



RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN  
LEHRSTUHL FÜR INFORMATIK 2  
SOFTWARE MODELING AND VERIFICATION  
PROF. DR. IR. JOOST-PIETER KATOEN

# STUTTER SIMULATION MINIMIZATION

LISA LENA VON BÜTTNER

MATRIKELNUMMER 273838

1. Gutachter: Prof. Dr. Ir. Joost-Pieter Katoen
  2. Gutachter: Prof. Dr. Dr.h.c. Wolfgang Thomas
- Betreuer: Prof. Dr. Ir. Joost-Pieter Katoen



---

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen sind, sind als solche kenntlich gemacht.

Aachen, den 07.04.2011

.....  
(Lisa Lena von Büttner)



---

**Abstract** Implementation relations, such as bisimulation, simulation or stutter bisimulation, play a central role in terms of the verification of a finite model of a hard- or software system. They enable the computation of a *minimal quotient system*, which preserves desired or relevant properties (specified for example by  $CTL^*$ -formulae) of the original system, and furthermore, the comparison of (originally) huge systems in an efficient (regarding time and space complexity) way. In 2009, Francesco Ranzato and Francesco Tapparo published a paper [6] that is concerned with the so-called *stutter simulation relation*. This relation is a weaker variant of the simulation relation and allows the computation of a quotient that preserves a large fragment of the temporal logic  $LTL$  and hence, certain properties of the original system. The aspects dealt with in this paper form the basis of this bachelor thesis, where we recess but also leave several matters unconsidered and introduce new, independent concepts.

**Zusammenfassung** Implementationsrelationen, wie beispielsweise Bisimulation, Simulation oder stotter Bisimulation, spielen eine zentrale Rolle hinsichtlich der Verifikation eines endlichen Modells eines Hard- oder Softwaresystems. Sie ermöglichen die Berechnung eines *minimalen Quotientensystems*, welches gewünschte oder relevante Eigenschaften (spezifiziert durch bspw.  $CTL^*$ -Formeln) des Ursprungssystems erhält, sowie den Vergleich (ursprünglich) grosser Systeme auf effiziente (im Sinne von Zeit- und Platzaufwand) Art und Weise. Francesco Ranzato und Francesco Tapparo brachten 2009 eine Arbeit [6] heraus, die sich mit der sog. *stotter Simulationsrelation* auseinandersetzt. Diese stellt eine abgeschwächte Variante der Simulationsrelation dar und erlaubt die Berechnung eines Quotienten, der einen grossen Teil der temporalen Logik  $LTL$  und damit bestimmte Eigenschaften des ursprünglichen Systems beibehält. Die in dem genannten Werk dargestellten Aspekte stellen die Grundlage dieser Bachelorarbeit dar, in der wir einige Inhalte vertiefen, andere aber auch vorläufig außer Acht lassen und neue, eigene Erkenntnisse hinzufügen.



---

## **Acknowledgements**

Thank you, dad, mom and sister, for being a strong shoulder to lean on.

Thank you, Bernhard, for being who you are.

Thank you, Henrik, for your great support and ability to find the needle in a haystack.

Thank you, Prof. Dr. Ir. Katoen, for your boundless patience, support and mentoring.

Thank you, Prof. Dr. Dr.h.c. Wolfgang Thomas, for co-correcting this thesis.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Preliminaries</b>	<b>3</b>
2.1	State of the Art . . . . .	3
2.2	Linear Temporal Logic . . . . .	3
<b>3</b>	<b>Stutter Simulation</b>	<b>7</b>
3.1	Stutter Simulation Preorder . . . . .	7
3.2	Stutter Simulation Equivalence . . . . .	19
3.3	Quotient Transition System under $\cong$ and $\cong^{div}$ . . . . .	20
3.4	Stutter Simulation Equivalence vs. Stutter Bisimulation Equivalence . . . . .	24
<b>4</b>	<b>Stutter Simulation Quotienting</b>	<b>33</b>
4.1	Basics . . . . .	34
4.2	Computing the Stutter Simulation Preorder $\leq$ . . . . .	35
4.3	Computing the Stutter Simulation Preorder $\leq$ and Equivalence $\cong$ . . . . .	43
4.4	Example Computation of $\Pi_{Eq}$ and $\leq_{Pre}$ . . . . .	54
<b>5</b>	<b>Conclusion</b>	<b>63</b>
	<b>Bibliography</b>	<b>65</b>



# 1 Introduction

In some cases computing solutions for certain problems is impossible or takes far too long, maybe even years or centuries. In this work, we focus on computable model checking problems to automatically prove if a created system model in form of a kripke structure exhibits a certain type of behavior in several specified cases, e.g. an elevator software that should run infinitely long. Such models can be very huge, i.e. great state space, or even infinite, and thus require a lot of time and space to be (fully) examined. Hence, the job is to find a (efficient) way to reduce its size, such that no 'significant' information concerning its behavior gets lost.

(Stutter bi-) Simulation defines a binary relation on given kripke structures, i.e. graphs with nodes and edges and several properties. In this case, we are talking about transition systems that describe the behavior of a piece of hard- or software, formally  $TS = (S, Act, \rightarrow, I, AP, L)$ . The nodes are interpreted as the states  $s \in S$  of the modeled system and each of those states is labelled by several atomic propositions  $a \in AP$  that are either true or false in the specific state. The edges are called transitions, usually labelled by some action  $\tau \in Act$ , which intuitively describe what actions the system can execute in the current state. Hence, a transition system is a mathematical model that represents a certain system. The (bi-) simulation relation compares the states of one (or two) transition system with respect to their linear- or branching-time behavior. If a state  $s$  can mimic all behaviors of a state  $s'$  then  $s$  can *simulate*  $s'$ . If the reverse also holds then  $s$  and  $s'$  are *bisimulation equivalent*.

Simulation and bisimulation define 'strong' relations on state transition systems, which means that *all* possible behavior patterns from the depicted states onwards are compared. The stuttering variant requires less strict conditions to identifying two states as '*stutter similar*' or '*stutter bisimilar*', since it focusses on *outer* or *observable* behavior, i.e. changing an equivalence class of states, and ignoring transitions that do not perform *visible* steps, where the system stays in one and the same equivalence class.

Those binary relations on transition systems, called *implementation relations*, refer to the state labels  $\in AP$  and enable the possibility to 'correctly' *minimizing* transition systems. Since each of those systems models selected properties that are formally describable by temporal formulae, *correctly minimizing* requires some sort of language preservation, such that the (hopefully) smaller transition system is similar or even equivalent to the refined, original one. The preservation of temporal logics is therefore of great importance in model checking considerations. The language analyzed in this thesis is named *Linear Temporal Logic (LTL)*. It will turn out that the truth value of a large fragment of *LTL* formulae is preserved if the system is minimized according to the *stutter simulation equivalence*. Furthermore, the *stutter simulation preorder* will be defined. Algorithms to compute the *stutter simulation quotient* of a given finite transition system and to check whether one transition system is stutter similar, i.e. an abstraction, to another one will be presented and are based on the approaches of Ranzato and Tapparo [6]. However, the action labels of the transitions are ignored in this

## *1 Introduction*

---

work but, according to the authors, the provided definitions, theorems and algorithms can be transformed for such considerations.

## 2 Preliminaries

In this chapter an overview of the current approaches in theoretical affairs, such as minimizing and comparing finite transition systems over a set of atomic propositions in the sense of equivalence of states, is provided.

Afterwards, the basic mathematical tools and notations<sup>1</sup> are defined which will be used throughout the whole thesis and can be skipped if the reader is already familiar with the approaches in model checking by Baier and Katoen [1].

### 2.1 State of the Art

Today's approaches exhibit several algorithms to compute simulation and (stutter) bisimulation quotients, i.e. minimized systems, for finite transition systems in an efficient way [1,5]. Baier and Katoen [1] developed an algorithm to compute the stutter bisimulation quotient  $TS/\approx$  for a finite transition system  $TS = (S, Act, \rightarrow, I, AP, L)$  that runs in time  $\mathcal{O}(|S| \cdot (|AP| + M))$ , where  $M$  defines the number of edges in  $TS$  and assuming that  $M \geq |S|$ . This allows to algorithmically checking whether two transition systems are stutter bisimulation equivalent (stutter bisimilar in short), i.e if  $TS_1 \approx TS_2$ . Secondly, towards minimizing a transition system and preserving the next-free fragment of the languages CTL and CTL\*, namely  $CTL_{\setminus \circ}$  and  $CTL^*_{\setminus \circ}$ , a version of stutter bisimulation, called *divergence-sensitive stutter bisimulation*, is defined. The resulting quotient  $TS/\approx^{div}$  can be computed in time  $\mathcal{O}((|S| + M) + |S| \cdot (|AP| + M))$  (with the same assumptions as for quotienting without divergence sensitivity). The simulation preorder  $\leq$  is a non-symmetric implementation relation, which induces an equivalence relation, the so-called simulation equivalence  $\simeq$ . Following Baier and Katoen, the quotient transition system  $TS/\simeq$  can be obtained in time  $\mathcal{O}(M \cdot |S| + |S| \cdot |AP|)$ , where the universal fragment of  $CTL^*$ , formally  $\forall CTL^*$ , is preserved. However, the stuttering notion of simulation is rudimentary explored and we will provide the fundamental components

### 2.2 Linear Temporal Logic

In the literature there are several models defined to describe the behavior of a certain system. The particular one used throughout this thesis is called a *transition system*, which basically consists of *states* connected via *transitions*. States are used to model all relevant informations about the examined system at a certain *moment* of its behavior and are labeled by *atomic propositions* that enable to evaluate formulae of temporal logics. Transitions describe *how* the system can evolve from one state to another and are often equipped with *action names* to model communication between processes. However, in this thesis we will ignore the labeling of transitions.

---

<sup>1</sup>All definitions, notations and explanations are taken from or based on Baier and Katoen [1] (20, 23, 95ff, 231ff).

**DEFINITION 2.2.1 ( TRANSITION SYSTEM (TS) )**

A transition system  $TS$  is a tuple  $(S, Act, \rightarrow, I, AP, L)$  where

- $S$  is a set of states,
- $Act$  is a set of actions,
- $\rightarrow \subseteq S \times Act \times S$  is a transition relation,
- $I \subseteq S$  is a set of initial states,
- $AP$  is a set of atomic propositions, and
- $L : S \rightarrow 2^{AP}$  is a labeling function. ■

A transition system is *finite* if  $S$ ,  $Act$  and  $AP$  are finite.

The following definitions allow to refer to a particular state or to specify certain properties of a transition system.

**DEFINITION 2.2.2 ( DIRECT SUCCESSORS )**

Let  $TS$  be a transition system over  $AP$  and  $s$  a state in  $S$ . The set of *direct successors* of  $s$  is defined as

$$Post(s) = \{ t \in S \mid s \rightarrow t \}. \quad \blacksquare$$

A state  $t \in S$  is a direct successor of  $s$  if  $s$  has an outgoing transition that leads to  $t$ .

**DEFINITION 2.2.3 ( TERMINAL STATE )**

State  $s \in S$  is called *terminal* if and only if  $Post(s) = \emptyset$ . ■

A state is a terminal state if it has *no* direct successor, e.g. to represent the termination of the modeled system.

**DEFINITION 2.2.4 ( PATH FRAGMENT )**

Let  $TS$  be some transition system.

- A *finite* path fragment  $\hat{\pi}$  of  $TS$  is a finite sequence  $s_0s_1\dots s_n$  ( $n \geq 0$ ), such that  $s_i \in Post(s_{i-1})$  for all  $0 < i \leq n$ .
- An *inifinite* path fragment  $\pi$  is an infinite sequence  $s_0s_1s_2\dots$ , such that  $s_i \in Post(s_{i-1})$  for all  $i > 0$ . ■

**DEFINITION 2.2.5 ( MAXIMAL AND INITIAL PATH FRAGMENT )**

Let  $TS$  be some transition system.

- A *maximal* path fragment is either a finite path fragment that ends in a terminal state, or an infinite path fragment.
- An *initial* path fragment starts in an initial state, i.e.  $s_0 \in I$ .
- The set of maximal path fragments starting in state  $s$  is denoted by  $Paths(s)$ . ■

Note that if a transition system has no terminal states then all maximal path fragments are infinite.

**DEFINITION 2.2.6 ( PATH )**

A *path* of a transition system is an initial, maximal path fragment. ■

**DEFINITION 2.2.7 ( TRACE AND TRACE FRAGMENT )**

Let  $TS$  be some transition system without terminal states.

- The *trace* of the finite path fragment  $\hat{\pi} = s_0s_1\dots s_n$  ( $n \geq 0$ ) is defined as  $trace(\hat{\pi}) = L(s_0)L(s_1)\dots L(s_n)$ .
- The *trace* of the infinite path fragment  $\pi = s_0s_1s_2\dots$  is defined as  $trace(\pi) = L(s_0)L(s_1)L(s_2)\dots$
- The set of traces emanating from state  $s$  is denoted by  $Traces(s) = trace(Paths(s))$ .
- The set of traces of the initial states of  $TS$  is denoted by  $Traces(TS) = \bigcup_{s \in I} Traces(s)$ . ■

A trace of a (finite or infinite) path fragment  $s_0s_1s_2\dots$  is the (finite or infinite) sequence of atomic propositions that are valid in the states of the path, i.e. a word over the alphabet  $2^{AP}$ . We will now give a short definition of the syntax and semantics of  $LTL_{\setminus \circ}$  formulae and how they are interpreted over transition systems.

**DEFINITION 2.2.8 ( SYNTAX OF  $LTL_{\setminus \circ}$  )**

$LTL_{\setminus \circ}$  formulae over a set  $AP$  of atomic propositions are formed according to the following grammar:

$$\varphi ::= \text{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid \varphi_1 \cup \varphi_2$$

where  $a \in AP$ . ■

The until operator allows to derive the temporal modalities  $\diamond$  ("eventually", sometimes in the future) and  $\square$  ("always", from now on forever) as follows:

$$\diamond \varphi := \text{true} \cup \varphi \quad \square \varphi := \neg \diamond \neg \varphi$$

Each  $LTL_{\setminus \circ}$  formula  $\varphi$  stands for a certain property, a so-called *LT property*, of paths (or in fact their trace), such that a path can either fulfill  $\varphi$  or not. If a path satisfies formula  $\varphi$  then the corresponding trace exhibits the linear-time behavior specified by  $\varphi$ . We will first give the semantics of  $LTL_{\setminus \circ}$  formula  $\varphi$  as a language  $Words(\varphi) \subseteq 2^{AP}$ . Then, the semantics is extended to an interpretation over paths and states of a transition system.

**DEFINITION 2.2.9 ( SEMNATICS OF  $LTL_{\setminus \circ}$  (INTERPRETATION OVER WORDS) )**

Let  $\varphi$  be an  $LTL_{\setminus \circ}$  formula over  $AP$ . The *LT property* induced by  $\varphi$  is

$$Words(\varphi) = \{ \sigma \in (2^{AP})^\omega \mid \sigma \models \varphi \},$$

where the satisfaction relation  $\models \subseteq (2^{AP})^\omega \times LTL$  is the smallest relation with the properties in Figure 2.1. ■

$\sigma \models \text{true}$	
$\sigma \models a$	iff $a \in A_0$ (i.e., $A_0 \models a$ )
$\sigma \models \varphi_1 \wedge \varphi_2$	iff $\sigma \models \varphi_1$ and $\sigma \models \varphi_2$
$\sigma \models \neg\varphi$	iff $\sigma \not\models \varphi$
$\sigma \models \varphi_1 \cup \varphi_2$	iff $\exists j \geq 0. \sigma[j\dots] \models \varphi_2$ and $\sigma[i\dots] \models \varphi_1$ , for all $0 \leq i < j$

Figure 2.1: *LTL* semantics (satisfaction relation  $\models$ ) for infinite words over  $2^{AP}$

Here, for  $\sigma = A_0A_1A_2\dots \in (2^{AP})^\omega$ ,  $\sigma[j\dots] = A_jA_{j+1}A_{j+2}\dots$  is the suffix of  $\sigma$  starting in the  $(j+1)$ st symbol  $A_j$ .

**DEFINITION 2.2.10 ( SEMANTICS OF *LTL* OVER PATHS AND STATES )**

Let  $TS = (S, Act, \rightarrow, I, AP, L)$  be a transition system without terminal states, and let  $\varphi$  be an *LTL*-formula over  $AP$ .

- For infinite path fragment  $\pi$  of  $TS$ , the satisfaction relation is defined by

$$\pi \models \varphi \quad \text{iff} \quad \text{trace}(\pi) \models \varphi.$$

- For state  $s \in S$ , the satisfaction relation  $\models$  is defined by

$$s \models \varphi \quad \text{iff} \quad (\forall \pi \in \text{Paths}(s). \pi \models \varphi).$$

- $TS$  satisfies  $\varphi$ , denoted  $TS \models \varphi$ , if  $\text{Traces}(TS) \subseteq \text{Words}(\varphi)$  ■

Thus, transition system  $TS$  satisfies a certain formula  $\varphi \in LTL_{\setminus \bigcirc}$  if and only if  $s_0 \models \varphi$  for all initial states  $s_0$  of  $TS$ .

## 3 Stutter Simulation

In this chapter the *stutter simulation relation* is defined and we will prove that the *coarsest* stutter simulation is a *preorder*. Later we will see that by applying the notion of the stutter simulation preorder, it can easily be shown whether one transition system is a *correct abstraction*—in the sense of stutter simulation—of another one.

Furthermore, the definition of the *stutter simulation equivalence* is given, which is the *symmetric reduction* of the stutter simulation preorder and essential to *algorithmically reducing* the state space of a certain model. Afterwards, the resulting (hopefully smaller) *quotient system* will be defined and presented.

Last but not least, a short comparison of stutter simulation and stutter bisimulation is offered to the reader.

### 3.1 Stutter Simulation Preorder

We already learned that simulation defines strong relations on transition systems, whereas stutter simulation imposes less strict conditions on a state to being able to stutter simulate another one. Nonetheless, both versions are preorders that identify states of a given model  $TS = (S, Act, \rightarrow, I, AP, L)$  which—in some way—exhibit similar behavior. For simulation, a state  $s_1 \in S$  can be simulated by a state  $s_2 \in S$ , if they are (1) labeled with the same set of atomic propositions in  $AP$  and (2) if  $s_2$  can mirror every possible step of  $s_1$ , i.e. for each successor  $s'_1 \in Post(s_1)$  there must exist at least one transition  $s_2 \rightarrow s'_2$ , such that  $s'_2$  can simulate  $s'_1$  [1]. In this case,  $s_2$  can mimic *each* stepwise behavior of  $s_1$ —and maybe even more—and is therefore at least as *powerful* as  $s_1$ . However, the reverse is not guaranteed.

Stutter simulation requires basically the same conditions, though the second one is not as strict. Whenever  $s_1$  has a successor  $s'_1 \in Post(s_1)$  that is *not stutter simulated* by  $s_2$ , state  $s_2$  must be able to reach a state  $s'_2$  that stutter imitates  $s'_1$ , via a finite path fragment  $s_2 u_1 \dots u_n s'_2$ ,  $n \geq 0$ , where all states  $u_i$  stutter simulate  $s_1$ . Roughly speaking, this condition requires that every *visible* behavior of  $s_1$  (i.e.  $s'_1 \in Post(s_1)$  and  $s'_1$  is not stutter simulated by  $s_2$ ) is mimicked by  $s_2$ , possibly by performing some invisible steps  $u_1 \dots u_n$  in between.

This leads to the following formal definition of stutter simulation for a pair of states of a single transition system.

#### DEFINITION 3.1.1 ( STUTTER SIMULATION )

Let  $TS = (S, Act, \rightarrow, I, AP, L)$  be a transition system. A *stutter simulation* for  $TS$  is a binary relation  $\mathcal{R}$  on  $S$  such that for all  $(s_1, s_2) \in \mathcal{R}$ :

- (1)  $L(s_1) = L(s_2)$ .
- (2) If  $s'_1 \in Post(s_1)$  with  $(s'_1, s_2) \notin \mathcal{R}$ , then there exists a finite path fragment  $s_2 u_1 \dots u_n s'_2$ ,  $n \geq 0$ , where  $(s_1, u_i) \in \mathcal{R}$  for all  $i \in \{1, \dots, n\}$  and  $(s'_1, s'_2) \in \mathcal{R}$ .

### 3 Stutter Simulation

A state  $s_1$  can be *stutter simulated* by a state  $s_2$  (or, equivalently,  $s_2$  stutter simulates  $s_1$ ), denoted  $s_1 \preceq_{TS} s_2$ , if there exists a stutter simulation  $\mathcal{R}$  for  $TS$  with  $(s_1, s_2) \in \mathcal{R}$ . ■

Since a state  $s_2$  cannot be a candidate to stutter mimic the behavior of a state  $s_1$  if they do not have the same labeling, the first condition is natural. The second one requires the following: Assume  $s'_1$  to be a successor of  $s_1$ , which is not stutter simulated by  $s_2$ . For each such  $s'_1$ , state  $s_2$  must exhibit a finite path that leads to a state  $s'_2$  that stutter imitates  $s'_1$  and where all intermediate states stutter simulate  $s_1$ .

For state  $s$ , let  $StSim_{TS}(s)$  denote the *stutter-simulator set* of  $s$ , i.e. the set of states that stutter simulate it:

$$StSim_{TS}(s) = \{ s' \in S \mid s \preceq_{TS} s' \}.$$

Figure 3.1 illustrates this concept.

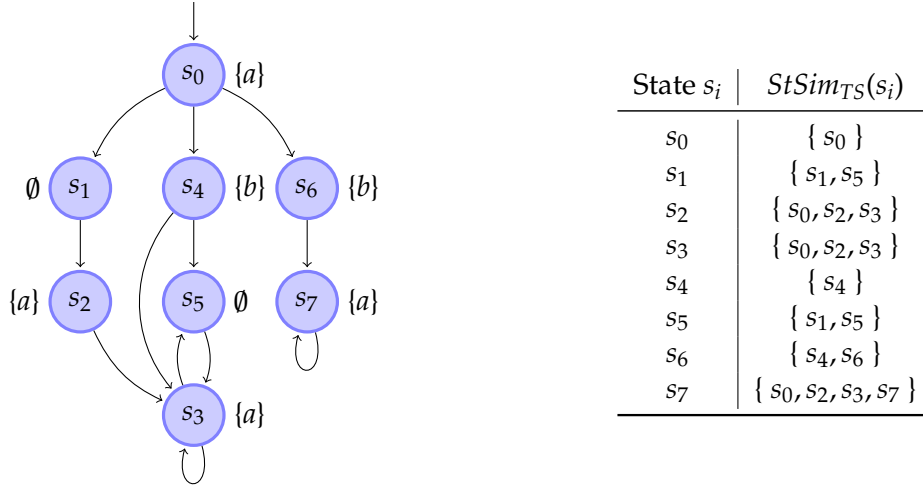


Figure 3.1: Stutter Simulator Sets

The notion of stutter simulation for a pair of states  $(s_1, s_2)$  of a single transition system can be adapted to one for pairs of transition systems  $(TS_1, TS_2)$  in order to check whether  $TS_2$  is a correct abstraction of  $TS_1$ :

If every initial state of  $TS_1$  can be stutter matched by an initial state of  $TS_2$ , then  $TS_2$  stutter simulates  $TS_1$  and hence  $TS_1$  can be represented by  $TS_2$  in a (hopefully) more abstract manner (smaller state space).

#### DEFINITION 3.1.2 ( STUTTER SIMULATING TRANSITION SYSTEMS )

Let  $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP, L_i)$ ,  $i = 1, 2$ , be two transition systems over the same set of atomic propositions  $AP$ . A *stutter simulation* for  $(TS_1, TS_2)$  is a binary relation  $\mathcal{R} \subseteq S_1 \times S_2$  such that

$$(A) \forall s_1 \in I_1. (\exists s_2 \in I_2. (s_1, s_2) \in \mathcal{R}).$$

(B) For all  $(s_1, s_2) \in \mathcal{R}$  the following conditions hold:

- (1)  $L(s_1) = L(s_2)$ .
- (2) If  $s'_1 \in \text{Post}(s_1)$  with  $(s'_1, s_2) \notin \mathcal{R}$ , then there exists a finite path fragment  $s_2 u_1 \dots u_n s'_2$ ,  $n \geq 0$ , where  $(s_1, u_i) \in \mathcal{R}$  for all  $i \in \{1, \dots, n\}$  and  $(s'_1, s'_2) \in \mathcal{R}$ .

Transition system  $TS_1$  can be *stutter simulated* by transition system  $TS_2$  (or, equivalently,  $TS_2$  stutter simulates  $TS_1$ ), denoted  $TS_1 \preceq TS_2$ , if there exists a stutter simulation  $\mathcal{R}$  for  $(TS_1, TS_2)$ . ■

$TS_2$  must be able to perform each stepwise behavior of  $TS_1$  in a stuttering manner. Since every possible path in a transition system starts in one of its initial states, it is reasonable that each  $s_1 \in I_1$  of the more detailed system model ( $TS_1$ ) is matched by an initial state in the more abstract one ( $TS_2$ ). Hence, Definition 3.1.2 requires that each state  $s_1 \in I_1$  of  $TS_1$  can be mimicked by at least one state  $s_2 \in I_2$  of  $TS_2$ , such that  $(s_1, s_2) \in \mathcal{R}$ .

Note that the stutter simulation relation of pairs of states  $(s_1, s_2)$  of a single transition system  $TS$  can be obtained from the above definition. First  $s_1$  is declared as the unique initial state of  $TS$  and the thus obtained transition system will be denoted by  $TS_{s_1}$ . Afterwards, this procedure is repeated for  $s_2$  to get  $TS_{s_2}$ . It immediately follows that  $s_1 \preceq_{TS} s_2$  if and only if  $TS_{s_1} \preceq TS_{s_2}$ .

On the other hand, if we take the disjoint union  $TS_1 \oplus TS_2 = TS_{1,2}$  of  $TS_1$  and  $TS_2$ , then  $TS_1$  is stutter simulated by  $TS_2$ , if each initial state of  $TS_1$  is stutter simulated by an initial state  $s_2$  of  $TS_2$  in  $TS_{1,2}$ . Consequently, the union leads to one single transition system, such that we have  $TS_1 \preceq TS_2$  if and only if  $s_1 \preceq_{TS_{1,2}} s_2$  for all  $s_1 \in I_1$  and some  $s_2 \in I_2$ .

#### EXAMPLE 3.1.1 ( STUTTER SIMULATING TRANSITION SYSTEMS )

Consider Figure 3.2. The transition system on the left is denoted by  $TS_1$ , the one on the right by  $TS_2$ . Each of them exhibits a single initial state,  $s_1$  and  $t_1$ , respectively. There exists a stutter simulation  $\mathcal{R}$  for  $(TS_1, TS_2)$ :

$$\mathcal{R} = \{(s_1, t_1), (s_2, t_2), (s_3, t_2), (s_2, t_3), (s_4, t_4)\},$$

and thus  $s_1$  is stutter simulated by  $t_1$ . Path fragment  $s_1 s_2$  can be mimicked by  $TS_2$  in the obvious way, i.e.  $t_1 \rightarrow t_2$ . Hence, by Definition 3.1.1, for the tuple  $(s_2, t_2)$  there must exist a stutter simulation: Observe that  $s_4$  is the only direct successor of  $s_2$  which cannot be stutter simulated by  $t_2$ :  $s_2 \rightarrow s_4$  is matched by  $t_2 \rightarrow t_3 \rightarrow t_4$ , where  $t_3$  and  $t_4$  stutter simulate  $s_2$  and  $s_4$ , respectively. Hence,  $(s_2, t_3), (s_4, t_4) \in \mathcal{R}$ . (Note that  $L(s_4) = L(t_4)$  and the only transition of  $s_4$  is a self-loop, which leads directly to the fact that  $s_4 \preceq t_4$ .) Thus,  $TS_1 \preceq TS_2$ .

The reverse,  $TS_2 \preceq TS_1$ , does not hold, since there exists no state  $s \in S_2$  that stutter simulates  $t_4$ . ■

#### LEMMA 3.1.1 ( STUTTER SIMULATION ORDER $\preceq$ IS A PREORDER )

For a fixed set of atomic propositions  $AP$ , the relation  $\preceq$  is a preorder.

PROOF Recall that a binary relation  $\mathcal{R}$  on some set  $N$  is a *preorder*, if it is (1) *reflexive* and (2) *transitive*.



Figure 3.2: Stutter Simulating Transition Systems

- (1) Clearly, for each state  $s \in S$  of a transition system  $TS$ , it holds that  $s \leq s$ . Hence, the identity relation  $\mathcal{R}_{id} = \{(s, s) \mid s \in S\}$  is a stutter simulation for the pair  $(TS, TS)$ . Thus, for any transition system  $TS$ , the relation  $\leq$  on  $(TS, TS)$  is reflexive.
- (2) Assume  $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP, L_i)$ ,  $i = 1, 2, 3$ , to be three transition systems over the same set of atomic propositions  $AP$ . Let  $\mathcal{R}_{1,2}$  and  $\mathcal{R}_{2,3}$  be the stutter simulations for the pairs  $(TS_1, TS_2)$  and  $(TS_2, TS_3)$ , respectively. The relation  $\mathcal{R}_{1,3} := \mathcal{R}_{1,2} \circ \mathcal{R}_{2,3}$  is defined as follows:

$$\mathcal{R}_{1,3} = \{(s_1, s_3) \mid \exists s_2 \in S_2. (s_1, s_2) \in \mathcal{R}_{1,2} \wedge (s_2, s_3) \in \mathcal{R}_{2,3}\}.$$

We now prove that  $\mathcal{R}_{1,3}$  is a stutter simulation for  $(TS_1, TS_3)$ :

Based on the assumption that  $\mathcal{R}_{1,2}$  is a stutter simulation, for each initial state  $s_1 \in I_1$  there exists a state  $s_2 \in I_2$  with  $(s_1, s_2) \in \mathcal{R}_{1,2}$ . Likewise, since  $\mathcal{R}_{2,3}$  is a stutter simulation, there exists an initial state  $s_3 \in I_3$  for every  $s_2 \in I_2$ , such that  $(s_2, s_3) \in \mathcal{R}_{2,3}$ . Hence, it follows immediately that for all  $s_1 \in I_1$  there exists a state  $s_3 \in I_3$  with  $(s_1, s_3) \in \mathcal{R}_{1,3}$ . This shows condition (A) of Definition 3.1.2.

It remains to show that every  $(s_1, s_3) \in \mathcal{R}_{1,3}$  satisfies conditions (B.1) and (B.2):

- (2.1)  $L(s_1) = L(s_3)$  is trivially satisfied, since by definition the pair  $(s_1, s_3)$  can only be included in  $\mathcal{R}_{1,3}$ , if there exists a state  $s_2$  with  $(s_1, s_2) \in \mathcal{R}_{1,2}$  (which requires that  $L(s_1) = L(s_2)$ ) and  $(s_2, s_3) \in \mathcal{R}_{2,3}$ —where  $s_2$  and  $s_3$  must have the same labeling and thus  $L(s_1) = L(s_2) = L(s_3)$ .
- (2.2) Let  $s_2$  be a state such that  $(s_1, s_2) \in \mathcal{R}_{1,2}$  and  $(s_2, s_3) \in \mathcal{R}_{2,3}$ . For each  $s'_1 \in Post(s_1)$  one of the following two cases applies:
  - (2.2.1)  $(s'_1, s_2) \in \mathcal{R}_{1,2}$ . Assume  $(s'_1, s_3) \notin \mathcal{R}_{1,3}$ . Then  $(s_2, s_3) \notin \mathcal{R}_{2,3}$ . This is a contradiction to the assumption that  $(s_1, s_3) \in \mathcal{R}_{1,3}$ . Thus,  $(s'_1, s_3) \in \mathcal{R}_{1,3}$ .
  - (2.2.2)  $(s'_1, s_2) \notin \mathcal{R}_{1,2}$  and there exists a finite path fragment  $s_2 u_1 \dots u_n s'_2$ ,  $n \geq 0$ , where  $(s_1, u_i) \in \mathcal{R}_{1,2}$  for all  $i \in \{1, \dots, n\}$  and  $(s'_1, s'_2) \in \mathcal{R}_{1,2}$ . Since  $(s_2, s_3) \in \mathcal{R}_{2,3}$ , for all  $s'_2 \in Post(s_2)$  either  $(s'_2, s_3) \in \mathcal{R}_{2,3}$  or there exists a finite path fragment  $s_3 v_1 \dots v_m s'_3$ ,  $m \geq 0$ , with  $(s_2, v_i) \in \mathcal{R}_{1,2}$  for all  $i \in \{1, \dots, m\}$  and  $(s'_2, s'_3) \in \mathcal{R}_{2,3}$ . In both cases it follows:  $(s'_1, s_3) \in \mathcal{R}_{1,3}$ .

Thus, the relation  $\leq$  is transitive. ■

**LEMMA 3.1.2 ( COARSEST STUTTER SIMULATION )**

For transition system  $TS = (S, Act, \rightarrow, I, AP, L)$  it holds that:

- (1)  $\leq_{TS}$  is a preorder on  $S$ .
- (2)  $\leq_{TS}$  is a stutter simulation on  $TS$ .
- (3)  $\leq_{TS}$  is the coarsest stutter simulation for  $TS$ .

PROOF Let  $TS = (S, Act, \rightarrow, I, AP, L)$  be a transition system and  $s_1, s_2 \in S$ .

The first claim was already proven in Lemma 3.1.1. The second statement can be seen as follows: Assume  $s_1 \leq_{TS} s_2$ . By Definition 3.1.1, there exists a stutter simulation  $\mathcal{R}$ , where  $(s_1, s_2) \in \mathcal{R}$ . Each pair of states in  $\mathcal{R}$  must satisfy conditions (1) and (2), which indicates that  $L(s_1) = L(s_2)$  and whenever there is a successor  $s'_1$  of  $s_1$  with  $(s'_1, s_2) \notin \mathcal{R}$ , then there exists a finite path fragment  $s_2 u_1 \dots u_n s'_2$ ,  $n \geq 0$ , such that  $(s_1, u_i) \in \mathcal{R}$ ,  $i = 1, \dots, n$ , and  $(s'_1, s'_2) \in \mathcal{R}$ . Then  $s'_1$  is stutter simulated by  $s'_2$ , i.e.  $s'_1 \leq_{TS} s'_2$ . It follows that  $\leq_{TS}$  is a stutter simulation on  $TS$ .

For claim (3) assume that there exists a stutter simulation  $\mathcal{R}$  on  $(TS, TS)$ , such that  $(s_1, s_2) \in \mathcal{R}$ , but  $s_1 \not\leq_{TS} s_2$ . Then either  $L(s_1) \neq L(s_2)$  or there exists no finite path fragment  $s_2 u_1 \dots u_n s'_2$  with the desired properties of condition (2) in Definition 3.1.1, if  $s'_1 \in Post(s_1)$  with  $(s'_1, s_2) \notin \mathcal{R}$ . Both cases contradict the assumption that  $(s_1, s_2) \in \mathcal{R}$ . Thus,  $s_1 \leq_{TS} s_2$  is satisfied whenever there exists a stutter simulation  $\mathcal{R}$  containing  $(s_1, s_2)$  for all  $s_1, s_2 \in S$  and since  $\leq_{TS}$  is a stutter simulation, it follows that  $\leq_{TS}$  is the coarsest stutter simulation. ■

The remainder of this chapter is concerned with *stutter simulating paths*, which is especially interesting if language preservation is considered, since for example the formulae of the temporal logic *LTL* are interpreted over the paths of a transition system  $TS$ .

For simulation and bisimulation, two paths are *statewise* comparable and hence statewise related to each other in the sense of bisimulation equivalence or simulation preorder. This concept cannot be adapted to stutter simulating paths: The notion of stuttering allows us—when identifying a state to stutter simulate another one—to match a set of states to one single state. In this case the two associated paths cannot be statewise related. For example consider Figure 3.2, where  $\pi_1 = s_1 s_2 s_4 s_4 \dots = s_1 s_2 (s_4)^\omega$  is an infinite path in  $TS_1$  (left),  $\pi_2 = t_1 t_2 t_3 (t_4)^\omega$  is an infinite path in  $TS_2$  (right) and there exists a stutter simulation  $\mathcal{R}$ , such that  $(s_1, t_1) \in \mathcal{R}$ . Observe that  $t_2$  and  $t_3$  both stutter simulate state  $s_2$ . Thus, a statewise comparison of  $\pi_1$  and  $\pi_2$  is not sensible here. De Nicola and Vaandrager [3] solved this problem for stutter *bisimulating* paths by partitioning the examined paths into certain segments, where each state of the  $j$ th segment of  $\pi_1$  is stutter bisimilar to each state of the  $j$ th segment of  $\pi_2$ , and vice versa. For stutter simulating paths this leads to the following definition.

**DEFINITION 3.1.3 ( STUTTER SIMULATING PATHS )**

Let  $TS = (S, Act, \rightarrow, I, AP, L)$  be a transition system and  $\pi_1 = s_0 s_1 s_2 \dots$  and  $\pi_2 = t_0 t_1 t_2 \dots$  be two (finite or infinite) path fragments. A *segment* is a (finite or infinite) path fragment of  $\pi_1$  or  $\pi_2$ . Then:

$$\pi_1 \leq_{TS} \pi_2$$

if and only if  $\pi_1$  can be partitioned as  $\pi_{1,0}\pi_{1,1}\pi_{1,2}\dots$  and  $\pi_2$  as  $\pi_{2,0}\pi_{2,1}\pi_{2,2}\dots$ , such that for all  $i$ , segments  $\pi_{1,i}$  and  $\pi_{2,i}$  are both nonempty and every state in  $\pi_{1,i}$  is stutter simulated by every state in  $\pi_{2,i}$ . ■

The number of segments never exceeds the number of states on  $\pi_1$ , since all segments are nonempty and  $\pi_{1,0}\pi_{1,1}\pi_{1,2}\dots$  is a partition, which leads to the fact that for each state  $s_j$  on  $\pi_1$  there exists exactly one segment that contains it.

EXAMPLE 3.1.2 (STUTTER SIMULATING PATHS)

Let paths  $\pi_1$  and  $\pi_2$  in Figure 3.3 be infinite paths of some transition system  $TS$ . For each state its labeling is indicated either above (in case it belongs to  $\pi_1$ ) or below (if it belongs to  $\pi_2$ ) its name. A dotted line between two states  $s_j, t_i$  indicates that  $s_j$  is stutter simulated by  $t_i$  (Note that if two states have the same labeling, this does not necessarily yield that they are related in the stutter simulation sense!). If the color of the dotted lines of two states  $s_i, s_j$  (or  $t_i, t_j$ ) is equal, then they are in the same segment of the particular partition of the related path.

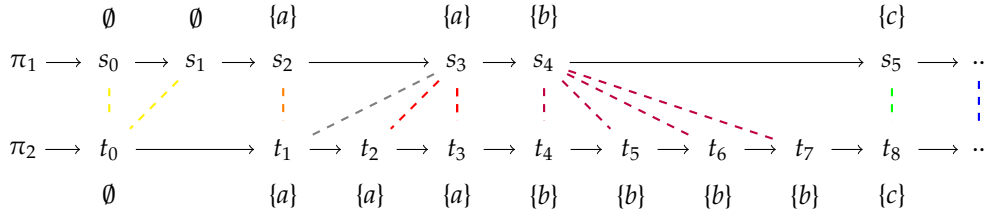


Figure 3.3: Stutter Simulating Path Fragments

The gray line is a special case:  $t_1$  stutter simulates both  $s_2$  and  $s_3$ , but not  $s_4$  and has no direct successor that stutter simulates  $s_4$ . Thus, there exists a finite path fragment  $t_2t_3$  that leads to  $t_4$ , where  $s_4 \leq_{TS} t_4$ . Observe that, by definition of stutter simulating states,  $s_3$  must be stutter simulated by all states on the path fragment  $t_2t_3$ , which is obviously the case (red dotted lines). The problem is that it does not require (and therefore it does not assure) that  $t_2$  and  $t_3$  stutter simulate  $s_2$ , too, and hence, they cannot be in the same segment as  $t_1$ . Thus, although  $s_3 \leq_{TS} t_1$ , state  $s_3$  is not in the segment of states that are stutter simulated by  $t_1$ .

Observe that  $\pi_1$  is stutter simulated by  $\pi_2$  (the blue line implies that the remaining infinite parts of  $\pi_1$  and  $\pi_2$  can be correctly partitioned). Table 3.4 shows the partition for each path up to the fifth segment. Note that—since the paths are infinite—the partitions might be finite, such that the respective last segment is infinite. Each state in  $\pi_{1,i}$  is stutter simulated by each state in  $\pi_{2,i}$ .

Now consider Figure 3.2 and construct  $TS = TS_1 \oplus TS_2$ . Let  $\pi_1$  and  $\pi_2$  be two infinite paths in  $TS$ , where  $\pi_1 = s_1s_2s_4s_4s_4\dots$  and  $\pi_2 = t_1t_2t_3t_4t_1t_2t_3t_3\dots$ . Then, there does not exist any partition of  $\pi_1$  and  $\pi_2$ , such that  $\pi_2 \leq_{TS} \pi_1$ . This is because once the  $c$ -labeled state  $s_4$  on path  $\pi_1$  is reached, it is never left again and since  $\pi_2$  is infinite and its only  $c$ -labeled state  $t_4$  is visited exactly once, it follows that the paths cannot be partitioned in the desired way and thus,  $\pi_1$  cannot stutter simulate  $\pi_2$ . ■

Segment of $\pi_i$	Path Fragment of $\pi_i$	
	$i = 1$	$i = 2$
$\pi_{i,0}$	$s_0s_1$	$t_0$
$\pi_{i,1}$	$s_2$	$t_1$
$\pi_{i,2}$	$s_3$	$t_2t_3$
$\pi_{i,3}$	$s_4$	$t_4t_5t_6t_7$
$\pi_{i,4}$	$s_5$	$t_8$
...	...	...

Figure 3.4: Partitioning of Path Fragments

Definition 3.1.3 leads to the question whether it can be adapted to stutter simulating states, i.e. if  $s_0 \leq_{TS} t_0$ , does this imply that for every path  $\pi_1 \in Paths(s_0)$  there exists a path  $\pi_2 \in Paths(t_0)$ , such that  $\pi_1$  is stutter simulated by  $\pi_2$ . And furthermore, if the response is positive, what can we conclude concerning language preservation. In fact, the answer is positive but considering some logic  $L$ , the so far defined standard variant of stutter simulation imposes two problems on language preservation. The first one concerns formulae containing the next-step operator  $\bigcirc$ , whereas the second is related to the differentiation between paths either exhibiting a certain property—the so called *divergence*-property—or not. Observe the transition system  $TS$  given in Figure 3.5, which illustrates the two cases.



Figure 3.5: Language Preservation vs. Stutter Simulation

Let  $L$  be the *linear temporal logic* ( $LTL$ , see Section 2.2),  $\varphi_1 = \bigcirc b$  and  $\varphi_2 = \text{true} \bigcup b \equiv \diamond b$  be two formulae in  $LTL$  and observe that  $s_0$  is stutter simulated by  $t_0$ .  $\varphi_1$  requires that each direct successor of  $s_0$  and  $t_0$  is labeled with  $b$ . State  $t_1$  is the unique successor of  $t_0$  and since  $L(t_1) = b$ , it follows that  $t_0$  satisfies  $\varphi_1$  ( $t_0 \models \varphi_1$ , for short). In contrast to that,  $s_0$  does not satisfy  $\varphi_1$ , since  $s_1 \in Post(s_0)$  and  $L(s_1) \neq b$ . The problem is rooted in the definition of stutter simulation, where  $s_0$  may perform the step  $s_0 \rightarrow s_1$  but it does not have to be mimicked by  $t_0$ , since  $s_1 \leq_{TS} t_0$ , and thus,  $s_0$  might have some successor  $s'_0$  with  $L(s'_0) \neq L(t'_0)$  for all  $t'_0 \in Post(t_0)$ , and vice versa. Fortunately, this case can easily be fixed by simply restricting to logics without the next-step operator, denoted  $L_{\setminus \bigcirc}$ .

The second problem, however, is more complex and requires to define a new variant of stutter simulation that we will call *divergence-sensitive stutter simulation*. Consider states  $s_1$  and  $t_0$ , where  $s_1$  is stutter simulated by  $t_0$ . Formula  $\varphi_2$  requires that *each* path emanating from  $s_1$  and  $t_0$  *eventually* reaches a state that is labeled by  $b$ . Clearly,  $t_0 \models \varphi_2$ , whereas this is not the case for  $s_1$ , since path  $\pi_1 = (s_1)^\omega \in Paths(s_1)$  and  $L(s_1) \neq b$ . The consequence is that,

although  $t_0$  stutter simulates  $s_1$ , they do not satisfy the same  $LTL_{\setminus \circ}$  formulae. This is caused by the fact that path  $\pi_1$  is *divergent*, i.e. only states are visited that are stutter simulated by  $t_0$ . In this case, there are no steps performed on  $\pi_1$  that are visible to  $t_0$  and hence, no steps that must be imitated by  $t_0$ , since  $s_1 \preceq_{TS} t_0$  is already valid. Thus, the standard variant of stutter simulation does not distinguish between divergent and non-divergent paths and consequently does not preserve  $LTL_{\setminus \circ}$ . We will now give a formal definition of the divergence-property and the *divergence-sensitive stutter simulation* and we will show that this variant preserves  $LTL_{\setminus \circ}$ .

**DEFINITION 3.1.4 (  $\mathcal{R}$ -DIVERGENT STATE, DIVERGENCE-SENSITIVITY )**

Let  $TS$  be a transition system,  $\mathcal{R}$  a preorder relation on  $S$  and  $s_0, t_0 \in S$ .

- $s_0$  is  $\mathcal{R}$ -divergent if there exists an infinite path fragment  $\pi = s_0s_1s_2\dots \in Paths(s_0)$ , such that  $(s_i, t_0) \in \mathcal{R}$  for all  $i \geq 0$ .
- $\mathcal{R}$  is *divergence-sensitive* if for any  $(s_0, t_0) \in \mathcal{R}$  and each infinite path fragment  $\pi = s_0s_1s_2\dots$  with  $(s_i, t_0) \in \mathcal{R}$  for all  $i \geq 0$  there exists  $t'_0 \in Post(t_0)$  with  $(s_j, t'_0) \in \mathcal{R}$  for some  $j \geq 1$ . ■

Roughly speaking this requires that whenever there is an infinite path fragment  $\pi$  emanating from some state  $s_0$ , then  $s_0$  is  $\mathcal{R}$ -divergent, if all steps performed on  $\pi$  are *invisible* to  $t_0$  and if there exists an infinite path starting in  $t_0$ , where each state is  $\mathcal{R}$ -related to infinitely many states on  $\pi$ .

**DEFINITION 3.1.5 ( STUTTER SIMULATION WITH DIVERGENCE )**

Let  $TS$  be a transition system and  $s_1, s_2$  be two states in  $S$ . Then:  $s_1$  is *divergent stutter simulated* by  $s_2$ , denoted  $s_1 \preceq_{TS}^{div} s_2$ , if there exists a divergence-sensitive stutter simulation  $\mathcal{R}$  for  $TS$  with  $(s_1, s_2) \in \mathcal{R}$  ■

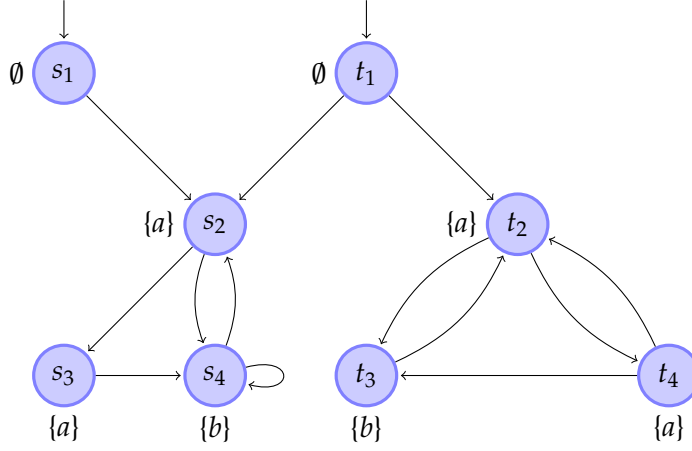
As in the case for  $\preceq_{TS}$ , one can prove that  $\preceq_{TS}^{div}$  is a preorder on  $S$  and the coarsest divergence-sensitive stutter simulation for  $TS$ . Since the proof is basically the same, the technical details are omitted here.

**EXAMPLE 3.1.3 ( STUTTER SIMULATION WITH DIVERGENCE )**

Consider Figure 3.6. State  $t_1$  is stutter simulated by  $s_1$ , i.e.  $t_1 \preceq_{TS} s_1$ , since there exists a stutter simulation  $\mathcal{R}$  that contains  $(t_1, s_1)$ :

$$\mathcal{R} = \{ (t_1, s_1), (t_2, s_2), (t_2, s_3), (s_2, s_2), (t_3, s_4), (t_4, s_2) \}$$

There exists, however, no divergence-sensitive stutter simulation for  $(t_1, s_1)$ . This can be seen as follows. Assume  $\mathcal{R}'$  is a divergence-sensitive stutter simulation for  $(t_1, s_1)$  and observe that  $t_2$  is a direct successor of  $t_1$  with  $L(s_1) \neq L(t_2)$ . Obviously,  $s_2$  is the only successor of  $s_1$  that is a *candidate* to divergent stutter simulate  $t_2$  and we have to check whether  $(t_2, s_2) \in \mathcal{R}'$ . From  $t_2$  there emanates an infinite path  $\pi = u_0u_1u_2\dots = (t_2t_4)^\omega$ , where  $s_2$  stutter simulates  $u_i$  for all  $i \geq 0$ . The condition for divergence-sensitivity requires that there exists a state  $s'_2 \in Post(s_2)$  with  $(u_j, s'_2) \in \mathcal{R}'$ ,  $j > 0$ . State  $s_2$  has two successors  $s_3, s_4$ , where  $s_3$  is the unique one labeled by  $a$ . Then, the pair  $(u_j, s_3)$  must be contained in  $\mathcal{R}'$  for some  $j \geq 1$ . Note that if  $u_j \preceq_{TS}^{div} s_3$ , then  $(u_k, s_3) \in \mathcal{R}'$  for all  $k \geq j$  and by definition there must exist some  $s'_3 \in Post(s_3)$  with  $u_l \preceq_{TS}^{div} s'_3$  for some  $l > k$ . But  $s_3$  has no such successor— $L(s_4) \neq a$  and  $s_4$  is the unique successor of  $s_3$ —and we obtain that  $(u_k, s_3) \notin \mathcal{R}'$  for all  $k > j$  and furthermore,  $(t_2, s_2) \notin \mathcal{R}'$ .


 Figure 3.6:  $t_1 \leq_{TS} s_1$  but  $t_1 \not\leq_{TS}^{div} s_1$ 

Thus, there exists no divergence-sensitive stutter simulation  $\mathcal{R}'$  with  $(t_1, s_1) \in \mathcal{R}'$ . Note that by introducing self-loops to states  $s_3$  and  $t_3$ , we obtain that  $t_1 \leq_{TS}^{div} s_1$ . The corresponding divergence-sensitive stutter simulation  $\mathcal{R}_{div}$  consists of the following pairs of states:

$$\mathcal{R}_{div} = \{ (t_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4), (t_2, s_2), (t_4, s_2), (t_2, s_3), (t_4, s_3), (t_3, s_4), \} \quad \blacksquare$$

Adapting this notion to pairs of transition systems  $TS_1, TS_2$ , we have that  $TS_1 \leq_{TS_2}^{div}$  iff for each initial state  $s_1 \in I_1$  there exists an initial state  $s_2 \in I_2$ , such that  $s_1 \leq_{TS_1 \oplus TS_2}^{div} s_2$ .

#### DEFINITION 3.1.6 ( DIVERGENT STUTTER SIMULATING PATHS )

Given transition system  $TS$  and let  $\pi_1 = s_0 s_1 s_2 \dots$  and  $\pi_2 = t_0 t_1 t_2 \dots$  be two (finite or infinite) path fragments. A *segment* is a (finite or infinite) path fragment of  $\pi_1$  or  $\pi_2$ . Then:

$$\pi_1 \leq_{TS}^{div} \pi_2$$

if and only if  $\pi_1$  can be partitioned as  $\pi_{1,0} \pi_{1,1} \pi_{1,2} \dots$  and  $\pi_2$  as  $\pi_{2,0} \pi_{2,1} \pi_{2,2} \dots$ , such that for all  $i$  segments  $\pi_{1,i}$  and  $\pi_{2,i}$  are both nonempty and every state in  $\pi_{1,i}$  is divergent stutter simulated by every state in  $\pi_{2,i}$ .  $\blacksquare$

According to Definition 3.1.3 the number of segments never exceeds the number of states on  $\pi_1$ .

We will now prove that if state  $s_0$  is divergent stutter simulated by a state  $t_0$ , then each path emanating from  $s_0$  is divergent stutter simulated by some path starting in  $t_0$ .

#### LEMMA 3.1.3 ( PATH LIFTING FOR DIVERGENT STUTTER SIMULATING STATES )

Let  $TS = (S, Act, \rightarrow, I, AP, L)$  be a transition system,  $s_0, t_0 \in S$ . Then:

$$\text{If } s_0 \leq_{TS}^{div} t_0 \text{ then } \forall \pi_1 \in Paths(s_0). (\exists \pi_2 \in Paths(t_0). \pi_1 \leq_{TS}^{div} \pi_2).$$

PROOF Let  $\pi_1 = s_0s_1s_2\dots \in Paths(s_0)$ . The idea is to segmentwise construct path  $\pi_2$  emanating from  $t_0$ , such that each state on segment  $i$  of path  $\pi_1$  is divergent stutter simulated by each state on segment  $i$  of  $\pi_2$ . Hence, whenever there is a transition  $s_j \rightarrow s_{j+1}$ , with  $s_j \preceq_{TS}^{div} t_n$ ,  $s_j$  and  $t_n$  are in segments  $\pi_{1,n}$  and  $\pi_{2,n}$ , respectively, and if  $s_{j+1} \not\preceq_{TS}^{div} t_n$ , then a finite path fragment  $t_n u_{n,1} \dots u_{n,m_n} t_{n+1}$  is generated, such that  $s_j \preceq_{TS}^{div} u_{n,i}$ ,  $i = 1, \dots, m_n$ , and  $s_{j+1} \preceq_{TS}^{div} t_{n+1}$ . Since  $s_{j+1} \not\preceq_{TS}^{div} t_n$ , segment  $\pi_{1,n+1}$  is created, which consists of  $s_{j+1}$  and likewise  $\pi_{2,n+1}$  that consists of  $t_{n+1}$ . Let  $\pi_{p,q}$  denote the specific segment, where  $p \in \{1, 2\}$  identifies the path fragment  $\pi_1$  or  $\pi_2$  it belongs to and  $q$  its position in the partition. The proof is by induction on  $j$ , that specifies the current position in  $\pi_1$ :

(1) Base case:  $j = 0$ .

(We will first show that if  $s_0$  is a terminal state, then there exists  $\pi_2$  emanating from  $t_0$ , such that  $\pi_1$  is divergent stutter simulated by  $\pi_2$ . The technique described in (1.1) can be applied to every case of  $j$ , where  $s_j$  is a terminal state. Thus, we will assume in all other cases without loss of generality that  $s_j$  is not a terminal state.)

$\pi_2$  emanates from  $t_0$ . Four cases:

(1.1)  $s_0$  is a terminal state, i.e.  $\pi_1 = s_0$ . Since  $s_0 \preceq_{TS}^{div} t_0$ , segment  $\pi_{1,0}$  consists of  $s_0$ , whereas state  $t_0$  belongs to  $\pi_{2,0}$ . It directly follows that  $\pi_1 \preceq_{TS}^{div} \pi_2$ .

(1.2)  $s_0$  is not a terminal state and  $s_1 \not\preceq_{TS}^{div} t_0$ . Since  $s_0 \preceq_{TS}^{div} t_0$ , there exists a finite path fragment  $t_0 u_{0,1} \dots u_{0,m_0} t_1$ , such that  $s_0 \preceq_{TS}^{div} u_{0,i}$  for all  $i \leq m_0$  and  $s_1 \preceq_{TS}^{div} t_1$ .  $\pi_{1,0} = s_0$  and  $\pi_{2,0} = t_0$  is extended with the intermediate path fragment  $u_{0,1} \dots u_{0,m_0}$ . For states  $s_1$  and  $t_1$  two segments are defined, namely  $\pi_{1,1} = s_1$  and  $\pi_{2,1} = t_1$ . Concatenation of the segments  $\pi_{2,0}$  and  $\pi_{2,1}$  leads to the path fragment  $t_0 u_{0,1} \dots u_{0,m_0} t_1$ . The segmentation and created path fragment fulfill the desired conditions.

(1.3)  $s_0$  is not a terminal state and  $s_1 \preceq_{TS}^{div} t_0$ . Distinguish between two cases:

(1.3.1)  $s_0$  is not divergent, i.e. there exists an  $l > 1$ , such that  $s_l \not\preceq_{TS}^{div} t_0$ . (W.l.o.g.)

Assume  $l$  is minimal, i.e. there exists no  $l'$  with  $1 < l' < l$  and  $s_{l'} \not\preceq_{TS}^{div} t_0$ . Since  $s_{l-1} \preceq_{TS}^{div} t_0$ , there exists a finite path fragment  $t_0 u_{0,1} u_{0,2} \dots u_{0,m_0} t_1$ ,  $m_0 \geq 0$ , with  $s_{l-1} \preceq_{TS}^{div} u_{0,i}$  for all  $i \leq m_0$  and  $s_l \preceq_{TS}^{div} t_1$ .

If  $m_0 = 0$ , then  $t_0$  has a direct successor  $t_1$  that divergent stutter simulates  $s_l$ . In this case segment  $\pi_{1,0}$  consists of the path fragment  $s_0 s_1 \dots s_{l-1}$ . As a consequence of the fact that  $s_l \not\preceq_{TS}^{div} t_0$ , a new segment  $\pi_{1,1}$ , which inherits  $s_l$ , must be generated. Clearly,  $\pi_{2,0} = t_0$ . Since  $t_0$  cannot divergent stutter simulate  $s_l$ , it cannot be in the same segment as  $t_1$ . Thus, create  $\pi_{2,1} = t_1$  and  $\pi_{1,1} = s_l$ . Last but not least segments  $\pi_{2,0}$  and  $\pi_{2,1}$  are concatenated and we obtain the path fragment  $t_0 t_1$ .

If  $m_0 > 0$ , then  $t_1$  is not a direct successor of  $t_0$  and is reached from  $t_0$  via the finite path fragment  $u_{0,1} u_{0,2} \dots u_{0,m_0}$ . Then,  $\pi_{1,0} = s_0 s_1 \dots s_{l-2}$ ,  $\pi_{2,0} = t_0$  and we have to define new segments  $\pi_{1,1} = s_{l-1}$  and  $\pi_{2,1} = u_{0,1} \dots u_{0,m_0}$ . States  $u_{0,i}$ , with  $s_{l-1} \preceq_{TS}^{div} u_{0,i}$ , do not necessarily divergent stutter simulate each state  $s_k$ ,  $0 \leq k < l-1$ , since the conditions for divergent stutter simulation only require that  $s_{l-1} \preceq_{TS}^{div} u_{0,i}$  for all  $i \leq m_0$ . Thus, the intermediate states  $u_{0,i}$

cannot be in the same segment as  $t_0$ . It remains to generate  $\pi_{1,2} = s_l$  and  $\pi_{2,2} = t_1$ . Concatenation of the segments  $\pi_{2,0}, \pi_{2,1}$  and  $\pi_{2,2}$  leads to the path fragment  $t_0 u_{0,1} \dots u_{0,m_0} t_1$ . The constructed path fragments and segments fulfill the conditions for divergent stutter simulating path fragments.

- (1.3.2)  $s_0$  is divergent, i.e. there exists no  $l > 0$ , such that  $s_l \not\leq_{TS}^{div} t_0$ . Since  $s_0 \leq_{TS}^{div} t_0$ , there exists a direct successor  $t_1$  of  $t_0$  and some  $i > 0$ , such that  $s_i \leq_{TS}^{div} t_1$ . Segment  $\pi_{1,0} = s_0, \dots, s_{i-1}$ , whereas  $\pi_{2,0} = t_0$ , since  $s_k \leq_{TS}^{div} t_0$  for all  $k \in \{0, \dots, i-1\}$ . It is not guaranteed that  $t_1$  divergent stutter simulates states  $s_0, \dots, s_{i-1}$  and thus, we have to generate new segments where  $\pi_{1,1} = s_i$  and  $\pi_{2,1} = t_1$ . Concatenation of the segments  $\pi_{2,0}, \pi_{2,1}$  leads to the path fragment  $t_0 t_1$  and fulfills the desired conditions.

- (2) Assume  $j > 0$ , where path fragment

$$t_0 u_{0,1} \dots u_{0,m_0} t_1 u_{1,1} \dots u_{1,m_1} t_2 \dots t_n \quad (3.1)$$

is correctly created and partitioned into  $n$  segments,  $n \leq j$ , and state  $s_j$  belongs to segment  $\pi_{1,n}$ , whereas  $t_n$  is associated with  $\pi_{2,n}$ . (Figure 3.7 illustrates the following cases and is interpreted in the same way as explained in Example 3.1.2 except for replacing stutter simulation by divergence-sensitive stutter simulation.)

- (2.1)  $s_{j+1} \not\leq_{TS}^{div} t_n$ . Since  $s_j \leq_{TS}^{div} t_n$ , there exists a finite path fragment  $t_n u_{n,1} \dots u_{n,m_n} t_{n+1}$ , such that  $s_j \leq_{TS}^{div} u_{n,i}$  for all  $i \leq m_n$  and  $s_{j+1} \leq_{TS}^{div} t_{n+1}$ .  $\pi_{1,n}$  remains the same and  $\pi_{2,n} = t_n$  is extended with the intermediate path fragment  $u_{n,1} \dots u_{n,m_n}$ .  $\pi_{1,n+1} = s_{j+1}$  and  $\pi_{2,n+1} = t_{n+1}$  are constructed, since  $t_n$  and  $t_{n+1}$  must not be in the same segment. Concatenation of (3.1) with  $u_{n,1} \dots u_{n,m_n}$  and  $\pi_{2,n+1}$  leads to a path fragment that satisfies the required conditions.

- (2.2)  $s_{j+1} \leq_{TS}^{div} t_n$ . Distinguish between two cases:

- (2.2.1)  $s_j$  is divergent, i.e. there exists some  $l > j + 1$ , such that  $s_l \not\leq_{TS}^{div} t_n$ . (W.l.o.g.) Assume  $l$  is minimal, i.e. there exists no  $l'$  with  $j + 1 < l' < l$  and  $s_{l'} \not\leq_{TS}^{div} t_n$ . Then,  $\pi_{1,n}$  is extended with all states  $s_k$ , where  $j \leq k < l$ . Since  $s_{l-1} \leq_{TS}^{div} t_n$ , there exists a finite path fragment  $t_n u_{n,1} u_{n,2} \dots u_{n,m_n} t_1$ ,  $m_n \geq 0$ , with  $s_{l-1} \leq_{TS}^{div} u_{n,i}$  for all  $i \leq m_n$  and  $s_l \leq_{TS}^{div} t_{n+1}$ .

If  $m_n = 0$ , then  $t_n \rightarrow t_{n+1}$ . Segments  $\pi_{1,n+1} = s_{j+1}$  and  $\pi_{2,n+1} = t_{n+1}$  are generated, since  $t_n$  cannot be in the same segment as  $t_{n+1}$ , based on the fact that  $s_l \not\leq_{TS}^{div} t_n$ , whereas  $s_l \leq_{TS}^{div} t_{n+1}$ . It remains to concatenate the existing path fragment (3.1) with segment  $\pi_{2,n+1}$ .

If  $m_n > 0$ , then  $t_{n+1}$  is reachable from  $t_n$  via the finite path fragment  $u_{n,1} \dots u_{n,m_n}$ . As a consequence of the fact that  $s_k \leq_{TS}^{div} u_{n,i}$  for all  $i \leq m_n$  and  $j \leq k < l - 1$ , is not mandatory, new segments  $\pi_{2,n+1}$  and  $\pi_{1,n+1}$ , that inherit  $u_{n,1} \dots u_{n,m_n}$  and  $s_{l-1}$ , respectively, must be defined. It remains to generate  $\pi_{1,n+2} = s_l$  and  $\pi_{2,n+2} = t_{n+1}$  and to concatenate (3.1) with the segments  $\pi_{2,1}$  and  $\pi_{2,2}$ .

In all cases we obtain path fragments and segments that fulfill the conditions for divergent stutter simulating path fragments.

(2.2.2)  $s_j$  is divergent, i.e.  $s_l \leq_{TS}^{div} t_n$  for all  $l \geq j$ . By definition of divergence-sensitive stutter simulation there exists  $t_{n+1} \in Post(t_n)$  and some  $i > j$ , such that  $s_i$  is divergent stutter simulated by  $t_{n+1}$ . Segment  $\pi_{1,n}$  is extended with states  $s_j, \dots, s_{i-1}$ , whereas in  $\pi_{2,n}$  state  $t_n$  is inserted, since  $s_k \leq_{TS}^{div} t_n$  for all  $k \in \{j, \dots, i-1\}$ . Clearly,  $t_n$  and  $t_{n+1}$  cannot be contained in the same segment, since it is not required that  $t_{n+1}$  divergent stutter simulates states  $s_j, \dots, s_{i-1}$ . Consequently we have to create two new segments  $\pi_{1,n+1}, \pi_{2,n+1}$ , which contain  $s_i$  and  $t_{n+1}$ , respectively. Concatenation of the segments  $\pi_{2,n}\pi_{2,n+1}$  with (3.1) leads to a path fragment satisfying the desired conditions.

The resulting path fragment  $\pi_2$  divergent stutter simulates  $\pi_1$ . ■

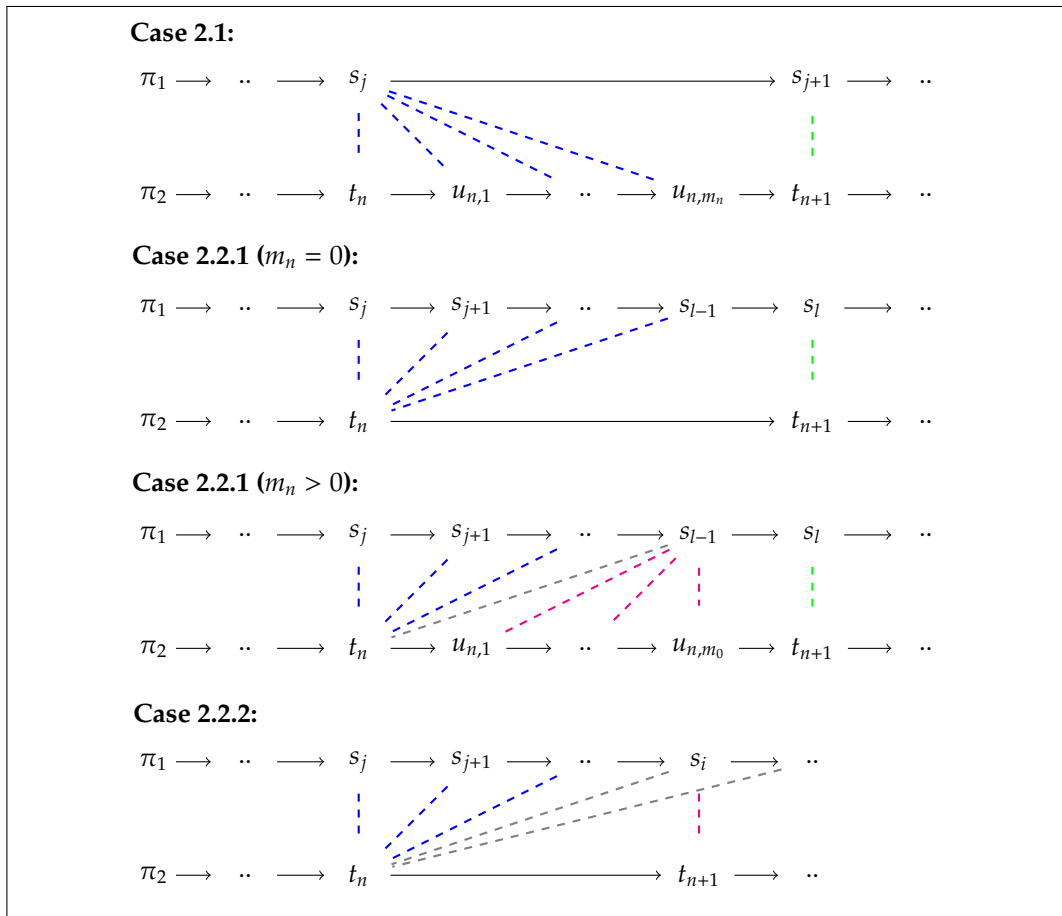


Figure 3.7: Illustration of Cases 2.1–2.2.2

The following Remark 3.1.1 is a direct consequence of Lemma 3.1.3.

**REMARK 3.1.1** ( $\leq_{TS}^{div}$  IMPLIES STUTTER-TRACE INCLUSION)

Let  $s, t$  be two states of some transition system  $TS$  without terminal states. Then:

$s \leq_{TS}^{div} t$  implies  $trace(Paths(s)) \subseteq trace(Paths(t))$ .

PROOF By Lemma 3.1.3 we have that if  $s \leq_{TS}^{div} t$  then  $\forall \pi_1 \in Paths(s). (\exists \pi_2 \in Paths(t). \pi_1 \leq_{TS}^{div} \pi_2)$ . By definition of  $\leq_{TS}^{div}$  it holds that  $L(u) = L(v)$  for all  $u \in \pi_{1,i}, v \in \pi_{2,i}$  and segments  $\pi_{1,i}, \pi_{2,i}$  of the partitions for  $\pi_1$  and  $\pi_2$ , respectively. Then, for each path  $\pi_1$  emanating from  $s$  there exists a path  $\pi_2$  starting in  $t$ , such that  $\pi_1$  is stutter-equivalent to  $\pi_2$ , i.e.  $\pi_1 \triangleq \pi_2$  and consequently,  $trace(Paths(s)) \subseteq trace(Paths(t))$ . ■

These two approaches enable to make a statement about the correlation between divergence-sensitive stutter simulation and preserving  $LTL_{\setminus \circ}$ .

**THEOREM 3.1.1 ( DIVERGENCE-SENSITIVE STUTTER SIMULATION ORDER AND  $LTL_{\setminus \circ}$  )**

Let  $TS$  be a transition system without terminal states and  $s, t$  be states in  $S$ .

If  $s \leq_{TS}^{div} t$ , then for all  $LTL_{\setminus \circ}$  formulae  $\varphi: t \models \varphi$  implies  $s \models \varphi$ .

PROOF Let  $\varphi$  be some  $LTL_{\setminus \circ}$ -formula and assume  $s \leq_{TS}^{div} t$ . By Remark 3.1.1 and the semantics of  $LTL_{\setminus \circ}$  over paths and states it directly follows that whenever  $t \models \varphi$  then  $s \models \varphi$ . ■

## 3.2 Stutter Simulation Equivalence

So far we have defined the stutter simulation relation  $\leq$  and the divergence-sensitive variant  $\leq_{TS}^{div}$  for pairs of states of a single transition system and for pairs of transition systems. Since both relations are preorders, they are not ensured to be a symmetric, i.e. " $TS_1$  (divergent) stutter simulates  $TS_2$ " does not *necessarily* imply the reverse relation " $TS_2$  (divergent) stutter simulates  $TS_1$ ". The aim of this section is to define the equivalences induced by these preorders, such that we can compare them to the (divergent) stutter bisimulation equivalence. To guarantee symmetry, we first define the *stutter simulation equivalence*  $\cong = \leq \cap \leq^{-1}$ . It is induced by the stutter simulation preorder  $\leq$  and consists of all pairs  $(TS_1, TS_2)$  that can stutter simulate *each other*. The second one is induced by the divergent variant of the stutter simulation preorder and consists of all pairs of transitions systems that divergent stutter simulate each other. It is called *divergence-sensitive stutter simulation equivalence* and will be denoted by  $\cong^{div} = \leq^{div} \cap \leq^{div -1}$ .

**DEFINITION 3.2.1 ( STUTTER SIMULATION EQUIVALENCE )**

Two transition systems  $TS_1$  and  $TS_2$  over the same set of atomic propositions  $AP$  are *stutter simulation equivalent*, denoted  $TS_1 \cong TS_2$ , if  $TS_1 \leq TS_2$  and  $TS_2 \leq TS_1$ . ■

The divergence-sensitive stutter simulation equivalence is defined analogously.

**DEFINITION 3.2.2 ( DIVERGENCE-SENSITIVE STUTTER SIMULATION EQUIVALENCE )**

Two transition systems  $TS_1$  and  $TS_2$  over the same set of atomic propositions  $AP$  are *divergent stutter simulation equivalent*, denoted  $TS_1 \cong^{div} TS_2$ , if  $TS_1 \leq^{div} TS_2$  and  $TS_2 \leq^{div} TS_1$ . ■

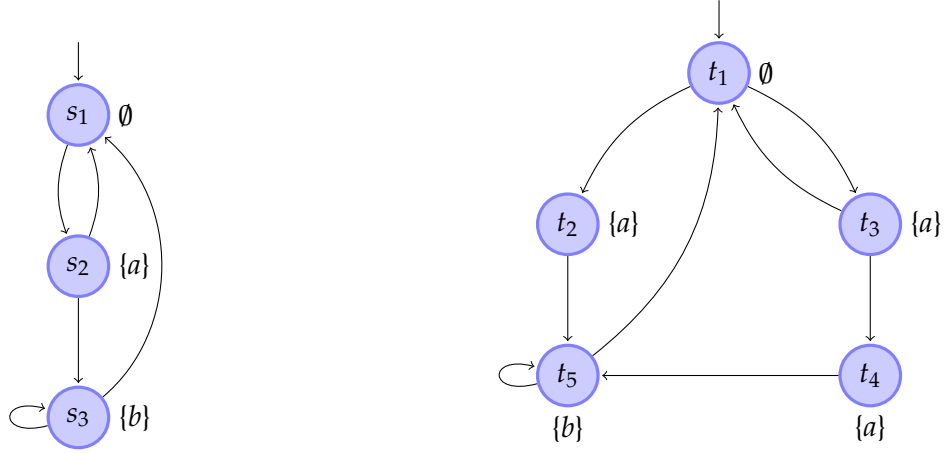


Figure 3.8: (Divergent) Stutter Simulating Transition Systems

EXAMPLE 3.2.1 (EQUIVALENT TRANSITION SYSTEMS UNDER  $\cong$  AND  $\cong^{div}$ )

A proof for the fact that the stutter simulation order is not symmetric in general was given by (Counter-)Example 3.1.1: Since  $TS_1$  does not stutter simulate  $TS_2$ , it follows that  $TS_1 \not\cong TS_2$ .

Observe Figure 3.8. The transition system on the left ( $TS_1$ ) is stutter simulated by the transition system on the right ( $TS_2$ ), i.e. there exists a stutter simulation  $\mathcal{R}_1$  for  $(TS_1, TS_2)$ :

$$\mathcal{R}_1 = \{ (s_1, t_1), (s_2, t_3), (s_2, t_4), (s_3, t_5) \}.$$

Vice versa,  $TS_2$  is stutter simulated by  $TS_1$ , where

$$\mathcal{R}_2 = \{ (t_1, s_1), (t_2, s_2), (t_3, s_2), (t_4, s_2), (t_5, s_3) \}$$

is a stutter simulation for  $(TS_2, TS_1)$ . Thus,  $TS_1 \preceq TS_2$  and  $TS_2 \preceq TS_1$  and hence it follows that  $TS_1$  and  $TS_2$  are stutter simulation equivalent, i.e.  $TS_1 \cong TS_2$ . In fact,  $\mathcal{R}_1$  and  $\mathcal{R}_2$  are also divergence-sensitive, which can be seen as follows. The tuple  $(s_3, t_5)$  is contained in  $\mathcal{R}_1$  and from state  $s_3$  there emanates an infinite path fragment  $\pi_1 = s_0 s_1 s_2 \dots (s_3)^\omega$  with  $(s_i, t_5) \in \mathcal{R}_1$  for all  $i > 0$ . Then, by definition of divergence-sensitive stutter simulation there must exist  $t'_5 \in Post(t_5)$ , such that  $(s_3, t'_5) \in \mathcal{R}_1$ . Since  $t_5 \in Post(t_5)$ , the condition is trivially satisfied. For  $TS_2 \preceq^{div} TS_1$  we have  $(t_5, s_3) \in \mathcal{R}_2$ . The infinite path fragment  $\pi_2 = (t_5)^\omega$  starts in  $t_5$  and never leaves it again. Since  $(t_5, s_3) \in \mathcal{R}_2$  and  $s_3 \in Post(s_3)$ , the condition for divergence sensitivity is again trivially satisfied for this pair of states. Since there exist no other divergent states, we obtain that  $\mathcal{R}_1$  and  $\mathcal{R}_2$  are divergence-sensitive stutter simulations for  $(TS_1, TS_2)$  and  $(TS_2, TS_1)$ , respectively, and thus,  $TS_1 \cong^{div} TS_2$ . ■

### 3.3 Quotient Transition System under $\cong$ and $\cong^{div}$

To either of the invented equivalences we can define and compute the according quotient systems, i.e. abstractions, of some transition system, resulting in a (hopefully) smaller state

space. The main idea is to identify states that exhibit in the sense of (divergent) stutter simulation an equivalent behavior and to merge them to a single state, such that each state of the obtained quotient system represents a set of equivalent states of the old transition system. We will show that this approach leads to abstract models which are guaranteed to be 'correct', i.e. no *important* information gets lost, such that it is ensured that the demanded behavior is preserved.

We first define the quotient system under stutter simulation equivalence  $\approx_{TS}$  for a given transition system (Note that we will omit the subscript  $TS$  for the sake of simplicity).

**DEFINITION 3.3.1 ( QUOTIENT SYSTEM UNDER  $\approx$  )**

For transition system  $TS = (S, Act, \rightarrow, I, AP, L)$  the *stutter simulation quotient* transition system  $TS/\approx$  is defined as follows.

$$TS/\approx = (S/\approx, Act, \rightarrow_{\approx}, I_{\approx}, AP, L_{\approx}),$$

where

- $S/\approx = \{ [s]_{\approx} \mid s \in S \}$ , with  $[s]_{\approx} = \{ s' \in S \mid s \approx s' \}$ ,
- $I_{\approx} = \{ [s]_{\approx} \mid s \in I \}$ ,
- $\rightarrow_{\approx}$  is defined by  $\frac{s \rightarrow s' \wedge s' \not\approx [s]_{\approx}}{[s]_{\approx} \rightarrow_{\approx} [s']_{\approx}}$ ,
- $L_{\approx}([s]_{\approx}) = L(s)$ . ■

If  $s \rightarrow s'$  in  $TS$  and  $s' \not\approx [s]_{\approx}$ , then we need to change the equivalence class from  $[s]_{\approx}$  to  $[s']_{\approx}$ , i.e. introduce a transition from  $[s]_{\approx}$  to  $[s']_{\approx}$  in  $TS/\approx$ . The obtained quotient system therefore does not have any self-loops.

**THEOREM 3.3.1 ( STUTTER SIMULATION EQUIVALENCE OF  $TS$  AND  $TS/\approx$  )**

For any transition system  $TS$ , we have  $TS \approx TS/\approx$ .

**PROOF** Let  $TS$  be a transition system and  $TS/\approx$  its stutter simulation quotient.

(1)  $TS \preceq TS/\approx$ :

Follows from the fact that  $\mathcal{R} = \{ (s, [s]_{\approx}) \mid s \in S \}$  is a stutter simulation for  $(TS, TS/\approx)$ .

(2)  $TS/\approx \preceq TS$ :

Here,  $\mathcal{R}' = \mathcal{R}^{-1} = \{ ([s]_{\approx}, s) \mid s \in S \}$  is not guaranteed to be a stutter simulation for  $TS/\approx \preceq TS$ .  $\mathcal{R}'$  requires that a state  $[s]_{\approx}$  in  $TS/\approx$  can only be stutter simulated by a state  $s$  in  $TS$ , if  $s$  is included in the equivalence class that is represented by  $[s]_{\approx}$ . This difficulty is illustrated in the following under consideration of Figure 3.9. By definition, the pair  $([s_0]_{\approx}, s_0)$  is contained in  $\mathcal{R}'$  and since  $t_0 \approx s_0$  and  $t_0 \rightarrow t_1$  with  $[t_1]_{\approx} \cap [t_0]_{\approx} = \emptyset$ , there must exist a finite path fragment  $\pi_1 = s_0 u_1 \dots u_n s'_0$  in  $TS$  that leads to some state  $s'_0$ , such that  $([s'_0]_{\approx}, s'_0) \in \mathcal{R}'$  with  $[s'_0]_{\approx} = [t_1]_{\approx}$  and each intermediate state  $u_i$  stutter simulates  $[t_0]_{\approx}$ . Clearly,  $([u_i]_{\approx}, u_i) \in \mathcal{R}'$  for all  $1 \leq i \leq n$ . Observe that there exists exactly one state, namely  $s'_1$ , that stutter simulates  $[t_1]_{\approx}$  and which is reachable from  $s_0$  via a path fragment which satisfies the required conditions of stutter simulation. However,

although  $[t_1]_{\approx} \leq s'_1$ , the pair  $([t_0]_{\approx}, s'_1)$  cannot be contained in  $\mathcal{R}'$ , since  $[s'_1]_{\approx} \neq [t_1]_{\approx}$ . Thus, there exists no pair  $([t_1]_{\approx}, t_1) \in \mathcal{R}'$ , such that  $t_1$  is reachable from  $s_0$  via the demanded path fragment and hence,  $\mathcal{R}'$  is *not* a stutter simulation for  $TS/\approx \leq TS$ . The solution to that problem is to allow tuples of the form  $([s]_{\approx}, t)$  in  $\mathcal{R}'$ , where we do not require that  $s = t$ , but simply that  $s \leq t$ , i.e.

$$\mathcal{R}' = \{ ([s]_{\approx}, t) \mid s \leq t \}.$$

Clearly, for each initial state  $[s_0]_{\approx} \in I_{\approx}$  there exists an initial state  $s_0 \in I$ , such that  $[s_0]_{\approx} \leq s_0$ . It remains to prove that for all  $([s]_{\approx}, t) \in \mathcal{R}'$ , it holds that  $[s]_{\approx}$  and  $t$  are equally labeled and  $t$  can mimic every outgoing transition of  $[s]_{\approx}$ :

Let  $([s]_{\approx}, t)$  be some pair in  $\mathcal{R}'$ . Since  $s \leq t$ , we have  $L(s) = L(t)$ . Furthermore, since all states of the same equivalence class are equally labeled, it follows that  $L(s) = L_{\approx}([s]_{\approx})$  and hence,  $L([s]_{\approx}) = L(t)$ .

Let  $B \in S/\approx$  be a successor of  $[s]_{\approx}$  with  $(B, t) \notin \mathcal{R}'$ , i.e.  $B \not\leq t$ . Then  $[s]_{\approx} \rightarrow_{\approx} B$  is a transition in  $TS/\approx$ , i.e. there exists  $s' \in [s]_{\approx}$  with  $s' \rightarrow b$  in  $TS$  and  $[b]_{\approx} = B$ . Note that  $s' \leq s$  and  $s \leq s'$ , since all states of the same equivalence class are stutter simulation equivalent. Recall that  $s \leq t$  and  $s \in [s]_{\approx}$ . By transitivity of  $\leq$ , this implies that  $s' \leq t$  and hence, there exists a finite path fragment  $tu_1 \dots u_n t'$ ,  $n \geq 0$ , such that  $s' \leq u_i$  and  $s \leq u_i$  (since  $s \leq s'$ ),  $i = 1, \dots, n$ , and  $b \leq t'$ . By definition of  $\mathcal{R}'$  this leads to  $([s']_{\approx}, u_i), ([s]_{\approx}, u_i) \in \mathcal{R}'$  (for all  $0 \leq i \leq n$ ) and  $(B, t') = ([b]_{\approx}, t') \in \mathcal{R}'$ . ■

The definition of the quotient system under  $\approx^{div}$  is similar to the one under  $\approx$ . The only but remarkable difference is that a state  $[s]_{\approx^{div}} \in S/\approx^{div}$  has a self-loop if it consists of divergent states to indicate the divergence. (Subscript  $TS$  again is omitted.)

**DEFINITION 3.3.2 ( QUOTIENT SYSTEM UNDER  $\approx^{div}$  )**

For transition system  $TS = (S, Act, \rightarrow, I, AP, L)$  the *divergence-sensitive stutter simulation quotient* transition system  $TS/\approx^{div}$  is defined as follows.

$$TS/\approx^{div} = (S/\approx^{div}, Act, \rightarrow_{\approx^{div}}, I_{\approx^{div}}, AP, L_{\approx^{div}}),$$

where

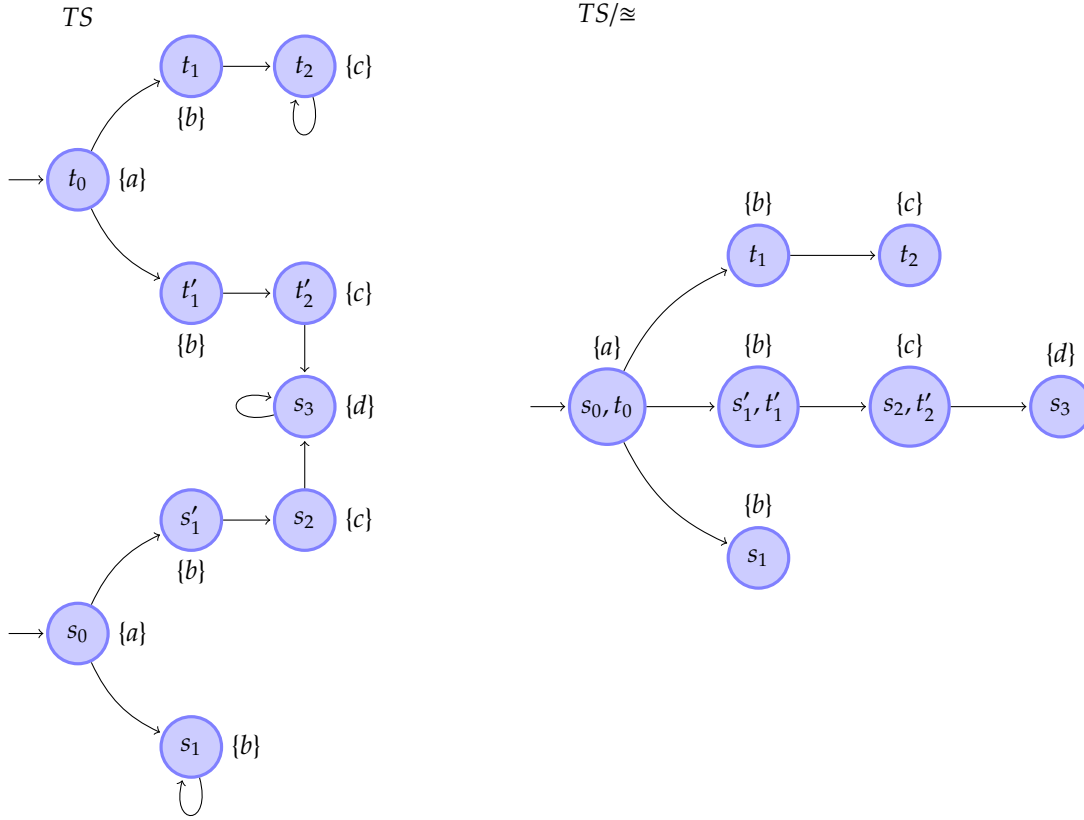
- $S/\approx^{div} = \{ [s]_{\approx^{div}} \mid s \in S \}$ , with  $[s]_{\approx^{div}} = \{ s' \in S \mid s \approx^{div} s' \}$ ,
- $I_{\approx^{div}} = \{ [s]_{\approx^{div}} \mid s \in I \}$ ,
- $\rightarrow_{\approx^{div}}$  is defined by

$$\frac{s \rightarrow s' \wedge s' \not\approx^{div} [s]_{\approx^{div}}}{[s]_{\approx^{div}} \rightarrow_{\approx^{div}} [s']_{\approx^{div}}} \quad \text{and} \quad \frac{s \text{ is } \approx^{div}\text{-divergent}}{[s]_{\approx^{div}} \rightarrow_{\approx^{div}} [s]_{\approx^{div}}}$$

- $L_{\approx^{div}}([s]_{\approx^{div}}) = L(s)$ . ■

**THEOREM 3.3.2 (  $\approx^{div}$ -EQUIVALENCE OF  $TS$  AND  $TS/\approx^{div}$  )**

For any transition system  $TS$ , we have  $TS \approx^{div} TS/\approx^{div}$ .


 Figure 3.9:  $\mathcal{R}' = \{ ([s]_{\approx}, s) \mid s \in S \}$  is not a stutter simulation for  $TS/\approx \leq TS$ 

PROOF The proof is analogous to the proof in Theorem 3.3.1. Note that for case (2) (when showing that if  $([s]_{\approx}^{div}, t) \in \mathcal{R}'$  then  $t$  divergent stutter simulates  $[s]_{\approx}^{div}$ ) we do not have to distinguish between the cases  $B = [s]_{\approx}^{div}$  and  $B \neq [s]_{\approx}^{div}$ , i.e. whether  $[s]_{\approx}^{div}$  has a self-loop or not, since it is not possible that  $B = [s]_{\approx}^{div}$  by the assumption  $[s]_{\approx}^{div} \leq^{div} t$  and  $B \not\leq^{div} t$ . Hence, as in the case for the quotient system under  $\approx$ , we simply have to examine the direct successors  $B$  of each state in  $TS/\approx^{div}$  with  $B \neq [s]_{\approx}^{div}$ . ■

Observe that divergence-sensitive stutter simulation equivalence is a special case of stutter simulation equivalence and thus, stutter simulation equivalence with divergence  $\approx^{div}$  is strictly finer than stutter simulation equivalence  $\approx$ .

#### EXAMPLE 3.3.1 ( QUOTIENT SYSTEM UNDER $\approx^{div}$ )

The transition system presented in Figure 3.10 is the quotient system under  $\approx^{div}$  of transition system  $TS$  in Figure 3.9. Note that  $t_0$  and  $s_0$  are equivalent under  $\approx$  but not under  $\approx^{div}$ . Furthermore, we have that  $s_0 \not\leq^{div} t_0$  and  $t_0 \not\leq^{div} s_0$ . This can be seen as follows. From  $s_1$  there emanates an infinite path fragment  $\pi_1 = u_0 u_1 u_2 \dots = (s_1)^\omega$ , where  $s_1$  is stutter simulated by  $t_1$ , but there exists no  $t'_1 \in Post(t_1)$ , such that  $s_1$  is divergent stutter simulated by  $t'_1$  (this also holds for state  $s'_1, t'_1$ ). Thus,  $s_1 \not\leq^{div} t_1$  (and  $s_1 \not\leq^{div} s'_1, t'_1$ ) and it follows that  $s_0 \not\leq^{div} t_0$ . With the same argumentation we obtain that there exists no finite path fragment satisfying the conditions

for divergence-sensitive stutter simulation that leads from  $s_0$  to a state that divergent stutter simulates  $t_2$  and thus,  $t_0 \not\approx^{div} s_0$ . This example also shows that  $\approx^{div}$  is strictly finer than  $\approx$ , since stutter simulation with divergence yields eight equivalence classes, whereas there are only seven obtained by the standard variant.

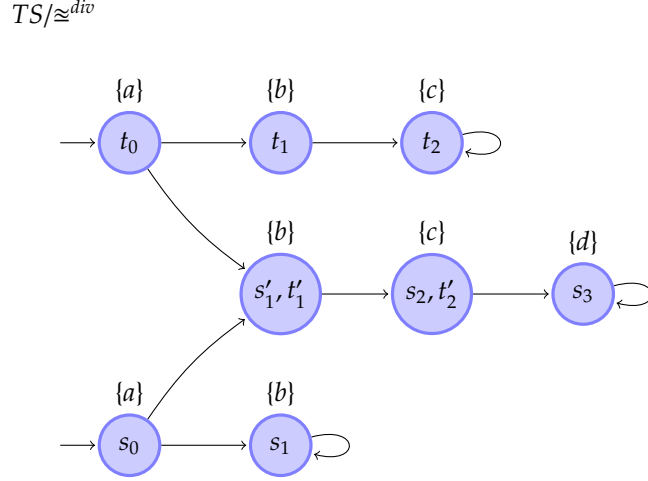


Figure 3.10: Quotient System under  $\approx^{div}$

### 3.4 Stutter Simulation Equivalence vs. Stutter Bisimulation Equivalence

In this section we will compare the stutter simulation equivalence to the stutter bisimulation equivalence. Furthermore, the divergence-sensitive stutter bisimulation will be introduced and compared to the stutter simulation equivalence and its divergence-sensitive variant. The following definitions are gathered from Baier and Katoen [1].

**DEFINITION 3.4.1 ( STUTTER BISIMULATION )**

Let  $TS = (S, Act, \rightarrow, I, AP, L)$  be a transition system over  $AP$ . A *stutter bisimulation* for  $TS$  is a binary relation  $\mathcal{R} \subseteq S \times S$  such that for all  $(s_1, s_2) \in \mathcal{R}$ :

- (1)  $L(s_1) = L(s_2)$ .
- (2) If  $s'_1 \in Post(s_1)$  with  $(s'_1, s_2) \notin \mathcal{R}$ , then there exists a finite path fragment  $s_2 u_1 \dots u_n s'_2$ ,  $n \geq 0$ , with  $(s_1, u_i) \in \mathcal{R}$  for all  $i \in \{1, \dots, n\}$  and  $(s'_1, s'_2) \in \mathcal{R}$ .
- (3) If  $s'_2 \in Post(s_2)$  with  $(s_1, s'_2) \notin \mathcal{R}$ , then there exists a finite path fragment  $s_1 v_1 \dots v_m s'_1$ ,  $m \geq 0$ , with  $(v_i, s_2) \in \mathcal{R}$  for all  $i \in \{1, \dots, m\}$  and  $(s'_1, s'_2) \in \mathcal{R}$ .

$s_1, s_2$  are *stutter bisimulation equivalent* (stutter-bisimilar, for short), denoted  $s_1 \approx_{TS} s_2$ , if there exists a stutter simulation  $\mathcal{R}$  for  $TS$  with  $(s_1, s_2) \in \mathcal{R}$ .

Conditions (1) and (2) are equal to the conditions for stutter simulation (see Definition 3.1.1), whereas the third one simply requires the symmetric counterpart of condition (2), such that whenever  $(s_1, s_2) \in \mathcal{R}$  then  $s_1$  stutter mimics the stepwise behavior of  $s_2$ , and vice versa. It is not difficult to show that  $\approx_{TS}$  is an equivalence on  $S$  and the coarsest stutter bisimulation for  $TS$ . It is obtained by the union of all stutter bisimulations.

As for stutter simulation equivalence this definition can be adapted to one for pairs of transition systems  $TS_1, TS_2$ :

**DEFINITION 3.4.2 ( STUTTER BISIMULATION EQUIVALENCE )**

Let  $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP, L_i)$ ,  $i = 1, 2$ , be two transition systems over the same set of atomic propositions  $AP$ .  $TS_1$  and  $TS_2$  are bisimulation equivalent, denoted  $TS_1 \approx TS_2$ , if there exists a stutter bisimulation  $\mathcal{R}$  on  $S_1 \times S_2$  such that

$$\forall s_1 \in I_1. (\exists s_2 \in I_2. (s_1, s_2) \in \mathcal{R}) \text{ and } \forall s_2 \in I_2. (\exists s_1 \in I_1. (s_1, s_2) \in \mathcal{R}). \quad \blacksquare$$

Note that stutter bisimulation requires that whenever  $s_1 \approx s_2$  and there is a transition  $s_1 \rightarrow s'_1$  with  $s'_1 \not\approx s_2$ , then there exists a finite path fragment  $s_2 u_1 \dots u_n s'_2$ ,  $n \geq 0$ , such that  $s_1 \approx u_i$  and  $s'_1 \approx s'_2$ . In contrast to that, stutter simulation equivalence requires less strict conditions, i.e. whenever  $s_1 \approx s_2$ , then path fragment  $s_1 s'_1$  with  $s'_1 \not\approx s_2$  can be stutter mimicked by  $s_2$  via a finite path fragment  $s_2 v_1 \dots v_m s'_2$ ,  $m \geq 0$ , where  $s_1 \approx v_i$  and  $s'_1 \approx s'_2$ . Since the reverse relation, i.e.  $u_i \approx s_1$  and  $s'_2 \approx s'_1$ , is not mandatory, we might obtain that  $s_1 \not\approx u_i$  or  $s'_1 \not\approx s'_2$ . Thus, stutter bisimulation equivalence requires at most more than stutter simulation equivalence and hence, if the conditions for stutter bisimilar states are fulfilled, then those for stutter simulation equivalent ones are satisfied anyway. It can be concluded that  $TS_1 \approx TS_2$  implies  $TS_1 \cong TS_2$ , whereas the reverse is not necessarily true (see Example 3.4.1). Furthermore, stutter bisimulation equivalence yields in general a finer abstraction, i.e. greater state space of the quotient system, than stutter simulation equivalence. This is caused by the following logical conclusion: If more conditions are imposed on the pairs of states, then at most less pairs will eventually be able to satisfy them, which results in more equivalence classes at the utmost and hence, less states that can be merged.

**EXAMPLE 3.4.1 (  $\cong$  DOES NOT IMPLY  $\approx$  )**

Transition systems  $TS_1$  (left) and  $TS_2$  (right) in Figure 3.11 are stutter simulation equivalent but not stutter bisimulation equivalent. This can be seen as follows:

There exist stutter simulations  $\mathcal{R}_1$  for  $(TS_1, TS_2)$  and  $\mathcal{R}_2$  for  $(TS_2, TS_1)$ , formally:

$$\begin{aligned} \mathcal{R}_1 &= \{ (s_0, t_0), (s_1, t_1), (s_1, t_2), (s_2, t_3), (s_3, t_5) \}, \\ \mathcal{R}_2 &= \{ (t_0, s_0), (t_1, s_1), (t_2, s_1), (t_3, s_2), (t_4, s_3), (t_4, s_1) \}. \end{aligned}$$

Thus,  $TS_1 \cong TS_2$ .

Assume that  $TS_1 \approx TS_2$ . Then,  $s_0$  must be stutter bisimilar to  $t_0$ , since these are the unique initial states of  $TS_1$  and  $TS_2$ , respectively. State  $t_0$  has an outgoing transition that leads to  $t_4$ , where  $s_0 \not\approx t_4$ . Thus, there must exist a finite path fragment  $s_0 u_1 \dots u_n s'_0$  in  $TS_1$ ,  $n \geq 0$ , such that  $t_0 \approx u_i$ ,  $i = 1, \dots, n$ , and  $t_4 \approx s'_0$ . State  $s_0$  has exactly one successor  $s_1$  that is equally labeled as  $t_4$ . However,  $s_1$  and  $t_4$  are not stutter bisimilar, since  $s_1$  has an outgoing transition to state  $s_2$  that is labeled with  $a$ , whereas  $t_4$  cannot reach a state that is equally labeled as  $s_2$  by visiting

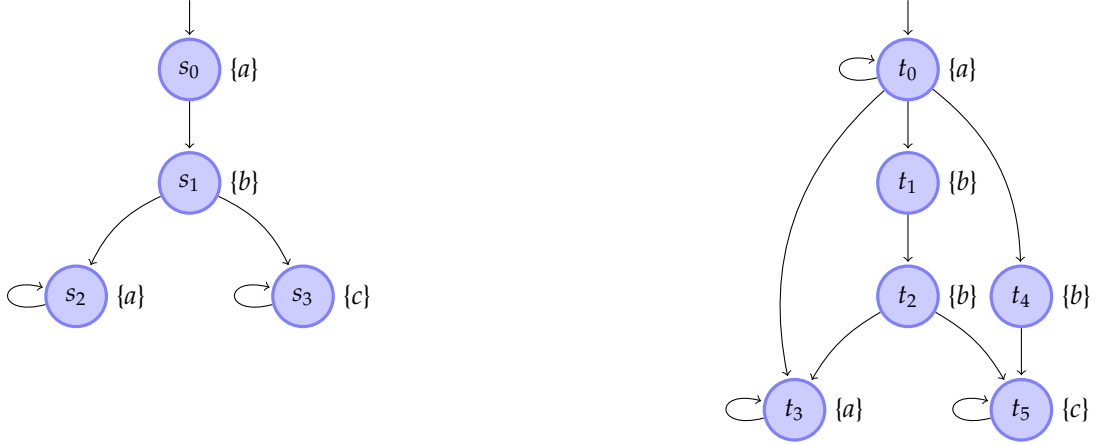


Figure 3.11: Stutter Simulation but not Stutter Bisimulation Equivalent Transition Systems

only states that are stutter bisimilar to  $s_1$  and hence, there exists no finite path fragment from  $s_0$  to  $s'_0$  that satisfies the required conditions. Thus,  $s_0$  cannot be stutter bisimilar to  $t_0$ , which violates condition (A) of Definition 3.4.2 and it follows that  $TS_1 \not\approx TS_2$ . ■

The following Lemma 3.4.1 reconsiders the correlation of stutter simulation and bisimulation equivalence.

**LEMMA 3.4.1 ( STUTTER BISIMULATION  $\approx$  VS. STUTTER SIMULATION EQUIVALENCE  $\cong$  )**

Let  $TS_1, TS_2$  be two transition systems. Then:

$TS_1 \approx TS_2$  iff there exists a stutter simulation  $\mathcal{R}_{1,2}$  for  $(TS_1, TS_2)$ , such that  $\mathcal{R}_{2,1} = \{ (s_2, s_1) \mid (s_1, s_2) \in \mathcal{R}_{1,2} \}$  is a stutter simulation for  $(TS_2, TS_1)$ .

**PROOF** Let  $TS_1, TS_2$  be two transition systems.

$\Rightarrow$ : Assume  $\mathcal{R}$  is a stutter bisimulation for  $(TS_1, TS_2)$  and let  $\mathcal{R}_{1,2} = \mathcal{R}$  and  $\mathcal{R}_{2,1} = \mathcal{R}^{-1}$ . Then,  $\mathcal{R}_{1,2}$  is a stutter simulation for  $(TS_1, TS_2)$ , which can be seen as follows. Since  $\mathcal{R}$  is a stutter bisimulation for  $(TS_1, TS_2)$ , it trivially follows that the initial condition (A) for stutter simulating transition systems is satisfied. Furthermore, the labeling condition (B.1) is fulfilled for each pair in  $\mathcal{R}_{1,2}$ . Let  $(s_1, s_2) \in \mathcal{R}_{1,2}$ . By definition of stutter bisimulation, we have that whenever there is some  $s'_1 \in \text{Post}(s_1)$  with  $(s'_1, s_2) \notin \mathcal{R}$  then there exists a finite path fragment  $s_2 u_1 u_2 \dots u_n s'_2$ ,  $n \geq 0$ , where  $(s_1, u_i) \in \mathcal{R}$  for all  $i = 1, \dots, n$  and  $(s'_1, s'_2) \in \mathcal{R}$ . Consequently,  $(s_1, u_i) \in \mathcal{R}_{1,2}$  and  $(s'_1, s'_2) \in \mathcal{R}_{1,2}$ , which satisfies condition (B.2) for stutter simulation and thus,  $\mathcal{R}_{1,2}$  is a stutter simulation for  $(TS_1, TS_2)$ .

As a consequence of the fact that  $\mathcal{R}$  is an equivalence relation, we obtain that its symmetric counterpart  $\mathcal{R}^{-1}$  is a stutter bisimulation for  $(TS_2, TS_1)$ , and hence, by applying the same argumentation as before, it follows that  $\mathcal{R}_{2,1}$  is a stutter simulation for  $(TS_2, TS_1)$ .

$\Leftarrow$ : Let  $\mathcal{R}_{1,2}$  and  $\mathcal{R}_{2,1} := \{ (s_2, s_1) \mid (s_1, s_2) \in \mathcal{R}_{1,2} \}$  be stutter simulations for  $(TS_1, TS_2)$  and  $(TS_2, TS_1)$ , respectively. By definition, for all  $s_1 \in I_1$  there exists  $s_2 \in I_2$  with  $s_1 \preceq s_2$ .

The symmetric counterpart also holds, since  $\mathcal{R}_{2,1}$  is a stutter simulation for  $(TS_2, TS_1)$ . Construct  $TS_1 \oplus TS_2 =: TS$  and recall that if  $s_1 \leq s_2$  then  $s_1 \leq_{TS} s_2$ . Thus,  $\mathcal{R}_{\oplus} := \mathcal{R}_{1,2} \cup \mathcal{R}_{2,1}$  is a stutter simulation for  $(TS, TS)$ . Note that  $\mathcal{R}_{\oplus}$  is a symmetric relation, since  $\mathcal{R}_{1,2}^{-1} = \mathcal{R}_{2,1}$  and thus, it immediately follows that  $\mathcal{R} := \{ (s_1, s_2) \mid (s_1, s_2) \in \mathcal{R}_{1,2} \}$  is a stutter bisimulation for  $(TS_1, TS_2)$ . Hence,  $TS_1 \approx TS_2$ . ■

Similar to the standard variant of stutter simulation, there are no restrictions imposed on divergent paths under stutter bisimulation equivalence. In the stutter bisimulation sense a path is divergent, if it stays forever in the same equivalence class without performing any visible step. This is (again) problematic if language preservation is considered [1]. In the standard variant of the temporal logic  $CTL^*$ , path formulae are interpreted over infinite paths, such that the underlying transition system is usually assumed to have no terminal states (otherwise a trap state is introduced). Emerson and Srinivasan [4] proposed a variant of the temporal logic  $CTL^*$  that interprets the path formulae over both finite and infinite paths of a given Kripke structure. Based on this interpretation, De Nicola and Vaandrager [3] proved that  $CTL^*_{\bigcirc}$  equivalence, i.e. equivalence with respect to  $CTL^*$  formulae that do not contain the next-step operator  $\bigcirc$ , coincides with the stutter bisimulation equivalence of Definition 3.4.1.

As far as the standard interpretation is considered, we have to focus on the divergence-sensitive stutter bisimulation. Here, states can only be related if they either both exhibit divergent paths or none of them.

**DEFINITION 3.4.3 (  $\mathcal{R}$ -DIVERGENCE, DIVERGENCE SENSITIVITY )**

Let  $TS = (S, Act, \rightarrow, I, AP, L)$  be a transition system over  $AP$  and  $\mathcal{R}$  an equivalence relation on  $S$ .

- $s \in S$  is  $\mathcal{R}$ -divergent if there exists an infinite path fragment  $\pi = ss_1s_2\dots \in Paths(s)$  such that  $(s, s_j) \in \mathcal{R}$  for all  $j > 0$ .
- $\mathcal{R}$  is divergence-sensitive if for any  $(s_1, s_2) \in \mathcal{R}$ : if  $s_1$  is  $\mathcal{R}$ -divergent, then  $s_2$  is  $\mathcal{R}$ -divergent. ■

Roughly speaking, if there emanates an infinite path fragment from  $s$ , where only states of the equivalence class  $[s]_{\sim}$  are visited, then  $s$  is  $\mathcal{R}$ -divergent. Furthermore, if the states in an equivalence class are either all  $\mathcal{R}$ -divergent or none of them is  $\mathcal{R}$ -divergent, then  $\mathcal{R}$  is divergence-sensitive, which leads to the following definition of divergence-sensitive stutter bisimulation.

**DEFINITION 3.4.4 ( DIVERGENCE-SENSITIVE STUTTER BISIMULATION )**

States  $s_1, s_2$  in transition system  $TS$  are *divergent sensitive stutter bisimilar*, denoted  $s_1 \approx_{TS}^{div} s_2$ , if there exists a divergence-sensitive stutter bisimulation  $\mathcal{R}$  on  $TS$ , such that  $(s_1, s_2) \in \mathcal{R}$ . ■

Note that  $\approx_{TS}^{div}$  is an equivalence relation on  $S$  and the coarsest divergence-sensitive stutter bisimulation. However, we will omit the technical details of the proof.

Clearly, this definition can be adapted to one for pairs of transition systems. We have  $TS_1 \approx_{TS_1 \oplus TS_2}^{div} TS_2$  if and only if each initial state of  $TS_1$  is divergence stutter bisimilar (according to  $\approx_{TS_1 \oplus TS_2}^{div}$ ) to an initial state in  $TS_2$ , and vice versa [1].

## EXAMPLE 3.4.2 ( DIVERGENCE-SENSITIVE STUTTER BISIMILAR TRANSITION SYSTEMS )

In Figure 3.12 we have three transition systems  $TS_1$  (left),  $TS_2$  (right) and  $TS_3$  (center), where  $TS_1 \not\approx^{div} TS_2$ . This can be seen as follows. Construct  $TS = TS_1 \oplus TS_2$ . Observe that there exists an infinite path fragment  $\pi_1 = (s_0)^\omega$  starting and staying forever in  $s_0$ . Since each state is divergent stutter bisimilar to itself, we have that  $s_0$  is divergent stutter bisimilar to each state on  $\pi_1$  and thus,  $s_0$  is  $\approx_{TS}^{div}$ -divergent. However, there exists no such path fragment emanating from  $t_0$ . Then, by definition,  $s_0$  and  $t_0$  cannot be in the same equivalence class and thus,  $s_0 \not\approx_{TS}^{div} t_0$ . Since  $s_0$  and  $t_0$  are the unique initial states of  $TS_1$  and  $TS_2$ , respectively, it follows immediately that  $TS_1 \not\approx^{div} TS_2$ .

In contrast to that, there exists a divergence-sensitive stutter bisimulation  $\mathcal{R}_{1,3}$  on  $TS' = TS_1 \oplus TS_3$ , such that  $(s_i, u_j) \in \mathcal{R}_{1,3}$  for all  $s_i \in I_1, u_j \in I_3$ :

$$\mathcal{R}_{1,3} = \{ (s_0, s_0), (u_0, u_0), (u_0, u_1), (s_0, u_0), (s_0, u_1), (s_1, s_1), (u_2, u_2), (s_1, u_2) \}.$$

The infinite path fragment  $\pi_2 = (u_0 u_1)^\omega$  starts in the unique initial state  $u_0$  of  $TS_3$ . Clearly,  $u_0 \approx_{TS'}^{div} u_0$  and  $u_0 \approx_{TS'}^{div} u_1$ , such that  $(u_0, u_0), (u_0, u_1) \in \mathcal{R}_{1,3}$ . Then, state  $u_0$  is divergent stutter bisimilar to each state on  $\pi_2$  and thus,  $u_0$  is  $\mathcal{R}_{1,3}$ -divergent. State  $s_1$  is divergent stutter bisimilar to state  $u_2$ , i.e.  $(s_1, u_2) \in \mathcal{R}_{1,3}$ , since both are  $\mathcal{R}_{1,3}$ -divergent and never reach a state that belongs to a different equivalence class. Note that state  $s_1$  is the only successor of  $s_0$  with  $s_0 \not\approx_{TS'}^{div} s_1$ . For  $u_0$  there also exists exactly one such successor, namely  $u_2$ . Then, since  $s_0$  and  $u_0$  are  $\mathcal{R}_{1,3}$ -divergent and  $s_1 \approx_{TS'}^{div} u_2$ , it follows that  $s_0$  is divergent stutter bisimilar to  $u_0$ . We already know that  $u_0 \approx_{TS'}^{div} u_1$  and since  $\approx_{TS'}^{div}$  is an equivalence relation (for any  $TS$ ), we obtain by transitivity that  $s_0 \approx_{TS'}^{div} u_1$ . Thus, for each initial state in  $TS_1$  and  $TS_3$  there exists a divergent stutter bisimilar initial state in  $TS_3$  and  $TS_1$ , respectively, and hence,  $TS_1 \approx^{div} TS_3$ . This, however, leads directly to the fact that  $TS_2 \not\approx^{div} TS_3$ : Recall that there

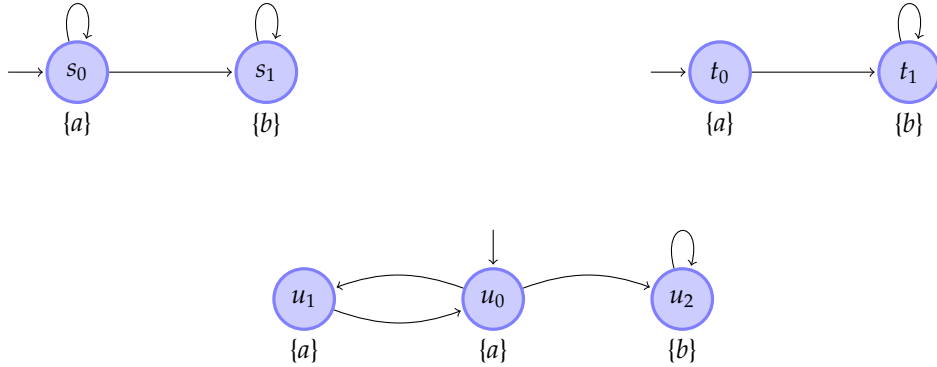


Figure 3.12: Divergence-Sensitive Stutter Bisimulation

exists no divergence-sensitive stutter bisimulation  $\mathcal{R}_{1,2}$  on  $TS_1 \oplus TS_2$  with  $(s_0, t_0) \in \mathcal{R}_{1,2}$ . We thus obtain by transitivity of  $\approx_{TS}^{div}$  that there cannot exist a divergence sensitive stutter bisimulation  $\mathcal{R}_{2,3}$  on  $TS_2 \oplus TS_3$  with  $(t_0, u_0) \in \mathcal{R}_{2,3}$ . ■

Note that  $\approx_{TS}^{div}$  is an equivalence relation on  $S$ , which is strictly finer than stutter bisimulation [1]. By applying the same logical argumentation as for the relation between stutter

simulation and stutter bisimulation, it can be stated that  $\approx_{TS}^{div}$  implies  $\approx_{TS}$ . Clearly, since stutter simulation does not imply stutter bisimulation equivalence, it cannot imply its stricter divergence-sensitive variant. We will illustrate this fact by the following Example 3.4.3.

EXAMPLE 3.4.3 ( $\cong$  AND  $\approx$  DO NOT IMPLY  $\approx^{div}$ )

Consider Figure 3.12. In Example 3.4.2 we have already shown that  $TS_1 \not\approx^{div} TS_2$  and  $TS_2 \not\approx^{div} TS_3$ . In contrast to that, there exists a stutter bisimulation  $\mathcal{R}_{1,2} = \{ (s_0, t_0), (s_1, t_1) \}$  containing the pair  $(s_0, t_0)$  and thus,  $TS_1 \approx TS_2$ . Since  $\approx$  is an equivalence relation and since  $TS_1 \approx^{div} TS_3$ , it follows immediately that  $TS_2 \approx TS_3$ .

Recall that the demanded requirements of stutter simulation equivalence are less strict than those of divergence-sensitive stutter bisimulation. Caused by the fact that  $TS_1$  and  $TS_3$  are divergent stutter bisimilar,  $TS_1$  stutter simulates  $TS_3$ , and vice versa, i.e.  $TS_1 \cong TS_3$ .

However, since  $TS_1 \approx TS_2$ , there exist stutter simulations  $\mathcal{R}_{1,2}$  on  $(TS_1, TS_2)$  and  $\mathcal{R}_{2,1}$  on  $(TS_2, TS_1)$ , formally

$$\mathcal{R}_{1,2} = \{ (s_0, t_0), (s_1, t_1) \} \text{ and } \mathcal{R}_{2,1} = \{ (t_0, s_0), (t_1, s_1) \},$$

such that  $TS_1 \leq TS_2$  and  $TS_2 \leq TS_1$  and consequently,  $TS_1 \cong TS_2$ . Furthermore, since  $TS_2$  is stutter bisimilar to  $TS_3$  and since  $\cong$  is an equivalence relation, we obtain directly that  $TS_2 \cong TS_3$ . It follows that neither stutter bisimulation nor stutter simulation implies divergence-sensitive stutter bisimulation. ■

The divergence-sensitive variant of stutter bisimulation requires that if two states  $s_1, s_2$  are in the same equivalence class then they are either both divergent or none of them. If they are divergent, then all states on these divergent paths are divergent stutter bisimilar. This is not the case for stutter simulation equivalence with divergence. If  $s_1$  is divergent stutter simulation equivalent to  $s_2$  and  $s_1$  is divergent, then there emanates an infinite path fragment  $\pi_1 = s_{1,0}s_{1,1}s_{1,2}\dots$  from  $s_1 = s_{1,0}$  and  $\pi_2 = s_{2,0}s_{2,1}s_{2,2}\dots$ , such that each state on  $\pi_2$  divergent stutter simulates some state on  $\pi_1$ . However, if  $s_{1,i} \leq^{div} s_{2,j}$  then the reverse is not necessarily true. Thus, the conditions required by divergent stutter bisimulation are stronger than those imposed by divergent stutter simulation equivalence and it cannot be expected that  $\cong^{div}$  implies  $\approx^{div}$ . However, with an analogous argumentation as for stutter bisimulation, we can conclude that divergence-sensitive stutter bisimulation implies divergence-sensitive stutter simulation equivalence.

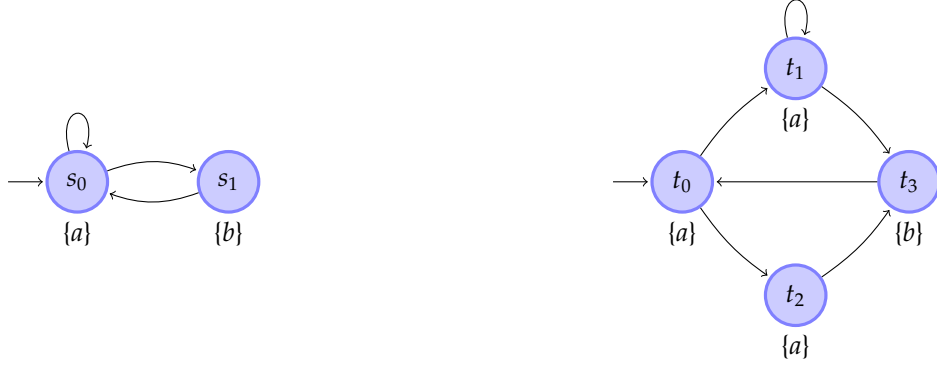
EXAMPLE 3.4.4 ( $\cong^{div}$  DOES NOT IMPLY  $\approx^{div}$ )

Observe Figure 3.13. The transition system on the left is denoted by  $TS_1$ , the one on the right by  $TS_2$ . It can easily be seen that  $TS_1 \leq^{div} TS_2$  and vice versa, since there exist the following divergence-sensitive stutter simulations  $\mathcal{R}_1$  and  $\mathcal{R}_2$  for  $(TS_1, TS_2)$  and  $(TS_2, TS_1)$ , respectively.

$$\begin{aligned} \mathcal{R}_1 &= \{ (s_0, t_0), (s_0, t_1), (s_1, t_3) \}, \\ \mathcal{R}_2 &= \{ (t_0, s_0), (t_1, s_0), (t_2, s_0), (t_3, s_1) \}. \end{aligned}$$

It follows that  $TS_1 \cong^{div} TS_2$ .

However, there exists no divergence-sensitive stutter bisimulation for  $(TS_1, TS_2)$ , since there exists no state in  $TS_1$  that is divergent stutter bisimilar to  $t_2$  in  $TS_2$ . Thus,  $TS_1 \not\approx^{div} TS_2$ . ■


 Figure 3.13:  $TS_1 \approx^{div} TS_2$  but  $TS_1 \not\approx^{div} TS_2$ 

Analogical to Lemma 3.4.1, the following Lemma 3.4.2 reconsiders the correlation of divergence-sensitive stutter simulation and bisimulation equivalence.

**LEMMA 3.4.2 ( STUTTER BISIMULATION  $\approx^{div}$  VS. STUTTER SIMULATION EQUIVALENCE  $\cong^{div}$  )**  
 Let  $TS_1, TS_2$  be two transition systems. Then:

$TS_1 \approx^{div} TS_2$  iff there exists a divergence-sensitive stutter simulation  $\mathcal{R}_{1,2}$  for  $(TS_1, TS_2)$ , such that  $\mathcal{R}_{2,1} = \{ (s_2, s_1) \mid (s_1, s_2) \in \mathcal{R}_{1,2} \}$  is a divergence-sensitive stutter simulation for  $(TS_2, TS_1)$ .

**PROOF** Let  $TS_1, TS_2$  be two transition systems.

$\Rightarrow$ : Let  $\mathcal{R}$  be a divergence-sensitive stutter bisimulation for  $(TS_1, TS_2)$ . The proof is similar to the one for Lemma 3.4.1. It remains to examine the case when  $(s_1, s_2) \in \mathcal{R}$  and  $s_1, s_2$  are  $\mathcal{R}$ -divergent. Then, from  $s_1$  there emanates an infinite path fragment  $s_1 = s_{1,0}s_{1,1}s_{1,2}\dots$ , such that  $(s_{1,i}, s_2) \in \mathcal{R}$  for all  $i \in \mathbb{N}$  and in  $s_2$  there starts an infinite path fragment  $s_2 = s_{2,0}s_{2,1}s_{2,2}\dots$ , where  $(s_1, s_{2,j}) \in \mathcal{R}$  for all  $j \in \mathbb{N}$ . Since  $\mathcal{R}$  is an equivalence relation, it follows that  $(s_{1,i}, s_{2,j}) \in \mathcal{R}$  for all  $i, j \in \mathbb{N}$ . Thus, if  $s_1$  is  $\mathcal{R}$ -divergent then there exists  $s'_2 \in Post(s_2)$  with  $(s_{1,k}, s'_2) \in \mathcal{R}$  for some  $k > 0$  and vice versa. Then,  $\mathcal{R}_{1,2} := \{ (s_1, s_2) \mid (s_1, s_2) \in \mathcal{R} \}$  and  $\mathcal{R}_{2,1} := \{ (s_2, s_1) \mid (s_1, s_2) \in \mathcal{R} \}$  are divergence-sensitive stutter simulations for  $(TS_1, TS_2)$  and  $(TS_2, TS_1)$ , respectively.

$\Leftarrow$ : Let  $\mathcal{R}_{1,2}, \mathcal{R}_{2,1}$  be divergence-sensitive stutter simulations for  $(TS_1, TS_2)$  and  $(TS_2, TS_1)$ , respectively, where  $\mathcal{R}_{2,1} := \{ (s_2, s_1) \mid (s_1, s_2) \in \mathcal{R}_{1,2} \}$ . Again the proof is similar to the one for Lemma 3.4.1 and we basically focus on the case when  $(s_1, s_2) \in \mathcal{R}_{1,2}$  and  $s_1$  is  $\mathcal{R}_{1,2}$ -divergent. Then from  $s_1$  there emanates an infinite path fragment  $s_1 = s_{1,0}s_{1,1}s_{1,2}\dots$ , where  $(s_{1,i}, s_2) \in \mathcal{R}_{1,2}$  and  $(s_2, s_{1,i}) \in \mathcal{R}_{2,1}$  for all  $i \geq 0$ . Furthermore, there exists some  $j > 0$  and  $s'_2 \in Post(s_2)$ , such that  $(s_{1,j}, s'_2) \in \mathcal{R}_{1,2}$  and  $(s'_2, s_{1,j}) \in \mathcal{R}_{2,1}$ . Since for each element  $(s_1, s_2) \in \mathcal{R}_{1,2}$  its symmetric pair is included in  $\mathcal{R}_{2,1}$ , it holds that  $s_1 \approx^{div} s_2$ . Hence, since  $\approx^{div}$  is an equivalence relation, it follows that  $s_{1,i} \approx^{div} s_2$  and  $s_{1,0} \approx^{div} s_{1,i}$  (for all  $i$ ) and since  $s_{1,j} \approx^{div} s'_2$ , we can conclude that  $s_2 \approx^{div} s'_2$  and  $s_{1,i} \approx^{div} s'_2$ . As a consequence, there exists an infinite path fragment  $s_{2,0}s_{2,1}s_{2,2}\dots$  with  $s_2 = s_{2,0}, s'_2 = s_{2,1}$

and  $s_2 \cong^{div} s_{2,k}$  for all  $k \in \mathbb{N}$ . Then,  $s_1$  and  $s_2$  are both divergent ( $\star$ ) and we can construct a divergence-sensitive stutter bisimulation  $\mathcal{R}$  for  $(TS_1, TS_2)$  as follows:

- (1) Insert all pairs  $(s_1, s_2) \in \mathcal{R}_{1,2}$  into  $\mathcal{R}$ .
- (2) If  $s_1$  is  $\mathcal{R}_{1,2}$ -divergent then there exist infinite path fragments  $s_1 = s_{1,0}s_{1,1}s_{1,2}\dots$  and  $s_2 = s_{2,0}s_{2,1}s_{2,2}\dots$ , such that all states on these paths are divergent stutter simulation equivalent. Then,  $s_1$  and  $s_2$  are  $\mathcal{R}$ -divergent and we have to insert all pairs of states on these paths that are of the following form:  $(s_{1,0}, s_{1,i+1})$ ,  $(s_{1,i}, s_{2,k})$ ,  $(s_{2,0}, s_{2,k+1})$  for all  $i, k \in \mathbb{N}$ .

The constructed binary relation  $\mathcal{R}$  is a stutter bisimulation for  $(TS_1, TS_2)$ , since **for all** pairs  $(s_1, s_2) \in \mathcal{R}$  the following statement holds: If there is some  $s'_1 \in Post(s_1)$  with  $(s'_1, s_2) \notin \mathcal{R}$  then there exists an infinite path fragment  $s_2u_1\dots u_n s'_2$ ,  $n \geq 0$ , where  $(s_1, u_i) \in \mathcal{R}$  for all  $i = 1, \dots, n$  (since  $s_1 \leq^{div} s_2$ ) and vice versa (since  $s_2 \leq^{div} s_1$ ). The labeling- and initial conditions for stutter bisimulation are trivially satisfied by definition of  $\cong^{div}$  and since  $\mathcal{R}_{1,2}$  and  $\mathcal{R}_{2,1}$  are divergence-sensitive stutter simulations and  $\mathcal{R}_{1,2} \subseteq \mathcal{R}$ . It remains to check whether  $\mathcal{R}$  is divergence-sensitive. By construction of  $\mathcal{R}$  (step (2)) and according to ( $\star$ ), we have that whenever  $s_1$  is  $\mathcal{R}$ -divergent, then  $s_2$  is  $\mathcal{R}$ -divergent for all pairs  $(s_1, s_2) \in \mathcal{R}$ . ■

Figure 3.14 provides an overview of the different notations of the so far introduced implementation relations.

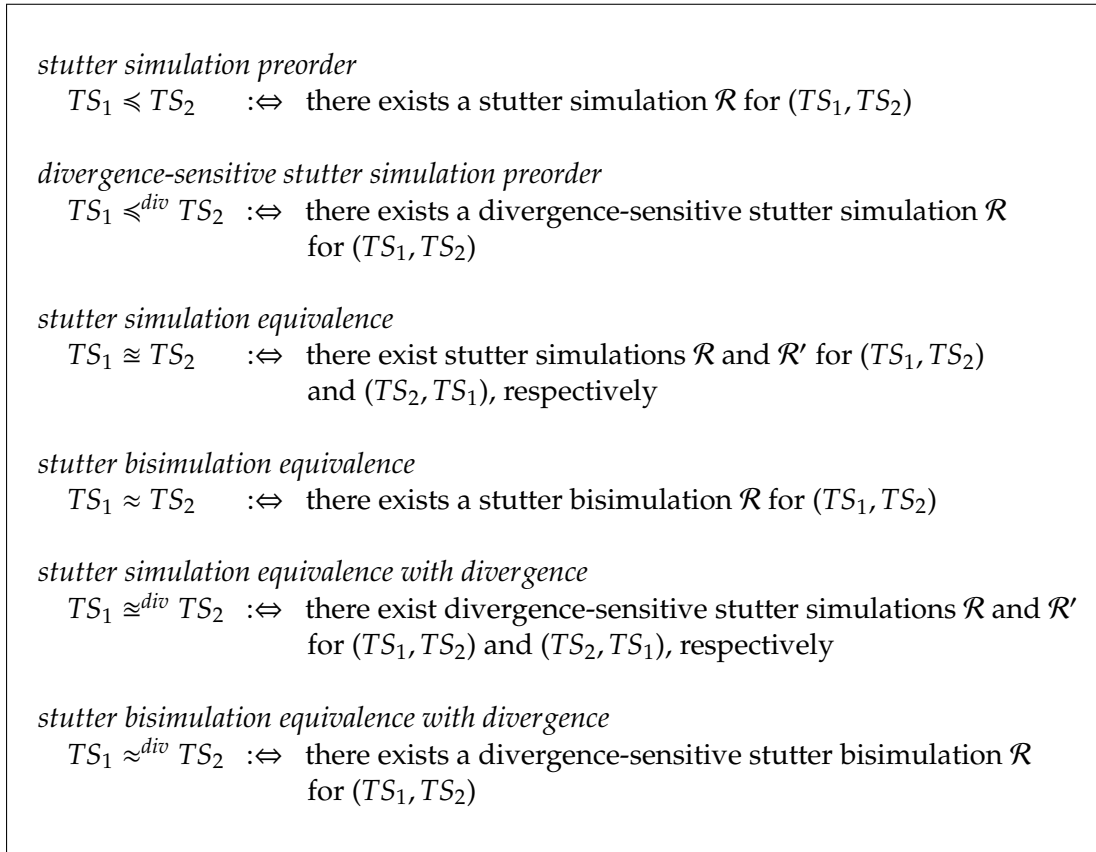


Figure 3.14: Stutter Implementation Relations

## 4 Stutter Simulation Quotienting

In this chapter we will step by step develop an algorithm to automatically computing (1) the stutter simulation preorder  $\leq_{TS}$  and (2) the quotient system  $TS/\cong_{TS}$  for finite transition systems based on the approach by Ranzato and Tapparo [6]. Furthermore, it can be checked whether a given relation  $\mathcal{R}$  is a stutter simulation preorder (or equivalence) and whether  $TS_1 \leq TS_2$  or  $TS_1 \cong TS_2$  by considering the disjoint union  $TS_1 \oplus TS_2$ . Thus, our algorithm always operates on a single transition system (and for the sake of simplicity we will omit the subscript  $TS$  of all defined relations). Note that we are here interested in the divergence-blind stutter simulation, i.e. divergence sensitivity does not play a role here.

The two main ingredients to computing the preorder  $\leq$  are the definition of a set of stutter simulator sets  $StSim$  and of an operation  $Refine(StSim, B)$  on this set, where  $B \subseteq S$ .  $StSim$  is a *representation* of the approximation of the stutter simulation preorder in the *current* iteration, where each  $StSim(s) \in StSim, s \in S$ , denotes those states that are *currently candidates* to stutter simulate  $s$ . Thus, the initial set of stutter simulator sets will be an *overapproximation* induced by the labelings of the states, i.e. if  $L(s) = L(t)$  then  $StSim(s) = StSim(t)$ . The refinement operation  $Refine(StSim, B)$  successively eliminates candidates from the stutter simulator sets according to the following observation. Whenever there is a state  $t \in StSim(s)$  that cannot stutter mimic some transition  $s \rightarrow s'$  of  $s$  then  $t$  must be excluded from the set of states that are candidates to stutter simulate  $s$ . Furthermore,  $t$  must be removed from all stutter simulator sets  $StSim(t')$ , where  $t'$  stutter simulates the desired behavior of  $s$ . For this, all such states  $t'$  are stored in the set  $B$  and the refinement operation simply needs to remove all states from  $StSim(t')$  which do not belong to  $B$ . If no more refinements are possible, then  $StSim$  represents the coarsest stutter simulation preorder, such that  $s \leq t$  if and only if  $t \in StSim(s)$ .

The computation of the stutter simulation equivalence  $\cong$  relies on the idea of *partition refinement*, where the state space  $S$  of the given transition system  $TS$  is partitioned into a set  $\Pi$  of disjoint sets of states, named *blocks*. A block  $B$  initially consists of all states that have the same labeling, such that each state  $s \in B$  is *candidate* to stutter simulate each state  $s'$  in the same block. Hence, in the beginning, the number of blocks in the initial partition  $\Pi_{AP}$  coincides with the number of atomic propositions, i.e.  $|\Pi_{AP}| = |AP|$ . In each iteration the algorithm checks whether there exists a so-called *Splitter*  $(B, C) \subseteq S \times S$ , such that at least one block  $B$  can be decomposed ("split") by the *set of constrained successors*  $Succ^*(B, C)$  of  $B$  with respect to  $C$  to obtain a partition  $\Pi_{i+1}$  that is finer than  $\Pi_i$  (where index  $i$  denotes the partition at the beginning of the  $i$ th iteration). The set  $Succ^*(B, C)$  consists of all states that can reach a state that stutter simulates each state in block  $C$  by only visiting states that stutter simulate all states in block  $B$ . The intuition is that if there is a state  $s \in B$  that has a direct successor in block  $C$  then each state  $t \in B$  must be able to stutter mimic this behavior. If this is not the case then  $t$  and  $s$  are no longer candidates to be stutter simulation equivalent and  $t$  must be excluded from  $B$ . To that end,  $B$  is split into two subblocks  $B_1, B_2$ ,

where  $B_1$  consists of all states  $t \in B \cap \text{Succ}^*(B, C)$ , and  $B_2$  inherits all other states. If no splitter exists, then all states of the same block are stutter simulation equivalent and  $\Pi = \Pi_{Eq}$  induces the stutter simulation equivalence  $\approx$ . As we will see, this requires a more advanced definition of  $\approx$  over blocks in  $\Pi$  rather than single states in  $S$ , which involves a redefinition of the set of stutter simulator sets  $StSim$ . However, if there exists a stutter simulation for  $TS$ , the algorithm will return a tuple  $(\Pi_{Eq}, \preceq_{pre})$ , where  $\Pi_{Eq}$  represents the stutter simulation equivalence  $\approx$  as a partition of the state space  $S$ , such that each block contains all stutter simulation equivalent states, and where the stutter simulation preorder  $\preceq$  is induced by the binary relation  $\preceq_{pre}$  between blocks in  $\Pi_{Eq}$ . In the remainder of this chapter we assume transition system  $TS = (S, Act, \rightarrow, I, AP, L)$  to be finite with possibly terminal states. Note that all notations, definitions, lemmas, etc. are reformulated according to Baier and Katoen [1].

## 4.1 Basics

We will now introduce the basic tools used throughout the sequel and which form the basis to understanding the procedural manner of the algorithms.

### DEFINITION 4.1.1 (PARTITION, BLOCK)

A *partition* for  $S$  is a set  $\Pi = \{ B_1, \dots, B_n \}$ , where  $B_i \neq \emptyset$ ,  $B_i \cap B_j = \emptyset$  for all  $i, j \in \{ 1, \dots, n \}$ ,  $i \neq j$  and  $S = \bigcup_{i=1}^n B_i$ .

$B_i \in \Pi$  is called a *block*. ■

For each state  $s \in S$  there exists exactly one block in  $\Pi$  that contains it, since  $\Pi$  is a partition of  $S$ . We will denote the unique block containing  $s$  by  $[s]_\Pi$ . Let  $\Pi_1, \Pi_2$  be two partitions of  $S$ .  $\Pi_1$  is called *finer* than  $\Pi_2$  (or, equivalently,  $\Pi_2$  is *coarser* than  $\Pi_1$ ), if the following statement holds:

$$\forall B_1 \in \Pi_1. \exists B_2 \in \Pi_2. B_1 \subseteq B_2.$$

Note that each block in  $\Pi_2$  represents the disjoint union of some blocks in  $\Pi_1$ . Partition  $\Pi_1$  is *strictly finer* than  $\Pi_2$  (or  $\Pi_2$  is *strictly coarser* than  $\Pi_1$ ) if  $\Pi_1$  is finer than  $\Pi_2$  and  $\Pi_1 \neq \Pi_2$ .

### NOTATION 4.1.1 (QUOTIENT SPACE)

A quotient space of  $S$  of some transition system  $TS$  under  $\mathcal{R}$ , denoted  $S/\mathcal{R} = \{ [s]_{\mathcal{R}} \mid s \in S \}$ , is the set consisting of all  $\mathcal{R}$ -equivalence classes. ■

### REMARK 4.1.1 (PARTITIONS AND EQUIVALENCES)

Each state of the quotient state space  $S/\mathcal{R}$ , induced by some equivalence relation  $\mathcal{R}$  on  $S$ , represents a set of  $\mathcal{R}$ -equivalent states in  $S$ , such that  $S/\mathcal{R}$  is a partition for  $S$ . Vice versa, a partition  $\Pi$  for  $S$  induces an equivalence relation:

$$\begin{aligned} \mathcal{R}_\Pi &= \{ (s_1, s_2) \mid \exists B \in \Pi. s_1 \in B \wedge s_2 \in B \} \\ &= \{ (s_1, s_2) \mid [s_1]_\Pi = [s_2]_\Pi \}, \end{aligned}$$

such that  $S/\mathcal{R}_\Pi$  corresponds to  $\Pi$ , i.e. each state in  $S/\mathcal{R}_\Pi$  represents one block of  $\Pi$ . For equivalence relation  $\mathcal{R}$ , the equivalence relation  $\mathcal{R}_\Pi$  induced by  $\Pi = S/\mathcal{R}$  corresponds to  $\mathcal{R}$ . ■

Thus, the quotient state space  $S/\cong$  induces a partition  $\Pi_{Eq}$ , such that each block in  $\Pi_{Eq}$  consists of states that are stutter simulation equivalent.

**DEFINITION 4.1.2 ( AP PARTITION )**

Let  $TS$  be some transition system. The  $AP$  partition, denoted  $\Pi_{AP}$ , is the quotient space  $S/\mathcal{R}_{AP}$  induced by  $\mathcal{R}_{AP} := \{ (s, t) \mid L(s) = L(t) \}$ . ■

Roughly speaking, the  $AP$  partition groups *all* states of the same labeling in a unique block, such that  $|\Pi_{AP}| = |AP|$ .

We already mentioned that the output of the main algorithm is a tuple  $(\Pi_{Eq}, \leq)$ , where  $\Pi_{Eq}$  induces the stutter simulation equivalence  $\cong$  and  $\leq$  the stutter simulation preorder. Recall that, in contrast to Definition 3.1.1, the relation  $\leq$  now operates on pairs of *blocks* of a single transition system rather than on pairs of *single states*.

**DEFINITION 4.1.3 ( PARTITION-RELATION PAIR )**

Let  $TS$  be a transition system. A *partition-relation pair*  $\mathcal{P} = (\Pi, \mathcal{R})$  is given by

- a *partition*  $\Pi$  of state space  $S$  and
- a binary *relation*  $\mathcal{R} \subseteq \Pi \times \Pi$  between blocks in  $\Pi$ . ■

## 4.2 Computing the Stutter Simulation Preorder $\leq$

In this section we will step by step engineer an algorithm to compute the stutter simulation preorder  $\leq$ . Therefore, we will introduce a new characterization of stutter simulation, which will be obtained from Notation 4.2.1. The equivalence relation  $\cong$ , however, is **not** considered, yet.

**DEFINITION 4.2.1 ( STUTTER SIMULATOR SET OF A SINGLE STATE )**

Let  $TS$  be some transition system and let  $\mathcal{R}$  be a binary relation on  $S$ . Then, the stutter-simulator set of  $s \in S$  w.r.t.  $\mathcal{R}$  is defined as

$$StSim_{\mathcal{R}}(s) := \{ t \in S \mid (s, t) \in \mathcal{R} \}.$$

$StSim_{\mathcal{R}}$  is the set of all stutter simulator sets of all states  $s \in S$  w.r.t.  $\mathcal{R}$ , i.e.  $StSim_{\mathcal{R}} := \bigcup_{s \in S} \{ StSim_{\mathcal{R}}(s) \}$ . ■

According to the following algorithm, the relation  $\mathcal{R}$  will denote the approximation of the stutter simulation preorder  $\leq$  in the *current* iteration, such that the stutter simulator set of an arbitrary state  $s$  identifies the set of states that are *currently* candidates to stutter simulate  $s$ . Note that  $StSim_{AP}(s) = \{ t \in S \mid L(s) = L(t) \}$  represents the set of states that have the same labeling as  $s$ . If  $\mathcal{R}$  is a stutter simulation for  $(TS, TS)$  then  $StSim_{\mathcal{R}}(s)$  is a subset of  $StSim_{TS}(s)$  (see p.8). The stutter simulation preorder  $\leq$  can be obtained from  $StSim_{TS}$  as follows:

$$\leq = \{ (s, t) \in S \times S \mid t \in StSim_{TS}(s) \}.$$

$StSim_{\mathcal{R}}$  is *finer* than  $StSim_{\mathcal{R}'}$  (or  $StSim_{\mathcal{R}'}$  is *coarser* than  $StSim_{\mathcal{R}}$ ) if  $\mathcal{R} \subseteq \mathcal{R}'$  or, equivalently, if  $StSim_{\mathcal{R}}(s) \subseteq StSim_{\mathcal{R}'}(s)$  for all  $s \in S$ .

**NOTATION 4.2.1 ( CONSTRAINED SUCCESSORS )**

Let  $TS$  be some transition system and  $B, C \subseteq S$ . We have that  $s \in Succ^*(B, C)$  whenever there exists a finite path fragment

$$s = s_0s_1\dots s_n \in Paths(s) \text{ with } s_i \in B \text{ for all } 0 \leq i < n \text{ and } s_n \in C, n \geq 0. \quad \blacksquare$$

Stated in words, some state  $s \in S$  belongs to  $Succ^*(B, C)$  if  $s$  is contained in block  $B$  and can reach block  $C$  via a finite path fragment where only states in  $B$  are visited.

**DEFINITION 4.2.2 ( REFINER, STABILITY )**

Let  $\mathcal{R}$  be a binary relation on  $S$ ,  $StSim_{\mathcal{R}}$  be the set of stutter simulator sets of all states  $s \in S$  and  $s, s'$  be two states.

- (1) The pair  $(s, s')$  is a *refiner* for  $StSim_{\mathcal{R}}$  if and only if  $s' \in Post(s)$  and  $StSim_{\mathcal{R}}(s) \not\subseteq Succ^*(StSim_{\mathcal{R}}(s), StSim_{\mathcal{R}}(s'))$ .
- (2)  $StSim_{\mathcal{R}}(s)$  is *stable* if there exists no  $s' \in Post(s)$  such that  $(s, s')$  is a refiner for  $StSim_{\mathcal{R}}(s)$ .
- (3)  $StSim_{\mathcal{R}}$  is *stable* if  $StSim_{\mathcal{R}}(s)$  is stable for all  $s \in S$ . \blacksquare

In terms of stutter simulation, the idea is that if a state  $s$  has a direct successor  $s'$  then each candidate  $t \in StSim_{\mathcal{R}}(s)$  must be able to reach some state in  $StSim_{\mathcal{R}}(s')$  by solely visiting states in  $StSim_{\mathcal{R}}(s)$ , i.e. to stutter mimic the behavior of  $s$ . Otherwise,  $t$  clearly cannot stutter simulate  $s$  and hence must be removed from the set of states  $StSim_{\mathcal{R}}(s)$  that are candidates to stutter simulate  $s$ .

**LEMMA 4.2.1 ( STUTTER SIMULATION )**

Let  $TS$  be a transition system and  $\mathcal{R}$  be a binary relation on  $S$ . Then:

$\mathcal{R}$  is a stutter simulation for  $(TS, TS)$   
if and only if for all  $s \in S$

$$StSim_{\mathcal{R}}(s) \subseteq StSim_{AP}(s) \text{ and if } s' \in Post(s) \text{ then } StSim_{\mathcal{R}}(s) \subseteq Succ^*(StSim_{\mathcal{R}}(s), StSim_{\mathcal{R}}(s')).$$

**PROOF** Let  $TS$  be some transition system with state space  $S$ .

- $\Rightarrow$ : Assume  $\mathcal{R} \subseteq S \times S$  is a stutter simulation for  $(TS, TS)$ . Then  $StSim_{\mathcal{R}}(s) \subseteq StSim_{AP}(s)$ , since for all  $(u, v) \in \mathcal{R}$  it holds that  $L(u) = L(v)$ . Let  $t \in StSim_{\mathcal{R}}(s)$  and  $s' \in Post(s)$ . The proof is by contraposition. Assume that  $t \notin Succ^*(StSim_{\mathcal{R}}(s), StSim_{\mathcal{R}}(s'))$ . Consequently, there exists no finite path fragment  $t = t_0t_1\dots t_n$  outgoing from  $t$  ( $n \geq 0$ ) with  $t_i \in StSim_{\mathcal{R}}(s)$ ,  $i = 0, \dots, n-1$ , and  $t_n \in StSim_{\mathcal{R}}(s')$ . Accordingly, there is no finite path fragment  $t = t_0t_1\dots t_n$  emanating from  $t$ , such that the pairs  $(s, t_i)$  and  $(s', t_n)$  are contained in  $\mathcal{R}$ . This contradicts the assumption that  $\mathcal{R}$  is a stutter simulation for  $(TS, TS)$ .
- $\Leftarrow$ : Let  $\mathcal{R}$  be a binary relation on  $S$ . Assume  $StSim_{\mathcal{R}}(s) \subseteq StSim_{AP}(s)$  and  $StSim_{\mathcal{R}}(s) \subseteq Succ^*(StSim_{\mathcal{R}}(s), StSim_{\mathcal{R}}(s'))$  for all  $s \in S, s' \in Post(s)$ . Let  $t$  be some state in  $StSim_{\mathcal{R}}(s)$ . By definition, it holds that all pairs  $(s, t)$  are contained in  $\mathcal{R}$ . Since the stutter-simulator set of state  $s$  is a subset of  $StSim_{AP}(s)$ , it directly follows that the labeling condition (1) for stutter simulation is satisfied for all pairs  $(s, t) \in \mathcal{R}$ . Furthermore, if  $t \in StSim_{\mathcal{R}}(s)$  then there exists a finite path fragment  $t = t_0t_1\dots t_n$  ( $n \geq 0$ ), such that  $t_i \in StSim_{\mathcal{R}}(s)$  for all  $0 \leq i < n$  and  $t_n \in StSim_{\mathcal{R}}(s')$ , since  $StSim_{\mathcal{R}}(s) \subseteq Succ^*(StSim_{\mathcal{R}}(s), StSim_{\mathcal{R}}(s'))$ . By definition, we have that  $(s, t_i) \in \mathcal{R}$  and  $(s', t_n) \in \mathcal{R}$  and hence, condition (2) for stutter simulation is trivially fulfilled. Thus,  $\mathcal{R}$  is a stutter simulation for  $(TS, TS)$ . \blacksquare

Thus, to obtain the stutter simulation preorder, we initially compute the set of stutter simulator sets  $StSim_{AP}$ , such that the stutter simulator set of each state  $s$  contains *all* states that are candidates to stutter simulate  $s$ . Then, these sets are iteratively refined until the current set of stutter simulator sets is *stable*, i.e. in every iteration it must be checked for all states  $s$  whether there exists a state  $u \in StSim_{\mathcal{R}}(s)$  that violates Lemma 4.2.1. In this case, there is at least one transition  $s \rightarrow s'$  that cannot be mimicked by state  $u$  by only visiting states in  $StSim_{\mathcal{R}}(s)$  and, by definition of stutter simulation,  $s$  is not stutter simulated by  $u$ . Hence,  $u$  must be removed from  $StSim_{\mathcal{R}}(s)$ . Furthermore, all states  $v$  that are candidate to stutter simulate  $s$  and which have an outgoing path  $v = v_0v_1\dots v_n$  ( $n \geq 0$ ), where  $v_i \in StSim_{\mathcal{R}}(s)$  for all  $0 \leq i < n$  and  $v_n \in StSim_{\mathcal{R}}(s')$ , cannot be stutter simulated by  $u$  for the sake of transitivity and thus,  $u$  must also be excluded from their stutter simulator sets. This leads to the following refinement operation of  $StSim_{\mathcal{R}}$ . (Note that  $Sat(s, s')$  is an abbreviation for the set of constrained successors  $Succ^*(StSim_{\mathcal{R}}(s), StSim_{\mathcal{R}}(s'))$ .)

**DEFINITION 4.2.3 ( REFINEMENT OPERATION )**

Let  $(s, s')$  be a refiner for  $StSim_{\mathcal{R}}$  and  $Sat(s, s') = Succ^*(StSim_{\mathcal{R}}(s), StSim_{\mathcal{R}}(s'))$ . Then,  $StSim_{\mathcal{R}}$  is refined as follows:

$$Refine(StSim_{\mathcal{R}}, Sat(s, s')) := \bigcup_{t \in S} \{ Refine(StSim_{\mathcal{R}}(t), Sat(s, s')) \},$$

where

$$Refine(StSim_{\mathcal{R}}(t), Sat(s, s')) := \begin{cases} StSim_{\mathcal{R}}(t) \cap Sat(s, s'), & \text{if } t \in Sat(s, s') \\ StSim_{\mathcal{R}}(t), & \text{if } t \notin Sat(s, s'). \end{cases}$$

■

Observe the situation given in Figure 4.1, where  $\{s, t, u\} \subseteq StSim_{\mathcal{R}}(s)$  and  $t \in StSim_{\mathcal{R}}(u)$ .

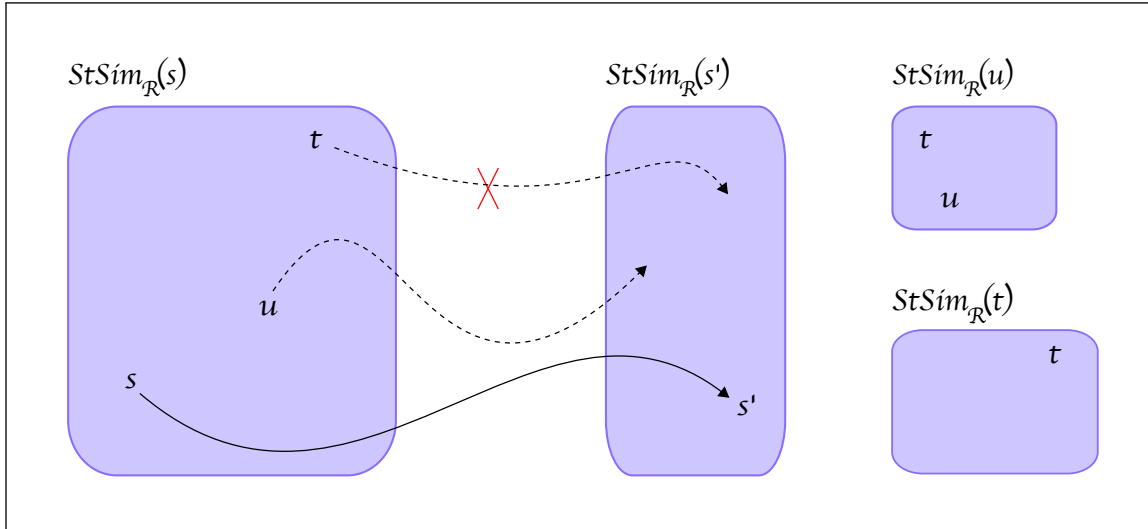


Figure 4.1: Current approximation of  $\leq$

Assume that  $(s, s')$  is a refiner for  $StSim_{\mathcal{R}}$ . Then, by the definition of refiners,  $s'$  is a direct

successor of  $s$ , i.e.  $s' \in \text{Post}(s)$  (solid arrow from  $s$  to  $s'$ ), and there exists  $t \in \text{StSim}_{\mathcal{R}}(s)$  with  $t \notin \text{Succ}^*(\text{StSim}_{\mathcal{R}}(s), \text{StSim}_{\mathcal{R}}(s'))$  (crossed out dotted arrow). Additionally, let  $u \in \text{Succ}^*(\text{StSim}_{\mathcal{R}}(s), \text{StSim}_{\mathcal{R}}(s'))$  (dotted arrow), that is,  $u$  stutter simulates the behavior of  $s$  w.r.t. state  $s'$ . Clearly, state  $t$  must be excluded from the stutter simulator set of  $s$  and furthermore, for the sake of transitivity of the stutter simulation preorder,  $t$  needs to be removed from all stutter simulator sets of all states contained in the current set of constrained successors. Hence, since  $t$  is included in  $\text{StSim}_{\mathcal{R}}(u)$ , where  $u$  exhibits the desired behavior,  $t$  can no longer be candidate to stutter simulate  $u$  (compare to Figure 4.2).

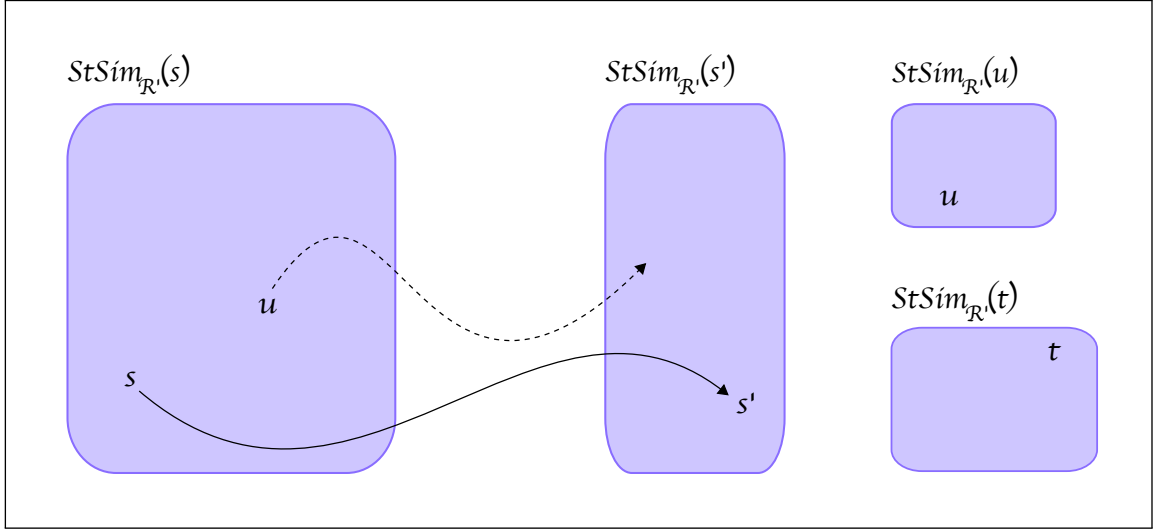


Figure 4.2: Refinement Operation

The updated stutter simulator sets are obtained by intersecting the old stutter simulator sets with the current set of constrained successors, i.e.  $\text{StSim}_{\mathcal{R}'}(s) := \text{StSim}_{\mathcal{R}}(s) \cap \text{Succ}^*(\text{StSim}_{\mathcal{R}}(s), \text{StSim}_{\mathcal{R}}(s'))$  and  $\text{StSim}_{\mathcal{R}'}(u) := \text{StSim}_{\mathcal{R}}(u) \cap \text{Succ}^*(\text{StSim}_{\mathcal{R}}(s), \text{StSim}_{\mathcal{R}}(s'))$ , and it follows that  $t$  is no longer candidate to stutter simulate  $s$  and  $u$ .

We will now prove that this operation is correct, i.e. starting with the initial approximation  $\text{StSim}_{AP}$ , the successive refinement provides a series of approximations  $\text{StSim}_{\mathcal{R}_0} = \text{StSim}_{AP}, \text{StSim}_{\mathcal{R}_1}, \dots, \text{StSim}_{\mathcal{R}_n}$ , which become increasingly finer and which are all coarser than  $\text{StSim}_{TS}$ . Once a stable situation is reached, e.g. in the  $n$ th iteration ( $n > 0$ ), there are no further refinements possible and  $\text{StSim}_{\mathcal{R}_n}$  is the stutter simulation preorder  $\text{StSim}_{TS}$ .

**REMARK 4.2.1 ( REFINEMENT OPERATION VS. PREORDER )**

Let  $\text{StSim}_{\mathcal{R}_i}$  be the current approximation of the stutter simulation preorder  $\preceq$ , where  $\text{StSim}_{\mathcal{R}_0} = \text{StSim}_{AP}$  and  $\text{StSim}_{\mathcal{R}_i} = \text{Refine}(\text{StSim}_{\mathcal{R}_{i-1}}, \text{Sat}(u, u'))$ . Then, if  $(s, s')$  is a refiner for  $\text{StSim}_{\mathcal{R}_i}$  and if  $\text{StSim}_{\mathcal{R}_i}$  is a preorder then  $\text{StSim}_{\mathcal{R}_{i+1}} = \text{Refine}(\text{StSim}_{\mathcal{R}_i}, \text{Sat}(s, s'))$  is a preorder.

**PROOF** Let  $\text{StSim}_{\mathcal{R}_i}$  be the current set of stutter simulator sets of all states  $s \in S$ , where  $i$  denotes the set of stutter simulator sets before the  $i$ th refinement. The proof is by induction on  $i$ .

- (1) Base case:  $i = 0$ .

Clearly,  $\text{StSim}_{\mathcal{R}_0} = \text{StSim}_{AP}$  is reflexive. We omit the details and show transitivity.

Recall that in the initial approximation a state  $t$  is contained in the stutter simulator set of some state  $s$  whenever  $L(s) = L(t)$ . Thus, if  $t \in StSim_{\mathcal{R}_0}(s)$  and  $s \in StSim_{\mathcal{R}_0}(u)$  then  $u \in StSim_{\mathcal{R}_0}(t)$ , since  $L(t) = L(s) = L(u)$ . Hence,  $StSim_{\mathcal{R}_0}$  is a preorder.

- (2) Assume  $i > 0$ , where  $StSim_{\mathcal{R}_i}$  is a preorder. Let  $StSim_{\mathcal{R}_{i+1}} = Refine(StSim_{\mathcal{R}_i}, Sat(s, s'))$ , where  $(s, s')$  is a refiner for  $StSim_{\mathcal{R}_i}$ . By definition of the refinement operation, the  $i + 1$ st approximation  $StSim_{\mathcal{R}_{i+1}}(t)$  of the stutter simulator set of each state  $t \in S$  is obtained from the intersection  $StSim_{\mathcal{R}_i} \cap Sat(s, s')$  if  $t \in Sat(s, s')$ . Otherwise,  $StSim_{\mathcal{R}_{i+1}}(t) = StSim_{\mathcal{R}_i}(t)$ . Consequently,  $StSim_{\mathcal{R}_{i+1}}(t) \subseteq StSim_{\mathcal{R}_i}(t)$  for all states  $t \in S$ . Furthermore, observe that the refinement operation never excludes a state from its current stutter simulator set and thus, it directly follows that  $StSim_{\mathcal{R}_{i+1}}$  is reflexive. To show transitivity of  $StSim_{\mathcal{R}_{i+1}}$ , let  $u$  be contained in  $StSim_{\mathcal{R}_{i+1}}(t_1) \subseteq StSim_{\mathcal{R}_i}(t_1)$  and  $t_1$  in  $StSim_{\mathcal{R}_{i+1}}(t_2) \subseteq StSim_{\mathcal{R}_i}(t_2)$ . By transitivity of  $StSim_{\mathcal{R}_i}$ , it holds that  $u \in StSim_{\mathcal{R}_i}(t_2)$ . The proof is by contradiction. Assume that  $u \notin StSim_{\mathcal{R}_{i+1}}(t_2)$  but  $u \in StSim_{\mathcal{R}_{i+1}}(t_1)$  and  $t_1 \in StSim_{\mathcal{R}_{i+1}}(t_2)$ . Then,  $u$  is excluded from  $StSim_{\mathcal{R}_i}(t_2)$  by the refinement operation and thus,  $t_2$  belongs to  $Sat(s, s')$ , whereas  $u$  does not. Furthermore, since  $t_1 \in StSim_{\mathcal{R}_{i+1}}(t_2)$ , we have that  $t_1 \in Sat(s, s')$  and thus, the next approximation of the stutter simulator set of  $t_1$  is obtained by the intersection  $StSim_{\mathcal{R}_i} \cap Sat(s, s')$ . Since  $u$  is not contained in  $Sat(s, s')$ , it must be excluded from  $StSim_{\mathcal{R}_{i+1}}(t_1)$ . This contradicts the assumption that  $u$  belongs to  $StSim_{\mathcal{R}_{i+1}}(t_1)$ . Thus, the approximation  $StSim_{\mathcal{R}_{i+1}}$  is transitive.

It follows that if there exists a refiner  $(s, s')$  for  $StSim_{\mathcal{R}_i}$  and if  $StSim_{\mathcal{R}_i}$  is a preorder then  $StSim_{\mathcal{R}_{i+1}}$  is a preorder. ■

**LEMMA 4.2.2 ( CORRECTNESS OF THE REFINEMENT OPERATION )**

Let  $StSim_{\mathcal{R}_i}$  be the current approximation of the stutter simulation preorder  $\leq$ , which is finer than  $StSim_{AP}$ , coarser than  $StSim_{TS}$  and where  $StSim_{\mathcal{R}_0} = StSim_{AP}$ . Then, if  $(s, s')$  is a refiner for  $StSim_{\mathcal{R}_i}$  it holds that

$StSim_{\mathcal{R}_{i+1}} = Refine(StSim_{\mathcal{R}_i}, Sat(s, s'))$  is (1) finer than  $StSim_{\mathcal{R}_i}$  and (2) coarser than  $StSim_{TS}$ .

PROOF Let  $StSim_{\mathcal{R}_i}$  be the current set of stutter simulator sets at the beginning of the  $i$ th refinement. The proof is by induction on  $i$ .

- (1) Base case:  $i = 0$ . Since no refinement was carried out yet, the first claim is trivially fulfilled. Furthermore, since  $StSim_{\mathcal{R}_0} = StSim_{AP}$ , it follows directly that  $StSim_{\mathcal{R}_0}$  is coarser than  $StSim_{TS}$ .
- (2) Assume  $i > 0$ , where  $StSim_{\mathcal{R}_i}$  is finer than  $StSim_{\mathcal{R}_{i-1}}$  and coarser than  $StSim_{TS}$ . Let  $(s, s')$  be a refiner for  $StSim_{\mathcal{R}_i}$  and  $StSim_{\mathcal{R}_{i+1}} = Refine(StSim_{\mathcal{R}_i}, Sat(s, s'))$ .
- (2.1) Recall that by definition of the refinement operation we only obtain a smaller stutter simulator set for a state  $t$  if  $t$  can reach a state in  $StSim_{\mathcal{R}_i}(s')$  via a path exclusively consisting of states in  $StSim_{\mathcal{R}_i}(s)$ , i.e. if  $t \in Sat(s, s')$ . Otherwise,  $StSim_{\mathcal{R}_i}(t)$  cannot be refined according to  $(s, s')$  and hence remains the same. It immediately follows that  $StSim_{\mathcal{R}_{i+1}} = Refine(StSim_{\mathcal{R}_i}, Sat(s, s'))$  is finer than  $StSim_{\mathcal{R}_i}$ .
- (2.2) For the second claim, we have to prove that  $Refine(StSim_{\mathcal{R}_i}(t), Sat(s, s'))$  is a subset of  $StSim_{TS}(t)$  for every state  $t \in S$ . Distinguish between two cases.

(2.2.1)  $t \notin \text{Sat}(s, s')$ . Then, we obtain that  $\text{Refine}(\text{StSim}_{\mathcal{R}_i}(t), \text{Sat}(s, s')) = \text{StSim}_{\mathcal{R}_{i+1}}(t)$  by definition of the refinement operation. Since  $\text{StSim}_{\mathcal{R}_i}(t) \subseteq \text{StSim}_{TS}(t)$ , it follows that  $\text{Refine}(\text{StSim}_{\mathcal{R}_{i+1}}(t), \text{Sat}(s, s')) \subseteq \text{StSim}_{TS}(t)$ .

(2.2.2)  $t \in \text{Sat}(s, s')$ . Then,  $t \in \text{StSim}_{\mathcal{R}_i}(s)$  and furthermore,  $\text{StSim}_{\mathcal{R}_{i+1}}(t) = \text{StSim}_{\mathcal{R}_i}(t) \cap \text{Sat}(s, s')$ . The proof is by contradiction and we assume that there exists a state  $u$  that is contained in  $\text{StSim}_{TS}(t)$  and  $\text{StSim}_{\mathcal{R}_i}(t)$  but not in  $\text{StSim}_{\mathcal{R}_{i+1}}(t)$ . Then,  $u$  is not inherited in  $\text{Sat}(s, s')$  and hence, either  $u \notin \text{StSim}_{\mathcal{R}_i}(s)$  or  $u \notin \text{StSim}_{\mathcal{R}_{i+1}}(s)$ . Observe that  $u$  must belong to  $\text{StSim}_{\mathcal{R}_i}(s)$ , since, by Remark 4.2.1,  $\text{StSim}_{\mathcal{R}_i}$  is a preorder and thus transitive. Consequently,  $u \notin \text{StSim}_{\mathcal{R}_{i+1}}(s)$ , i.e. there exists no finite path fragment  $u = u_0 \dots u_n$  ( $n \geq 0$ ) with  $u_i \in \text{StSim}_{\mathcal{R}_i}(s)$  for all  $i \in \{0, \dots, n-1\}$  and  $u_n \in \text{StSim}_{\mathcal{R}_i}(s')$ . Then, since  $t \in \text{Sat}(s, s')$ , this contradicts the assumption that  $u \in \text{StSim}_{TS}(t)$  and it follows that  $\text{StSim}_{\mathcal{R}_{i+1}}(t) \subseteq \text{StSim}_{TS}(t)$ .

Hence,  $\text{StSim}_{\mathcal{R}_{i+1}}$  is coarser than  $\text{StSim}_{TS}$ .

We obtain that if  $(s, s')$  is a refiner for  $\text{StSim}_{\mathcal{R}_i}$  and if  $\text{StSim}_{\mathcal{R}_i}$  is finer than  $\text{StSim}_{\mathcal{R}_{i-1}}$  and coarser than  $\text{StSim}_{TS}$  then  $\text{StSim}_{\mathcal{R}_{i+1}}$  is finer than  $\text{StSim}_{\mathcal{R}_i}$  and coarser than  $\text{StSim}_{TS}$ . ■

The essential steps to iteratively refining the set of stutter simulator sets  $\text{StSim}$  are outlined in Algorithm 4.2.1, which takes as input a finite transition system  $TS = (S, \text{Act}, \rightarrow, I, \text{AP}, L)$  with possibly terminal states and computes the stutter simulation preorder  $\preceq$ , which will be represented by  $\text{StSim} := \bigcup_{s \in S} \{ \text{StSim}(s) \}$ .

---

**Algorithm 4.2.1:** Computing the Stutter Simulation Preorder

---

**Input** : Transition System  $TS$

**Output**: Stutter simulation preorder  $\text{StSim}_{TS}$

```

1 forall  $s \in S$  do (* initialization *)
2    $\text{StSim}(s) := \{ t \in S \mid L(s) = L(t) \};$ 
3 while  $\exists s \in S. \exists s' \in \text{Post}(s)$  with  $\text{StSim}(s) \not\subseteq \text{Succ}^*(\text{StSim}(s), \text{StSim}(s'))$  do
4    $\text{Sat}(s, s') := \text{Succ}^*(\text{StSim}(s), \text{StSim}(s'));$  (*  $\exists u \in \text{StSim}(s)$  with  $s \not\preceq u$  *)
5   forall  $t \in \text{Sat}(s, s')$  do
6      $\text{Refine}(\text{StSim}(t), \text{Sat}(s, s'));$  (* refinement operation *)
(*  $\text{StSim}(s) = \text{StSim}_{TS}(s)$  for any  $s$  *)
7 return  $\{ \text{StSim}(s) \}_{s \in S};$  (*  $\bigcup_{s \in S} \{ \text{StSim}(s) \} = \text{StSim}_{TS}$  *)

```

---

In the beginning of each execution of the *while*-loop (ll. 3-6), the set  $\text{StSim} := \bigcup_{s \in S} \{ \text{StSim}(s) \}$  represents the current approximation  $\mathcal{R}$  of the stutter simulation preorder  $\preceq$ . Here, a pair of states  $(s, t)$  belongs to  $\mathcal{R}$  iff  $t \in \text{StSim}(s)$ . Since the state space of the given transition system is finite and following Lemma 4.2.2, Algorithm 4.2.1 terminates after  $n \in \mathbb{N}_0$  iterations of the *while*-loop on input  $TS$ . It remains to prove that Algorithm 4.2.1 computes the *coarsest* stutter simulation preorder.

**THEOREM 4.2.1 ( CORRECTNESS OF ALGORITHM 4.2.1 )**

Let  $TS$  be a transition system and  $\text{StSim}$  be the output of Algorithm 4.2.1. Then, for all  $s, t \in S$  it holds that

$t \in StSim(s)$  if and only if  $s \leq t$ .

PROOF Let  $TS$  be some transition system.

$\Rightarrow$ : Assume  $StSim = StSim_n$  is the output of Algorithm 4.2.1 after  $n \geq 0$  iterations of the *while*-loop. Then, for all  $s \in S$  there exists no  $s' \in Post(s)$  with  $StSim(s) \not\subseteq Succ^*(StSim(s), StSim(s'))$ . Furthermore,  $StSim(s) \subseteq StSim_{AP}(s)$  which can be seen as follows. Once the initial approximation  $StSim_0 = StSim_{AP}$  is computed (ll. 1-2), in each iteration  $i > 0$  the refinement operation is applied as long as a refiner exists (ll. 3-6). Let  $StSim_i$  denote the current approximation in the beginning of the  $i$ th iteration of the *while*-loop. By Lemma 4.2.2, we have that  $StSim_{i+1}(s) \subseteq StSim_i(s)$  and hence,  $StSim = StSim_n \subseteq StSim_{AP}$ . Following Lemma 4.2.1,  $StSim$  is a stutter simulation for  $(TS, TS)$  and thus, for all  $s, t \in S$  it holds that if  $t \in StSim(s)$  then  $s \leq t$ .

$\Leftarrow$ : Assume that  $s \leq t$ , i.e.  $t \in StSim_{TS}(s)$ , and Algorithm 4.2.1 terminates after  $n \geq 0$  iterations of the *while*-loop with output  $StSim = StSim_n$ . Let  $i \leq n$  denote the current approximation  $StSim_i$  in the beginning of the  $i$ th iteration of the *while*-loop, where the initial approximation  $StSim_0 = StSim_{AP}$  was computed in lines 1-2. Clearly,  $t \in StSim_{AP}(s)$ . Note that the *while*-loop implements the refinement operation, i.e. if there exists a refiner in the  $i$ th iteration then the refinement operation is applied to the current approximation  $StSim_i$ . Following Lemma 4.2.2,  $StSim_i$  is coarser than  $StSim_{TS}$  for all  $0 \leq i \leq n$ , since  $StSim_0 = StSim_{AP}$ , and hence,  $t$  is inherited in  $StSim_i(s)$  in each iteration. It follows that for all  $s, t \in S$  if  $s \leq t$  then  $t \in StSim(s)$  and thus,  $StSim$  is the coarsest stutter simulation for  $(TS, TS)$ . ■

#### THEOREM 4.2.2 ( TIME COMPLEXITY OF ALGORITHM 4.2.1 )

The stutter simulation quotient space can be computed by Algorithm 4.2.1 with time complexity  $O(|S| \cdot |AP| + |\rightarrow| \cdot |S|^5)$ .

PROOF Let  $n = |S|$  denote the number of states,  $m = |\rightarrow|$  the number of transitions, where we assume that  $m \geq n$ .

(1) Initialization (ll.1-2):

Following Baier and Katoen [1] (p.478ff), computing the initial approximation takes time  $O(n \cdot |AP|)$ , where  $AP$  is the set of atomic propositions in  $TS$ .

(2) The *while*-Loop (ll.3-6):

(2.1) Number of Executions of the *while*-loop:

Once there is a pair of states  $s, s'$  that satisfies the guard of the *while*-loop its body is executed. A stutter simulator set consists of at most  $n$  elements. Since every state can stutter simulate itself, it follows that at most  $n - 1$  elements can be removed from a single set. Hence, the body of the *while*-loop can be entered at most  $O(n^2)$  times.

(2.2) Checking the Guard of the Loop (l.3):

The loop-condition in l.3 is checked for each successor  $s'$  of every state  $s$ , i.e.  $\sum_{s \in S} |Post(s)| \in O(m)$  times in total.

(2.2.1) Computing the Set of Constrained Successors:

Let  $s, s'$  be the current pair of states for which we need to test whether they fulfill the guard of the *while*-loop. Therefore, we first have to compute the set of constrained successors of the stutter simulator set of  $s$  with respect to the stutter simulator set of  $s'$ . This can be done in time  $\mathcal{O}(n^3)$  by applying the Floyd-Warshall Algorithm [2] for computing the transitive closure of a given graph with a slight modification. Assume transition system  $TS$  is represented by an adjacency matrix. Then, instead of computing the existence of a path between *each* pair of states, we need to consider tuples  $t, t'$ , where  $t \in StSim(s)$  and  $t' \in StSim(s')$ , and focus on the existence of a path from  $t$  to  $t'$  via states in  $StSim(s)$ . This boils down to examining a subgraph of the given transition system that consists of the states in  $StSim(s)$  and  $StSim(s')$  and once a path from  $t$  to at least one state in  $StSim(s')$  via states in  $StSim(s)$  is detected,  $t$  can be added to  $Sat(s, s')$ . To guarantee that the path from  $t$  to  $t'$  exhibits the desired pattern, our algorithm simply compares all possible paths from  $v$  to  $w$  visiting  $k$ , where states  $v$  and  $k$  are contained in the stutter simulator set of  $s$  and  $w$  belongs to  $StSim(s')$ , i.e.  $\mathcal{O}(|StSim(s)|^2 \cdot |StSim(s')|)$  comparisons at most. Thus,  $\mathcal{O}(n^3)$  is an upper bound to computing the set of constrained successors  $Sat(s, s')$  of  $StSim(s)$  w.r.t.  $StSim(s')$ .

(2.2.2) Testing the Subset-Relation:

It remains to test whether the stutter simulator set of  $s$  is a subset of  $Sat(s, s')$ . Note that  $Sat(s, s')$  can at most include all elements inherited in  $StSim(s)$  and consequently,  $|Sat(s, s')| \leq n$ . The subset-relation can be tested by adding all states in  $Sat(s, s')$  to a hash table  $H$  and checking if for each state in  $StSim(s)$  there exists an entry in  $H$ . This takes time  $\mathcal{O}(2n) = \mathcal{O}(n)$ .

It follows that the costs required to check the condition of the *while*-loop for a single pair of states are in  $\mathcal{O}(n^3)$ .

(2.3) Loop-Body (ll.4-6):

Observe that  $Sat(s, s') = Succ^*(StSim(s), StSim(s'))$  was already computed in the guard of the *while*-loop (l.3) and we only have to consider the number of iterations of the *forall*-loop in line 5, which depends on the cardinality of  $Sat(s, s')$ . In case  $Sat(s, s') = StSim(s)$ , the body of the loop would not be entered and thus,  $|Sat(s, s')|$  yields at most  $n - 1$ . Then, for every state  $t \in Sat(s, s')$  the intersection  $StSim(t) \cap Sat(s, s')$  is computed in  $\mathcal{O}(n)$  by adding once all states contained in  $Sat(s, s')$  into a hash table  $H$  and then excluding state  $t'$  from  $StSim(t)$  if there exists no entry in  $H$  for  $t'$ . An upper bound for this computation is  $\mathcal{O}(n^2)$ .

The total complexity of Algorithm 4.2.1 is  $\mathcal{O}(n \cdot |AP| + n^2 \cdot (m \cdot (n^3 + n) + n^2)) = \mathcal{O}(n \cdot |AP| + n^2 \cdot (m \cdot n^3 + n^2)) = \mathcal{O}(|S| \cdot |AP| + |\rightarrow| \cdot |S|^5)$ . ■

Note that this is a rather coarse upper bound for the time complexity of Algorithm 4.2.1, since e.g. the set  $Sat(s, s')$  is not necessarily needed to be recomputed in every iteration. However, the given approximation suffices for the moment.

### 4.3 Computing the Stutter Simulation Preorder $\leq$ and Equivalence $\approx$

The additional computation of the stutter simulation equivalence  $\Pi_{Eq}$  clearly requires an adaption of the previous definitions. Here, the stutter simulation preorder  $\leq$  will be given in a more abstract manner, i.e. it is no longer explicitly represented as a set of stutter simulator sets for every single state in  $S$  but instead as a set of stutter simulator sets of blocks of states contained in partition  $\Pi_{Eq}$ . This abstraction yields a *relation between pairs of blocks* in  $\Pi_{Eq}$ , where each block groups all stutter simulation equivalent states in  $TS$ . Thus, the relation  $\leq$  needs to be redefined. The coarsest stutter simulation preorder is induced by the partition relation pair  $\mathcal{P} = (\Pi_{Eq}, \leq_{Pre})$  and represented by the set of stutter simulator sets  $StSim_{\mathcal{P}}$ .

#### DEFINITION 4.3.1 ( ABSTRACTION OF THE STUTTER SIMULATION EQUIVALENCE )

Let  $\mathcal{R}$  be a binary relation on  $S$  and  $s, t \in S$ . Then, the stutter simulation equivalence is a partition  $\Pi_{\mathcal{R}}$  of  $S$  induced by  $\mathcal{R}$ , where

$$[s]_{\Pi_{\mathcal{R}}} = [t]_{\Pi_{\mathcal{R}}} \text{ if and only if } \forall u \in S. (u, s) \in \mathcal{R} \Leftrightarrow (u, t) \in \mathcal{R}.$$

If  $\mathcal{R}$  is the stutter simulation preorder  $\leq$  then  $\Pi_{\mathcal{R}}$  induces  $\approx$  and will be denoted by  $\Pi_{Eq}$ . ■

In the following,  $\mathcal{R}$  will denote the current approximation of the stutter simulation preorder according to Algorithm 4.3.1. Thus, if  $[s]_{\Pi_{\mathcal{R}}} = [t]_{\Pi_{\mathcal{R}}}$  then  $s$  and  $t$  are currently candidates to be stutter simulation equivalent and furthermore, if  $\Pi_{\mathcal{R}} = \Pi_{Eq}$  then  $\approx$  is given by  $\{ (s, t) \in S \times S \mid [s]_{\Pi_{Eq}} = [t]_{\Pi_{Eq}} \}$ . The *initial approximation*  $\Pi_0$  is denoted by the *AP-Partition*  $\Pi_{AP}$ .

#### DEFINITION 4.3.2 ( ABSTRACTION OF THE STUTTER SIMULATION )

Let  $\mathcal{R}$  be a binary relation on  $S$ ,  $\Pi_{\mathcal{R}}$  be the partition of  $S$  w.r.t.  $\mathcal{R}$  and  $B, C \in \Pi_{\mathcal{R}}$ . Then,  $\mathcal{R}$  is induced by the binary relation  $\leq_{\mathcal{R}} \subseteq \Pi_{\mathcal{R}} \times \Pi_{\mathcal{R}}$ , where

$$B \leq_{\mathcal{R}} C \text{ if and only if } \forall s \in B. \forall t \in C. (s, t) \in \mathcal{R}.$$

If  $\mathcal{R}$  is the stutter simulation preorder  $\leq$  then  $\leq_{\mathcal{R}}$  induces the coarsest stutter simulation preorder and will be denoted by  $\leq_{Pre}$ . ■

Similar to Definition 4.3.1, the relation  $\mathcal{R}$  will indicate the current approximation of the stutter simulation preorder  $\leq$  according to the computations of Algorithm 4.3.1 and we have that block  $C \in \Pi_{\mathcal{R}}$  is currently candidate to stutter simulate block  $B \in \Pi_{\mathcal{R}}$  iff all states in  $C$  are currently candidates to stutter simulate all states in  $B$ , i.e.  $(s, t) \in \mathcal{R}$  for all  $s \in B, t \in C$ . The *initial approximation*  $\leq_0$  is induced by the state-labelings, such that  $[s]_{\Pi_{AP}} \leq_0 [t]_{\Pi_{AP}}$  iff  $[s]_{\Pi_{AP}} = [t]_{\Pi_{AP}}$  iff  $L(s) = L(t)$  and will be denoted by  $\leq_{AP}$ . Recall that whenever a state  $s$  is contained in an arbitrary block  $D \in \Pi$  then  $[s]_{\Pi} = D$ , since  $\Pi$  is a partition. Consequently,  $[s]_{\Pi_{Eq}} \leq_{Pre} [t]_{\Pi_{Eq}}$  iff  $s \leq t$  for all  $s \in [s]_{\Pi_{Eq}}, t \in [t]_{\Pi_{Eq}}$ . The stutter simulation preorder  $\leq$  as a relation between pairs of states thus can be obtained from  $\leq_{Pre}$  and  $\Pi_{Eq}$  as follows:

$$\leq = \{ (s, t) \in S \times S \mid [s]_{\Pi_{Eq}} \leq_{Pre} [t]_{\Pi_{Eq}} \}.$$

Since the stutter simulation preorder  $\leq \subseteq S \times S$  is now induced by  $\leq_{Pre}$  as a relation between sets of states, we define the stutter simulator set of a set of states.

**DEFINITION 4.3.3 ( STUTTER SIMULATOR SET OF A SET OF STATES )**

Let  $\mathcal{P} = (\Pi_{\mathcal{R}}, \leq_{\mathcal{R}})$  be a partition-relation pair, where  $\Pi_{\mathcal{R}}$  and  $\leq_{\mathcal{R}}$  encode the current approximations of  $\cong$  and  $\leq$ , respectively. Then, the stutter-simulator set of  $D \subseteq S$  w.r.t.  $\mathcal{P}$  is defined as

$$StSim_{\mathcal{P}}(D) := \bigcup \{ C \in \Pi_{\mathcal{R}} \mid \exists B \in \Pi_{\mathcal{R}}. B \cap D \neq \emptyset \wedge B \leq_{\mathcal{R}} C \}.$$

The set of stutter simulator sets  $StSim_{\mathcal{P}}(D)$  of all blocks  $D \in \Pi_{\mathcal{R}}$  is denoted by  $StSim_{\mathcal{P}}$ . ■

If  $\mathcal{P} = (\Pi_{Eq}, \leq_{Pre})$  and  $D \in \Pi_{Eq}$  then  $StSim_{\mathcal{P}}(D)$  is the set of blocks that stutter simulate  $D$  and will be denoted by  $StSim_{TS}(D)$ . Note that, if  $D$  is a block in  $\Pi_{\mathcal{R}}$ , the set of blocks that are candidates to stutter simulate  $D$  is identified by  $StSim_{\mathcal{P}}(D) = \bigcup \{ C \in \Pi_{\mathcal{R}} \mid D \leq_{\mathcal{R}} C \}$ , such that each state in  $C$  is candidate to stutter simulate each state in  $D$ . Furthermore, if  $D$  is a singleton, i.e.  $D = \{ s \}$ , then the corresponding stutter simulator set specifies the set of states that are currently candidates to stutter simulate  $s$  and is obtained by

$$\begin{aligned} StSim_{\mathcal{P}}(\{ s \}) &= StSim_{\mathcal{P}}([s]_{\Pi_{\mathcal{R}}}) \\ &= \bigcup \{ C \in \Pi_{\mathcal{R}} \mid \exists B \in \Pi_{\mathcal{R}}. [s]_{\Pi_{\mathcal{R}}} \cap B \neq \emptyset \wedge B \leq_{\mathcal{R}} C \} \\ &= \bigcup \{ C \in \Pi_{\mathcal{R}} \mid [s]_{\Pi_{\mathcal{R}}} \leq_{\mathcal{R}} C \}. \end{aligned}$$

Recall that, since  $[s]_{\Pi_{\mathcal{R}}}$  specifies the unique block in  $\Pi_{\mathcal{R}}$  containing  $s$  and since  $\Pi_{\mathcal{R}}$  is a partition, there exists exactly one block  $B$  with  $B \cap [s]_{\Pi_{\mathcal{R}}} \neq \emptyset$  and furthermore,  $B = [s]_{\Pi_{\mathcal{R}}}$ . The current approximation of the stutter simulation preorder  $\leq$  as a relation between pairs of states is thus given by:

$$\{ (s, t) \in S \times S \mid [t]_{\Pi_{\mathcal{R}}} \in StSim_{\mathcal{P}}([s]_{\Pi_{\mathcal{R}}}) \} = \{ (s, t) \in S \times S \mid [s]_{\Pi_{\mathcal{R}}} \leq [t]_{\Pi_{\mathcal{R}}} \}.$$

**DEFINITION 4.3.4 ( REFINER, SPLITTER, STABILITY )**

Let  $\mathcal{P} = (\Pi_{\mathcal{R}}, \leq_{\mathcal{R}})$  be a partition-relation pair, where  $\Pi_{\mathcal{R}}$  and  $\leq_{\mathcal{R}}$  induce the current approximations of  $\cong$  and  $\leq$ , respectively,  $StSim_{\mathcal{P}}$  the corresponding set of stutter simulator sets and  $B, C, D \in \Pi_{\mathcal{R}}$ .

- (1) The pair  $(B, C)$  is a *refiner* for  $StSim_{\mathcal{P}}$  if and only if  $\exists s \in B. \exists s' \in C. s' \in Post(s)$  and  $StSim_{\mathcal{P}}(B) \not\subseteq Succ^*(StSim_{\mathcal{P}}(B), StSim_{\mathcal{P}}(C))$ .
- (2)  $StSim_{\mathcal{P}}(B)$  is *stable* if there exists no  $C \in \Pi_{\mathcal{R}}$  such that  $(B, C)$  is a refiner for  $StSim_{\mathcal{P}}(B)$ .
- (3)  $StSim_{\mathcal{P}}$  is *stable* if  $StSim_{\mathcal{P}}(B)$  is stable for all  $B \in \Pi_{\mathcal{R}}$ .
- (4) The pair  $(B, C)$  is a *splitter* for  $\Pi_{\mathcal{R}}$  if and only if  $\exists D \in StSim_{\mathcal{P}}(B). \exists t_1, t_2 \in D$  such that  $t_1 \in Succ^*(StSim_{\mathcal{P}}(B), StSim_{\mathcal{P}}(C))$  but  $t_2 \notin Succ^*(StSim_{\mathcal{P}}(B), StSim_{\mathcal{P}}(C))$ .

■

The initial approximations of  $\cong$  and  $\leq$  are overapproximations that are successively refined. Therefore, we have to check if there exists a state  $s$ , such that the *current* stutter simulator set of  $[s]_{\Pi_{\mathcal{R}}}$  includes state  $t$  of some block  $D \in \Pi_{\mathcal{R}}$  which cannot stutter mimic some behavior of  $s$ , i.e.  $t \notin Succ^*(StSim_{\mathcal{P}}([s]_{\Pi_{\mathcal{R}}}), StSim_{\mathcal{P}}([s']_{\Pi_{\mathcal{R}}}))$  but  $s$  has an outgoing transition to a state in  $[s']_{\Pi_{\mathcal{R}}} \in \Pi_{\mathcal{R}}$ . In this sense, we will basically *split* each such state  $t$  from  $D$  and group them in a new (generated) block  $D_2 \notin StSim_{\mathcal{P}}([s]_{\Pi_{\mathcal{R}}})$ , such that all states remaining in  $D$

provide a finite path fragment that leads to  $[s']_{\Pi_{\mathcal{R}}}$  by only visiting states in  $StSim_{\mathcal{P}}([s]_{\Pi_{\mathcal{R}}})$ . To avoid any confusion, we denote this set of states by  $D_1 = D \setminus D_2$ . Note that the splitting of  $D$  is necessary, since the states in  $D_2$  are no longer candidates to be stutter simulation equivalent to the states left in  $D$ . The splitting of blocks clearly involves an update of the stutter simulation relation, where  $D_1$  is candidate to stutter simulate  $D_2$ , whereas the reverse does not hold. Furthermore, the stutter simulator sets of all other candidates  $C \subseteq Succ^*(StSim_{\mathcal{P}}([s]_{\Pi_{\mathcal{R}}}), StSim_{\mathcal{P}}([s']_{\Pi_{\mathcal{R}}}))$  need to be refined accordingly.

**LEMMA 4.3.1 ( STUTTER SIMULATION )**

Let  $TS$  be a transition system,  $\mathcal{R}$  a binary relation on  $S$  and  $\mathcal{P} = (\Pi_{\mathcal{R}}, \leq_{\mathcal{R}})$  a partition-relation pair, where  $\Pi_{\mathcal{R}}$  is a partition of  $S$  w.r.t.  $\mathcal{R}$  and  $\leq_{\mathcal{R}} \subseteq \Pi_{\mathcal{R}} \times \Pi_{\mathcal{R}}$  induces  $\mathcal{R}$ . Then:

$$\begin{aligned} &\mathcal{R} \text{ is a stutter simulation for } (TS, TS) \\ &\text{if and only if for all } s \in S \\ &\quad StSim_{\mathcal{P}}([s]_{\Pi_{\mathcal{R}}}) \subseteq [s]_{\Pi_{AP}} \text{ and} \\ &\quad \text{if } \exists s' \in Post(s) \text{ then } StSim_{\mathcal{P}}([s]_{\Pi_{\mathcal{R}}}) \subseteq Succ^*(StSim([s]_{\Pi_{\mathcal{R}}}), StSim([s']_{\Pi_{\mathcal{R}}})) \end{aligned}$$

PROOF The proof is analogous to the one of Lemma 4.2.1. ■

Consequently, our algorithm has to check in every iteration whether there exists a state in  $StSim_{\mathcal{P}}(B)$  that violates Lemma 4.3.1. If this is the case, the corresponding blocks in  $\Pi_{\mathcal{R}}$  will be split into two blocks each, whereas  $StSim_{\mathcal{P}}$  will be refined in a similar way as specified by the refinement operation of the algorithm to solely computing the stutter simulation preorder (see Definition 4.2.3). These two operations form the heart of Algorithm 4.3.1 and will be precisely defined, explained and illustrated in the following. (Note that  $Sat(B, C)$  is an abbreviation for the set of constrained successors  $Succ^*(StSim_{\mathcal{P}}(B), StSim_{\mathcal{P}}(C))$ .)

**DEFINITION 4.3.5 ( REFINEMENT AND SPLITTING OPERATION )**

Let  $(B, C)$  be a refiner for  $StSim_{\mathcal{P}}$  and a splitter for  $\Pi_{\mathcal{R}}$ , where  $\mathcal{P} = (\Pi_{\mathcal{R}}, \leq_{\mathcal{R}})$  is the current partition relation pair and let  $Sat(B, C) = Succ^*(StSim_{\mathcal{P}}(B), StSim_{\mathcal{P}}(C))$  be the set of states in  $StSim_{\mathcal{P}}(B)$  that reach a state in  $StSim_{\mathcal{P}}(C)$  by exclusively visiting states in  $StSim_{\mathcal{P}}(B)$ . The splitting operation is defined by:

(1) Splitting Operation:

$$Split(\Pi_{\mathcal{R}}, Sat(B, C)) := \bigcup_{D \in \Pi_{\mathcal{R}}} Split(D, Sat(B, C)),$$

where

$$Split(D, Sat(B, C)) := \begin{cases} \underbrace{\{D \cap Sat(B, C)\}}_{D_1}, \underbrace{\{D \setminus Sat(B, C)\}}_{D_2}, & \text{if } D_1 \neq \emptyset \neq D_2 \\ \{D\}, & \text{otherwise.} \end{cases}$$

(2) Refinement Operation:

Let  $\mathcal{P}' = (\Pi'_{\mathcal{R}}, \leq_{\mathcal{R}})$  be the partition relation pair obtained by splitting all blocks in  $\Pi_{\mathcal{R}}$  w.r.t.  $Sat(B, C)$ . The set of stutter simulator sets  $StSim_{\mathcal{P}'}$  is accordingly updated as follows:

$$\text{Refine}(\text{StSim}_{\mathcal{P}}, \text{Sat}(B, C)) := \bigcup_{D' \in \Pi_{\mathcal{R}}} \{ \text{Refine}(\text{StSim}_{\mathcal{P}}(D'), \text{Sat}(B, C)) \},$$

where

$$\text{Refine}(\text{StSim}_{\mathcal{P}}(D'), \text{Sat}(B, C)) := \begin{cases} \text{StSim}_{\mathcal{P}}(D) \cap \text{Sat}(B, C), & \text{if } D' = D_1 \subsetneq D \in \Pi_{\mathcal{R}} \text{ and } D' \subseteq \text{Sat}(B, C) \text{ or} \\ & \text{if } D' = D \in \Pi_{\mathcal{R}} \text{ and } D \subseteq \text{Sat}(B, C) \\ \text{StSim}_{\mathcal{P}}(D), & \text{if } D' = D_2 \subsetneq D \in \Pi_{\mathcal{R}} \text{ and } D' \cap \text{Sat}(B, C) = \emptyset \text{ or} \\ & \text{if } D' = D \in \Pi_{\mathcal{R}} \text{ and } D \cap \text{Sat}(B, C) = \emptyset. \end{cases}$$

■

Given the situation in Figure 4.3, where  $\Pi = \{ B, C, D, E, F \}$ ,  $\text{StSim}_{\mathcal{P}}(B) = B \cup D \cup E \cup F$ ,  $\text{StSim}_{\mathcal{P}}(C) = C$ ,  $\text{StSim}_{\mathcal{P}}(D) = D \cup F$ ,  $\text{StSim}_{\mathcal{P}}(E) = D \cup E$ ,  $\text{StSim}_{\mathcal{P}}(F) = F$ , and we assume the pair  $(B, C) \in \Pi_{\mathcal{R}} \times \Pi_{\mathcal{R}}$  is a refiner for  $\text{StSim}_{\mathcal{P}}$ . Then, there exists a state  $s \in B$ ,  $s' \in C$  and  $t_2 \in \text{StSim}_{\mathcal{P}}(B)$ , such that  $s \rightarrow s'$  (solid arrow from  $s$  to  $s'$ ) and  $t_2 \notin \text{Succ}^*(\text{StSim}_{\mathcal{P}}(B), \text{StSim}_{\mathcal{P}}(C))$  (crossed out dotted arrow). Furthermore, we assume that  $t_2$  is included in block  $D$  and there exist states  $t_1 \in D$ ,  $v \in F$  and  $u \in E$  which belong to  $\text{Succ}^*(\text{StSim}_{\mathcal{P}}(B), \text{StSim}_{\mathcal{P}}(C))$  (dotted arrow), i.e. which stutter simulate the behavior of  $s$  with respect to block  $C$ . In Section 4.2

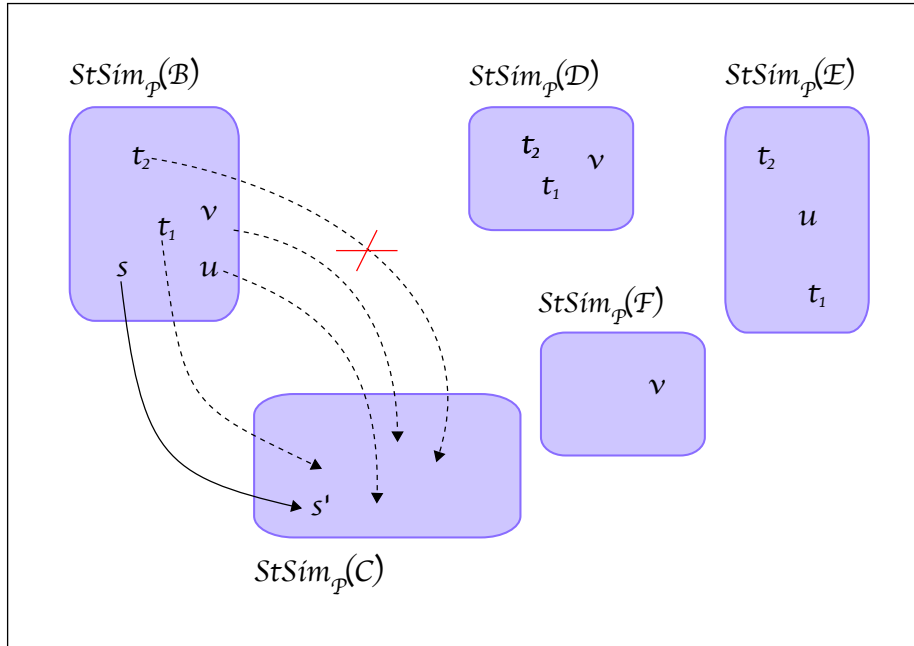


Figure 4.3: Current approximation of  $\approx$  and  $\leq$

the refinement operation removed  $t_2$  from  $\text{StSim}_{\mathcal{R}}(s)$  and simultaneously from all stutter simulator sets of all states in  $\text{Succ}^*(\text{StSim}_{\mathcal{R}}(s), \text{StSim}_{\mathcal{R}}(s'))$ . Since  $\text{StSim}_{\mathcal{P}}$  now is defined over blocks in  $\Pi_{\mathcal{R}}$  instead of single states in  $S$ , we proceed as follows: Split block  $D$  into the subblocks  $D_1 = D \cap \text{Succ}^*(\text{StSim}_{\mathcal{P}}(B), \text{StSim}_{\mathcal{P}}(C))$  and  $D_2 = D \setminus \text{Succ}^*(\text{StSim}_{\mathcal{P}}(B), \text{StSim}_{\mathcal{P}}(C))$ ,

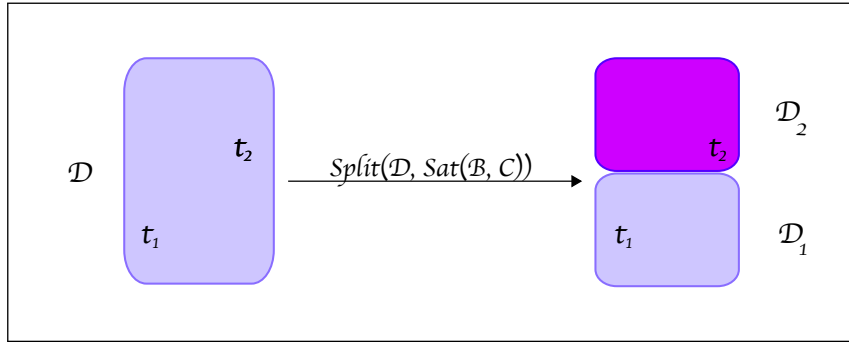


Figure 4.4: Splitting Operation

where  $t_1$  is included in  $D_1$  and  $t_2$  in  $D_2$  after the splitting (see Figure 4.4). Then, the refinement operation excludes  $D_2$  from  $StSim_{\varphi}(B)$  and simultaneously from all stutter simulator sets of all blocks (in the new partition obtained by applying the splitting operation to **all** blocks that need to be split) that are a subset of  $Succ^*(StSim_{\varphi}(B), StSim_{\varphi}(C))$ . Thus,  $t_2$  is excluded from  $StSim_{\varphi}(B)$  but also from the stutter simulator set of  $E$ . Since  $D_2$  is a new generated block, we must define its stutter simulator set. This will be the stutter simulator set of the *old* (that is, *before* splitting) block  $D$ , which can be seen as follows. The refiner yields that the states in  $D_1$  exhibit some behavior not stutter simulatable by  $D_2$ , whereas the reverse is *not disproven* yet. The same argumentation holds for all other blocks (which are here blocks  $F$  and  $D_2$  itself) in  $StSim_{\varphi}(D)$  and hence, they are still *candidates* to stutter simulate all states in  $D_2$  (compare to Figure 4.5).

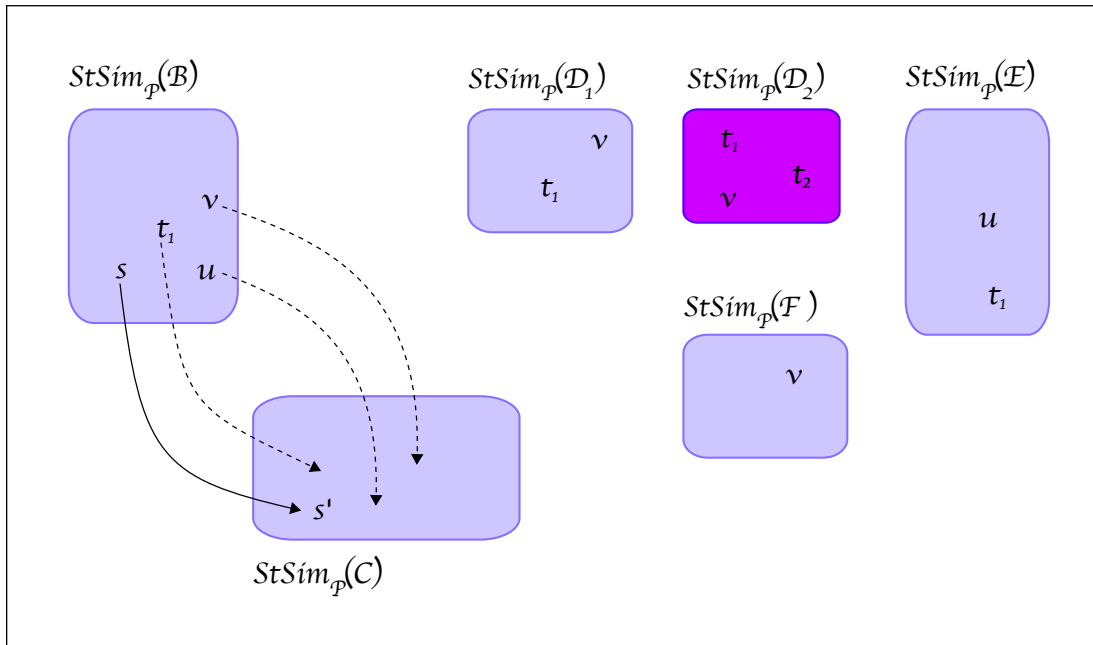


Figure 4.5: Refinement Operation

Note that in case  $D_1 = \emptyset$  no state in  $D$  can reach a state in  $StSim_{\mathcal{P}}(C)$  via a path through states in  $StSim_{\mathcal{P}}(B)$ . In this case, the splitter-conditions in Definition 4.3.4 are violated and no splitting takes place, whereas the refinement operation processes as usual, i.e. excluding the states in block  $D$  from  $StSim_{\mathcal{P}}(B)$  and from all stutter simulator sets of all blocks in  $Succ^*(StSim_{\mathcal{P}}(s), StSim_{\mathcal{P}}(s'))$ .

**LEMMA 4.3.2 ( CORRECTNESS OF SPLITTING AND REFINEMENT OPERATION )**

Let  $\mathcal{P} = (\Pi_{\mathcal{R}_i}, \preceq_{\mathcal{R}_i})$  be the current partition relation pair. The current approximation  $\Pi_{\mathcal{R}_i}$  of the stutter simulation equivalence  $\cong$  is finer than  $\Pi_{AP}$ , coarser than  $\Pi_{TS}$  and  $\Pi_{\mathcal{R}_0} = \Pi_{AP}$ . The relation  $\preceq_{\mathcal{R}_i}$  is the approximation of the stutter simulation preorder  $\preceq$ , which is finer than  $\preceq_{AP}$ , coarser than  $\preceq_{TS}$ , and where  $\preceq_{\mathcal{R}_0} = \preceq_{AP}$ .

- If  $(B, C)$  is a splitter for  $\Pi_{\mathcal{R}_i}$  then

$$\Pi_{\mathcal{R}_{i+1}} = Split(\Pi_{\mathcal{R}_i}, Sat(B, C)) \text{ is (1.1) finer than } \Pi_{\mathcal{R}_i} \text{ and (1.2) coarser than } \Pi_{Eq}.$$

- If  $(B, C)$  is a refiner for  $StSim_{\mathcal{P}_i}$  then

$$StSim_{\mathcal{P}_{i+1}} = Refine(StSim_{\mathcal{P}_i}, Sat(B, C)) \text{ is (2.1) finer than } StSim_{\mathcal{P}_i} \text{ and} \\ \text{(2.2) coarser than } StSim_{TS}.$$

**PROOF** We first show the correctness of claims (1.1) and (2.1). Let  $\Pi_{\mathcal{R}_i}$  be the current approximation of the stutter simulation equivalence  $\cong$  in the beginning of the  $i$ th splitting operation. The proof is by induction on  $i$ .

- (1) Base case:  $i = 0$ . Since  $\Pi_{\mathcal{R}_0} = \Pi_{AP}$ , claim (1.2) follows immediately. Furthermore, since no splitting took place yet, claim (1.1) trivially holds.
- (2) Assume  $i > 0$ , where  $\Pi_{\mathcal{R}_i}$  is finer than  $\Pi_{\mathcal{R}_{i-1}}$  and coarser than  $\Pi_{Eq}$ . Let  $(B, C)$  be a splitter for  $\Pi_{\mathcal{R}_i}$  and let  $\Pi_{\mathcal{R}_{i+1}} = Split(\Pi_{\mathcal{R}_i}, Sat(B, C))$ .
  - (2.1) By definition of the splitting operation, a block  $D \in \Pi_{\mathcal{R}_i}$  can only be split by  $Sat(B, C)$  into two new subblocks if neither  $D \cap Sat(B, C)$  nor  $D \setminus Sat(B, C)$  yields the empty set. Otherwise,  $D$  remains the same and hence, it directly follows that  $\Pi_{\mathcal{R}_{i+1}}$  is finer than  $\Pi_{\mathcal{R}_i}$ .
  - (2.2) For claim (1.2) we have to show that  $Split([s]_{\Pi_{\mathcal{R}_i}}, Sat(B, C))$  yields a set  $[s]_{\Pi_{\mathcal{R}_{i+1}}} \subseteq [s]_{\Pi_{Eq}}$  for all states  $s \in S$ . Distinguish between two cases.
    - (2.2.1) Either  $[s]_{\mathcal{R}_i} \cap Sat(B, C) = \emptyset$  or  $[s]_{\mathcal{R}_i} \setminus Sat(B, C) = \emptyset$ . In both cases we have that  $[s]_{\mathcal{R}_{i+1}} = Split([s]_{\mathcal{R}_i}, Sat(B, C))$ , i.e.  $[s]_{\mathcal{R}_i} = [s]_{\mathcal{R}_{i+1}}$  and since  $[s]_{\mathcal{R}_i} \subseteq [s]_{\Pi_{Eq}}$ , it holds that  $Split([s]_{\mathcal{R}_i}, Sat(B, C)) = [s]_{\mathcal{R}_{i+1}} \subseteq [s]_{\Pi_{Eq}}$ .
    - (2.2.2)  $[s]_{\mathcal{R}_i} \cap Sat(B, C) \neq \emptyset$  and  $[s]_{\mathcal{R}_i} \setminus Sat(B, C) \neq \emptyset$ . Then, either  $[s]_{\mathcal{R}_{i+1}} = [s]_{\mathcal{R}_i} \cap Sat(B, C)$  or  $[s]_{\mathcal{R}_{i+1}} = [s]_{\mathcal{R}_i} \setminus Sat(B, C)$ , since  $s$  either belongs to  $Sat(B, C)$  or not. The proof is by contradiction. Assume that there exists a state  $u$  that is contained in  $[s]_{\mathcal{R}_i}$  and  $[s]_{\Pi_{Eq}}$  but not in  $[s]_{\mathcal{R}_{i+1}}$ . Furthermore, assume w.l.o.g. that  $[s]_{\mathcal{R}_{i+1}} = [s]_{\mathcal{R}_i} \cap Sat(B, C)$ . Then,  $s \in Sat(B, C)$  and since  $u \notin [s]_{\mathcal{R}_{i+1}}$  and by definition of the splitting operation, we have that  $u \notin Sat(B, C)$ . It immediately follows that  $s$  and  $u$  are not stutter simulation equivalent, which contradicts the assumption that  $u \in [s]_{\Pi_{Eq}}$  and it holds that  $[s]_{\Pi_{\mathcal{R}_{i+1}}} \subseteq [s]_{\Pi_{Eq}}$ .

Hence,  $\Pi_{\mathcal{R}_{i+1}}$  is coarser than  $\Pi_{Eq}$ .

The proofs of claims (2.1) and (2.2) are analogous to those in Lemma 4.2.2. Observe that here the refinement operation is applied to the blocks in the *new* partition  $\Pi_{\mathcal{R}_{i+1}}$ , i.e. we first split the blocks by the splitter  $(B, C)$  and then refine the relation  $\leq_{\mathcal{R}_i}$  between the blocks in  $\Pi_{\mathcal{R}_{i+1}}$  w.r.t. the refiner  $(B, C) \in \Pi_{\mathcal{R}_i} \times \Pi_{\mathcal{R}_i}$ . This is correct since the splitting operation is correct and we obtain the correctness of both the splitting and the refinement operation. ■

The following Algorithm 4.3.1 outlines the basic steps to compute the coarsest stutter simulation preorder and equivalence of a transition system  $TS = (S, Act, \rightarrow, I, AP, L)$  with possibly terminal states.

---

**Algorithm 4.3.1:** Computing the Stutter Simulation Preorder and Equivalence
 

---

**Input** : Transition system  $TS$

**Output**: Stutter simulation equivalence  $\Pi_{Eq}$  and preorder  $StSim_{TS}$

---

```

1   $\Pi := \Pi_{AP};$  (* initialization *)
2  forall  $B \in \Pi$  do
3  |  $StSim(B) := B;$ 
4  while  $\exists s \in S. \exists s' \in Post(s)$  with  $StSim([s]_{\Pi}) \not\subseteq Succ^*(StSim([s]_{\Pi}), StSim([s']_{\Pi}))$  do
5  | compute  $Sat([s]_{\Pi}, [s']_{\Pi});$  (*  $\exists u \in StSim([s]_{\Pi})$  with  $s \not\leq u$  *)
6  |  $\Pi_{old} := \Pi;$ 
7  | forall  $D \in \Pi_{old}$  with  $D \cap Sat([s]_{\Pi_{old}}, [s']_{\Pi_{old}}) \neq \emptyset$  and  $D \setminus Sat([s]_{\Pi_{old}}, [s']_{\Pi_{old}}) \neq \emptyset$  do
8  | | pick  $D' \in \Pi$  with  $D' = D;$ 
9  | |  $D' := D \cap Sat([s]_{\Pi_{old}}, [s']_{\Pi_{old}});$ 
10 | |  $\Pi := \Pi \cup D \setminus Sat([s]_{\Pi_{old}}, [s']_{\Pi_{old}});$  (* splitting operation *)
11 | forall  $D' \in \Pi$  do
12 | |  $Refine(StSim(D'), Sat([s]_{\Pi_{old}}, [s']_{\Pi_{old}}));$  (* refinement operation *)
(*  $StSim([s]_{\Pi}) = StSim_{TS}([s]_{\Pi_{Eq}})$  for any  $s$  *)
(*  $[s]_{\Pi} = [s]_{\Pi_{Eq}}$  for any  $s$  *)
13 return  $(\Pi, \{ StSim(B) \}_{B \in \Pi});$  (*  $\bigcup_{B \in \Pi} \{ StSim(B) \} = StSim_{TS}$  *)
    
```

---

The assignment  $\Pi := \Pi_{AP}$  denotes the computation of the  $AP$ -partition according to Baier and Katoen [1]. In the beginning of the *while*-loop (ll. 4-12), the sets  $StSim$  and  $\Pi$  represent the current approximations of the coarsest stutter simulation preorder and equivalence, respectively. A block  $B \in \Pi$  is candidate to stutter simulate block  $C \in \Pi$  iff  $B \subseteq StSim(C)$ . Furthermore, states  $s$  and  $t$  are candidates to be stutter simulation equivalent iff they belong to the same block in  $\Pi$ . The *forall*-loop (ll. 7-10) implements the splitting operation, whereas ll. 11-12 represent the refinement operation. Note that the splitting precedes the refinement operation, since otherwise newly generated blocks resulting from splitting are not included in the  $StSim$ -relation. Since the state space of the given transition system is finite and thanks to Lemma 4.3.2, Algorithm 4.3.1 terminates after  $n \in \mathbb{N}_0$  iterations of the *while*-loop on input  $TS$ . We will now prove the correctness of Algorithm 4.3.1.

**THEOREM 4.3.1 ( CORRECTNESS OF ALGORITHM 4.3.1 )**

Let  $TS$  be a transition system and  $(\Pi, StSim)$  be the output of Algorithm 4.3.1. Then, for all  $s, t \in S$  it holds that

- (1)  $t \in StSim([s]_{\Pi})$  if and only if  $s \leq t$  and
- (2)  $[s]_{\Pi} = [t]_{\Pi}$  if and only if  $s \approx t$ .

**PROOF** The proof of the first claim is similar to the one of Theorem 4.2.1. To show claim (2), let  $TS$  be some transition system and  $(\Pi, StSim)$  be the output of Algorithm 4.3.1 after  $n \geq 0$  iterations of the *while*-loop.

$\Rightarrow$ : Assume  $[s]_{\Pi} = [t]_{\Pi}$ , where  $s, t$  are two states in  $S$ . Then,  $StSim([s]_{\Pi}) = StSim([t]_{\Pi})$  and since  $StSim$  represents the coarsest stutter simulation preorder, it follows that  $s \leq t$  and  $t \leq s$  and hence,  $s \approx t$ .

$\Leftarrow$ : Assume  $s \approx t$ , where  $s, t$  are two states in  $S$ . By Lemma 4.3.2, we have that  $\Pi$  is coarser than  $\Pi_{Eq}$  and hence, it cannot be the case that  $s$  and  $t$  belong to different blocks in  $\Pi$ . It follows directly that  $[s]_{\Pi} = [t]_{\Pi}$ . ■

**THEOREM 4.3.2 ( TIME COMPLEXITY OF ALGORITHM 4.3.1 )**

The stutter simulation quotient space can be computed by Algorithm 4.3.1 with time complexity  $\mathcal{O}(|S| \cdot |AP| + |\rightarrow| \cdot |S|^5)$ .

**PROOF** Let  $n = |S|$  denote the number of states,  $m = |\rightarrow|$  the number of transitions, where we assume that  $m \geq n$ .

- (1) Initialization (ll.1-3):

Following Baier and Katoen [1] (p.478ff), computing the initial partition  $\Pi_{AP}$  and initial approximation of the stutter simulation preorder  $StSim$  takes time  $\mathcal{O}(n \cdot |AP|)$ , where  $AP$  is the set of atomic propositions in  $TS$ .

- (2) The *while*-Loop (ll.4-12):

- (2.1) Number of Executions of the *while*-loop:

Similar to Algorithm 4.2.1, the number of iterations of the *while*-loop is bounded by  $\mathcal{O}(n^2)$ .

- (2.2) Checking the Guard of the Loop (l.4):

The loop-condition in l.4 is checked for each successor  $s'$  of every state  $s$ , i.e.  $\sum_{s \in S} |Post(s)| \in \mathcal{O}(m)$  times in total.

- (2.2.1) Computing the Set of Constrained Successors:

The proceeding goes analogous to Algorithm 4.2.1, where we computed the set of constrained successors using Floyd-Warshall [2] with a slight modification in time  $\mathcal{O}(n^3)$ . Note that determining the blocks states  $s$  and  $s'$  belong to takes time  $\mathcal{O}(n)$  each in the worst case if  $\Pi$  and the blocks it contains are represented by lists. Furthermore, if  $StSim$  is given as a list of lists then identifying the stutter simulator set of a block can be done in time  $\mathcal{O}(n)$ . Hence,  $\mathcal{O}(n^3 + 2n + 2n) = \mathcal{O}(n^3)$  is an upper bound to computing the set of constrained successors  $Sat([s]_{\Pi}, [s']_{\Pi})$  of  $StSim([s]_{\Pi})$  w.r.t.  $StSim([s']_{\Pi})$ .

## (2.2.2) Testing the Subset-Relation:

According to Section (2.2.2) of Theorem 4.2.2, this can be done in time  $O(2n) = O(n)$  using a hash table  $H_1$  for the set  $Sat([s]_{\Pi}, [s']_{\Pi})$ .

Thus, checking the condition of the *while*-loop for a single pair of states takes time  $O(n^3)$ .

## (2.3) Loop-Body (ll.5-11):

Note that the set  $Sat([s]_{\Pi_{old}}, [s']_{\Pi_{old}})$  is the same set  $Sat([s]_{\Pi}, [s']_{\Pi})$  which was already computed in the guard of the *while*-loop (l.4).

## (2.3.1) Splitting Operation (ll.7-10):

Here, the list  $\Pi_{old}$  is sequentially traversed. To check whether the guard of the first *forall*-loop is satisfied for the current block  $D \in \Pi_{old}$ , we compute the sets  $D_1 = D \cap Sat([s]_{\Pi_{old}}, [s']_{\Pi_{old}})$  and  $D_2 = D \setminus Sat([s]_{\Pi_{old}}, [s']_{\Pi_{old}})$  by reusing hash table  $H_1$  and testing for each state  $s \in D$  whether it has an entry in  $H_1$  (such that  $s$  belongs to  $D_1$ ) or not (and hence is a member of  $D_2$ ) in time  $O(n)$ . Defining boolean variables  $empty_i$  ( $i \in \{1, 2\}$ ), where  $empty_i$  is set to *false* once a state is inserted in set  $D_i$ , enables to checking whether  $D_1$  or  $D_2$  is the empty set in time  $O(1)$ . In this case, no splitting is possible for block  $D$ . Otherwise, the body of the *forall*-loop is entered, where block  $D' \in \Pi$  with  $D' = D$  is searched and assigned to  $D_1$  (in time  $O(n)$ ). Inserting  $D_2$  in the current partition  $\Pi$  takes time  $O(1)$ . We clearly must not split the blocks in the original partition  $\Pi_{old}$ , since the *forall*-loop iterates over those blocks that are candidates to be split w.r.t the current set of constrained successors. Note that if  $|\Pi_{old}| = n$  then each block in  $\Pi_{old}$  consists of exactly one single state and no splitting can take place. Thus, the cardinality of  $|\Pi_{old}|$  will be strictly smaller than  $n$ , such that the number of iterations of the first *forall*-loop is in  $O(n)$ . Furthermore, it follows that per execution of the *while*-loop the total costs of this loop are bounded by  $O(n \cdot (n + 1 + n + 1)) = O(2n^2 + 2n) = O(n^2)$ .

## (2.3.2) Refinement Operation (ll. 11-12):

The second *forall*-loop sequentially traverses the list  $\Pi$  (which was possibly updated by the splitting operation). Here, we check for each block  $D' \in \Pi$  whether it is a subset of  $Sat([s]_{\Pi_{old}}, [s']_{\Pi_{old}})$ . To that end, it suffices to pick a single state in  $D'$  and to test whether it has a correspondent in  $H_1$  (in  $O(1)$ ), since the splitting operation decomposed each block containing states included *and* not included in  $Sat([s]_{\Pi_{old}}, [s']_{\Pi_{old}})$ . Thus, if  $s$  is an arbitrary state in  $D'$  then  $D' \subseteq Sat([s]_{\Pi_{old}}, [s']_{\Pi_{old}})$  iff  $s \in Sat([s]_{\Pi_{old}}, [s']_{\Pi_{old}})$ .

 (2.3.2.1)  $D' \subseteq Sat([s]_{\Pi_{old}}, [s']_{\Pi_{old}})$ :

According to the definition of the refinement operation,  $D'$  either is a block in or a real subset of a block in the old partition  $\Pi_{old}$ . In both cases, the stutter simulator set of  $D'$  is assigned to  $StSim(D) \cap Sat([s]_{\Pi_{old}}, [s']_{\Pi_{old}})$ . Similar to Theorem 4.2.2, the intersection can be computed in time  $O(n)$ . Under the assumption that the relation  $StSim$  is stored as a list of lists, it takes time  $O(n)$  to find the stutter simulator set of some block  $D$ . Thus, in case  $D' \subseteq Sat([s]_{\Pi_{old}}, [s']_{\Pi_{old}})$ ,  $O(1 + 3n) = O(n)$  is an upper bound for

refining a single block  $D' \in \Pi$ .

$$(2.3.2.2) \quad D' \cap \text{Sat}([s]_{\Pi_{old}}, [s']_{\Pi_{old}}) = \emptyset:$$

Following Definition 4.3.5, if  $D'$  is not a subset of the set of constrained successors then a refinement only takes place if it is a newly generated block resulting from the splitting operation of some block  $D$  in the old partition, i.e. if  $D' \subseteq D$  where  $D \in \Pi_{old}$ . Here,  $D'$  is not yet included in the relation  $StSim$  and we need to assign  $StSim(D') := StSim(D)$ . To determine the desired block  $D$  algorithmically, all elements in  $D'$  are stored in hash table  $H_2$  (in  $O(n)$ ) and each block in  $\Pi_{old}$  is sequentially traversed (in  $O(n)$ ). Then, once a state in a block in  $\Pi_{old}$ ,  $D$  say, has a correspondent in  $H_2$ , the search is aborted, since  $\Pi_{old}$  is a partition and it remains to test whether *each* state in  $D$  has an entry in  $H_2$  (in  $O(n)$ ). If the answer is negative then  $D' \subsetneq D$  and  $StSim(D)$  is set as the stutter simulator set of  $D$  (in  $O(1)$ ). Hence, in case  $D' \cap \text{Sat}([s]_{\Pi_{old}}, [s']_{\Pi_{old}}) = \emptyset$ , the refinement operation (for a single block) takes time  $O(1 + 3n + 1) = O(n)$ .

Since both cases effort basically the same amount of time, we can conclude that the refinement operation is bounded by  $O(n^2)$ , since the number of blocks in the current partition  $\Pi$  is at most  $n$ .

The total complexity of Algorithm 4.3.1 is  $O(n \cdot |AP| + n^2 \cdot (m \cdot n^3 + n^2 + n^2)) = O(n \cdot |AP| + m \cdot n^5 + n^4) = O(|S| \cdot |AP| + |\rightarrow| \cdot |S|^5)$ . ■

Note that Theorem 4.3.2 provides an impression on the complexity of Algorithm 4.3.1 but is, however, just an approximation—similar to Algorithm 4.2.1, the set  $\text{Sat}([s]_{\Pi}, [s']_{\Pi})$  is not necessarily needed to be recomputed in every iteration.

Notation	Interpretation
$\approx \subseteq S \times S$	stutter simulation equivalence as a binary relation between <i>pairs of states</i>
$\Pi_{AP}$	<i>initial</i> partition of state space $S$ , where each block groups all states labeled with the same atomic propositions $a \in AP$
$\Pi_{\mathcal{R}}$	partition of state space $S$ , where each block groups all <i>currently</i> stutter simulation equivalent states (according to Algorithm 4.3.1)
$\Pi_{Eq}$	partition of state space $S$ , where each block groups all stutter simulation equivalent states
$\preceq \subseteq S \times S$	stutter simulation preorder as a binary relation between <i>pairs of states</i>
$\preceq_{AP} \subseteq \Pi_{AP} \times \Pi_{AP}$	<i>initial approximation</i> of the stutter simulation preorder as the identity relation over all blocks (in $\Pi_{AP}$ ), i.e. $B \preceq_{AP} B$ for all $B \in \Pi_{AP}$
$\preceq_{\mathcal{R}} \subseteq \Pi_{\mathcal{R}} \times \Pi_{\mathcal{R}}$	the <i>current approximation</i> (according to Algorithm 4.3.1) of the stutter simulation preorder as a binary relation between pairs of blocks (in $\Pi_{\mathcal{R}}$ ) of <i>currently</i> stutter simulation equivalent states
$\preceq_{pre} \subseteq \Pi_{Eq} \times \Pi_{Eq}$	stutter simulation preorder as a binary relation between <i>pairs of blocks</i> (in $\Pi_{Eq}$ ) of stutter simulation equivalent states
$StSim_{\mathcal{P}}(D) \subseteq \Pi_{\mathcal{R}}$	<i>current</i> stutter-simulator set of block $D \in \Pi_{\mathcal{R}}$ consisting of all blocks $C \in \Pi_{\mathcal{R}}$ , such that $D \preceq_{\mathcal{R}} C$ (according to Algorithm 4.3.1)
$StSim_{TS}(D) \subseteq \Pi_{Eq}$	stutter-simulator set of block $D \in \Pi_{Eq}$ consisting of all blocks $C \in \Pi_{Eq}$ , such that $D \preceq_{pre} C$
$\mathcal{P} = (\Pi_{\mathcal{R}}, \preceq_{\mathcal{R}})$	<i>current</i> partition-relation pair, where $\Pi_{\mathcal{R}}$ and $\preceq_{\mathcal{R}}$ encode the <i>current approximations</i> (according to Algorithm 4.3.1) of $\approx$ and $\preceq$ , respectively

Figure 4.6: Overview of implementation relations and their representations

#### 4.4 Example Computation of $\Pi_{Eq}$ and $\leq_{Pre}$

In this section Algorithm 4.3.1 is applied to transition system  $TS$  given in Figure 4.7, where different colors denote different labelings. Furthermore, the colors indicate the initial approximation  $\Pi_{AP}$  of  $\Pi_{Eq}$ , i.e.  $[s]_{\Pi_{AP}} = [t]_{\Pi_{AP}}$  iff  $s$  and  $t$  have the same color. The divergence-blind stutter simulation preorder  $\leq_{TS}$  and equivalence  $\approx_{TS}$  will be computed and we will finally present the stutter simulation quotient transition system  $TS/\approx_{TS}$ .

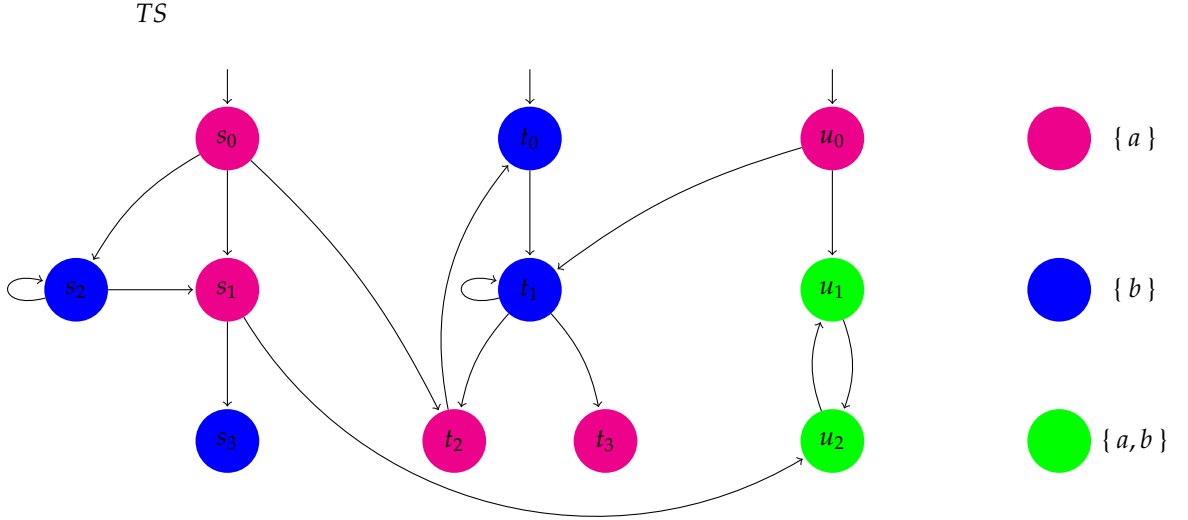


Figure 4.7: Transition system  $TS$

The initialization leads to the following approximations.

$$\Pi = \Pi_{AP} = \{ \underbrace{\{s_0, s_1, t_2, t_3, u_0\}}_{=:B}, \underbrace{\{s_2, s_3, t_0, t_1\}}_{=:C}, \underbrace{\{u_1, u_2\}}_{=:D} \},$$

$$StSim(B) = B, StSim(C) = C, StSim(D) = D \text{ and } StSim = StSim_{AP} = \bigcup_{A \in \Pi} \{ StSim(A) \}.$$

##### Iteration 1:

In the first step, **choose** the pair  $(u_0, u_1)$ , where  $u_1$  is a direct successor of  $u_0$ . State  $u_0$  is contained in block  $B$ , where  $StSim(B) = B$ , and  $u_1$  belongs to  $D$ , where  $StSim(D) = D$ . The guard of the *while*-loop (l. 4) requires to compute the set of constrained successors that can reach a state in  $D$  by exclusively visiting states in  $B$ . Recall that  $B$  consists of states  $s_0, s_1, t_2, t_3$  and  $u_0$ , whereas  $Succ^*(StSim(B), StSim(D)) = Succ^*(\{s_0, s_1, t_2, t_3, u_0\}, \{u_1, u_2\}) = \{s_0, s_1, u_0\} = Sat(B, D)$ . It follows that the stutter simulator set of  $B$  is not a subset of the set of constrained successors and hence, the guard is violated and the pair  $(B, D)$  is a refiner for  $StSim$ . Furthermore, it is a splitter for  $\Pi$ , since  $B$  contains states included and not included in  $Sat(B, D)$ . Entering the body of the *while*-loop, we store the current approximation of  $\Pi_{Eq}$ , i.e.  $\Pi_{old} := \Pi = \{ \{s_0, s_1, t_2, t_3, u_0\}, \{s_2, s_3, t_0, t_1\}, \{u_1, u_2\} \}$  and apply the splitting operation

(ll. 7-10). If  $A$  is a block in  $\Pi$  and  $\Pi_{old}$  then we will denote  $A$  by  $A'$  to indicate that block  $A$  of the *new* partition  $\Pi$  is considered.

### Splitting Operation:

- Since the guard of the first *forall*-loop is satisfied for block  $B \in \Pi_{old}$ , we assign to  $B' \in \Pi$  the set  $B \cap Sat(B, D) = \{s_0, s_1, u_0\}$  and insert  $B_2 := B \setminus Sat(B, D) = \{t_2, t_3\}$  in  $\Pi$ .
- For all other blocks in  $\Pi_{old}$  the guard is violated and no further splittings are carried out.
- $\Pi = \{ \underbrace{\{s_0, s_1, u_0\}}_{=:B}, \underbrace{\{s_2, s_3, t_0, t_1\}}_{=:C}, \underbrace{\{u_1, u_2\}}_{=:D} \} \cup \{ \underbrace{\{t_2, t_3\}}_{=:B_2} \}$

After that, the refinement operation takes place, where all blocks in the *new* partition  $\Pi$  are successively refined.

### Refinement Operation:

- $B' \subsetneq B \in \Pi_{old}$  and  $B' \subseteq Sat(B, D)$ :  
 $StSim(B') = Refine(StSim(B'), Sat(B, D)) = StSim(B) \cap Sat(B, D) = \{s_0, s_1, u_0\}$
- $B'_2 \subsetneq B \in \Pi_{old}$  and  $B'_2 \cap Sat(B, D) = \emptyset$ :  
 $StSim(B'_2) = StSim(B) = \{s_0, s_1, t_2, t_3, u_0\}$
- For all other blocks  $A' \in \Pi$  we have that  $A' = A \in \Pi_{old}$  and  $A' \cap Sat(B, D) = \emptyset$  and hence, no further refinement is possible.
- $StSim = \{ \underbrace{\{s_0, s_1, u_0\}}_{=:StSim(B)}, \underbrace{\{s_0, s_1, t_2, t_3, u_0\}}_{=:StSim(B_2)}, \underbrace{\{s_2, s_3, t_0, t_1\}}_{=:StSim(C)}, \underbrace{\{u_1, u_2\}}_{=:StSim(D)} \}$

The current approximation  $\Pi$  of  $\Pi_{Eq}$  is indicated in Figure 4.8 by the coloring of the nodes.

### Iteration 2:

**Choose**  $(t_2, t_0)$  and compute the set of constrained successors for  $([t_2]_{\Pi}, [t_0]_{\Pi}) = (B_2, C)$ :

$$\begin{aligned} & Succ^*(StSim(B_2), StSim(C)) \\ &= Succ^*(\{s_0, s_1, t_2, t_3, u_0\}, \{s_2, s_3, t_0, t_1\}) \\ &= \{s_0, t_2, u_0\} \end{aligned}$$

Assign  $\Pi_{old} := \Pi$ .

**Splitting Operation:** For blocks  $B, B_2 \in \Pi_{old}$  the guard of the first *forall*-loop is satisfied and we update  $\Pi$  accordingly:

- $B' := B \cap Sat(B_2, C) = \{s_0, u_0\}$
- generate and add  $B_3 := B \setminus Sat(B_2, C) = \{s_1\}$  to  $\Pi$ .

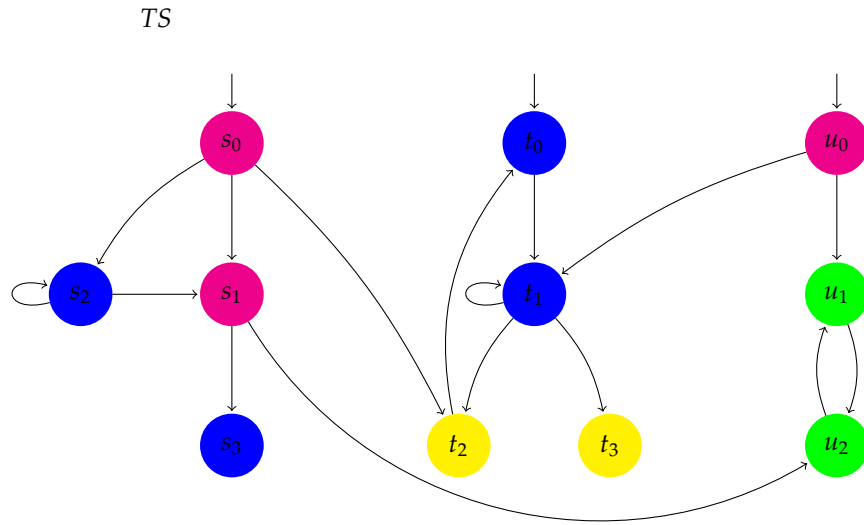


Figure 4.8: TS after 1st update

- $B'_2 := B_2 \cap \text{Sat}(B_2, C) = \{ t_2 \}$
- generate and add  $B_4 := B_2 \setminus \text{Sat}(B_2, C) = \{ t_3 \}$  to  $\Pi$ .
- For all other blocks in  $\Pi_{old}$  the guard is violated and no further splittings are carried out.
- $\Pi = \underbrace{\{ \{ s_0, u_0 \} \}}_{=:B} \cup \underbrace{\{ \{ t_2 \} \}}_{=:B_2} \cup \underbrace{\{ \{ s_2, s_3, t_0, t_1 \} \}}_{=:C} \cup \underbrace{\{ \{ u_1, u_2 \} \}}_{=:D} \cup \underbrace{\{ \{ s_1 \} \}}_{=:B_3} \cup \underbrace{\{ \{ t_3 \} \}}_{=:B_4}$

**Refinement Operation:**

- $B' \subsetneq B \in \Pi_{old}$  and  $B' \subseteq \text{Sat}(B_2, C)$ :  
 $\text{StSim}(B') = \text{Refine}(\text{StSim}(B'), \text{Sat}(B_2, C)) = \text{StSim}(B) \cap \text{Sat}(B_2, C) = \{ s_0, u_0 \}$
- $B'_3 \subsetneq B \in \Pi_{old}$  and  $B'_3 \cap \text{Sat}(B_2, C) = \emptyset$ :  
 $\text{StSim}(B'_3) = \text{StSim}(B) = \{ s_0, s_1, u_0 \}$
- $B'_2 \subsetneq B_2 \in \Pi_{old}$  and  $B'_2 \subseteq \text{Sat}(B_2, C)$ :  
 $\text{StSim}(B'_2) = \text{Refine}(\text{StSim}(B'_2), \text{Sat}(B_2, C)) = \text{StSim}(B_2) \cap \text{Sat}(B_2, C) = \{ s_0, t_2, u_0 \}$
- $B'_4 \subsetneq B_2 \in \Pi_{old}$  and  $B'_4 \cap \text{Sat}(B_2, C) = \emptyset$ :  
 $\text{StSim}(B'_4) = \text{StSim}(B_2) = \{ s_0, s_1, t_2, t_3, u_0 \}$
- For all other blocks  $A' \in \Pi$  we have that  $A' = A \in \Pi_{old}$  and  $A' \cap \text{Sat}(B_2, C) = \emptyset$  and hence, no further refinement is possible.

- $StSim = \{ \underbrace{\{s_0, u_0\}}_{=:StSim(B)}, \underbrace{\{s_0, s_1, u_0\}}_{=:StSim(B_3)}, \underbrace{\{s_0, t_2, u_0\}}_{=:StSim(B_2)}, \underbrace{\{s_0, s_1, t_2, t_3, u_0\}}_{=:StSim(B_4)}, \underbrace{\{s_2, s_3, t_0, t_1\}}_{=:StSim(C)}, \underbrace{\{u_1, u_2\}}_{=:StSim(D)} \}$

The current approximation  $\Pi$  of  $\Pi_{Eq}$  is indicated in Figure 4.9 by the coloring of the nodes.

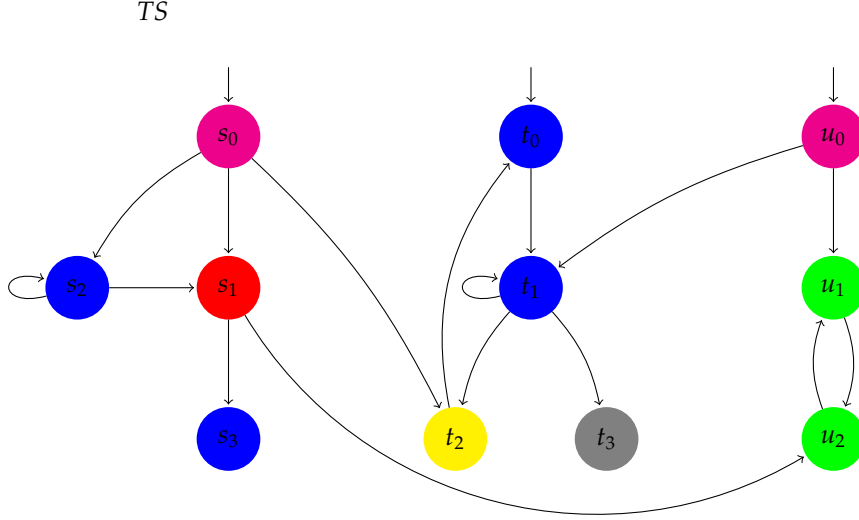


Figure 4.9: TS after 2nd update

### Iteration 3:

Choose  $(u_0, u_1)$  and compute the set of constrained successors for  $([u_0]_{\Pi}, [u_1]_{\Pi}) = (B, D)$ :

$$\begin{aligned}
 & Succ^*(StSim(B), StSim(D)) \\
 &= Succ^*(\{s_0, u_0\}, \{u_1, u_2\}) \\
 &= \{u_0\}
 \end{aligned}$$

Assign  $\Pi_{old} := \Pi$ .

**Splitting Operation:** For block  $B \in \Pi_{old}$  the guard of the first *forall*-loop is satisfied and we update  $\Pi$  accordingly:

- $B' := B \cap Sat(B, D) = \{u_0\}$
- generate and add  $B_5 := B \setminus Sat(B, D) = \{s_0\}$  to  $\Pi$ .
- For all other blocks in  $\Pi_{old}$  the guard is violated and no further splittings are carried out.
- $\Pi = \{ \underbrace{\{u_0\}}_{=:B}, \underbrace{\{s_1\}}_{=:B_3}, \underbrace{\{t_2\}}_{=:B_2}, \underbrace{\{t_3\}}_{=:B_4}, \underbrace{\{s_2, s_3, t_0, t_1\}}_{=:C}, \underbrace{\{u_1, u_2\}}_{=:D} \} \cup \{ \underbrace{\{s_0\}}_{=:B_5} \}$

**Refinement Operation:**

- $B' \subsetneq B \in \Pi_{old}$  and  $B' \subseteq Sat(B, D)$ :  
 $StSim(B') = Refine(StSim(B'), Sat(B, D)) = StSim(B) \cap Sat(B, D) = \{ u_0 \}$
- $B'_5 \subsetneq B \in \Pi_{old}$  and  $B'_5 \cap Sat(B, D) = \emptyset$ :  
 $StSim(B'_5) = StSim(B) = \{ s_0, u_0 \}$
- For all other blocks  $A' \in \Pi$  we have that  $A' = A \in \Pi_{old}$  and  $A' \cap Sat(B, D) = \emptyset$  and hence, no further refinement is possible.
- $StSim =$   
 $\{ \underbrace{\{ u_0 \}}_{=:StSim(B)}, \underbrace{\{ s_0, u_0 \}}_{=:StSim(B_5)}, \underbrace{\{ s_0, s_1, u_0 \}}_{=:StSim(B_3)}, \underbrace{\{ s_0, t_2, u_0 \}}_{=:StSim(B_2)}, \underbrace{\{ s_0, s_1, t_2, t_3, u_0 \}}_{=:StSim(B_4)}, \underbrace{\{ s_2, s_3, t_0, t_1 \}}_{=:StSim(C)}, \underbrace{\{ u_1, u_2 \}}_{=:StSim(D)} \}$

The current approximation  $\Pi$  of  $\Pi_{Eq}$  is indicated in Figure 4.11 by the coloring of the nodes.

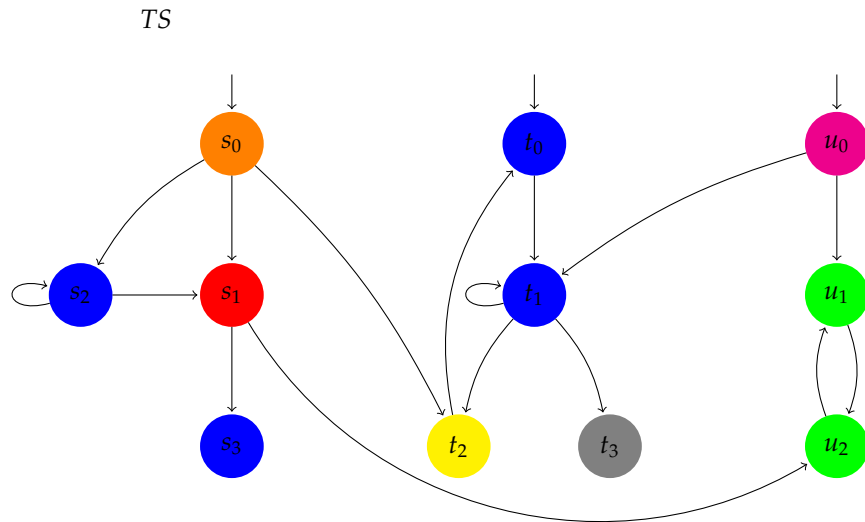


Figure 4.10: TS after 3rd update

**Iteration 4:**

**Choose**  $(s_2, s_1)$  and compute the set of constrained successors for  $([s_2]_{\Pi}, [s_1]_{\Pi}) = (C, B_3)$ :

$$\begin{aligned}
 & Succ^*(StSim(C), StSim(B_3)) \\
 &= Succ^*({ s_2, s_3, t_0, t_1 }, { s_0, s_1, u_0 }) \\
 &= { s_2 }
 \end{aligned}$$

Assign  $\Pi_{old} := \Pi$ .

**Splitting Operation:** For block  $C \in \Pi_{old}$  the guard of the first *forall*-loop is satisfied and we update  $\Pi$  accordingly:

- $C' := C \cap Sat(C, B_3) = \{s_2\}$
- generate and add  $C_2 := C \setminus Sat(C, B_3) = \{s_3, t_0, t_1\}$  to  $\Pi$ .
- For all other blocks in  $\Pi_{old}$  the guard is violated and no further splittings are carried out.
- $\Pi = \{ \underbrace{\{u_0\}}_{=:B}, \underbrace{\{s_0\}}_{=:B_5}, \underbrace{\{s_1\}}_{=:B_3}, \underbrace{\{t_2\}}_{=:B_2}, \underbrace{\{t_3\}}_{=:B_4}, \underbrace{\{s_2\}}_{=:C}, \underbrace{\{u_1, u_2\}}_{=:D} \} \cup \{ \underbrace{\{s_3, t_0, t_1\}}_{=:C_2} \}$

**Refinement Operation:**

- $C' \subsetneq C \in \Pi_{old}$  and  $C' \subseteq Sat(C, B_3)$ :  
 $StSim(C') = Refine(StSim(C'), Sat(C, B_3)) = StSim(C) \cap Sat(C, B_3) = \{s_2\}$
- $C'_2 \subsetneq C \in \Pi_{old}$  and  $C'_2 \cap Sat(C, B_3) = \emptyset$ :  
 $StSim(C'_2) = StSim(C) = \{s_2, s_3, t_0, t_1\}$
- For all other blocks  $A' \in \Pi$  we have that  $A' = A \in \Pi_{old}$  and  $A' \cap Sat(C, B_3) = \emptyset$  and hence, no further refinement is possible.
- $StSim = \{ \underbrace{\{u_0\}}_{=:StSim(B)}, \underbrace{\{s_0, u_0\}}_{=:StSim(B_5)}, \underbrace{\{s_0, s_1, u_0\}}_{=:StSim(B_3)}, \underbrace{\{s_0, t_2, u_0\}}_{=:StSim(B_2)}, \underbrace{\{s_0, s_1, t_2, t_3, u_0\}}_{=:StSim(B_4)}, \underbrace{\{s_2\}}_{=:StSim(C)}, \underbrace{\{s_2, s_3, t_0, t_1\}}_{=:StSim(C_2)}, \underbrace{\{u_1, u_2\}}_{=:StSim(D)} \}$

The current approximation  $\Pi$  of  $\Pi_{Eq}$  is indicated in Figure 4.11 by the coloring of the nodes.

**Iteration 5:**

**Choose**  $(t_1, t_2)$  and compute the set of constrained successors for  $([t_1]_{\Pi}, [t_2]_{\Pi}) = (C_2, B_2)$ :

$$\begin{aligned} & Succ^*(StSim(C_2), StSim(B_2)) \\ &= Succ^*({s_2, s_3, t_0, t_1}, {s_0, t_2, u_0}) \\ &= \{t_0, t_1\} \end{aligned}$$

Assign  $\Pi_{old} := \Pi$ .

**Splitting Operation:** For block  $C_2 \in \Pi_{old}$  the guard of the first *forall*-loop is satisfied and we update  $\Pi$  accordingly:

- $C'_2 := C_2 \cap Sat(C_2, B_2) = \{t_0, t_1\}$
- generate and add  $C_3 := C_2 \setminus Sat(C_2, B_2) = \{s_3\}$  to  $\Pi$ .

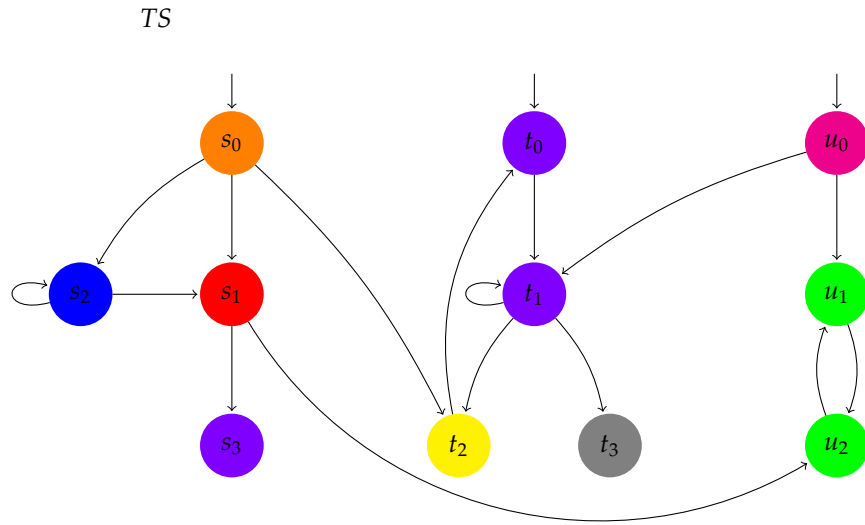


Figure 4.11: TS after 4th update

- For all other blocks in  $\Pi_{old}$  the guard is violated and no further splittings are carried out.
- $\Pi = \{ \underbrace{\{u_0\}}_{=:B}, \underbrace{\{s_0\}}_{=:B_5}, \underbrace{\{s_1\}}_{=:B_3}, \underbrace{\{t_2\}}_{=:B_2}, \underbrace{\{t_3\}}_{=:B_4}, \underbrace{\{s_2\}}_{=:C}, \underbrace{\{t_0, t_1\}}_{=:C_2}, \underbrace{\{u_1, u_2\}}_{=:D} \} \cup \{ \underbrace{\{s_3\}}_{=:C_3} \}$

**Refinement Operation:**

- $C'_2 \subsetneq C_2 \in \Pi_{old}$  and  $C'_2 \subseteq Sat(C, B_3)$ :  
 $StSim(C'_2) = Refine(StSim(C'_2), Sat(C, B_2)) = StSim(C_2) \cap Sat(C, B_2) = \{t_0, t_1\}$
- $C'_3 \subsetneq C_2 \in \Pi_{old}$  and  $C'_3 \cap Sat(C, B_2) = \emptyset$ :  
 $StSim(C'_3) = StSim(C_2) = \{s_2, s_3, t_0, t_1\}$
- For all other blocks  $A' \in \Pi$  we have that  $A' = A \in \Pi_{old}$  and  $A' \cap Sat(C, B_2) = \emptyset$  and hence, no further refinement is possible.
- $StSim = \{ \underbrace{\{u_0\}}_{=:StSim(B)}, \underbrace{\{s_0, u_0\}}_{=:StSim(B_5)}, \underbrace{\{s_0, s_1, u_0\}}_{=:StSim(B_3)}, \underbrace{\{s_0, t_2, u_0\}}_{=:StSim(B_2)}, \underbrace{\{s_0, s_1, t_2, t_3, u_0\}}_{=:StSim(B_4)}, \underbrace{\{s_2\}}_{=:StSim(C)}, \underbrace{\{t_0, t_1\}}_{=:StSim(C_2)}, \underbrace{\{s_2, s_3, t_0, t_1\}}_{=:StSim(C_3)}, \underbrace{\{u_1, u_2\}}_{=:StSim(D)} \}$

The current approximation  $\Pi$  of  $\Pi_{Eq}$  is indicated in Figure 4.12 by the coloring of the nodes.

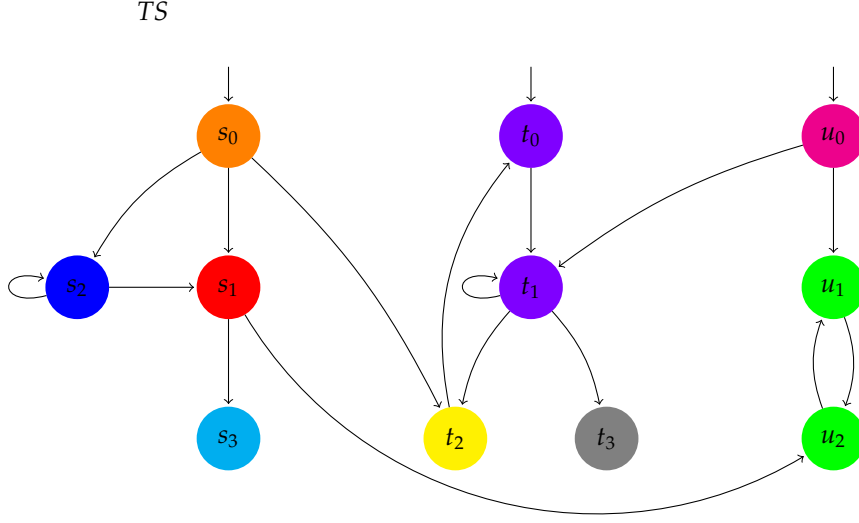


Figure 4.12: TS after 5th update

**Iteration 6:**

Choose  $(s_0, s_2)$  and compute the set of constrained successors for  $([s_0]_{\Pi}, [s_2]_{\Pi}) = (B_5, C)$ :

$$\begin{aligned}
 & Succ^*(StSim(B_5), StSim(C)) \\
 &= Succ^*({s_0, u_0}, {s_2}) \\
 &= {s_0}
 \end{aligned}$$

Assign  $\Pi_{old} := \Pi$ .

**Splitting Operation:** For all blocks in  $\Pi_{old}$  the guard is violated and no splittings are possible, i.e.  $\Pi_{old} = \Pi = \{ \underbrace{{u_0}}_{=:B}, \underbrace{{s_0}}_{=:B_5}, \underbrace{{s_1}}_{=:B_3}, \underbrace{{t_2}}_{=:B_2}, \underbrace{{t_3}}_{=:B_4}, \underbrace{{s_2}}_{=:C}, \underbrace{{t_0, t_1}}_{=:C_2}, \underbrace{{s_3}}_{=:C_3}, \underbrace{{u_1, u_2}}_{=:D} \}$ .

**Refinement Operation:**

- $B'_5 \subsetneq B_5 \in \Pi_{old}$  and  $B'_5 \subseteq Sat(B_5, C)$ :  
 $StSim(B'_5) = Refine(StSim(B'_5), Sat(B_5, C)) = StSim(B_5) \cap Sat(B_5, C) = {s_0}$
- For all other blocks  $A' \in \Pi$  we have that  $A' = A \in \Pi_{old}$  and  $A' \cap Sat(B_5, C) = \emptyset$  and hence, no further refinements are carried out.
- $StSim = \{ \underbrace{{u_0}}_{=:StSim(B)}, \underbrace{{s_0}}_{=:StSim(B_5)}, \underbrace{{s_0, s_1, u_0}}_{=:StSim(B_3)}, \underbrace{{s_0, t_2, u_0}}_{=:StSim(B_2)}, \underbrace{{s_0, s_1, t_2, t_3, u_0}}_{=:StSim(B_4)}, \underbrace{{s_2}}_{=:StSim(C)}, \underbrace{{t_0, t_1}}_{=:StSim(C_2)}, \underbrace{{s_2, s_3, t_0, t_1}}_{=:StSim(C_3)}, \underbrace{{u_1, u_2}}_{=:StSim(D)} \}$

After this iteration there does not exist any pair of states that violates the guard of the *while*-loop and hence, we have computed the coarsest stutter simulation preorder

$$\begin{aligned}
 \leq &= \{ (s, t) \mid t \in StSim(s) \} \\
 &= \{ (u_0, u_0), \\
 &\quad (s_0, s_0), \\
 &\quad (s_1, s_1), (s_1, s_0), (s_1, u_0), \\
 &\quad (t_2, t_2), (t_2, s_0), (t_2, u_0), \\
 &\quad (t_3, t_3), (t_3, s_0), (t_3, s_1), (t_3, t_2), (t_3, u_0), \\
 &\quad (s_2, s_2), \\
 &\quad (t_0, t_0), (t_1, t_1), (t_0, t_1), (t_1, t_0), \\
 &\quad (s_3, s_3), (s_3, s_2), (s_3, t_0), (s_3, t_1), \\
 &\quad (u_1, u_1), (u_2, u_2), (u_1, u_2), (u_2, u_1) \}
 \end{aligned}$$

and equivalence

$$\begin{aligned}
 \cong &= \{ (s, t) \mid [s]_{\Pi} = [t]_{\Pi} \} \\
 &= \{ (s, t) \mid [s]_{\Pi_{Eq}} = [t]_{\Pi_{Eq}} \} \\
 &= \{ (u_0, u_0), (s_0, s_0), (s_1, s_1), (t_2, t_2), (t_3, t_3), (s_2, s_2), (t_0, t_0), (t_1, t_1), (t_0, t_1), (t_1, t_0), \\
 &\quad (s_3, s_3), (u_1, u_1), (u_2, u_2), (u_1, u_2), (u_2, u_1) \},
 \end{aligned}$$

which enables to define the quotient system  $TS/\cong = (S/\cong, Act, \rightarrow_{\cong}, I_{\cong}, AP, L_{\cong})$  given in Figure 4.13.

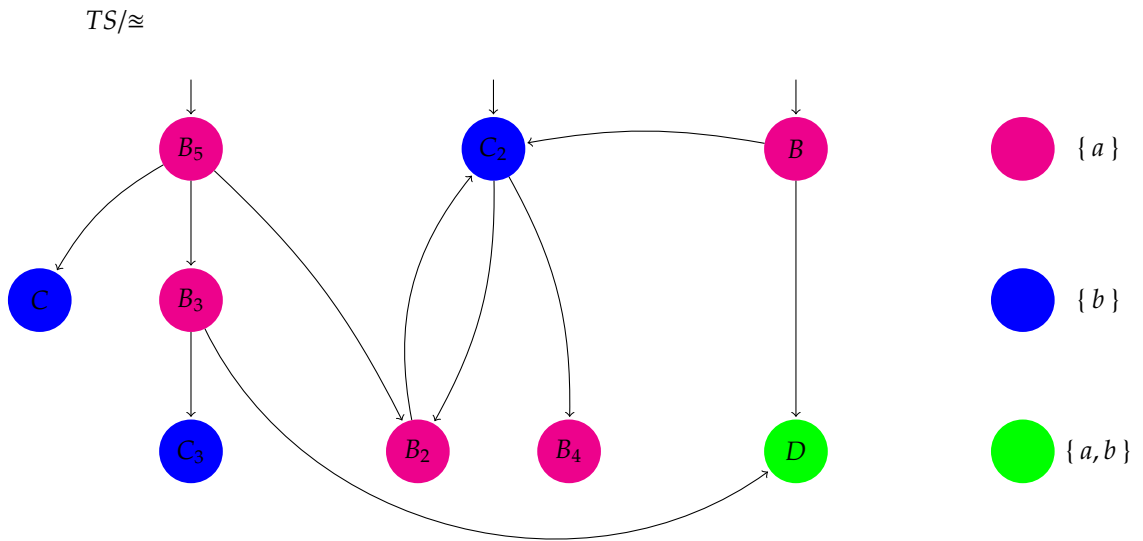


Figure 4.13: Quotient transition system  $TS/\cong$

## 5 Conclusion

In this thesis we introduced and formally defined the implementation relations *divergence-blind stutter simulation preorder* and *equivalence*. It turned out that defining the divergence-insensitive variant of stutter simulation equivalence on transition systems to obtaining an *LTL*-equivalent minimized quotient is too weak, since it does not guarantee stutter trace equivalence. The solution to that problem is the additional notion of *divergence-sensitivity*, which requires the distinction between states exhibiting an infinite path, where only *invisible* steps are performed, and those that do not. These considerations are not mentioned in Ranzato and Tapparo [6] but the authors claim that divergence-blind stutter simulation equivalence preserves the existential fragment of *CTL*, excluding the  $\bigcirc$  and  $\square$  operators, i.e.  $\exists CTL_{\setminus \bigcirc, \square}$ . However, for future research it is important to clearly define the language fragments that are preserved by divergence-sensitive and/or -blind stutter simulation relation, since this is one of the essential informations needed when minimizing and comparing system models.

In the last section of Chapter 3, we compared these achievements to the divergence-blind and -sensitive bisimulation equivalences, which impose more strict conditions on two states to being equivalent than the stutter simulation relations. We have shown that stutter bisimulation implies stutter simulation, whereas the reverse (clearly) does not hold in general. Based on these results and since stutter bisimulation preserves  $\forall CTL_{\setminus \bigcirc}^*$ , it can be concluded that divergence-sensitive stutter simulation equivalence does not preserve this fragment, since otherwise stutter simulation equivalent states satisfied the same  $\forall CTL_{\setminus \bigcirc}^*$ -formulae, which implies stutter bisimulation equivalence.

The last chapter provides two algorithms. The first is needed to exclusively compute the divergence-blind stutter simulation preorder, whereas the second additionally returns the corresponding equivalence relation. The initial approximations in both algorithms are over-approximations indicated by the state-labelings and are successively refined. Therefore, we introduced two tools, the *refinement* and the *splitting operation*. The first one basically refines the approximation of the stutter simulation preorder, whereas the latter one updates the approximation of the equivalence. We have shown that the given algorithms are correct and polynomial in the size of the number of states of the input transition system. However, the given time complexities are approximations and require a more detailed analysis, especially in terms of the data structures used and redundant computations. In this context it should be mentioned that Ranzato and Tapparo [6] proposed an efficiency improvement that is based on the idea of exit states and collapsing stutter cycles, which was not taken into any further consideration in this thesis.

In summary, this thesis structures and expands several aspects of the underlying paper [6], gives a detailed introduction into stutter simulation relations and paved the way for further research. Nevertheless, since these approaches only consider the divergence-blind variant, there is enough room for research concerning the divergence-sensitive stutter simulation.



## Bibliography

- [1] C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
- [2] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.
- [3] R. De Nicola and F. Vaandrager. Three logics for branching bisimulation. *J. ACM*, 42(2):458–487, 1995.
- [4] E. A. Emerson and J. Srinivasan. Branching time temporal logic. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency — Proceedings of REX School/Workshop 1988*, volume 354 of *Lecture Notes in Computer Science*, pages 123–172. Springer-Verlag, May-June 1988.
- [5] J. Groote and F. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In *Proceedings 17th Int'l Coll. Automata, Languages and Programming*, pages 626–638, Warwick, UK, 1990. Springer-Verlag.
- [6] F. Ranzato and F. Tapparo. Computing stuttering simulations. In *Proceedings of the 20th International Conference on Concurrency Theory (CONCUR'09)*, pages 542–556, Bologna, Italy, 2009. Springer-Verlag.