [\mathcal{E}_{\star}[P_1]] +_{\mathfrak{N}} [\mathcal{E}_{\star}[P_2]]$ , and this completes the proof of the case.

CASE  $P \mid P' \rightarrow_{\bar{\ell}} Q \mid P'$  because  $P \rightarrow_{\bar{\ell}} Q$ . From the induction hypothesis we have  $[\mathcal{E}_{\star}[Q]] \leq_{\mathfrak{N}} [\mathcal{E}_{\star}[P]] -_{\mathfrak{N}} \mathcal{K}_{\star}[P](\bar{\ell}) +_{\mathfrak{N}} \mathcal{G}_{\star}[P](\bar{\ell})$ . Using the monotonicity of  $+_{\mathfrak{N}}$  we can therefore calculate

$$\begin{aligned}
 [\mathcal{E}_{\star}[Q \mid P']] &= [\mathcal{E}_{\star}[Q]] +_{\mathfrak{N}} [\mathcal{E}_{\star}[P']] \\
 &\leq_{\mathfrak{N}} (([\mathcal{E}_{\star}[P]] -_{\mathfrak{N}} \mathcal{K}_{\star}[P](\bar{\ell})) +_{\mathfrak{N}} \mathcal{G}_{\star}[P](\bar{\ell})) +_{\mathfrak{N}} [\mathcal{E}_{\star}[P']] \\
 &\leq_{\mathfrak{N}} (([\mathcal{E}_{\star}[P]] +_{\mathfrak{N}} [\mathcal{E}_{\star}[P']]) -_{\mathfrak{N}} \mathcal{K}_{\star}[P](\bar{\ell})) +_{\mathfrak{N}} \mathcal{G}_{\star}[P](\bar{\ell}) \\
 &= ([\mathcal{E}_{\star}[P \mid P']] -_{\mathfrak{N}} \mathcal{K}_{\star}[P](\bar{\ell})) +_{\mathfrak{N}} \mathcal{G}_{\star}[P](\bar{\ell})
 \end{aligned}$$

where we use the second part of Lemma 4.3 (2), which is easily lifted to  $\mathfrak{N}$ , to establish the second inequation. Note that we can apply Lemma 4.3 (2) because we know  $\inf_{\mathfrak{N}} \mathcal{K}_{\star}[P](\bar{\ell}) \leq_{\mathfrak{N}} \mathcal{E}_{\star}[P] = \sup_{\mathfrak{N}} [\mathcal{E}_{\star}[P]]$  from Lemma 5.3. The last equation holds because of the definition of exposed actions in Table III.

Furthermore it holds that  $\mathcal{K}_{\star}[P](\bar{\ell}) \leq_{\mathfrak{N}} \mathcal{K}_{\star}[P \mid P'](\bar{\ell})$  and  $\mathcal{G}_{\star}[P](\bar{\ell}) \leq_{\mathfrak{N}} \mathcal{G}_{\star}[P \mid P'](\bar{\ell})$ . Using that  $+_{\mathfrak{N}}$  is monotonic and that  $-_{\mathfrak{N}}$  is monotonic in its right argument as stated in Lemma 4.3 (1) we may calculate

$$\begin{aligned}
 ([\mathcal{E}_{\star}[P \mid P']] -_{\mathfrak{N}} \mathcal{K}_{\star}[P](\bar{\ell})) +_{\mathfrak{N}} \mathcal{G}_{\star}[P](\bar{\ell}) \\
 \leq_{\mathfrak{N}} ([\mathcal{E}_{\star}[P \mid P']] -_{\mathfrak{N}} \mathcal{K}_{\star}[P \mid P'](\bar{\ell})) +_{\mathfrak{N}} \mathcal{G}_{\star}[P \mid P'](\bar{\ell})
 \end{aligned}$$

which concludes the proof of this case.

CASE  $\text{new } x P \rightarrow_{\bar{\ell}} \text{new } x P'$  because  $P \rightarrow_{\bar{\ell}} P'$ . The proof follows directly from the induction hypothesis as  $\mathcal{E}_{\star}[\text{new } x P] = \mathcal{E}_{\star}[P]$ ,  $\mathcal{G}_{\star}[\text{new } x P] = \mathcal{G}_{\star}[P]$  and  $\mathcal{K}_{\star}[\text{new } x P] = \mathcal{K}_{\star}[P]$ .

CASE  $P \rightarrow_{\bar{\ell}} Q$  because  $P \equiv P', P' \rightarrow_{\bar{\ell}} Q'$  and  $Q' \equiv Q$ . This case is straightforward as  $\mathcal{E}_{\star}$ ,  $\mathcal{G}_{\star}$  and  $\mathcal{K}_{\star}$  are all invariant under the structural congruence as stated in Lemmas 4.2 and 5.2. This completes the proof of the theorem.  $\square$

## 5.2 The Transfer Function

Theorem 5.4 enables us to adapt the generic transfer function of Section 1 to our setting

$$\text{transfer}_{P_0, \tilde{\ell}}(\hat{s}) = (\hat{s} -_{\mathfrak{N}} \mathcal{K}_*[[P_0]](\tilde{\ell})) +_{\mathfrak{N}} \mathcal{G}_*[[P_0]](\tilde{\ell}),$$

where  $\hat{s}$  is the the approximative multiset of exposed actions available at a particular program point, which in turn is identified by the actions  $\tilde{\ell}$  that may be executed. Clearly, the transfer function is monotonic in  $\hat{s}$ . Also note that the transfer function can be precomputed as it relies only on the initial program  $P_0$ , and the following corollary confirms that in this manner it still provides a safe approximation similar to the one in Theorem 5.4.

**COROLLARY 5.5.** *Consider the program let  $A_1 \triangleq P_1; \dots; A_k \triangleq P_k$  in  $P_0$  and assume  $P_0 \rightarrow^* P \rightarrow_{\tilde{\ell}} Q$ . Then  $[\mathcal{E}_*[[Q]]] \leq_{\mathfrak{N}} \text{transfer}_{P_0, \tilde{\ell}}([\mathcal{E}_*[[P]])$ .*

**PROOF.** This is an immediate consequence of Theorem 5.4, Lemmas 4.3, and 5.2.  $\square$

*Example 5.6.* Following up on Examples 3.2, 4.1, and 5.1, we have  $S \mid Q \mid Q \rightarrow^* r^2.S \mid \bar{r}^6.Q \mid Q \rightarrow_{26} S \mid Q \mid Q$  and

$$\begin{aligned} [\mathcal{E}_*[[r^2.S \mid \bar{r}^6.Q \mid Q]]] &= \mathbf{0}_{\mathfrak{N}}[2 \mapsto [1, 1], 3 \mapsto [1, 1], 6 \mapsto [1, 1]] \\ \mathcal{K}_*[[S \mid Q \mid Q]](26) &= \mathbf{0}_{\mathfrak{N}}[2 \mapsto [1, 1], 6 \mapsto [1, 1]] \\ \mathcal{G}_*[[S \mid Q \mid Q]](26) &= \mathbf{0}_{\mathfrak{N}}[1 \mapsto [1, 1], 3 \mapsto [1, 1]] \end{aligned}$$

and hence we can confirm Corollary 5.5 for this example:

$$[\mathcal{E}_*[[S \mid Q \mid Q]]] \leq_{\mathfrak{N}} \text{transfer}_{P_0, 26}([\mathcal{E}_*[[r^2.S \mid \bar{r}^6.Q \mid Q]]) = \mathbf{0}_{\mathfrak{N}}[1 \mapsto [1, 1], 3 \mapsto [2, 2]]$$

One may note that all intervals are singletons showing that the analysis is exact in this case.

### Identifying “May” Transitions

Looking back at Definition 2.2 it is useful to interpret Corollary 5.5 as follows. If there is a transition  $P \rightarrow_{\tilde{\ell}} Q$  in the concrete model and  $\beta(P) \leq_{\mathfrak{N}} \hat{s}$ , i.e.  $\hat{s}$  is the abstract state representing  $P$ , then  $\hat{s}' = \text{transfer}_{P_0, \tilde{\ell}}(\hat{s})$  represents  $Q$ , i.e.  $\beta(Q) \leq_{\mathfrak{N}} \hat{s}'$ . Hence, in the modal transition system to be constructed in Section 7, there has to be a “may” transition labelled  $\tilde{\ell}$  between  $\hat{s}$  and  $\hat{s}'$ .

## 6. MODAL ABSTRACTIONS: “MUST”-TRANSITIONS

The development of Subsection 5.2 concluded by suggesting when to insert “may” transitions into an abstract modal transition system. In Subsection 6.2 we improve on the precision of this technique, and find the cases where we can insert a “must” transition. The machinery of Kleene’s three-valued logic [Kleene 1952] helps to express our considerations and is the topic of Subsection 6.1.

### 6.1 Three-valued Logic

In Kleene’s 3-valued logic [Kleene 1952] the classical set of truth values  $\mathbb{B} = \{0, 1\}$  is extended with a value  $1/2$  for expressing uncertainty, i.e.  $\mathbb{B}_3 = \{0, 1/2, 1\}$ . We say that 0 and 1 are *definite* values, while  $1/2$  is *indefinite*. The set  $\mathbb{B}_3$  can be ordered in two ways. In the *logical order* (also known as the truth order)  $\leq^3$  we take

$0 \leq^3 1/2 \leq^3 1$ , and conjunction  $\wedge^3$  (resp. disjunction  $\vee^3$ ) can then be defined as the minimum (resp. maximum) of its arguments. These operators can be extended to indexed formulae in the obvious way, in particular  $\bigwedge_{i \in \emptyset}^3 \phi_i = 1$  and  $\bigvee_{i \in \emptyset}^3 \phi_i = 0$ . 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 [Nielsen et al. 2001].

The *information order*  $\sqsubseteq^3$  on the other hand is defined by  $0 \sqsubseteq^3 1/2$ ,  $1 \sqsubseteq^3 1/2$ , and  $b \sqsubseteq^3 b$  for all  $b \in \mathbb{B}_3$ . This order captures the notion of (un)certainty and can be used to relate 2- and 3-valued interpretations. We use square operators (e.g.  $\sqcap^3$ ) in the information order, and angle ones (e.g.  $\wedge^3$ ) in the logical order.

An abstract state  $\hat{s}$  represents the exposed actions of some concrete process. For  $\hat{s}(\ell)$  evaluating to  $[m, n]$  we can distinguish three cases: if  $m, n > 0$ , then the label  $\ell$  is definitely exposed; if  $m = 0$  and  $n > 0$ , it may be exposed; and, if  $m = n = 0$ , it is definitely not exposed. Using three-valued logic, we can express this by the function  $L_{\hat{s}}$ :

$$L_{\hat{s}}(\ell) = \begin{cases} 1 & \text{if } \hat{s}(\ell) = [m, n] \text{ and } m, n > 0 \\ 1/2 & \text{if } \hat{s}(\ell) = [0, n] \text{ and } n > 0 \\ 0 & \text{if } \hat{s}(\ell) = [0, 0] \end{cases}$$

Note that the case  $\hat{s}(\ell) = \perp_{\mathcal{J}}$  will never arise during our construction because  $[\mathcal{E}_* \llbracket Q \rrbracket] \neq \perp_{\mathfrak{N}}$  for all processes  $Q$ ; in fact  $\perp_{\mathcal{J}}$  denotes absence of information and is only needed for initialising our fixed point computations. (If we were to cater for the possibility of  $\perp_{\mathcal{J}}$  we would have to step out of the setting of three-valued logic and set  $L_{\hat{s}}(\ell) = \perp$  in this case.)

**FACT 6.1.** *If  $\hat{s}_1 \leq_{\mathfrak{N}} \hat{s}_2$  then  $L_{\hat{s}_1}(\ell) \sqsubseteq^3 L_{\hat{s}_2}(\ell)$ .*

We shall see that the value 1 will be used to construct “*must*” transitions, the value 1/2 for the remaining “*may*” transitions, and the value 0 for the absence of any transitions.

## 6.2 Enabled Actions

Not all exposed actions are able to execute; as an example, if an action  $x^{\ell_1}$  is exposed in a parallel composition but there is no exposed counterpart of the form  $\bar{x}^{\ell_2}$  then the action  $x^{\ell_1}$  cannot execute.

This motivates us to take a closer look at the labels of exposed actions. An action  $\tau^{\ell}$  can execute whenever  $\tau^{\ell}$  is exposed; the label  $\ell$  is said to be *enabled* in this case. Two actions  $x^{\ell_1}$  and  $\bar{x}^{\ell_2}$  can execute whenever they are both exposed and occur in *parallel* processes; the label sequence  $\ell_1 \ell_2$  (or the label pair  $(\ell_1, \ell_2)$ ) is said to be *enabled* in this case. The formal definition needs to go a bit deeper into when two actions are matching actions that occur in parallel processes.

*Compatible Actions.* We shall now specify the set of *compatible* actions, that is, pairs of matching actions that may occur in parallel processes. Consider the process  $P \mid P'$ . Clearly interactions may occur locally within each of  $P$  and  $P'$  but since the two processes are in parallel it is also possible for an action of  $P$  to interact with one in  $P'$ . To capture this assume that  $\ell$  and  $\ell'$  are two labels occurring in  $P$  and  $P'$ , respectively. Recalling that  $\partial(\ell)$  is the canonical action associated with

$$\begin{aligned}
\mathcal{C}^1[\text{new } x P]env &= \mathcal{C}^1[P]env \\
\mathcal{C}^1[P \mid P']env &= \mathcal{C}^1[P]env \sqcup_{\mathfrak{R}_1} \mathcal{C}^1[P']env \\
\mathcal{C}^1[\sum_{i \in I} \alpha_i^{\ell_i}. P_i]env &= \bigsqcup_{\mathfrak{R}_1, i \in I} (\perp_{\mathfrak{R}_1}[\ell_i \mapsto 1] \sqcup_{\mathfrak{R}_1} \mathcal{C}^1[P_i]env) \\
\mathcal{C}^1[A]env &= env(A) \\
\mathcal{C}_*^1[P] &= \mathcal{C}^1[P]env_{\mathcal{C}^1} \\
\text{where } \mathcal{F}_{\mathcal{C}^1}(env) &= [A_1 \mapsto \mathcal{C}^1[P_1]env, \dots, A_k \mapsto \mathcal{C}^1[P_k]env] \\
\text{and } env_{\perp_{\mathfrak{R}_1}} &= [A_1 \mapsto \perp_{\mathfrak{R}_1}, \dots, A_k \mapsto \perp_{\mathfrak{R}_1}] \\
\text{and } env_{\mathcal{C}^1} &= \bigsqcup_{\mathfrak{R}_1, j \geq 0} \mathcal{F}_{\mathcal{C}^1}^j(env_{\perp_{\mathfrak{R}_1}})
\end{aligned}$$

Table VI. Single actions for the program let  $A_1 \triangleq P_1; \dots; A_k \triangleq P_k$  in  $P_0$ 

the label  $\ell$  in a consistently labelled process we take

$$\text{match}(\ell, \ell') = \exists x : (\partial(\ell) = [x] \wedge \partial(\ell') = [\bar{x}]) \vee (\partial(\ell') = [x] \wedge \partial(\ell) = [\bar{x}])$$

to specify the potential matching pairs of actions from the two processes. Note that we are testing on the canonical actions as the analysis does not capture the disciplined alpha-renaming of the semantics – it cannot distinguish between the different instances of the names obtained by e.g. unfolding a recursive process that introduces new names.

Let the set  $\{0, 1\}$  be a complete lattice with  $0 \leq 1$ , and let  $\mathcal{P}(\{0, 1\})$  be a complete lattice ordered by ordinary set inclusion. By pointwise extension we obtain the following complete lattices:

$$\mathfrak{R}_1 = \mathbf{Lab} \rightarrow \{0, 1\} \quad \mathfrak{R}_2 = \mathbf{Lab} \times \mathbf{Lab} \rightarrow \mathcal{P}(\{0, 1\})$$

Using the overall pattern developed in Sections 4 and 5, we define the two functions

$$\mathcal{C}_*^1 : \mathbf{Proc} \rightarrow \mathfrak{R}_1 \quad \mathcal{C}_*^2 : \mathbf{Proc} \rightarrow \mathfrak{R}_2$$

explained in the sequel. We intend that  $\mathcal{C}_*^1[[P]](\ell)$  should yield 1 if  $\ell$  labels an action that may arise during the execution of  $P$ , and 0 otherwise. The formal definition is given in Table VI.

Similarly we intend that  $\mathcal{C}_*^2[[P]](\ell\ell')$  should express whether a pair of labels in  $P$  *can*, *may*, or *cannot* interact when taking the values  $\{1\}$ ,  $\{0, 1\}$ , or  $\{0\}$ , respectively; a pair of labels is not reachable when its value equals  $\emptyset$ . The formal definition is given in Table VII. Note that in the clause for parallel composition the set  $J$  contains all pairs of labels that match and are exposed according to  $\mathcal{C}_*^1$ ; these pairs of labels are recorded as definite candidates for interaction,  $\perp_{\mathfrak{R}_2}[\ell\ell' \mapsto \{1\}]$ . On the other hand, in the clause for sums,  $J$  contains all matching labels that are exposed within a sum; since only parallel processes can interact, these pairs of labels are recorded as impossible candidates for interaction,  $\perp_{\mathfrak{R}_2}[\ell\ell' \mapsto \{0\}]$ . Should a pair of labels  $\ell\ell'$  both be recorded to appear in a parallel composition and in a sum (see Example 6.2 below), the join operator  $\sqcup_{\mathfrak{R}_2}$  ensures that  $\perp_{\mathfrak{R}_2}[\ell\ell' \mapsto \{0, 1\}]$ , i.e. the pair of labels *may* interact. The domain  $\mathfrak{R}_2$  thus helps to distinguish between over-approximation and precise computation of executable actions. Note that in many ways,  $\{1\}$  corresponds to the 1 produced by  $L_{\hat{s}}$ ,  $\{0, 1\}$  to the  $1/2$ , and  $\{0\}$  to the 0 (with  $\emptyset$  corresponding to the  $\perp$  briefly discussed in Subsection 6.1).

$$\begin{aligned}
 \mathcal{C}^2[\text{new } x P] \text{env} &= \mathcal{C}^2[P] \text{env} \\
 \mathcal{C}^2[P \mid P'] \text{env} &= \text{let } J = \{\ell\ell', \ell'\ell \mid \mathcal{C}_*^1[P](\ell) = 1, \mathcal{C}_*^1[P'](\ell') = 1, \text{match}(\ell, \ell')\} \\
 &\quad \text{in } \mathcal{C}^2[P] \text{env} \sqcup_{\mathfrak{R}_2} \mathcal{C}^2[P'] \text{env} \sqcup_{\mathfrak{R}_2} \bigsqcup_{\mathfrak{R}_2 \ell\ell' \in J} \perp_{\mathfrak{R}_2} [\ell\ell' \mapsto \{1\}] \\
 \mathcal{C}^2[\Sigma_{i \in I} \alpha_i^{\ell_i} . P_i] \text{env} &= \text{let } J = \{\ell_i \ell_j \mid i, j \in I, \text{match}(\ell_i, \ell_j)\} \\
 &\quad \text{in } \bigsqcup_{\mathfrak{R}_2 i \in I} \mathcal{C}^2[P_i] \text{env} \sqcup_{\mathfrak{R}_2} \bigsqcup_{\mathfrak{R}_2 \ell\ell' \in J} \perp_{\mathfrak{R}_2} [\ell\ell' \mapsto \{0\}] \\
 \mathcal{C}^2[A] \text{env} &= \text{env}(A) \\
 \mathcal{C}_*^2[P] &= \mathcal{C}^2[P] \text{env}_{\mathcal{C}^2} \\
 \text{where } \mathcal{F}_{\mathcal{C}^2}(\text{env}) &= [A_1 \mapsto \mathcal{C}^2[P_1] \text{env}, \dots, A_k \mapsto \mathcal{C}^2[P_k] \text{env}] \\
 \text{and } \text{env}_{\perp_{\mathfrak{R}_2}} &= [A_1 \mapsto \perp_{\mathfrak{R}_2}, \dots, A_k \mapsto \perp_{\mathfrak{R}_2}] \\
 \text{and } \text{env}_{\mathcal{C}^2} &= \bigsqcup_{\mathfrak{R}_2 j \geq 0} \mathcal{F}_{\mathcal{C}^2}^j(\text{env}_{\perp_{\mathfrak{R}_2}})
 \end{aligned}$$

 Table VII. Compatible actions for the program  $\text{let } A_1 \triangleq P_1; \dots; A_k \triangleq P_k \text{ in } P_0$ 

*Example 6.2.* Continuing Example 3.1, we have  $\mathcal{C}_*^2[P] = \perp_{\mathfrak{R}_2}[13 \mapsto \{1\}, 24 \mapsto \{1\}, 26 \mapsto \{1\}]$ , where we disregard the order of the label pairs.

On the other hand, consider the program  $\text{let } A \triangleq a^1.A + \bar{a}^2.0 \text{ in } A \mid A$ . We have  $\mathcal{C}_*^2[A \mid A] = \perp_{\mathfrak{R}_2}[12 \mapsto \{0, 1\}]$ . This expresses the fact that, during the evolution of the program,  $a^1$  and  $\bar{a}^2$  may both be exposed but can sometimes execute,  $A \mid A \rightarrow_{12} A$ , and sometimes not,  $A \not\rightarrow_{12} A$ .

LEMMA 6.3. *The functions  $\mathcal{C}_*^1$  and  $\mathcal{C}_*^2$  enjoy the following properties:*

- (1) (a) *If  $P \equiv Q$  then  $\mathcal{C}_*^1[P] = \mathcal{C}_*^1[Q]$ ; (b) *If  $P \rightarrow_{\ell} Q$  then  $\mathcal{C}_*^1[Q] \leq_{\mathfrak{R}_1} \mathcal{C}_*^1[P]$  and  $\mathcal{C}_*^1[P](\ell) = 1$ .**
- (2) (a) *If  $P \equiv Q$  then  $\mathcal{C}_*^2[P] = \mathcal{C}_*^2[Q]$ ; (b) *If  $P \rightarrow_{\ell\ell'} Q$  then  $\mathcal{C}_*^2[Q] \leq_{\mathfrak{R}_2} \mathcal{C}_*^2[P]$  and  $1 \in \mathcal{C}_*^2[P](\ell\ell')$ .**

PROOF. The proof is analogous to that of Lemma 4.2.  $\square$

It follows that the  $\emptyset$  discussed previously does not arise in the evaluation of CCS processes.

*Enabled Actions.* The set of enabled actions can now be defined relative to a program  $\text{let } A_1 \triangleq P_1; \dots; A_k \triangleq P_k \text{ in } P_0$  as follows:

$$\begin{aligned}
 \text{enabled}_{P_0}(\hat{s}) &= \{(\ell, \mathsf{L}_{\hat{s}}(\ell)) \mid \mathcal{C}_*^1[P_0](\ell) = 1, \mathsf{L}_{\hat{s}}(\ell) \geq^3 1/2, \partial(\ell) = \tau\} \\
 &\quad \cup \{(\ell\ell', \mathsf{L}_{\hat{s}}(\ell) \wedge^3 \mathsf{L}_{\hat{s}}(\ell')) \mid \mathcal{C}_*^2[P_0](\ell\ell') = \{1\}, \mathsf{L}_{\hat{s}}(\ell), \mathsf{L}_{\hat{s}}(\ell') \geq^3 1/2\} \\
 &\quad \cup \{(\ell\ell', 1/2) \mid \mathcal{C}_*^2[P_0](\ell\ell') = \{0, 1\}, \mathsf{L}_{\hat{s}}(\ell), \mathsf{L}_{\hat{s}}(\ell') \geq^3 1/2\}
 \end{aligned}$$

Hence  $\text{enabled}_{P_0}$  provides a set of single labels  $\ell$  of  $\tau$ -actions (the only individual actions that are allowed to execute by the semantics of Table I) and label pairs  $\ell\ell'$  of synchronising actions, in both cases paired with a value  $b$  that expresses the degree of certainty (among 1/2 and 1) for being enabled. Note that  $\text{enabled}_{P_0}(\hat{s})$  is functional in the sense that  $(\tilde{\ell}, b), (\tilde{\ell}, b') \in \text{enabled}_{P_0}(\hat{s})$  implies  $b = b'$ . The function yields 1 (definitely enabled) only in the case that all involved labels are also definitely exposed (and, in the case of label pairs, definitely compatible), and otherwise 1/2 (maybe enabled). The following lemma relates the degree of certainty  $b$  to the behaviour of the process under consideration.

LEMMA 6.4. *Consider the program  $\text{let } A_1 \triangleq P_1; \dots; A_k \triangleq P_k \text{ in } P_0$  and suppose that  $P_0 \rightarrow^* P$ .*

- (1) If  $P \rightarrow_{\tilde{\ell}} Q$  then  $(\tilde{\ell}, b) \in \text{enabled}_{P_0}(\llbracket \mathcal{E}_* \llbracket P \rrbracket \rrbracket)$  with  $b \geq^3 1/2$ .  
(2) If  $(\tilde{\ell}, 1) \in \text{enabled}_{P_0}(\llbracket \mathcal{E}_* \llbracket P \rrbracket \rrbracket)$  then there exists a process  $Q$  such that  $P \rightarrow_{\tilde{\ell}} Q$ .

PROOF. The first part is an immediate consequence of Lemmas 4.2 and 6.3, the definition of `enabled`, and the definition of  $L_{\hat{s}}$ .

In the proof of the second part, we write  $\hat{s}$  for  $\llbracket \mathcal{E}_* \llbracket P \rrbracket \rrbracket$ . If  $(\tilde{\ell}, 1) \in \text{enabled}_{P_0}(\hat{s})$  then we know  $\mathcal{C}_*^2 \llbracket P_0 \rrbracket (\ell \ell') = \{1\}$  and  $L_{\hat{s}}(\ell) = L_{\hat{s}}(\ell') = 1$  by the definition of `enabled`. By the definition of  $L_{\hat{s}}$  actions labelled  $\ell$  and  $\ell'$  are thus exposed in  $P$ . It just remains to show that they are exposed in *parallel* processes within  $P$ , because then the operational semantics ensures a transition  $P \rightarrow_{\ell \ell'} Q$  (using the synchronisation rule). Suppose  $\ell$  and  $\ell'$  are not exposed in parallel processes within  $P$ . Then both must be exposed in a sum  $\sum_{i \in I} \alpha_i^{\ell_i} . P_i$ , i.e. there are  $i, j \in I$  such that  $\ell = \ell_i$  and  $\ell' = \ell_j$ . However, this sum is also analysed in the initial process  $P_0$  where  $\mathcal{C}_*^2 \llbracket \sum_{i \in I} \alpha_i^{\ell_i} . P_i \rrbracket (\ell \ell') = \{0\}$  by Definition VII, and therefore  $0 \in \mathcal{C}_*^2 \llbracket P_0 \rrbracket (\ell \ell')$ . This constitutes a contradiction to  $\mathcal{C}_*^2 \llbracket P_0 \rrbracket (\ell \ell') = \{1\}$ .  $\square$

FACT 6.5. *The function  $\text{enabled}_{P_0}$  is monotonic in the following sense: if  $\hat{s}_1 \leq_{\mathfrak{N}} \hat{s}_2$  and  $(\tilde{\ell}, b_1) \in \text{enabled}_{P_0}(\hat{s}_1)$  then there exists  $b_2$  such that  $(\tilde{\ell}, b_2) \in \text{enabled}_{P_0}(\hat{s}_2)$  and  $b_1 \sqsubseteq^3 b_2$ .*

### Identifying “Must” Transitions

Looking back at Definition 2.2 it is useful to interpret Lemma 6.4 (and Corollary 5.5) as follows: if  $\beta(P) \leq_{\mathfrak{N}} \hat{s}$  and  $P \rightarrow_{\tilde{\ell}} Q$  then there exists  $b \geq^3 1/2$  such that  $(\tilde{\ell}, b) \in \text{enabled}_{P_0}(\hat{s})$  and in the modal transition system to be constructed in Section 7 there has to be a “*may*” transition labelled  $\tilde{\ell}$  from  $\hat{s}$  to some  $\hat{s}'$  such that  $\text{transfer}_{P_0, \tilde{\ell}}(\hat{s}) \leq_{\mathfrak{N}} \hat{s}'$ ; furthermore, if  $\beta(P) \leq_{\mathfrak{N}} \hat{s}$  and  $(\tilde{\ell}, 1) \in \text{enabled}_{P_0}(\hat{s})$  then there exists  $Q$  such that  $P \rightarrow_{\tilde{\ell}} Q$  and in the modal transition system to be constructed in Section 7 there has to be a “*must*” transition labelled  $\tilde{\ell}$  from  $\hat{s}$  to some  $\hat{s}'$  such that  $\text{transfer}_{P_0, \tilde{\ell}}(\hat{s}) \leq_{\mathfrak{N}} \hat{s}'$ .

## 7. THE CONSTRUCTION OF MODAL ABSTRACTIONS

In the previous sections we got a good handle on when to construct “*may*” transitions and when to construct “*must*” transitions for a program let  $A_1 \triangleq P_1; \dots; A_k \triangleq P_k$  in  $P_0$ . In Subsection 7.1 we develop the worklist algorithm that succeeds in always constructing a finite abstract modal transition system that represents the concrete transition system thereby concluding the development of our main contribution. This is achieved by using a widening and a careful choice of so-called granularity function. In Subsection 7.2 we report on our implementation of the algorithm and its performance on a few examples. One is an example of a buffer that can hold two types of entities and where we show how to adapt our algorithm in the amount of detail produced; we shall return to this example in the next section. Another is the Ingemarsson-Tang-Wong (ITW) protocol that we have previously addressed using an over-approximation based analysis [Nielson and Nielson 2007a].

### 7.1 The Worklist Algorithm

Given a program let  $A_1 \triangleq P_1; \dots; A_k \triangleq P_k$  in  $P_0$  we shall now construct a finite modal transition system that represents the potentially infinite labelled transition system of the program.

```

INPUT:    let  $A_1 \triangleq P_1; \dots; A_k \triangleq P_k$  in  $P_0$ 
OUTPUT:   $(\hat{S}, \mathbf{ALL}, \dashrightarrow, \longrightarrow)$  where  $\mathbf{ALL} = \mathbf{Lab} \cup (\mathbf{Lab} \times \mathbf{Lab})$ 

1   $\hat{S}_0 := \{\llbracket \mathcal{E}_* \llbracket P_0 \rrbracket \rrbracket\}; W := \hat{S}_0; \dashrightarrow := \emptyset; \longrightarrow := \emptyset; i := 0;$ 
2  while  $W \neq \emptyset$  do
3    select  $\hat{s}$  from  $W$ ;  $W := W \setminus \{\hat{s}\}; i := i + 1;$ 
4    for each  $(\tilde{\ell}, b) \in \text{enabled}_{P_0}(\hat{s})$  do
5      let  $\hat{t} = \text{transfer}_{P_0, \tilde{\ell}}(\hat{s})$  in
6         $\dashrightarrow := (\dashrightarrow \setminus \{\hat{s} \dashrightarrow_{\tilde{\ell}} \hat{t}' \mid \hat{t}' \in \hat{S}_i\}) \cup \{\hat{s} \dashrightarrow_{\tilde{\ell}} \hat{t}\};$ 
7         $\longrightarrow := (\longrightarrow \setminus \{\hat{s} \longrightarrow_{\tilde{\ell}} \hat{t}' \mid \hat{t}' \in \hat{S}_i\}) \cup \{\hat{s} \longrightarrow_{\tilde{\ell}} \hat{t} \mid b = 1\}$ 
8        if there exists  $\hat{t}' \in \hat{S}_i$  with  $H(\hat{t}') = H(\hat{t})$  then
9          if  $\hat{t} \leq_{\mathfrak{N}} \hat{t}'$  then  $\hat{u} := \hat{t}'$  else  $\hat{u} := \hat{t}' \nabla_{\mathfrak{N}} \hat{t}; W := W \cup \{\hat{u}\}$  fi
10          $(\hat{S}_{i+1}, W, \dashrightarrow, \longrightarrow) := (\hat{S}_i \cup \{\hat{t}\}, W, \dashrightarrow, \longrightarrow)[\hat{u}/\hat{t}, \hat{u}/\hat{t}']$ 
11        else  $W := W \cup \{\hat{t}\}; \hat{S}_{i+1} := \hat{S}_i \cup \{\hat{t}\}$  fi
12  $(\hat{S}, \mathbf{ALL}, \dashrightarrow, \longrightarrow) := (\hat{S}_{i+1}, \mathbf{Lab} \cup (\mathbf{Lab} \times \mathbf{Lab}), \dashrightarrow, \longrightarrow)$ 
    
```

Table VIII. The worklist algorithm for constructing the modal abstraction

We design a worklist algorithm, based on the function `transfer` of Section 5 and the function `enabled` of Section 6, that produces a faithful modal abstraction for any input process. The main data structures of the algorithm are:

- (1) a set  $\hat{S}_i \subseteq \mathfrak{N}$  of the states introduced up until iteration  $i$ ;
- (2) a worklist  $W$  of states that have yet to be processed; and,
- (3) two sets  $\dashrightarrow$  and  $\longrightarrow$  defining the current “must” and “may” transitions.

*The Iterative Algorithm.* The overall algorithm has the form displayed in Table VIII. Several initialisations are performed in line 1. Line 2 contains the classical loop inspecting the contents of the worklist. A source state  $\hat{s}$  is selected and removed from the worklist in line 3 and the set of (definite and potential) interactions is constructed using the function call `enabledP0( $\hat{s}$ )` in line 4.

For each pair  $(\tilde{\ell}, b) \in \text{enabled}_{P_0}(\hat{s})$  the function call `transferP0,  $\tilde{\ell}$ ( $\hat{s}$ )` of line 5 will return an abstract target state  $\hat{t}$ . The transition relations are updated in lines 6 and 7. In all cases, a “may” edge between  $\hat{s}$  and  $\hat{t}$  is added. But only if  $b = 1$ , i.e. if we are sure that there is a corresponding concrete behaviour due to Lemma 6.4 (2), we add a “must” edge. In both cases, old transitions labelled  $\tilde{\ell}$  from  $\hat{t}$  will be removed, which ensures that we get a modal transition system where no two transitions out of a state have the same label.

Lines 8-11 are essential for the termination of the algorithm and are concerned with deciding whether the transition system can be shrunk by joining one of the existing states with the target state  $\hat{t}$ . In line 8 we first decide, using a *granularity function*  $H$ , which we will further discuss below, whether there exists a candidate state  $\hat{t}'$  for such a join.

If no candidate state  $\hat{t}'$  exists, the transition system cannot be shrunk and the state  $\hat{t}$  is included in the set of states and put on the worklist (for future processing) in line 11.

If a candidate state  $\hat{t}'$  is found in line 8 then the transition system can be shrunk by merging  $\hat{t}$  and  $\hat{t}'$ . We perform this merge by replacing both  $\hat{t}$  and  $\hat{t}'$  with a new common representative  $\hat{u}$  calculated in line 9. The common representative  $\hat{u}$

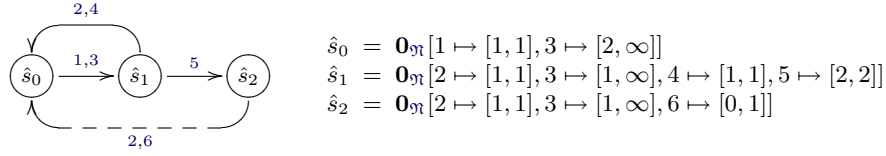


Fig. 2. The modal abstraction for the program of Example 3.1

equals  $\hat{t}'$  in case it subsumes the new state  $\hat{t}$ , otherwise we construct the common representative as the state  $\hat{t}' \nabla_{\mathfrak{N}} \hat{t}$  which is then put on the worklist (for future processing). Here, the *widening operator*  $\nabla_{\mathfrak{N}}$ , which we will further discuss below, makes sure to combine the states in such a way that termination of the overall algorithm is ensured. In all cases we construct the next iteration of the transition system in line 10 by including the abstract state  $\hat{t}$  in the set of states and by replacing both  $\hat{t}$  and  $\hat{t}'$  by their new common representative (by means of the substitution  $[\hat{u}/\hat{t}, \hat{u}/\hat{t}']$ ).

*The Widening Operator.* The *widening operator*  $\nabla_{\mathfrak{N}} : \mathfrak{N} \times \mathfrak{N} \rightarrow \mathfrak{N}$  used in line 9 of Table VIII to combine approximative multisets is defined in two steps. First we define the following widening on natural numbers that was already considered in [Nielson and Nielson 2007a; 2009]:

$$i_1 \nabla i_2 = \begin{cases} i_1 & \text{if } i_2 \leq i_1 \\ i_2 & \text{if } i_1 = 0 \wedge i_2 > 0 \\ \infty & \text{otherwise} \end{cases}$$

Next we extend it to a widening of approximative multisets:

$$(N_1 \nabla_{\mathfrak{N}} N_2)(\ell) = \begin{cases} \perp_{\mathfrak{J}} & \text{if } N_1(\ell) = N_2(\ell) = \perp_{\mathfrak{J}} \\ [\inf(\inf_{\mathfrak{J}} N_1(\ell), \inf_{\mathfrak{J}} N_2(\ell)), (\sup_{\mathfrak{J}} N_1(\ell)) \nabla (\sup_{\mathfrak{J}} N_2(\ell))] & \text{othw.} \end{cases}$$

It will ensure that the sequence  $(\hat{S}_i)_{i \in \mathbb{N}}$  always stabilises after a finite number of steps. We refer to [Cousot and Cousot 1977; Nielson et al. 1999] for a detailed explanation of widening and merely establish the correctness of our choice.

**FACT 7.1.**  $\nabla_{\mathfrak{N}}$  is a widening operator, i.e.  $N_1 \sqcup_{\mathfrak{N}} N_2 \leq_{\mathfrak{N}} N_1 \nabla_{\mathfrak{N}} N_2$  and for any sequence  $(N_i)_i$  the sequence  $(N'_i)_i$  defined by  $N'_0 = N_0$  and  $N'_{i+1} = N'_i \nabla_{\mathfrak{N}} N_{i+1}$  is non-decreasing and eventually stabilises.

*The Granularity Function.* We now return to the choice of a *granularity function*  $H : \mathfrak{N} \rightarrow \mathfrak{H}$ , a concept introduced in [Nielson and Nielson 2009] and adapted here for the use in line 8 of Table VIII. The function  $H$  abstracts  $\mathfrak{N}$  by mapping it into some space  $\mathfrak{H}$ , hence potentially reducing the size of the state space.

Recall that  $\mathfrak{N} = \mathbf{Lab} \rightarrow \mathfrak{J}$  and that while the set  $\mathbf{Lab}$  of labels is infinite, the set of labels occurring in a given program let  $A_1 \triangleq P_1; \dots; A_k \triangleq P_k$  in  $P_0$  is a finite set  $\mathbf{Lab}_{\text{fin}}$ . A granularity function is called *finitary* whenever for each finite set  $\mathbf{Lab}_{\text{fin}}$  there exists a finite subset  $\mathfrak{H}_{\text{fin}} \subseteq \mathfrak{H}$  such that  $H$  specialises to have functionality  $H : (\mathbf{Lab}_{\text{fin}} \rightarrow \mathfrak{J}) \rightarrow \mathfrak{H}_{\text{fin}}$ .

A granularity function  $H$  is called *stable* whenever the following property holds:

$$H(\hat{s}_1) = H(\hat{s}_2) \text{ implies } H(\hat{s}_1 \nabla_{\mathfrak{N}} \hat{s}_2) = H(\hat{s}_i) \text{ for } i = 1, 2$$

Finitary and stable granularity functions are of interest because they ensure termination of our algorithm:

**THEOREM 7.2.** *If the granularity function  $H$  is finitary and stable, then the algorithm of Table VIII always terminates.*

**PROOF.** We proceed by contradiction. So let us fix a finite set  $\mathbf{Lab}_{\text{fin}}$  as appropriate for the program considered and let us consider a non-terminating execution of the worklist algorithm of Table VIII. It is immediate that line 3 must be executed infinitely often and hence that the sequence  $(\hat{S}_i)_{i \in \mathbb{N}}$  exists.

We first show that the sequence  $(H(\hat{S}_i))_{i \in \mathbb{N}}$  grows in a non-decreasing manner. When a new element is added to  $\hat{S}_{i+1}$  in line 11 this is immediate. When a new element is added to  $\hat{S}_{i+1}$  in line 10 this follows from our assumption that  $H$  is stable.

We next show that the sequence  $(H(\hat{S}_i))_{i \in \mathbb{N}}$  eventually stabilises. But this is immediate given our assumption that  $H$  is finitary. Write  $i_0$  for the value of  $i$  at which  $(H(\hat{S}_i))_{i \geq 0}$  stabilises and consider  $i > i_0$ . From this point onwards line 8 always evaluates to true and lines 9 and 10 will be executed in each iteration.

We next show that the else branch of line 9 must be executed infinitely often. For this we proceed by contradiction, as otherwise there is some  $i_1 \geq i_0$  after which only the then branch is executed. But then the set  $\hat{S}_i$  has stabilised and hence the worklist  $W$  will get smaller in each iteration until it eventually becomes empty and the algorithm terminates. This would constitute a contradiction.

Hence there must be one value  $h$  such that we construct a strictly increasing sequence  $(\hat{t}_j)_{j \geq 0}$  by  $\hat{t}_0 = \hat{t}'_0$  and  $\hat{t}_{j+1} = \hat{t}_j \nabla_{\mathfrak{N}} \hat{t}'_{j+1}$  (for some sequence  $(\hat{t}'_j)_{j \geq 0}$ ) such that  $H(\hat{t}_j) = h$  for all values of  $j$ . However, this contradicts the fact that  $\nabla_{\mathfrak{N}}$  is a widening operator. This finishes our proof by contradiction.  $\square$

Stability of a granularity function also enables us to establish the following injectivity property. It ensures that the algorithm only includes one representative from each of the equivalence classes defined by the granularity function  $H$  (and hence that the choice of  $\hat{t}'$  in line 9 is unique whenever it exists).

**THEOREM 7.3.** *If the granularity function  $H$  is stable and the algorithm of Table VIII terminates and constructs the modal transition system  $\hat{T}_0 = (\hat{S}, A, \dashrightarrow, \longrightarrow)$  then we have the following injectivity property:*

$$\forall \hat{s}_1, \hat{s}_2 \in \hat{S}_i : H(\hat{s}_1) = H(\hat{s}_2) \Rightarrow \hat{s}_1 = \hat{s}_2$$

**PROOF.** We shall show that the formula is an invariant at line 3 of the worklist algorithm of Table VIII. If a new state  $\hat{t}$  is added in line 11 of Table VIII then it is trivially maintained. If on the other  $H(\hat{t}') = H(\hat{t})$  holds at line 8, then the substitutions in line 10 ensure that both  $\hat{t}$  and  $\hat{t}'$  are replaced by a single state  $\hat{u}$  which is either equal to  $\hat{t}'$  or to  $\hat{t}' \nabla_{\mathfrak{N}} \hat{t}$ . In both cases  $H(\hat{u}) = H(\hat{t}) = H(\hat{t}')$  holds, where in case of the widening we use that  $H$  is stable. By the invariant we know that  $H(\hat{u}) = H(\hat{s}')$  fails for any other state  $\hat{s}'$  and therefore the invariant is maintained.  $\square$

As an example, consider the following family of granularity functions  $H_{i,j}$ , where

$i, j \in \mathbb{N}$  and  $i \leq j$ :

$$\begin{aligned} \mathbf{H}_{i,j}(\hat{s}) = & \{(\ell, [m, n]) \mid \hat{s}(\ell) = [m, n], i \leq m, n \leq j\} \\ & \cup \{(\ell, [m, \infty]) \mid \hat{s}(\ell) = [m, n], i \leq m, n > j\} \\ & \cup \{(\ell, [0, n]) \mid \hat{s}(\ell) = [m, n], i > m, n \leq j\} \\ & \cup \{(\ell, [0, \infty]) \mid \hat{s}(\ell) = [m, n], i > m, n > j\} \end{aligned}$$

For instance,  $\mathbf{H}_{1,1}$  focuses on lower and upper bounds being either 0 or 1, thus abstracting from the actual counts. In this way,  $\hat{s}[\ell \mapsto [1, 2]]$  and  $\hat{s}[\ell \mapsto [2, \infty]]$  would both be represented by the same state, while  $\hat{s}[\ell \mapsto [0, 2]]$  would not. This also illustrates that a choice of  $i > 0$  in  $\mathbf{H}_{i,j}$  helps to find good underapproximations since a state where some action is necessarily exposed is not merged with those states where the same action is only possibly exposed (compare with function  $\mathbf{L}_{\hat{s}}$  on page 21).

**FACT 7.4.** *The granularity function  $\mathbf{H}_{i,j}$  is finitary as well as stable for all choices of  $i, j \in \mathbb{N}$  satisfying  $i \leq j$ .*

*Example 7.5.* For the program of Example 3.1 and granularity function  $\mathbf{H}_{1,1}$ , the worklist algorithm produces the modal abstraction shown in Figure 2 (where we do not show the “may” edges corresponding to the “must” edges shown). Note that the two “must” edges originating from  $\hat{s}_1$  express that, after synchronising on actions  $a^1/\bar{a}^3$ , the system is in a state where it can both synchronise on  $r^2/\bar{r}^4$  and perform a  $\tau$ -action  $\tau^5$ . Furthermore the “may” edge originating from  $\hat{s}_2$  describes that after executing  $\tau^5$  we cannot be sure whether the lock is released using the synchronisation of the actions  $r^2$  and  $\bar{r}^6$ , or whether the process is stuck. This is indeed as we would expect.

### Correctness

The following result states the correctness of the modal abstraction produced by the worklist algorithm with respect to Definition 2.2.

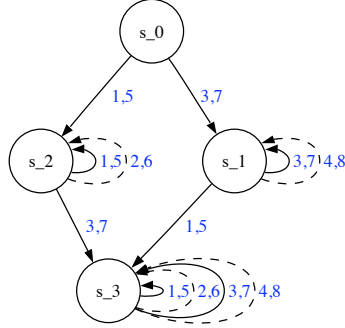
**THEOREM 7.6.** *Suppose that the algorithm of Table VIII terminates and produces a modal transition system  $\hat{T} = (\hat{S}, \mathbf{ALL}, \dashrightarrow, \longrightarrow) : \mathfrak{R}$  for an input process let  $A_1 \triangleq P_1; \dots; A_k \triangleq P_k$  in  $P_0$ . Then  $\hat{T}$   $\beta$ -represents the labelled transition system of let  $A_1 \triangleq P_1; \dots; A_k \triangleq P_k$  in  $P_0$  where  $\beta$  is defined by  $\beta(P) = \lfloor \mathcal{E}_\star \llbracket P \rrbracket \rfloor$ .*

**PROOF.** With respect to Definitions 2.2 and 2.1 consider a state  $\hat{s} \in \hat{S}$  and a process  $P \in \{P \mid P_0 \rightarrow^* P\}$  such that  $\beta(P) \leq_{\mathfrak{R}} \hat{s}$ , i.e. such that  $\lfloor \mathcal{E}_\star \llbracket P \rrbracket \rfloor \leq_{\mathfrak{R}} \hat{s}$ .

Consider the last time where the state  $\hat{s}$  was removed from the worklist  $W$  in line 3 of the worklist algorithm of Table VIII, and let  $\hat{S}_i$  denote the corresponding value of the set of current states.

In order to prove the “may” part required by Definitions 2.2 and 2.1 we proceed as follows. Consider some  $P \xrightarrow{\bar{\ell}} Q$  and observe that  $(\bar{\ell}, b) \in \mathbf{enabled}_{P_0}(\hat{s})$  with  $b \geq 1/2$  using Lemma 6.4 (1) and the fact that  $\mathbf{enabled}_{P_0}$  is monotonic, and hence that  $\bar{\ell}$  is selected for consideration in line 4. By Corollary 5.5 and the fact that  $\mathbf{transfer}_{P_0, \bar{\ell}}$  is monotonic, it follows that line 5 produces  $\hat{t}$  such that  $\beta(Q) \leq_{\mathfrak{R}} \hat{t}$ .

In line 6 of the algorithm, an edge  $\hat{s} \dashrightarrow_{\bar{\ell}} \hat{t}$  will be inserted in the current edge set, and in line 10 this edge might only be updated to  $\hat{s} \dashrightarrow_{\bar{\ell}} \hat{u}$  with  $\hat{t} \leq_{\mathfrak{R}} \hat{u}$ . Line 6


 Fig. 3. Modal transition system for unbounded data structure –  $H_{0,0}$ -abstraction

will not subsequently remove  $\hat{s} \dashrightarrow_{\tilde{\ell}} \hat{u}$  from  $\dashrightarrow$  because we assumed that there are no subsequent choices of  $\hat{s}$  in line 3. This establishes the “*may*” part.

Now to the proof of the “*must*” part. Whenever  $\hat{s} \longrightarrow_{\tilde{\ell}} \hat{u}$  we know that we have executed line 7 with  $b = 1$ . Thus  $(\tilde{\ell}, 1) \in \text{enabled}_{P_0}(\hat{s})$  in line 4. Using Lemma 6.4 (2) and the fact that  $\text{enabled}_{P_0}$  is monotonic we know that there exists a process  $Q$  such that  $P \rightarrow_{\tilde{\ell}} Q$ . Using the proof of the “*may*” part of this theorem, it is also immediate that  $\beta(Q) \leq_{\mathfrak{M}} \hat{u}$ , and this concludes the proof.  $\square$

## 7.2 Examples

We present three examples to illustrate different aspects of the construction of modal abstractions by the worklist algorithm. The first example focuses on processes with infinite behaviour and the ability of the algorithm to produce abstractions of varying degrees of coarseness for them. In the second example we consider Milner’s well-known scheduler algorithm [Milner 1999] and show that our analysis can be used to pinpoint implementations that cannot be correct with respect to the specification. The third example investigates the precision of the algorithm by contrasting it with our previous work on purely overapproximating abstractions [Nielson and Nielson 2007a].

*Unbounded Data Structures.* We consider a CCS program describing a data structure that can hold items of two different types. Using the actions  $putA$  and  $putB$  we model that items of type  $A$  and  $B$ , respectively, can be put into the data structure. Likewise, items can be retrieved again using the actions  $getA$  and  $getB$ . The CCS program, consisting of the process  $P$  describing the data structure and a user  $U$  that accesses it, can be modelled as follows:

$$\begin{aligned} \text{let } P &\triangleq putA^1.(getA^2.0 \mid P) + putB^3.(getB^4.0 \mid P) \\ U &\triangleq \overline{putA}^5.(\overline{getA}^6.0 \mid U) + \overline{putB}^7.(\overline{getB}^8.0 \mid U) \\ \text{in } P &\mid U \end{aligned}$$

Note that the modelled data structure is unbounded and can thus hold arbitrarily many items of each type. Therefore the concrete transition system corresponding to the process is infinite.

We consider abstractions produced by our algorithm using the family of gran-

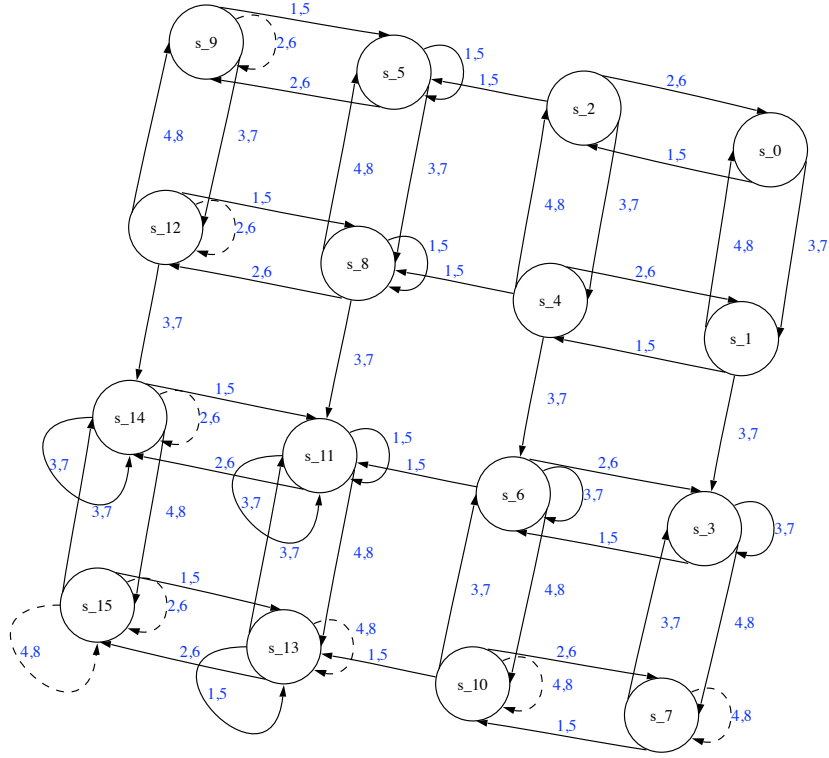


Fig. 4. Model transition system for unbounded data structure –  $H_{1,1}$ -abstraction

ularity functions  $H_{i,j}$  defined on page 28. Figure 3 shows the modal abstraction produced using  $H_{0,0}$ , the coarsest abstraction that can be obtained using this family of granularity functions. Starting in the abstract state  $s_0$  representing the initial process, the user can decide to either put an element of type  $A$  (transition with label  $1,5$ ) or type  $B$  (transition with label  $3,7$ ). Then the elements may be removed again (transitions with label  $2,6$  and  $4,8$ , respectively), or other elements can be inserted.

Note that using the  $H_{0,0}$ -abstraction, it is possible to show the property ( $P1$ ) “it is always possible to execute actions  $putA$  and  $putB$ ”, because there are “*must*”-edges for transitions with label  $1,5$  and  $3,7$  out of every state. However, because of the occurring “*may*”-edges for transitions labeled  $2,6$ , this abstraction is too coarse-grained to show the property ( $P2$ ) “it is always possible to execute  $getA$  at least once after having executed  $putA$ ”.

In this case we have to resort to a finer grained abstraction: Figure 4 shows the modal abstraction produced using  $H_{1,1}$ , which is precise enough to show the second property, because it expresses that there is a “*must*”-edge for transition  $2,6$  in every state reachable by a path containing a transition  $1,5$  not followed by any transition  $2,6$ . The following intuition helps to understand the abstraction, where

$$\begin{aligned}
 \text{let } P_1 & \triangleq \overline{a_1^1}. \overline{b_1^2}. P_1 \\
 P_2 & \triangleq \overline{a_2^3}. \overline{b_2^4}. P_2 \\
 \text{Sched}_{1,\{\}} & \triangleq a_1^5. \text{Sched}_{2,\{1\}} \\
 \text{Sched}_{1,\{1\}} & \triangleq b_1^6. \text{Sched}_{1,\{\}} \\
 \text{Sched}_{1,\{2\}} & \triangleq b_2^7. \text{Sched}_{1,\{\}} + a_1^8. \text{Sched}_{2,\{1,2\}} \\
 \text{Sched}_{1,\{1,2\}} & \triangleq b_1^9. \text{Sched}_{1,\{2\}} + b_2^{10}. \text{Sched}_{1,\{1\}} \\
 \text{Sched}_{2,\{\}} & \triangleq a_2^{11}. \text{Sched}_{1,\{2\}} \\
 \text{Sched}_{2,\{1\}} & \triangleq b_1^{12}. \text{Sched}_{2,\{\}} + a_2^{13}. \text{Sched}_{1,\{1,2\}} \\
 \text{Sched}_{2,\{2\}} & \triangleq b_2^{14}. \text{Sched}_{2,\{\}} \\
 \text{Sched}_{2,\{1,2\}} & \triangleq b_1^{15}. \text{Sched}_{2,\{2\}} + b_2^{16}. \text{Sched}_{2,\{1\}} \\
 \text{in } P_1 & | P_2 | \text{Sched}_{1,\{\}}
 \end{aligned}$$

 Table IX. Specification of Milner’s scheduler in the case  $n = 2$ .

we interpret Figure 4 as a 4-by-4 matrix of abstract states: the rightmost column of states (s\_0, s\_1, s\_3, and s\_7) describes states where no action  $getA$  is exposed; the second column from the right describes states where exactly one  $getA$  is exposed; the third, where at least one  $getA$  is exposed; and the leftmost column, where we don’t know how many actions  $getA$  are exposed. Analogously, the rows of states from the top to the bottom of the figure describe the different numbers of actions  $getB$  which are exposed.

We might also like to show that the data structure does not behave like a set, i.e. (P3) “it is sometimes possible to execute  $getA$  twice in a row”. Once again, the  $H_{1,1}$ -abstraction turns out to be too coarse to prove the property, but indeed a  $H_{2,2}$ -abstraction (not shown) suffices as it preserves label counts up until 2.

*Milner’s scheduler.* The idea of Milner’s scheduler [Milner 1999] can be summarized as follows: We assume to have a set of  $n$  processes, each of which performs a certain task repeatedly. A process  $P_i$  signals the start of task  $i$  by performing the action  $\overline{a_i}$ , and the end of the task by  $\overline{b_i}$ . The process  $P_i$  for  $i = 1, \dots, n$  is specified as follows:

$$P_i \triangleq \overline{a_i}. \overline{b_i}. P_i$$

Another process, the *Scheduler*, has to ensure that the tasks are started in a cyclic order, i.e. has to offer actions  $a_i$  in the sequence  $a_1, \dots, a_n$ . While the task  $i$  is running, the scheduler must offer an action  $b_i$  to allow process  $P_i$  to signal completion of the task. The scheduler is specified as follows:

$$\begin{aligned}
 \text{let } \text{Sched}_{i,X} & \triangleq \begin{cases} \sum_{j \in X} b_j. \text{Sched}_{i,X-j} & (i \in X) \\ \sum_{j \in X} b_j. \text{Sched}_{i,X-j} + a_i. \text{Sched}_{i+1, X \cup \{i\}} & (i \notin X) \end{cases} \\
 \text{in } \text{Sched}_{1,\emptyset}
 \end{aligned}$$

We shall now specialise this to the case  $n = 2$  and label the process as shown in Table IX. (Larger values of  $n$  do not illustrate fundamentally new problems.) The modal transition system constructed in this case is displayed in Figure 5 and only contains “*must*”-edges.

Also [Milner 1999] discusses two ways of building the scheduler as a ring of identical processes. Again we shall specialise to the case of  $n = 2$  where the two

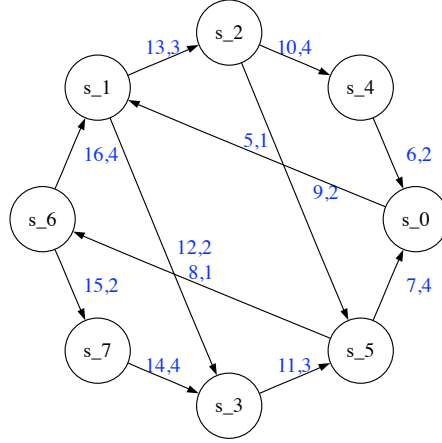


Fig. 5. The modal transition system for the specification of Milner's scheduler ( $n = 2$ ) –  $H_{1,1}$ -abstraction

$\text{let } P_1 \triangleq \overline{a_1^1}. \overline{b_1^2}. P_1$ $P_2 \triangleq \overline{a_2^3}. \overline{b_2^4}. P_2$ $A_1 \triangleq (a_1^5. C_1)$ $C_1 \triangleq (c_1^6. B_1)$ $B_1 \triangleq (b_1^7. D_1)$ $D_1 \triangleq (\overline{c_2^8}. A_1)$ $A_2 \triangleq (a_2^9. C_2)$ $C_2 \triangleq (c_2^{10}. B_2)$ $B_2 \triangleq (b_2^{11}. D_2)$ $D_2 \triangleq (\overline{c_1^{12}}. A_2)$ $\text{in } P_1 \mid P_2 \mid \text{new } c_1(\text{new } c_2(A_1 \mid D_2))$	$\text{let } P_1 \triangleq \overline{a_1^1}. \overline{b_1^2}. P_1$ $P_2 \triangleq \overline{a_2^3}. \overline{b_2^4}. P_2$ $A_1 \triangleq (a_1^5. C_1)$ $C_1 \triangleq (c_1^6. E_1)$ $E_1 \triangleq (b_1^7. D_1 + \overline{c_2^8}. B_1)$ $B_1 \triangleq (b_1^9. A_1)$ $D_1 \triangleq (\overline{c_2^{10}}. A_1)$ $A_2 \triangleq (a_2^{11}. C_2)$ $C_2 \triangleq (c_2^{12}. E_2)$ $E_2 \triangleq (b_2^{13}. D_2 + \overline{c_1^{14}}. B_2)$ $B_2 \triangleq (b_2^{15}. A_2)$ $D_2 \triangleq (\overline{c_1^{16}}. A_2)$ $\text{in } P_1 \mid P_2 \mid \text{new } c_1(\text{new } c_2(A_1 \mid D_2))$
--	--

Table X. Two implementations of Milner's scheduler ( $n = 2$ ).

processes are can be specified as in Table X. The corresponding modal transition systems are shown in Figure 6 and only contain “*must*”-edges.

As pointed out in [Milner 1999] the leftmost implementation is wrong whereas the rightmost is correct. Indeed this can easily be seen by comparing the modal transition systems. The channels  $c_1$  and  $c_2$  are only used for internal communication and we may therefore consider merging states that are only connected by transitions expressing communications over these channels; this gives the following correspondances between the states of Figures 5 and 6:

Figure 5:	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
Figure 6 left:	s_0	(s_1,s_2)	s_3	s_4	–	(s_5, s_6)	s_7	–
Figure 6 right:	s_0	(s_1,s_2)	(s_3,s_5)	s_4	s_7	(s_6,s_8)	(s_9,s_10)	s_11

The missing entries in the row for the modal transition system of the leftmost

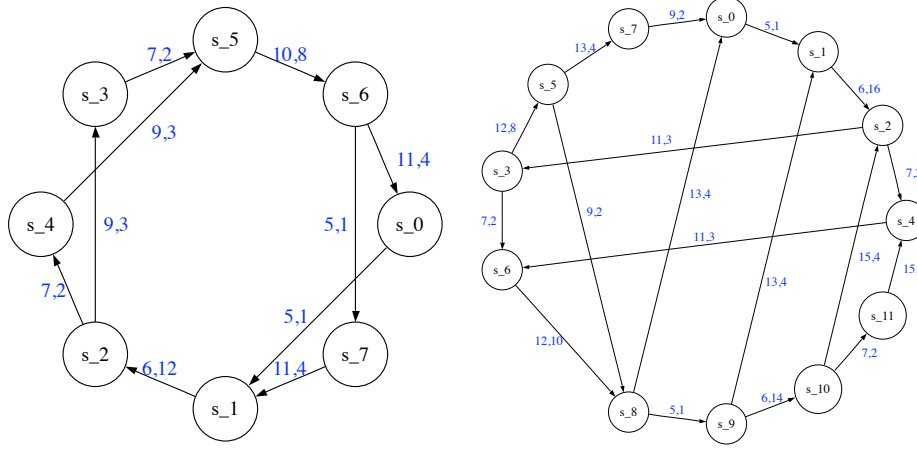


Fig. 6. The modal transition system for two implementations of Milner’s scheduler ( $n = 2$ ) –  $H_{1,1}$ -abstraction

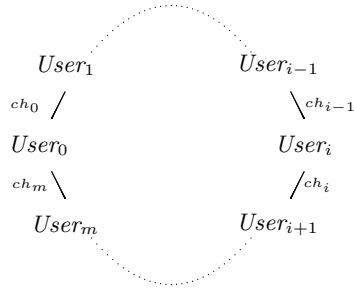


Fig. 7. The ITW key agreement protocol with  $m$  users

implementation clearly shows that it cannot be right and it is also easy to pinpoint the problem. The implementation to the right above is indeed correct; if the states are merged as suggested above the modal transition system to the right of Figure 6 becomes equal to that of Figure 5.

*The ITW Protocol.* We now consider a scalable specification of the Ingemarsson-Tang-Wong (ITW) key agreement protocol [Boyd and Mathuria 2003]. In the following we give a brief overview of the ITW protocol, show the results obtained by our analysis and give information about the performance of the algorithm.

The Ingemarsson-Tang-Wong (ITW) key agreement protocol is of special interest to us because we have dealt with it previously [Nielson and Nielson 2007a] using an approach that performs over-approximation only and that therefore can only produce “*may*” edges. We shall regard the extent to which our present algorithm can produce “*must*” edges as a test upon the precision of our algorithm.

The ITW protocol is a generalisation of the Diffie-Hellman key agreement protocol for establishing a joint secret key between a number of users. The idea is as follows:  $m$  users  $User_1, \dots, User_m$  are organised in a ring so that  $User_i$  only receives messages from  $User_{i-1}$  and only sends messages to  $User_{i+1}$  (with indices

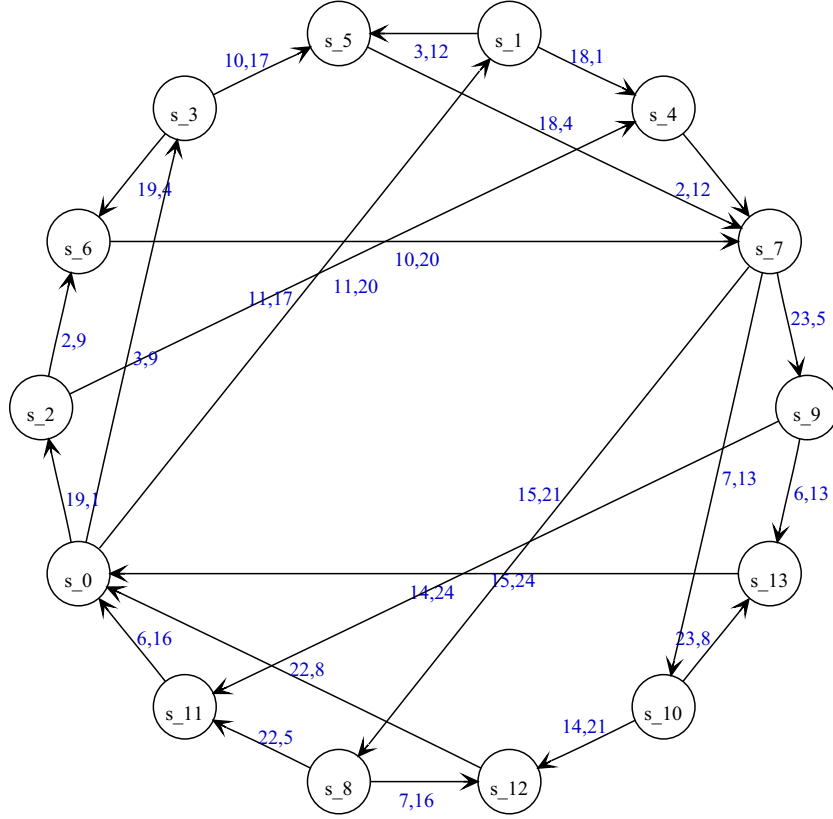


Fig. 8. The modal transition system for the ITW protocol with 3 users –  $H_{1,1}$ -abstraction

calculated modulo  $m$ ); this is illustrated in Figure 7. Initially, all the users have agreed upon a large prime  $p$  and a generator constant  $g$  that will be used to construct the shared key; preferably  $g$  is a primitive element in the field of integers modulo  $p$  such that inverses exist with respect to  $g$ . The protocol then proceeds in  $m - 1$  rounds. In the first round each of the users selects a random number  $r_i$ , raises  $g$  to the power of  $r_i$  and sends the value modulo  $p$  to its neighbour. Thus, in the first round  $User_i$  will receive  $g^{r_{i-1}} \bmod p$  from  $User_{i-1}$  and send  $g^{r_i} \bmod p$  to  $User_{i+1}$ . In the second round  $User_i$  will then raise the received value  $g^{r_{i-1}} \bmod p$  to the power of  $r_i$  to construct  $g^{r_i - 1 r_i} \bmod p$  and send it to  $User_{i+1}$ ; obviously  $User_{i+1}$  will receive a value constructed in a similar manner from  $User_{i-1}$  and uses this value in the next round. This process is repeated for  $m - 1$  rounds after which each user will know  $g^{r_1 \cdots r_m} \bmod p$ ; this value will then serve as the joint secret key of a symmetric crypto system.

*ITW in CCS.* To comply with the syntax of CCS we shall focus on the synchronisation structure of the protocol and hence ignore the calculations performed by the users and the actual values being communicated; this suffices for getting a

```

let User11 ≜ (ch01.(ch12.User12) + ch13.(ch04.User12))
    User12 ≜ (ch05.(ch16.User11) + ch17.(ch08.User11))
    User21 ≜ (ch19.(ch210.User22) + ch211.(ch112.User22))
    User22 ≜ (ch113.(ch214.User21) + ch215.(ch116.User21))
    User31 ≜ (ch217.(ch018.User32) + ch019.(ch220.User32))
    User32 ≜ (ch221.(ch022.User31) + ch023.(ch224.User31))
in User11 | User21 | User31.
    
```

Table XI. The code for the ITW protocol with 3 users

first understanding of the operation of the protocol. The users  $User_i$  and  $User_{i+1}$  (modulo  $m$ ) synchronise over channels  $ch_i$  and in each round they may interact with their neighbours in any order. Thus the code for  $User_i$  in round  $n$  (for  $n < m$ ) may be written as

$$User_i^n \triangleq ch_{i-1}. \overline{ch_i}. User_i^{n+1} + \overline{ch_i}. ch_{i-1}. User_i^{n+1}$$

and in the case of three users, where we let users recursively initiate a new run of a protocol as soon as the current run has been completed, we obtain the code in Table XI.

*The Modal Transition System for ITW.* Using an  $H_{1,1}$ -abstraction, the analysis constructs the modal transition system displayed in Figure 8 for the case of three users. Any path between initial state `s_0` and the final state `s_13` corresponds to a possible run of the protocol; the multitude of paths reflects the many possible interleavings that are made possible by the use of the summation in the definition of  $User_i^n$  above. Note that the  $H_{1,1}$ -abstraction is very precise in this case and computes that all edges are “*must*” edges (where for readability we once more do not display the “*may*” edges corresponding to the “*must*” edges).

In an earlier analysis of this protocol [Nielson and Nielson 2007a] all constructed edges corresponded to our notion of “*may*” edges because only over-approximation was performed. The fact that in this case all “*may*” edges can also be classified as “*must*” edges demonstrates that our use of widening and our choice of granularity functions allows us to obtain a usable amount of precision. As an example, the present analysis is able to demonstrate liveness of the ITW protocol which clearly could not be done based on the results of [Nielson and Nielson 2007a] (see also Example 8.3).

*The Implementation.* We have implemented the worklist algorithm in OCaml, and have used it to produce the modal transition systems for the running example, as well as the examples of this section.

We shall provide our benchmarks based on the Ingemarsson-Tang-Wong (ITW) key agreement protocol as it is easily scalable. Using the  $H_{1,1}$ -abstraction and a moderately powerful PC with a 2.4 GHz Core2 Duo processor and 2 GB of RAM, we obtain the performance results in Table XII. Here,  $m$  is the number of users in the ITW protocol, “#labels” the number of labels in the corresponding CCS term, “#states” and “#edges” the number of states and edges in the modal transition system, and “time” the running time of the algorithm.

$m$	#labels	#states	#edges	time
3	24	14	24	0.0s
4	48	57	120	0.0s
5	80	204	520	0.1s
6	120	705	2,100	0.7s
7	168	2,358	8,064	4.4s
8	224	7,749	29,904	26.0s
9	288	25,112	108,000	153.9s

Table XII. Performance of the worklist algorithm on the ITW protocol with  $m$  users

The results suggest practicability of the algorithm, but also its current limits given the size of the state space which grows exponentially in the number of labels (memory errors for more than 9 users). This is to be expected since the prototype implementation uses an explicit, rather than symbolic, encoding of the graph of the modal transition system. To be able to deal with substantially larger examples would necessitate a redevelopment of the implementation using symbolic techniques (BDDs). Naturally, by sacrificing precision (e.g. using a coarser abstraction than  $H_{1,1}$ , or a different labelling scheme) the size of the state space could be shrunk, then exposing a greater number of “*may*” edges.

## 8. LOGICAL REASONING ABOUT MODAL ABSTRACTIONS

In Sections 4–7 we have developed a technique to compute modal transition systems as abstractions of systems presented as processes in the process calculus CCS of Section 3. We have performed the development in the context of the general framework of labelled transition systems and modal transition systems developed in Section 2. The result is an algorithm for the construction of a finite modal transition system that represents the concrete transition system defined by the CCS process.

In this section we show that our construction of finite modal transition systems is suitable as a basis for model checking modal formulae over modal structures. Interpretations of temporal logics like  $CTL^*$  and the *modal  $\mu$ -calculus* on 3-valued structures have certainly been considered before, e.g. [Dams et al. 1997; Bruns and Godefroid 1999; Shoham and Grumberg 2007; Grumberg et al. 2007], and our development should therefore be seen as a “proof of concept” development rather than a substantial development of model checking in its own right; indeed, our prototype does not incorporate a model checking algorithm.

For our development we shall use a fragment of Action Computation Tree Logic ( $ACTL^*$ ) [De Nicola and Vaandrager 1990]. We reconsider the example of Unbounded Data Structures from Subsection 7.2 and show how our choice of granularity function  $H_{i,j}$  determines how precise answers we can get for the properties ( $P1$ ), ( $P2$ ) and ( $P3$ ) studied there.

$P \models \mathbf{tt}$	iff true
$P \models \ell$	iff $\ell \in \text{dom}(\mathcal{E}_\star[[P]])$
$P \models \neg\phi$	iff $P \not\models \phi$
$P \models \phi_1 \wedge \phi_2$	iff $P \models \phi_1$ and $P \models \phi_2$
$P \models \exists\gamma$	iff there exists a path $\Pi$ such that $\Pi(0) = P$ and there exists a number $k$ such that $ \Pi  \geq k$ and $\Pi \models_k \gamma$
$P \models \forall\gamma$	iff for all paths $\Pi$ such that $\Pi(0) = P$ there exists a number $k$ such that $ \Pi  \geq k$ and $\Pi \models_k \gamma$
$\Pi \models_k \mathbf{X}_\Omega \phi$	iff $k = 1$ and $\Pi[0] \in \Omega$ and $\Pi(1) \models \phi$
$\Pi \models_k \phi_1 \overline{\mathbf{U}}_\Omega \phi_2$	iff $\Pi(k) \models \phi_2$ and for all $i < k : \Pi[i] \in \Omega$ and $\Pi(i) \models \phi_1$

Table XIII. Interpretation of ActCTL formulae on processes

### 8.1 Interpretation of Formulae

The following grammar describes the syntax of state formulae  $\phi$  and path formulae  $\gamma$  in our fragment *ActCTL* of Action Computation Tree Logic:

$$\begin{aligned} \phi &::= \mathbf{tt} \mid \ell \mid \neg\phi \mid \phi \wedge \phi \mid \exists\gamma \mid \forall\gamma \\ \gamma &::= \mathbf{X}_\Omega \phi \mid \phi \overline{\mathbf{U}}_\Omega \phi \end{aligned}$$

Here,  $\ell$  is a single label, and  $\Omega$  is a set of (single or pairs of) labels, used to constrain the paths a formula is evaluated on. The operators  $\exists$  and  $\forall$  are path quantifiers,  $\mathbf{X}_\Omega$  is the *next* operator, and  $\overline{\mathbf{U}}_\Omega$  is the *until* operator we have chosen to include in our fragment (leaving out some more general versions to be found in Action Computation Tree Logic and using overlining to avoid any confusion).

We shall first devise a 2-valued interpretation (in  $\mathbb{B}$ ) over the labelled transition systems of Sections 2 and 3 and next devise a 3-valued interpretation (in  $\mathbb{B}_3$ ) over the modal transition systems of Sections 2 and 7. The 3-valued interpretation coincides with the 2-valued interpretation in case the modal transition system takes the form of a labelled transition system (i.e. “*may*” and “*must*” transitions coincide). As we shall see in Example 8.1 the definition of the 3-valued interpretation requires special care due to the presence of maximal paths that are *finite* and have “*may*” transitions – recall, that a “*may*” transition means that the transition may be either present or absent in the represented concrete labelled transitions systems. This relates to our choice of dealing with transition systems that have stuck configurations unlike the more usual treatment based on Kripke structures (e.g. [Clarke et al. 1999]) or transition systems without terminal states (e.g. [Baier and Katoen 2008]); indeed the formulae  $\forall \mathbf{X}_\Omega \phi$  and  $\neg(\exists \mathbf{X}_\Omega (\neg\phi))$  are not equivalent.

The 2-valued interpretation of formulae defines the meaning of the Boolean judgement  $P \models \phi$ , for the satisfaction of a state formula by the process  $P$ , and the Boolean judgement  $\Pi \models \gamma$  for the satisfaction of a path formulae by the path  $\Pi$ . We write  $\Pi$  for process paths of the form  $(P_0, \tilde{\ell}_0, P_1, \tilde{\ell}_1, \dots)$  where  $\Pi(i) \rightarrow_{\tilde{\ell}_i} \Pi(i+1)$  for  $i \geq 0$ , with  $\Pi(i)$  standing for  $P_i$ , and  $\Pi[i]$  for  $\tilde{\ell}_i$ . Process paths are always assumed to be *maximal*, i.e. infinite unless they are stuck in which case they end in a state; as a special case we have the path  $(P)$  in case  $P$  is stuck, i.e. has no outgoing transitions.

The 2-valued interpretation of formulae on processes is defined in Table XIII. For the state formulae not involving path formulae this is mostly straightforward. The clause for labels  $\ell$  (dealing with our basic predicates) evaluates to true on a

process  $P$  if  $\ell$  is exposed in  $P$ . For path formulae the interpretation of the labels of the transitions are taken into account. We use the subscript  $k$  in  $\Pi \models_k \gamma$  to indicate the length of the prefix of  $\Pi$  needed to demonstrate that  $\gamma$  holds. This allows us to be very careful in the interpretation of state formulae that do involve path formulae in that we are explicit about the path being sufficiently long. This is a complication not usual for CTL since the Kripke structures used there are usually required only to have infinite paths, as opposed to finite paths that are stuck. The formula “there exists a number  $k$  such that  $|\Pi| \geq k$  and  $\Pi \models_k \gamma$ ” thus corresponds to what is conventionally written  $\Pi \models \gamma$ .

The 3-valued interpretation of formulae defines the meaning of the 3-valued judgement  $[\hat{s} \models^3 \phi]$ , for the satisfaction of a state formula by an abstract state  $\hat{s}$ , and the 3-valued judgement  $[\hat{\pi} \models^3 \gamma]$  for the satisfaction of path formulae by the path  $\hat{\pi}$ . We write  $\hat{\pi}$  for state paths of the form  $(\hat{s}_0, \tilde{\ell}_0, \hat{s}_1, \dots)$  where  $\hat{\pi}(i) \xrightarrow{\tilde{\ell}_i} \hat{\pi}(i+1)$  for  $i \geq 0$ . State paths are always assumed to be *maximal* as explained above.

The 3-valued interpretation of formulae on abstract states is defined in Table XIV. For the state formulae not involving path formulae this is mostly a straightforward generalisation of Table XIII. In the clause for labels  $\ell$  we instead use the function  $L_{\hat{s}}$  of Section 6.1, which gives a 3-valued evaluation of the exposedness of the argument label in the state  $\hat{s}$ :  $L_{\hat{s}}(\ell) = 1$  if  $\hat{s}(\ell) = [m, n]$  and  $m > 0$ ,  $L_{\hat{s}}(\ell) = 1/2$  if  $\hat{s}(\ell) = [0, n]$  and  $n > 0$ ,  $L_{\hat{s}}(\ell) = 0$  if  $\hat{s}(\ell) = [0, 0]$ .

For path formulae the interpretation is once more a mostly straightforward generalisation of Table XIII. We allow to freely use 2-valued conditions within the 3-valued formula, e.g. in the interpretation of  $[\hat{\pi} \models_k^3 \mathbf{X}_\Omega \phi]$  neither  $k = 1$  nor  $\hat{\pi}[0] \in \Omega$  can give the indefinite truth value  $1/2$ . We define the predicate  $\text{must}_k$  on paths by

$$\text{must}_k(\hat{\pi}) \text{ iff } \hat{\pi}(i) \xrightarrow{\hat{\pi}[i]} \hat{\pi}(i+1) \text{ for all } i < k$$

to indicate whether or not the first  $k$  transitions of  $\hat{\pi}$  are “*must*” transitions.

In the interpretation of  $[\hat{s} \models^3 \exists \gamma]$  and  $[\hat{s} \models^3 \forall \gamma]$  it is always safe to give  $1/2$ , as will be made clear in Theorem 8.4 below, so we shall focus on motivating when we give one of the definite truth values.

To give 0 in the case of  $[\hat{s} \models^3 \exists \gamma]$  it suffices to show that no path can give anything but 0. To give 1 in the case of  $[\hat{s} \models^3 \exists \gamma]$  is not enough to merely find a path that gives 1 – we must also ensure that it uses enough “*must*” transitions in order to be sure that a corresponding path exists in the concrete system.

Next, to give 0 in the case of  $[\hat{s} \models^3 \forall \gamma]$  it is not enough to merely find a path that gives 0 – we must also ensure that it uses enough “*must*” transitions in order to be sure that a corresponding path exists in the concrete system. To give 1 in the case of  $[\hat{s} \models^3 \forall \gamma]$  it is not enough to show that all paths give 1 – we must also ensure that they use enough “*must*” transitions in order to be sure that we do not overlook a finite violating path in the concrete system. We shall clarify this definition below. (Indeed, if all paths  $\hat{\pi}$  with  $\hat{\pi}(0) = \hat{s}$  satisfy  $|\hat{\pi}| \geq k$ , we only have  $|\Pi| \geq k$  for all paths  $\Pi$  with  $\Pi(0) = P$  and  $\beta(P) \leq_{\mathfrak{N}} \hat{s}$  in case  $\text{must}_k(\hat{\pi})$  holds.)

*Example 8.1.* It is possible to rephrase the definitions of  $[\hat{s} \models^3 \exists \gamma]$  and  $[\hat{s} \models^3 \forall \gamma]$  in Table XIV using more intuitive (but slightly less precise) concepts.

We may say that  $[\hat{s} \models^3 \exists \gamma]$  gives 0 if there is no “*may* witness” for  $\gamma$ ; indeed a path is always a “*may* path” so it suffices to check that all such paths are counterex-

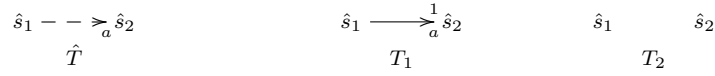


Fig. 9. Transition systems of Example 8.1

amples. Similarly, we may say that  $[\hat{s} \models^3 \exists \gamma]$  gives 1 if there is an “*must* witness” for  $\gamma$ ; here a path is a “*must* path” if  $\text{must}_k$  holds (for a suitable value of  $k$ ).

Dually, we may say that  $[\hat{s} \models^3 \forall \gamma]$  gives 0 if there is a “*must* counterexample” for  $\gamma$ . However, we do not say that  $[\hat{s} \models^3 \forall \gamma]$  gives 1 if there is no “*may* counterexample” — rather we roughly say that  $[\hat{s} \models^3 \forall \gamma]$  gives 1 if all witnesses are “*must* paths”.

To illustrate the point consider the abstract modal transition system  $\hat{T} = (\hat{S}, A, \dashrightarrow, \longrightarrow)$  having  $\hat{S} = \{\hat{s}_1, \hat{s}_2\}$  and  $A = \{a\}$  and the only transition being  $\hat{s}_1 \dashrightarrow_a \hat{s}_2$  (see Figure 9). Using our definition  $[\hat{s}_1 \models^3 \forall \mathbf{X}_\Omega \text{tt}]$  gives 1/2 rather than 1 because the path  $(\hat{s}_1, a, \hat{s}_2)$  is a “*may* path” rather than a “*must* path”.

To see that our choice is the desired one consider the labelled transition systems  $T_1 = (\hat{S}, A, \rightarrow^1)$  and  $T_2 = (\hat{S}, A, \rightarrow^2)$  with the only transition being  $\hat{s}_1 \rightarrow_a^1 \hat{s}_2$  (see Figure 9). Here  $\hat{s}_1 \models \forall \mathbf{X}_\Omega \text{tt}$  is true in  $T_1$  but false in  $T_2$ . Since  $\hat{T}$  is an *id*-representation of both  $T_1$  and  $T_2$  it follows that  $[\hat{s}_1 \models^3 \forall \mathbf{X}_\Omega \text{tt}]$  must give 1/2.

Intuitively, “duality” breaks down because “*may*” transitions may be both present and absent in the corresponding concrete labelled transition systems.

*Example 8.2.* Consider again the Unbounded Data Structures of Section 7.2, and the three properties that have been discussed informally. Using the abbreviation

$$\forall \overline{\mathbf{G}}_\Omega \phi \equiv \neg \exists (\text{tt} \overline{\mathbf{U}}_\Omega \neg \phi)$$

and writing  $\mathbf{ALL} = \mathbf{Lab} \cup (\mathbf{Lab} \times \mathbf{Lab})$  for the set of all labels, the properties (P1) “it is always possible to execute actions *putA* and *putB*”, (P2) “it is always possible to execute *getA* at least once after having executed *putA*”, and (P3) “it is sometimes possible to execute *getA* twice in a row” can be formalized using our ActCTL as follows:

$$\begin{aligned} (P1) & \quad \forall \overline{\mathbf{G}}_{\mathbf{ALL}} (\exists \mathbf{X}_{\{(1,5)\}} \text{tt} \wedge \exists \mathbf{X}_{\{(3,7)\}} \text{tt}) \\ (P2) & \quad \forall \overline{\mathbf{G}}_{\mathbf{ALL}} (\neg \exists \mathbf{X}_{\{(1,5)\}} (\forall \overline{\mathbf{G}}_{\mathbf{ALL} \setminus \{(2,6)\}} \neg (\exists \mathbf{X}_{\{(2,6)\}} \text{tt}))) \\ (P3) & \quad \exists (\text{tt} \overline{\mathbf{U}}_{\mathbf{ALL}} (\exists \mathbf{X}_{\{(2,6)\}} \exists \mathbf{X}_{\{(2,6)\}} \text{tt})) \end{aligned}$$

The result of the evaluation of the formulae in the state  $s_0$  on abstractions of varying precision confirms the intuition discussed in Section 7.2. Property (P1) evaluates to 1 on any  $\mathbf{H}_{i,j}$ -abstraction produced by the worklist algorithm, in particular also on the coarsest one ( $\mathbf{H}_{0,0}$ , see Figure 3). Property (P2) evaluates to 1/2 on the  $\mathbf{H}_{0,0}$ -abstraction, but to 1 on all abstractions which are more precise than  $\mathbf{H}_{1,1}$  (see Figure 4). Finally, property (P3) evaluates to 1 on  $\mathbf{H}_{2,2}$ - and finer-grained abstractions, and to 1/2 otherwise.

*Example 8.3.* Consider again the ITW protocol with three users from Section 7.2 and its corresponding modal transition system in Figure 8. Using the notation introduced in Example 8.2, we can verify that

$$[s_0 \models^3 \forall \overline{\mathbf{G}}_{\mathbf{ALL}} (\forall \mathbf{X}_{\mathbf{ALL}} \text{tt})] = 1$$

$[\hat{s} \models^3 \mathbf{tt}]$	=	1
$[\hat{s} \models^3 \ell]$	=	$\mathbf{L}_{\hat{s}}(\ell)$
$[\hat{s} \models^3 \neg\phi]$	=	$\neg^3([\hat{s} \models^3 \phi])$
$[\hat{s} \models^3 \phi_1 \wedge \phi_2]$	=	$[\hat{s} \models^3 \phi_1] \wedge^3 [\hat{s} \models^3 \phi_2]$
$[\hat{s} \models^3 \exists\gamma]$	=	$\begin{cases} 1 & \text{if there exists a path } \hat{\pi} \text{ such that } \hat{\pi}(0) = \hat{s} \text{ and} \\ & \text{there exists a number } k \text{ such that }  \hat{\pi}  \geq k \text{ and } [\hat{\pi} \models_k^3 \gamma] = 1 \text{ and } \mathbf{must}_k(\hat{\pi}) \\ 0 & \text{if for all paths } \hat{\pi} \text{ such that } \hat{\pi}(0) = \hat{s} \text{ and} \\ & \text{for all numbers } k \text{ such that }  \hat{\pi}  \geq k \text{ we have } [\hat{\pi} \models_k^3 \gamma] = 0 \\ 1/2 & \text{otherwise} \end{cases}$
$[\hat{s} \models^3 \forall\gamma]$	=	$\begin{cases} 1 & \text{if for all paths } \hat{\pi} \text{ such that } \hat{\pi}(0) = \hat{s} \\ & \text{there exists a number } k \text{ such that }  \hat{\pi}  \geq k \text{ and } [\hat{\pi} \models_k^3 \gamma] = 1 \text{ and } \mathbf{must}_k(\hat{\pi}) \\ 0 & \text{if there exists a path } \hat{\pi} \text{ such that } \hat{\pi}(0) = \hat{s} \text{ and} \\ & \text{for all numbers } k \text{ such that }  \hat{\pi}  \geq k \text{ we have } [\hat{\pi} \models_k^3 \gamma] = 0 \text{ and } \mathbf{must}_k(\hat{\pi}) \\ 1/2 & \text{otherwise} \end{cases}$
$[\hat{\pi} \models_k^3 \mathbf{X}_{\Omega} \phi]$	=	$k = 1 \wedge^3 \hat{\pi}[0] \in \Omega \wedge^3 [\hat{\pi}(1) \models^3 \phi]$
$[\hat{\pi} \models_k^3 \phi_1 \mathbf{U}_{\Omega} \phi_2]$	=	$[\hat{\pi}(k) \models^3 \phi_2] \wedge^3 \bigwedge_{i < k}^3 (\hat{\pi}[i] \in \Omega \wedge^3 [\hat{\pi}(i) \models^3 \phi_1])$

Table XIV. Interpretation of ActCTL formulae on states

holds, i.e. there are *no stuck paths* in the modal transition system.

In the next section we prove a soundness theorem which allows us to conclude that the property  $\forall \mathbf{G}_{\mathbf{ALL}} (\forall \mathbf{X}_{\mathbf{ALL}} \mathbf{tt})$  also holds on the corresponding labelled transition system, meaning that the protocol can never get stuck.

## 8.2 The Soundness Theorem

In preparation for the soundness theorem we lift the notion of  $\beta$ -representation (see Section 2) from states to paths by defining

$$\beta(\Pi) \sqsubseteq_{\mathfrak{N}}^k \hat{\pi} \text{ iff } |\Pi| \geq k \wedge |\hat{\pi}| \geq k \wedge \forall i \leq k : \beta(\Pi(i)) \leq_{\mathfrak{N}} \hat{\pi}(i) \wedge \Pi[i] = \hat{\pi}[i]$$

and can then concisely formulate the soundness result which informally expresses that a formula evaluating to a definite value on the abstract system allows for deciding about the truth and falsity of the same formula in the concrete setting.

**THEOREM 8.4.** *Suppose the modal transition system  $(\hat{S}, A, \dashrightarrow, \longrightarrow) : \mathfrak{N}$   $\beta$ -represents the labelled transition system associated with a process  $\text{let } A_1 \triangleq P_1; \dots; A_k \triangleq P_k$  in  $P_0$ . If  $\beta(P) \leq_{\mathfrak{N}} \hat{s}$  then  $[P \models \phi] \sqsubseteq^3 [\hat{s} \models^3 \phi]$ , and if  $\beta(\Pi) \sqsubseteq_{\mathfrak{N}}^k \hat{\pi}$  then  $[\Pi \models_k \gamma] \sqsubseteq^3 [\hat{\pi} \models_k^3 \gamma]$ .*

**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  $[\hat{s} \models^3 \phi] = 1/2$  or  $[\hat{\pi} \models_k^3 \gamma] = 1/2$ ; we therefore distinguish only the cases where these judgements evaluate to definite truth values.

**CASE  $\phi = \mathbf{tt}$ .** Clearly,  $[P \models \mathbf{tt}] \sqsubseteq^3 [\hat{s} \models^3 \mathbf{tt}]$ .

**CASE  $\phi = \ell$ .** If  $[\hat{s} \models^3 \ell] = 0$  then  $\hat{s}(\ell) = [0, 0]$ . Because  $\beta(P) \leq_{\mathfrak{N}} \hat{s}$ , we also have  $\ell \notin \text{dom}(\mathcal{E}_*[P])$ , and hence  $P \not\models \ell$ . On the other hand, if  $[\hat{s} \models^3 \ell] = 1$  then  $\hat{s}(\ell) = [m, n]$  with  $m, n > 0$ . Hence  $\ell \in \text{dom}(\mathcal{E}_*[P])$  and  $P \models \ell$ . Thus,  $[P \models \ell] \sqsubseteq^3 [\hat{s} \models^3 \ell]$ .

**CASE  $\phi = \neg\phi$ .** If  $[\hat{s} \models^3 \neg\phi] = 0$  then  $[\hat{s} \models^3 \phi] = 1$  because of the semantics of

$\neg^3$ . We can apply the induction hypothesis to obtain  $P \models \phi$  which is equivalent to  $P \not\models \neg\phi$ . The case  $[\hat{s} \models^3 \neg\phi] = 1$  is analogous.

CASE  $\phi = \phi_1 \wedge \phi_2$ . If  $[\hat{s} \models^3 \phi_1 \wedge \phi_2] = 0$  then  $[\hat{s} \models^3 \phi_1] = 0$  or  $[\hat{s} \models^3 \phi_2] = 0$ . By the induction hypothesis we thus have  $P \not\models \phi_1$  or  $P \not\models \phi_2$ , hence  $P \not\models \phi_1 \wedge \phi_2$ .

If  $[\hat{s} \models^3 \phi_1 \wedge \phi_2] = 1$  then  $[\hat{s} \models^3 \phi_1] = 1$  and  $[\hat{s} \models^3 \phi_2] = 1$ . By the induction hypothesis we thus have  $P \models \phi_1$  and  $P \models \phi_2$ , hence  $P \models \phi_1 \wedge \phi_2$ .

CASE  $\phi = \exists\gamma$ . If  $[\hat{s} \models^3 \exists\gamma] = 0$  then for all paths  $\hat{\pi}$  such that  $\hat{\pi}(0) = \hat{s}$  and all numbers  $k$  such that  $|\hat{\pi}| \geq k$  we have  $[\hat{\pi} \models_k^3 \gamma] = 0$ . We proceed by contradiction. Suppose there exists a path  $\Pi$  such that  $\Pi(0) = P$  and a number  $k$  such that  $\Pi \models_k \gamma$  and  $|\Pi| \geq k$ . By assumption we know  $\beta(\Pi(0)) \leq_{\mathfrak{N}} \hat{s}$ , and by Definitions 2.1 and 2.2 the path  $\Pi$  would be  $\beta$ -represented by some path  $\hat{\pi}'$  with  $\hat{\pi}'(0) = \hat{s}$  and  $|\hat{\pi}'| \geq k$ , and hence  $\beta(\Pi) \sqsubseteq_{\mathfrak{N}}^k \hat{\pi}'$ . By the induction hypothesis we have  $[\Pi \models_k \gamma] \sqsubseteq^3 [\hat{\pi}' \models_k^3 \gamma]$ , where we know that  $[\hat{\pi}' \models_k^3 \gamma] = 0$ . Hence  $\Pi \not\models_k \gamma$ , which is a contradiction. This establishes  $P \not\models_k \exists\gamma$ .

If  $[\hat{s} \models^3 \exists\gamma] = 1$  then there exists a path  $\hat{\pi}$  with  $\hat{\pi}(0) = \hat{s}$  and a number  $k$  such that  $|\hat{\pi}| \geq k$  and  $[\hat{\pi} \models^3 \gamma] = 1$  and  $\text{must}_k(\hat{\pi})$ , i.e. the first  $k$  transitions of  $\hat{\pi}$  are “*must*” transitions. Thus by Definitions 2.1 and 2.2 there exists a path  $\Pi$  with  $\Pi(0) = P$  such that  $\beta(\Pi) \sqsubseteq_{\mathfrak{N}}^k \hat{\pi}$ . By the induction hypothesis we have  $[\Pi \models_k \gamma] \sqsubseteq^3 [\hat{\pi} \models_k^3 \gamma]$ , and hence  $\Pi \models_k \gamma$  and  $P \models \exists\gamma$ .

CASE  $\phi = \forall\gamma$ . If  $[\hat{s} \models^3 \forall\gamma] = 0$  then there exists a path  $\hat{\pi}$  with  $\hat{\pi}(0) = \hat{s}$  and for all numbers  $k$  such that  $|\hat{\pi}| \geq k$  we have  $[\hat{\pi} \models^3 \gamma] = 0$  and  $\text{must}_k(\hat{\pi})$ , i.e. the first  $k$  transitions of  $\hat{\pi}$  are “*must*” transitions. Thus by Definitions 2.1 and 2.2 there exists a path  $\Pi$  with  $\Pi(0) = P$  such that for all numbers  $k$  with  $|\hat{\pi}| \geq k$  we have  $\beta(\Pi) \sqsubseteq_{\mathfrak{N}}^k \hat{\pi}$ . By the induction hypothesis we have  $[\Pi \models_k \gamma] \sqsubseteq^3 [\hat{\pi} \models_k^3 \gamma]$ , and hence  $\Pi \not\models_k \gamma$  for all  $k \leq |\hat{\pi}|$ . In the case where  $\hat{\pi}$  is an infinite path this suffices for establishing  $P \not\models \forall\gamma$ . In the case where  $\hat{\pi}$  is a finite path let  $k = |\hat{\pi}|$  and note that  $\hat{\pi}(k)$  is stuck. We know that  $\text{must}_k(\hat{\pi})$  and that  $|\Pi| \geq k$ . We now deduce that  $|\Pi| = k$  as otherwise  $\Pi(k) \rightarrow_{\Pi[k]} \Pi(k+1)$  in which case  $\hat{\pi}(k)$  cannot be stuck. This shows  $P \not\models \forall\gamma$  also in this case.

If  $[\hat{s} \models^3 \forall\gamma] = 1$  then for all paths  $\hat{\pi}$  such that  $\hat{\pi}(0) = \hat{s}$  there exists a number  $k$  such that  $|\hat{\pi}| \geq k$  and  $[\hat{\pi} \models_k^3 \gamma] = 1$  and  $\text{must}_k(\hat{\pi})$ , i.e. the first  $k$  transitions of  $\hat{\pi}$  are “*must*” transitions. We proceed by contradiction. Suppose there exists a path  $\Pi$  such that  $\Pi(0) = P$  and but there is no number  $k$  such that  $\Pi \models_k \gamma$  and  $|\Pi| \geq k$ . By assumption we know  $\beta(\Pi(0)) \leq_{\mathfrak{N}} \hat{s}$ , and by Definitions 2.1 and 2.2 the path  $\Pi$  would be  $\beta$ -represented by some path  $\hat{\pi}'$  with  $\hat{\pi}'(0) = \hat{s}$  and hence  $\beta(\Pi) \sqsubseteq_{\mathfrak{N}}^k \hat{\pi}'$  for all  $k \leq |\Pi|$ . For this path  $\hat{\pi}'$  we know that there exists a number  $k$  such that  $|\hat{\pi}'| \geq k$  and  $[\hat{\pi}' \models_k^3 \gamma] = 1$  and that  $\text{must}_k(\hat{\pi}')$ . It follows that  $k \geq |\Pi|$  and hence that  $\beta(\Pi) \sqsubseteq_{\mathfrak{N}}^k \hat{\pi}'$ . By the induction hypothesis we have  $[\Pi \models_k \gamma] \sqsubseteq^3 [\hat{\pi}' \models_k^3 \gamma]$ , where we know that  $[\hat{\pi}' \models_k^3 \gamma] = 1$ . Hence  $\Pi \models_k \gamma$ , which is a contradiction. This establishes  $P \models \forall\gamma$ .

CASE  $\gamma = \mathbf{X}_{\Omega} \phi$ . If  $[\hat{\pi} \models_k^3 \mathbf{X}_{\Omega} \phi] = 0$  then  $k \neq 1$  or  $\hat{\pi}[0] \notin \Omega$  or  $[\hat{\pi}(1) \models^3 \phi] = 0$ . If  $k \neq 1$  or  $\hat{\pi}[0] \notin \Omega$  then we have  $\Pi \not\models \mathbf{X}_{\Omega} \phi$  immediately. If  $[\hat{\pi}(1) \models^3 \phi] = 0$ , then by the induction hypothesis we have  $[\Pi(1) \models \phi] \sqsubseteq^3 [\hat{\pi}(1) \models^3 \phi]$ , and therefore  $\Pi \not\models_k \mathbf{X}_{\Omega} \phi$ .

If  $[\hat{\pi} \models_k^3 \mathbf{X}_{\Omega} \phi] = 1$  then  $k = 1$  and  $\hat{\pi}[0] \in \Omega$  and  $[\hat{\pi}(1) \models^3 \phi] = 1$ . By the induction

hypothesis we have  $[\Pi(1) \models \phi] \sqsubseteq^3 [\hat{\pi}(1) \models^3 \phi]$ , and therefore  $\Pi(1) \models \phi$ . Together with  $k = 1$  and  $\hat{\pi}[0] \in \Omega$  we have  $\Pi \models_k \mathbf{X}_\Omega \phi$ .

CASE  $\phi = \phi_1 \overline{\mathbf{U}}_\Omega \phi_2$ . If  $[\hat{\pi} \models_k^3 \phi_1 \overline{\mathbf{U}}_\Omega \phi_2] = 0$  then  $[\hat{\pi}(k) \models^3 \phi_2] = 0$  or  $\hat{\pi}[i] \notin \Omega$  or  $[\hat{\pi}(i) \models^3 \phi_1] = 0$  for some  $i < k$ . If  $[\hat{\pi}(k) \models^3 \phi_2] = 0$  then we have by the induction hypothesis that  $\Pi(k) \not\models \phi_2$  and therefore  $\Pi \not\models_k \phi_1 \overline{\mathbf{U}}_\Omega \phi_2$ . Likewise we can establish  $\Pi(i) \not\models \phi_1$  in the case that  $[\hat{\pi}(i) \models^3 \phi_1] = 0$ , or we have  $\hat{\pi}[i] \notin \Omega$  for some  $i < k$ , and in each of these cases  $\Pi \not\models_k \phi_1 \overline{\mathbf{U}}_\Omega \phi_2$ .

If  $[\hat{\pi} \models_k^3 \phi_1 \overline{\mathbf{U}}_\Omega \phi_2] = 1$  then  $[\hat{\pi}(k) \models^3 \phi_2] = 1$  and  $\hat{\pi}[i] \in \Omega$  and  $[\hat{\pi}(i) \models^3 \phi_1] = 1$  for all  $i < k$ . By applying the induction hypothesis, we thus get  $\Pi(k) \models \phi_2$  and  $\hat{\pi}[i] \in \Omega$  and  $\Pi(i) \models \phi_1$  for all  $i < k$ , which establishes  $\Pi \models_k \phi_1 \overline{\mathbf{U}}_\Omega \phi_2$ .  $\square$

Our *soundness* theorem corresponds to the notion of embedding theorem established in 3-valued shape analysis [Sagiv et al. 1999] and similarly in [Bruns and Godefroid 2000]. This perhaps also gives the best intuition for understanding why there is the “lack of symmetry” in our 3-valued interpretation as demonstrated in Example 8.1. Indeed, the 3-valued interpretation of a modal formula over a 3-valued structure is a formula directly expressed in terms of the formula and the 3-valued structure; the formula approximates the more precise result obtainable by combining the 2-valued interpretation of the modal formula over all the 2-valued structures described by the 3-valued structures. This corresponds to the development of “expected induced forms” approximating the “induced forms” in static analysis [Nielson 1985]; indeed, it is not algorithmically feasible to perform interpretations over a possible infinite set of structures. In our setting we interpret “*may*” transitions as transitions that may be present in some 2-valued structures and absent in other 2-valued structures and this gives rise to a weaker interpretation on “*may* paths” than in approaches based on mixed transition systems.

We expect that the model checking of our 3-valued interpretation of ActCTL can be performed by reducing it to a combination of 2-valued model checking problems in the manner of [Grumberg et al. 2007; Godefroid et al. 2001].

## 9. CONCLUSION

The main contribution of the paper is the development of an effective algorithm for constructing a finite modal transition system even if the corresponding concrete transition system is infinite; this is achieved by working directly on the syntactic presentation in the form of a process in a process calculus. We have also adapted the framework of concrete and abstract modal transition systems such that our development can be described as an abstraction of the concrete transition system, and we have shown the suitability of our development for interpreting a simple 3-valued modal logic over the modal transition system constructed.

The main part of our contribution was performed on Milner’s CCS that quite naturally gives rise to a labelled transition system working on process calculus terms. Processes were abstractly represented by interval representations of the multiplicity of exposed actions which allowed us to perform simultaneous over- and underapproximations which eventually allowed the 3-valued modal logic to validate as well as refute safety and liveness properties. The construction of the modal transition system used the approach of defining a transfer function of a Monotone Framework (including generalisations of generated and killed actions) that formed

the basis of a worklist algorithm whose termination was ensured by suitable uses of granularity functions and widening operators.

This work extends our previous works on extracting (overapproximating) finite transition systems, which in part address more powerful calculi, e.g. the pi-calculus [Nielson and Nielson 2007b], BioAmbients [Pilegaard et al. 2008] and broadcast calculi [Nanz et al. 2007]. In our future work we plan to extend these analyses and to make use of symbolic representation techniques in the construction of the modal transition systems. Furthermore we will try to extend the approach by using our technique within a counterexample-guided abstraction refinement (CEGAR) framework [Clarke et al. 2003; Gurfinkel and Chechik 2006]. Here, parametrised granularity functions could play an important role in finding a sequence of gradually finer abstractions until a certain property can be proved or refuted (as illustrated in Example 8.2).

#### ACKNOWLEDGMENTS

This work was supported in part by MT-LAB, a VKR Centre of Excellence. The main part of the work was carried out while the second author was supported by a H. C. Ørsted Stipend at DTU Informatics, Technical University of Denmark.

#### REFERENCES

- BAIER, C. AND KATOEN, J.-P. 2008. *Principles of Model Checking*. MIT Press.
- BALDAN, P., CORRADINI, A., AND KÖNIG, B. 2008. A framework for the verification of infinite-state graph transformation systems. *Information and Computation* 206, 7, 869–907.
- BALL, T., PODELSKI, A., AND RAJAMANI, S. K. 2001. Boolean and cartesian abstraction for model checking c programs. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*. Lecture Notes in Computer Science, vol. 2031. Springer, 268–283.
- BODEI, C., DEGANO, P., NIELSON, F., AND NIELSON, H. R. 1998. Control flow analysis for the pi-calculus. In *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*. Lecture Notes in Computer Science, vol. 1466. Springer, 84–98.
- BOYD, C. AND MATHURIA, A. 2003. *Protocols for Authentication and Key Establishment*. Springer.
- BRUNS, G. AND GODEFROID, P. 1999. Model checking partial state spaces with 3-valued temporal logics. In *Proceedings of the 11th International Conference on Computer Aided Verification (CAV'99)*. Lecture Notes in Computer Science, vol. 1633. Springer, 274–287.
- BRUNS, G. AND GODEFROID, P. 2000. Generalized model checking: Reasoning about partial state spaces. In *11th International Conference on Concurrency Theory (CONCUR'00)*. Lecture Notes in Computer Science, vol. 1877. Springer, 168–182.
- BUCHHOLTZ, M., NIELSON, H. R., AND NIELSON, F. 2004. A calculus for control flow analysis of security protocols. *International Journal of Information Security* 2, 3-4, 145–167.
- CHECHIK, M., DEVEREUX, B., EASTERBROOK, S. M., AND GURFINKEL, A. 2003. Multi-valued symbolic model-checking. *ACM Trans. Softw. Eng. Methodol.* 12, 4, 371–408.
- CLARKE, E. M., GRUMBERG, O., JHA, S., LU, Y., AND VEITH, H. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM* 50, 5, 752–794.
- CLARKE, E. M., GRUMBERG, O., AND LONG, D. E. 1994. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems* 16, 5, 1512–1542.
- CLARKE, E. M., GRUMBERG, O., AND PELED, D. A. 1999. *Model Checking*. MIT Press.
- COUSOT, P. AND COUSOT, R. 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'77)*. ACM, 238–252.

- COUSOT, P. AND COUSOT, R. 1979. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'79)*. ACM, 269–282.
- DAMS, D., GERTH, R., AND GRUMBERG, O. 1997. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems* 19, 2, 253–291.
- DAMS, D. AND NAMJOSHI, K. S. 2004. The existence of finite abstractions for branching time model checking. IEEE Computer Society, 335–344.
- DAMS, D. AND NAMJOSHI, K. S. 2005. Automata as abstractions. In *Proceedings of the 6th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'05)*. Lecture Notes in Computer Science, vol. 3385. 216–232.
- DE NICOLA, R. AND VAANDRAGER, F. W. 1990. Action versus state based logics for transition systems. In *Proceedings of the LITP Spring School on Semantics of Systems of Concurrent Processes*. Lecture Notes in Computer Science, vol. 469. 407–419.
- FECHER, H. AND SHOHAM, S. 2007. Local abstraction-refinement for the mu-calculus. In *Proceedings of the 14th International SPIN Workshop (SPIN'07)*. Lecture Notes in Computer Science, vol. 4595. Springer, 4–23.
- GODEFROID, P., HUTH, M., AND JAGADEESAN, R. 2001. Abstraction-based model checking using modal transition systems. In *Proceedings of the 12th International Conference on Concurrency Theory (CONCUR'01)*. Lecture Notes in Computer Science, vol. 2154. Springer, 426–440.
- GRAF, S. AND SAÏDI, H. 1997. Construction of abstract state graphs with pvs. In *9th International Conference on Computer Aided Verification (CAV'97)*. Lecture Notes in Computer Science, vol. 1254. Springer, 72–83.
- GRUMBERG, O., LANGE, M., LEUCKER, M., AND SHOHAM, S. 2007. When not losing is better than winning: Abstraction and refinement for the full  $\mu$ -calculus. *Information and Computation* 205, 8, 1130–1148.
- GURFINKEL, A. AND CHECHIK, M. 2006. Why waste a perfectly good abstraction? In *12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*. Lecture Notes in Computer Science, vol. 3920. Springer, 212–226.
- GURFINKEL, A., WEI, O., AND CHECHIK, M. 2006. Systematic construction of abstractions for model-checking. In *Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI 2006*, E. A. Emerson and K. S. Namjoshi, Eds. Lecture Notes in Computer Science, vol. 3855. Springer, 381–397.
- HUTH, M. 2005. On finite-state approximants for probabilistic computation tree logic. *Theor. Comput. Sci.* 346, 1, 113–134.
- HUTH, M., JAGADEESAN, R., AND SCHMIDT, D. A. 2001. Modal transition systems: A foundation for three-valued program analysis. In *Proceedings of the 10th European Symposium on Programming Languages and Systems (ESOP'01)*. Lecture Notes in Computer Science, vol. 2028. Springer, 155–169.
- JONSSON, B. AND LARSEN, K. G. 1991. Specification and refinement of probabilistic processes. In *Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science, 15-18 July, 1991, Amsterdam, The Netherlands*. IEEE Computer Society, 266–277.
- KAM, J. B. AND ULLMAN, J. D. 1977. Monotone data flow analysis frameworks. *Acta Inf.* 7, 305–317.
- KATOEN, J.-P., KLINK, D., LEUCKER, M., AND WOLF, V. 2007. Three-valued abstraction for continuous-time markov chains. In *19th International Conference on Computer Aided Verification (CAV'07)*. Lecture Notes in Computer Science, vol. 4590. Springer, 311–324.
- KATOEN, J.-P., KLINK, D., LEUCKER, M., AND WOLF, V. 2008. Abstraction for stochastic systems by erlang's method of stages. In *19th International Conference on Concurrency Theory (CONCUR'08)*. Lecture Notes in Computer Science, vol. 5201. Springer, 279–294.
- KLEENE, S. C. 1952. *Introduction to Metamathematics*. Biblioteca Mathematica, vol. 1. North-Holland.
- LARSEN, K. G. AND THOMSEN, B. 1988. A modal process logic. In *Proceedings of the Third Annual Symposium on Logic in Computer Science (LICS'88)*. IEEE Computer Society, 203–210.
- ACM Transactions on Computational Logic, Vol. V, No. N, M 20YY.

- LARSEN, K. G. AND XINXIN, L. 1990. Equation solving using modal transition systems. In *LICS*. IEEE Computer Society, 108–117.
- MILNER, R. 1989. *Communication and Concurrency*. Prentice Hall.
- MILNER, R. 1999. *Communicating and Mobile Systems: The pi-calculus*. Cambridge University Press.
- NANZ, S., NIELSON, F., AND NIELSON, H. R. 2007. Topology-dependent abstractions of broadcast networks. In *Proceedings of the 18th International Conference on Concurrency Theory (CONCUR'07)*. Lecture Notes in Computer Science, vol. 4703. Springer, 226–240.
- NANZ, S., NIELSON, F., AND NIELSON, H. R. 2008. Modal abstractions of concurrent behaviour. In *Proceedings of the 15th International Static Analysis Symposium (SAS'08)*. Lecture Notes in Computer Science, vol. 5079. Springer, 159–173.
- NIELSON, F. 1985. Expected forms of data flow analyses. In *Programs as Data Objects*. Lecture Notes in Computer Science, vol. 217. Springer, 172–191.
- NIELSON, F., NIELSON, H. R., AND HANKIN, C. 1999. *Principles of Program Analysis*. Springer.
- NIELSON, F., NIELSON, H. R., AND SAGIV, M. 2000. A Kleene analysis of mobile ambients. In *European Symposium on Programming (ESOP'00)*. Lecture Notes in Computer Science, vol. 1782. Springer, 305–319.
- NIELSON, F., NIELSON, H. R., AND SAGIV, M. 2001. Kleene’s logic with equality. *Information Processing Letters* 80, 131–137.
- NIELSON, H. R. AND NIELSON, F. 2002. Flow logic: A multi-paradigmatic approach to static analysis. *2566*, 223–244.
- NIELSON, H. R. AND NIELSON, F. 2007a. Data flow analysis for CCS. In *Program Analysis and Compilation. Theory and Practice*. Lecture Notes in Computer Science, vol. 4444. Springer.
- NIELSON, H. R. AND NIELSON, F. 2007b. A flow-sensitive analysis of privacy properties. *Proceedings of the 20th Computer Security Foundations Symposium (CSF'07)*, 249–264.
- NIELSON, H. R. AND NIELSON, F. 2009. A monotone framework for CCS. *Computer Languages, Systems & Structures* 35, 365–394. Article in press.
- NIELSON, H. R., NIELSON, F., AND PILEGAARD, H. 2004. Spatial analysis of bioambients. In *Proceedings of the 11th International Static Analysis Symposium (SAS'04)*. Lecture Notes in Computer Science, vol. 3148. Springer, 69–83.
- PILEGAARD, H., NIELSON, F., AND NIELSON, H. R. 2008. Pathway analysis for BioAmbients. *Journal of Logic and Algebraic Programming* 77, 1-2, 92–130.
- SAGIV, M., REPS, T., AND WILHELM, R. 1999. Parametric shape analysis via 3-valued logic. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'99)*. ACM, 105–118.
- SAGIV, M., REPS, T., AND WILHELM, R. 2002. Parametric shape analysis via 3-valued logic. *ACM Transactions on Programming Languages and Systems* 24, 3, 217–298.
- SCHMIDT, D. A. 2007. A calculus of logical relations for over- and underapproximating static analyses. *Science of Computer Programming* 64, 1, 29–53.
- SHOHAM, S. AND GRUMBERG, O. 2007. A game-based framework for CTL counterexamples and 3-valued abstraction-refinement. *ACM Transactions on Computational Logic* 9, 1.

Received February 2009; revised April 2010; accepted May 2010