Unified bisimulation applied to incremental abstraction of Petri nets

Discrete Event Dynamic Systems (2026) 36:20
https://doi.org/10.1007/s10626-026-00434-z

RESEARCH

Unified bisimulation applied to incremental abstraction of Petri nets

Bengt Lennartson[1]

Received: 5 December 2024 / Accepted: 19 February 2026

Abstract
Bisimulation is a powerful abstraction method, which can be used to perform model reduction, especially for modular transition systems. A unified formulation of strong, weak, stutter, and branching bisimulation is presented. For branching bisimulation an extended relation is shown to coincide with the original branching bisimulation when the largest relations (equivalence relations) are considered. A block transition based description that is more natural from a model reduction perspective is also shown to be equivalent to the original relation based bisimulations. All bisimulation formulations are based on general transition system models, which means that systems both including state and transition labels are handled in a unified way. An incremental abstraction based on divergence sensitive branching bisimulation is then formulated and applied to Petri nets. The strength of the proposed method is demonstrated especially for Petri nets, combining both analytical and computational abstraction.

Keywords Bisimulation · Model reduction · Transition systems · Temporal logic · Petri nets

1 Introduction

The well known state space explosion in verification and synthesis of discrete event systems can be handled in different ways. One popular approach is to represent models by binary decision diagrams (BDDs) (Bryant 1992). SAT solvers have also shown to be very effective (Eén and Sörensson 2004). An attractive alternative is to use abstractions such that reduced models, which preserve critical properties, can be used. One of the most well known abstractions is bisimulation (Park 1981; Milner 1989), which is a general technique to determine if individual states of a transition system have the same future behavior. States with such common behavior are related and are said to be bisimilar. Bisimulation can therefore be used to reduce the number of states for transition systems.

✉ Bengt Lennartson
bengt.lennartson@chalmers.se

[1] Division of Systems and Control, Department of Electrical Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden

Published online: 25 May 2026

Two bisimulation relations that have a strong coupling to temporal logic are 1) branching bisimulation for labeled transition systems and process algebra with event/transition labels (automata) (Van Glabbeek and Weijland 1996; Nicola and Vaandrager 1995), and 2) stutter bisimulation for Kripke structures including state labels (Browne et al. 1988; Nicola and Vaandrager 1995; Baier and Katoen 2008). In Lennartson and Noori-Hosseini (2018), these two formulations are unified in a bisimulation, where both state and transition labels are included. Two other similar formulations that unify state and transition labels are presented in Gerth et al. (1999) and Trčka (2007). They are, however, based on traditional relation based bisimulation formulations. Our alternative definition is directly formulated as an equivalence relation. All earlier bisimulation definitions are based on relations that are shown to be equivalence relations, sometimes including complex proofs, especially for branching bisimulation (Van Glabbeek et al. 2009).

In this paper the four most common bisimulations, strong, weak, stutter, and branching bisimulation are formulated in a unified manner, by introducing a generic transition operator followed by specific instances. Traditional relation formulations are given, as well as our block transition formulations, which by construction generate non-trivial equivalence relations. The branching bisimulation formulation is evaluated in more detail and an extended branching bisimulation is proposed. This relation is shown to coincide with the original branching bisimulation when the largest relations are considered, and the equivalence between the relation formulation and our block transition formulation is also shown.

Including divergence sensitivity in the branching simulation (Van Glabbeek and Weijland 1996; Nicola and Vaandrager 1995; Lennartson and Noori-Hosseini 2018), it is also shown how temporal logic verification can be performed for modular systems based on incremental abstraction. The temporal logic is an extension of CTL*, including specifications on both state and transition labels, introduced in Lennartson and Noori-Hosseini (2018). This strategy is adapted to verify temporal logic properties of bounded modular Petri nets by including a transition system model for each place. It is demonstrated how both analytical and algorithmic abstraction can be combined, to be able to solve a fairness problem that has not been possible to solve by powerful model checking tools. This application of branching bisimulation state reduction summarizes and extend results presented in Lennartson (2021).

The paper starts in Section 2 with a survey on the most important results for the basic bisimulation, also called strong bisimulation. In Section 3 a deeper analysis of branching and stutter bisimulation is given, and in Section 4 weak bisimulation is presented in the unified framework, including both state and transition labels, followed by a unification in Section 5 of strong, weak, and branching bisimulation, all three also formulated in the alternative block transition formulation. It is also proven in this section that the traditional relation based bisimulation formulation and our block transition formulation are equivalent. In Section 6 divergence sensitive branching bisimulation is further investigated. This follows by a temporal logic verification method based on incremental abstraction, which is formulated and applied to Petri nets in Section 7. Finally, some conclusions are given in Section 8.

2 Bisimulation

Bisimulation is a binary relation that determines which individual states in a transition system have the same future behavior. States with such common behavior are said to be *bisimilar*. The basic bisimulation relation, which is also called *strong bisimulation* (Park 1981; Milner 1989), is defined and illustrated in this section. It is also emphasized that strong bisimulation becomes an equivalence relation when the maximum number of related states is included. The name of the relation is motivated by the fact that it is the strongest and most detailed bisimulation which is naturally formulated.

2.1 Strong bisimulation

Before the definition of strong bisimulation is given, a *transition system* G is defined as a six-tuple $G = \langle X, \Sigma, T, I, AP, \lambda \rangle$ where X is a finite set of states, Σ is a finite set of events, $T \subseteq X \times \Sigma \times X$ is a transition relation, where $t = (x, a, x') \in T$ includes the source state x, the event a, and the target state x' of the transition t, $I \subseteq X$ is a set of possible initial states, AP is a set of atomic propositions, and $\lambda : X \to 2^{AP}$ is a state labeling function. A transition (x, a, x') is also denoted $x \xrightarrow{a} x'$. A transition system without state labels is called an automaton or a labeled transition system (LTS) (Keller 1976), and a transition system without transition labels (events) is called a Kripke structure (Kripke 1963; Baier and Katoen 2008).

Definition 1 (Strong bisimulation) Given a transition system $G = \langle X, \Sigma, T, I, AP, \lambda \rangle$, a binary relation $\mathcal{R} \subseteq X \times X$ is a *strong bisimulation* (SB) if, for any states $x, y \in X$ and event $a \in \Sigma$, the implication

$$x \mathcal{R} y \Rightarrow \lambda(x) = \lambda(y) \wedge p_{\mathcal{R}}^{SBx} \wedge p_{\mathcal{R}}^{SBy}$$

holds with the *transfer properties*

$$p_{\mathcal{R}}^{SBx} := \forall x' : x \xrightarrow{a} x' \Rightarrow \exists y' : y \xrightarrow{a} y' \wedge x' \mathcal{R} y',$$
$$p_{\mathcal{R}}^{SBy} := \forall y' : y \xrightarrow{a} y' \Rightarrow \exists x' : x \xrightarrow{a} x' \wedge x' \mathcal{R} y'.$$

All related states $(x, y) \in \mathcal{R}$ are said to be *strongly bisimilar*, denoted $x \sim_s y$. □

The transfer property $p_{\mathcal{R}}^{SBx}$ holds if, for the two source states x and y, every transition $x \xrightarrow{a} x'$, for all existing target states $x' \in X$ and events $a \in \Sigma$, is matched by at least one transition $y \xrightarrow{a} y'$, and the target states are also related, i.e. $x' \mathcal{R} y'$. The second transfer property $p_{\mathcal{R}}^{SBy}$ requires furthermore that every transition $y \xrightarrow{a} y'$ is matched by at least one transition $x \xrightarrow{a} x'$, but still the target states are related in the same order as in $p_{\mathcal{R}}^{SBx}$. This condition is close to symmetric, but the same target state relation $x' \mathcal{R} y'$ in both transfer properties still implies that a strong bisimulation can be non-symmetric, as is illustrated in Example 1.

Terminal states do not need any extra handling, since the transfer properties in Definition 1 are defined as implications with a universal quantification in the premise. If no transition

starts from state x, the implication in $p_{\mathcal{R}}^{SBx}$ is vacuously true for this state. This means that all terminal states with the same state label are strongly bisimilar.

Both state and transition labels Bisimulation is normally defined either for Kripke structures, only including state labels, or alternatively for labeled transition systems or process algebra (Milner 1989), only including transition labels. In this paper both state and transition labels are accepted in bisimulation relations. Indeed, state labels are simply introduced by adding the equality condition $\lambda(x) = \lambda(y)$. Thus, a prerequisite for two states to be bisimilar is that they have the same state label.

Example 1 Consider the transition system G in Fig. 1. Based on Definition 1, the relation $\mathcal{R} = \{(1,4),(0,0),(1,1),(2,2),(3,3),(4,4)\}$ is a strong bisimulation. The first state pair $(1,4) \in \mathcal{R}$, since $\lambda(1) = \lambda(4) = \{p\}$, $1 \xrightarrow{b} 0$ is matched by $4 \xrightarrow{b} 0$ and $(0,0) \in \mathcal{R}$, and symmetrically $4 \xrightarrow{b} 0$ is matched by $1 \xrightarrow{b} 0$ and $(0,0) \in \mathcal{R}$. Observe, however, that this bisimulation relation is not symmetric because the element (4, 1) is not included in $\mathcal{R}$. The identity relations hold, since every state transition can be matched by itself. In Example 2 it will be shown that G includes a number of additional strongly related states. □

2.2 Equivalence relation and quotient transition system

Based on the SB relation in Definition 1, an equivalence relation can be achieved. It is obtained by taking the union of all possible SBs, which generates the *strong bisimilarity* relation $\sim_s$ in Definition 1, i.e.

$$\sim_s = \bigcup \{\mathcal{R} \mid \mathcal{R} \text{ is a strong bisimulation}\}$$

In Milner (1989) this union is shown to be both the largest SB and an equivalence relation. More detailed proofs on these technical results are given in Gorrieri and Versari (2015); Sangiorgi (2012). The inclusion of state labels in this presentation is a minor extension, where we note that bisimulation for Kripke structures with state labels but no transition labels is handled in Baier and Katoen (2008). The union of all SBs can be computed as the greatest fixed point of a relation function.

Proposition 1 (Largest strong bisimulation) By introducing the relation function $\mathcal{F} : X \times X \to X \times X$, where

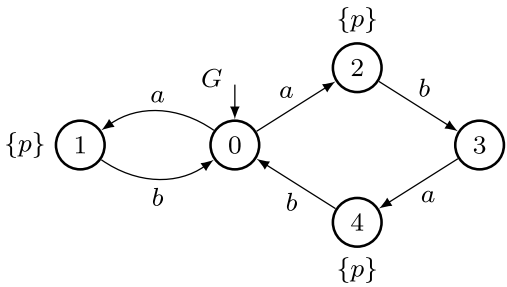


Fig. 1 Transition system G with no state label in state 0 and 3, and state label $\{p\}$ in the remaining states

$$\mathcal{F}(\mathcal{R}) = \{(x,y)|\lambda(x) = \lambda(y) \wedge p_{\mathcal{R}}^{SBx} \wedge p_{\mathcal{R}}^{SBy}\}$$

and $p_{\mathcal{R}}^{SBx}$ and $p_{\mathcal{R}}^{SBy}$ are defined in Definition 1, the union of all SBs is achieved as the greatest fixed point of the relation equation $\mathcal{R} = \mathcal{F}(\mathcal{R})$ (Milner 1989). The solution to this greatest fixed point is the *largest strong bisimulation*, denoted $\mathcal{R}_\omega$. □

A simple way to compute $\mathcal{R}_\omega$ is to iterate the recursive equation

$$\mathcal{R}_{i+1} = \mathcal{F}(\mathcal{R}_i), \quad \mathcal{R}_0 = \{(x,y)|\lambda(x) = \lambda(y)\} \tag{1}$$

until $\mathcal{R}_{i+1} = \mathcal{R}_i \stackrel{\text{def}}{=} \mathcal{R}_\omega$ (Gorrieri and Versari 2015). This algorithm is illustrated in Example 2.

Difference between strong bisimulation and strong bisimilarity The fact that $\sim_s = \mathcal{R}_\omega$ is an equivalence relation implies that it is both reflexive, symmetric, and transitive. Thus, there is a clear difference between SB and strong bisimilarity $\sim_s$. An arbitrary SB relation can be both non-symmetric, non-transitive, and non-reflexive, while the largest SB relation, including the union of all possible SBs, always generates a reflexive, symmetric, and transitive relation.

Quotient transition system $G/\sim$ To obtain reduced transition systems, equivalent states for any equivalence relation are merged into equivalence classes $[x] = \{y \in X | x \sim y\}$, also called blocks. These blocks, which are non-overlapping subsets of X, divide the state space into the quotient set $X/\sim$, also called a partition Π of X. The block/equivalence class including state x is denoted $\Pi(x) = [x]$. A partition Π_1 that is finer than a partition Π_2 means that $\Pi_1(x) \subseteq \Pi_2(x)$ for all $x \in X$. It is denoted $\Pi_1 \preceq \Pi_2$.

Blocks are the states in reduced transition systems, and the notion partition Π is used in the computation of this model, while the resulting reduced model takes the equivalence perspective using the notion quotient. For a transition system G, the reduced model is therefore called *quotient transition system*, denoted $G/\sim$, and the state space of $G/\sim$ is the quotient set $X/\sim = \Pi$.

Relation between G and $G/\sim_s$ By generating an extended model, including both G and the quotient transition system $G/\sim_s$ based on Definition 1, it can also be shown (Baier and Katoen 2008) that every state $x \in X$ in G is strongly bisimilar to the corresponding block state $[x] \in X/\sim_s$ in $G/\sim_s$, that is $[x] \sim_s x$. The equivalence between every state x in G and corresponding block state $[x]$ in $G/\sim_s$ also means that the complete transition systems G and $G/\sim_s$ are said to be *strongly bisimulation equivalent*, denoted $G \sim_s G/\sim_s$.

Example 2 For the transition system G in Fig. 1, the largest SB relation $\mathcal{R}_\omega$, generated by Eq. 1, becomes

$$\mathcal{R}_\omega = \mathcal{R}_1 = \mathcal{R}_0 = \begin{matrix}\{(0,0),(0,3),\\(3,0),(3,3)\}\end{matrix} \cup \begin{matrix}\{(1,1),(1,2),(1,4),\\(2,1),(2,2),(2,4),\\(4,1),(4,2),(4,4)\},\end{matrix}$$

which clearly shows the reflexive, symmetric, and transitive properties of this relation. The partition $\Pi = \{\{0,3\},\{1,2,4\}\}$ generates the reduced quotient transition system $G/\sim_s$ in

Fig. 2. The repeated *ab* string means that $G/\sim_s$ only includes the block state $\{0,3\}$ followed by the event *a*, and the block state $\{1,2,4\}$ followed by the event *b*. Note that the individual states in G are followed by the same event as the related block state in $G/\sim_s$. □

2.3 Alternative bisimulation definition

The equivalence classes in the reduced model $G/\sim_s$ can be generated by the largest SB R_ω, according to Proposition 1. An interesting alternative is to formulate bisimulation based on a state partition, and therefore directly generate an equivalence formulation. In this subsection we introduce this concept, while a more detailed presentation is given in Section 5, based on a unified formulation of strong, weak, stutter, and branching bisimulation.

As stated above, an equivalence relation, due to its symmetric, transitive, and reflexive properties, can be represented more compactly by partitioning the state space X into non-overlapping subsets including equivalent states, here called *block states*.

Strong bisimulation defined as a fixed point on block transitions Given a partition Π of the state space X, bisimilar states generate block states $\Pi(x) = \{y \in X | x \sim_s y\}$. Furthermore, a *block transition relation* between these block states is defined as

$$T_\Pi(x) = \{\Pi(x) \xrightarrow{a} \Pi(x') \,|\, \exists x', a : x \xrightarrow{a} x'\}. \tag{2}$$

For any state $x \in X$, a block state can then be defined as the greatest fixed point on equal state labels and equal block transitions as

$$\Pi(x) = \{y \in X \,|\, \lambda(y) = \lambda(x) \wedge T_\Pi(x) = T_\Pi(y)\}. \tag{3}$$

This formulation is an alternative way to directly define strong bisimilarity as an equivalence, but here based on a more condensed state partition Π instead of an equivalence relation $\mathcal{R}_\omega$. The computation of the greatest fixed point Π in Eq. 3 is obtained by iterating the following recursive equations

$$T_{\Pi_k}(x) = \{\Pi_k(x) \xrightarrow{a} \Pi_k(x') | \exists x', a : x \xrightarrow{a} x'\} \tag{4}$$

$$\Pi_{k+1}(x) = \{y \in X | T_{\Pi_k}(x) = T_{\Pi_k}(y)\} \tag{5}$$

for all $x \in X$ until $\Pi_{k+1}(x) = \Pi_k(x)$, with the initial partition $\Pi_0(x) = \{y | \lambda(x) = \lambda(y)\}$. The state labels are only required in the initial partition Π_0, since the iteration of Eqs. 4 and 5 generates a monotonically decreasing partition Π_k as k increases, which is shown in Proposition 4 in Section 5. This procedure is illustrated in the following example.

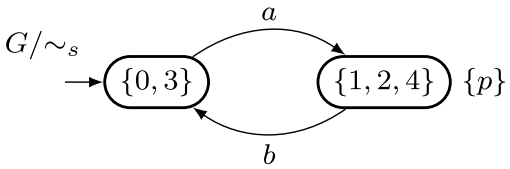


Fig. 2 Quotient transition system $G/\sim_s$ of the transition system G in Fig. 1

Example 3 For the transition system G in Fig. 1 the state label $\{p\}$ is now excluded, which generates the initial partition $\Pi_0 = \{X\} = \{\{0,1,2,3,4\}\}$. According to Eq. 4, this results in the block transition relations

$$T_{\Pi_0}(0) = T_{\Pi_0}(3) = \{X \xrightarrow{a} X\}, \quad T_{\Pi_0}(1) = T_{\Pi_0}(2) = T_{\Pi_0}(4) = \{X \xrightarrow{b} X\}.$$

Updated block states $\Pi_1(x) = \{y \in X | T_{\Pi_0}(x) = T_{\Pi_0}(y)\}$ for $x \in X$ then generate the partition $\Pi_1 = \{\{0,3\},\{1,2,4\}\} \stackrel{\text{def}}{=} \{B_0, B_1\}$. In the next iteration the block transition relations become

$$T_{\Pi_1}(0) = T_{\Pi_1}(3) = \{B_0 \xrightarrow{a} B_1\}, \quad T_{\Pi_1}(1) = T_{\Pi_1}(2) = T_{\Pi_1}(4) = \{B_1 \xrightarrow{b} B_0\},$$

which generates the updated partition $\Pi_2 = \{B_0, B_1\}$ and the fixed point $\Pi_2 = \Pi_1$. Thus, the same quotient transition system as achieved in Fig. 2 also without the state label $\{p\}$, simply because the events *a* and *b* partition the transition system in the same way. □

The equivalence between a unified relation based bisimulation formulation and the more condensed block state based formulation introduced in this subsection is shown in Section 5, including more examples and related existing formulations.

3 Branching bisimulation and stutter bisimulation

Events that are not synchronized with other subsystems are said to be local, and they can be *hidden* by replacing them with the transition label τ. Such τ events are also called *internal* or *silent* events. Some τ transitions $x \xrightarrow{\tau} x'$ can be removed by joining the source and target states into a block state $\{x, x'\}$, but not all of them.

Both in weak bisimulation (WB) and branching bisimulation (BB), some τ transitions are removed. WB is less restrictive than BB, which implies that the block states in a WB equivalence are often somewhat larger, resulting in less block states and a coarser state partition. See further details in Section 4. On the other hand, more details are preserved in BB, where the name expresses that the branching structure is preserved. More specifically this implies that most temporal logic properties are preserved when abstraction is based on BB, while some relevant temporal properties are lost when WB is applied.

Normally, BB does not consider any state labels, but in this paper this is generalized to include both transition and state labels. BB for this type of general transition systems, without any label restrictions, is not common but has been formulated under the names visible bisimulation (Gerth et al. 1999; Lennartson and Noori-Hosseini 2018) and silent bisimulation (Trčka 2007). Since the only difference between these bisimulations and BB is that state labels are also included in a straightforward way, we keep the BB name used for labeled transition systems also for general transition systems, including both state and transition labels. This means that *stutter bisimulation* (Browne et al. 1988; Nicola and Vaandrager 1995) is a special case, where no transition labels are involved. This can be interpreted as all transitions being labelled by τ, and the transition system is reduced to a Kripke structure.

3.1 Branching bisimulation

Branching bisimulation, as introduced in Van Glabbeek and Weijland (1996), is now defined for transition systems also including state labels. In this definition a path, only including hidden τ transitions $x = x_0 \xrightarrow{\tau} x_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} x_n = x'$, $n \geq 0$, is denoted $x \xrightarrow{\tau^*} x'$.

Definition 2 (Branching bisimulation) Given a transition system $G = \langle X, \Sigma, T, I, AP, \lambda \rangle$, a binary relation $\mathcal{R} \subseteq X \times X$ is a *branching bisimulation* (BB) if, for any states $x, y \in X$ and event $a \in \Sigma$, the implication

$$x \mathcal{R} y \;\Rightarrow\; \lambda(x) = \lambda(y) \wedge p_{\mathcal{R}}^{BBx} \wedge p_{\mathcal{R}}^{BBy}$$

holds with the *transfer properties*

$$\begin{aligned} p_{\mathcal{R}}^{BBx} := \forall x' : x \xrightarrow{a} x' \Rightarrow\; & (a = \tau \wedge x' \mathcal{R} y) \vee \\ & (\exists y'', y' : y \xrightarrow{\tau^*} y'' \xrightarrow{a} y' \wedge x \mathcal{R} y'' \wedge x' \mathcal{R} y'), \\ p_{\mathcal{R}}^{BBy} := \forall y' : y \xrightarrow{a} y' \Rightarrow\; & (a = \tau \wedge x \mathcal{R} y') \vee \\ & (\exists x'', x' : x \xrightarrow{\tau^*} x'' \xrightarrow{a} x' \wedge x'' \mathcal{R} y \wedge x' \mathcal{R} y'). \end{aligned}$$

States x and y are said to be *branching bisimilar*, denoted $x \sim_b y$, if there exists a branching bisimulation $\mathcal{R}$ such that $(x, y) \in \mathcal{R}$. A transition in $p_{\mathcal{R}}^{BBx}$ or $p_{\mathcal{R}}^{BBy}$, which satisfies the first statement in the disjunctive consequence, is called a *stutter transition*, while a transition, which does not satisfy this statement, is called a *visible transition*. A transition which satisfies the second alternative statement in the consequence is called a *matching transition*. □

The transfer property $p_{\mathcal{R}}^{BBx}$ is illustrated in Fig. 3. In this definition, a stutter transition $x \xrightarrow{a} x'$ is a τ transition where both the source state x and the target state x' are related to the same common state y. Thus, a transition $x \xrightarrow{\tau} x'$ is a stutter transition when $\exists y : (x, y), (x', y) \in \mathcal{R}$. For a transition to be called a stutter transition or a stutter step, it is often enough that the source and target states have the same state label (Baier and Katoen 2008). Hence, the definition of a stutter transition is here stronger, where the source and target states must also be BB related to the same state.

The following example illustrates the difference between strong and branching bisimulation. It is also shown why some visible τ transitions can be transformed to stutter transitions by adding more state pairs, while other transitions must remain visible to satisfy the restrictions involved in the BB relation.

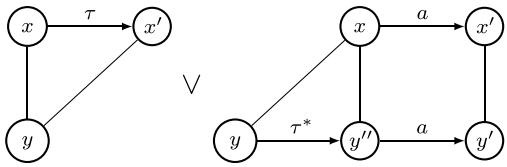


Fig. 3 Transfer diagrams for the two disjunctive relations in the first transfer property $p_{\mathcal{R}}^{BBx}$ in branching bisimulation

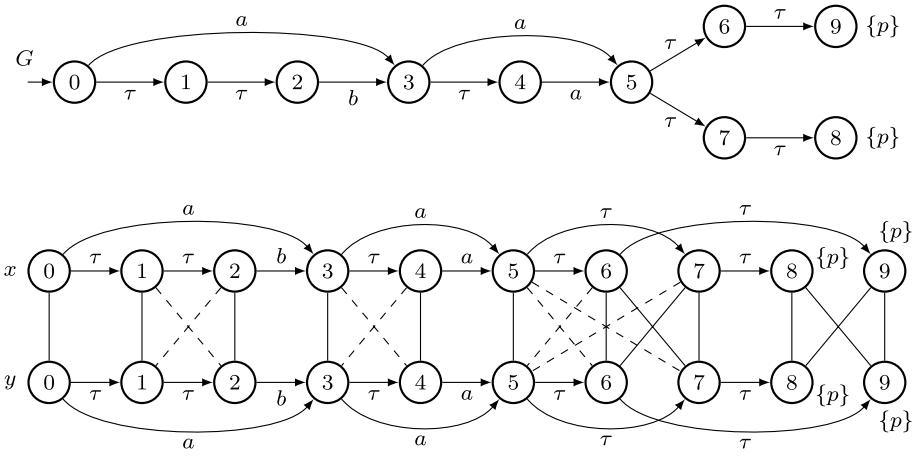


Fig. 4 Transition system G and corresponding state relation diagram where state pairs (x, y) that are both SB and BB related are denoted by solid lines between the states, while only BB related states are denoted by dashed lines

Example 4 Consider the transition system G in Fig. 4. Introducing the identity relation $I_d = (x, x)$ for all states $x \in \{0, \ldots, 9\}$, the relation $\mathcal{R} = I_d \cup \{(6,7), (7,6), (8,9), (9,8)\}$ is according to Definition 1 and Definition 2 both a strong and a branching bisimulation. The states 8 and 9 are related since they have the same state label $\{p\}$, and they are both terminating states. Since these states are related and the transition $6 \xrightarrow{\tau} 9$ is matched by $7 \xrightarrow{\tau} 8$ and vice versa, also $(6,7), (7,6) \in \mathcal{R}$. The fact that SB is a special case of BB follows by applying the matching transitions in Definition 2, see also the right part of Fig. 3, without including any additional τ transitions before the matching a transition ($a = \tau$ is also possible in a matching transition).

Utilizing the larger flexibility BB is offering, additional state pairs can be included in the BB relation. For instance, adding the pairs $(1,2), (2,1) \in \mathcal{R}$ makes the transition $1 \xrightarrow{\tau} 2$ a stutter transition. This is possible, since this transition together with the pairs $(1,1), (1,2), (2,1), (2,2) \in \mathcal{R}$ satisfy the stutter transition property. Indeed, this stutter transition follows by the simple rule that any sequence of τ transitions can be interpreted as stutter transitions, if no alternative paths are involved and the same state label holds after each τ transition.

Since a visible a transition follows both after state 3 and 4, the transition $3 \xrightarrow{\tau} 4$ can also be transformed to a stutter transition by adding the state pairs (3, 4), (4, 3) to $\mathcal{R}$. The transitions $5 \xrightarrow{\tau} 6$ and $5 \xrightarrow{\tau} 7$ also become stutter transitions by adding the state pairs (5, 6), (6, 5), (5, 7), $(7,5) \in \mathcal{R}$. This is possible since the states 6 and 7 are already SB related. Removing the state label $\{p\}$ in the states 8 and 9, all combinations of state pairs involving the states a 5, 6, 7, 8 and 9 are also BB related.

An interesting observation is that the transition $0 \xrightarrow{\tau} 1$ cannot be extended to a stutter transition. The reason is that the state pairs (0, 1) and (1, 0) do not satisfy the BB transfer property. Evaluating $p_{\mathcal{R}}^{BBx}(0,1)$, we find that the transition $0 \xrightarrow{a} 3$ cannot be matched by a

$1 \xrightarrow{\tau^* a}$ path; only a $1 \xrightarrow{\tau^* b}$ path is possible. In the same way, evaluating $p_{\mathcal{R}}^{BBy}(1,0)$ results in the same conflict, and therefore $(0,1) \notin \mathcal{R}$. A similar analysis also shows that $(1,0) \notin \mathcal{R}$.

Clearly, more stutter transitions are obtained by extending the BB relation $\mathcal{R}$. In cases where this is not possible, the visible τ transitions will remain. The transitions $6 \xrightarrow{\tau} 9$ and $7 \xrightarrow{\tau} 8$ are visible since the target states have another state label $\{p\}$ than the corresponding source states. In other words, the states 6 and 9 are not related since $\lambda(6) \neq \lambda(9)$, and with the same argument 7 and 8 are also not related. Different future branches generate visible τ transitions as well, where we remind that $(0,1),(1,0) \notin \mathcal{R}$, see also Example 8 and Fig. 9. □

3.2 Stutter-branching bisimulation

To obtain a unified bisimulation formulation, the BB definition will now be slightly reformulated. The stutter transitions, which satisfy the first statement in the disjunctive consequence in Definition 2, are replaced by a transfer property based on sequences of stutter transitions called *stutter paths*. Since the modified BB has similarities with stutter bisimulation (Browne et al. 1988; Nicola and Vaandrager 1995), which is further motivated based on Definition 4, it is called stutter-branching bisimulation.

Definition 3 (Stutter-branching bisimulation) Consider a transition system with state space X, a relation $\mathcal{R} \subseteq X \times X$, and two sequences of stutter transitions $x = x_0 \xrightarrow{\tau} x_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} x_n = x''$ and $y = y_0 \xrightarrow{\tau} y_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} y_m = y''$. When the states $x_i, i \in \{0,\ldots,n\} = \mathbb{N}_{0,n}$ are related to a state y in the second sequence, and the states y_j, $j \in \mathbb{N}_{0,m}$ are related to a state x in the first sequence, it is denoted

$$x \xrightarrow[*\mathcal{R}y]{\tau^*} x'' := x \xrightarrow{\tau^*} x'' \wedge \bigwedge_{i=0}^{n} x_i \mathcal{R} y, \qquad y \xrightarrow[x\mathcal{R}*]{\tau^*} y'' := y \xrightarrow{\tau^*} y'' \wedge \bigwedge_{j=0}^{m} x \mathcal{R} y_j$$

Such sequences of stutter transitions are called *stutter paths*. The relation $\mathcal{R}$ is a *stutter-branching bisimulation* (SBB) when the transfer properties in Definition 2 are replaced with

$$p_{\mathcal{R}}^{BBx} := \forall x'', x' : x \xrightarrow[*\mathcal{R}y]{\tau^*} x'' \xrightarrow{a} x' \Rightarrow \exists y'', y' : y \xrightarrow[x''\mathcal{R}*]{\tau^*} y'' \xrightarrow{a} y' \wedge x' \mathcal{R} y',$$
$$p_{\mathcal{R}}^{BBy} := \forall y'', y' : y \xrightarrow[*\mathcal{R}y]{\tau^*} y'' \xrightarrow{a} y' \Rightarrow \exists x'', x' : x \xrightarrow[*\mathcal{R}y'']{\tau^*} x'' \xrightarrow{a} x' \wedge x' \mathcal{R} y'.$$

and every terminal state includes a self-loop labeled by an event not equal τ. □

The transfer property $p_{\mathcal{R}}^{BBx}$ is illustrated in Fig. 5. The individual stutter transition conditions in Definition 2 $\forall x_{i+1} : x_i \xrightarrow{\tau} x_{i+1} \Rightarrow x_{i+1} \mathcal{R} y$, $i \in \mathbb{N}_{0,n-1}$ are here collected in the premise as

$$x \mathcal{R} y \wedge \bigwedge_{i=0}^{n-1} (\forall x_{i+1} : x_i \xrightarrow{\tau} x_{i+1} \wedge x_{i+1} \mathcal{R} y) = \forall x_1, \ldots, x'' : x \xrightarrow{\tau^*} x'' \wedge \bigwedge_{i=0}^{n} x_i \mathcal{R} y,$$

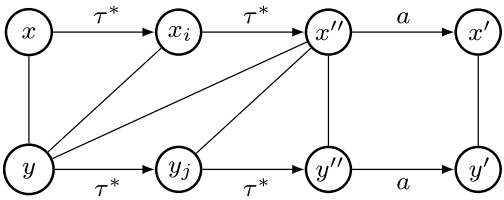


Fig. 5 Transfer diagram for the relations in the first transfer property $p^{BBx}_{\mathcal{R}}$ in stutter-branching bisimulation

followed by the matching transition in Definition 2

$$x''\mathcal{R}y \wedge \forall x' : x'' \xrightarrow{a} x' \Rightarrow \exists y_1, \ldots, y'', y' : y \xrightarrow{\tau^*} y'' \xrightarrow{a} y' \wedge \bigwedge_{j=0}^{m} x''\mathcal{R}y_j \wedge x'\mathcal{R}y'$$

Here, the condition $x''\mathcal{R}y''$ has been extended to $\bigwedge_{j=0}^{m} x''\mathcal{R}y_j$ where not only y'' but all intermediate states in the sequence $y \xrightarrow{\tau^*} y''$ are related to x''. Together, this results in the transfer property

$$p^{BBx}_{\mathcal{R}} = \forall x_1, \ldots, x'', x' : x \xrightarrow{\tau^*} x'' \xrightarrow{a} x' \wedge \bigwedge_{i=0}^{n} x_i\mathcal{R}y \Rightarrow$$
$$\exists y_1, \ldots, y'', y' : y \xrightarrow{\tau^*} y'' \xrightarrow{a} y' \wedge \bigwedge_{j=0}^{m} x''\mathcal{R}y_j \wedge x'\mathcal{R}y'$$

where shortcut notations results in the formulation in Definition 3.

Matching visible and stutter transitions Before the relation between SBB and BB is clarified in more detail, it is observed that matching transitions in both BB and SBB often are visible, but stutter transitions may also be matching transitions. This is illustrated in the following example.

Example 5 Consider the transition system G in Fig. 6, where the relation $\mathcal{R} = \{(0,3),(0,4),(1,4),(2,5)\}$ is both a BB and an SBB. The state pair $(1,4) \in \mathcal{R}$ since $1 \xrightarrow{\tau} 2$ and $4 \xrightarrow{\tau} 5$ are matching transitions and $(2,5) \in \mathcal{R}$, since 2 and 5 are terminal states. The state pair (0, 4) is included in both BB and SBB since $4 \xrightarrow{\tau} 5$ is matched by $0 \xrightarrow{\tau} 1 \xrightarrow{\tau} 2$ and $(1,4),(2,5) \in \mathcal{R}$. In the other direction SBB is symmetric, while the BB condition

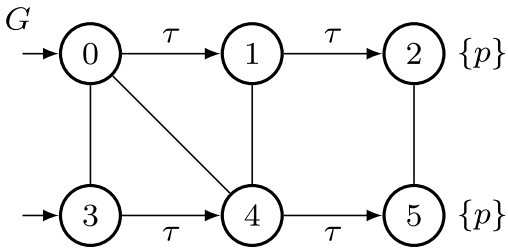


Fig. 6 Transition system G with initial states 0 and 3, where $\mathcal{R} = \{(0,3),(0,4),(1,4),(2,5)\}$ is both a BB and an SBB

includes the stutter transition $0 \xrightarrow{\tau} 1$ where $(1,4) \in \mathcal{R}$. Here we note that the transitions $1 \xrightarrow{\tau} 2$ and $4 \xrightarrow{\tau} 5$ are visible, and they cannot be extended to stutter transitions due to the state label $\{p\}$ in both state 2 and 5. The state pair (0, 3) is included in both BB and SBB due to the matching transitions $0 \xrightarrow{\tau} 1$ and $3 \xrightarrow{\tau} 4$ where $(1,4) \in \mathcal{R}$. At the same time it is observed that these matching transitions are also stutter transitions, due to the state pair $(0,4) \in \mathcal{R}$. □

In the following proposition it is shown that an SBB is a BB, while a BB is not necessarily an SBB, which is also illustrated in an example after the proposition.

Proposition 2 (Relation between branching and stutter-branching bisimulation) Consider a transition system with state space X and a stutter-branching bisimulation $\mathcal{R} \subseteq X \times X$ according to Definition 3. This relation is also a branching bisimulation, while there exist branching bisimulations that are not stutter-branching bisimulations.

Proof Based on the notations in Definition 3 and the sets $X_n = \{x_1, \ldots, x_n\}$ and $Y_m = \{y_1, \ldots, y_m\}$, an SBB where $x\mathcal{R}y$ implies that

$$\mathcal{R} = \{x, x''\} \times \{y, y''\} \cup X_{n-1} \times \{y, y''\} \cup \{x, x''\} \times Y_{m-1} \cup \{(x', y')\} \tag{6}$$

This relation is also a BB, which follows by recursively applying the transfer properties in Definition 2 for $i \in \mathbb{N}_{0,n-1}$ and $j \in \mathbb{N}_{0,m-1}$ with $x'' = x_n$ and $y'' = y_m$. More specifically, evaluating the first statement in the disjunctive consequence in $p_{\mathcal{R}}^{BBx}$ for $i \in \mathbb{N}_{0,n-1}$ generates the condition $x''\mathcal{R}y$, followed by an evaluation of the second statement in the consequence one time, which gives $x''\mathcal{R}y''$ and $x'\mathcal{R}y'$. In the same way, evaluation of $p_{\mathcal{R}}^{BBy}$ results in $x\mathcal{R}y''$ and again $x''\mathcal{R}y''$ and $x'\mathcal{R}y'$. To summarize, given an SBB relation $\mathcal{R}$, the same relation can be created according to Definition 2, which is then a BB. Thus, an SBB is a BB.

In the evaluation of the second statement in the consequences, the conditions $x''\mathcal{R}y_j$ for $j \in \mathbb{N}_{1,m-1}$ in $p_{\mathcal{R}}^{BBx}$ and $x_i\mathcal{R}y''$ for $i \in \mathbb{N}_{1,n-1}$ in $p_{\mathcal{R}}^{BBy}$, included in SBB, are not required in BB, according to Definition 2. This means that

$$\mathcal{R} \setminus \big((\{x''\} \times Y_{m-1}) \cup (X_{n-1} \times \{y''\})\big) \tag{7}$$

is also a BB but not an SBB. Thus, a stronger condition (more state pairs) is required in SBB than in BB, which is the reason why there exist BBs that are not SBBs, also illustrated in Example 6 and Fig. 7. □

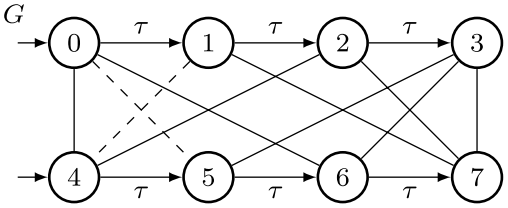


Fig. 7 Transition system G with initial states 0 and 4, where $\mathcal{R} = \{(0,4), (0,6), (1,7), (2,4), (2,7), (3,5), (3,6), (3,7)\}$ is a BB, while adding the dashed state pairs (0, 5) and (1, 4) to $\mathcal{R}$ results in both an SBB and a BB

Example 6 Consider the transition system G in Fig. 7, where the relation $\mathcal{R} = \{(0,4),(0,6),(1,7),(2,4),(2,7),(3,5),(3,6),(3,7)\}$ is a BB. The state pairs (1, 7), $(2,7),(3,7),(3,5),(3,6) \in \mathcal{R}$ since 3 and 7 are terminal states. The state pair $(0,6) \in \mathcal{R}$ since $0 \xrightarrow{\tau} 1$ and $6 \xrightarrow{\tau} 7$ are matching transitions where $(1,7) \in \mathcal{R}$. In the same way, by symmetry it follows that $(2,4) \in \mathcal{R}$. Now, the state pair $(0,4) \in \mathcal{R}$ because $0 \xrightarrow{\tau} 1$ is matched by $4 \xrightarrow{\tau^*} 6 \xrightarrow{\tau} 7$ and $(0,6),(1,7) \in \mathcal{R}$, while $4 \xrightarrow{\tau} 5$ is matched by $0 \xrightarrow{\tau^*} 2 \xrightarrow{\tau} 3$ and $(2,4),(3,5) \in \mathcal{R}$.

This condition is not enough for $\mathcal{R}$ to be an SBB where also the state pairs (0, 5) and (1, 4) need to be added. Then, $(0,4) \in \mathcal{R}$ since $0 \xrightarrow{\tau} 1$ is still matched by $4 \xrightarrow{\tau^*} 6 \xrightarrow{\tau} 7$ but now including the state pairs $(0,5),(0,6),(1,7) \in \mathcal{R}$, while $4 \xrightarrow{\tau} 5$ is matched by $0 \xrightarrow{\tau^*} 2 \xrightarrow{\tau} 3$ and $(1,4),(2,4),(3,5) \in \mathcal{R}$. The state pairs (0, 5) and (1, 4) are introduced in $\mathcal{R}$ with the same type of arguments as $(0,4) \in \mathcal{R}$. □

3.3 Largest stutter-branching and branching bisimulation

To obtain a unified equivalence and block transition based bisimulation formulation, the largest branching bisimulation (LBB), called branching bisimilarity, is generated. First it will be shown that the largest stutter branching bisimulation (LSBB) is an equivalence relation, followed by the observation that LSBB and LBB coincide.

Lemma 1 (Composition of stutter-branching bisimulation) The composition of two stutter-branching bisimulations $\mathcal{R}_1$ and $\mathcal{R}_2$ with state pairs $x\mathcal{R}_1 y$ and $y\mathcal{R}_2 z$, defined as

$$x\mathcal{R}_1 \circ \mathcal{R}_2 z := \exists y : x\mathcal{R}_1 y \wedge y\mathcal{R}_2 z,$$

is also a stutter-branching bisimulation.

Proof Consider a transition system with three sequences of stutter transitions $x = x_0 \xrightarrow{\tau} x_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} x_n = x''$, $y = y_0 \xrightarrow{\tau} y_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} y_m = y''$, and $z = z_0 \xrightarrow{\tau} z_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} z_p = z''$. Since multiple states $y_j, j \in \mathbb{N}_{0,m}$ are involved in the composition of $\mathcal{R}_1$ and $\mathcal{R}_2$, see Definition 3 and Fig. 5, and

$$x\mathcal{R}_1 \circ \mathcal{R}_2 z \equiv \bigwedge_{j=0}^{m} x\mathcal{R}_1 \circ \mathcal{R}_2 z,$$

this composition is equivalently extended to

$$x\mathcal{R}_1 \circ \mathcal{R}_2 z \equiv \bigwedge_{j=0}^{m} x\mathcal{R}_1 \circ \mathcal{R}_2 z = \exists y, y_1, \ldots, y'' : \bigwedge_{j=0}^{m} x\mathcal{R}_1 y_j \wedge y_j \mathcal{R}_2 z.$$

Based on this extended composition, the following equivalences and implications show that the first transfer property $p_{\mathcal{R}}^{BBx}$ in Definition 3 is also valid for the composition $\mathcal{R}_1 \circ \mathcal{R}_2$.

$$
\begin{aligned}
p^{BBx}_{\mathcal{R}_1\circ\mathcal{R}_2} &= \forall x'', x' : x \xrightarrow[*\mathcal{R}_1\circ\mathcal{R}_2 z]{\tau^*} x'' \xrightarrow{a} x' \\
&= \forall x_1, \ldots, x'', x' : x \xrightarrow{\tau^*} x'' \xrightarrow{a} x' \wedge \bigwedge_{i=0}^{n} x_i \,\mathcal{R}_1\circ\mathcal{R}_2\, z \\
&= \forall x_1, \ldots, x'', x', \exists y, y_1, \ldots, y'' : x \xrightarrow{\tau^*} x'' \xrightarrow{a} x' \wedge \bigwedge_{i=0}^{n}\bigwedge_{j=0}^{m} (x_i \mathcal{R}_1 y_j \wedge y_j \mathcal{R}_2 z) \\
&\Rightarrow \forall x_1, \ldots, x'', x', \exists y, y_1, \ldots, y'' : x \xrightarrow{\tau^*} x'' \xrightarrow{a} x' \wedge \bigwedge_{i=0}^{n} x_i \mathcal{R}_1 y \wedge \bigwedge_{j=0}^{m} y_j \mathcal{R}_2 z \\
&\Rightarrow \exists y, y_1, \ldots, y'', y' : y \xrightarrow{\tau^*} y'' \xrightarrow{a} y' \wedge \bigwedge_{j=0}^{m} x'' \mathcal{R}_1 y_j \wedge \bigwedge_{j=0}^{m} y_j \mathcal{R}_2 z \wedge x' \mathcal{R}_1 y' \\
&\Rightarrow \exists y'', y', z_1, \ldots, z'', z' : z \xrightarrow{\tau^*} z'' \xrightarrow{a} z' \wedge \bigwedge_{k=0}^{p} (x'' \mathcal{R}_1 y'' \wedge y'' \mathcal{R}_2 z_k) \wedge x' \mathcal{R}_1 y' \wedge y' \mathcal{R}_2 z' \\
&= \exists z_1, \ldots, z'', z' : z \xrightarrow{\tau^*} z'' \xrightarrow{a} z' \wedge \bigwedge_{k=0}^{p} x'' \,\mathcal{R}_1\circ\mathcal{R}_2\, z_k \wedge x' \,\mathcal{R}_1\circ\mathcal{R}_2\, z' \\
&= \exists z'', z' : z \xrightarrow[x''\mathcal{R}_1\circ\mathcal{R}_2 *]{\tau^*} z'' \xrightarrow{a} z' \wedge x' \,\mathcal{R}_1\circ\mathcal{R}_2\, z'
\end{aligned}
$$

The second transfer property in Definition 3 is shown in the same way, followed by the conclusion that the composition $\mathcal{R}_1\circ\mathcal{R}_2$ is also a stutter-branching bisimulation. □

This lemma is the main contribution in the following equivalence proof of stutter-branching bisimilarity.

Theorem 1 (Stutter-branching bisimilarity is an equivalence relation) Consider a transition system with a stutter-branching bisimulation $\mathcal{R}$, according to Definition 3. The largest stutter-branching bisimulation, also called stutter-branching bisimilarity, is an equivalance relation.

Proof What needs to be shown is reflexivity, symmetry, and transitivity of the stutter-branching bisimilarity.

Reflexivity: Since the relation $\mathcal{R}$ with state pair $x\mathcal{R}x$, obtained by replacing state y with x in Definition 3, is also an SBB, stutter-branching bisimilarity is reflexive.

Symmetry: Introduce the inverse relation $\mathcal{R}^{-1} = \{(y, x) \mid x\mathcal{R}y\}$ in the transfer properties in Definition 3. Since the resulting inverse relation, where $y\mathcal{R}^{-1}x$, is also an SBB, stutter-branching bisimilarity is symmetric.

Transitivity: The transitivity of stutter-branching bisimilarity follows, since the composition of two SBBs is also an SBB according to Lemma 1. □

Given the fact that LSBB is an equivalence relation in the same way as LBB is an equivalence relation (Van Glabbeek and Weijland 1996; Basten 1996), the conclusion in the following proposition is that LSBB and LBB are equal.

Proposition 3 (Branching and stutter-branching bisimilarity are equal) The largest branching bisimulation based on Definition 2 is equal to the largest stutter-branching bisimulation based on Definition 3.

Proof Since LBB is an equivalence relation, BB related states are joined in equivalence classes, here also called block states. Thus, a transition $x \xrightarrow{\tau} x'$ between any related states in the same block state means that both $x\mathcal{R}y$ and $x'\mathcal{R}y$, where y is any state in the same block. The conclusion is therefore that all transitions that belong to the same block state are stutter transitions. For two sequences of stutter transitions $x = x_0 \xrightarrow{\tau} x_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} x_n = x''$ and $y = y_0 \xrightarrow{\tau} y_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} y_m = y''$, this implies that $x_i\mathcal{R}y_j$ for $i \in \{0, \ldots, n\} = \mathbb{N}_{0,n}$ and $j \in \{0, \ldots, m\} = \mathbb{N}_{0,m}$.

Obviously, this condition for LBB is stronger but includes the required condition to be an SBB, according to Definition 3, see also Eq. 6. The conclusion is therefore that LBB is an SBB. Since any SBB is also a BB, according to Proposition 2, this is also valid for the largest SBB, implying that LSBB is an LBB. Thus, for any transition system, LSBB is equal to LBB, and in other words, stutter branching bisimilarity is equal to branching bisimilarity. □

As observed in Example 4, the more transitions that are extended to stutter transitions, the more states are related to each other. The only difference between BB and SBB is indeed that some state pairs representing stutter transitions included in SBB are not required in BB, cf. Equation 7. LBB generated by the union of all BBs includes on the other hand all possible stutter transitions, in the same way as for LSBB. Thus, no transitions that can be extended to stutter transitions by adding more state pairs remain, neither for LBB nor LSBB. This implies that the remaining transitions are visible, either because the transition label $a \neq \tau$ or the source and target states are not related to each other. The latter case means that the states have different state labels, or there are restrictions in the BB and SBB transfer properties due to different future branches, as illustrated in the end of Example 4. In the next example, LBB is generated based on the SBB transfer properties in Definition 3.

Example 7 The BB in Example 4 and Fig. 4 is indeed an LBB, and the relation is therefore an equivalence relation. The corresponding partition is

$$\Pi = \{\{0\}, \{1, 2\}, \{3, 4\}, \{5, 6, 7\}, \{8, 9\}\},$$

which results in the quotient transition system $G/\!\sim_b$ in Fig. 8. The same equivalence relation is obtained by the SBB transfer properties in Definition 3. For instance $(5, 5) \in \mathcal{R}$, since $5 \xrightarrow{\tau} 6 \xrightarrow{\tau} 9$ and $5 \xrightarrow{\tau} 7 \xrightarrow{\tau} 8$ are matching paths, and $(6, 5), (6, 7), (5, 7), (9, 8) \in \mathcal{R}$. □

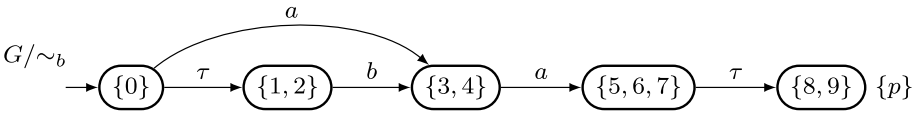


Fig. 8 Quotient transition systems $G/\!\sim_b$ of the transition system G in Fig. 4

3.4 Relation between stutter-branching, semi-branching and stutter bisimulation

To obtain a further perspective on SBB, the relation to some other bisimulations is finally evaluated.

Semi-branching bisimulation It has been shown that SBB coincides with BB when the largest relations are considered. This technique was also used to prove that BB is indeed an equivalence relation by starting with a more flexible relation, called *semi-branching bisimulation*. In Van Glabbeek and Weijland (1996); Basten (1996) it was shown that the largest BB and the largest semi-BB coincide.

There is, however, an interesting difference between the two modified BBs. Semi-BB is more flexible, meaning that there are semi-BBs that are not BBs, while SBB is more restrictive, meaning that SBB is also a BB. Compared to semi-BB, it can be argued that SBB has a closer connection to branching bisimulation, because stutter transition is the basic element in both BB and SBB, while an extended stutter transition is introduced in semi-BB.

Stutter bisimulation In the introduction of SBB, we mentioned that there is a close connection between SBB and stutter bisimulation (Browne et al. 1988; Nicola and Vaandrager 1995). The following definition, based on the formulation in Nicola and Vaandrager (1995), but adapted to the notations introduced in Definition 3, confirms this connection.

Definition 4 (Stutter bisimulation) Consider a transition system with state space X and a relation $\mathcal{R} \subseteq X \times X$. Based on the definition of branching bisimulation in Definition 2 and the additional notations introduced in Definition 3, the relation $\mathcal{R}$ is a *stutter bisimulation* when the event a is replaced with τ, and the transfer properties in Definition 2 are replaced with

$$p_{\mathcal{R}}^{BBx} := \forall x' : x \xrightarrow{\tau} x' \Rightarrow \exists y'', y' : y \xrightarrow[x\mathcal{R}*]{\tau^*} y'' \xrightarrow{\tau} y' \wedge x'\mathcal{R}y',$$
$$p_{\mathcal{R}}^{BBy} := \forall y' : y \xrightarrow{\tau} y' \Rightarrow \exists x'', x' : x \xrightarrow[*\mathcal{R}y]{\tau^*} x'' \xrightarrow{\tau} x' \wedge x'\mathcal{R}y'.$$

□

In the original definition of stutter bisimulation, no transition labels are involved, which in the framework presented here is equivalent to adding the hidden transition label τ at all transitions. Based on SBB, the same stutter bisimulation definition as in Definition 4 is obtained by replacing a with τ and removing the optional initial τ transitions in the premises, meaning that $x = x''$ in $p_{\mathcal{R}}^{BBx}$ and $y = y''$ in $p_{\mathcal{R}}^{BBy}$. We also observe that the added stutter transition condition in the consequences of the transfer properties for SBB in Definition 3 (not included in BB) is also included in stutter bisimulation, although then limited to Kripke structures.

Replacing the transition label τ in the matching transition in Definition 4 with a is indeed a possible alternative definition compared to SBB, which generates the same largest bisimulation as the largest BB and SBB. The reason why we still suggest SBB is the symmetry

between the premises and the consequences that is achieved in Definition 5, which is the basis for the unification that is achieved in Section 5.

To summarize, SBB both unifies the definition of branching bisimulation and stutter bisimulation, and simplifies the framework due to the straightforward equivalence proof. Based on these arguments, one can argue that SBB is a more natural definition of BB than the original one. This is, however, up to the bisimulation community to decide. This is, however, up to the bisimulation community to decide.

Relation to strong bisimulation The formulation of SBB has also similarities with SB. This becomes more clear when the largest SBB is considered. According to the proof of Proposition 3, all states in the stutter paths are then related to each other as $x_i \mathcal{R} y_j$ for $i \in \mathbb{N}_{0,n}$ and $j \in \mathbb{N}_{0,m}$. Introducing the corresponding partition Π, any relation $x\mathcal{R}y$ can also be expressed as $y \in \Pi(x)$. In the following definition, the largest SBB is expressed by this partition in a way that clearly mimics strong bisimulation.

Definition 5 (Largest stutter-branching bisimulation) By introducing the notation

$$\diagup x' : x \underset{\Pi}{\overset{a}{\twoheadrightarrow}} x' := \diagup x'', x' : x \overset{\tau^*}{\rightarrow} x'' \overset{a}{\rightarrow} x', \quad \diagup \in \{\forall, \exists\},$$

where $\Pi(x)$ is the block state that includes all states related to state x, the transfer properties in Definition 3 for the largest stutter-branching bisimulation are reformulated as

$$p_{\mathcal{R}}^{BBx} := \forall x' : x \underset{\Pi}{\overset{a}{\twoheadrightarrow}} x' \Rightarrow \exists y' : y \underset{\Pi}{\overset{a}{\twoheadrightarrow}} y' \wedge y' \in \Pi(x'),$$
$$p_{\mathcal{R}}^{BBy} := \forall y' : y \underset{\Pi}{\overset{a}{\twoheadrightarrow}} y' \Rightarrow \exists x' : x \underset{\Pi}{\overset{a}{\twoheadrightarrow}} x' \wedge y' \in \Pi(x').$$

□

In the next section a similar SB related formulation will be given for WS, before these three bisimulations are joined into one generic formulation in Section 5.

4 Weak Bisimulation

As already mentioned, WB is less restrictive than BB, which results in less block states and a coarser state partition. Before this is illustrated by a small example, a definition of WB on the same form as SB is introduced, see Lemma 4.2.9 in Sangiorgi (2012).

Definition 6 (Weak bisimulation) Given a transition system $G = \langle X, \Sigma, T, I, AP, \lambda \rangle$, a binary relation $\mathcal{R} \subseteq X \times X$ is a *weak bisimulation* (WB) if, for any states $x, y \in X$ and event $a \in \Sigma$, the implication

$$x\mathcal{R}y \Rightarrow \lambda(x) = \lambda(y) \wedge p_{\mathcal{R}}^{WBx} \wedge p_{\mathcal{R}}^{WBy},$$

holds, using the notation

$$\nearrow x' : x \overset{a}{\twoheadrightarrow} x' := \begin{cases} \nearrow x', x'', x''' : x \overset{\tau^*}{\to} x'' \overset{a}{\to} x''' \overset{\tau^*}{\to} x', & a \neq \tau, \\ \nearrow x' : x \overset{\tau^*}{\to} x', & a = \tau, \end{cases} \qquad \nearrow \in \{\forall, \exists\}$$

in the transfer properties

$$p_{\mathcal{R}}^{WBx} := \forall x' : x \overset{a}{\twoheadrightarrow} x' \Rightarrow \exists y' : y \overset{a}{\twoheadrightarrow} y' \wedge x' \,\mathcal{R}\, y',$$
$$p_{\mathcal{R}}^{WBy} := \forall y' : y \overset{a}{\twoheadrightarrow} y' \Rightarrow \exists x' : x \overset{a}{\twoheadrightarrow} x' \wedge x' \,\mathcal{R}\, y'.$$

Related states $(x, y) \in \mathcal{R}$ are said to be *weakly bisimilar*, denoted $x \sim_w y$. □

Example 8 For the transition system G in Fig. 9, branching bisimulation does not give any reduction, while the first two states are joined in the weak bisimular quotient transition system $G/\!\sim_w$. The reason is that WB does not differentiate between the direct path to the final state $0 \overset{a}{\to} 3$ and the path via state 1 which also has an alternative *b* transition to the final state. This reduction is not accepted by BB that preserves the branching information. □

5 Unified bisimulation relation and equivalence

The three bisimulations we have presented in this paper, where stutter bisimulation has been incorporated in BB, can all three be formulated in the same way by introducing, for a transition label *a*, a generic transition operator $\overset{a}{\hookrightarrow}$. This results in the following generic bisimulation relation.

Definition 7 (Generic bisimulation relation) Given a transition system $G = \langle X, \Sigma, T, I, AP, \lambda \rangle$, a binary relation $\mathcal{R} \subseteq X \times X$ is a *generic bisimulation* if, for any states $x, y \in X$ and event $a \in \Sigma$, the implication

$$x \mathcal{R} y \Rightarrow \lambda(x) = \lambda(y) \wedge p_{\mathcal{R}}^{Bx} \wedge p_{\mathcal{R}}^{By}$$

holds, where the transfer properties are

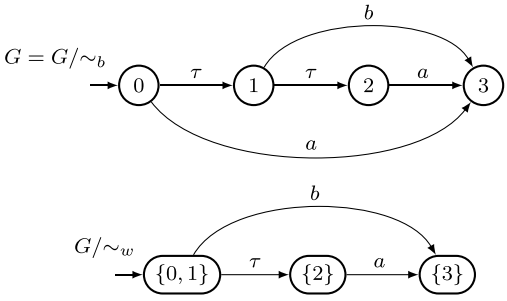


Fig. 9 Transition system G that is equivalent to the branching bisimilar quotient transition system $G/\!\sim_b$ and the coarser weak bisimilar quotient transition system $G/\!\sim_w$

$$p_{\mathcal{R}}^{Bx} := \forall x' : x \overset{a}{\hookrightarrow} x' \Rightarrow \exists y' : y \overset{a}{\hookrightarrow} y' \wedge x'\mathcal{R}y',$$
$$p_{\mathcal{R}}^{By} := \forall y' : y \overset{a}{\hookrightarrow} y' \Rightarrow \exists x' : x \overset{a}{\hookrightarrow} x' \wedge x'\mathcal{R}y'.$$

Related states $(x, y) \in \mathcal{R}$ are said to be *bisimilar*, denoted $x \sim y$. Every terminal state is assumed to include a self-loop labeled by an event not equal τ. □

5.1 Unified bisimulation equivalence

A largest bisimulation relation can be formulated based on the generic bisimulation in Definition 7, resulting in an equivalence relation, see Proposition 1 and Theorem 1. An alternative formulation of BB was presented in Lennartson and Noori-Hosseini (2018), directly based on a state partition that generates an equivalence formulation. This formulation is here called bisimulation equivalence, and will in this section be proved to be equivalent to the largest generic bisimulation relation, but also coupled to the specific bisimulation relations presented in this paper. More comments on relations to other formulations and algorithms are given in the end of this section.

To be able to show the equivalence between the following bisimulation equivalence for BB, the SBB proposed in this paper is used. We take the freedom from now to exclude the stutter prefix in stutter branching bisimulation, mainly because our focus is on the largest version that is equal for BB and SBB, but also since SBB is argued to be a natural and unified BB formulation, see Section 3.4. Since the unified transition relation in Definition 7 is achieved for BB first when the largest BB is considered and the partition Π is introduced, this relation formulation is from now also used in Definition 7, specifically meaning that $x'\mathcal{R}y'$ is replaced with $y' \in \Pi(x')$.

Definition 8 (Bisimulation equivalences) Given a transition system $G = \langle X, \Sigma, T, I, AP, \lambda \rangle$, a partition Π, which for any state $x \in X$ satisfies the greatest fixed point

$$\Pi(x) = \{y \in X | \lambda(y) = \lambda(x) \wedge T_\Pi(x) = T_\Pi(y)\}, \tag{8}$$

is a *bisimulation equivalence* where the set of block transitions

$$T_\Pi(x) = \{\Pi(x) \overset{a}{\rightarrow} \Pi(x') | \exists x', a : x \overset{a}{\hookrightarrow} x'\}. \tag{9}$$

(i) It is a *strong bisimulation equivalence* when the generic transition operator $\overset{a}{\hookrightarrow}$ is replaced by $\overset{a}{\rightarrow}$, and states $x, y \in \Pi(x)$ are strongly bisimilar, denoted $x \sim_s y$.
(ii) It is a *weak bisimulation equivalence* when the generic transition operator $\overset{a}{\hookrightarrow}$ is replaced by $\overset{a}{\Longrightarrow}$, and states $x, y \in \Pi(x)$ are weakly bisimilar, denoted $x \sim_w y$.
(iii) It is a *branching bisimulation equivalence* when the generic transition operator $\overset{a}{\hookrightarrow}$ is replaced by $\overset{a}{\underset{\Pi}{\Rightarrow}}$, and states $x, y \in \Pi(x)$ are branching bisimilar, denoted $x \sim_b y$. □

The equivalence between these bisimulation equivalences and the corresponding largest bisimulation relations will now be proved.

Theorem 2 (Bisimulation equivalences and largest bisimulation relations) The bisimulation equivalence given by the fixed point Eq. 8 in Definition 8 is equivalent to the largest bisimulation relation

$$\mathcal{R} = \{(x, y) | \lambda(x) = \lambda(y) \wedge p_{\mathcal{R}}^{Bx} \wedge p_{\mathcal{R}}^{By}\}, \tag{10}$$

where the transfer properties $p_{\mathcal{R}}^{Bx}$ and $p_{\mathcal{R}}^{By}$ are defined in Definition 7, but with $x'\mathcal{R}y'$ replaced by $y' \in \Pi(x')$. Strong, weak, and branching bisimilarity are obtained by replacing the generic transition operator $\overset{a}{\hookrightarrow}$ with $\overset{a}{\rightarrow}$, $\overset{a}{\twoheadrightarrow}$, and $\overset{a}{\underset{\Pi}{\twoheadrightarrow}}$, respectively.

Proof The equality $T_\Pi(x) = T_\Pi(y)$ in Eq. 8 means that $\Pi(x) = \Pi(y)$ and $\Pi(x') = \Pi(y')$, and therefore $y \in \Pi(x)$ and $y' \in \Pi(x')$. Furthermore, $T_\Pi(x) = T_\Pi(y)$ can equivalently be formulated as $\big(T_\Pi(x) \subseteq T_\Pi(y)\big) \wedge \big(T_\Pi(y) \subseteq T_\Pi(x)\big)$. The definition of subsets applied to $T_\Pi(x) \subseteq T_\Pi(y)$ then gives the equivalent formulation

$$\forall x', a : \Pi(x) \overset{a}{\rightarrow} \Pi(x') \in T_\Pi(x) \Rightarrow \Pi(x) \overset{a}{\rightarrow} \Pi(x') \in T_\Pi(y)$$

Furthermore, the fact that $y \in \Pi(x) = \Pi(y)$ and $y' \in \Pi(x') = \Pi(y')$ results in one more equivalent expression

$$\forall x', a : \Pi(x) \overset{a}{\rightarrow} \Pi(x') \in T_\Pi(x) \Rightarrow \exists y' : \Pi(y) \overset{a}{\rightarrow} \Pi(y') \in T_\Pi(y) \wedge y' \in \Pi(x')$$

Finally, the definition of $T_\Pi(x)$ Eq. 9 in Definition 8 gives the equivalent implication

$$\forall x', a : x \overset{a}{\hookrightarrow} x' \Rightarrow \exists y' : y \overset{a}{\hookrightarrow} y' \wedge y' \in \Pi(x')$$

This corresponds to the transfer properties $p_{\mathcal{R}}^{Bx}$ and $p_{\mathcal{R}}^{By}$ in Definition 7 with $y' \in \Pi(x')$. Hence, the greatest fixed point of $\Pi(x)$ in Eq. 8 can equivalently be expressed as

$$\Pi(x) = \{y \in X | \lambda(x) = \lambda(y) \wedge p_{\mathcal{R}}^{Bx} \wedge p_{\mathcal{R}}^{By}\},$$

which corresponds to the largest bisimulationl relation Eq. 10. □

An interesting observation in this proof is that the transfer property $p_{\mathcal{R}}^{Bx}$ equivalently can be formulated as the subset expression $T_\Pi(x) \subseteq T_\Pi(y)$, and that the conjunction of $p_{\mathcal{R}}^{Bx}$ and $p_{\mathcal{R}}^{By}$ corresponds to the equality $T_\Pi(x) = T_\Pi(y)$. The equivalence relation is finally obtained by generating the greatest fixed point of these expressions conjuncted with the state label equality $\lambda(x) = \lambda(y)$. This equivalence relation, now formulated as the greatest fixed point of $\Pi(x)$, is obtained by a simple monotonicity property, shown in the following proposition.

Proposition 4 (Bisimulation greatest fixed point) The greatest fixed point of Eq. 8 in Definition 8 is obtained by iterating

$$\Pi_{k+1}(x) = \{y \in X | T_{\Pi_k}(x) = T_{\Pi_k}(y)\} \tag{11}$$

until $\Pi_{k+1}(x) = \Pi_k(x)$, with the initial partition $\Pi_0(x) = \{y | \lambda(x) = \lambda(y)\}$.

Proof The block $\Pi_{k+1}(x) = \{y \in X | T_{\Pi_k}(x) = T_{\Pi_k}(y)\}$, and therefore $\Pi_{k+1}(x) \preceq \Pi_k(x)$ for all states $x \in X$ and $k \geq 0$. This holds since

$$\Pi_{k+1}(x) = \{y \in X | \exists x', y', a : \{\big(\Pi_k(x), a, \Pi_k(x')\big) = \big(\Pi_k(y), a, \Pi_k(y')\big)\} \preceq \Pi_k(x)$$

Obviously, $\Pi_{k+1}(x)$ is further restricted compared to $\Pi_k(x)$ by also requiring the next block states to be equal, i.e. $\Pi_k(x') = \Pi_k(y')$. Thus, the iteration in Eq. 11 generates a monotonically decreasing partition Π_k as k increases, until the greatest fixed point $\Pi_{k+1} = \Pi_k$ is reached, according to Knaster–Tarski's famous fixed-point theorem (Tarski 1955). □

Example 9 For the transition system G in Fig. 10 and the branching block transition

$$T_\Pi(x) = \{\Pi(x) \xrightarrow{a} \Pi(x') | \exists x', a : x \overset{a}{\underset{\Pi}{\twoheadrightarrow}} x'\},$$

the first fixed-point iteration of Eq. 11 gives for $\Pi_0 = X = \{0, 1, 2, 3, 4, 5\}$

$$\begin{aligned} T_{\Pi_0}(0) &= \{X \xrightarrow{a} X, X \xrightarrow{b} X\}, \\ T_{\Pi_0}(1) &= T_{\Pi_0}(2) = T_{\Pi_0}(3) = \{X \xrightarrow{c} X\}, \\ T_{\Pi_0}(4) &= T_{\Pi_0}(5) = \{X \xrightarrow{d} X\}, \end{aligned}$$

resulting in $\Pi_1 = \{\{0\}, \{1, 2, 3\}, \{4, 5\}\}$. The second iteration gives

$$\begin{aligned} T_{\Pi_1}(0) &= \{\{0\} \xrightarrow{a} \{1, 2, 3\}, \{0\} \xrightarrow{b} \{1, 2, 3\}\}, \\ T_{\Pi_1}(1) &= T_{\Pi_1}(2) = T_{\Pi_1}(3) = \{\{1, 2, 3\} \xrightarrow{c} \{4, 5\}\}, \\ T_{\Pi_1}(4) &= T_{\Pi_1}(5) = \{\{4, 5\} \xrightarrow{d} \{4, 5\}\}, \end{aligned}$$

which results in the fixed point $\Pi_2 = \Pi_1$. The obtained branching bisimilar quotient transition system $G/\sim_b$ is shown in Fig. 10. □

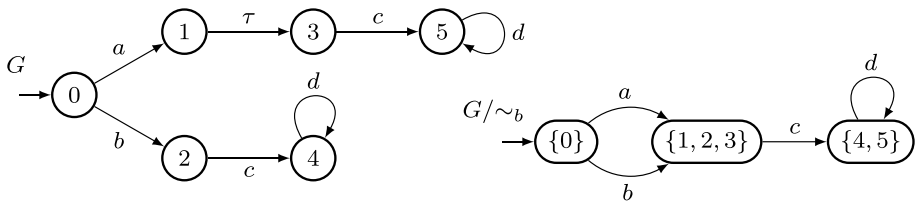


Fig. 10 Transition system G and its branching bisimilar quotient transition system $G/\sim_b$

5.2 Alternative equivalence based definition of bisimulation

To summarize what we have achieved so far in this paper, an alternative and unified formulation of bisimulation has been introduced, including both strong, weak, branching and stutter bisimulation, formulated in the same way, always including both state and transition labels. The unified formulation is also short and concise compared to the ordinary definition of a bisimulation that is relation based. Our partition formulation consists of a minimum of information, because it is an equivalence relation. The definition of bisimulation is for sure more sofisticated, but also easy to misunderstand. The fact that a bisimulation can be both non-symmetric, non-transitive, and non-reflexive, while the largest bisimulation called bisimilarity is an equivalence relation and therefore is always a reflexive, symmetric, and transitive relation, is one example of complications that are avoided by focusing directly on an equivalence relation.

One strength of our formulation is that it is based on a block transition relation $T_{\Pi}(x) = \{\Pi(x) \xrightarrow{a} \Pi(x') | \exists x', a : x \xrightarrow{a} x'\}$. To our best knowledge we have not seen that formulation before, and it highlights the state reduction in a very clear way. The fixed point is also very condensed, where all states in a block requires equal state labels and equal block transition relations, i.e. $\Pi(x) = \{y \in X | \lambda(y) = \lambda(x) \wedge T_{\Pi}(x) = T_{\Pi}(y)\}$. The focus on equal future behavior is obvious.

The fact that the unified bisimulation is a fixed point is not unique. The relation based fixed point was introduced already by Park (1981) and Milner (1989), and more or less all efficient algorithms are based on a partition formulation of the state space. An interesting survey is given in Aceto et al. (2012). The strengths of our formulation are the simplicity and unification of the proposed bisimulation formulation. Although the algorithmic performance is not the main point, the fixed point computation can be efficient, sometimes even faster than Groote's famous stutter and BB algorithm (Groote and Vaandrager 1990), especially when parallel computation is utilized. This result is not surprising, since the idea behind our alternative formulation is the signature algorithm, proposed in Blom and Orzan (2003), where our block states are closely related to their signatures. Our formulation includes state labels, and can therefore also be applied to stutter bisimulation, and the proofs in this paper are simpler and more generic compared to the signature formulation.

6 Divergent Sensitive Branching Bisimulation

To guarantee that temporal logic properties are preserved when reduced models are generated based on BB, infinite paths of hidden τ transitions require a special treatment. The reason is that such infinite paths are not preserved by BB.

Divergent paths and divergent blind branching bisimulation For a given BB relation $\mathcal{R}$, an infinite path of τ transitions, $x = x_0 \xrightarrow{\tau} x_1 \xrightarrow{\tau} x_2 \xrightarrow{\tau} \cdots$, is said to be *divergent* when all states in the path are related to each other, i.e. $x_i \mathcal{R} x$ for all $i > 0$. All states x_i included in a divergent path are called *divergent states*.

Since all transitions in a divergent path are labelled by τ and are related to each other, it means that divergent paths do not generate any visible transitions in a BB abstraction. In other words, divergent paths are hidden by BB, and BB is said to be *divergent blind* (Nicola and Vaandrager 1995). In Fig. 11 this is illustrated by a transition system G with two divergent paths, $0 \xrightarrow{\tau} 1 \xrightarrow{\tau} 2 \xrightarrow{\tau} 1 \xrightarrow{\tau} 2 \xrightarrow{\tau} \cdots$ and $0 \xrightarrow{\tau} 1 \xrightarrow{\tau} 2 \xrightarrow{\tau} 3 \xrightarrow{\tau} 3 \xrightarrow{\tau} \cdots$, both of which are hidden in the branching bisimilar quotient transition system $G/\!\sim_b$.

Divergent sensitive branching bisimulation Since the focus is now on state space abstraction, the largest BB and its related partition Π are considered. The resulting block states are also shown in Fig. 11. To make divergent paths visible, a *divergent sensitive branching bisimulation* (DBB) is introduced. DBB block states are then separated such that either all states in a block are divergent, or no state in a block is divergent. A block that consists of divergent states is called a *divergent block*, and a τ self-loop is added to each divergent block. In Fig. 11, the divergent sensitive branching bisimilar quotient transition system $G/\!\sim_d$ is also shown with its divergent block $\{0, 1, 2, 3\}$, followed by the two non-divergent blocks $\{4\}$ and $\{5\}$.

Stutter cycle block states A cycle $x_0 \xrightarrow{\tau} x_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} x_{n-1} \xrightarrow{\tau} x_0$, where all *n* states in the cycle have the same state label, is called a *stutter cycle*. The states in a stutter cycle are always a part of a divergent path, and all states in the cycle are divergent sensitive branching bisimilar. This follows since a visible transition $x_i \xrightarrow[\Pi]{a} x'$ from any state x_i in a stutter cycle can be matched by a visible transition $x_j \xrightarrow[\Pi]{a} x'$ from any other state x_j in the same cycle. Recall that all state labels are equal in the hidden path from x_j to x_i.

For finite transition systems (finite number of states), divergent paths must contain at least one stutter cycle. Divergent paths can be preserved in DBB by first collapsing all stutter cycles into stutter cycle block (SCB) states, one SCB state for each cycle. The stutter cycles are computed as strongly connected components by Tarjan's linear time algorithm (Tarjan 1972).

Temporary visible ↻ *self-loops* Before each stutter cycle, a number of non-cyclic states can also be a part of a divergent path. For instance, the divergent paths in transition system G in Fig. 11 include the non-cyclic state 0 followed by two stutter cycles. To include such non-cyclic states in the divergent paths and also be able to use the ordinary BB partition compu-

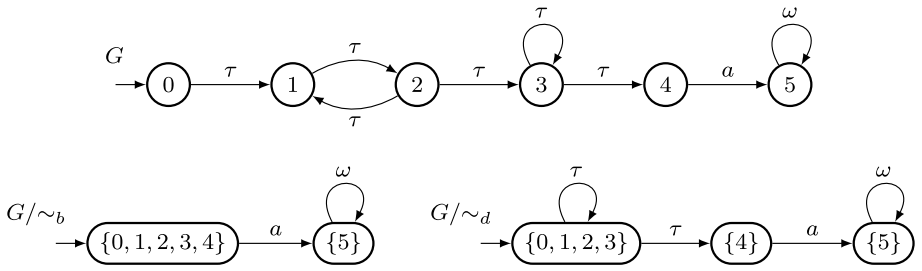


Fig. 11 Transition system G and its branching bisimilar quotient transition system $G/\!\sim_b$ and divergence sensitive branching bisimilar quotient transition system $G/\!\sim_d$

tation for DBB, τ transitions from the SCB states to a sink-state with a unique state label can be introduced (Nicola and Vaandrager 1995). A simple example illustrating this approach is presented in Section 7.8.1 in Baier and Katoen (2008). This is one way to introduce a marker for the divergent paths. Alternatively, temporary visible $\circlearrowleft$ self-loops are added to the SCB states, and the remaining $\circlearrowleft$ events are replaced by τ after the ordinary BB partition has been computed (Lennartson and Noori-Hosseini 2018). This more simple strategy to generate a DBB, based on visible $\circlearrowleft$ self-loops, is illustrated in the following example.

Example 10 Consider the transition system G in Fig. 12, where the three stutter cycles are collapsed to SCB states in G_{SCB}. In this model the temporary visible $\circlearrowleft$ self-loops are also added to the SCB states $\{0,2,7\}$, $\{4\}$, and $\{5\}$. Based on the state labels the initial partition in G_{SCB} is $\Pi_0 = \{\{0,2,7\},\{1,3,4,5,6\}\} \stackrel{\text{def}}{=} \{B_0, B_1\}$, which generates the following block transitions

$$
\begin{aligned}
&T_{\Pi_0}(0) = T_{\Pi_0}(2) = T_{\Pi_0}(7) = \{B_0 \stackrel{\circlearrowleft}{\to} B_0, B_0 \stackrel{\tau}{\to} B_1\},\\
&T_{\Pi_0}(1) = T_{\Pi_0}(6) = \{B_1 \stackrel{\tau}{\to} B_0\},\\
&T_{\Pi_0}(3) = T_{\Pi_0}(4) = T_{\Pi_0}(5) = \{B_1 \stackrel{\circlearrowleft}{\to} B_1, B_1 \stackrel{\tau}{\to} B_0\},
\end{aligned}
$$

resulting in the updated partition $\Pi_1 = \{\{0,2,7\},\{1,6\},\{3,4,5\}\} \stackrel{\text{def}}{=} \{B_0, B_1, B_2\}$. The second iteration gives

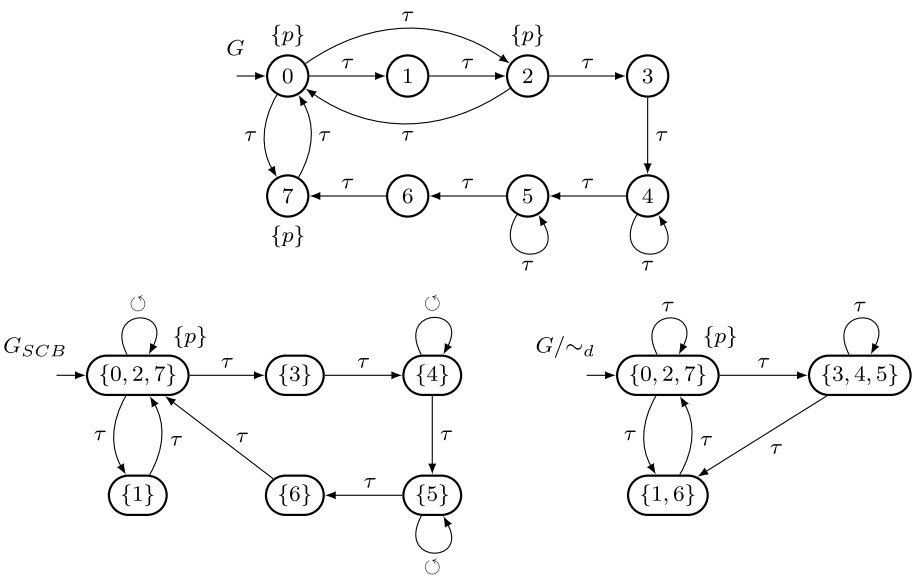


Fig. 12 Transition systems G and G_{SCB} and the divergence sensitive branching bisimilar quotient transition system $G/\sim_d$. The stutter cycles in G are collapsed to SCB states in G_{SCB}, where $\circlearrowleft$ self-loops are also added to the SCB states

$$T_{\Pi_1}(0) = T_{\Pi_1}(2) = T_{\Pi_1}(7) = \{B_0 \overset{\circlearrowleft}{\to} B_0, B_0 \overset{\tau}{\to} B_1, B_0 \overset{\tau}{\to} B_2\},$$
$$T_{\Pi_1}(1) = T_{\Pi_1}(6) = \{B_1 \overset{\tau}{\to} B_0\},$$
$$T_{\Pi_1}(3) = T_{\Pi_1}(4) = T_{\Pi_1}(5) = \{B_2 \overset{\circlearrowleft}{\to} B_2, B_2 \overset{\tau}{\to} B_1\},$$

which results in the fixed point $\Pi_2 = \Pi_1$. The obtained divergent sensitive branching bisimilar quotient transition system $G/\!\sim_d$ is shown in Fig. 12, where the remaining $\circlearrowleft$ self-loops have been changed to τ self-loops, specifying the divergent blocks. □

In the following section, applications and generalizations will be presented where the strength of DBB is demonstrated. More specifically, verification of temporal logic expressions will be shown based on incremental abstraction of modular systems, applied to Petri nets.

7 Incremental abstraction applied to Petri nets

Abstraction of modular transition systems based on DBB is highly dependent on hidden τ events. Local events only included in one subsystem can be hidden. The reason is that such events do not share any communication and synchronization with other subsystems, and therefore can be replaced by the hidden τ event.

Once a number of subsystems have been synchronized, more local events will appear, only involved in the hitherto synchronized subsystems. In this way, more and more τ events are generated after each synchronization. The basic idea behind *incremental abstraction* is to combine this repeated hiding and synchronization mechanism with abstraction based on DBB every time additional local events have been hidden by τ events.

7.1 Incremental abstraction

The formal definition of incremental abstraction is now presented, including synchronous composition and repeated hiding and abstraction. The *synchronous composition* (Hoare 1978) of two transition systems is adapted to τ events. Since only local events are replaced by τ, such events in different subsystems are not synchronized, although they share the same event label.

Definition 9 (Synchronous composition) Let $G_i = \langle X_i, \Sigma_i, T_i, I_i, AP_i, \lambda_i \rangle$, $i = 1, 2$, be two transition systems. The synchronous composition of G_1 and G_2 is defined as

$$G_1 \,\|\, G_2 = \langle X_1 \times X_2, \Sigma_1 \cup \Sigma_2, T, I_1 \times I_2, AP_1 \cup AP_2, \lambda \rangle ,$$

where the transitions

$$\begin{array}{ll} (x_1, x_2) \overset{a}{\to} (x_1', x_2') \in T & \text{if } a \in (\Sigma_1 \cap \Sigma_2) \setminus \{\tau\},\ x_1 \overset{a}{\to} x_1' \in T_1,\ x_2 \overset{a}{\to} x_2' \in T_2, \\ (x_1, x_2) \overset{a}{\to} (x_1', x_2) \in T & \text{if } a \in (\Sigma_1 \setminus \Sigma_2) \cup \{\tau\},\ x_1 \overset{a}{\to} x_1' \in T_1, \\ (x_1, x_2) \overset{a}{\to} (x_1, x_2') \in T & \text{if } a \in (\Sigma_2 \setminus \Sigma_1) \cup \{\tau\},\ x_2 \overset{a}{\to} x_2' \in T_2, \end{array}$$

and $\lambda : X_1 \times X_2 \to 2^{AP_1 \cup AP_2}$. □

Based on this synchronous composition, a modular system including n transition subsystems $G_1, G_2, \ldots, G_n$ is now defined as

$$G = G_1 \,\|\, G_2 \,\|\, \cdots \,\|\, G_n$$

Combined hiding, abstraction, and synchronization A transition system G, where the events in Σ^h are local (only included in G) and therefore hidden and replaced by τ, is denoted G^{Σ^h}. Also recall that for a given partition Π, if a state x in G is equivalent to its block state $\Pi(x)$ in $G/\!\sim$, i.e. $x \sim \Pi(x)$ for all $x \in X$, it means that G and $G/\!\sim$ are equivalent, denoted $G \sim G/\!\sim$. For DBB this equivalence is proven in Noori-Hosseini et al. (2018), where BB is called visible bisimulation. Combining hiding of a set of events Σ^h, followed by the generation of the quotient transition system, results in the *abstracted transition system* $G^{\mathcal{A}^{\Sigma^h}} \stackrel{\text{def}}{=} G^{\Sigma^h}/\!\sim$. This also means that $G^{\Sigma^h} \sim G^{\mathcal{A}^{\Sigma^h}}$.

Consider the synchronized system $(G_1 \,\|\, G_2)^{\Sigma^h}$, where $\Sigma^h = \Sigma_1^h \,\dot\cup\, \Sigma_2^h \,\dot\cup\, \Sigma_{12}^h$ and Σ_i^h includes the local events in G_i, $i = 1, 2$ while the events in Σ_{12}^h become local after the synchronization of G_1 and G_2. The abstraction of this synchronized system can now be incrementally generated as

$$(G_1 \,\|\, G_2)^{\Sigma^h} \sim \left(G_1^{\mathcal{A}^{\Sigma_1^h}} \,\|\, G_2^{\mathcal{A}^{\Sigma_2^h}}\right)^{\mathcal{A}^{\Sigma_{12}^h}},$$

as long as the abstraction is congruent wrt synchronization, which is also shown for DBB in Noori-Hosseini et al. (2018).

This step by step combined hiding, abstraction and synchronization, here called *incremental abstraction*, was proposed by Graf and Steffen (1991) and Tai and Koppol (1993) for WB, and by Flordal and Malik (2009) for nonblocking and controllability verification based on conflict equivalence. From now on, DBB is the default abstraction, and the strength of this incremental abstraction will here be demonstrated on reduction of Petri nets, such that interesting temporal logic properties are preserved.

7.2 Temporal logic including both state and transition labels

Temporal logic is often used to specify system properties, where the two most well known logics are linear temporal logic (LTL) and computation tree logic (CTL). They are complementary in expressive power, while CTL* is more general than both LTL and CTL. These temporal logics are used to specify and verify Kripke structures (Baier and Katoen 2008). An equivalence between stutter bisimulation and $\text{CTL}^*_{\setminus\bigcirc}$ where the next step operator is removed is also shown in Nicola and Vaandrager (1995).

The bisimulation equivalences considered in this paper are based on transition systems, including both state and event labels. Therefore, CTL* is now extended to also include events, by adding a reformulation of the Hennessy-Milner logic (HML) called "HML with Until" (Nicola and Vaandrager 1995). Nicola and Vandraager show that this logic is equivalent to the original event based BB (Van Glabbeek and Weijland 1996). The event based extension of CTL* presented here is called $\text{CTL}^*_{\setminus\bigcirc}$, where the subscript is added in the same ways as in $\text{CTL}^*_{\setminus\bigcirc}$, to denote that the next step operator $\bigcirc$ is removed in $\text{CTL}^*_{\setminus\bigcirc}$. This extended temporal logic is proposed in Lennartson and Noori-Hosseini (2018) to

emphasize the connection between temporal logic and the combined branching and stutter bisimulation equivalence considered in this paper. The syntax of $\mathrm{CTL}^*_{\setminus\bigcirc}$ formulas is separated into a grammar for state formulas and a grammar for path formulas.

Definition 10 (Syntax of $\mathrm{ECTL}^*_{\setminus\bigcirc}$) Given a transition system $G = \langle X, \Sigma, T, I, AP, \lambda\rangle$, state formulas in $\mathrm{ECTL}^*_{\setminus\bigcirc}$ are defined inductively by the grammar

$$\psi :: = p \,|\, \neg\psi \,|\, \psi_1 \wedge \psi_2 \,|\, \psi_1\langle a\rangle\psi_2 \,|\, \exists\varphi,$$

where p is an element in a set of atomic propositions AP, ψ, ψ_1 and ψ_2 are state formulas, φ is a path formula, and $a \in \Sigma \setminus \{\tau\}$. Path formulas in $\mathrm{ECTL}^*_{\setminus\bigcirc}$ are defined inductively by the grammar

$$\varphi :: = \psi \,|\, \neg\varphi \,|\, \varphi_1 \wedge \varphi_2 \,|\, \varphi_1 \mathcal{U} \varphi_2,$$

where ψ is a state formula, while φ, φ_1, and φ_2 are path formulas. □

Semantics of $\mathrm{ECTL}^*_{\setminus\bigcirc}$ For the semantics of $\mathrm{ECTL}^*_{\setminus\bigcirc}$ we refer to the semantics of $\mathrm{CTL}^*_{\setminus\bigcirc}$ in Baier and Katoen (2008), except for the semantics of the state formula $\psi_1\langle a\rangle\psi_2$. A state x satisfies this formula, i.e. $x \vDash \psi_1\langle a\rangle\psi_2$, if and only if there exists a sequence of τ transitions $x = x_0 \xrightarrow{\tau} x_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} x_n$, where $x_k \vDash \psi_1$ for $0 \leq k \leq n$, followed by a transition $x_n \xrightarrow{a} x'$ where $x' \vDash \psi_2$.

We observe that the demand on equal source and target-block states for invisible transitions in BB is replaced by the weaker condition that the source and target states must satisfy the same temporal logic formula ψ_1. A special case of equal temporal logic formula is that the state labels are equal in the source and target states. Also note that the until operator does not care about any event labels included in a transition model. The only way to specify properties related to events is by the event modality $\langle a\rangle$. This modality also has an until property in terms of the state labels, ψ_1 until ψ_2, but the events are here restricted to be invisible τ events before the visible event a occurs and the second state label condition ψ_2 holds.

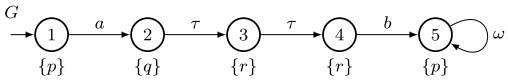
Derived operators A number of derived operators are also introduced. The more specific ones for $\mathrm{ECTL}^*_{\setminus\bigcirc}$ are

$$\begin{array}{lll} \Diamond\varphi = \top\mathcal{U}\varphi, & \Box\varphi = \neg\Diamond\neg\varphi, & \forall\varphi = \neg\exists\neg\varphi, \\ \langle a\rangle\psi = \top\langle a\rangle\psi, & [a]\psi = \neg\langle a\rangle\neg\psi. & \end{array}$$

The first row defines the standard eventually and always modalities, as well as the "for all paths (branches)", the duality of "there exists a path (branch)". The second line specifies that after an arbitrary number of τ transitions there exists a transition labeled a, followed by condition ψ. Finally, the dual modality $[a]$ is defined, where every transition has this property.

Example 11 For the transition system G in Fig. 13, consider the following $\mathrm{ECTL}^*_{\setminus\bigcirc}$ formulas and their evaluation for the different states in G.

Fig. 13 Transition system G



$$
\begin{array}{lcll}
x & \models & p\,\langle a\rangle\, q & \text{holds for } x = 1, \\
x & \models & r\,\langle b\rangle\, p & \text{holds for } x = 3, 4, \\
x & \models & (q \vee r)\,\langle b\rangle \top & \text{holds for } x = 2, 3, 4, \\
x & \models & \exists q \mathcal{U} r & \text{holds for } x = 2, 3, 4,
\end{array}
$$

Recall that the transitions before the event modality $\langle b\rangle$ must be τ transitions and the same formula ψ must hold in all states from x to the state right before the event. For $x = 2$, this means that the third statement holds, because the formula $q \vee r$ holds both in state 2, 3 and 4, and the transitions have the event label τ. The fourth statement holds also in state 3 and 4, because it is enough that the second formula r is true in the current state. □

$\mathrm{ECTL}^*_{\setminus\bigcirc}$ *formulas are preserved by DBB reduction* Temporal logic properties specified by $\mathrm{ECTL}^*_{\setminus\bigcirc}$ formulas are preserved when a transition system is abstracted by DBB. This was proven in Lennartson and Noori-Hosseini (2018), and will now be applied to verification of Petri nets.

Since our incremental abstraction and verification approach is based on transition systems, each place in the Petri nets that are introduced in the next subsection will be modeled as a bounded buffer. The result is a modular transition system that can be verified in a flexible and generic way. This approach avoids the state space explosion that normally occurs with a standard global reachability graph transformation of a PN to a transition system.

7.3 Petri nets transformed to modular transition systems

DBB abstraction is obviously a state reduction method for transition systems, illustrated in Examples 9 and 10, which also preserves most temporal logic properties. Reduction of Petri nets (PNs) is a well known technique to reduce the complexity of large models. In Murata (1989), six reduction transformations are presented that preserve liveness, safeness, and boundedness.

We will now show how DBB can be used as reduction method for bounded PNs. This can be done algorithmically by fixed-point calculations, cf. Proposition 4 and Bunte et al. (2019); Lennartson et al. (2020), but also analytically for some special cases similar to the reduction rules in Murata (1989). The added features compared to the established reduction rules are that temporal logic properties are now guaranteed to be preserved, and the reduction can be computed by efficient algorithms with polynomial complexity (Bunte et al. 2019).

Abstraction of a single Petri net flow Consider a bounded PN with a straight sequence of places defined as $\mathrm{N}(a_0, \ldots, a_n, m_1, \ldots, m_n)$ and shown in Fig. 14. This PN has event (transition)

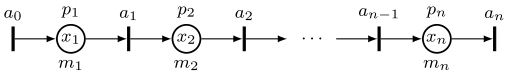


Fig. 14 A PN $\mathrm{N}(a_0, \ldots, a_n, m_1, \ldots, m_n)$ with event (transition) labels $a_0, \ldots, a_n$ and bounded number of tokens m_i in each place p_i

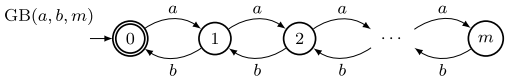


Fig. 15 Transition system generator $\mathrm{GB}(a, b, m)$ for a bounded buffer with marked initial state 0 and space for m elements

labels $a_0, \ldots, a_n$ and a bounded number of tokens m_i, also called capacity m_i, in each place p_i. The capacity m_i means that the number of tokens x_i is always limited to $0 \leq x_i \leq m_i$.

From Petri net to bounded buffers Since BB and DBB are based on finite-state transition systems, bounded PNs have to be transformed to transition systems to be able to apply these abstractions. However, instead of a traditional reachability analysis, a PN is here decomposed into modular PNs such that the incremental abstraction mentioned above can be applied. In its basic form, every bounded place with input and output transitions can be represented as a bounded buffer $\mathrm{GB}(a, b, m)$ shown in Fig. 15.

Definition 11 (Transition system for one place Petri net) Let $\mathrm{TS}(\cdot)$ be the operator that transforms a PN to a transition system, where also the marked state of the net is defined. For the one place PN $\mathrm{N}(a, b, m)$, which is a special case of the PN $\mathrm{N}(\cdot)$ in Fig. 14, the corresponding transition system is defined as

$$\mathrm{TS}(\mathrm{N}(a, b, m)) \stackrel{\mathrm{def}}{=} \mathrm{GB}(a, b, m),$$

where the bounded buffer generator $\mathrm{GB}(\cdot)$ is shown in Fig. 15. □

A reduced PN will be presented in the following proposition for the PN in Fig. 14 based on the DBB abstraction. The assumption is that only the first event a_0 and the last event a_n are shared with other subsystems. Thus, the events $a_1, \ldots, a_{n-1}$ are local and therefore replaced by τ. The result of this reduction is the same as one of the PN reduction rules in Murata (1989). The contribution of the following proposition is that all $\mathrm{ECTL}^*_{\setminus\bigcirc}$ formulas presented in Section 7.2 are preserved by using this well known reduction rule.

Proposition 5 (Abstraction of PN keeping first and last events) Applying the DBB abstraction, when all events except the first and last ones are hidden in the bounded PN in Fig. 14, results in a buffer transition system and a single place PN, that is

$$\mathrm{TS}\big(\mathrm{N}(a_0, \ldots, a_n, m_1, \ldots, m_n)\big)^{\mathcal{A}^{\{a_1, \ldots, a_{n-1}\}}} =$$
$$\mathrm{GB}(a_0, a_n, \sum_{i=1}^{n} m_i) = \mathrm{TS}\big(\mathrm{N}(a_0, a_n, \sum_{i=1}^{n} m_i)\big).$$

Proof The PN with n places in Fig. 14 can be considered as a synchronous composition of the one-place PNs $\mathrm{N}(a_{i-1}, a_i, m_i)$, $i \in \mathbb{N}_n^+ = \{1, \ldots, n\}$, i.e.

$$\mathrm{N}(a_0, \ldots, a_n, m_1, \ldots, m_n) = \|_{i \in \mathbb{N}_n^+} \mathrm{N}(a_{i-1}, a_i, m_i),$$

The corresponding transition system can according to Definition 11 be expressed as

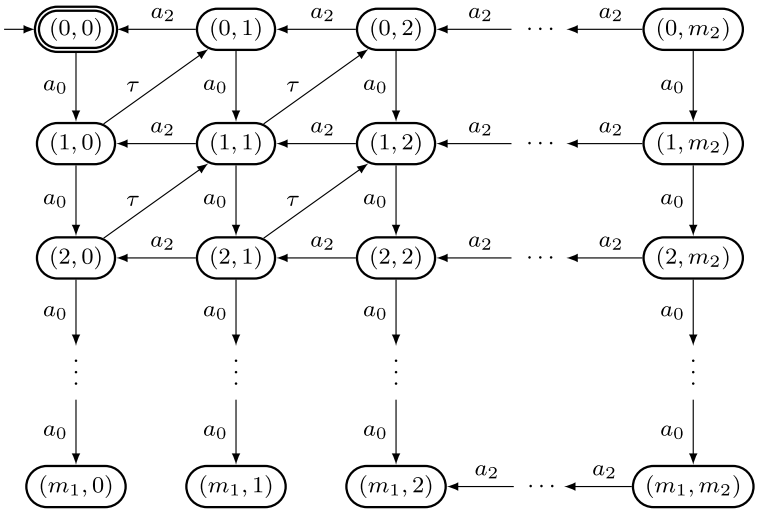
$$\mathrm{TS}\big(\|_{i\in\mathbb{N}_n^+}\mathrm{N}(a_{i-1},a_i,m_i)\big) = \|_{i\in\mathbb{N}_n^+}\mathrm{GB}(a_{i-1},a_i,m_i).$$

In the synchronous composition $\big(\mathrm{GB}(a_0,a_1,m_1)\,\|\,\mathrm{GB}(a_1,\,a_2,m_2)\big)^{\{a_1\}}$, shown in Fig. 16, the local event a_1 is replaced by τ, followed by a DBB abstraction in the same figure. The states in every sequence of diagonal τ transitions are then joined in block states, resulting in the partition $\Pi = \{B_0, B_1, B_2, \ldots, B_{m_1+m_2}\}$ where for instance $B_0 = \{(0,0)\}$, $B_1 = \{(1,0),(0,1)\}$, $B_2 = \{(2,0),(1,1),(0,2)\}$, and $B_{m_1+m_2} = \{(m_1,m_2)\}$. Thus,

$$\big(\mathrm{GB}(a_0,a_1,m_1)\,\|\,\mathrm{GB}(a_1,a_2,m_2)\big)^{\mathcal{A}^{\{a_1\}}} = \mathrm{GB}(a_0,a_2,m_1{+}m_2)$$

where the last equality is indeed an isomorphism on the state names, which also shows that this abstracted model is an extended buffer, now with m_1+m_2 elements. Note that each block transition $\Pi(x) \xrightarrow{a} \Pi(x')$ for $a = a_0$ and a_2 in the abstracted system has an extended transition $x \xrightarrow[\Pi]{\tau^*} x'' \xrightarrow{a} x'$ in the original system before the abstraction. According to Defini-

$\big(\mathrm{GB}(a_0,a_1,m_1)\,\|\,\mathrm{GB}(a_1,a_2,m_2)\big)^{\{a_1\}}$



$\big(\mathrm{GB}(a_0,a_1,m_1)\,\|\,\mathrm{GB}(a_1,a_2,m_2)\big)^{\mathcal{A}^{\{a_1\}}}$

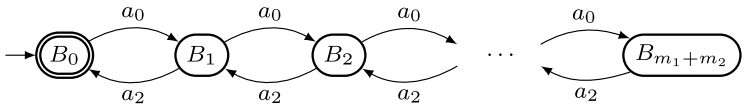


Fig. 16 The synchronous composition $\big(\mathrm{GB}(a_0,a_1,m_1)\,\|\,\mathrm{GB}(a_1,a_2,m_2)\big)^{\{a_1\}}$ and below its DBB abstraction $\big(\mathrm{GB}(a_0,a_1,m_1)\,\|\,\mathrm{GB}(a_1,a_2,m_2)\big)^{\mathcal{A}^{\{a_1\}}}$, where the block states B_i include the states in every sequence of diagonal τ transitions. For instance, the block state $B_2 = \{(2,0),(1,1),(0,2)\}$

tion 8 this means that Π is a BB equivalence. Repeating this synchronous composition n times generates the second expression in this theorem, since

$$\begin{aligned}&\mathrm{TS}\big(\mathrm{N}(a_0,\ldots,a_n,m_1,\ldots,m_n)\big)^{\mathcal{A}^{\{a_1,\ldots,a_{n-1}\}}} =\\ &\big(\|_{i\in\mathbb{N}_n^+}\mathrm{GB}(a_{i-1},a_i,m_i)\big)^{\mathcal{A}^{\{a_1,\ldots,a_{n-1}\}}} = \mathrm{GB}(a_0,a_n,\textstyle\sum_{i=1}^n m_i).\end{aligned}$$

and the last expression follows again from Definition 11. □

Complexity This theorem shows that a sequence of bounded PN places can be replaced by a one-place PN with a capacity equal to the sum of the capacities of the individual places in the original PN. The number of states in $\mathrm{TS}\big(\mathrm{N}(a_0,\ldots,a_n,m_1,\ldots,m_n)\big)$ is $\prod_{i=1}^n(1+m_i)$, while only $1+\sum_{i=1}^n m_i$ states is included in the abstraction $\mathrm{TS}\big(\mathrm{N}(a_0,\ldots,a_n,m_1,\ldots,m_n)\big)^{\mathcal{A}^{\{a_1,\ldots,a_{n-1}\}}}$. Based on this result, we find that if the capacity in each place is m, this results in a reduction from $(1+m)^n$ to $1+nm$ states in the abstracted PN in Proposition 5.

The results so far take into account the marked state labels, introduced in the local transition system models. In the next example we will investigate a fairness problem where these marked state labels are neglected. Thus, it is also of interest to evaluate the DBB abstraction when the marked state labels are removed.

Proposition 6 (DBB neglecting marked state labels) Applying DBB abstraction on the bounded PN in Fig. 14, neglecting the marked state label in the related buffer model $\mathrm{GB}(\cdot)$, results in the following one-state transition systems:

$$\begin{aligned}\mathrm{TS}\big(\mathrm{N}(a_0,\ldots,a_n,m_1,\ldots,m_n)\big)^{\mathcal{A}^{\{a_1,\ldots,a_n\}}} &= x \xrightarrow{a_0} x,\\ \mathrm{TS}\big(\mathrm{N}(a_0,\ldots,a_n,m_1,\ldots,m_n)\big)^{\mathcal{A}^{\{a_0,\ldots,a_{n-1}\}}} &= x \xrightarrow{a_n} x.\end{aligned}$$

Proof These one-state transition systems follow from Proposition 5, applying the DBB abstraction on $\mathrm{GB}(\cdot)$ without any state labels involved. □

Observe that in this extreme reduction, only the first or last event is preserved. Thus, it is only relevant for a first or last pure sequence of places in a PN. An example which illustrates this phenomenon is given in the following subsection, where the benefit of both computational and analytical PN reduction will be shown.

7.4 Fairness verification of a bounded Petri net

Consider the bounded Petri net model *PN*2 in Fig. 17, modeling two parallel PN sequences with three shared resources R_1, R_2, and R_3. Capacity one is selected for every second resource in the two sequences, modeling for instance machines in a production line, and capacity m is assumed for buffers between these resources. The shared resources R_2 and R_3 are booked in opposite order, which generates a classical deadlock that is avoided by add-

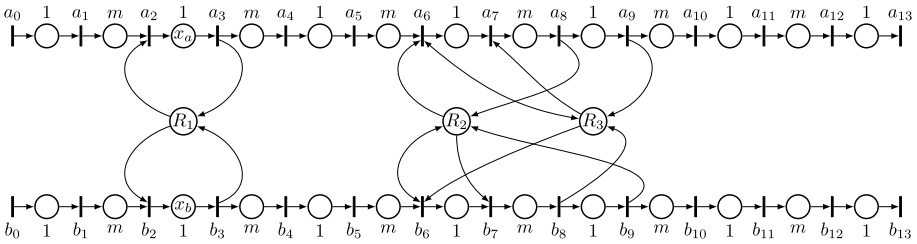


Fig. 17 Petri net *PN*2 with two sequences, each including 13 bounded places and three shared resources R_1, R_2, and R_3

ing an extra condition on transition a_6, only allowed to occur when R_3 is free, and b_6 only allowed when R_2 is free. Capacity one on these resources implies one token in corresponding resource place when the resource is free.

This Petri net is a typical example of a concurrent system with multiple paths of parts, one token for each part, together with a number of shared resources. A common industrial application area is flexible production systems, where multiple products are manipulated by machines and robots, representing shared resources.

Since incremental DBB abstraction is performed on modular transition systems, the Petri net *PN*2 is translated to such a model. According to Definition 11, each place is translated to a bounded buffer with the same capacity. The restrictions on the shared resources are easily translated to modular transition systems which are synchronized with the bounded buffer models, see further details in Lennartson (2021). The modular transition system is called $\mathrm{TS}(PN2)$.

Fairness condition A fairness condition is evaluated on $\mathrm{TS}(PN2)$, namely that resource R_1 is used infinitely often by both the first and the second PN sequence. This is done by including the state labels $\{x_a\}$ and $\{x_b\}$ in the states where R_1 is used by the first and second sequence, respectively. The fairness is expressed by the CTL* formula

$$\forall\Box\Diamond x_a \wedge \forall\Box\Diamond x_b.$$

This formula does not hold for $\mathrm{TS}(PN2)$, since R_1 may serve only one of the PN sequences. On the other hand, including the transition system $\mathrm{GB}(a_2, b_2, 1)$ implies that the events a_2 and b_2 must be alternated. Thus, replacing $\mathrm{TS}(PN2)$ with

$$\mathrm{TS}(PN2F) = \mathrm{TS}(PN2) \,\|\, \mathrm{GB}(a_2, b_2, 1)$$

means that the fairness condition is satisfied.

Incremental DBB abstraction The modular transition system $\mathrm{TS}(PN2F)$ is now evaluated based on the incremental DBB algorithm presented in Section 7.1 and Proposition 4. Applying this abstraction on $\mathrm{TS}(PN2F)$, where all events are finally hidden, results in a loop including four states, see Fig. 18. The state labels $\{x_a\}$ and $\{x_b\}$, added in the state where R_1 is serving corresponding sequence, means that $\{x_a\}$ holds in the second and $\{x_b\}$ in the fourth state. Due to the loop, both state labels are passed infinitely often, resulting in a fair system. Without any analytical PN reductions, execution times for the incremental

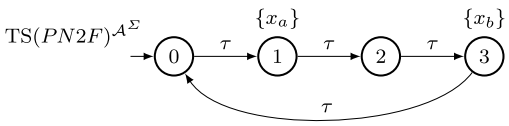


Fig. 18 DBB abstraction $\mathrm{TS}(PN2F)^{\mathcal{A}^\Sigma}$ where all events in the event set Σ are hidden. This abstracted system satisfies the fairness condition $\forall\Box\Diamond x_a \wedge \forall\Box\Diamond x_b$, meaning that both the state with label $\{x_a\}$ and the state with label $\{x_b\}$ are passed infinitely often

DBB abstraction are given in Table 1 for m = 10, 20, and 40. The computations have been performed on a MacBook Pro M4 with 24 GB RAM. The number of states in $\mathrm{TS}(PN2F)$ without abstraction is $7 \cdot (2m+2)^{10}$, which for $m = 40$ results in $9.6 \cdot 10^{19}$ states.

nuXmv model checking Since DBB preserves $\mathrm{ECTL}^*_{\setminus\bigcirc}$ including the actual fairness condition, this abstraction is also compared with the symbolic model-checker nuXmv (Cavada et al. 2014), which provides a number of efficient CTL and LTL verification algorithms. The default CTL and LTL algorithms are based on binary decision diagrams (BDDs) (Bryant 1992). For verification of LTL, a number of alternative more recent SAT-based algorithms are also available in nuXmv. In Lennartson et al. (2020), the most efficient and robust alternative to the default BDD algorithm was found to be the *k*-liveness algorithm (Claessen and Sorensson 2012). Somewhat surprisingly, neither this algorithm nor the BDD is able to evaluate the fairness condition, in LTL formulated as $\Box\Diamond x_a \wedge \Box\Diamond x_b$, on the modular transition system $\mathrm{TS}(PN2F)$ for $m > 1$.

Analytical DBB abstraction An analytical DBB (ADBB) abstraction of the local PN sequences in the two main sequences in *PN*2 can also be performed. Applying Proposition 6 on the subsystems $\mathrm{N}(a_0, a_1, a_2, 1, m)$, $\mathrm{N}(b_0, b_1, b_2, 1, m)$, $\mathrm{N}(a_9, a_{10}, a_{11}, a_{12}, a_{13}, m, 1, m, 1)$, and $\mathrm{N}(b_9, b_{10}, b_{11}, b_{12}, b_{13}, m, 1, m, 1)$ means that these subsystems can be neglected, since the DBB abstraction results in one state models. Furthermore, using Proposition 5 on $\mathrm{N}(a_3, a_4, a_5, a_6, m, 1, m)$ and $\mathrm{N}(b_3, b_4, b_5, b_6, m,\ 1,\ m)$ implies that they can be replaced by $\mathrm{N}(a_3, a_6, 2m+1)$ and $\mathrm{N}(b_3, b_6, 2m+1)$. This reduction is obtained without any computations, just utilizing Proposition 5 and Proposition 6. The reduced PN model can be evaluated by the BDD solver in nuXmv, see ADBB+BDD in Table 1. Still, the incremental DBB solver is more than sixty times faster, even without any support from the analytical abstraction.

7.5 Summary of the incremental verification of bounded Petri nets

An efficient method for temporal logic verification of any bounded Petri net has been presented in this section. The key elements are:

Table 1 Execution times (sec) for DBB and ADBB as preprocessing + BDD, evaluating the fairness condition $\forall\Box\Diamond x_a \wedge \forall\Box\Diamond x_b$ on the modular transition system $\mathrm{TS}(PN2F)$

m	DBB	ADBB+BDD
10	0.021	1.63
20	0.15	13.6
40	1.81	121

- *Analytical abstraction* of individual PN sequences, as described above in the paragraph "Analytical DBB abstraction".
- Generic transformation of a bounded PN to a modular transition system by replacing each remaining bounded place in the PN after the analytical abstraction with a *buffer transition model*, according to Fig. 15. All such buffer models for say n places generate a total modular transition system model $\mathrm{GB}_1 \parallel \mathrm{GB}_2 \parallel \cdots \parallel \mathrm{GB}_n$.
- Algorithmic *incremental abstraction* of the resulting modular transition system including one transition model for each bounded place. Each step in the incremental abstraction includes hiding of local events in each local transition system, followed by DBB abstraction and synchronization. This procedure is repeated since more local events are generally obtained after each synchronization.
- When all subsystems have been synchronized, only hidden τ events remain together with any state labels that where introduced in the original PN model. Desired $\mathrm{ECTL}^*_{\setminus\bigcirc}$ *temporal logic formula can be evaluated on the final abstracted transition model*, observing that any $\mathrm{ECTL}^*_{\setminus\bigcirc}$ temporal logic property in the original PN is preserved in the final abstracted model. Figure 18 is an example of such an abstracted model where the fairness property is preserved from the original model $\mathrm{TS}(PN2F) = \mathrm{TS}(PN2) \parallel \mathrm{GB}(a_2, b_2, 1)$.

The proposed method is shown to be able to solve a complex fairness problem that is not possible to solve by existing model checking tools. A deeper evaluation of this PN strategy, involving more details on the generation of the modular transition system, but also a broader evaluation including nonblocking verification formulated in CTL, can be found in Lennartson (2021). An added contribution here, not included in the earlier publication, is the proof of Proposition 5.

8 Conclusions

A unified formulation of strong, weak, stutter, and branching bisimulation has been presented in this paper. An alternative block transition based formulation that is more natural from a model reduction and computational perspective is also shown to be equivalent to the original relation based bisimulation formulations. Including divergence sensitivity in the branching simulation, it is also shown how temporal logic verification can be performed for modular systems based on incremental abstraction. This strategy is adapted to verify temporal logic properties for Petri nets, where it is demonstrated how both analytical and algorithmic abstraction can be combined. By this strategy, problems have been verified that have not been possible to solve by powerful model checking tools.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10626-026-00434-z.

Acknowledgements Many thanks to Professor Rob van Glabbeek, the founder of BB, who kindly evaluated our proposed alternative bisimulation formulation and suggested to prove the equivalence between his original formulation and the block transition based formulation presented in Sections 3 and 5. The author is also very thankful to Professor Michel Reniers and the reviewers for their important comments and suggested improvements of the manuscript.

Author Contributions Paper written only by the first and only author.

Funding Open access funding provided by Chalmers University of Technology. No funding was received to assist with the preparation of this manuscript.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

Aceto L, Ingolfsdottir A, Srba J (2012) The algorithmics of bisimilarity. In: Advanced Topics in Bisimulation and Coinduction, Cambridge University Press, pp 100–172

Baier C, Katoen JP (2008) Principles of Model Checking. The MIT Press, Cambridge, MA

Basten T (1996) Branching bisimilarity is an equivalence indeed! Inf Process Lett 58(3):141–147

Blom S, Orzan S (2003) Distributed branching bisimulation reduction of state spaces. Electronic Notes Theoret Comput Sci 89:99–113

Browne MC, Clarke EM, Grumberg O (1988) Characterizing finite Kripke structures in propositional temporal logic. Theoret Comput Sci 59:115–131

Bryant RE (1992) Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Comput Surv 24(3):293–318

Bunte O, Groote JF, Keiren JJA, Laveaux M, Neele T, de Vink EP, Wesselink W, Wijs A, Willemse TAC (2019) The mCRL2 toolset for analysing concurrent systems improvements in expressivity and usability. In: Vojnar T, Zhang L (eds) Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Part II, Springer, Lecture Notes in Computer Science, vol 11428, pp 21–39

Cavada R, Cimatti A, Dorigatti M, Griggio A, Mariotti A, Micheli A, Mover S, Roveri M, Tonetta S (2014) The nuXmv symbolic model checker. Lecture Notes Comput Sci, Springer, 8559:334–342

Claessen K, Sörensson N (2012) A liveness checking algorithm that counts. In: Cabodi G, Singh S (eds) Formal Methods in Computer-Aided Design (FMCAD), IEEE, pp 52–59

Eén N, Sörensson N (2004) An extensible SAT-solver. In: Giunchiglia E, Tacchella A (eds) Lecture Notes in Computer Science, Springer, vol 2919, p 502–518

Flordal H, Malik R (2009) Compositional verification in supervisory control. SIAM J Control Optim 48:1914–1936

Gerth R, Kuiper R, Peled D, Penczek W (1999) A partial order approach to branching time logic model checking. J ACM 150:132–152

Gorrieri R, Versari C (2015) Introduction to Concurrency Theory. Texts in Theoretical Computer Science, Springer

Graf S, Steffen B (1991) Compositional minimization of finite state systems. In: Clarke E, Kurshan R (eds) Computer-Aided Verification (CAV 1990), Springer, Lecture Notes in Computer Science, vol 531, pp 186–196

Groote JF, Vaandrager FW (1990) An efficient algorithm for branching bisimulation and stuttering equivalence. In: The 40th International Colloquium on Automata, Languages and Programming, Riga, Latvia, vol 443, pp 626–638

Hoare CAR (1978) Communicating Sequential Processes, Series in Computer Science, vol 21. ACM

Keller R (1976) Formal verification of parallel programs. Comm of the ACM 19(7):561–572

Kripke S (1963) Semantical considerations on modal logic. Acta Philosophica Fennica 16:83–94

Lennartson B (2021) Incremental abstraction - An analytical and algorithmic perspective on Petri net reduction. In: Proc. 17th IEEE Conference on Automation Science and Engineering (CASE 2021), pp 557–562
Lennartson B, Noori-Hosseini M (2018) Visible bisimulation equivalence— a unified abstraction for temporal logic verification. In: Proc. 14th International Workshop on Discrete Event Systems, WODES'18
Lennartson B, Liang X, Noori-Hosseini M (2020) Efficient temporal logic verification by incremental abstraction. In: Proc. 16th IEEE Conference on Automation Science and Engineering (CASE 2020), pp 894–899
Milner R (1989) Communication and Concurrency. Series in Computer Science, Prentice-Hall
Murata T (1989) Petri nets: Properties, analysis and applications. Proc IEEE 77(4):541–580
Nicola RD, Vaandrager F (1995) Three logics for branching bisimulation. J ACM 42:458–487
Noori-Hosseini M, Lennartson B, Hadjicostis CN (2018) Compositional visible bisimulation abstraction applied to opacity verification. In: Proceedings of 14th International Workshop on Discrete Event Systems, WODES'18
Park D (1981) Concurrency and automata on infinite sequences. In: Deussen P (ed) Theoretical Computer Science, Springer, Lecture Notes in Computer Science, vol 104, pp 167–183
Sangiorgi D (2012) Introduction to Bisimulation and Coinduction. Cambridge University Press
Tai K, Koppol P (1993) Hierarchy-based incremental analysis of communication protocols. In: Proc. IEEE Int. Conf. on Network Protocols, pp 318–325
Tarjan RE (1972) Depth-first search and linear graph algorithms. SIAM J Comput 1(2):146–160
Tarski A (1955) A Lattice-Theoretical Fixpoint Theorem and Its Applications. Pac J Math 5(2):285–309
Trčka N (2007) Silent Steps in Transition Systems and Markov Chains. IPA Dissertation Series 2007–08, Technische Universiteit Eindhoven
Van Glabbeek RJ, Weijland WP (1996) Branching time and abstraction in bisimulation semantics. J ACM 43:555–600
Van Glabbeek RJ, Luttik B, Trčka N (2009) Branching bisimilarity with explicit divergence. Fund Inform 93:371–392

Bengt Lennartson received the Ph.D. degree from the Chalmers University of Technology, Gothenburg, Sweden, in 1986. Since 1999 he has been a Professor of the Chair of Automation at the Department of Electrical Engineering, Chalmers University of Technology, where he was the Dean of Education 2004-2007. He was the General Chair of the 11th IEEE Conference on Automation Science and Engineering, CASE 2015, and the 9th International Workshop on Discrete Event Systems, WODES'08, and he has been Associate Editor of Automatica and IEEE Transactions on Automation Science and Engineering. He is the (co)author of more than 300 peer reviewed international papers, and his research is currently focused on discrete-event systems, AI planning and learning, as well as sustainable production and robust feedback control. He is a Fellow of the IEEE for his contributions to hybrid and discrete event systems for automation and sustainable production.