On the Expressiveness of Refinement Settings*

Harald Fecher[1], David de Frutos-Escrig[2], Gerald Lüttgen[3], and Heiko Schmidt[4]

[1] Albert-Ludwigs-Universität Freiburg, Germany, fecher@informatik.uni-freiburg.de
[2] Universidad Complutense Madrid, Spain, defrutos@sip.ucm.es
[3] University of York, U.K., gerald.luettgen@cs.york.ac.uk
[4] Christian-Albrechts-Universität Kiel, Germany, hsc@informatik.uni-kiel.de

Abstract. Embedded-systems designers often use transition system-based notations for specifying, with respect to some refinement preorder, sets of deterministic implementations. This paper compares popular such refinement settings — ranging from transition systems equipped with failure-pair inclusion to disjunctive modal transition systems — regarding the sets of implementations they are able to express. The paper's main result is an expressiveness hierarchy, as well as language-preserving transformations between various settings. In addition to system designers, the main beneficiaries of this work are tool builders who wish to reuse refinement checkers or model checkers across different settings.

1 Introduction

Many of today's embedded systems employ control software that runs on specialized computer chips, performing dedicated tasks often without the need of an operating system. System designers typically specify such software using notations based on labeled transition systems: a possibly nondeterministic specification allows for a set of deterministic implementations, amenable to quality checks via testing or model checking. Verifiers benefit from the reduced state space in possibly nondeterministic abstractions from deterministic implementations. Choosing a suitable *refinement setting* for a given application in hand depends on various aspects, e.g., expressiveness, conciseness, and verification support.

In the concurrency-theory literature many refinement settings have been studied, with a focus on compositionality and full abstraction of, and logical characterizations and decision procedures for the underlying refinement preorders, see, e.g., [17] and the numerous references therein. Less attention has been paid to questions of expressiveness. In the context of top-down development, where sets of allowed implementations are specified at different design levels, it is of special interest to characterize the expressible sets of implementations. In general, the more sets a formalism can describe, the more expressive it is and the more flexibility a system designer has by describing finer sets of implementations.

* Research support provided by DFG (FE 942/2-1, RO 1122/12-2), EPSRC (EP/E034853/1) and MEC (TIN2006-15660-C02-01, TIN2006-15578-C02-01).

We perform the expressiveness comparison using language-preserving transformations, where the *language* of a refinement setting is the expressible set of deterministic implementations. This is analogous for trace-based languages, where language-preserving transformations have been developed between automata that differ in their fairness notion (Büchi, Muller, Rabin, Streett, parity), see [19] and the references therein.

Language-preserving transformations are valuable in the context of model checking, too, where abstract models reduce the size of the state space, while at the same time staying amenable to quality checks: if a property is model checked for an abstract model, then it is guaranteed to hold for each of its implementations. Therefore, a model checking tool over a refinement setting $\mathfrak{A}_1$ can be reused for another setting $\mathfrak{A}_2$ if every model from $\mathfrak{A}_2$ can be converted into an equivalent model from $\mathfrak{A}_1$ that defines the same language.

This paper studies and compares the expressiveness of almost a dozen refinement settings designed for deterministic transition systems. To do so, several intricate language-preserving transformations are developed. We also show how algorithms for checking a specification's consistency and for checking refinement can be derived from our transformations. While the expressiveness hierarchy is valuable for informing system designers on their choice of refinement setting, our transformations allow tool builders to reuse their refinement checkers or model checking algorithms across different settings.

2 Basic notions: refinement settings and expressiveness

To begin with, let $\mathcal{L}$ denote a *finite* set of possible actions (i.e., transition labels[5]), $|M|$ the cardinality of a set M, and $\mathcal{P}(M)$ its power set. M^* stands for the set of finite sequences over M, and $\cdot$ for sequence concatenation. For $R \subseteq M_1 \times M_2$, we write $m_1 R m_2$ if $(m_1, m_2) \in R$ and let $R^{-1} = \{(m_2, m_1) \mid (m_1, m_2) \in R\}$. If $X \subseteq M_1$, $Y \subseteq M_2$, we let $X \circ R = \{m_2 \in M_2 \mid \exists m_1 \in X : (m_1, m_2) \in R\}$ and $R \circ Y = \{m_1 \in M_1 \mid \exists m_2 \in Y : (m_1, m_2) \in R\}$, which are, e.g., used to describe the successors, resp. predecessors, of a transition relation. For $\rightsquigarrow \subseteq M_1 \times \mathcal{L} \times M_2$, we define the set of outgoing labels of $m_1 \in M_1$ by $\mathbf{O}_{\rightsquigarrow}(m_1) = \{a \in \mathcal{L} \mid \exists m_2 \in M_2 : m_1 \overset{a}{\rightsquigarrow} m_2\}$, and let $\overset{a}{\rightsquigarrow}$ stand for the relation $\{(m_1, m_2) \mid m_1 \overset{a}{\rightsquigarrow} m_2\}$. Relation $\rightsquigarrow$ is deterministic if $\forall m, a : |\{m\} \circ \overset{a}{\rightsquigarrow}| \leq 1$. Depending on the context, a function $f : M_1 \to M_2$ is also interpreted as a higher order function from $\mathcal{P}(M_1)$ to $\mathcal{P}(M_2)$ with $f(X) = \{f(m_1) \mid m_1 \in X\}$.

Definition 1. *A* transition system *(TS) $\mathcal{T}$ is a tuple $(S, S^0, \to)$ such that S is its set of states, $S^0 \subseteq S$ its non-empty set of initial states, and $\to \subseteq S \times \mathcal{L} \times S$ its transition relation. $\mathcal{T}$ is* finite *if $|S| < \infty$, and it is* deterministic *if $|S^0| = 1$ and $\to$ is deterministic. $\mathbf{T}_{\text{det}}$ denotes the set of all deterministic transition systems. (DetTSs), which we also call* implementations*.*

[5] State predicates can be encoded via transition labels and are therefore omitted.

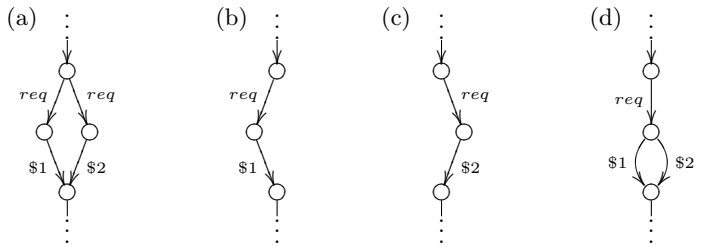


Fig. 1. Vending machine example.

Note that DetTSs (up to equivalence) are the natural model for implementations in the context of open systems, where communication with the environment takes place via actions: the executions behave deterministically up to the behavior of the environment, which can only control the kind of communication (i.e., which action is executed). Since it is unusual in system modeling to specify exactly one implementation, abstract models are used to describe sets of implementations. In the context of closed systems, this is commonly done by a Kripke structure or by an automaton (it describes a set of traces, known as its language). Analogously we are looking for abstract models in the context of open systems, i.e., looking for models that describe sets of DetTSs, as illustrated by the following example:

Example 1. Consider a part of a vending machine specification, as shown in Fig. 1(a). The TS is nondeterministic, since following a *req* action it can either ask for \$1 or for \$2. This nondeterminism is desired, because the specification should be refinable to either a cheap machine (requesting \$1, shown in Fig. 1(b)) or an expensive machine (requesting \$2, shown in Fig. 1(c)). However, it depends on the employed refinement setting whether these two implementations can be modeled without also modeling the undesired implementation shown in Fig. 1(d), which gives the user the choice whether to pay \$1 or \$2. For instance, in failure pair semantics [4] model (a) has the undesired implementation (d), whereas in ready pair semantics [27] it has not.

Many equivalences on TSs are introduced in the literature, see [17] for an overview. Their preorders lead to different refinement notions; however, if restricted to DetTSs, they collapse as was first observed by Park [28] and further examined by Engelfriet [10]. Therefore, it is sufficient to present as equivalence notion on DetTSs only one of them, e.g., *bisimulation*.

Definition 2. *$R \subseteq S_1 \times S_2$ is a* simulation *between two TSs $\mathcal{T}_1$ and $\mathcal{T}_2$ if $\forall s_1 \in S_1^0 : \exists s_2 \in S_2^0 : (s_1, s_2) \in R$, and for all $(s_1, s_2) \in R$, $a \in \mathcal{L}$ we have $\forall s_1' \in \{s_1\} \circ \xrightarrow{a}_1 : \exists s_2' \in \{s_2\} \circ \xrightarrow{a}_2 : s_1' R s_2'$, which can equivalently be written as $(\{s_1\} \circ \xrightarrow{a}_1) \subseteq R \circ (\{s_2\} \circ \xrightarrow{a}_2)$. We say that $\mathcal{T}_1$* is simulated by *$\mathcal{T}_2$ if there is a simulation R between $\mathcal{T}_1$ and $\mathcal{T}_2$. Further, we say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are* bisimilar *(or simply* equivalent*), and then write $\mathcal{T}_1 \equiv \mathcal{T}_2$, if there is a simulation R between $\mathcal{T}_1$ and $\mathcal{T}_2$ such that R^{-1} is a simulation between $\mathcal{T}_2$ and $\mathcal{T}_1$.*

We can now define refinement settings as families of *models*, into which DetTSs are embedded, with an order relating them.

Definition 3. *A* refinement setting $\mathfrak{A}$ *is a tuple* $(A, A^f, \preceq, h)$*, where* A *is a set of so called* models*,* $A^f \subseteq A$ *is a distinguished subclass of so called* finite models*,* $\preceq$ *is a preorder on* A*, called* refinement*, and* $h : \mathbf{T}_{\mathrm{det}} \to A$ *is an* embedding*, i.e.,* $\forall \mathcal{T}_1, \mathcal{T}_2 \in \mathbf{T}_{\mathrm{det}} : \mathcal{T}_1 \equiv \mathcal{T}_2 \Leftrightarrow h(\mathcal{T}_1) \preceq h(\mathcal{T}_2)$*. The* language $\mathfrak{A}(\alpha)$ *of a model* $\alpha \in A$ *(also called its* set of implementations *or its* possible worlds*) is the set of refining implementations of* α*, i.e.,* $\{\mathcal{T} \in \mathbf{T}_{\mathrm{det}} \mid h(\mathcal{T}) \preceq \alpha\}$*.*

Though this is not required by the definition, it is best to first think of $h(\mathbf{T}_{\mathrm{det}})$ as the bottom elements of the refinement preorder $\preceq$. They correspond to the implementations. Then, equivalence on DetTSs ($\mathcal{T}_1 \equiv \mathcal{T}_2$) must imply "refinement equivalence", i.e., $h(\mathcal{T}_1) \preceq h(\mathcal{T}_2)$, and, directly implied by equivalence of $\equiv$, $h(\mathcal{T}_2) \preceq h(\mathcal{T}_1)$ on these bottom elements. For the other direction, refinement between models on the implementation level must be enough to establish equivalence on DetTSs, which makes sure that every implementation can be specified alone, without any other, non-equivalent refining implementations.

Now in fact, the definition also allows non-implementations below implementation level, i.e., below elements of $h(\mathbf{T}_{\mathrm{det}})$. These appear, e.g., in (disjunctive) mixed transition systems [26] and are unsatisfiable, i.e., have an empty language. Our notion of expressiveness is based on the expressible languages of a refinement setting:

Definition 4. *Let* $\mathfrak{A} = (A, A^f, \preceq, h)$ *be a refinement setting. A* language-preserving transformation *from* $\mathfrak{A}_1$ *to* $\mathfrak{A}_2$ *is a total function* $f : A_1^f \to A_2^f$ *such that* $\mathfrak{A}_1(\alpha) = \mathfrak{A}_2(f(\alpha))$ *for all* $\alpha \in A_1^f$*. We say that* $\mathfrak{A}_1$ *is* at least as expressive as $\mathfrak{A}_2$ *if there is a language-preserving transformation from* $\mathfrak{A}_2$ *to* $\mathfrak{A}_1$*.*

Reconsider Ex. 1 where we claimed that model (a) expresses, with respect to ready pair semantics, implementations (b) and (c), whereas it also has the further implementation (d) with respect to failure pair semantics. If we show that there is also no other specification that expresses exactly (b) and (c) (up to equivalence) in failure pair semantics, we know that there can be no language-preserving transformation from ready pair semantics to failure pair semantics. To prove that ready pair semantics is more expressive than failure pair semantics, we also have to prove that every language expressible in failure pair semantics can also be expressed in ready pair semantics.

By Def. 4, language-preserving transformations are mapping *finite* abstract models to *finite* abstract models (though implementations may be infinite). We are especially interested in such mappings because they preserve the (direct) amenability to applications like model checking. Furthermore, expressiveness results for *infinite* models are mostly trivial, because infinite initial state sets can be used to describe any desired language (by dedicating one initial state to each desired implementation).

3 A wide collection of refinement settings

This section recalls popular refinement settings that have been studied in the literature, where models are either TSs, *synchronously communicating* TSs [12], *modal/mixed* TSs [25, 7], or *disjunctive modal/mixed* TSs [26]. It is easily checked that all these settings are indeed refinement settings.

Transition systems. First note that TS-based models equipped with trace inclusion or simulation, when taking h as the identity function, do not yield refinement settings, since DetTSs cannot be embedded into these refinement preorders, i.e., DetTSs $\mathcal{T}_1, \mathcal{T}_2$ can be found such that $h(\mathcal{T}_1) \preceq h(\mathcal{T}_2)$ but $\mathcal{T}_1 \not\equiv \mathcal{T}_2$, e.g., $\mathcal{T}_1 = \rightarrow\bigcirc$ and $\mathcal{T}_2 = \rightarrow\bigcirc\xrightarrow{a}\bigcirc$. Therefore, preorders in refinement settings must preserve, in both directions, the enabledness of actions when comparing DetTSs, e.g., every refinement of $\mathcal{T}_2$ above must have action a enabled in its root state, which $\mathcal{T}_1$ has not. We present refinement settings based on *failure pairs* (also called failures) [4], *failure traces* (also called refusal) [29], *ready pairs* (also called readiness) [27], *ready traces* [1], *possible worlds* [32], and *ready simulations* [2][6]. For a TS $\mathcal{T}$,

- a ready trace of $s \in S$ is a trace starting in s, together with the sets of enabled actions after every subtrace. Formally, the set of *ready traces* of $\mathcal{T}$ is the smallest set $\mathrm{Tr}^{\mathcal{T}}_{\mathrm{RT}} \subseteq S \times ((\mathcal{P}(\mathcal{L}) \cdot \mathcal{L})^* \cdot \mathcal{P}(\mathcal{L}))$ with $(s, \mathbf{O}_{\rightarrow}(s)) \in \mathrm{Tr}^{\mathcal{T}}_{\mathrm{RT}}$ and $((s', \sigma) \in \mathrm{Tr}^{\mathcal{T}}_{\mathrm{RT}} \wedge s \xrightarrow{a} s') \Rightarrow (s, \mathbf{O}_{\rightarrow}(s) a \sigma) \in \mathrm{Tr}^{\mathcal{T}}_{\mathrm{RT}}$, for any $s, s' \in S$, $a \in \mathcal{L}$.
- a ready pair of $s \in S$ is a trace starting in s, together with the set of enabled actions after the complete trace. Formally, the set of *ready pairs* $\mathrm{Tr}^{\mathcal{T}}_{\mathrm{R}} \subseteq S \times (\mathcal{L}^* \cdot \mathcal{P}(\mathcal{L}))$ is the set of traces from $\mathrm{Tr}^{\mathcal{T}}_{\mathrm{RT}}$ where each but the last element from $\mathcal{P}(\mathcal{L})$ is removed.
- a failure pair of $s \in S$ is a trace starting in s, together with a set of actions that are not enabled after the complete trace. Formally, the set of *failure pairs* $\mathrm{Tr}^{\mathcal{T}}_{\mathrm{F}} \subseteq S \times (\mathcal{L}^* \cdot \mathcal{P}(\mathcal{L}))$ is the set of traces that can be obtained by replacing the last element (the one from $\mathcal{P}(\mathcal{L})$) in a trace from $\mathrm{Tr}^{\mathcal{T}}_{\mathrm{R}}$ by any subset of its complement.
- a failure trace of $s \in S$ is a trace starting in s, together with, for every subtrace, a set of actions that are not enabled after the subtrace. Formally, the set of *failure traces* $\mathrm{Tr}^{\mathcal{T}}_{\mathrm{FT}} \subseteq S \times ((\mathcal{P}(\mathcal{L}) \cdot \mathcal{L})^* \cdot \mathcal{P}(\mathcal{L}))$ is the set of traces that can be obtained by replacing every element in $\mathcal{P}(\mathcal{L})$ from a trace in $\mathrm{Tr}^{\mathcal{T}}_{\mathrm{RT}}$ by any subset of its complement.

Now the refinement settings of *ready pair inclusion* $\mathbb{T}_{\mathrm{r}}$, *ready trace inclusion* $\mathbb{T}_{\mathrm{rt}}$, *failure pair inclusion* $\mathbb{T}_{\mathrm{f}}$, *failure trace inclusion* $\mathbb{T}_{\mathrm{ft}}$, *ready simulation* $\mathbb{T}_{\mathrm{rs}}$ and *possible worlds inclusion* $\mathbb{T}_{\mathrm{pw}}$ consist of TSs, the identity embedding on DetTSs, and the refinement notion given by ready pair inclusion: $(S^0_1 \circ \mathrm{Tr}^{\mathcal{T}_1}_{\mathrm{R}}) \subseteq (S^0_2 \circ \mathrm{Tr}^{\mathcal{T}_2}_{\mathrm{R}})$; resp. ready trace inclusion: $(S^0_1 \circ \mathrm{Tr}^{\mathcal{T}_1}_{\mathrm{RT}}) \subseteq (S^0_2 \circ \mathrm{Tr}^{\mathcal{T}_2}_{\mathrm{RT}})$; resp. failure pair inclusion: $(S^0_1 \circ \mathrm{Tr}^{\mathcal{T}_1}_{\mathrm{F}}) \subseteq (S^0_2 \circ \mathrm{Tr}^{\mathcal{T}_2}_{\mathrm{F}})$; resp. failure trace inclusion: $(S^0_1 \circ \mathrm{Tr}^{\mathcal{T}_1}_{\mathrm{FT}}) \subseteq (S^0_2 \circ \mathrm{Tr}^{\mathcal{T}_2}_{\mathrm{FT}})$; resp. ready simulation: $\mathcal{T}_1$ is *ready simulated by* $\mathcal{T}_2$ if there exists a simulation R

[6] For space reasons, compact definitions, conforming with the standard ones, are used.

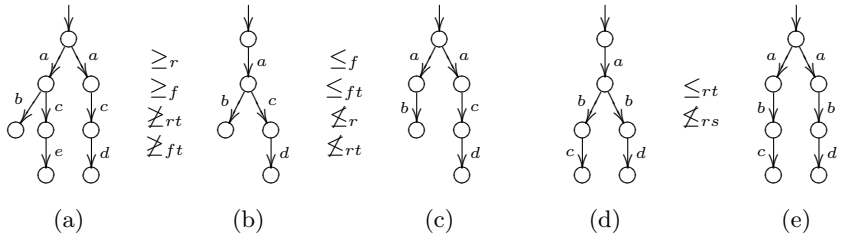


Fig. 2. Illustration of the refinement preorders on TSs, where $\leq_x$ stands for refinement with respect to refinement notion x. These examples are derived from Counterexamples 5, 6, and 8 of [17].

between $\mathcal{T}_1$ and $\mathcal{T}_2$ (i.e., $\mathcal{T}_1$ is simulated by $\mathcal{T}_2$) such that the enabled actions are the same for related elements, i.e., $(s_1, s_2) \in R \Rightarrow \mathbf{O}_{\rightarrow_1}(s_1) = \mathbf{O}_{\rightarrow_2}(s_2)$; resp. possible worlds inclusion: $\mathbb{T}_{\mathrm{rs}}(\mathcal{T}_1) \subseteq \mathbb{T}_{\mathrm{rs}}(\mathcal{T}_2)$. Fig. 2 illustrates some differences between the refinement notions. In all these cases, and also in the forthcoming ones when we will consider other more sophisticated classes of transition systems, a system is finite if and only if its set of states is finite.

Synchronously-communicating transition systems. Synchronously-communicating TSs [12] extend TSs by a predicate $e(s)$ on states s that indicates which actions must be present (i.e., enabled) at s and thus cannot be removed by refinements. Formally, a *synchronously-communicating transition system* (STS) without fairness is a tuple $(\mathcal{T}, e)$ such that $\mathcal{T}$ is a TS and $e : S \to \mathcal{P}(\mathcal{L})$ is its *existence predicate*. It is *must-saturated* if $a \in e(s)$ implies the existence of an outgoing transition labeled by a, i.e., $\forall s \in S : e(s) \subseteq \mathbf{O}_{\rightarrow}(s)$. For the definition of the refinement settings $\mathbb{S}$ of *STSs* and $\mathbb{S}_{\mathrm{ms}}$ of *must-saturated STSs*, DetTSs are embedded by taking e to be $\mathbf{O}_{\rightarrow}$, and $(\mathcal{T}_1, e_1)$ refines $(\mathcal{T}_2, e_2)$ if there exists a simulation R between $\mathcal{T}_1$ and $\mathcal{T}_2$ such that $(s_1, s_2) \in R \Rightarrow e_2(s_2) \subseteq e_1(s_1)$. For example, $\rightarrow\!\bigcirc \xrightarrow{a} \bigcirc \xrightarrow{a} \bigcirc$ is an implementation of the STS on the left of Fig. 3, whereas $\rightarrow\!\bigcirc \xrightarrow{a} \bigcirc \xrightarrow{b} \bigcirc$ is not.

Modal/mixed transition systems. Mixed TSs [7] have *must-transitions* (that must be present in an implementation) and *may-transitions* (nothing else may be present in an implementation). A modal TS [25] has the additional requirement that every must-transition also has to be a may-transition. Formally, a *mixed transition system* is a tuple $(\mathcal{T}, \hookrightarrow)$ such that $\mathcal{T}$ is a TS, where its transition relation is called *may*-transition relation here, and $\hookrightarrow \subseteq S \times \mathcal{L} \times S$ is its *must*-transition relation. It is a *modal transition system* if $\hookrightarrow \subseteq \rightarrow$. For the definition of the refinement settings $\mathbb{M}$ of *mixed TSs* and $\mathbb{M}_{\mathrm{mod}}$ of *modal TSs*, DetTSs are embedded by taking $\rightarrow$ as the must-transition relation, and $(\mathcal{T}_1, \hookrightarrow_1)$ refines $(\mathcal{T}_2, \hookrightarrow_2)$ if there exists a simulation R between $\mathcal{T}_1$ and $\mathcal{T}_2$ such that R^{-1} is a simulation between $(S_2, \emptyset, \hookrightarrow_2)$ and $(S_1, \emptyset, \hookrightarrow_1)$.

Modal/mixed TSs, as well as their disjunctive variants presented in the following, were originally designed for general transition systems as implementa-

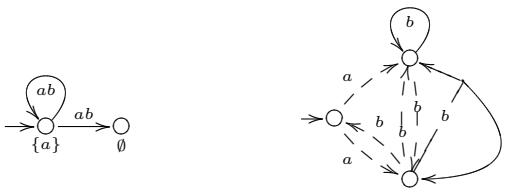


Fig. 3. An example STS (left) and disjunctive mixed TS (right).

tions. Interpreting them with respect to DetTS leads to new kinds of modeling techniques, which improve the succinctness of these settings, allowing for more compact representations, as, e.g., discussed in [14]. For example, two a-labeled must-transitions from the same state leading to states s_1 and s_2 require any implementation to implement, after the only possible a-step, both the behavior of s_1 *and* s_2. We call such behavior *conjunctive behavior*.

Disjunctive modal/mixed transition systems. Disjunctive modal/mixed TSs [26] generalize modal/mixed TSs by introducing hypertransitions that point to sets of states rather than single states. A *must-hypertransition* t indicates that the implementation must have a transition with corresponding label to a state that is related to at least one element in the target set of t, i.e., the targets are interpreted disjunctively. We present disjunctive modal TSs as a special case of their mixed version, where must-hypertransitions need not necessarily occur as may-transitions: a *disjunctive mixed transition system* is a tuple $(\mathcal{T}, \mapsto)$ such that $\mathcal{T}$ is a TS, where its transition relation is called *may*-transition relation here and $\mapsto \subseteq S \times \mathcal{L} \times \mathcal{P}(S)$ is its *must*-hypertransition relation. It is a *disjunctive modal transition system* if all must-hypertransition target sets are non-empty and only have elements that are also targets of may-transitions, i.e., $\forall s \in S, a \in \mathcal{L}, \ddot{S} \in \{s\}\circ \overset{a}{\mapsto}\colon \ddot{S} \neq \emptyset \wedge \ddot{S} \subseteq \{s\}\circ \overset{a}{\rightarrow}$. For the definition of the refinement settings $\mathbb{D}$ of *disjunctive mixed TSs* and $\mathbb{D}_{\text{mod}}$ of *disjunctive modal TSs*, DetTSs are embedded by taking $\{(s, a, \{s'\}) \mid s \overset{a}{\rightarrow} s'\}$ as must-hypertransitions, and $(\mathcal{T}_1, \mapsto_1)$ refines $(\mathcal{T}_2, \mapsto_2)$ if there is a simulation R between $\mathcal{T}_1$ and $\mathcal{T}_2$ such that $\forall (s_1, s_2) \in R$, $a \in \mathcal{L}, \ddot{S}_2 \in \{s_2\}\circ \overset{a}{\mapsto}_2\colon \exists \ddot{S}_1 \in \{s_1\}\circ \overset{a}{\mapsto}_1\colon \forall s_1' \in \ddot{S}_1 : \exists s_2' \in \ddot{S}_2 : s_1' R s_2'$. For example, $\rightarrow\circ\xrightarrow{a}\circlearrowright_b$ is an implementation of the disjunctive mixed TS on the right of Fig. 3, whereas $\rightarrow\circ\xrightarrow{a}\circ\xrightarrow{b}\circ$ is not.

4 Comparison

This section establishes an expressiveness hierarchy constructively by presenting language-preserving transformations or showing their non-existence by counter-example. In particular, we have paid attention to *simple* transformations and *small-sized* transformed models. All transformations also work for infinite-state systems but, not surprisingly, their mappings are not guaranteed to be finite.

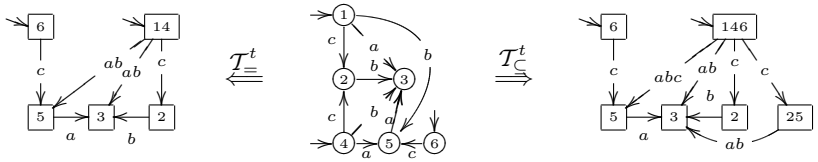
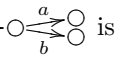


Fig. 4. Illustration of the transformations' images $\mathcal{T}^t_{=}$ and $\mathcal{T}^t_{\subseteq}$. Targets of transitions without source indicate initial states. Transitions having a set as label indicate a set of transitions, one for each label. The numbers of the state names in the left and right systems correspond to the state subset encoding.

To begin with, the identity function is a transformation from $\mathbb{S}_{\text{ms}}$ into $\mathbb{S}$; from $\mathbb{M}_{\text{mod}}$ into $\mathbb{M}$; from $\mathbb{D}_{\text{mod}}$ into $\mathbb{D}$; from $\mathbb{T}_{\text{pw}}$ into $\mathbb{T}_{\text{rs}}$; and from $\mathbb{T}_{\text{rs}}$ into $\mathbb{T}_{\text{pw}}$.

Trace inclusions. Due to the coinductive definition of simulation, checking refinement in simulation-based settings only depends on what remains to be considered in the future, e.g.,  is not an implementation of $\mathcal{T}_{\text{rs,rt}}$ of Fig. 6 in simulation-like approaches, because the refinement relation has to decide for one of the two initial states. This is different in trace-like approaches, where at any time it is possible to go back in a trace and resolve nondeterminism differently, as long as the traces still coincide. Consequently  *is* a refinement of $\mathcal{T}_{\text{rs,rt}}$ in trace-like settings. A transformation from a trace-like setting to $\mathbb{T}_{\text{rs}}$ therefore has to make every previous nondeterministic choice explicit in the state space. Hence, power sets over states are used in the transformations from $\mathbb{T}_{\text{rt}}$, resp. from $\mathbb{T}_{\text{ft}}$, as illustrated in Fig. 4.

Transformation 1. *For any TS* $\mathcal{T}$, $\mathbb{T}_{\text{rt}}(\mathcal{T}) = \mathbb{T}_{\text{rs}}(\mathcal{T}^t_{=}) = \mathbb{T}_{\text{rt}}(\mathcal{T}^t_{=})$ *and* $\mathbb{T}_{\text{ft}}(\mathcal{T}) = \mathbb{T}_{\text{rs}}(\mathcal{T}^t_{\subseteq}) = \mathbb{T}_{\text{rt}}(\mathcal{T}^t_{\subseteq}) = \mathbb{T}_{\text{ft}}(\mathcal{T}^t_{\subseteq})$ *with*

$$\mathcal{T}^t_{\lhd} = (\mathcal{P}(S), \Psi^t_{\lhd}(S^0), \{(\ddot{S}, a, \ddot{S}') \mid a \in \mathcal{L} \wedge \ddot{S}' \in \Psi^t_{\lhd}(\ddot{S} \circ \xrightarrow{a})\}) \text{ , where}$$
$$\lhd \in \{=, \subseteq\} \quad \text{and} \quad \Psi^t_{\lhd}(\widehat{S}) = \{\{s \in \widehat{S} \mid \mathbf{O}_{\rightarrow}(s) \lhd L\} \mid L \subseteq \mathcal{L}\} \setminus \{\emptyset\}.$$

Transformations from pair to trace approaches are similar, except that all reachable states with respect to the underlying label trace are collected (since a failure/ready pair has as history information only the underlying label trace and no intermediate failure/ready sets). Hence, pairs of original states and allowed labels are the state set of these transformations, as illustrated in Fig. 5.

Transformation 2. *For any TS* $\mathcal{T}$, $\mathbb{T}_{\text{r}}(\mathcal{T}) = \mathbb{T}_{\text{rs}}(\mathcal{T}^p_{=}) = \mathbb{T}_{\text{rt}}(\mathcal{T}^p_{=}) = \mathbb{T}_{\text{r}}(\mathcal{T}^p_{=})$ *and* $\mathbb{T}_{\text{f}}(\mathcal{T}) = \mathbb{T}_{\text{rs}}(\mathcal{T}^p_{\subseteq}) = \mathbb{T}_{\text{rt}}(\mathcal{T}^p_{\subseteq}) = \mathbb{T}_{\text{ft}}(\mathcal{T}^p_{\subseteq}) = \mathbb{T}_{\text{r}}(\mathcal{T}^p_{\subseteq}) = \mathbb{T}_{\text{f}}(\mathcal{T}^p_{\subseteq})$ *with*

$$\mathcal{T}^p_{\lhd} = (\mathcal{P}(S) \times \mathcal{P}(\mathcal{L}), \Psi^p_{\lhd}(S^0), \{((\ddot{S}, L), a, Z') \mid a \in L \wedge Z' \in \Psi^p_{\lhd}(\ddot{S} \circ \xrightarrow{a})\}),$$
$$\text{where} \quad \lhd \in \{=, \subseteq\} \quad \text{and} \quad \Psi^p_{\lhd}(\widehat{S}) = \{(\widehat{S}, \widehat{L}) \mid \exists s \in \widehat{S} : \mathbf{O}_{\rightarrow}(s) \lhd \widehat{L}\}.$$

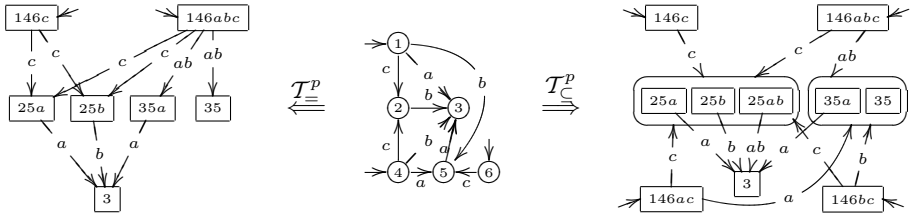


Fig. 5. Illustration of the transformations' images $\mathcal{T}^p_{=}$ and $\mathcal{T}^p_{\subseteq}$. In the right picture, states that have the same targets are identified. A transition pointing to an oval indicates a set of transitions pointing to each element inside the oval. The numbers (resp. labels) of the state names in the left and right pictures correspond to the state (resp. label) subset encoding in the respective transformation.

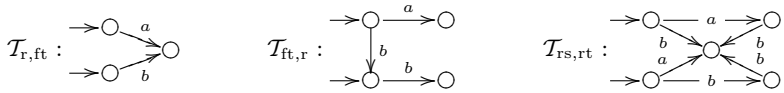
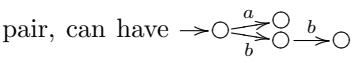
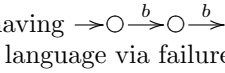
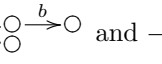
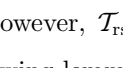


Fig. 6. TSs illustrating increases of expressiveness.

The increase of expressiveness of these settings is illustrated in Fig. 10: the failure approach *cannot* express an exclusive alternative between two labels. However, the ready approach can do it, as shown by $\mathcal{T}_{\mathrm{r,ft}}$ in Fig. 6. This is reflected by the axiom $c.P_a + c.P_b \equiv c.P_a + c.P_b + c(P_a + P_b)$, which is valid for failure semantics, but not for the semantics based on ready sets. In pair approaches, behavior can only be described up to alternatives having the same label path histories, whereas a trace approach can also distinguish alternatives that have the same label path history but different next-step possibilities (up to failure or ready interpretation). For example, no TS with respect to failure pair, resp. ready pair, can have  and as implementations, without also having  as implementation. However, $\mathcal{T}_{\mathrm{ft,r}}$ of Fig. 6 defines such a language via failure trace (resp. ready trace). Ready simulation increases the expressiveness even more by distinguishing also alternatives with the same label path history and next-step possibilities, but different future behaviors in the past. For example, no TS with respect to a trace approach can have  and as implementations, without also having  . However, $\mathcal{T}_{\mathrm{rs,rt}}$ of Fig. 6 defines such a language via ready simulation. The following lemma summarizes the above results, from which the 'strictly greater' expressiveness results for the TS-based settings are derived by transitivity arguments.

Lemma 1. *For $\mathcal{T}_{\mathrm{r,ft}}$ $\mathcal{T}_{\mathrm{ft,r}}$, $\mathcal{T}_{\mathrm{rs,rt}}$ in Fig. 6 and arbitrary $\mathcal{T}$, we have:*

$$\mathbb{T}_{\mathrm{r}}(\mathcal{T}_{\mathrm{r,ft}}) \neq \mathbb{T}_{\mathrm{ft}}(\mathcal{T}),\ \mathbb{T}_{\mathrm{ft}}(\mathcal{T}_{\mathrm{ft,r}}) \neq \mathbb{T}_{\mathrm{r}}(\mathcal{T})\ \textit{and}\ \mathbb{T}_{\mathrm{rs}}(\mathcal{T}_{\mathrm{rs,rt}}) \neq \mathbb{T}_{\mathrm{rt}}(\mathcal{T}).$$

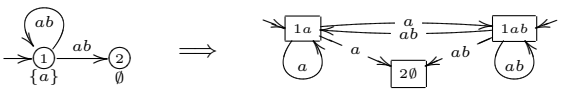


Fig. 7. Example of the transformation from $\mathbb{S}_{\text{ms}}$ to $\mathbb{T}_{\text{rs}}$. For STSs, the image of e is depicted close to the state. The numbers (resp. labels) of the state names in the right picture correspond to the state (resp. label) subset encoding of the transformation.

The following proposition shows how our transformations and the efficient decision procedures for simulation-like preorders [6] can be used to decide the corresponding inclusion problems. However, in general such derived algorithms would have limited practical relevance since deciding trace-like preorders is PSPACE-complete [31], but in some particular cases the complexity of the decision procedure is certainly much lower.

Proposition 1. *$\mathcal{T}$ is ready trace (failure trace, ready pair, failure pair) included in $\widetilde{\mathcal{T}}$ iff $\mathcal{T}^t_=$ (resp. $\mathcal{T}^t_\subseteq$, $\mathcal{T}^p_=$, $\mathcal{T}^p_\subseteq$) is ready simulated by $\widetilde{\mathcal{T}}^t_=$ (resp. $\widetilde{\mathcal{T}}^t_\subseteq$, $\widetilde{\mathcal{T}}^p_=$, $\widetilde{\mathcal{T}}^p_\subseteq$).*

Ready simulation and must-saturated STS. $\mathbb{T}_{\text{rs}}$ is transformed to $\mathbb{S}_{\text{ms}}$ by setting the existence predicate to the set of labels for which an outgoing transition exists.

Transformation 3. *For any TS $\mathcal{T}$, $\mathbb{T}_{\text{rs}}(\mathcal{T}) = \mathbb{S}_{\text{ms}}((\mathcal{T}, \mathbf{O}_\to))$.*

For transforming $\mathbb{S}_{\text{ms}}$ to $\mathbb{T}_{\text{rs}}$, every state s is combined with a ready set $L \subseteq \mathcal{L}$, indicating that exactly these labels may *not* be removed. The incoming transitions are determined by the incoming ones of s. Fig. 7 presents a simple example.

Transformation 4. *For any must-saturated STS $(\mathcal{T}, e)$, $\mathbb{S}_{\text{ms}}((\mathcal{T}, e)) = \mathbb{T}_{\text{rs}}((S',$ $S' \cap (S^0 \times \mathcal{P}(\mathcal{L})), \to'))$ with*

$$S' = \{(s, L) \mid e(s) \subseteq L \subseteq \mathbf{O}_\to(s)\}, \quad \to' = \{((s, L), a, (s', L')) \mid s \xrightarrow{a} s' \wedge a \in L\}.$$

$\mathbb{S}$ is indeed strictly more expressive than $\mathbb{T}_{\text{rs}}$, because the latter does not allow one to specify the empty language.

Lemma 2. *For the STS $\to\!\underset{\{a\}}{\bigcirc}$ and arbitrary $\mathcal{T}$, we have $\mathbb{S}(\to\!\underset{\{a\}}{\bigcirc}) \neq \mathbb{T}_{\text{rs}}(\mathcal{T})$.*

If we allow the initial set of a TS to be empty and therefore not to have any DetTS as refinement, we obtain, by the following algorithm, that $\mathbb{S}$ and $\mathbb{S}_{\text{ms}}$ are equally expressive. Hence, the empty set is the only language that increases the expressive power of $\mathbb{S}$ and any other equally expressive refinement setting.

Proposition 2. *An STS can be linearly transformed to an equivalent must-saturated one (possibly with an empty initial state set) by successively removing those states s and their in- and outgoing transitions, for which $e(s) \not\subseteq \mathbf{O}_\to(s)$.*

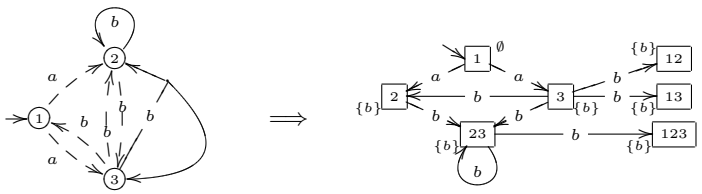


Fig. 8. Example of the transformation from $\mathbb{D}$ to $\mathbb{S}$. For disjunctive mixed TSs, solid (dashed) arrows model must-transitions (resp. may-transitions). Branching solid arrows model must-hypertransitions. The numbers of the state names in the right picture correspond to the state subset encoding; e.g., the self loop of state $\{2,3\}$ is obtained by choosing g and h such that $g(2) = 3$, $g(3) = 2$, $h(\{2\}) = 2$, and $h(\{2,3\}) = 2$.

Remaining settings. $\mathbb{S}$ is transformed to $\mathbb{M}$ by modeling predicate e via must-transitions to a special state s_{all} that is refined by each implementation state.

Transformation 5. *For any STS* $(\mathcal{T}, e)$, $\mathbb{S}((\mathcal{T}, e)) = \mathbb{M}(((S \cup \{s_{\text{all}}\}, S^0, \to \cup (\{s_{\text{all}}\} \times \mathcal{L} \times \{s_{\text{all}}\})), \hookrightarrow'))$ *with* $s_{\text{all}} \notin S$ *and* $\hookrightarrow' = \bigcup_{s \in S} \{s\} \times e(s) \times \{s_{\text{all}}\}$.

$\mathbb{M}$ (resp. $\mathbb{M}_{\text{mod}}$) is transformed to $\mathbb{D}$ (resp. $\mathbb{D}_{\text{mod}}$) by turning each must-transition pointing to s into a must-hypertransition pointing to $\{s\}$.

Transformation 6. *Let* $\mapsto' = \{(s, a, \{s'\}) \mid s \xrightarrow{a} s'\}$. *Then for any mixed TS* $(\mathcal{T}, \hookrightarrow)$, $\mathbb{M}((\mathcal{T}, \hookrightarrow)) = \mathbb{D}((\mathcal{T}, \mapsto'))$, *and for any modal transition system* $(\mathcal{T}, \hookrightarrow)$, $\mathbb{M}_{\text{mod}}((\mathcal{T}, \hookrightarrow)) = \mathbb{D}_{\text{mod}}((\mathcal{T}, \mapsto'))$.

We proceed with the transformation from $\mathbb{D}$ to $\mathbb{S}$. The new states are subsets $\ddot{S} \subseteq S$, with the intuition that a related implementation state has to be related to all elements of $\ddot{S}$. Transitions from $\ddot{S}$ lead to those subset states that consist of a combination of targets of must-hypertransitions from states $s \in \ddot{S}$, together with one may-target for each $s \in \ddot{S}$. In the definition of these successor sets $C^a_{\ddot{S}}$, we use choice functions $h : \mathcal{P}(S) \to S$ for the selection of an element from a must-hypertransition target, and $g : S \to S$ for the selection of a may-transition target. The existence predicate holds for a at $\ddot{S}$ iff there is a must-hypertransition with label a and leaving a state in $\ddot{S}$. Fig. 8 shows an example of this transformation.

Transformation 7. *For any disjunctive mixed TS* $(\mathcal{T}, \mapsto)$, $\mathbb{D}((\mathcal{T}, \mapsto)) = \mathbb{S}(((\mathcal{P}(S), \{\{s^0\} \mid s^0 \in S^0\}, \bigcup_{\ddot{S} \subseteq S, a \in \mathcal{L}} \{\ddot{S}\} \times \{a\} \times C^a_{\ddot{S}}), \mathbf{O}_{\mapsto}))$ *with* $C^a_{\ddot{S}} = \{g(\ddot{S}) \cup h(\ddot{S} \circ \xrightarrow{a}) \mid \forall s \in \ddot{S} : s \xrightarrow{a} g(s) \wedge \forall \ddot{S}' \in (\ddot{S} \circ \xrightarrow{a}) : h(\ddot{S}') \in \ddot{S}'\}$.

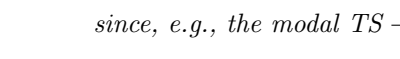
Finally, we present the transformation from $\mathbb{D}$ to $\mathbb{M}_{\text{mod}}$ with complexity $\mathcal{O}((2^{|S|})^{|\mathcal{L}|})$ and, by restriction to $\mathbb{M}$, obtain a transformation from $\mathbb{M}$ into $\mathbb{M}_{\text{mod}}$ with complexity $\mathcal{O}(|S|^{|\mathcal{L}|})$.

A first observation is that $\mathbb{M}_{\text{mod}}$ can represent conjunctive behavior since the properties described by all must-transition targets have to hold after the step. However, a state in $\mathbb{M}_{\text{mod}}$ cannot enforce the existence of a label a and,

at the same time, model disjunctive behavior after the execution of a (via nondeterminism) because, as soon as an outgoing must-transition (with implicit may-transition) exists, all further outgoing may-transitions with the same label are redundant: in deterministic refinements, the unique transition of the implementation already has to match with the may-transition corresponding to the must-transition. The solution is to distribute the needed requirements to multiple states, where one state $(s, -)$ enforces action existence, and several further states $(s, \ddot{S})$, with $\ddot{S} \subseteq S$, encode the nondeterministic behavior. Must-transitions to all these states make sure that each of them is related to a single implementation state, which therefore has to meet all of the requirements. These must-transitions originate from another kind of state $(s, f) \in S \times \mathrm{F}_s$, which encodes a complete resolution of the next-step nondeterminism in the $\mathbb{D}$-system. Here, F_s is a set of functions that collect, per label a, a set from $C^a_{\{s\}}$, i.e., an element from every must-hypertransition target together with one may-transition target. The resulting collection contains those $\mathbb{D}$-states to which the successor of a related implementation state must be related. To be precise, no element can be collected if no must-transition is present ($a \notin \mathbf{O}_{\mapsto}(s) \wedge g(a) = \emptyset$). For contradictory states, F_s is empty. A state (s, f) points, via a-labeled must-transitions, to every element of $g(a) \times (\{-\} \cup \mathcal{P}(S))$. A state $(s, -)$ encodes the labels necessary in s via must-transitions to state s_{all}, which is refined by any implementation state. A state $(s, \ddot{S})$ is used to model the nondeterministic behavior of (i) the must-hypertransition target $\ddot{S}$, or (ii) the may-transitions if $\ddot{S} = \{s\}\circ \xrightarrow{a}$. This is achieved by outgoing may-transitions to every element s' of $\ddot{S}$, combined with any value of $\mathrm{F}_{s'}$. For technical reasons, $(s, \ddot{S})$ points to s_{all} if $\ddot{S}$ does not correspond to a must-hypertransition target or to the may-transition targets. Fig. 9 shows an example of this transformation.

Transformation 8. *For any disjunctive mixed TS* $(\mathcal{T}, \mapsto)$, $\mathbb{D}((\mathcal{T}, \mapsto)) = \mathbb{M}_{\mathrm{mod}}(((S', S^{0\prime}, \rightarrow'), \hookrightarrow'))$ *with* $C^a_{\{s\}}$ *as in Transf. 7 and*

$$
\begin{aligned}
\mathrm{F}_s &= \{f : \mathcal{L} \to \mathcal{P}(S) \mid \forall a \in \mathcal{L} : f(a) \in C^a_{\{s\}} \vee (a \notin \mathbf{O}_{\mapsto}(s) \wedge f(a) = \emptyset)\} \\
S' &= \{s_{\mathrm{all}}\} \cup \{(s, x) \mid s \in S \wedge x \in \mathrm{F}_s \cup \{-\} \cup \mathcal{P}(S)\} \\
S^{0\prime} &= \{(s, x) \mid s \in S^0 \wedge x \in \mathrm{F}_s\} \qquad\qquad W^a_s = (\{s\}\circ \overset{a}{\mapsto}) \cup \{\{s\}\circ \xrightarrow{a}\} \\
\rightarrow' &= \hookrightarrow' \cup (\{s_{\mathrm{all}}\} \times \mathcal{L} \times \{s_{\mathrm{all}}\}) \cup \{((s, x), a, s_{\mathrm{all}}) \mid x \in \{-\} \cup \mathcal{P}(S) \setminus W^a_s\} \cup \\
&\quad \{((s, \ddot{S}), a, (s', f')) \mid s' \in \ddot{S} \wedge \ddot{S} \in W^a_s \wedge f' \in \mathrm{F}_{s'}\} \\
\hookrightarrow' &= \{((s, f), a, (s', x')) \mid f \in \mathrm{F}_s \wedge s' \in f(a) \wedge x' \in \{-\} \cup \mathcal{P}(S)\} \cup \\
&\quad \{((s, -), a, s_{\mathrm{all}}) \mid a \in \mathbf{O}_{\mapsto}(s)\}
\end{aligned}
$$

Here, we allow the initial state set to be empty; this does not affect expressiveness, since, e.g., the modal TS  *also describes the empty language.*

This concludes our presentation of transformations since all remaining transformations can be obtained by composition, yielding quite competitive (efficient) equivalent models.

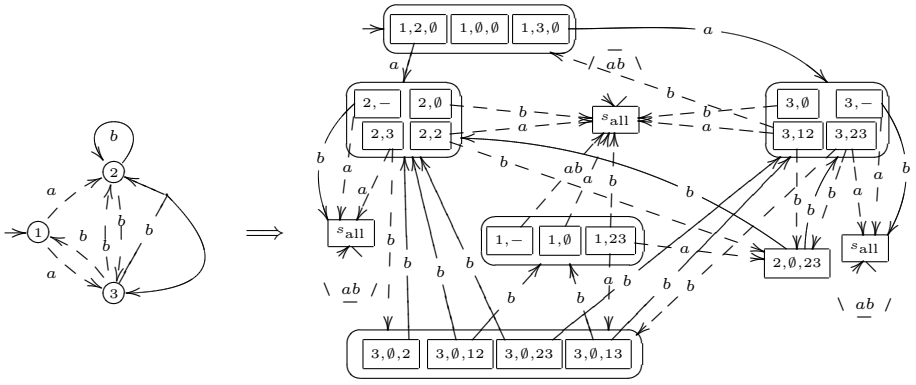


Fig. 9. Example of the transformation from $\mathbb{D}$ to $\mathbb{M}_{\text{mod}}$. On the right, the may-transitions that are implied by must-transitions are omitted, and the symbols of the state names correspond to the encoding of the transformation: (i) pairs with second element "−" correspond to the states that encode the existent labels; (ii) the remaining pairs, which have a subset as second element, encode the may-transition targets and the must-hypertransitions (states $(s, \ddot{S})$ are omitted if $\ddot{S} \notin W_s^a \cup W_s^b$, since they do not influence refinement); (iii) a triple corresponds to states that have a complete resolution where the second (resp. third) component encodes the image of a (resp. b). To improve readability, several copies of state s_{all} are used.

Consistency checking and expressiveness hierarchy. As a corollary, our transformations also yield a technique for checking consistency, i.e., whether the language of a model is non-empty. This is trivial for trace-like settings, ready simulation settings and must-saturated STSs, because these settings cannot describe the empty language.

Corollary 1. *For a disjunctive mixed (resp. disjunctive modal, mixed, modal, synchronously-communicating) TS, consistency can be checked by transforming it via our transformations into an STS, applying the algorithm given in Prop. 2, and finally checking if the initial state set is non-empty.*

Corollary 2. *Our transformations yield the expressiveness hierarchy of refinement settings depicted in Fig. 10.*

5 Related work

For trace and tree languages, many transformations have been developed between automata having different fairness constraints (Büchi, Muller, Rabin, Streett, parity), see, e.g., [19] and the references therein. Transformations between non-automata settings are given in [5] and [24], where the must-testing and ready simulation ($\frac{2}{3}$-bisimulation) preorders, respectively, are transformed to prebisimulation. In [16], forward/backward simulation and trace inclusion are

Considered refinement settings:

$\mathbb{D}$: Disjunctive mixed transition systems
$\mathbb{D}_{\text{mod}}$: Disjunctive modal transition systems
$\mathbb{M}$: Mixed transition systems
$\mathbb{M}_{\text{mod}}$: Modal transition systems
$\mathbb{S}$: Synchronously-communicating transition systems
$\mathbb{S}_{\text{ms}}$: Must-saturated s.-c. transition systems
$\mathbb{T}_{\text{pw}}$: Possible worlds inclusion
$\mathbb{T}_{\text{rs}}$: Ready simulation
$\mathbb{T}_{\text{rt}}$: Ready trace inclusion
$\mathbb{T}_{\text{r}}$: Ready pair inclusion
$\mathbb{T}_{\text{ft}}$: Failure trace inclusion
$\mathbb{T}_{\text{f}}$: Failure pair inclusion

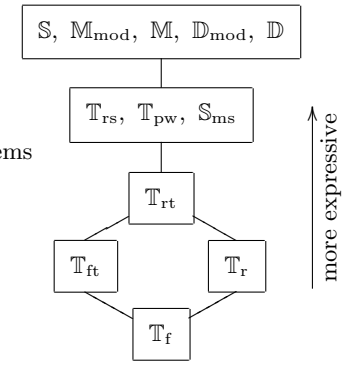


Fig. 10. Refinement settings for DetTSs ordered with respect to their expressiveness.

transformed to disjunctive modal TSs (*underspecified* TSs). These transformations implicitly demonstrate that the transformed settings are less or equally expressive with respect to the describable sets of (not necessarily deterministic) TSs. Transformations preserving the complete preorder (and not only the languages) are given in [15] and [18], where [15] proves that disjunctive modal TSs can be transformed into 1-selecting modal TSs but not vice versa, whereas [18] presents transformations between modal TS variants with transition labels and predicates on states.

An alternative approach to the examination of expressiveness is taken, e.g., in [11, 17], where preorders are compared regarding their coarseness. This comparison approach obviously does not lead to applicable transformations between settings.The obtained hierarchy – known as the linear-time branching-time spectrum – coincides with ours for many TS-based settings, but not in general, as is illustrated by possible worlds semantics and ready simulation semantics: by definition of possible-worlds semantics, they trivially have the same expressiveness in our language-based sense although the possible worlds preorder is finer than the ready simulation preorder. For the coinciding settings, our results cannot be immediately derived from the corresponding results in [17]. Consider, e.g., the increase of expressiveness between ready trace inclusion and ready simulation. It cannot be derived from Counterexample 8 of [17], illustrated here in Figs. 2(d) and (e), since both systems have exactly the same sets of implementations, both with respect to ready trace inclusion and ready simulation.

Yet another approach to studying the expressiveness of refinement settings is via modal logics in the style of Hennessy-Milner [20]. While much work focuses on characterizing preorders on general TSs, [3] shows a correspondence between the preorder underlying modal TSs and the prime and consistent formulas of Hennessy-Milner logic.

The problem of consistency checking is considered, e.g., in [21]. The authors present an algorithm, along with a complexity study, for checking the consistency of sets of modal TSs, i.e., for checking non-emptiness of the intersection of the

modal TSs' implementation sets in terms of general TSs. [23] gives algorithms and complexity results of consistency checks for several refinement notions.

Further refinement settings have been proposed in the literature, albeit for general TSs rather than DetTSs, e.g., in [22, 8, 9, 30, 13, 14]. We believe that all of them (when ignoring their possible fairness constraints) can be transformed into disjunctive mixed/modal TSs and vice versa while preserving their languages in terms of general TSs.

6 Conclusions

This paper studied the expressiveness of popular TS-based specification formalisms with respect to their describable languages in terms of deterministic TSs. Our results are summarized in the expressiveness hierarchy depicted in Fig. 10. Our work is of importance for system designers and verification-tool builders alike. The established expressiveness hierarchy aids system designers in selecting the right specification formalism for a problem in hand, while our transformations allow tool builders to reuse refinement checking algorithms across different formalisms. The results of this paper reveal that STSs combine expressiveness with succinctness and easy-to-comprehend models and thus seem to be a good choice for modeling sets of DetTSs.

Regarding future work, we wish to examine the succinctness of our refinement settings, show that our transformations lie in optimal complexity classes, and compare refinement settings based on preorders that abstract from internal computation, e.g., those mentioned in [23].

References

1. J. Baeten, J. Bergstra, and J. Klop. Ready-trace semantics for concrete process algebra with the priority operator. *Computer J.*, 30(6):498–506, 1987.
2. B. Bloom, S. Istrail, and A. Meyer. Bisimulation can't be traced. *J. ACM*, 42(1):232–268, 1995.
3. G. Boudol and K. G. Larsen. Graphical vs. logical specifications. *Theoretical Computer Science*, 106(1):3–20, 1992.
4. S. Brookes, C. Hoare, and A. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.
5. R. Cleaveland and M. Hennessy. Testing equivalence as a bisimulation equivalence. *Formal Aspects of Computing*, 5(1):1–20, 1993.
6. R. Cleaveland and O. Sokolsky. Equivalence and preorder checking for finite-state systems. In *Handbook of Process Algebra*, pp. 391–424. North-Holland, 2001.
7. D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM TOPLAS*, 19(2):253–291, 1997.
8. D. Dams and K. S. Namjoshi. The existence of finite abstractions for branching time model checking. In *LICS*, pp. 335–344. IEEE, 2004.
9. D. Dams and K. S. Namjoshi. Automata as abstractions. In *VMCAI*, vol. 3385 of *LNCS*, pp. 216–232, 2005.
10. J. Engelfriet. Determinacy → (observation equivalence = trace equivalence). *Theoretical Computer Science*, 36:21–25, 1985.

11. R. Eshuis and M. M. Fokkinga. Comparing refinements for failure and bisimulation semantics. *Fundam. Inf.*, 52(4):297–321, 2002.
12. H. Fecher and I. Grabe. Finite abstract models for deterministic transition systems: Fair parallel composition and refinement-preserving logic. In *FSEN*, vol. 4767 of *LNCS*, pp. 1-16, 2007.
13. H. Fecher and M. Huth. Ranked predicate abstraction for branching time: Complete, incremental, and precise. In *ATVA*, vol. 4218 of *LNCS*, pp. 322–336, 2006.
14. H. Fecher, M. Huth, H. Schmidt, and J. Schönborn. Refinement sensitive formal semantics of state machines with persistent choice. In *AVoCS*, 2007. To appear in *ENTCS*.
15. H. Fecher and H. Schmidt. Comparing disjunctive modal transition systems with an one-selecting variant. *J. Logic and Algebraic Programming*, 77:20–39, 2008.
16. H. Fecher and M. Steffen. Characteristic μ-calculus formula for an underspecified transition system. In *EXPRESS'04*, vol. 128 of *ENTCS*, pp. 103–116, 2005.
17. R. v. Glabbeek. The linear time–branching time spectrum I. The semantics of concrete, sequential processes. In *Handbook of Process Algebra*, pp. 3–99. North-Holland, 2001.
18. P. Godefroid and R. Jagadeesan. On the expressiveness of 3-valued models. In *VMCAI*, vol. 2575 of *LNCS*, pp. 206–222, 2003.
19. E. Grädel, W. Thomas, and T. Wilke, eds. *Automata, Logics, and Infinite Games: A Guide to Current Research*, vol. 2500 of *LNCS*, 2002.
20. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985.
21. A. Hussain and M. Huth. Automata games for multiple-model checking. *ENTCS*, 155:401–421, 2006.
22. D. Janin and I. Walukiewicz. Automata for the modal mu-calculus and related results. In *MFCS*, vol. 969 of *LNCS*, pp. 552–562, 1995.
23. K. G. Larsen, U. Nyman, and A. Wasowski. On modal refinement and consistency. In *CONCUR*, vol. 4703 of *LNCS*, pp. 105–119, 2007.
24. K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.
25. K. G. Larsen and B. Thomsen. A modal process logic. In *LICS*, pp. 203–210. IEEE, 1988.
26. K. G. Larsen and L. Xinxin. Equation solving using modal transition systems. In *LICS*, pp. 108–117. IEEE, 1990.
27. E. Olderog and C. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23(1):9–66, 1986.
28. D. Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, vol. 104 of *LNCS*, pp. 167–183, 1981.
29. I. Phillips. Refusal testing. *Theoretical Computer Science*, 50(3):241–284, 1987.
30. S. Shoham and O. Grumberg. 3-valued abstraction: More precision at less cost. In *LICS*, pp. 399–410. IEEE, 2006.
31. S. Shukla, H. Hunt, D. Rosenkrantz, and R. Stearns. On the complexity of relational problems for finite state processes. In *ICALP*, vol. 1099 of *LNCS*, pp. 466–477, 1996.
32. S. Veglioni and R. de Nicola. Possible worlds for process algebras. In *CONCUR*, vol. 1466 of *LNCS*, pp. 179–193, 1998.