

Finite Abstract Models for Deterministic Transition Systems: Fair Parallel Composition and Refinement-Preserving Logic

Harald Fecher and Immo Grabe

Christian-Albrechts-University at Kiel, Germany
{hf,igb}@informatik.uni-kiel.de

Abstract. Since usually no scheduler is given at the programming or modeling language level, abstract models together with a refinement notion are necessary to model concurrent systems adequately. Deterministic transition systems are an appropriate model for implementations of (concurrent) reactive programs based on synchronous communication. In this paper, we develop a suitable setting for modeling and reasoning about deterministic transition systems. In particular, we (i) develop a class of abstract models together with a refinement notion; (ii) define parallel composition guaranteeing fairness; and (iii) develop a 3-valued logic with a satisfaction relation that is preserved under refinement.

1 Introduction

The execution of concurrent reactive programs, where the scheduler is given, e.g., by the operating system, behaves (if no real random generator exists) deterministically up to the environment, i.e., the system behaves in the same way whenever the environment behaves in the same way (including points in time). Deterministic transition systems, where no two transitions leaving the same state have the same label, are an appropriate model for reactive systems based on synchronous communication, whenever the environment will provide at most one action (resp. will request at most one of the actions provided by the system) at once. For example, they are in particular an appropriate model for implementations of a UML state machine [1], where only synchronous communication between the state machine and its event pool, which can provide at most one ‘event’ at the same time, occurs.

Deterministic transition systems are also appropriate as model for components of closed concurrent systems, whenever every component has its own scheduler, i.e., determines which process(es) of the component performs the next action. Here, a global scheduler decides if a (and which) communication between the component and its environment takes place or if an internal computation takes place.

Models for programming languages that contain concurrency are usually non-deterministic, since the scheduler is not known at that level (i.e., will be provided by the operating system). Therefore, those models as well as models for

modeling languages should contain nondeterminism, which will be resolved (via refinements) in later design phases and/or by the operating system until deterministic computations are reached. Properties valid on the abstract level, i.e., on the model containing nondeterminism, should be preserved under refinement to maintain the relation between the model and the system. Furthermore, a model for the abstract level should provide a compact and finite description of sets of implementations, especially to improve verification. Moreover, it should be closed under standard operators to be suitable for defining semantics of programming languages and for compositional reasoning. Note that often programmers, software engineers, and computer scientists stay on the abstract level and never reach the concrete level in their contribution to the software development process. Nevertheless, it is important to know what exactly the systems are, since the definition of, e.g., sound satisfaction at the abstract level heavily depends on this information.

Contribution. We develop a setting for modeling and reasoning about deterministic transition systems.

In particular:

- We develop a class of abstract models together with a refinement notion, where exactly the deterministic transition systems are the concrete ones. Our model allows finite/compact modeling by (i) abstracting labels, (ii) having a predicate over labels indicating whether the removal of all transitions having a label is allowed as a refinement step or not, and (iii) having Streett acceptance conditions for restricting infinite computations.
- We define parallel composition for our model that (i) preserves refinement, (ii) preserves satisfiability (i.e., the existence of a refining implementation), and (iii) guarantees fairness, in that, roughly speaking, every component as well as internal synchronization gets an infinite number of opportunities to execute. Here, Streett acceptance conditions are naturally generated by parallel composition between deterministic transition systems.
- We develop a logic together with its satisfaction relation. The logic has as its basic operator $\langle\langle\alpha\rangle\rangle q$ indicating that α can be executed and after executing α property q is guaranteed to hold. This logic yields a 3-valued satisfaction relation on our model, but is 2-valued on concrete abstractions (implementations). We show soundness, i.e., that satisfaction is preserved under refinement. Furthermore, deciding our satisfaction relation is in NP and approximates the EXPTIME-hard language inclusion problem which asks whether all implementations that refine abstraction \mathcal{M} satisfy property ϕ . The PSPACE-complete LTL model checking problem is also approximated.

Related work. Kripke structures (with Streett fairness constraints) together with trace inclusion as refinement notion are used as abstract settings for linear time, where implementations are traces. In this context, LTL [2] is an appropriate logic. Abstract models used for abstraction of linear time settings are not appropriate for our purpose, since they do not model the branching time sensitivity obtained by communications on different actions.

Transition systems with (forward or backward) simulation [3, 4] are not an appropriate setting for abstraction of deterministic transition systems, since deterministic transition systems can be refined further and, therefore, refinement preserving satisfaction relations are in general not 2-valued on them. Therefore, alternating refinement [5] also yields no appropriate setting for our purpose, since it coincides with simulation on labeled transition systems.

On the other hand, transition systems with ready simulation [6] yield an appropriate setting if deterministic transition systems are the implementations. The predicate over labels and the fairness constraint in our setting allow a more compact representation than ready simulation, which will be illustrated later. Note that ready simulation coincides with our refinement notion for the canonical embedding of transition systems into our setting. Transition systems are already extended in [7, 8] by a predicate over labels indicating divergence (infinitely many internal computations are possible). Therefore, the relation introduced there, called *prebisimulation*, does not yield a comparable refinement notion. The refinement notions of failure, failure trace, ready, and ready trace inclusion [9] are also appropriate settings if deterministic transition systems are the implementations. Their trace based approach makes it hard to define an approximated, compositional satisfaction relation that is preserved under refinement.

Standard branching time logics, which are interpreted on transition systems, are, e.g., CTL [10] and the μ -calculus [11]. But these logics are not appropriate for our setting, since these logics are not preserved under ready simulation: the property that “there is a transition labeled a such that b is possible afterwards” holds in the labeled transition system $\square \xleftarrow{b} \square \xleftarrow{a} \square \xrightarrow{a} \square$ but not in its refinement $\square \xrightarrow{a} \square$. μ -automata [12], (disjunctive) modal transition systems [13, 14] and their variants [15, 16, 17, 18] are used as abstraction model for transition systems in order to improve verification of full branching time properties, as, e.g., in [19, 20, 21]. These models are not appropriate for our purpose, since they consider transition systems rather than deterministic transition systems as implementations. Consequently, these models contain additional complex structures that are unnecessary if the implementations are guaranteed to be deterministic. For example, a state in a modal transition system can have more than one outgoing must-transitions, which makes it, e.g., hard to determine satisfiability w.r.t. deterministic transition systems.

To the best of our knowledge there is no abstract model (beside the model developed here) that can create finite abstraction of labeled (deterministic) transition systems in case infinitely many different transition labels are used.

Outline. Our model together with its refinement notion is formally introduced in Section 2, whereas in Section 3 the parallel composition is presented. Section 4 introduces the logic together with the satisfaction relation and Section 5 presents illustration how the setting can be used for modeling and for verification. Section 6 concludes the paper and discusses future work.

2 Synchronously-Communicating Transition Systems

Here, we present the model of interest, including the refinement notion. In the following, $|M|$ denotes the cardinality of a set M , $\mathbb{P}(M)$ denotes its power set, and $\overline{M} = \{\overline{m} \mid m \in M\}$ denotes the set of conames of M . Furthermore, let \mathcal{Act} be the set of actions such that (i) \mathcal{Act} , $\{\tau\}$, and $\{\Delta\}$ are pairwise disjoint and (ii) $\forall h \in \mathcal{Act} : \overline{h} \in \mathcal{Act} \wedge \overline{\overline{h}} = h$.¹ Here, $\overline{h} \in \mathcal{Act}$ is the co-action on which $h \in \mathcal{Act}$ synchronizes, τ indicates an internal computation, and Δ indicates that a not yet specified communication is possible. Note that in reactive programs fairness depends more on the communication than on the states. Therefore, defining fairness constraints on transitions rather than on states leads to a smoother approach. Formally, our abstract model for (programming and modeling languages of) deterministic, concurrent, reactive systems based on synchronous communication is:

Definition 1 (STS). A synchronously-communicating transition system (STS) \mathcal{M} is a tuple $(S, S^i, \Omega, T, \gamma, e, \mathbb{S})$ such that

- $(s \in)S$ is its set of states,
- $S^i \subseteq S$ is its nonempty set of initial states,
- $\Omega \subseteq \mathcal{Act}$ is its set of explicitly modeled communication,
- $(t \in)T$ is its set of transitions,
- $\gamma : T \rightarrow S \times (\mathcal{Act} \cup \{\tau, \Delta\}) \times S$ is its transition relation;
to simplify later definitions, we assume that every initial state is a target of an τ -transition, i.e., $\forall s' \in S^i : \exists t, s : \gamma(t) = (s, \tau, s')$, (those transitions, from a non-reachable state s , are often omitted in later illustrations)
- $e : S \rightarrow \mathbb{P}(\mathcal{Act} \cup \{\tau\})$ is its action existence predicate,
- $\mathbb{S} \in \mathbb{P}(\mathbb{P}(T) \times \mathbb{P}(T))$ a finite set representing a Streett acceptance condition.

\mathcal{M} is called finite if $|S| + |\Omega| + |T| + |\bigcup_{s \in S} e(s)|$ is finite.

Before we give comments on the above definition, we introduce the following notations: The components of a STS \mathcal{M} are denoted by $S, S^i, \Omega, T, \gamma, e, \mathbb{S}$ and tagged with indices if needed. We write $\text{src}_{\mathcal{M}}(t)$ for the source, $\text{lab}_{\mathcal{M}}(t)$ for the label, and $\text{tar}_{\mathcal{M}}(t)$ for the target of $t \in T$ (where subscript \mathcal{M} is omitted if it is clear from the context), i.e., if $\gamma(t) = (s, \alpha, s')$ then $\text{src}(t) = s$, $\text{lab}(t) = \alpha$, and $\text{tar}(t) = s'$. Furthermore, $\mathcal{O}_{\mathcal{M}}(s)$ denotes the set of labels that occur on transitions leaving s , i.e., $\mathcal{O}(s) = \{\text{lab}(t) \mid t \in T \wedge \text{src}(t) = s\}$.

Transitions having labels α outside Ω can be matched by α or by the default label. The following function is used later to model this circumstance:

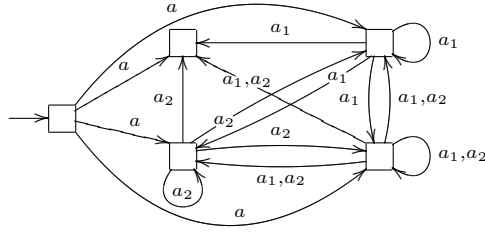
$$\tilde{\gamma}(t) \stackrel{\text{def}}{=} \begin{cases} \overline{\mathcal{A}} \setminus \Omega \{(\text{src}(t), h, \text{tar}(t)) \mid h \in \mathcal{Act} \setminus \Omega\} & \text{if } \text{lab}(t) = \Delta \\ \{\gamma(t)\} & \text{otherwise.} \end{cases}$$

Predicate e is used to obtain compact abstractions, especially for the definition of the parallel composition. Here, $h \in e(s) \cap \mathcal{Act}$ indicates that the existence of

¹ Note that overlined labels (\overline{h}) can be omitted if an undirected synchronous communication is used.

a transition labeled with h is guaranteed iff $h \in e(s)$. Similarly, the existence of a transition labeled with τ is only guaranteed iff $\tau \in e(s)$. Otherwise, a process, where no internal computation exists (can only communicate), is an allowed refinement. That predicate e really yields more compact representation than ready simulation is illustrated by the following example:

Example 2. Consider the STS $\longrightarrow \square \xrightarrow{a} \square \xrightarrow{a_1, \dots, a_n} \square$. In order to describe the same set of deterministic transition system of this STS via ready simulation at least $1 + 2^n$ states are necessary, since the non initial state has to be modeled by using all possible subsets of $\{a_1, \dots, a_n\}$. Furthermore, $2^n + 2^n (\sum_{i=0}^n (i \cdot \binom{n}{i}))$ transitions instead of the $1 + n$ of the STS are needed, since per label and state either all or none states of the $1 + 2^n$ derived states from the non-initial state have to be reached. For example, if $n = 2$ then the following transition system describes the same set of deterministic transition system via ready simulation:



The Streett acceptance condition for model \mathcal{M} is a predicate $\text{Acc}_{\mathcal{M}}$ that characterizes the *allowed* infinite sequences of transitions, those $(t_n)_{n \in \mathbb{N}}$ satisfying “for all $(E, F) \in \mathbb{S}$ set $\{n \in \mathbb{N} \mid t_n \in E\}$ is infinite or set $\{n \in \mathbb{N} \mid t_n \in F\}$ is finite”. We chose a Streett condition since they are closed under, and naturally appear in, parallel composition, which is not the case for RabinChain or Rabin fairness conditions. Moreover, a Streett condition guarantees that checking formulas of our logics, introduced later, for such models is in NP. Two STSs are illustrated in Figure 1. We continue by introducing implementations formally:

Definition 3 (Concrete STS). A concrete STS is a STS \mathcal{M} such that

- there is exactly one initial element, i.e., $|S^i| = 1$,
- every communication is explicitly modeled, i.e., $\Omega = \text{Act}$,
- the default label is not used, i.e., $\forall t \in T : \text{lab}(t) \neq \Delta$,
- a transition exists iff its existence predicate holds, i.e., $\forall s \in S : \mathcal{O}(s) = e(s)$,
- the underlying transition system is deterministic, i.e.,
 $\forall t, t' \in T : (\text{src}(t) = \text{src}(t') \wedge \text{lab}(t) = \text{lab}(t')) \Rightarrow t = t'$, and
- no acceptance condition exists, i.e., $\mathbb{S} = \emptyset$.

The STS $\check{\mathcal{N}}$ of Figure 1 is, e.g., a concrete STS, whereas $\hat{\mathcal{N}}$ of that figure is not concrete. We turn to defining a refinement notion between models, using game based definition, similar as, e.g., in [22], since fairness can be nicely handled

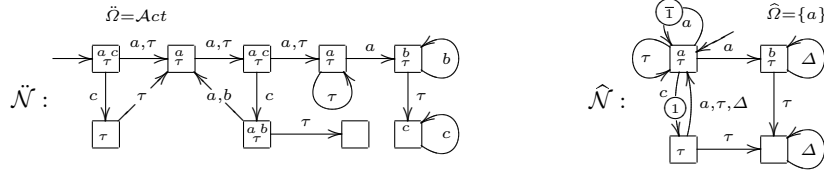


Fig. 1. Two synchronously-communicating transition systems. Here, $\mathcal{Act} = \{a, b, c\}$. Targets of transitions having no source indicate initial states. The elements of $e(s)$ are written inside the state borders of s . Transitions labeled with tuples describe a set of transitions each having a label from that tuple. A circled number i (resp., \bar{i}) on transition t denotes that t is contained in the first (resp., second) component of the i -th Streett condition pair.

Table 1. Moves of refinement game at configuration $(t_1, t_2) \in T_1 \times T_2$. Refinement plays are sequences of configurations generated thus.

Transition: Player II chooses $t'_1 \in T_1$ such that $\text{tar}(t_1) = \text{src}(t'_1)$; Player I responds with $t'_2 \in T_2$ such that $\text{tar}(t_2) = \text{src}(t'_2)$ and $\text{lab}(t'_1) = \text{lab}(t'_2) \vee (\text{lab}(t'_1) \in \mathcal{Act} \setminus \Omega_2 \wedge \text{lab}(t'_2) = \Delta)$; the next configuration is (t'_1, t'_2) .
Existence predicate: Player II chooses $a \in e_2(\text{tar}(t_2))$; Player I wins iff $a \in e_1(\text{tar}(t_1))$.

by games. For a sequence of tuples Φ we write $\Phi[i]$ for the sequence obtained from Φ through projection onto the i -th coordinate. Note that $\text{Acc}_{\mathcal{M}}$ is defined on page 5.

Definition 4 (Refinement).

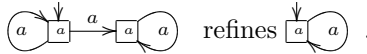

- Finite refinement plays for models \mathcal{M}_1 and \mathcal{M}_2 have the rules and winning conditions as stated in Table 1. An infinite play Φ is a win for Player I iff $\text{Acc}_{\mathcal{M}_1}(\Phi[1]) \Rightarrow \text{Acc}_{\mathcal{M}_2}(\Phi[2])$ holds; otherwise, it is won by Player II.
- Model \mathcal{M}_1 refines \mathcal{M}_2 iff $\Omega_2 \subseteq \Omega_1$ and Player I has a strategy for the corresponding refinement game between \mathcal{M}_1 and \mathcal{M}_2 such that for all $t_1 \in T_1$ with $\text{tar}(t_1) \in S_1^i$ there is $t_2 \in T_2$ with $\text{tar}(t_2) \in S_2^i$ and Player I wins all refinement plays started at (t_1, t_2) with her strategy.

Every label explicitly modeled at the abstract level has to be explicitly modeled at a more concrete level ($\Omega_2 \subseteq \Omega_1$). The necessity of a communication action (or computation) at the abstract level must be maintained at the concrete level (see Table 1). Furthermore, every action that is possible at the concrete level must have already been possible at the abstract level, possibly via the default label if the label does not have to be explicitly modeled at the abstract level (see Table 1). The acceptance on the concrete level has to be maintained at the abstract level in an infinite play.

Example 5. $\tilde{\mathcal{N}}$ of Figure 1 refines $\hat{\mathcal{N}}$ of that figure (the 4 upper states on the left side are abstracted by the upper left state, the upper right by the upper right,

the two bottom left by the bottom left, and the two bottom right by the bottom right one).

The refinement notion of simulation on transition system is embedded into our setting by mapping e always to the empty set, whereas the notion of ready simulation is embedded by mapping $e(s)$ always onto $\mathcal{O}(s)$.

Note that if an STS \mathcal{M} refines a concrete STS it is not necessarily the case that \mathcal{M} has to be concrete, e.g.,  refines  .

Theorem 6. *Refinement is reflexive and transitive. Moreover, refinement yields an equivalence relation on concrete STSs.*

Note that the complexity of checking refinement is high, since [Streett \Rightarrow Streett] does not reduce in general to Rabin or Streett conditions, i.e., in general no Player has a memoryless winning strategy. Nevertheless, this is not very problematic for our purpose, since in practice refinement will be guaranteed by construction: using refinement patterns for top-down developments and using abstraction techniques, like predicate abstraction [23], for bottom-up developments. To us it is more important to obtain efficient satisfaction check, which we get.

Definition 7 (Satisfiability). *Suppose \mathcal{M} is an STS, then \mathcal{M} is satisfiable if there is a concrete STS that refines \mathcal{M} .*

For example, the STSs of Figure 1 are satisfiable, whereas $\Rightarrow \square^a$ is not.

3 Parallel Composition

Only parallel composition, which is the most complex one, and no further operators, like hiding, is presented.² Parallel operators following different views, where, e.g., a CSP [24] based communication instead of a CCS [25] based handshake communication is used, can also be straightforwardly defined on STS. Below we write π_i for the projection onto the i -th coordinate of an ordered tuple.

Definition 8 (Parallel composition). *Suppose \mathcal{M}_1 and \mathcal{M}_2 are two STSs such that, without loss of generality, S_1, T_1 as well as S_2, T_2 are disjoint and $\Omega_1 = \Omega_2 = \text{Act}$. Then the parallel composition $\mathcal{M}_1 \parallel \mathcal{M}_2$ is the STS $(S_1 \times S_2, S_1^i \times S_2^i, \Omega_1, T, \gamma, e, \mathbb{S})$, where*

- $T \subseteq (T_1 \times S_2) \cup (S_1 \times T_2) \cup (T_1 \times T_2)$, where only those t are taken for which $\gamma(t)$, given in Table 2, is defined,

² Note that in order to define sequential composition, the definition of STS has to be extended such that termination can be modeled.

Table 2. Transition relation of the parallel composition.

$$\begin{array}{c}
\frac{(s_1, h, s'_1) \in \tilde{\gamma}_1(t_1) \quad (s_2, \bar{h}, s'_2) \in \tilde{\gamma}_2(t_2) \quad h \in \mathcal{Act}}{\gamma(t_1, t_2) = ((s_1, s_2), \tau, (s'_1, s'_2))} \\
\\
\frac{\gamma_1(t_1) = (s_1, \alpha, s'_1)}{\gamma(t_1, s_2) = ((s_1, s_2), \alpha, (s'_1, s_2))} \quad \frac{\gamma_2(t_2) = (s_2, \alpha, s'_2)}{\gamma(s_1, t_2) = ((s_1, s_2), \alpha, (s_1, s'_2))} \\
\\
- e(s_1, s_2) = \begin{cases} \{\tau\} & \text{if } e_{(s_1, s_2)}^- \wedge e_{(s_1, s_2)}^+ \\ \{\tau\} \cap (e_1(s_1) \cup e_2(s_2)) & \text{if } \neg e_{(s_1, s_2)}^- \wedge e_{(s_1, s_2)}^+ \\ \{\tau\} \cup e_1(s_1) \cup e_2(s_2) & \text{if } e_{(s_1, s_2)}^- \wedge \neg e_{(s_1, s_2)}^+ \\ e_1(s_1) \cup e_2(s_2) & \text{if } \neg e_{(s_1, s_2)}^- \wedge \neg e_{(s_1, s_2)}^+ \end{cases} \\
\text{with } e_{(s_1, s_2)}^- \text{ if communication is guaranteed, i.e., } e(s_1) \cap e(s_2) \cap \mathcal{Act} \neq \emptyset, \text{ and} \\
e_{(s_1, s_2)}^+ \text{ holds if communication is possible, i.e., } \mathcal{O}(s_1) \cap \mathcal{O}(s_2) \cap \mathcal{Act} \neq \emptyset, \\
- \mathbb{S} = \bigcup_{i \in \{1, 2\}} \{(\{t \in T \mid \pi_i(t) \in E\}, \{t \in T \mid \pi_i(t) \in F\}) \mid (E, F) \in \mathbb{S}_i\} \cup \\
\{(\text{Co}_1 \cup \text{NC}_1, \text{Co}^\parallel), (\text{Co}_2 \cup \text{NC}_2, \text{Co}^\parallel), (\text{Sy} \cup \text{NS}, \text{Co}^\parallel)\} \\
\text{where for } i \in \{1, 2\}, \text{Co}^\parallel = \{t \in T \mid \text{lab}(t) = \tau\} \text{ is the set of all transitions} \\
\text{obtained by internal computations, } \text{Co}_i = \{t \in \text{Co}^\parallel \mid \pi_i(t) \in T_i \wedge \text{lab}(\pi_i(t)) = \tau\} \\
\text{are those transitions obtained by internal synchronization of } \mathcal{M}_i, \text{Sy} = \\
\{t \in \text{Co}^\parallel \mid \pi_1(t) \in T_1 \wedge \text{lab}(\pi_1(t)) \neq \tau\} \text{ are those transitions obtained by} \\
\text{synchronization of } \mathcal{M}_1 \text{ and } \mathcal{M}_2, \text{NC}_i = \{t \in T \mid \tau \notin e(\pi_i(\text{src}(t)))\} \text{ are} \\
\text{those transitions where no internal computation is guaranteed in the } i\text{-th} \\
\text{component of its source, and } \text{NS} = \{t \in T \mid \neg e_{(\text{src}(t))}^-\} \text{ are those transition} \\
\text{where no synchronization is guaranteed in its source.}
\end{array}$$

A label is explicitly modeled if it is explicitly modeled by both sides. The transitions of the parallel composition are (i) the tuple of the transitions from both sides, if they correspond to communication between the two components and (ii) the transitions of each side not corresponding to communication between the two sides combined with all states of the other side. The latter kind of transitions is described by the last two rules in Table 2, whereas the first rule of this table describes the synchronization between the two components, which yields an internal computation. A computation is guaranteed in state (s_1, s_2) if at least one side guarantees a computation or an synchronization is guaranteed. A communication action is guaranteed if it is guaranteed by at least one side and no synchronization is possible. The latter point is reasonable, since the scheduler for communication may always favor a synchronization instead of an external communication.

The Streett condition of each side is preserved and the scheduler gives (if infinite computations take place) every component as well as every synchronization an infinite number of opportunities to execute, i.e., these computations infinitely often occur or they are infinitely often disabled. These fairness constraints correspond to weak fairness. Strong fairness, where, e.g., one compo-

ment executes infinitely often unless its computation is continuously disabled at the global scheduling points (i.e., points different from communication with the environment), can be obtained as follows: Replace the last three tuples in \mathbb{S} by $\{(Co_1, PC_1), (Co_2, PC_2), (Sy_1, PH)\}$, where PH (PC_i) consists of those transitions where an asynchronization is possible in (the i -th component of) its source. Note that in case of communications the scheduler (for the weak as well for the strong variant) is not completely fair: If both sides always provide h then it is possible that the environment only communicates via h with the left component. Extra constraints can be added to \mathbb{S} in order to restrict the scheduler further. Also different weak fairness constraints can be defined, e.g., by taking only those position into account, where an internal computation takes place (restrict NC_i and NS to elements from Co^\parallel and add into the first components of the Streett pairs those transitions where no internal computation is guaranteed at their sources). The kind of application determines which parallel operator is the appropriate one.

It is easily seen that $\mathcal{M}_1 \parallel \mathcal{M}_2$ indeed yields an STS and that \parallel is commutative (e^- and e^+ turns out to be symmetric). An example of parallel composition is given in Figure 2. As seen in that example, the parallel composition of two

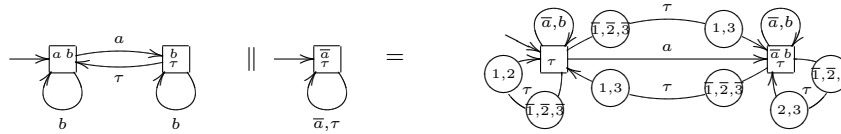


Fig. 2. Parallel composition of two STS. For notations we refer to Figure 1.

concrete STS does not yield in general a concrete STS. This is reasonable, since the distribution of the concurrent computation is not yet given. Refinement and satisfiability is preserved under parallel composition:

Theorem 9 (Refinement preservation). *Let \mathcal{M}_1 , \mathcal{M}'_1 , \mathcal{M}_2 , and \mathcal{M}'_2 be STSs with \mathcal{M}_1 refines \mathcal{M}'_1 and \mathcal{M}_2 refines \mathcal{M}'_2 . Then $\mathcal{M}_1 \parallel \mathcal{M}_2$ refines $\mathcal{M}'_1 \parallel \mathcal{M}'_2$.*

Theorem 10 (Satisfiability preservation). *Suppose \mathcal{M}_1 and \mathcal{M}_2 are two satisfiable STS. Then $\mathcal{M}_1 \parallel \mathcal{M}_2$ is satisfiable.*

4 EF-Logic

We define a satisfaction relation between our models and a tree automata version similar to [26]. An automaton description of the logic rather than a BNF-grammar is used, since this allows an appropriate satisfaction definition via games even if the model has fairness constraints.

Definition 11 (EF automata). *An exists-forall automaton (EF automaton) is a tuple $\mathcal{A} = (Q, q^i, \delta, \Theta)$, where*

- $(q \in)Q$ is a finite, nonempty set of states,
- $q^i \in Q$ is its initial automaton state,
- δ is a transition relation, which maps an automaton state to one of the following forms, where q, q_1, q_2 are automaton states and $\alpha \in \text{Act} \cup \{\tau, \tilde{\Delta}\}$:
 $\text{true} \mid \text{false} \mid q \mid q_1 \tilde{\wedge} q_2 \mid q_1 \tilde{\vee} q_2 \mid \langle \alpha \rangle q \mid [\alpha]q$, and
- $\Theta: Q \rightarrow \mathbb{N}$ is an acceptance condition with finite image.

\mathcal{A} is guarded if every cycle in the underlying graph of automaton \mathcal{A} contains an element that is labeled with $\langle \alpha \rangle$ or $[\alpha]$ for some α .

Label q only moves to a next state; it is used to obtain effective transformation of fixpoint formulas in terms of a BNF-grammar representation into a EF automaton representation [26]. $\tilde{\wedge}$ ($\tilde{\vee}$) corresponds to the logical *and* (respectively, *or*). Formula $\langle \alpha \rangle q$ with $\alpha \neq \tilde{\Delta}$ means that α is present and after its execution the property of q is guaranteed to hold. Its dual formula $[\alpha]q$ with $\alpha \neq \tilde{\Delta}$ indicates that either no α is possible or after the execution of α the property of q is guaranteed to hold. Furthermore, $[\tilde{\Delta}]q$ indicates that after any possible communication the property of q is guaranteed to hold. Consequently, its dual operator $\langle \tilde{\Delta} \rangle q$ holds if a communication is possible such that the property of q is guaranteed to hold afterwards. In other words, $[\tilde{\Delta}]q$ ($\langle \tilde{\Delta} \rangle q$) encodes some special infinite conjunctions (respectively, disjunctions).

Definition 12 (Dual automaton). *The dual EF automaton of an EF automaton \mathcal{A} , written $\mathcal{A}^{\text{dual}}$, is $(Q, q^i, \delta^{\text{dual}}, \Theta^{\text{dual}})$, where $\forall q: \Theta^{\text{dual}}(q) = \Theta(q) + 1$ and δ^{dual} is obtained from δ by replacing true by false , $\tilde{\wedge}$ by $\tilde{\vee}$, $\langle \alpha \rangle$ by $[\alpha]$, and vice versa.*

An alternating tree automaton and its dual one is depicted in Figure 3. Throughout this paper, we restrict ourselves without loss of generality to guarded

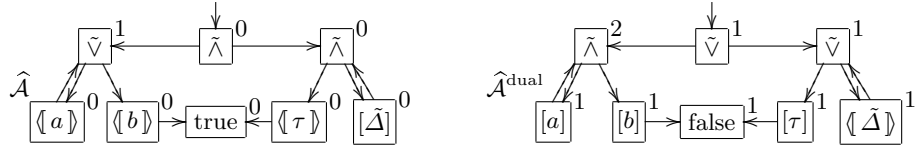


Fig. 3. An alternating tree automata and its dual one. Accepting values are depicted next to states. $\hat{\mathcal{A}}$ says that (i) if only a happens, b has to be possible after a finite number of steps and (ii) after any communication a computation remains guaranteed. $\hat{\mathcal{A}}^{\text{dual}}$ states that (i) after any a -communication b is never enabled or (ii) there is a finite sequence of communications such that no computation is possible thereafter.

automata (formulas) [11]. Also, for any bounded sequence \mathbf{n} of elements in \mathbb{N} we write $\text{sup}(\mathbf{n})$ for the largest m that occurs in \mathbf{n} infinitely often. Let $\text{map}(f, \Phi)$ be the sequence obtained from the sequence Φ by applying function f to all elements of Φ pointwise. In the following, we give a satisfaction definition generalizing the intuition of satisfaction on concrete STS to general STS such that

Table 3. Moves of satisfaction game at configuration $(t, q) \in T \times Q$, specified through a case analysis on the value of $\delta(q)$. Satisfaction plays are sequences of configurations generated thus.

true: is won by Player I.	false: is won by Player II.
q' : the next configuration is (t, q') .	
$q_1 \wedge q_2$: Player II picks a q' from $\{q_1, q_2\}$; the next configuration is (t, q') .	
$q_1 \vee q_2$: Player I picks a q' from $\{q_1, q_2\}$; the next configuration is (t, q') .	
$[\alpha]q'$ and $\alpha \neq \tilde{\Delta}$: Player II picks t' such that $\text{tar}(t) = \text{src}(t')$ and $\text{lab}(t') = \alpha \vee (\alpha \in \mathcal{Act} \setminus \Omega \wedge \text{lab}(t') = \Delta)$; the next configuration is (t', q') .	
$[\tilde{\Delta}]q'$: Player II picks t' such that $\text{tar}(t) = \text{src}(t')$ and $\text{lab}(t') \in \mathcal{Act} \cup \{\Delta\}$; the next configuration is (t', q') .	
$\langle \alpha \rangle q'$ and $\alpha \neq \tilde{\Delta}$: Player II wins if $\alpha \notin e(\text{tar}(t))$; otherwise the play continues as in case $[\alpha]q'$.	
$\langle \tilde{\Delta} \rangle q'$: Player I picks $h \in \mathcal{Act}$; the play continues as in case $\langle h \rangle q'$.	

it is preserved under refinement. Note that this satisfaction relation is also the suitable approximative satisfaction relation for ready simulation (remind that transition systems can be embedded such that our refinement restricted to this embedding coincides with ready simulation).

Definition 13 (Satisfaction).

- *Finite satisfaction plays for model \mathcal{M} and EF automaton \mathcal{A} have the rules and winning conditions as stated in Table 3. An infinite play Φ is a win for Player I iff $[\text{Acc}_{\mathcal{D}}(\Phi[1]) \Rightarrow \text{sup}(\text{map}(\Theta, \Phi[2]))]$ is even; otherwise, it is won by Player II.*
- *The model \mathcal{M} satisfies the automaton \mathcal{A} , written as $\mathcal{M} \models \mathcal{A}$, iff Player I has a strategy for the corresponding satisfaction game between \mathcal{M} and \mathcal{A} such that for any $t \in T$ with $\text{tar}(t) \in S^i$ Player I wins all satisfaction plays started at (t, q^i) with her strategy.*

We give some comments on the non standard steps used in Table 3: In $[\alpha]q'$ with $\alpha \neq \tilde{\Delta}$ any transition labeled with α and in case the label is not explicitly modeled also any transition labeled with the default label has to be matched, since it can be refined to one having this label. If $\alpha = \tilde{\Delta}$ then any transition labeled different from τ has to be matched. In $\langle \alpha \rangle q'$ the transition has to be existent, which is guaranteed by $\alpha \in e(\text{tar}(t))$. Furthermore, all possible transitions have to be matched, which is handled via $[\alpha]q'$. The latter point is necessary, since a concrete refinement is deterministic and the corresponding concrete transition only has to be matched by one transition at the abstraction. In $\langle \tilde{\Delta} \rangle q'$ a communication action has to be existent that always leads to a state satisfying q' . The acceptance condition for satisfaction plays between a model \mathcal{M} and an automaton \mathcal{A} is a variant of those familiar from the literature: An infinite play Φ is a win for Player I iff either the projection of Φ onto the automata \mathcal{A} is accepting in \mathcal{A} (i.e., $\text{sup}(\text{map}(\Theta, \Phi[2]))$ is even), or the projection of Φ onto \mathcal{M} is non-accepting in \mathcal{M} (i.e., $\neg \text{Acc}_{\mathcal{D}}(\Phi[1])$). Note that the possible infinite

choice in rule $\langle \tilde{\Delta} \rangle$ can be easily reduced to a finite one whenever \mathcal{M} is finite. Furthermore, negation of a formula is modeled via the dual automaton:

Theorem 14 (2-valuedness). *Suppose \mathcal{A} is a guarded EF automaton and $\ddot{\mathcal{M}}$ is a concrete STS, then $\ddot{\mathcal{M}} \models \mathcal{A} \iff \neg(\ddot{\mathcal{M}} \models \mathcal{A}^{\text{dual}})$.*

The satisfaction relation on general STS is inherently 3-valued, since (i) any instance $\mathcal{M} \models \mathcal{A}$ attempts to establish whether all refinements \mathcal{M}' of \mathcal{M} satisfy \mathcal{A} and (ii) some, but not all, refinements of \mathcal{M} may satisfy \mathcal{A} in q . The winning conditions for the satisfaction game are Rabin conditions as they have form $[\text{Streett} \Rightarrow \text{RabinChain}]$ which reduces to Rabin; so deciding $\mathcal{M} \models \mathcal{A}$ is in NP for finite models. We prove soundness of $\mathcal{M} \models \mathcal{A}$ as an approximation of the EXPTIME-hard relation which asks whether all concrete STS $\ddot{\mathcal{M}}$ that refine \mathcal{M} satisfy \mathcal{A} . In particular, we approximate the PSPACE-complete LTL model checking problem [27], since LTL, as well as the linear μ -calculus, can be embedded into our logic if we transform unlabeled transition systems having predicates as STS by (i) labeling all transitions with τ , (ii) encoding predicates via labeled transitions, and (iii) putting $e(s) = \mathcal{O}(s)$.³

Theorem 15 (Soundness). *Let \mathcal{A} be a guarded EF automaton and \mathcal{M}_1 and \mathcal{M}_2 be two STSs such that \mathcal{M}_1 refines \mathcal{M}_2 and $\mathcal{M}_2 \models \mathcal{A}$. Then $\mathcal{M}_1 \models \mathcal{A}$.*

Example 16. The automaton $\hat{\mathcal{A}}$ of Figure 3 is satisfied by the STS $\hat{\mathcal{N}}$ of Figure 1, and thus by Theorem 15 also by \mathcal{N} of Figure 1. The three-valuedness of the satisfaction relation can be seen, since neither all concrete refinements of $\hat{\mathcal{N}}$ satisfy $\langle [c] \rangle \text{true}$ nor its dual one, $[c] \text{false}$. The approximation of our satisfaction definition is seen by the fact that $\langle [c] \rangle \text{true} \vee [c] \text{false}$ is not satisfied by $\hat{\mathcal{N}}$, but by all of its concrete refinements.

Corollary 17. *A STS \mathcal{M} is not satisfiable if \mathcal{M} satisfies an EF automaton as well as its dual one.*

5 Application

Here, we present two small examples illustrating the advantages of our setting. One in the context of modeling the other one in the context of verification.

5.1 Modeling

Suppose a program as well as a firewall that prechecks the incoming messages for the program are executed on a single processor computer. The program can beside its internal computation always react on an incoming message from the

³ To be precise, we have NP over the number of transitions, whereas LTL is PSPACE-complete over the number of states. Nevertheless, we can straightforwardly adapt our satisfaction definition w.r.t. state-based fairness. In other words, we really have an NP approximation of the PSPACE-complete LTL model checking problem.

firewall. This is done via handshake communication of the firewall action \bar{p} (pass) and the program action p . After such a handshake communication the program directly replies to the environment via action \bar{r} (reply). The firewall is able to receive a message from the environment via action g (get) at the initial state. Then it either drops it (and goes back to the initial state) or \bar{p} is enabled. In the later case, further get actions can be received. Additionally, after \bar{p} the initial state is reached or messages can still be passed on. Furthermore, at every point in time internal computation is possible. The program and firewall are modeled by the STS of Figure 4 (a), respectively (b).

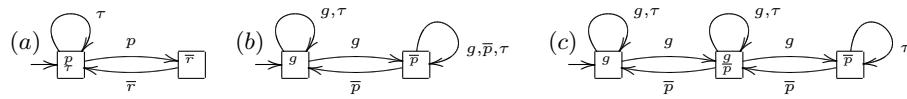


Fig. 4. (a) A program, (b) an abstract firewall model, and (c) a less abstract firewall modeled as STS, where $\Omega = \{p, \bar{p}, g, \bar{r}\}$ in all three models

As already illustrated in Figure 2, Streett fairness constraint are obtained after parallel composition of the models of the program and the firewall. This is reasonable, since we are only interested in operating systems that give the program as well as the firewall infinitely often the opportunity to execute. It is straightforward to see that the current firewall specification satisfies the EF automata from Figure 5. Now the model of the firewall is made more precise

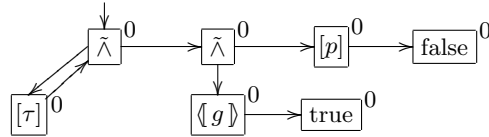


Fig. 5. An alternating tree automata. It says that (i) a message can only be passed to the program if the firewall received one and (ii) if no message is received so far, the firewall must be able to receive one.

by modeling one having a message buffer of size two. This leads to the STS of Figure 4 (c). Then one can see that this is indeed a refinement of the previous firewall model. Hence, it also satisfies the EF automaton of Figure 5 by Theorem 15.

5.2 Abstraction

Consider the firewall implementation of Figure 6 (a), where (i) the message is added to the buffer, x , if the maximal buffer size, y , is not exceeded, otherwise

the message is lost; (ii) the maximal buffer size can be extended by one via an internal computation if the maximal buffer size is currently reached; and (iii) a message in a buffer can be removed and passed on. It is obvious that this firewall

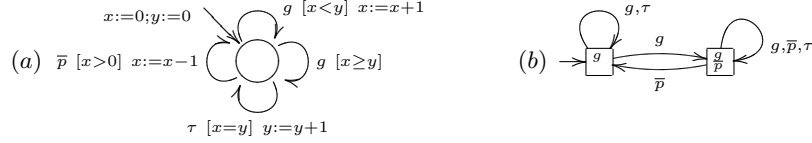


Fig. 6. (a) A firewall implementation in terms of a state machine and (b) its abstraction w.r.t. predicate $x = 0$ in terms of STSs, where $\Omega = \{p, \bar{p}, g, \bar{r}\}$

is indeed deterministic and that it satisfies the property of the automaton from Figure 5. Nevertheless, it cannot be automatically verified, since the underlying state space is infinite. A predicate abstraction [23] technique yielding STSs, which is not yet formally defined, is illustrated in Figure 6 (b). This abstract STS is finite and indeed satisfies the automaton from Figure 5 as required. Note that modal transition systems [13] are not sufficient as abstract model to verify this property by using predicate $x = 0$ for abstraction, since no outgoing must transition from the abstract state $x = 0$ exists. In other words at least unnecessary complex abstractions (greater state space) have to be derived. That modal transition also fails in some cases is illustrated in Figure 7. Disjunctive modal transition systems [14], which are sufficient, are unnecessary complex, since additional must hypertransitions are needed.

In order to handle arbitrary liveness properties the predicate abstraction can be extended with ranking functions, as it is done in [15], where arbitrary transitions systems are the implementations. By this abstraction technique, Streett acceptance condition naturally occur by construction. Note that by using this

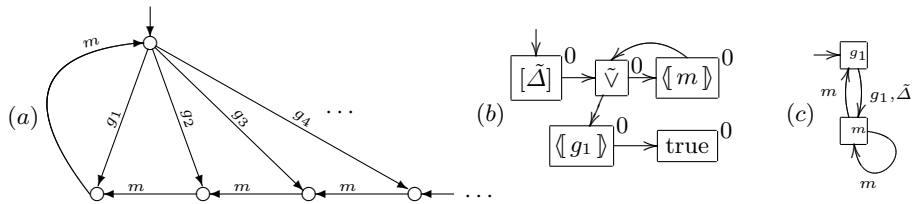


Fig. 7. A deterministic transition systems (a) for which no finite abstraction in terms of modal transition system exists that satisfies EF-automata (b), which describes that after the first action either infinitely many m -actions are possible or g_1 is possible after finitely many m -actions. On the other hand, the STS (c) satisfies EF-automata (b) and is an abstraction of (a).

ranked predicate abstraction technique, STS are complete in the sense that if a deterministic transition satisfies an EF automaton \mathcal{A} , then there is a ranked predicate abstraction such that the obtained STS abstraction also satisfies \mathcal{A} .

6 Conclusion

Synchronously-communicating transition systems (STS), which are a suitable setting for modeling and reasoning about deterministic transition systems, were presented. In particular, we presented a refinement notion, fair parallel composition, and 3-valued satisfaction on a logic for STSs. Therefore, whenever implementations behave deterministically and synchronous communication is considered, STSs (i) are appropriate as semantical model of programming and modeling languages and (ii) yield an appropriate foundation for verification via abstraction as well as via compositional reasoning.

Future work will be an extension of STSs such that also asynchronous and shared variable communication, as well as termination is possible. Here, a combination of transition systems with termination [28, 29] and I/O-automata [30] might be a good starting point. Furthermore, an adaption where sets of actions rather than single actions are used as labels is also of interest, since this allows to model that communication on different actions can take place (via parallel components) at the same time step.

Acknowledgments. This work is in part financially supported by the DFG FE 942/1-1 project Refism and by the EU IST-33826 project CREDO.

References

- [1] Object Management Group: UML Superstructure Specification, v2.0 formal/05-07-04. (2005)
- [2] Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag, New York (1992)
- [3] Park, D.: Concurrency and automata on infinite sequences. In Deussen, P., ed.: Theoretical Computer Science. Volume 104 of LNCS., Springer (1981) 167–183
- [4] Lynch, N., Vaandrager, F.: Forward and backward simulations: I. Untimed systems. Information and Computation **121** (1995) 214–233
- [5] Alur, R., Henzinger, T., Kupferman, O., Vardi, M.Y.: Alternating refinement relations. In Sangiorgi, D., de Simone, R., eds.: CONCUR. Volume 1466 of LNCS., Springer (1998) 163–178
- [6] Bloom, B., Istrail, S., Meyer, A.: Bisimulation can't be traced. J. ACM **42**(1) (1995) 232–268
- [7] Walker, D.J.: Bisimulation and divergence. Information and Computation **85**(2) (1990) 202–241
- [8] Milner, R.: A modal characterization of observable machine-behaviour. In Aste-siano, E., Böhm, C., eds.: CAAP'81. Volume 112 of LNCS., Springer (1981) 25–34
- [9] Glabbeek, R.v.: The linear time–branching time spectrum I. The semantics of concrete, sequential processes. [31] 3–99

- [10] Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In Kozen, D., ed.: Logic of Programs. Volume 131 of LNCS., Springer (1982) 52–71
- [11] Kozen, D.: Results on the propositional μ -calculus. *Theor. Comput. Sci.* **27** (1983) 333–354
- [12] Janin, D., Walukiewicz, I.: Automata for the modal mu-calculus and related results. In Wiedermann, J., Hájek, P., eds.: Mathematical Foundations of Computer Science. Volume 969 of LNCS., Springer (1995) 552–562
- [13] Larsen, K.G., Thomsen, B.: A modal process logic. In: LICS, IEEE Computer Society Press (1988) 203–210
- [14] Larsen, K.G., Xinxin, L.: Equation solving using modal transition systems. In: LICS, IEEE Computer Society Press (1990) 108–117
- [15] Fecher, H., Huth, M.: Ranked predicate abstraction for branching time: Complete, incremental, and precise. In Graf, S., Zhang, W., eds.: ATVA. Volume 4218 of LNCS., Springer (2006) 322–336
- [16] Shoham, S., Grumberg, O.: 3-valued abstraction: More precision at less cost. In: LICS, IEEE Computer Society Press (2006) 399–410
- [17] Dams, D., Namjoshi, K.S.: The existence of finite abstractions for branching time model checking. In: LICS, IEEE Computer Society Press (2004) 335–344
- [18] Dams, D., Namjoshi, K.S.: Automata as abstractions. [32] 216–232
- [19] Huth, M.: Refinement is complete for implementations. *Formal Asp. Comput.* **17**(2) (2005) 113–137
- [20] Grumberg, O., Lange, M., Leucker, M., Shoham, S.: *Don't know* in the μ -calculus. [32] 233–249
- [21] de Alfaro, L., Godefroid, P., Jagadeesan, R.: Three-valued abstractions of games: Uncertainty, but with precision. In: LICS, IEEE Computer Society Press (2004) 170–179
- [22] Henzinger, T.A., Majumdar, R.: Fair bisimulation. In Graf, S., Schwartzbach, M.I., eds.: TACAS. Volume 1785 of LNCS., Springer (2000) 299–314
- [23] Graf, S., Saidi, H.: Construction of abstract state graphs with PVS. In Grumberg, O., ed.: CAV. Volume 1254 of LNCS., Springer (1997) 72–83
- [24] Hoare, C.A.R.: *Communications Sequential Processes*. International Series in Computer Science. Prentice Hall (1985)
- [25] Milner, R.: *Communication and Concurrency*. International Series in Computer Science. Prentice Hall (1989)
- [26] Wilke, Th.: Alternating tree automata, parity games, and modal μ -calculus. *Bull. Soc. Math. Belg.* **8**(2) (May 2001) 359–391
- [27] Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. *Journal of the ACM* **32**(3) (1985) 733–749
- [28] Bergstra, J.A., Fokkink, W., Ponse, A.: Process algebra with recursive operations. [31] 333–389
- [29] Fecher, H., Majster-Cederbaum, M.: Event structures for arbitrary disruption. *Fundamenta Informaticae* **68**(1,2) (2005) 103–130
- [30] Lynch, N., Tuttle, M.: An introduction to input/output automata. *CWI-Quarterly* **2**(3) (1989) 219–246
- [31] Bergstra, J.A., Ponse, A., Smolka, S.A., eds.: *Handbook of Process Algebra*. North-Holland (2001)
- [32] Cousot, R., ed.: *Verification, Model Checking, and Abstract Interpretation*, 6th International Conference, VMCAI 2005, Paris, France, January 17-19, 2005, Proceedings. In Cousot, R., ed.: VMCAI. Volume 3385 of LNCS., Springer (2005)