Process Algebra for Modal Transition Systemses*

Nikola Beneš[1] and Jan Křetínský[1,2]

1 **Faculty of Informatics, Masaryk University**
Botanická 68a, 602 00 Brno, Czech Republic
xbenes3@fi.muni.cz, jan.kretinsky@fi.muni.cz
2 **Institut für Informatik, TU München**
Boltzmannstr. 3, D-85748, Garching, Germany

Abstract

The formalism of modal transition systems (MTS) is a well established framework for systems specification as well as abstract interpretation. Nevertheless, due to incapability to capture some useful features, various extensions have been studied, such as e.g. mixed transition systems or disjunctive MTS. Thus a need to compare them has emerged. Therefore, we introduce transition system with obligations as a general model encompassing all the aforementioned models, and equip it with a process algebra description. Using these instruments, we then compare the previously studied subclasses and characterize their relationships.

Keywords and phrases modal transition systems, process algebra, specification

Digital Object Identifier 10.4230/OASIcs.MEMICS.2010.9

1 Introduction

Design and verification of parallel systems is a difficult task for several reasons. Firstly, a system usually consists of a number of components working in parallel. Component based design thus receives much attention and *composition* is a crucial element to be supported in every reasonable specification framework for parallel systems. Secondly, the behaviour of the components themselves is not trivial. One thus begins the design process with an underspecified system where some behaviour is already prescribed and some may or may not be present. The specification is then successively refined until a real implementation is obtained, where all details of the behaviour are settled. Therefore, a need for support of *stepwise refinement* design arises. This is indispensable, either due to incapability of capturing all the required behaviour in the early design phase, or due to leaving a bunch of possibilities for the implementations, such as in e.g. product lines [6]. Modal transition systems is a framework supporting both these fundamental features.

Modal transition systems (MTS) is a specification formalism introduced by Larsen and Thomsen [7, 1] allowing both for stepwise refinement design of systems and their composition. A considerable attention has been recently paid to MTS due to many applications, e.g. component-based software development [9], interface theories [10], or modal abstractions and program analysis [5], to name just a few.

The MTS formalism is based on transparent and simple to understand model of *labelled transition systems* (LTS). While LTS has only one labelled transition relation between the states determining the behaviour of the system, MTS as a specification formalism is equipped with two types of transitions: the *must* transitions capture the required behaviour, which

* The word "Systemses" in the title is deliberate. Modal transition systems is a formalism. We consider several formalisms based on modal transition systems here.

Sixth Doctoral Workshop on Math. and Eng. Methods in Computer Science (MEMICS'10)—Selected Papers.
Editors: L. Matyska, M. Kozubek, T. Vojnar, P. Zemčík, D. Antoš; pp. 9–18

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

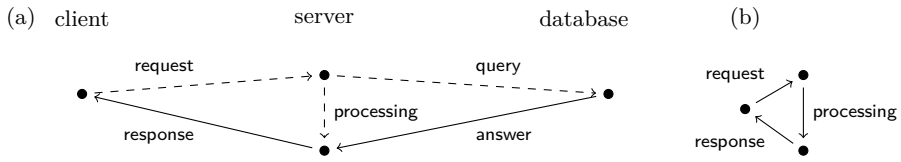

Figure 1 An example of (a) a modal transition system (b) its implementation

is present in all its implementations; the *may* transitions capture the allowed behaviour, which need not be present in all implementations. Such a system can be refined in two ways: a may transition is either implemented (and becomes a must transition) or omitted (and disappears as a transition). Figure 1 depicts an MTS that has arisen as a composition of three systems and specifies the following. A request from a client may arrive. Then we can process it directly or make a query to a database where we are guaranteed an answer. In both cases we send a response. On the right there is an implementation of the system where the processing branch is implemented and the database query branch is omitted. Note that in this formalism we can easily compose implementations as well as specifications.

While specifying may transitions brings guarantees on safety, liveness can be guaranteed to some extent using must transitions. Nevertheless, at an early stage of design we may not know which of several possible different ways to implement a particular functionality will later be chosen, although we know at least one of them has to be present. We want to specify e.g. that either processing or query will be implemented, otherwise we have no guarantee on receiving response eventually. Therefore, several formalisms extending MTS have been introduced. *Disjunctive modal transition systems* (DMTS) do not enforce a particular transition, but specify a whole set of transitions at least one of which must be present. (In our example, it would be the set consisting of processing and query transitions.) DMTS have been introduced in several flavours [8, 4, 2]. Another extension guaranteeing more structured requirements on the behaviour are *mixed transition systems* (MixTS) [3]. Here the required behaviour is not automatically allowed (not all must transitions are necessarily also may transitions) and it must be realized using other allowed behaviour. This corresponds to the situation where a new requirement can be implemented using some reused components. Moreover, it allows for some liveness properties as well. All in all, a need for more structured requirements has emerged. Therefore, we want to compare these formalisms and their expressive power.

We introduce *transition system with obligations* (OTS), a framework that encompasses all the aforementioned systems. Further, we introduce a new process algebra, since there was none for any of the discussed classes of systems. The algebra comes with the respective structural operational semantics, and thus enriches the ways to reason about all these systems. More importantly it allows us to obtain their alternative characterization and provide a more compact description language for them. Altogether, these two new tools allow us to compare all the variants of MTS and we indeed show interesting relationships among the discussed systems. We characterize the process algebra fragments corresponding to the various subclasses of OTS, such as MTS, MixTS or variants of DMTS. Since bisimulation is a congruence w.r.t. all operators of the algebra, this allows for modular analysis of the systems and also for practical optimizations based on minimization by bisimulation quotienting. Finally, since OTS allow to specify requirements in quite a general form, we can perform some important optimizations in the composition of systems. E.g., when composing DMTS we can avoid an additional exponential blowup that was unavoidable so far.

2 Preliminaries

In order to define the framework we will work in, we need a tool to handle complex requirements imposed on the systems. For this we use positive boolean formulae.

▶ **Definition 1.** A *positive boolean formula* over set X of atomic propositions is given by the following syntax:

$$\varphi ::= x \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{tt} \mid \mathbf{ff}$$

where x ranges over X. The set of all positive boolean formulae over X is denoted as $\mathcal{B}^+(X)$. The semantics $[\![\varphi]\!]$ of a positive boolean formula φ is a set of subsets of X satisfying φ. It is inductively defined as follows:

$$[\![x]\!] = \{Y \subseteq X \mid x \in Y\} \quad [\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!] \quad [\![\mathbf{tt}]\!] = 2^X \quad [\![\mathbf{ff}]\!] = \emptyset \quad [\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$$

Every positive boolean formula can be uniquely represented in conjunctive normal form (CNF). It can also be uniquely represented in disjunctive normal form (DNF). In the disjunctive normal form of φ, the disjuncts are precisely the minimal elements of $[\![\varphi]\!]$ (with set inclusion). The formulae $\mathbf{tt}$ and $\mathbf{ff}$ are never needed as proper subformulae of any other formula.

We now proceed with the definition of the systems that are general enough to capture features of all the systems that we discuss in the paper.

▶ **Definition 2.** A *transition system with obligations* (OTS) over an action alphabet Σ is a triple $(\mathcal{P}, \dashrightarrow, \mathbf{\Omega})$, where P is a set of *processes*, $\dashrightarrow \subseteq P \times \Sigma \times P$ is the *may* transition relation and $\mathbf{\Omega}\colon P \to \mathcal{B}^+(\Sigma \times P)$ is the set of *obligations*.

For simplicity we also require the systems to be finitely branching, i.e. for every $P \in \mathcal{P}$ there are only finitely many $P' \in \mathcal{P}$ with $(P, a, P') \in \dashrightarrow$ for some a. Nevertheless, we could easily drop this assumption if we allowed conjunctions and disjunctions of infinite arities.

Various subclasses of OTS have been studied. We list the most important ones and depict their syntactic relationships in Fig. 2.

- A *disjunctive modal transition system* (DMTS) [8] is an OTS where the must obligations are in CNF. An arbitrary OTS can thus be expressed as a DMTS. Indeed, as noted above, any formula can be translated into CNF. However, this can cost an exponential blowup.
- A *mixed transition system* (MixTS) [3] is an OTS where the must obligations are just conjunctions of atomic predicates.

Moreover, we can impose the following *consistency requirement*

$$\mathbf{\Omega}(S) \neq \mathbf{ff} \text{ and if } \mathbf{\Omega}(S) \text{ contains } (a, T) \text{ then } S \overset{a}{\dashrightarrow} T,$$

which guarantees that all required behaviour is also allowed. This gives rise to the following systems:

- A *consistent DMTS* (cDMTS) [2] is a DMTS satisfying the consistency requirement.
- A *modal transition system* (MTS) [7] is a MixTS satisfying the consistency requirement.
- A *labelled transition system* (LTS) is an MTS such that whenever $S \overset{a}{\dashrightarrow} T$ then $\mathbf{\Omega}(S) = (a, T) \wedge \varphi$ for some φ. Since all behaviour of an LTS is both allowed and required at the same time, we also call LTS an *implementation*.

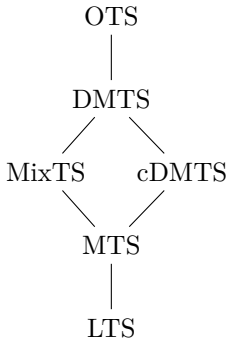

Figure 2 The syntactic hierarchy of MTS extensions

In order to define the refinement relation on the systems, we need the following auxiliary notion of refinement on formulae motivated by the following example.

▶ **Example 3.** Let us assume formulae $\varphi = (a \wedge b) \vee c$ and $\psi = A \vee C \vee D$. The renaming $R: \ a = A, c = C$ then guarantees that $\varphi \Rightarrow \psi$. This logical refinement (entailment) up to renaming is formalized in the following definition.

▶ **Definition 4.** Let $R \subseteq X \times X$, let $\varphi, \psi \in \mathcal{B}^+(X)$. We write $\varphi \sqsubseteq_R \psi$ to denote

$$\forall M \in [\![\varphi]\!] \ \exists N \in [\![\psi]\!] \ \forall n \in N \ \exists m \in M : (m, n) \in R$$

Note that if we take $R = id$, $\varphi \sqsubseteq_{id} \psi$ if and only if $\varphi \Rightarrow \psi$ (i.e. $[\![\varphi]\!] \subseteq [\![\psi]\!]$). Before proceeding to the fundamental definition of OTS, we prove the following lemmata that will be useful in later proofs. The first lemma is straightforward.

▶ **Lemma 5.** *Let $\varphi \in \mathcal{B}^+(X)$. Then $[\![\varphi]\!]$ is an upwards closed set in $(2^X, \subseteq)$.*

For the two following lemmata, assume this situation: Let X be an arbitrary set and let Y_x be an arbitrary finite set for all $x \in X$. Let $\varphi \in \mathcal{B}^+(X)$ and $\widehat{\varphi}$ be the formula that is created from φ by replacing all occurrences of x by $\bigvee Y_x$ (where $\bigvee \emptyset = \mathbf{ff}$).

▶ **Lemma 6.** *Let $Z \subseteq X$ and let $Z' \subseteq \bigcup_{z \in Z} Y_z$ such that for all $z \in Z$, there is some $y \in Y_z \cap Z'$. Then $Z \in [\![\varphi]\!]$ implies $Z' \in [\![\widehat{\varphi}]\!]$.*

Proof. The proof is done by induction on φ.

- The cases of $\varphi = \mathbf{tt}$ and $\varphi = \mathbf{ff}$ are trivial.
- $\varphi = x$. Then $\widehat{\varphi} = \bigvee Y_x$. $Z \in [\![\varphi]\!]$ implies $x \in Z$ and thus there is some $y \in Y_x \cap Z'$. Therefore $Z' \in [\![\widehat{\varphi}]\!]$ as $[\![\widehat{\varphi}]\!]$ contains $\{y\}$ and it is an upwards closed set.
- $\varphi = \psi \wedge \xi$, then $\widehat{\varphi} = \widehat{\psi} \wedge \widehat{\xi}$. Let $Z \in [\![\varphi]\!] = [\![\psi]\!] \cap [\![\xi]\!]$. Then $Z \in [\![\psi]\!]$ and $Z \in [\![\xi]\!]$. Due to the induction hypothesis, $Z' \in [\![\widehat{\psi}]\!]$ and $Z' \in [\![\widehat{\xi}]\!]$, thus also $Z' \in [\![\widehat{\psi} \wedge \widehat{\xi}]\!] = [\![\widehat{\varphi}]\!]$.
- The case of $\vee$ is similar to the previous case. ◀

▶ **Lemma 7.** *Let $Z' \subseteq \bigcup_x Y_x$ and let $Z = \{x \mid \exists y \in Y_x \cap Z'\}$. Then $Z' \in [\![\widehat{\varphi}]\!]$ implies $Z \in [\![\varphi]\!]$.*

Proof. The proof is done by induction on φ.

- The cases of $\varphi = \mathbf{tt}$ and $\varphi = \mathbf{ff}$ are trivial.
- $\varphi = x$. Then $\widehat{\varphi} = \bigvee Y_x$. As $Z' \in [\![\widehat{\varphi}]\!]$, there has to be some $y \in Y_x \cap Z'$. Thus $x \in Z$, which means that $Z \in [\![\varphi]\!]$.

- The cases of $\wedge$ and $\vee$ are similar to the proof of the previous lemma. ◀

We can now proceed to the fundamental definition of refinement of OTS.

▶ **Definition 8.** Let $(\mathcal{P}_1, \dashrightarrow_1, \mathbf{\Omega}_1)$, $(\mathcal{P}_2, \dashrightarrow_2, \mathbf{\Omega}_2)$ be two OTS and $R \subseteq \mathcal{P}_1 \times \mathcal{P}_2$. We say that R is a *refinement relation*, if $(S,T) \in R$ implies that:

- Whenever $S \overset{a}{\dashrightarrow} S'$ there is $T \overset{a}{\dashrightarrow} T'$ such that $(S',T') \in R$.
- $\mathbf{\Omega}_1(S) \sqsubseteq_{\Sigma R} \mathbf{\Omega}_2(T)$ where $\Sigma R = \{((a,S),(a,T)) \mid a \in \Sigma, (S,T) \in R\}$.

We say that S refines T (denoted as $S \leq T$) if there is a refinement relation R such that $(S,T) \in R$. Further, we say that a process I is an *implementation of a process* S if I is an implementation and $I \leq S$. We denote the set of all implementations of S by $[\![S]\!] = \{I \mid I \leq S, I \text{ is an implementation}\}$.

▶ Remark. Clearly, our definition of refinement coincides with modal refinements on all discussed subclasses of OTS.

One can easily see that every system satisfying the consistency requirement has an implementation, whereas DMTS and MixTS do not necessarily have one. We can compare various flavours of modal transition systems according to expressivity. Due to previous observation, we only consider nonempty sets of implementations.

▶ **Definition 9.** Let $\mathcal{C}, \mathcal{D}$ be subclasses of OTS. We say that $\mathcal{D}$ is at least as expressive as $\mathcal{C}$, written $\mathcal{C} \preceq \mathcal{D}$, if for every $C \in \mathcal{C}$ with $[\![C]\!] \neq \emptyset$ there is $D \in \mathcal{D}$ such that $[\![D]\!] = [\![C]\!]$.We write $\mathcal{C} \equiv \mathcal{D}$ to indicate $\mathcal{C} \preceq \mathcal{D}$ and $\mathcal{C} \succeq \mathcal{D}$, and $\mathcal{C} \prec \mathcal{D}$ to indicate $\mathcal{C} \preceq \mathcal{D}$ and not $\mathcal{C} \equiv \mathcal{D}$.

3 Process Algebra for DMTS

In this section we define a process algebra for OTS. However, since the processes represent sets of implemented systems (i.e. sets of sets of behaviours), we still need the obligation function to fully capture them. For the sake of simplicity, we introduce the parallel composition operator only in the following subsection.

▶ **Definition 10.** Let $\mathcal{X}$ be a set of process names. A *term of process algebra for OTS* is given by the following syntax:

$$P ::= \text{nil} \mid \text{co-nil} \mid a.P \mid X \mid P \wedge P \mid P \vee P \mid \lightning P$$

where X ranges over $\mathcal{X}$ and every $X \in \mathcal{X}$ is assigned a defining equality of the form $X := P$ where P is a term. The semantics is given by the following structural operational semantics rules:

$$\frac{}{a.P \overset{a}{\dashrightarrow} P} \qquad \frac{P \overset{a}{\dashrightarrow} P'}{X \overset{a}{\dashrightarrow} P'} X := P \qquad \frac{P \overset{a}{\dashrightarrow} P'}{P \wedge Q \overset{a}{\dashrightarrow} P'} \qquad \frac{P \overset{a}{\dashrightarrow} P'}{P \vee Q \overset{a}{\dashrightarrow} P'}$$

The obligation function on terms is defined structurally as follows:

$$\begin{array}{lll} \mathbf{\Omega}(\text{nil}) & = \mathbf{tt} & \mathbf{\Omega}(P \wedge Q) = \mathbf{\Omega}(P) \wedge \mathbf{\Omega}(Q) \\ \mathbf{\Omega}(\text{co-nil}) & = \mathbf{ff} & \mathbf{\Omega}(P \vee Q) = \mathbf{\Omega}(P) \vee \mathbf{\Omega}(Q) \\ \mathbf{\Omega}(a.P) & = (a,P) & \mathbf{\Omega}(\lightning P) = \mathbf{\Omega}(P) \\ \mathbf{\Omega}(X) & = \mathbf{\Omega}(P) \quad \text{for } X := P & \end{array}$$

As a convenient shortcut we introduce $?P \equiv (P \vee \text{nil})$ to capture the may transitions, i.e. an allowed behaviour that is not necessarily forced. Hence we easily obtain the following using the rules above:

$$\frac{P \overset{a}{\dashrightarrow} P'}{?P \overset{a}{\dashrightarrow} P'} \qquad \mathbf{\Omega}(?P) = \mathbf{tt}$$

We now obtain the discussed subclasses of OTS as syntactic subclasses generated by the following syntax equations (modulo transformation to CNF):

DMTS	$P ::= \text{nil} \mid a.P \mid X \mid P \wedge P \mid P \vee P \mid \not\downarrow P \mid \text{co-nil}$
cDMTS	$P ::= \text{nil} \mid a.P \mid X \mid P \wedge P \mid P \vee P$
MixTS	$P ::= \text{nil} \mid a.P \mid X \mid P \wedge P \mid P \vee \text{nil} \mid \not\downarrow P \mid \text{co-nil}$
MTS	$P ::= \text{nil} \mid a.P \mid X \mid P \wedge P \mid P \vee \text{nil}$
LTS	$P ::= \text{nil} \mid a.P \mid X \mid P \wedge P$

3.1 Composition

We define the composition operator based on synchronous message passing, as it encompasses the synchronous product as well as interleaving.

▶ **Definition 11.** Let $\Gamma \subseteq \Sigma$ be a *synchronizing alphabet*. For processes S_1 and S_2 we define the process $S_1 \parallel S_2$ as follows.

$$\frac{S_1 \overset{a}{\dashrightarrow} S_1' \quad S_2 \overset{a}{\dashrightarrow} S_2'}{S_1 \parallel S_2 \overset{a}{\dashrightarrow} S_1' \parallel S_2'}\ a \in \Gamma$$

$$\frac{S_1 \overset{a}{\dashrightarrow} S_1'}{S_1 \parallel S_2 \overset{a}{\dashrightarrow} S_1' \parallel S_2}\ a \in \Sigma \setminus \Gamma \qquad \frac{S_2 \overset{a}{\dashrightarrow} S_2'}{S_1 \parallel S_2 \overset{a}{\dashrightarrow} S_1 \parallel S_2'}\ a \in \Sigma \setminus \Gamma$$

As we may assume obligations to be in disjunctive normal form, let us denote $\mathbf{\Omega}(S_1) = \bigvee_i \bigwedge_j (a_{ij}, P_{ij})$ and $\mathbf{\Omega}(S_2) = \bigvee_k \bigwedge_\ell (b_{k\ell}, Q_{k\ell})$. We define $\mathbf{\Omega}(S_1 \parallel S_2)$ by

$$\bigvee_{i,k} \Big(\bigwedge_{j,\ell : a_{ij} = b_{k\ell} \in \Gamma} (a_{ij}, P_{ij} \parallel Q_{kl}) \wedge \bigwedge_{j : a_{ij} \notin \Gamma} (a_{ij}, P_{ij} \parallel S_2) \wedge \bigwedge_{\ell : b_{k\ell} \notin \Gamma} (b_{k\ell}, S_1 \parallel Q_{kl}) \Big)$$

Intuitively, for a process S, the set $[\![\mathbf{\Omega}(S)]\!] \subseteq 2^{\Sigma \times P}$ consists of all possible choices of successors of S that realize all obligations. Composing $[\![\mathbf{\Omega}(S_1)]\!]$ and $[\![\mathbf{\Omega}(S_2)]\!]$ in the same manner as may transitions above generates $[\![\mathbf{\Omega}(S_1 \parallel S_2)]\!]$.

Note that $[\![\mathbf{\Omega}(S)]\!]$ corresponds to DNF of obligations. Nevertheless, they can also be written equivalently in the form of a set of must transitions of DMTS, which corresponds to CNF. During the design process CNF is more convenient to use, whereas the composition has to be done in DNF even for DMTS and then translated back, thus causing an exponential blowup. However, using OTS allows for only one transformation and then the compositions are done using DNF, as the result is again in DNF. As our definition extends the previous definitions on all the discussed models, this shows another use of OTS.

▶ Remark. Refinement is a precongruence with respect to all operators of the process algebra (including the composition operator). Hence, refinemental equivalence, i.e. $\leq \cap \leq^{-1}$, is a congruence.

4 Hierarchy Results

In this section, we study the relationship between the OTS subclasses and establish the following complete result:

$$\text{LTS (implementations)} \prec \text{MTS} \prec \text{MixTS} \prec \text{cDMTS} \equiv \text{DMTS (OTS)}$$

We first show that cDMTS $\equiv$ DMTS. We do that by showing that every OTS process that has an implementation can be substituted by an OTS process that satisfies the consistency requirement and has the same set of implementations. To that end, we use an auxiliary definition of a consistency relation. This definition is a slight modification of the consistency relation defined in [8]. In the definition, the notation $2^{\mathcal{P}}_{\text{Fin}}$ stands for the set of all finite subsets of $\mathcal{P}$.

▶ **Definition 12** (consistency)**.** Let $(\mathcal{P}, \dashrightarrow, \mathbf{\Omega})$ be a OTS. A subset C of $2^{\mathcal{P}}_{\text{Fin}}$ is called a *consistency relation* if for all $\{S_1, \ldots, S_n\} \in C$ and $i \in \{1, \ldots, n\}$ there is $X \in [\![\mathbf{\Omega}(S_i)]\!]$ such that for all $(a, U) \in X$ there are $S_j \overset{a}{\dashrightarrow} T_j$ (for all j) such that $\{U, T_1, \ldots, T_n\} \in C$.

It may be easily seen that an arbitrary union of consistency relations (for given OTS) is also a consistency relation. Therefore, we may talk about the greatest consistency relation. The following lemma explains the motivation behind the consistency relation, i.e. that a set of processes is consistent if it has a common implementation.

▶ **Lemma 13.** *Let $S_1, \ldots, S_n$ be processes. There exists a consistency relation C containing $\{S_1, \ldots, S_n\}$ if and only if $\bigcap_{1 \le i \le n} [\![S_i]\!] \neq \emptyset$.*

Proof. Recall that $\varphi \sqsubseteq_{\Sigma\le} \psi$ if and only if for all $M \in [\![\varphi]\!]$ there is some $N \in [\![\psi]\!]$ such that for all $(a, T) \in N$ there is some $(a, S) \in M$ such that $S \le T$.

We show that $C = \{\{S_1, \ldots, S_k\} \mid k \in \mathbb{N}, \bigcap_i [\![S_i]\!] \neq \emptyset\}$ is a consistency relation. Let $\{S_1, \ldots, S_n\} \in C$, let $I \in \bigcap_i [\![S_i]\!]$ and let $i \in \{1, \ldots, n\}$ be arbitrary. Take $M = \{(a, J) \mid I \overset{a}{\dashrightarrow} J\}$. Clearly, $M \in [\![\mathbf{\Omega}(I)]\!]$ as I is an implementation. Due to $\mathbf{\Omega}(I) \sqsubseteq_{\Sigma\le} \mathbf{\Omega}(S_i)$ there has to be some $N \in [\![\mathbf{\Omega}(S_i)]\!]$ such that for each $(a, U) \in N$ there is $(a, J) \in M$ such that $J \le U$.

Let now $X = N$ and let $(a, U) \in X$. Then $I \overset{a}{\dashrightarrow} J$ with $J \le U$. Therefore, as $I \le S_j$, $S_j \overset{a}{\dashrightarrow} T_j$ and $J \le T_j$ for all j. Thus $J \in [\![U]\!] \cap \bigcap_i [\![T_i]\!]$ and $\{U, T_1, \ldots, T_n\} \in C$.

To show the converse, assume that there is a consistency relation C containing $\{S_1, \ldots, S_n\}$. We know that for all i there is some $X \in \mathbf{\Omega}(S_i)$ such that for all $(a, U) \in X$ there are $S_j \overset{a}{\dashrightarrow} T_j$ (for all j) such that $\{U, T_1, \ldots, T_n\} \in C$. For fixed i, we denote the chosen X as X_i. We construct I coinductively as follows:

$$\mathbf{\Omega}(I) = \bigwedge_i \bigwedge_{(a,U) \in X_i} (a, J_i^U)$$

with $I \dashrightarrow$ transitions to all (a, J_i^U), where J_i^U is a common implementation of U, T_1, …, T_n with T_i given above. Clearly, I is an implementation of all S_i. ◀

We now proceed with the construction of a new consistent OTS that is equivalent to the original OTS.

▶ **Definition 14.** Let $(\mathcal{P}, \dashrightarrow, \mathbf{\Omega})$ be a OTS, Con its greatest consistency relation. We create a new OTS as $(\mathsf{Con}, \dashrightarrow, \mathbf{\Omega})$ where

- $\mathcal{S} \overset{a}{\dashrightarrow} \mathcal{T}$ whenever for all $S \in \mathcal{S}$, $S \overset{a}{\dashrightarrow} T$ with $T \in \mathcal{T}$.

- $\mathbf{\Omega}(\mathcal{S}) = \bigwedge_{S\in\mathcal{S}} \widehat{\mathbf{\Omega}(S)}$ where $\widehat{\mathbf{\Omega}(S)}$ is the formula that is created from $\mathbf{\Omega}(S)$ by replacing all occurrences of (a, U) by $\bigvee\{(a, \{U, T_1, \ldots, T_n\}) \mid \forall i : S_i \xrightarrow{a} T_i, \{U, T_1, \ldots, T_n\} \in \mathsf{Con}\}$ (where $\bigvee \emptyset = \mathbf{ff}$).

Note that due to the properties of Con, $\mathbf{\Omega}(\mathcal{S})$ is never $\mathbf{ff}$. We prove that the construction is correct, i.e. for every consistent process of the original OTS, we have indeed a process of the new OTS with the same set of implementations.

▶ **Theorem 15.** *Let S be a process. Then $[\![S]\!] \neq \emptyset$ if and only if $\{S\} \in \mathsf{Con}$. Moreover, if $\{S\} \in \mathsf{Con}$ then $[\![S]\!] = [\![\{S\}]\!]$.*

Proof. The first part of the theorem is already included in Lemma 13. We thus prove the second part. We first show that $I \in [\![S]\!]$ implies $I \in [\![\{S\}]\!]$. We define R as:

$$R = \{(I, \{S_1, \ldots, S_n\}) \mid n \in \mathbb{N}, \forall i : I \in [\![S_i]\!], \{S_1, \ldots, S_n\} \in \mathsf{Con}\}$$

and prove that R is a refinement relation. Let $(I, \{S_1, \ldots, S_n\}) \in R$.

- Let $I \xrightarrow{a} J$. Then, as $I \leq S_i$, $S_i \xrightarrow{a} T_i$ with $J \leq T_i$ for all i. Thus also $\{S_1, \ldots, S_n\} \xrightarrow{a} \{T_1, \ldots, T_n\}$ and $(J, \{T_1, \ldots, T_n\}) \in R$.
- Let $\mathbf{\Omega}(I) = \varphi$, $\mathbf{\Omega}(\{S_1, \ldots, S_n\}) = \psi$. We need to show that $\varphi \sqsubseteq_{\Sigma R} \psi$.
  Let $M \in [\![\varphi]\!]$. Then, as $I \leq S_i$ for all i, there exist $N_i \in [\![\mathbf{\Omega}(S_i)]\!]$ such that for all $(a, U) \in N_i$ exists $(a, J) \in M$ with $J \leq U$ (due to $\varphi \sqsubseteq_{\Sigma\leq} \mathbf{\Omega}(S_I)$). We use the notation $J_{(a,U)}$ to denote such J.
  Let now $N = \{(a, \{U, T_1, \ldots, T_n\}) \mid \exists i : (a, U) \in N_i, \forall j : S_j \xrightarrow{a} T_j, J_{(a,U)} \leq T_j, \{U, T_1, \ldots, T_n\} \in \mathsf{Con}\}$. Clearly, for all $(a, \{U, T_1, \ldots, T_n\}) \in N$ there is some $(a, J) \in M$ such that $(J, \{U, T_1, \ldots, T_n\}) \in R$ (we take $J = J_{(a,U)}$).
  We need to prove that $N \in [\![\psi]\!]$. In other words, we need to prove that for all i, $N \in [\![\widehat{\mathbf{\Omega}(S_i)}]\!]$. That is, however, a straightforward corollary of Lemma 6 (take $Z = N_i$, $Z' = N$).

We now show that $I \in [\![\{S\}]\!]$ implies $I \in [\![S]\!]$. We define R as:

$$R = \{(I, S) \mid I \leq \mathcal{S} \text{ with } S \in \mathcal{S} \in \mathsf{Con}\}$$

and prove that R is again a refinement relation. Let $(I, S) \in R$ and let $\mathcal{S}$ be such that $I \leq \mathcal{S}$ and $S \in \mathcal{S}$.

- Let $I \xrightarrow{a} J$. Then $\mathcal{S} \xrightarrow{a} \mathcal{T}$ with $J \leq \mathcal{T}$ and thus $S \xrightarrow{a} T$ with $T \in \mathcal{T}$. Thus also $(J, T) \in R$.
- Let $\mathbf{\Omega}(I) = \varphi$, $\mathbf{\Omega}(S) = \psi$. We need to show that $\varphi \sqsubseteq_{\Sigma R} \psi$. Let $M \in [\![\varphi]\!]$. Due to the fact that $\varphi \sqsubseteq_{\Sigma\leq} \mathbf{\Omega}\mathcal{S}$, we know that there exists $N' \in [\![\mathbf{\Omega}(\mathcal{S})]\!]$ such that for all $(a, \{U, T_1, \ldots, T_k\}) \in N'$ there exists $(a, J) \in M$ with $J \leq \{U, T_1, \ldots, T_k\}$. Take $N = \{(a, U) \mid (a, \mathcal{T}) \in N' \text{ with } U \in \mathcal{T}\}$. Clearly, as $N' \in [\![\mathbf{\Omega}(\mathcal{S})]\!]$ also $N' \in [\![\widehat{\mathbf{\Omega}(S)}]\!]$. Using Lemma 7, we get that $N \in [\![\mathbf{\Omega}(S)]\!]$ (take $Z' = N', Z = N$). ◀

The following lemma shows that MixTS $\prec$ cDMTS.

▶ **Lemma 16.** *There is no MixTS M such that $[\![M]\!] = [\![a.\mathrm{nil} \vee b.\mathrm{nil}]\!]$.*

Proof. We first note that any equation defining a MixTS may be written in the following normal form:

$$X := \bigwedge_i ?a_i.S_i \wedge \bigwedge_j \not\downarrow a_j.T_j$$

Clearly, there are three implementations of $a.\text{nil} \vee b.\text{nil}$, namely $a.\text{nil}$, $b.\text{nil}$ and $a.\text{nil} \wedge b.\text{nil}$. Let thus M have these three implementations. Clearly, the $\not\downarrow$ part of M has to be empty (i.e. $\mathbf{\Omega}(M) = \mathbf{tt}$) as M can force neither a transition nor b transition. But then $\text{nil} \in [\![M]\!]$. ◀

Due to Theorem 15, we have a syntactic characterization of consistent OTS. Since we now know MixTS $\prec$ DMTS, a question arises whether such a characterization can be obtained also for consistent MixTS. Observe that the previous construction transforms every MixTS into a consistent OTS with formulae in CNF where all literals in one clause have the same action. One might be tempted to consider the following syntactic characterization of consistent MixTS:

$$P ::= \text{nil} \mid X \mid a.P \mid P \wedge P \mid \bigvee_i a.P_i$$

However, that is not the case, as shown by the following lemma. Hence, this question remains open.

▶ **Lemma 17.** *There is no MixTS M such that $[\![M]\!] = [\![(a.(a.\text{nil} \wedge b.\text{nil}) \vee a.nil) \wedge ?a.a.\text{nil}]\!]$.*

Proof. Let $M = \bigwedge_i ?a.N_i \wedge \bigwedge_j \not\downarrow a.O_j$. (All outgoing transitions from M have to be a-transitions.) We make the following observations:

- For all j, $\mathbf{\Omega}(O_j) = \mathbf{tt}$. Otherwise, $a.\text{nil}$ could not be an implementation of M.
- Also, for all j, $O_j \xrightarrow{a}$. Otherwise, $a.(a.\text{nil} \wedge b.\text{nil})$ could not be an implementation of M.
- There has to be some k such that $a.\text{nil} \in [\![N_k]\!]$, as $a.\text{nil} \wedge a.a.\text{nil}$ also has to be an implementation of M.

We now show that $a.a.\text{nil}$ is an implementation of M. Let R' be an arbitrary refinement relation such that $(a.\text{nil}, N_k) \in R'$ (we know that such R' exists as $a.\text{nil} \leq N_k$). Take R as

$$R = id \cup R' \cup \{(a.a.\text{nil}, M)\} \cup \{(a.\text{nil}, O_j) \mid \forall j\}$$

We now show that R is a refinement.

- $a.a.\text{nil} \xrightarrow{a} a.\text{nil}$ is matched by $M \xrightarrow{a} N_k$.
- $\mathbf{\Omega}(a.a.\text{nil}) = (a, a.\text{nil})$, $\mathbf{\Omega}(M) = \bigwedge_j (a, O_j)$, thus $\mathbf{\Omega}(a.a.\text{nil}) \sqsubseteq_{\Sigma R} \mathbf{\Omega}(M)$.
- $((a.\text{nil}), N_k) \in R'$, therefore the conditions of refinement are satisfied.
- $a.\text{nil} \xrightarrow{a} \text{nil}$ is matched by $O_j \xrightarrow{a} \text{nil}$
- As $\mathbf{\Omega}(O_j) = \mathbf{tt}$, clearly $\mathbf{\Omega}(a.\text{nil}) \sqsubseteq_{\Sigma R} \mathbf{\Omega}(O_j)$.

However, $a.a.\text{nil}$ is not an implementation of $(a.(a.\text{nil} \wedge b.\text{nil}) \vee a.nil) \wedge ?a.a.\text{nil}$. ◀

Finally, we show that MTS $\prec$ MixTS.

▶ **Lemma 18.** *There is no MTS S such that $[\![S]\!] = [\![?a.b.\text{nil} \wedge ?a.c.\text{nil} \wedge \not\downarrow a.(?b.\text{nil} \wedge ?c.\text{nil})]\!]$.*

Proof. Similarly to MixTS, any equation defining a MTS can be written in the following normal form:

$$X = \bigwedge_i ?a_i.S_i \wedge \bigwedge_j a_j.T_j$$

There are three implementations which S has to possess and those are $a.b.\text{nil}$, $a.c.\text{nil}$, and $a.b.\text{nil} \wedge a.c.\text{nil}$ and S cannot possess any other implementation. Clearly, S cannot be of the form $a.T \wedge P$, as then T would have to satisfy $b.\text{nil} \leq T$ (as $a.b.\text{nil} \leq S$), also $c.\text{nil} \leq T$ (as $a.c.\text{nil} \leq S$), yet it cannot satisfy $(b.\text{nil} \wedge c.\text{nil}) \leq T$ (as $a.(b.\text{nil} \wedge c.\text{nil}) \not\leq S$). This is not possible as it can be proven that if $P \leq T$ and $Q \leq T$ then also $P \wedge Q \leq T$ for all OTS. Therefore $S = \bigvee_i ?a.S_i$ and thus $\mathbf{\Omega}(S) = \mathbf{tt}$. But then $\text{nil} \leq S$ and S has more implementations than $?a.b.\text{nil} \wedge ?a.c.\text{nil} \wedge \not\downarrow a.(?b.\text{nil} \wedge ?c.\text{nil})$. ◀

For the sake of completeness, we also state that LTS $\prec$ MTS. This trivially follows, as every LTS only has one implementation, whereas e.g. $?a.\mathrm{nil}$ has two implementations $a.\mathrm{nil}$ and nil.

5 Conclusion and Future Work

We have introduced a new formalism of transition system with obligations together with its process algebra. We have used it to compare various previously studied systems. The main result shows that general DMTS are not more powerful than consistent DMTS, whereas mixed transition systems are strictly less expressive. Furthermore, we have given an alternative syntactic characterizations of the studied systems, although a complete syntactic criterion for consistent mixed transition systems remains as a future work. Surprisingly, using more general OTS leads to some optimizations in computation of the composition that were not possible in the previously used frameworks (as discussed in [2]).

Acknowledgements Nikola Beneš has been supported by Czech Grant Agency grant no. GD102/09/H042. Jan Křetínský is a holder of Brno PhD Talent Financial Aid.

References

1 A. Antonik, M. Huth, K. G. Larsen, U. Nyman, and A. Wasowski. 20 years of modal and mixed specifications. *Bulletin of the EATCS no. 95*, pages 94–129, 2008.

2 N. Beneš, I. Černá, and J. Křetínský. Disjunctive modal transition systems and generalized LTL model checking. Technical report FIMU-RS-2010-12, Faculty of Informatics, Masaryk University, Brno, 2010.

3 Dennis Dams, Rob Gerth, and Orna Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997.

4 H. Fecher and M. Steffen. Characteristic mu-calculus formulas for underspecified transition systems. *ENTCS*, 128(2):103–116, 2005.

5 M. Huth, R. Jagadeesan, and D. A. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In *Proc. of ESOP'01*, volume 2028 of *LNCS*, pages 155–169. Springer, 2001.

6 K. G. Larsen, U. Nyman, and A. Wasowski. Modeling software product lines using color-blind transition systems. *STTT*, 9(5-6):471–487, 2007.

7 K. G. Larsen and B. Thomsen. A modal process logic. In *LICS*, pages 203–210. IEEE Computer Society, 1988.

8 K. G. Larsen and L. Xinxin. Equation solving using modal transition systems. In *LICS*, pages 108–117. IEEE Computer Society, 1990.

9 J.-B. Raclet. Residual for component specifications. In *Proc. of the 4th International Workshop on Formal Aspects of Component Software*, 2007.

10 J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, and R. Passerone. Why are modalities good for interface theories? In *ACSD*, pages 119–127. IEEE, 2009.