CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Slicing and Reduction Techniques for Model Checking Petri Nets

Dissertation zur Erlangung des Grades eines
Doktors der Naturwissenschaften

vorgelegt von

Dipl.-Inform. Astrid Rakow

Disputation am 18. Juli 2011

Erstgutachter: Prof. Dr. E. Best
Zweitgutachter: Prof. Dr. E.-R. Olderog

Zusammenfassung

Model Checking ist ein Ansatz zur Validierung der Korrekheit eines Hard- oder Softwaresystems. Dazu wird das System durch ein formales Modell beschrieben und Systemeigenschaften werden meist in temporaler Logik spezifiziert. Ein Model Checker untersucht dann vollautomatisch, ob das Modell eine Eigenschaft erfüllt, indem er dessen Zustandsraum untersucht. Da jedoch die Anzahl der Zustände exponentiell mit der Größe des Systems wachsen kann –was als *Zustandsraumexplosion* bezeichnet wird– ist die Entwicklung und Anwendung von Methoden unumgänglich, die es ermöglichen beim Model Checking mit Systemen umzugehen, die einen großen Zustandsraum haben.

Ein etablierter Formalismus zur Beschreibung von asynchronen Systemen, die durch Nebenläufigkeit, Parallelität und Nichtdeterminismus gekennzeichnet sind, sind Petri-Netze. Der Petri-Netz-Formalismus bietet eine Fülle von Analysetechniken und eine intuitive graphische Darstellung.

In der vorliegenden Arbeit werden zwei Reduktionsansätze für Petri-Netze vorgestellt, *Petri-Netz Slicing* und *Cutvertex Reduktionen*. Beide Ansätze zielen darauf ab, der Zustandsraumexplosion beim Model Checken entgegenzuwirken. Dazu transformieren sie ein gegebenes Petri-Netz in ein kleineres Netz, so dass gleichzeitig die untersuchte Eigenschaft bewahrt wird. Da Petri-Netz-Reduktionen das Modell transformieren, können sie leicht mit anderen Methoden kombiniert werden.

Die Kernidee beider Reduktionsansätze ist, dass temporal-logische Eigenschaften sich meist auf nur wenige Stellen eines Petri-Netzes beziehen und daher häufig Teile eines Petri-Netzes identifiziert werden können, die die untersuchte Eigenschaft nicht oder nur unwesentlich beeinflussen. Wir

nennen die Menge der Petri-Netzstellen auf die sich eine temporal-logische Formel φ bezieht $scope(\varphi)$. Für ein gegebenes Netz Σ und eine temporal-logische Eigenschaft φ bestimmen beide Ansätze ein Netz Σ', das wenigstens $scope(\varphi)$ enthält, und vereinfachen das übrige Netz so, dass Σ' in Bezug auf φ äquivalent zu Σ ist. Wir zeigen, dass es genügt, eine schwache Form von Fairness anzunehmen, die wir *relative Fairness* nennnen, um Lebendigkeitseigenschaften zu erhalten.

Petri-Netz Slicing, ein durch Program Slicing [112] inspirierter Ansatz, bestimmt ein reduziertes Netz beginnend von $scope(\varphi)$, indem das Netz um relevante Transitionen und deren Eingabestellen iterativ erweitert wird. Wir formulieren zwei solcher Slicingalgorithmen, *$CTL^*_{-\mathsf{X}}$ Slicing* und *Safety Slicing*, und zeigen, dass die so reduzierten Netze Falsifikation von $\forall$CTL*-Eigenschaften erlauben. Wir zeigen weiterhin, dass $\mathrm{CTL}^*_{-\mathsf{X}}$ Slicing $\mathrm{CTL}^*_{-\mathsf{X}}$-Eigenschaften bewahrt, wenn relative Fairness für Σ angenommen wird. Das üblicherweise aggressivere Safety Slicing bewahrt stotter-invariante Sicherheitseigenschaften.

Cutvertex Reduktionen sind ein dekompositioneller Ansatz. Ein monolithisches Petri-Netz wird in einen Kernel, der $scope(\varphi)$ enthält, und Umgebungsnetze zerlegt. Die Umgebungsnetze werden durch eines von sechs vorgegebenen, sehr kleinen *Summarynetzen* ersetzt. Um das geeignete Summarynetz zu identifizieren, wird ein Umgebungsnetz isoliert vom Gesamtsystem durch Model Checking untersucht. Dieser Identifikationsschritt wird durch unsere strukturellen *Pre/Postset Optimierungen* beschleunigt. Wir führen außerdem Mikroreduktionen ein, die die kleinsten Umgebungen direkt, das heißt ohne Untersuchung durch einen Model Checker, ersetzen. Wir zeigen, dass unter relativer Fairness Cutvertex Reduktionen alle Eigenschaften erhält, die in $\mathrm{LTL}_{-\mathsf{X}}$ formulierbar sind.

Abstract

Model checking is a method to validate the correct functioning of a piece of hard- or software. Specifications are expressed in temporal logic. A model checking algorithm determines automatically whether or not the checked model satisfies a given specification by examining the model's state space. In their basic form model checking algorithms explore the state space exhaustively. As the number of states may grow exponentially in the size of the system—which constitutes the infamous *state space explosion problem*—the development and application of methods to deal with huge state spaces are crucial.

Petri nets are a well established formalism to specify asynchronous systems that involve concurrency, parallelism and nondeterminism. They offer an intuitive graphical notation along with an abundance of analysis techniques.

In this work we develop two *Petri net reduction* approaches to tackle the state space explosion problem for model checking. Petri net reductions are transformations of the Petri net that decrease its size. As a mean against the state space explosion problem for model checking they have to preserve temporal properties and reduce its state space. Petri net reductions can conveniently be daisy chained with other methods fighting state space explosion.

The key idea for both of our approaches is that often parts of the net can be identified not to influence the temporal logic property, which usually refers to a few places of a net only. In the following $scope(\varphi)$ denotes the set of places referred to by a temporal logic formula φ. For a given net Σ and temporal logic formula φ, both approaches determine a net Σ' that contains at least $scope(\varphi)$ and simplifies the remaining net such that Σ' is equivalent

with respect to φ. To preserve liveness properties, we show that it suffices to assume a form of weak fairness, which we call *relative fairness*.

Petri net slicing, an approach inspired by program slicing [112], builds a reduced net starting from $scope(\varphi)$ by iteratively including relevant transitions and their input places. We define two such algorithms, *$CTL^*_{-\mathsf{X}}$ slicing* and *safety slicing*, and formally prove that the slices of both can be used to falsify $\forall$CTL* properties. We show that CTL$^*_{-\mathsf{X}}$ slicing also preserves CTL$^*_{-\mathsf{X}}$ properties under relative fairness, whereas the usually more aggressive safety slicing preserves stutter-invariant safety properties.

Cutvertex reductions is a decompositional approach. A monolithic Petri net is decomposed into a kernel containing $scope(\varphi)$ and several environment nets that are replaced by one out of six fixed, small summary nets. To identify the appropriate summary, an environment is model checked in isolation. We developed *pre/postset optimisation* as a structural optimisation to accelerate this identification step and *micro reductions*, which are structural reductions, that even allow to replace very small environments without model checking. We prove that under relative fairness cutvertex reductions preserve all LTL$_{-\mathsf{X}}$ expressible properties.

Acknowledgements

Many people contributed to this work in various ways and I am glad about the support I received. First of all, I would like to thank my supervisor Professor Dr. Eike Best who encouraged and challenged me throughout my academic program, while at the same time gave me the freedom to self-determinedly organize my work. He and Dr. Hans Fleischhack guided me through the dissertation process, always willing to discuss the topics at hand.

I wish to thank my fellow colleagues at the TrustSoft Graduate School. Doing a PhD poses a lot of new challenges. Thank you guys, I could often profit from your experiences and sometimes it helped just to know that I am not the only one facing a problem. Being a member of the TrustSoft Graduate School also gave me the opportunity to learn much about a researcher's trade. I thank the organizers and supervisors of the graduate school for their ambition to let us benefit from their experiences and for the transparency of the organizational business.

The DFG scholarship made it possible to spend all my work power on my studies. Without this financial support it would not have been possible for me to master the challenge of being a mom and becoming a scientist. I am very grateful for this opportunity.

I thank Uschi and Karin for providing a loving day care for my daughter, which gave me the peace of mind to concentrate on my studies.

Of course, my family—my parents, siblings and Andre and Tabea—deserves many thanks for being there for me, enriching my private live in so many ways, while I was at times so caught up in work. I would like to express a special thanks to my brother (and temporary colleague) Jan for so many fruitful discussions, especially on scientific writing style.

Contents

Chapter 1

Introduction

Model checking is a method to validate the correct functioning of a piece of hard or software. Specifications are expressed in temporal logic. A model checking algorithm determines automatically whether or not the checked model satisfies a given specification by examining the model's state space. In its basic form model checking algorithms explore the state space exhaustively. As the number of states may grow exponentially in the size of the system—which constitutes the infamous *state space explosion problem*—the development and application of methods to deal with huge state spaces are crucial.

Petri nets are a prominent formalism to specify asynchronous systems that involve concurrency, parallelism and nondeterminism. They offer an intuitive graphical notation along with an abundance of analysis techniques and find applications in many different domains, e.g. flexible manufacturing systems, biochemical processes, workflows or asynchronous hardware. Petri nets come in several variants. Here we consider systems modelled as place/transition Petri nets, the basic formalism.

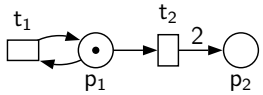


Figure 1.1: A small place/transition Petri net. For an introduction on P/T Petri nets see Sect. 2.2.

In this work we develop two *Petri net reduction* approaches to tackle the state space explosion problem for model checking. Petri net reductions are transformations of the Petri net that decrease its size. As a means against the state space explosion problem for model checking they have to preserve temporal properties and also decrease its state space. Then the reduced Petri net can be model checked for the considered property instead of the original. Certainly, other well established methods to alleviate the state space explosion exist like symbolic model checking, abstraction methods or on-the-fly model checking. The combination of different approaches promises an even more effective defence against state space explosion. Petri net reductions can conveniently be daisy chained with other methods fighting state space explosion.

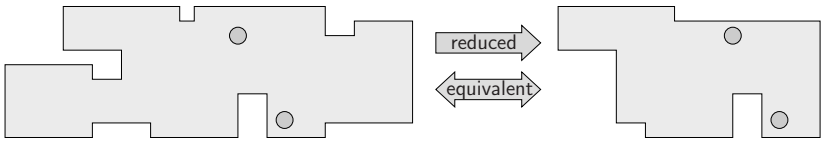


Figure 1.2: The principle of Petri net reductions for model checking: The reduced net preserves the temporal property φ and is not only a smaller Petri net but also has a smaller state space.

The key idea for both of our approaches is that often parts of the net can be identified not to influence the temporal logic property, which usually refers to a few places of a net only. For a given net Σ and temporal logic formula φ, both approaches determine a kernel net that contains at least $scope(\varphi)$ (the places referred to by φ) and simplifies the remaining net such that the reduced net Σ' is equivalent with respect to φ.

We examine which CTL* properties are preserved by our reductions. We face two general restrictions, as we examine systems in interleavings semantics and the reductions may eliminate concurrent behaviours: Firstly, properties using the next-time operator X are not preserved, since using X it is possible to count steps until a certain transition fires. But omitting concurrent behaviours influences the number of steps until a certain transition fires, as concurrent behaviours are interleaved in all possible ways. Secondly, when

the original system has a divergent subsystem, then there is an interleaving where only this subsystem evolves whereas other concurrent system parts do not progress. So eliminating the divergent subsystem will influence liveness properties. We hence assume a weak fairness notion, which we call *relative fairness*, to guarantee progress on the kernel and show that this suffices to preserve liveness properties.

Petri net slicing is a purely structural approach, i.e. inspecting the Petri net graph only, and is hence not influenced by the size of the system's state space. A reduced net is built by starting from $scope(\varphi)$ and iteratively including relevant transitions and their input places until reaching a fix point. We define two such algorithms, $CTL^*_{\text{-x}}$ *slicing* and *safety slicing*, and formally prove that the slices of both can be used to falsify $\forall$CTL* properties. We show that a net reduced by CTL$^*_{\text{-x}}$ slicing satisfies a given CTL$^*_{\text{-x}}$ property under relative fairness if and only if the original net does, whereas the usually more aggressive safety slicing preserves stutter-invariant safety properties.

Cutvertex reductions is a decompositional approach. A monolithic Petri net is decomposed into a kernel and several environments that share just a 1-safe place with the kernel. Each environment is replaced by one out of six fixed, very small summary nets, yielding a smaller state space. To identify the appropriate summary, an environment is model checked in isolation. Thus the combinatorial blow up is avoided. This step is optimised by two structural optimisation approaches. *Pre/postset optimisation* accelerates the identification of the appropriate summary and *micro reductions* even allow to replace the very small environments without model checking. We prove that under relative fairness cutvertex reductions preserve all LTL$_{\text{-x}}$ expressible properties.

An empirical evaluation of our reductions demonstrates their effectiveness also in combination with partial order reductions.

Thesis Structure

In Chapter 2 we recall basic notions like Petri nets, stutter-invariance, CTL* and considered sublogics. There we also introduce relative fairness, examine

its properties and compare it with the more commonly used notions of weak and strong fairness.

Chapter 3 gives a brief overview of approaches to tackle the state space explosion problem of model checking. We introduce Petri net reductions and compositional methods in more detail, since our two approaches, Petri net slicing and cutvertex reductions, classify as Petri net reductions and cutvertex reductions classify also as compositional method. Stubborn-set-type methods as partial order methods and agglomerations as prominent Petri net reductions are presented.

Chapter 4 presents the algorithms for $\mathrm{CTL}^*_{-\mathsf{X}}$ and safety slicing. It is proven that $\mathrm{CTL}^*_{-\mathsf{X}}$ slicing preserves $\mathrm{CTL}^*_{-\mathsf{X}}$ properties under relative fairness and allows for falsification of $\forall\mathrm{CTL}^*$, whereas safety slicing preserves stutter-invariant safety properties and can also be used to falsify $\forall\mathrm{CTL}^*$ properties.

Cutvertex reductions are developed in Chapter 5. It presents the six reduction rules that together allow to reduce any environment net. We examine which temporal properties are preserved by each reduction rule and give an algorithm that determines a decomposition into a kernel and environments that runs in linear time for a 1-safe net. Finally, we present micro-reductions and pre- and postset optimisations as structural optimisations for determining the appropriate summary net.

In Chapter 6 we demonstrate the effectiveness of our approaches on a benchmark set. We compare both our approaches to agglomerations and CFFD reductions and examine their effect on state spaces condensed by partial-order reductions.

We conclude in Chapter 7 with a summary of our results and outline ideas for future work.

Chapter 2

Preliminaries

Contents

In the following chapters we will present two approaches for reducing a Petri net Σ with the aim to alleviate the state space explosion problem for model checking temporal logics. Therefore the reduced net Σ' has to satisfy the same temporal properties as the original Σ, so that it can be used to *falsify*, that is to disprove, and to *verify*, that is to prove, that the temporal properties hold on Σ.

This chapter introduces basic notions (Sect. 2.1 to 2.6) as well as first results of technical nature that are used for both approaches (Sect. 2.5 to 2.6).

2.1 Sets and Sequences

For a set X we denote the union of finite and infinite words over X, $X^* \cup X^\omega$, as X^∞. For a finite sequence $\gamma = x_1 x_2 ... x_n \in X^\infty$, $|\gamma|$ is n, the length of γ. If γ is infinite, $|\gamma| = \infty$. $\gamma(i)$ denotes the i-th element, $1 \leq i < |\gamma| + 1$, and γ^i denotes the suffix of γ that truncates the first i positions of γ, $0 \leq i < |\gamma| + 1$. $\gamma' = proj_{X'}(\gamma)$ denotes the projection of γ to $X' \subseteq X$, i.e. γ' is derived from γ by omitting every $x_i \in X \setminus X'$. Two sequences γ_1 and γ_2 are stutter-equivalent iff $unstutter(\gamma_1) = unstutter(\gamma_2)$, where *unstutter* merges finitely many successive repetitions of the same sequence element into one. So $\gamma_1 = x_1 x_2 x_3$ and $\gamma_2 = x_1 x_2 x_2 x_2 x_3$ are stutter-equivalent whereas $\gamma_3 = x_1 x_2 x_3 x_3 ....$ is not stutter-equivalent to γ_1 or γ_2. We extend the functions *unstutter* and *proj* to sets of sequences in the usual way.

2.2 Petri Net Definitions

A *Petri net* N is a triple (P, T, W) where P and T are disjoint sets and $W : ((P \times T) \cup (T \times P)) \to \mathbb{N}$[1]. We consider here only *finite nets* that is P and T are finite sets. An element $p \in P$ is called a *place* and $t \in T$ a *transition*. The function W defines weighted *arcs* between places and transitions. A *marking* M of a Petri net N is a function $M : P \to \mathbb{N}$ that assigns a number of *tokens* to each place.

Petri nets are known for their intuitive graphical representation. A place is denoted as a circle and a transition as a box. A token is represented by a black dot within the place the token resides in. There is an arc from $p \in P$ to $t \in T$, if $W(p, t) > 0$, and, respectively, there is an arc from $t \in T$ to $p \in P$, if $W(t, p) > 0$. An arc weight greater one appears as number inscription next

[1] $\mathbb{N}$ is the set of natural numbers and includes zero.

to the respective arc. An example Petri net graph is shown in Fig. 2.1.

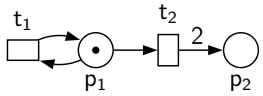


Figure 2.1: The Petri net graph of $N = (P, T, W)$ with $P = \{p_1, p_2\}$, $T = \{t_1, t_2\}$, $W = \{(p_1, t_1) \mapsto 1, (t_1, p_1) \mapsto 1, (p_1, t_2) \mapsto 1, (t_2, p_2) \mapsto 2, (t_1, p_2) \mapsto 0, (t_2, p_1) \mapsto 0, (p_2, t_1) \mapsto 0\}$ under marking $M = \{(p_1 \mapsto 1, p_2 \mapsto 0)\}$ is depicted.

With a given order on the places $P = \{p_1, ..., p_n\}$, a marking $M : P \to \mathbb{N}$ can be represented as a vector in $\mathbb{N}^{|P|}$, where the i-th component is $M(p_i)$. For convenience, we denote markings as row vectors as well as column vectors. As $M^{q=x}$ we denote the marking that places x tokens on q and $M(p)$ tokens on any other place p. $M|_{P'}$ is the restriction of M to places $P' \subseteq P$. We also denote the restriction of W to $((P' \times T') \cup (T' \times P'))$ as $W|_{(P',T')}$ for $P' \subseteq P$, $T' \subseteq T$.

The *preset* of $t \in T$ is ${}^\bullet t = \{p \in P \mid W(p,t) > 0\}$, its *postset* is $t^\bullet = \{p \in P \mid W(t,p) > 0\}$. Analogously ${}^\bullet p$ and $p^\bullet$ are defined. A transition $t \in T$ is *enabled* at marking M, $M[t\rangle$, iff $\forall p \in {}^\bullet t : M(p) \geq W(p,t)$. If t is enabled it can *fire*. The firing of t generates a new marking M', $M[t\rangle M'$, which is determined by the *firing rule* as $M'(p) = M(p) + W(t,p) - W(p,t), \forall p \in P$.

In Fig. 2.1 both transitions t_1 and t_2 are enabled at marking M. Firing t_1 generates marking M and firing t_2 generates marking $(0\ \ 2)$.

The definition of $[\rangle$ is extended to transition sequences σ as follows. A marking M always enables the empty firing sequence ε and its firing generates M. M enables a transition sequence σt, $M[\sigma t\rangle$, iff $M[\sigma\rangle M'$ and $M'[t\rangle$. If $M[\sigma\rangle$, the transition sequence σ is called a *firing sequence* of N from M. $\mathsf{Fs}_N(M)$ denotes the set of firing sequences from M on N. The effect of σ on a place $p \in P$, $\Delta(\sigma, p) \in \mathbb{Z}$, is defined by $\Delta(\varepsilon, p) = 0$ and $\Delta(\sigma t, p) = \Delta(\sigma, p) + W(t,p) - W(p,t)$.

A marking M is *final* if M does not enable any transition of N. A marking M' is *reachable* from M if there is a firing sequence from M that generates M'. A firing sequence σ from M is *maximal* iff either σ is infinite or σ cannot be extended, i.e., $\neg M[\sigma t\rangle$, $\forall t \in T$. Given a firing sequence

$\sigma = t_1 t_2 ...$ with $M_0[t_1\rangle M_1[t_2\rangle M_2 ...$, the sequence $M_0 M_1 M_2 ...$ is called the *marking sequence* from M_0, $\mathcal{M}(M_0, \sigma)$. As $\mathcal{M}(M_0, \sigma)|_{\tilde{P}} := M_0|_{\tilde{P}} M_1|_{\tilde{P}} M_2|_{\tilde{P}} ...$ we denote the elementwise restriction of $\mathcal{M}(M_0, \sigma)$ to $\tilde{P} \subseteq P$. A marking sequence $\mathcal{M}(M, \sigma)$ is *maximal* iff it contains a final marking. By convention (c.f. Sect. 2.4, Def. 2.4.1), we regard a finite maximal marking sequence μ as equivalent to the infinite marking sequence μ' that repeats the final marking of μ infinitely often.

In Fig. 2.1 the firing sequences t_2, $t_1\,t_2$, $t_1\,t_1\,t_2$,... are all maximal firing sequences of N from M and generate the final marking (0 2)*. The infinite firing sequence $t_1\,t_1\,t_1$ ... is the only other maximal firing sequence of N from M.*

A Petri net $\Sigma = (N, M_{\mathsf{init}})$ with a designated initial marking M_{init} is called a *marked Petri net*. If a transition sequence σ is enabled at the initial marking M_{init}, σ is called a firing sequence of Σ. The *set of reachable markings* of Σ is denoted as $[M_{\mathsf{init}}\rangle$. A place p is *k-bounded* if any reachable marking has at most k tokens at p. Σ is k-bounded if all of its places are k-bounded. 1-boundedness is also referred to as *1-safeness*.

A Petri net $\tilde{\Sigma} = (\tilde{P}, \tilde{T}, \tilde{W}, \tilde{M}_{\mathsf{init}})$ is a *subnet* of $\Sigma = (P, T, W, M_{\mathsf{init}})$ with $\tilde{P} \subseteq P$, $\tilde{T} \subseteq T$, $\tilde{W} \subseteq W|_{(\tilde{P},\tilde{T})}$ and $M_{\mathsf{init}}|_{\tilde{P}} = \tilde{M}_{\mathsf{init}}$. A subnet $\tilde{\Sigma}$ is called *proper* if it is neither empty, $\tilde{P} \cup \tilde{T} \neq \emptyset$, nor equals Σ. A Petri net Σ is *strongly connected* iff from each place and each transition every other place and transition of the net is reachable by following the arcs defined by W.

Convention

- In the following we use N synonymous with its defining triple (P, T, W) and Σ synonymous with (N, M_{init}). Also subscripts carry over to components, e.g. $\Sigma_{\mathrm{e}} = (N_{\mathrm{e}}, M_{\mathsf{init},\mathrm{e}}) = (P_{\mathrm{e}}, T_{\mathrm{e}}, W_{\mathrm{e}}, M_{\mathsf{init},\mathrm{e}})$.

- A marking generated by firing $\sigma \in T^*$ from the initial marking M_{init} is denoted as M_σ.

2.3 Logics

In this section we introduce the temporal logics we consider. Although we are mainly interested in *CTL (computation tree logic)* and *LTL (linear temporal logic)* as they are very prominent in model checking, we also introduce the branching-time logic CTL^* and its *universal fragment* $\forall CTL^*$. This allows us to show stronger results for our approaches that follow as easily as the more restricted results for CTL and LTL.

We define the semantics based on transition systems. Next we introduce the notion of transition system and after that define the logics.

2.3.1 Transition Systems

A transition system is one standard model to describe a system. We use transition systems here as an intermediate: We define the semantics of temporal logics on transition systems and we define the transition system (representation) of a marked Petri net in order to define the semantics of temporal logics for Petri nets.

Definition 2.3.1 (Transition System) *A* transition system *TS with initial state is a tuple* $(S, Act, R, AP, L, s_{\mathsf{init}})$ *where*

- S *is the set of states,*
- Act *is a set of actions,*
- $R \subseteq S \times Act \times S$ *is the transition relation with* $\forall s \in S : \exists \alpha \in Act : \exists s' \in S : (s, \alpha, s') \in R$*,*
- AP *is a set of atomic propositions,*
- $L : S \to 2^{AP}$ *is a state labelling function.*
- s_{init} *is a designated initial state of TS*

Note In literature many different types of transition systems are considered. We use transition systems with action names Act and atomic propositions as state labels. This way we can conveniently bridge between transition systems and Petri nets, as we will see in Sect. 2.4.

Some authors consider transition systems with terminal states. Terminal states are states without successor states. Since problems arise when considering the next-time operator (cf. Def. 2.3.2) it is usually more convenient to have at least one successor state for every state.

By convention a transition system with terminal states is therefore modified by extending R by $\{(s, \tau, s) \mid \text{s is a terminal state}\}$, where τ is a new action, $\tau \notin Act$. Then a terminal state s reaches via τ itself again.

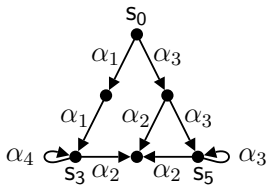
We also use the notion *state space of* Σ to refer to a transition system representation of a system Σ.

Notation We denote the number of states and state transitions of the state space of Σ as $|TS_\Sigma| = |S_\Sigma| + |R_\Sigma|$.

Paths, Fair Paths A finite *path* π from s to s_n is a finite sequence of states $\pi = s_0 s_1 s_2 ... s_n$ such that $s_0 = s$ and $\forall i,\, 0 \leq i < n : \exists \alpha_i \in Act : (s_i, \alpha_i, s_{i+1}) \in R$. An infinite path from s is an infinite sequence of states $\pi = s_0 s_1 s_2 ...$ such that $s_0 = s$ and $\forall i,\, 0 \leq i : \exists \alpha_i \in Act : (s_i, \alpha_i, s_{i+1}) \in R$.

An infinite path π is called *relatively fair* w.r.t. a *fairness constraint* $F \subseteq Act$ iff in case an action $\alpha \in F$ is from some point onward executable in every state of π, then some action $\tilde{\alpha} \in F$ occurs infinitely often along π —α may or may not equals $\tilde{\alpha}$. Formally, an infinite path $\pi = s_0 s_1 ...$ is called *relatively fair* w.r.t. a *fairness constraint* $F \subseteq Act$ iff in case there is an action $\alpha \in F$ such that $\exists i \in \mathbb{N} : \forall j,\, j \geq i : \exists \tilde{s} \in S : (s_j, \alpha, \tilde{s}) \in R$, then there is an action $\tilde{\alpha} \in F$ with $(s_i, \tilde{\alpha}, s_{i+1}) \in R$ for infinitely many s_i.

In the figure below the infinite path taking $\alpha_1 \alpha_1 \alpha_4 \alpha_4 ...$ from s_0 is relatively fair with respect to $\{\alpha_2, \alpha_4\}$. Both actions α_2 and α_4 can be executed in s_3 but it suffices that α_4 is taken infinitely often. The path taking $\alpha_3 \alpha_3 ...$ from s_0 is not relatively fair with respect to $\{\alpha_2, \alpha_4\}$ because α_2 is executable in s_5 but neither α_2 nor α_4 are taken infinitely often.



We simply call a path *relatively fair* if the fairness constraint F is known from the context.

Since we study state-based logics, we introduce another important notion: *traces*. A trace abstracts from a path by observing only the labels of states visited along the path. A *trace* ϑ of a finite path $\pi = s_0 s_1 ... s_n$ is $L(\pi) := L(s_0)L(s_1)...L(s_n)$. A trace ϑ of an infinite path $\pi = s_0 s_1 ...$ is $L(\pi) := L(s_0)L(s_1)...$.

Notation $\Pi_{TS}(s)$ denotes the set of all paths of TS from s. $\Pi_{TS,\mathsf{fin}}(s)$ denotes the set of all finite paths of TS from s and $\Pi_{TS,\mathsf{inf}}(s)$ is the set of all infinite paths of TS from s. Given a set of fairness constraints $\mathsf{Fair} \subseteq 2^{Act}$, $\Pi_{TS,\mathsf{Fair}}(s)$ is the set of all (infinite) paths of TS from s that are relatively fair w.r.t. every $F \in \mathsf{Fair}$. $\mathsf{Traces}_{TS}(s)$ denotes the set traces of (TS, s), that is $\mathsf{Traces}_{TS}(s) := \bigcup_{\pi \in \Pi_{TS}(s)} L(\pi)$. Analogously, $\mathsf{Traces}_{TS,\mathsf{fin}}(s)$ denotes the set of finite traces, $\mathsf{Traces}_{TS,\mathsf{inf}}(s)$ the set of infinite traces and $\mathsf{Traces}_{TS,\mathsf{Fair}}(s)$ the set of traces, generated by paths that are fair w.r.t. Fair.

Convention In the following we use TS and $(S, Act, R, AP, L, s_{\mathsf{init}})$ synonymously.

2.3.2 The Logics

In this section we define syntax and semantics of the temporal logics CTL*, $\forall$CTL*, LTL and CTL.

Definition 2.3.2 (CTL*, $\forall$CTL*, LTL, CTL) *Let TS be a transition system.*

A CTL* *formula is a state formula of the following syntax:*

Every atomic proposition $p \in AP$ is a state formula.

If φ_1 and φ_2 are state formulas, then $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$ are state formulas.

If ψ is a path formula, $\mathsf{E}\psi$ is a state formula.

If φ is a state formula, $\mathsf{D}\varphi$ is a path formula.

If ψ_1 and ψ_2 are path formulas, so are $\neg\psi_1$, $\mathsf{X}\psi_1$, $\psi_1 \vee \psi_2$ and $\psi_1 \mathsf{U} \psi_2$.

CTL* Semantics: Let $\mathsf{Fair} \subseteq 2^{Act}$ be a set of fairness constraints, s a state of TS and $\pi \in S^\omega$ an infinite state sequence.

$$TS, s \models_{\mathsf{Fair}} p \qquad \Leftrightarrow \quad p \in L(s).$$

$$
\begin{array}{lcl}
TS, s \models_{\mathsf{Fair}} \neg\varphi_1 & \Leftrightarrow & \textit{not } TS, s \models_{\mathsf{Fair}} \varphi_1. \\
TS, s \models_{\mathsf{Fair}} \varphi_1 \vee \varphi_2 & \Leftrightarrow & TS, s \models_{\mathsf{Fair}} \varphi_1 \textit{ or } TS, s \models_{\mathsf{Fair}} \varphi_2. \\
TS, s \models_{\mathsf{Fair}} \mathsf{E}\psi_1 & \Leftrightarrow & \textit{there is a path } \pi \textit{ from } s \\
 & & \textit{that is fair w.r.t. } \mathsf{Fair} \textit{ and } TS, \pi \models_{\mathsf{Fair}} \psi_1. \\
TS, \pi \models_{\mathsf{Fair}} \mathsf{D}\varphi_1 & \Leftrightarrow & TS, \pi(1) \models_{\mathsf{Fair}} \varphi_1 \\
TS, \pi \models_{\mathsf{Fair}} \neg\psi_1 & \Leftrightarrow & \textit{not } TS, \pi \models_{\mathsf{Fair}} \psi_1 \\
TS, \pi \models_{\mathsf{Fair}} \psi_1 \vee \psi_2 & \Leftrightarrow & TS, \pi \models_{\mathsf{Fair}} \psi_1 \textit{ or } TS, \pi \models_{\mathsf{Fair}} \psi_2 \\
TS, \pi \models_{\mathsf{Fair}} \mathsf{X}\psi_1 & \Leftrightarrow & TS, \pi^1 \models_{\mathsf{Fair}} \psi_1 \\
TS, \pi \models_{\mathsf{Fair}} \psi_1 \mathsf{U} \psi_2 & \Leftrightarrow & \exists i,\, 0 \leq i : TS, \pi^i \models_{\mathsf{Fair}} \psi_2 \wedge \forall j,\, 0 \leq j < i : \\
 & & TS, \pi^j \models_{\mathsf{Fair}} \psi_1
\end{array}
$$

We use the following abbreviations:

$\mathit{true} \equiv p \vee \neg p$, $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $F\varphi \equiv \mathit{true}\,\mathsf{U}\varphi$, $\mathsf{A}\varphi \equiv \neg\mathsf{E}(\neg\varphi)$, $\mathsf{G}\varphi \equiv \neg F(\neg\varphi)$ *and* $\varphi\mathsf{R}\psi \equiv \neg(\neg\varphi\mathsf{U}\neg\psi)$.

An $\forall$CTL* *formula is a state formula of the following syntax:*

If $p \in AP$ *is an atomic proposition,* p *and* $\neg p$ *are state formulas.*

If φ_1 *and* φ_2 *are state formulas,* $\varphi_1 \wedge \varphi_2$ *and* $\varphi_1 \vee \varphi_2$ *are state formulas.*

If ψ *is a path formula, then* $\forall\psi$ *is an state formula.*

If φ *is an state formula,* $\mathsf{D}(\varphi)$ *is a path formula.*

If ψ_1 *and* ψ_2 *are paths formulas, so are* $\mathsf{X}\psi_1$, $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $\psi_1\mathsf{U}\psi_2$, $\psi_1\mathsf{R}\psi_2$.

An LTL *formula is a path formula of the following syntax:*

If $p \in AP$, *then* $\mathsf{D}p$ *is a path formula.*

If ψ_1 *and* ψ_2 *are path formulas, then* $\neg\psi_1$, $\psi_1 \wedge \psi_2$, $\mathsf{X}\psi_1$, *and* $\psi_1\mathsf{U}\psi_2$ *are path formulas.*

A CTL *formula is a state formula of the following syntax:*

If $p \in AP$, *then* p *is a state formula.*

If φ_1 *and* φ_2 *are state formulas, then* $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$ *are state formulas.*

If ψ *is a path formula,* $\mathsf{E}\psi$ *is a state formula.*

If φ_1 *and* φ_2 *are state formulas, then* $\mathsf{X}(\mathsf{D}\varphi_1)$, $\mathsf{G}(\mathsf{D}\varphi_1)$, *and* $(\mathsf{D}\varphi_1)\mathsf{U}(\mathsf{D}\varphi_2)$ *are path formulas.*

The semantics of a $\forall$*CTL** *and CTL formula* φ *is defined by the semantics of CTL**, *and* $TS, s \models_{\mathsf{Fair}} \varphi$ *for an LTL formula* φ *is defined as* $TS, s \models_{\mathsf{Fair}} \mathsf{A}\varphi$.

A $\mathrm{CTL}^*_{\text{-}\mathsf{X}}$ *(*$\forall\mathrm{CTL}^*_{\text{-}\mathsf{X}}$*/*$\mathrm{LTL}_{\text{-}\mathsf{X}}$*/*$\mathrm{CTL}_{\text{-}\mathsf{X}}$*) formula is a CTL* (*$\forall$*CTL*/LTL/CTL) formula built without using the* X *operator.*

We define the length of a formula ψ to be the number of operators in ψ expressed as $\neg, \wedge, \mathsf{X}, \mathsf{U}$ [6] and denote it by $|\psi|$.

Convention

- For brevity we omit the operator D in formulas. We introduced the D-operator to make proofs over the structure of a formula more intelligible.
- Instead of "$\models_{\mathsf{Fair}}$" we also write more explicitely "$\models$ relatively fair w.r.t. Fair".
- If the transition system TS is known from the context, we also write $s \models \varphi$ or $\pi \models \psi$ without referencing the transition system explicitely.
- LTL formulas are often denoted using special symbols, that is $\Box$ denotes G and $\Diamond$ denotes F. We refrain from using these extra symbols. Also, we will often denote an LTL (path) formula ψ by the corresponding $\forall$CTL* (state) formula $\mathsf{A}\psi$.

The semantics is parameterised by a set of fairness constraints Fair. Commonly, the semantics is defined considering the *maximal* paths within the transition system, i.e. paths that cannot be extended. In our setting any maximal path is infinite, since every state has at least one successor. The "standard" semantics is thus derived by using $\mathsf{Fair} = \{Act\}$.

As we have to make fairness assumptions to derive some of the main results, we already give a more general definition of the semantics here. The fairness assumption used is tailored to our needs. We will have a closer look on relative fairness in Sect. 2.5.

Definition 2.3.3 ($TS, s \models \varphi$) *Let TS be a transition system and s a state in S. Let* φ *be a CTL* formula.*

$TS, s \models \varphi$ *iff* $TS, s \models_{\mathsf{Fair}} \varphi$ *and* Fair *of Def. 2.3.2 is the set* $\{Act\}$.

Verification and Falsification Figure 2.2 illustrates the relationship between the logics introduced in Def. 2.3.2. So CTL and LTL can express different properties. All properties expressible in LTL are also expressible in $\forall$CTL* and CTL* includes all the others.

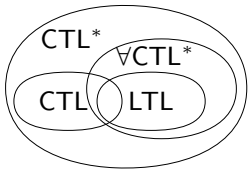


Figure 2.2: Relationship Between the Logics

If we examine what properties can be verified or falsified, it thus follows that if "$TS_1, s_1 \models_{\mathsf{Fair}} \varphi \Rightarrow TS_2, s_2 \models_{\mathsf{Fair}_2} \varphi$" holds for the formulas of a more expressible logic, it follows that "$TS_1, s_1 \models_{\mathsf{Fair}} \varphi \Rightarrow TS_2, s_2 \models_{\mathsf{Fair}_2} \varphi$" also holds for the formulas of a less expressible logic. So if we want to show "$TS_1, s_1 \models_{\mathsf{Fair}} \varphi \Rightarrow TS_2, s_2 \models_{\mathsf{Fair}_2} \varphi$" does not hold for formulas of a certain logic, it suffices to show that "$TS_1, s_1 \models_{\mathsf{Fair}} \varphi \Rightarrow TS_2, s_2 \models_{\mathsf{Fair}_2} \varphi$" does not hold for a less expressible logic.

If we want to show that neither "$TS_1, s_1 \models_{\mathsf{Fair}} \varphi \Rightarrow TS_2, s_2 \models_{\mathsf{Fair}_2} \varphi$" nor "$TS_1, s_1 \models_{\mathsf{Fair}} \varphi \Leftarrow TS_2, s_2 \models_{\mathsf{Fair}_2} \varphi$" holds, it is suffices to show that the implication in one direction does not hold, if we examine CTL and CTL*, because for CTL and CTL* formulas φ it holds that $TS, s \not\models_{\mathsf{Fair}} \varphi$ if and only if $TS, s \models_{\mathsf{Fair}} \neg\varphi$ (c.f. 2.3.2). This is not the case for LTL and $\forall$CTL*, because $\forall$CTL* allows negations only on atomic propositions and the semantics of LTL is defined by $TS, s \models_{\mathsf{Fair}} \mathsf{A}\psi$ and $\mathsf{A}\neg\psi$ is not equivalent to $\neg\mathsf{A}\psi$.

2.3.3 Stutter-invariant Safety Properties

In Sect. 4.3 we present a reduction approach preserving stutter-invariant safety properties only. In the following we introduce the notion of stutter-invariance and characterise safety properties following [6]. For the following we fix a set of atomic propositions AP.

To give safety properties a formal definition, we slightly shift our point of view. Whereas in Def. 2.3.2 we introduced state and path formulas syn-

tactically and straight-forwardly gave a satisfaction relation, $\models_{\mathsf{Fair}}$, for states and paths, we now take a step back and think more abstractly, instead of formulas, of *properties*, which may be expressed by a logical formula but are as such independent of the formalism expressing them. So we formally define *linear time properties*. Opposed to branching-time properties, that also consider the branching off of paths at the states of TS, linear-time properties express constraints on infinite paths or more precisely on infinite traces.

Definition 2.3.4 (LT Property) *A linear-time property (LT Property) over the set of atomic propositions AP is a subset of $(2^{AP})^\omega$.*

Although we introduced LTL as sublogic of the branching-time logic CTL*, any LTL formula is a path formula and hence specifies a constraint on traces, the sequences of labels along paths. So LTL is a logic that defines linear-time properties.

Since linear-time properties refer to traces only, their satisfaction relation can be expressed more simply than in Def. 2.3.2 but consistently as:

Definition 2.3.5 (Satisfaction Relation for LT Properties) *Let $\mathcal{P}$ be an LT property over AP and TS a transition system.*

$TS, s \models \mathcal{P} \Leftrightarrow \mathsf{Traces}_{TS,\mathsf{max}}(s) \subseteq \mathcal{P}$.

So if $TS, s \models \mathcal{P}$, then all traces starting from s satisfy $\mathcal{P}$.

Stutter-invariant linear-time properties do not distinguish between stutter-equivalent traces.

Definition 2.3.6 (Stutter-invariant [65, 80]) *Let ϑ and ϑ_2 be in $(2^{AP})^\omega$.*

A property $\mathcal{P}_{stutter} \subseteq (2^{AP})^\omega$ is stutter-invariant if whenever ϑ and ϑ_2 are stutter-equivalent then either both ϑ and ϑ_2 satisfy $\mathcal{P}_{stutter}$ or both violate $\mathcal{P}_{stutter}$.

We are now ready to define safety properties. A safety property can be thought of as stating that nothing bad will eventually happen [66]. When a safety property $\mathcal{P}_{safe}$ is violated, a finite prefix already exposes the behaviour forbidden by $\mathcal{P}_{safe}$. Formally a safety property is an LT property that, if any possible infinite trace ϑ violates $\mathcal{P}_{safe}$, it has a bad finite prefix ϑ_{pref}, such that any other (possible) trace $\tilde{\vartheta}$ with prefix $\vartheta_{\mathsf{pref}}$ also violates $\mathcal{P}_{safe}$.

Definition 2.3.7 (safety property) *An LT property $\mathcal{P}_{safe}$ over AP is a safety property if for all words $\vartheta \in (2^{AP})^\omega \setminus \mathcal{P}_{safe}$ there is a finite prefix ϑ_{pref} of ϑ such that $\mathcal{P}_{safe} \cap \{\tilde{\vartheta} \in (2^{AP})^\omega \mid \vartheta_{pref}$ is a prefix of $\tilde{\vartheta}\} = \emptyset$.*

Any such prefix ϑ_{pref} is called a bad prefix for $\mathcal{P}_{safe}$. The set of all bad prefixes for $\mathcal{P}_{safe}$ is denoted by $BadPref(\mathcal{P}_{safe})$.

This definition allows to derive a satisfaction relation referring to the finite behaviours of TS only. A transition system satisfies a safety property $\mathcal{P}_{safe}$ from state s iff the set of finite traces from s does not have a bad prefix, that means nothing bad happens starting from s.

Proposition 2.3.8 (Satisfaction Relation for Safety Properties) *For a transition system TS and safety property $\mathcal{P}_{safe}$ it holds that*

$TS, s \models \mathcal{P}_{safe}$ *if and only if* $\mathsf{Traces}_{TS,\mathsf{fin}}(s) \cap BadPref(\mathcal{P}_{safe}) = \emptyset$.

The fact that satisfiability of safety properties can be characterised by the finite behaviours of TS will allow us to define more effective reductions as we will demonstrate in Section 4.3.

2.4 Petri Net Semantics

In this section we define the transition system of a Petri net, in order to give the temporal logics a semantics on Petri nets. We also lift some of the previously introduced notions on transition systems to Petri nets. This allows us to shorten proofs by arguing about Petri nets directly.

In the following we assume that AP refers to the token count on a set of places $P' \subseteq P$. An atomic proposition ap may express that place p_5 has 2 tokens ($ap = (p_5, 2)$) or p_5 has no tokens ($ap = (p_5, 0)$). AP is hence a subset of $P' \times \mathbb{N}$. We also denote the set of places a temporal logic formula φ refers to as $scope(\varphi)$.

The behaviour of a marked Petri net Σ can be captured by a transition system TS_Σ. The reachable markings of Σ are the states. If $M[t\rangle M'$, then there is also a transition from state M to M' via action t in TS_Σ. Hence a

path $\mu = M_0M_1 ... M_n$ in TS_Σ corresponds to the firing sequence $\sigma = t_1t_2...t_n$ with marking sequence $\mathcal{M}(M_0, \sigma) = M_0M_1...M_n$.

A maximal firing sequence may be finite and thus generate a final marking that does not enable any transition, but every state M of TS_Σ has to have at least one successor. As discussed in the note on page 10, we introduce a new action symbol τ and define that a final marking M reaches itself via τ. By this extension any marking sequence corresponds to a path and thus any maximal firing sequence corresponds to an infinite path.

Definition 2.4.1 (TS_Σ) *TS_Σ is the tuple $(S_\Sigma, Act_\Sigma, R_\Sigma, AP_\Sigma, L_\Sigma, M_{\mathsf{init}})$ with*

- $S_\Sigma = [M_{\mathsf{init}}\rangle$,
- $Act_\Sigma = T \uplus \{\tau\}$,
- $R_\Sigma = \{(M, t, M') \mid M, M' \in [M_{\mathsf{init}}\rangle \wedge t \in T \wedge M[t\rangle M'\} \cup \{(M, \tau, M) \mid M \in [M_{\mathsf{init}}\rangle \wedge \forall t \in T : \neg M[t\rangle\}$
- $AP_\Sigma \subseteq P \times \mathbb{N}$
- $L_\Sigma = \{(M \mapsto A) \mid M \in S_\Sigma \wedge A = \{(p, x) \in AP_\Sigma \mid M(p) = x\}\}$.

We have already noted that (marking sequences of) maximal firing sequences of Σ and infinite paths of TS_Σ correspond. The following definitions introduce relatively fair firing sequences, the counterparts of relatively fair paths. Firstly, we define when a transition is *eventually permanently enabled* by a firing sequence. A firing sequence σ eventually permanently enables a transition t if from some point onward all markings generated during the execution of σ enable t. Note, that we did not introduce a corresponding notion on paths, since we mostly argue about the behaviour of a system on the Petri net model.

Definition 2.4.2 (Eventually Permanently Enabled) *Let $\sigma = t_1t_2...$ be an infinite firing sequence of Σ with $M_i[t_{i+1}\rangle M_{i+1}, \forall i, 0 \leq i < |\sigma|$.*

σ eventually permanently enables $t \in T$ iff $\exists i, 0 \leq i : \forall j, i \leq j : M_j[t\rangle$.

Definition 2.4.3 (Fairness with respect to F) *Let $F \subseteq T$ be a fairness constraint, let σ be a firing sequence of Σ and M be a marking of Σ.*

σ is relatively fair w.r.t. *F iff*

- *either σ is finite and maximal,*
- *or σ is infinite, and, if it eventually permanently enables some $t \in F$, it then fires infinitely often some transition of F (which may or may not be t itself).*

Let $\mathsf{Fair} \subseteq 2^T$ *be a set of fairness constraints.* σ *is* relatively fair w.r.t. Fair *iff* σ *is relatively fair w.r.t. every* $F \in \mathsf{Fair}$.

For infinite firing sequences it is obvious that this notion captures relative fairness as introduced for transition systems. For Petri nets we also consider finite, maximal firing sequences. Above, a finite, maximal firing sequence σ_{max} is defined as relatively fair w.r.t. to any $F \subseteq T$. σ_{max} generates a final marking M that by definition does not enable any transition in T and its marking sequence loops at M. Hence the corresponding path in TS_Σ is also fair w.r.t. F, since only τ can be executed at M.

Notation $\mathsf{Fs}_{N,\mathsf{Fair}}(M) := \{\sigma \mid M[\sigma\rangle$ and σ is fair w.r.t. $\mathsf{Fair}\}$ denotes the set of firing sequences from M, that are relatively fair w.r.t. every $F \in \mathsf{Fair} \subseteq 2^T$. We denote $\mathsf{Fs}_{N,\{\mathsf{T}\}}(M)$ also as $\mathsf{Fs}_{N,\mathsf{max}}(M)$.

Convention We say "Σ is relatively fair w.r.t. T'" to express that we only consider firing sequences of Σ that are relatively fair w.r.t. T'.

We now define $\Sigma \models \varphi$ via TS_Σ for the temporal logics defined in Def 2.3.2.

Definition 2.4.4 ($\Sigma \models_{\mathsf{Fair}} \varphi$) *Let φ be a $CTL^*_{-\mathsf{X}}$ formula such that the set of atomic propositions of φ is contained in AP_Σ and $\mathsf{Fair} \subseteq 2^T$ be a set of fairness constraints.*

$\Sigma \models_{\mathsf{Fair}} \varphi$, *iff* $(TS_\Sigma, M_{\mathsf{init}}) \models_{\mathsf{Fair}} \varphi$.

Convention If we interpret in the following a temporal property φ on a Petri net Σ, we always assume that the set of atomic propositions of φ is contained in AP_Σ, the set of atomic propositions of TS_Σ.

We denote fairness constraints on Petri nets analogously to fairness constraints on transition systems. If we refer to the behaviour of Σ that satisfies a fairness constraint Fair, we also write Σ_{Fair}.

2.5 Properties of Relative Fairness

We defined the notion of a relatively fair path of a transition system in Sect. 2.3 and the corresponding notion of relatively fair firing sequence of a Petri net in Sect. 2.4. We decided to use the non-standard notion of relative fairness, since it suffices to derive our results. As we will see in what follows, the more commonly considered notions like weak fairness and strong fairness are more restrictive.

In the sequel $F, F_1, F_2 \subseteq T$ denote fairness constraints and $\mathsf{Fair} \subseteq 2^T$ a set of fairness constraints. We also fix a Petri net N.

Relative Fairness, Weak Fairness, Strong Fairness We now compare our notion of relative fairness to the notions of weak and strong fairness. A firing sequence σ is *strongly fair* w.r.t. a set of transitions F iff whenever infinitely often transitions in F are enabled, then transitions in F occur in σ infinitely often. σ is *weakly fair* w.r.t. a set of transitions F iff whenever F is eventually permanently enabled (i.e. from some point onward permanently transitions in F are enabled), then transitions in F occur in σ infinitely often.

Definition 2.5.1 (Weak Fairness, Strong Fairness) *Let N be a Petri net, and M_0 a marking of Σ. Let $F \subseteq T$ be a set of transitions and $\mathsf{Fair} \subseteq 2^T$ be a set of fairness constraints. Let $\sigma = t_1\, t_2\, t_3 \ldots$ be an infinite firing sequence with $M_i[t_{i+1}\rangle M_{i+1}$, $\forall i \geq 0$.*

σ is strongly fair *w.r.t. F iff*

$$\begin{aligned}&\textit{whenever } \forall i \in \mathbb{N} : \exists j \in \mathbb{N}, j \geq i : \exists t \in F : M_j[t\rangle,\\ &\textit{then } \forall i \in \mathbb{N} : \exists j \in \mathbb{N}, j \geq i : \exists t' \in F : M_j[t_{j+1}\rangle M_{j+1} \wedge t_{j+1} = t'.\end{aligned}$$

σ is weakly fair *w.r.t. F iff*

$$\begin{aligned}&\textit{whenever } \exists i \in \mathbb{N} : \forall j \in \mathbb{N}, j \geq i : \exists t \in F : M_j[t\rangle,\\ &\textit{then } \forall i \in \mathbb{N} : \exists j \in \mathbb{N}, j \geq i : \exists t' \in F : M_j[t_{j+1}\rangle M_{j+1} \wedge t_{j+1} = t'.\end{aligned}$$

Any finite, maximal firing sequence σ is strongly and weakly fair w.r.t. F.

Notation $\mathsf{Fs}_{\mathsf{s(Fair)}}(M)$ denotes the set of firing sequences from M that are strongly fair w.r.t. every $F \in \mathsf{Fair}$ and $\mathsf{Fs}_{\mathsf{w(Fair)}}(M)$ denotes the set of firing sequences from M, that are weakly fair w.r.t. every $F \in \mathsf{Fair}$.

Strong fairness is more restrictive then weak fairness, i.e. every strongly fair firing sequence is weakly fair but a weakly fair firing sequence is not necessarily strongly fair. For place/transition nets (P/T nets)—the kind of Petri nets we consider—weak or strong fairness is usually assumed w.r.t. singletons. The more general form introduced here accords to the definition in [6].

For convenience, we repeat the definition of relative fairness (cf. Def. 2.4.3):

An infinite σ is *relatively fair* w.r.t. F iff

whenever $\exists t \in F : \exists i \in \mathbb{N} : \forall j \in \mathbb{N}, j \geq i : M_j[t\rangle$,
then $\forall i \in \mathbb{N} : \exists j \in \mathbb{N}, j \geq i : \exists t' \in F : M_j[t_{j+1}\rangle M_{j+1} \wedge t_{j+1} = t'$ [2].

If σ is finite and maximal, it is relatively fair w.r.t. F.

Let us now compare the different fairness notions for the same fairness constraint $F \subseteq T$: Obviously our notion is less restrictive than strong fairness, as relative fairness only rules out infinite firing sequences where a transition in F is *eventually permanently enabled* and F is fired *finitely* often only[3]. In contrast, strong fairness already rules out infinite firing sequences where transitions are *infinitely often enabled* and F is fired *finitely* often.

The difference between weak fairness and relative fairness is more subtle. Both notions refer to permanent enabledness. Loosely speaking, weak fairness rules out certain infinite firing sequences where *the set of transitions* F is eventually permanently enabled: From some point onward every marking enables a transition in F; consecutive markings do not necessarily enable the same transition. Our notion of relative fairness only rules out infinite firing sequences where at least *one transition of* F is eventually permanently enabled and F is only fired finitely often.

[2] As $F \subseteq T$ is finite, this is equivalent to $\exists t' \in F : \forall i \in \mathbb{N} : \exists j \in \mathbb{N}, j \geq i : M_j[t_{j+1}\rangle M_{j+1} \wedge t_{j+1} = t'$.

[3] More precisely: Transitions of F are fired a finite number of times.

Let us contrast the three notions by means of an example. Consider the net in Fig. 2.3 (a). The firing sequence $\sigma = t_1\, t_2\, t_1\, t_2 \ldots$ is not strongly fair w.r.t. $\{t_3\}$, because t_3 is enabled infinitely often and never fired. However, σ is weakly fair and relatively fair w.r.t. $\{t_3\}$, since t_3 is not eventually permanently enabled. σ is not weakly fair w.r.t. $\{t_3, t_4\}$, because permanently either t_3 or t_4 are enabled and neither t_3 nor t_4 are fired infinitely often. σ is relatively fair w.r.t. $\{t_3, t_4\}$, since neither t_3 nor t_4 are eventually permanently enabled.

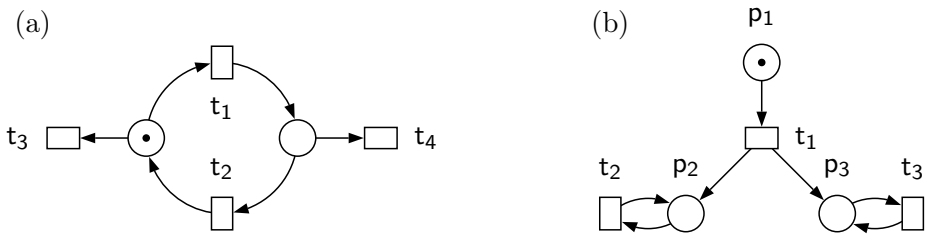


Figure 2.3: Two simple Petri nets.

The above example shows that relative fairness does not imply weak fairness. The following proposition summarises the relations of the three fairness notions. As discussed, strong fairness implies weak fairness, which implies relative fairness, but in general not vice versa. Strong, weak and relative fairness coincide for special fairness constraints.

Proposition 2.5.2 *Let N be a Petri net, M a marking of N and $\mathsf{Fair} \subseteq 2^T$ a set of fairness constraints.*

(i) $\mathsf{Fs}_{N,\mathsf{s(Fair)}}(M) \subset \mathsf{Fs}_{N,\mathsf{w(Fair)}}(M) \subseteq \mathsf{Fs}_{N,\mathsf{Fair}}(M)$ *for any N, M,* Fair*, but there are N, M,* Fair *such that* $\mathsf{Fs}_{N,w(\mathsf{Fair})}(M) \not\subseteq \mathsf{Fs}_{N,\mathsf{s(Fair)}}(M)$ *and there are $N, M,$* Fair *such that* $\mathsf{Fs}_{N,\mathsf{Fair}}(M) \not\subseteq \mathsf{Fs}_{N,\mathsf{w(Fair)}}(M)$*.*

(ii) For singleton fairness constraints, weak fairness equals relative fairness. $\mathsf{Fs}_{N,\{\{t\}\}}(M) = \mathsf{Fs}_{N,\mathsf{w}(\{\{t\}\})}(M)$ *for any $t \in T$.*

(iii) Relative fairness, weak fairness and strong fairness coincide for the fairness constraint T.
$\mathsf{Fs}_{N,\{T\}}(M) = \mathsf{Fs}_{N,\mathsf{w}(\{T\})}(M) = \mathsf{Fs}_{N,\mathsf{s}(\{T\})}(M) = \mathsf{Fs}_{N,\mathsf{max}}(M)$*.*

Proof For (i) we only show that $\mathsf{Fs}_{N,\mathsf{w(Fair)}}(M) \subseteq \mathsf{Fs}_{N,\mathsf{Fair}}(M)$. Let σ be a firing sequence that is not relatively fair w.r.t. $F \in \mathsf{Fair}$. So there is a transition $t \in F$ eventually permanently enabled but only finitely many transitions in F are fired. Since $t \in F$ is eventually permanently enabled, also F is eventually permanently enabled, and hence σ is not weakly fair w.r.t. F. Similarly it can be shown that strong fairness implies weak fairness. We have seen above examples showing that relative fairness does not imply weak fairness and weak fairness does not imply strong fairness. Straight-forwardly (ii) and (iii) follow from the fairness definitions. □

Basic Properties To get a better intuition for relative fairness, we briefly summarise its basic properties.

Proposition 2.5.3 *Let M be a marking of N. Let $F_1, F_2 \subseteq T$ be fairness constraints.*

(i) $\mathsf{Fs}_{N,\{F_1,F_2\}} \subseteq \mathsf{Fs}_{N,\{F_1\}}(M)$ *holds, but in general* $\mathsf{Fs}_{N,\{F_1\}}(M) \subseteq \mathsf{Fs}_{N,\{F_1,F_2\}}(M)$ *does not hold.*

(ii) Neither $\mathsf{Fs}_{N,\{F_1\cup F_2\}}(M) \subseteq \mathsf{Fs}_{N,\{F_1\}}(M)$ *nor* $\mathsf{Fs}_{N,\{F_1\}}(M) \subseteq \mathsf{Fs}_{N,\{F_1\cup F_2\}}(M)$ *hold in general.*

(iii) $\mathsf{Fs}_{N,\{F_1,F_2\}}(M) \subseteq \mathsf{Fs}_{N,\{F_1\cup F_2\}}(M)$ *holds, but in general* $\mathsf{Fs}_{N,\{F_1\cup F_2\}}(M) \subseteq \mathsf{Fs}_{N,\{F_1,F_2\}}(M)$ *does not hold.*

Proof (i) It follows directly from Def. 2.4.3 that $\mathsf{Fs}_{\{F_1,F_2\}} \subseteq \mathsf{Fs}_{\{F_1\}}$. Let us consider the Petri net of Fig. 2.3 (b) and the firing sequence $\sigma = t_1\, t_2\, t_2 \ldots$. σ is relatively fair w.r.t. $\{\{t_2\}\}$ but not relatively fair w.r.t. $\{\{t_2\},\{t_3\}\}$.

(ii) σ is also relatively fair w.r.t. $\{\{t_2,t_3\}\}$ but is not relatively fair w.r.t. $\{\{t_3\}\}$, and σ is relatively fair w.r.t. $\{\{t_1\}\}$ but not relatively fair w.r.t. $\{\{t_1,t_3\}\}$.

(iii) Let σ be fair w.r.t. F_1 and F_2. If there is a $t \in F_1$ (or F_2) eventually permanently enabled, then a transition $t' \in F_1$ (F_2) is fired infinitely often. Hence σ is fair w.r.t. $F_1 \cup F_2$. □

2.6 Fair Simulation and Stuttering Fair Bisimulation

In the course of this work we will introduce reduction rules, that allow us to derive from a given Petri net Σ a reduced net Σ'. We aim to preserve temporal logic properties, so that we can use the reduced instead of the original net when model checking. We will have to make fairness assumptions on the original net to derive some of our main results. Bisimulations and simulations will allow us to derive a couple of results:

- To show that we can use a reduced Petri net Σ' to falsify $\forall$CTL* properties on Σ_{Fair}, we show that the transition system of Σ, TS_Σ, fairly simulates $TS_{\Sigma'}$.
- To show that the fair Σ and a reduced Σ' satisfy the same CTL$^*_{\text{-x}}$ properties, we use stuttering fair bisimilarity.

As stuttering fair bisimulation is not a standard notion, we prove here that if two transition systems under their respective fairness constraints are *stuttering fair bisimilar*, then they fairly satisfy the same CTL$^*_{\text{-x}}$ formulas. But first we introduce fair simulation.

Definition 2.6.1 (Fair Simulation) *Let TS and TS_2 be transition systems with $AP = AP_2$. Let $s_{\mathsf{init}} \in S$ and $s_{\mathsf{init2}} \in S_2$ be their initial states.*

A relation $\mathcal{S} \subseteq S \times S_2$ is a fair simulation relation *between TS and TS_2 if and only if for all $s \in S$ and $s_2 \in S_2$ with $(s, s_2) \in \mathcal{S}$ holds:*

(L) $L(s) = L_2(s_2)$, *and*

(F) $\forall \pi \in \Pi_{TS,\mathsf{Fair}}(s) : \exists \pi_2 \in \Pi_{TS_2,\mathsf{Fair}_2}(s_2) : (\pi(i), \pi_2(i)) \in \mathcal{S}, \forall i \geq 1.$

TS_2 under fairness constraints Fair_2 simulates *TS under fairness constraints* Fair *if there is a fair simulation relation $\mathcal{S}$ between TS and TS_2 and $(s_{\mathsf{init}}, s_{\mathsf{init2}}) \in \mathcal{S}$.*

Condition (F) holds if for any fair path π of TS from s a fair path π_2 of TS_2 from s_2 exists such that when stepping though both paths simultaneously similar states are visited. If TS_2 fairly simulates TS, then whatever TS does

respecting fairness constraints Fair, TS_2 can do while respecting Fair_2. TS_2 can use its fair behaviour to mimic the fair behaviour of TS but TS_2 might also expose additional behaviour which might be fair or not. Consequently if $(TS_2, s_{\mathsf{init2}})_{\mathsf{Fair}_2}$ fairly simulates $(TS, s_{\mathsf{init}})_{\mathsf{Fair}}$, then $TS_2, s_{\mathsf{init2}} \models_{\mathsf{Fair}_2} \varphi$ implies $TS, s_{\mathsf{init}} \models_{\mathsf{Fair}} \varphi$ for any $\forall$CTL* formula φ with its atomic propositions in AP [22].

The definition of stuttering fair bisimulation is more involved, because stuttering does not require a one to one matching of states. Before we define stuttering fair bisimulation, we need to introduce the notions *partition* and *segment* [77], which will help us to express the more complicated matching of corresponding states.

A function $\theta : \mathbb{N} \to \mathbb{N}$ is called a *partition* if $\theta(0) = 1$ and θ is strictly increasing (i.e. $\theta(i) < \theta(i+1), \forall i \geq 0$).

We use a partition to divide an infinite state sequence ρ into *segments* of corresponding states. Segment i ranges from index $\theta(i)$ to $\theta(i+1) - 1$. The set of states in segment i on ρ is $seg_{\theta,\rho}(i) = \{\rho(\theta(i)), ..., \rho(\theta(i+1) - 1)\}$.

Definition 2.6.2 (Stuttering Fair Bisimilar) *Let TS and TS_2 be transition systems with $AP = AP_2$. Let $\mathsf{Fair} \subseteq 2^{Act}$ and $\mathsf{Fair}_2 \subseteq 2^{Act_2}$ be sets of fairness constraints. Let $s_{\mathsf{init}} \in S$ and $s_{\mathsf{init2}} \in S_2$ be initial states of TS and TS_2.*

A relation $\mathcal{B} \subseteq S \times S_2$ is a stuttering fair bisimulation relation *between TS under fairness constraints Fair and TS_2 under fairness constraints Fair_2 if and only if for all $s \in S$ and $s_2 \in S_2$ with $(s, s_2) \in \mathcal{B}$ holds:*

L $L(s) = L_2(s_2)$, *and*

SF1 $\forall \pi \in \Pi_{TS,\mathsf{Fair}}(s) : \exists \pi_2 \in \Pi_{TS_2,\mathsf{Fair}_2}(s_2) : \mathsf{match}(\mathcal{B}, \pi, \pi_2)$

SF2 $\forall \pi_2 \in \Pi_{TS_2,\mathsf{Fair}_2}(s_2) : \exists \pi \in \Pi_{TS,\mathsf{Fair}}(s) : \mathsf{match}(\mathcal{B}, \pi, \pi_2)$.

where match *for infinite state sequences π, π_2 and relation $\mathcal{R} \subseteq S \times S_2$ is defined as: $\mathsf{match}(\mathcal{R}, \pi, \pi_2)$ is true iff there are partitions θ and θ_2 such that $\forall i \geq 0 : \forall s \in seg_{\theta,\pi}(i) : \forall s_2 \in seg_{\theta_2,\pi_2}(i) : \mathcal{R}(s, s_2)$. Otherwise, $\mathsf{match}(\mathcal{R}, \pi, \pi_2)$ is false.*

TS_{Fair} and $(TS_2)_{\mathsf{Fair}_2}$ are stuttering fair bisimilar*, $TS_{\mathsf{Fair}} \cong (TS_2)_{\mathsf{Fair}_2}$, if such an $\mathcal{B}$ exists and also $(s_{\mathsf{init}}, s_{\mathsf{init2}}) \in \mathcal{B}$.*

The function $\mathsf{match}(\mathcal{R}, \pi, \pi_2)$ is true if the states along π and π_2 can be partitioned into infinitely many segments such that any state s in segment i on π and any state s_2 in segment i on π_2 are related, i.e. $\mathcal{R}(s, s_2)$. Figure 2.4 illustrates the matching of paths π and π'. Paths π and π_2 match (i.e. $\mathsf{match}(\mathcal{R}, \pi, \pi_2)$ is true), iff all states in correspoding segments match, with other words if any state in $seg_{\theta_2,\pi_2}(i)$ is in relation $\mathcal{R}$ with any state $seg_{\theta,\pi}(i)$ and vice versa.

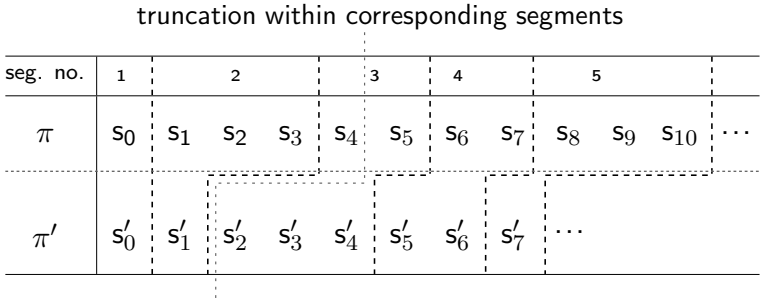


Figure 2.4: Matching for bisimulation: Corresponding segments have bisimilar states.

In comparison to condition (F) of Def. 2.6.1, (SF) also requires that whatever TS does under its fairness constraint, TS_2 can do respecting fairness constraints of TS_2 but in contrast to (F), (SF) allows that TS and TS_2 visit a different number of (equivalent) states along their way.

Next we show that two stuttering fair bisimilar transition systems satisfy the same $\mathrm{CTL}^*_{\text{-X}}$ properties.

We use the following insight in the proof of Prop. 2.6.3: Given two matching partitions θ of a state sequence π and θ_2 of π_2. If we truncate prefixes of π and π_2 within corresponding segments such that after truncation both sequences start within truncated but corresponding segments, we still get matching partitions for π and π_2 by shifting the segments by the length of the respective truncated prefix. This principle is illustrated by Fig. 2.4.

Firstly, we show that states $s \in S$ and $s_2 \in S_2$ satisfy the same $\mathrm{CTL}^*_{\text{-x}}$ state formulas if s and s_2 are stuttering fairly bisimilar; and if paths π and π_2 match then π and π_2 satisfy the same $\mathrm{CTL}^*_{\text{-x}}$ path formulas.

Proposition 2.6.3 *Let TS and TS_2 be two stuttering fair bisimilar transition systems with fairness constraints $\mathsf{Fair} \subseteq Act$, $\mathsf{Fair}_2 \subseteq Act_2$, respectively. Let $\mathcal{B}$ be a stuttering fair bisimulation relation between TS and TS_2. Let s be a state of TS and π be one of its infinite paths that is fair w.r.t. Fair. Let s_2 be a state of TS_2 and π_2 be an infinite path fair w.r.t. Fair_2. Let φ be a $CTL^*_{\text{-x}}$ state formula and ψ be a $CTL^*_{\text{-x}}$ path formula.*

If $(s, s_2) \in \mathcal{B}$, then $TS, s \models_{\mathsf{Fair}} \varphi$ if and only if $TS_2, s_2 \models_{\mathsf{Fair}_2} \varphi$.

If $\mathsf{match}(\mathcal{B}, \pi, \pi_2)$, then $TS, \pi \models_{\mathsf{Fair}} \psi$ if and only if $TS_2, \pi_2 \models_{\mathsf{Fair}_2} \psi$.

Proof The proof is by induction on the structure of ψ and φ.

$\varphi = p$: $TS, s \models p$ iff $TS_2, s_2 \models p$ follows immediately from $L(s) = L_2(s_2)$.

The cases $\varphi_1 \vee \varphi_2$ and $\neg\varphi_1$ follow directly by the induction hypothesis.

$\varphi = \mathsf{E}\psi$: Let us assume that $TS, s \models \mathsf{E}\psi$. Hence there is a fair path π from s with $TS, \pi \models \psi$. Since (s, s_2) are stuttering fair bisimilar, it follows by Def. 2.6.2 that there is a path π_2 that is fair w.r.t. Fair_2 such that $\mathsf{match}(\mathcal{B}, \pi, \pi_2)$ holds. By the induction hypothesis follows that $TS_2, \pi_2 \models \psi$.

Analogously we derive that $TS_2, s_2 \models \mathsf{E}\psi$ implies $TS, s \models \mathsf{E}\psi$.

$\psi = \mathsf{D}(\varphi)$: $TS_2, \pi_2 \models \mathsf{D}(\varphi)$ holds iff $TS_2, \pi_2(1) \models \varphi$ holds. We assume that $\mathsf{match}(\mathcal{B}, \pi, \pi_2)$ holds. Hence $(\pi(1), \pi_2(1)) \in \mathcal{B}$ holds. By the induction hypothesis, $TS_2, \pi_2(1) \models \varphi$ iff $TS, \pi(1) \models \varphi$.

The cases $\psi_1 \vee \psi_2$ and $\neg\psi_1$ follow again directly by the induction hypothesis.

$\psi = \psi_1 \mathsf{U} \psi_2$: Let us assume that $TS, \pi \models \psi_1 \mathsf{U} \psi_2$. $TS, \pi \models \psi_1 \mathsf{U} \psi_2$ iff there is an index $i \geq 0$ with $TS, \pi^i \models \psi_2$ and $\forall j,\, 0 \leq j < i : TS, \pi^j \models \psi_1$. Since we assume $\mathsf{match}(\mathcal{B}, \pi, \pi_2)$, there are partitions θ of π and θ_2 of π_2, that divide π and π_2 into segments of bisimilar states. Let segment k contain the $(i+1)$-th state of π. Let segment k on π_2 start at the $(i_2 + 1)$-th position. It follows that $\mathsf{match}(\mathcal{B}, \pi^i, \pi_2^{i_2})$ holds. By the induction hypothesis, $TS, \pi^i \models \psi_2$ iff $TS_2, \pi_2^{i_2} \models \psi_2$.

We now show that any $\pi_2^{j_2}$, $0 \leq j_2 < i_2$ satisfies ψ_1. Let segment l contain the $(j_2{+}1)$-th state of π_2. Since segment k starts at the $(i_2{+}1)$-th position, the (j_2+1)-th position is within a preceeding segment, i.e. $l < k$. Let segment l on π start at the $(j+1)$-th position. It follows that $\mathsf{match}(\mathcal{B}, \pi^j, {\pi_2}^{j_2})$ holds. Since segment l preceeds segment k, position j preceeds position i on π. Hence $TS, \pi^j \models \psi_1$ holds. By the induction hypothesis follows that $TS_2, {\pi_2}^{j_2} \models \psi_1$.

Analogously it can be shown that $TS_2, \pi_2 \models \psi_1 \mathsf{U} \psi_2$ implies that $TS, \pi \models \psi_1 \mathsf{U} \psi_2$. □

Theorem 2.6.4 (Stuttering Fair Bisimilarity Implies $\mathrm{CTL}^*_{\text{-X}}$ Equivalence) *Let TS and TS_2 be two transition systems with initial states s_{init} and s_{init2}, respectively, and $AP = AP_2$. Let $\mathsf{Fair} \subseteq 2^{Act}$ and $\mathsf{Fair}_2 \subseteq 2^{Act_2}$ be sets of fairness constraints. Let φ be an $\mathrm{CTL}^*_{\text{-X}}$ formula referring to AP only.*

If $TS_{\mathsf{Fair}} \cong TS_{2\mathsf{Fair}_2}$, then

$TS, s_{\mathsf{init}} \models_{\mathsf{Fair}} \varphi$ *if and only if* $TS_2, s_{\mathsf{init2}} \models_{\mathsf{Fair}_2} \varphi$.

Proof Since $TS_{\mathsf{Fair}} \cong TS_{2\mathsf{Fair}_2}$, there is a stuttering fair bisimulation relation with $(s_{\mathsf{init}}, s_{\mathsf{init2}}) \in \mathcal{B}$. By Lemma 2.6.3 follows, that $TS, s_{\mathsf{init}} \models_{\mathsf{Fair}} \varphi$ if and only if $TS_2, s_{\mathsf{init2}} \models_{\mathsf{Fair}_2} \varphi$. □

Convention When we speak of fairness in the following chapters, we refer to relative fairness unless stated otherwise.

2.7 Summary

In this chapter we introduced the basic terminology used in the following chapters.

In particular, we introduced the basic terminology for Petri nets, we defined the syntax and semantics of the temporal logics CTL* $\forall$CTL*, LTL and CTL (with and without X) and presented the notion of stutter-invariant safety properties. The notion of relative fairness was defined and compared to weak and strong fairness. Finally, we showed how simulation can be used to prove the preservation of $\forall$CTL* properties and stuttering fair bisimulation can be used to prove equivalence w.r.t. $\mathrm{CTL}^*_{\text{-X}}$ properties.

Chapter 3

Alleviating State Space Explosion

Contents

State space explosion is often a major hindrance when model checking real word systems. To combat state space explosion we developed cutvertex reductions and two flavours of Petri net slicing. Certainly other methods exist to combat the state space explosion problem and to accelerate model checking.

In this chapter we give Petri net slicing and cutvertex reductions a place within the landscape of approaches fighting state space explosion.

In Sect. 3.1 we coarsely survey approaches fighting state space explosion and give pointers to literature. In Sect. 3.2 we introduce in more detail Petri net reductions and decompositional methods, the pigeonholes for our

approaches. The combination of different approaches promises even more effective defence against state space explosion, as discussed in Sect. 3.3.

3.1 Alleviating State Space Explosion – An Overview

Formally the model checking problem is the following decision problem:

Given a model M and a temporal logic property φ, does M satisfies φ?

It is well known that the model checking problem for finite state systems and CTL* properties is decidable. Consequently model checking is also decidable on finite state systems for all logics introduced in Sect. 2. To determine whether a system M satisfies a CTL formula φ is linear in the size of φ and the size of the system's state space TS_M. LTL and CTL* model checking can be performed in $\mathcal{O}(|TS_M| \cdot 2^{|\psi|})$ time and space. Since the temporal properties are usually short, the main hindrance of model checking is the immense size of state spaces that arise from the most of interesting systems. The number of states tends to grow exponentially in the system size, which is often referred to as *state space explosion*. Systems of loosely coupled components and many local states suffer more from the state space explosion problem than tightly coupled with little concurrency.

Many ideas exists on how to combat the state space explosion problem and made it possible to successfully verify more and more complex systems. These approaches can be classified according to which aspects of the system they exploit into *state space based methods* on the one side and *structural methods* on the other side. Structural methods exploit the system description—Petri nets in our case—whereas state space based methods target the state space directly. State space based methods can be subdivided into methods that *handle the state space efficiently* (e.g. symbolic [71] or on-the-fly model checking [49]) or they *build an optimised state space* (e.g. partial order reductions [45]) or *use another state space representation* (e.g. unfoldings [34]).

Each of these methods has its strengths but also its weaknesses, i.e. they may work very well for one system but do not lead to any improvement for another. For instance, symbolic model checking tackles state space explosion

by using an efficient encoding of the state space. It examines not single states and state transitions but rather operates on sets of states and state transitions that are symbolically represented as propositional logic formulas. How efficient this encoding is depends on the variable order chosen for the encoding. Determining an optimal order is computationally hard [1], so that heuristics are used and in some cases the encoding is of exponential size.

State space based methods are very powerful, as they can use the full information of the state space. Usually they use of coarse heuristics based on only some information, as there is a trade-off between their reduction impact and the cost of applying them. Structural methods analyse the model structure and do not consider the model's state space. Hence structural methods do not suffer from the state space explosion problem and are usually cheap to apply and even small savings pay off.

For a more detailed overview of methods alleviating state space explosion the interested reader is referred to [6, 22, 102].

3.2 Classifying Slicing and Cutvertex Reductions

Our Petri net slicing techniques are purely structural approaches. Based solely on the Petri net graph an equivalent subnet is determined. Cutvertex reductions implement a compositional minimisation approach and are as such a state based approach. A given monolithic Petri net is decomposed (based on a structural criterion) into a kernel containing the set of places φ refers to, and environment nets. To determine the appropriate replacement for an environment net, the environment is model checked in isolation. The optimisations of cutvertex reductions, micro reductions and pre-/postset optimisations, examine structural criteria.

Both approaches, cutvertex reductions and slicing, are Petri net reductions, since they transform the Petri net graph decreasing its size. This allows

[1] Already the problem to decide whether a given variable ordering is optimal is NP-hard [6]

to conveniently combine them with other techniques.

In the sequel we give a short introduction to compositional minimisation and Petri net reductions.

3.2.1 Compositional Methods

Compositional methods try to bypass the combinatorial blow-up by avoiding the construction of the global state space. Instead they focus on examining a system component-wise. *Compositional minimisation/reduction* constructs a semantically equivalent representation of the global system out of minimised/reduced component state spaces. The resulting minimised/reduced (global) system can then be model checked. *Compositional verification* allows to infer global properties from verification of local properties on the system's components. There, a principal challenge is to find the local properties which are to be checked on the components. *Assume-guarantee reasoning* is one example of a compositional verification technique. For assume-guarantee reasoning it is checked whether a component guarantees the local property ϕ, when it can assume that its environment satisfies an assumption ψ. Then it must be shown that its actual environment satisfies the assumption ψ.

A principal challenge of using compositional methods for monolithic Petri nets is to find an appropriate decomposition, because of the so called *environment problem*. It is possible that a component exposes spurious behaviour in isolation, that is behaviour that the component does not have as part of the global system due to context constraints imposed by the component's environment.

There are many works on compositional reasoning. In Sect. 5.8 we will discuss decompositional approaches on Petri nets as related work of cutvertex reductions. As an entry point to compositional methods we recommend the survey [8] on compositional model checking techniques used in practice and the surveys of [92, 81].

3.2.2 Petri Net Reductions

Petri net reductions are transformations of the Petri net graph that decrease its size. Many Petri net reduction rules have been defined over the years [10, 75, 25, 30] but focus of most of this research was on special properties like liveness or boundedness rather than the preservation of temporal logic properties.

Petri net reductions for model checking aim to transform a Petri net such that the reduced net has a smaller state space but is equivalent with respect to a given property. Temporal property preserving reductions were presented in more recent works which were partly based on the former reduction rules [85, 38]. Poitrenaud and Pradat-Peyre showed in [85] that the local net reduction rules called pre- and postagglomeration of Berthelot [9] preserve $\mathrm{LTL}_{-\mathsf{X}}$ properties. In [38] Esparza and Schröter presented a set of reduction rules based on invariants, implicit places and local net reductions to speed up $\mathrm{LTL}_{-\mathsf{X}}$ model checking using unfoldings, so that only some the reductions preserve linear time properties. There they also adopted the pre-/postagglomerations.

In the following we introduce pre-/postagglomerations, as one of the most established Petri net reductions for model checking. We mainly follow [85]. Then in Sect. 4.4 we contrast our slicing methods to agglomerations and in Chap. 6 we compare the effects of agglomerations to our approaches.

Pre- and Postagglomerations To apply an agglomeration we consider a place, its set of input transitions H and its set of output transitions F. The aim is to define restrictions so that H and F can be agglomerated—that is we introduce a transition for each pair $(h, f) \in H \times F$ and eliminate place p. More formally, two sets of transitions, F and H, and a place p satisfy the agglomeration scheme iff (1) ${}^\bullet p = H$, $p^\bullet = F$, (2) $M_{\mathsf{init}}(p) = 0$, (3) $\forall h \in H, \forall f \in F : W(p, f) = W(h, p) = 1$ and (4) $F \cap H = \emptyset$. The agglomeration scheme is illustrated in Figure 3.1 (a).

A set of transitions F is *preagglomerateable* if there is a place p and a transition h such that (1) ${}^\bullet p = \{h\}$ and F, $\{h\}$, p satisfy the agglomer-

ation scheme, (2) $h^\bullet = p$ and (3) ${}^\bullet h \neq \emptyset$ and $\forall q \in {}^\bullet h, q^\bullet = \{h\}$. The preagglomeration rule is illustrated in Fig. 3.1 (b).

Fig. 3.1 (c) illustrates the postagglomeration rule. A set of transitions H is *postagglomerateable* iff there is a place p and a set of transitions F such that F, H, p satisfy the agglomeration scheme, ${}^\bullet F = \{p\}$ and $h^\bullet \neq \emptyset$.

Intuitively when transition sets H and F are agglomerateable, the place p reliably stores the token that transitions in F consume. If F is preagglomerateable h just takes tokens from its input place to generate a token onto p without any other side-effects. If H is postagglomerateable, transitions in F are enabled right after firing a transition in H.

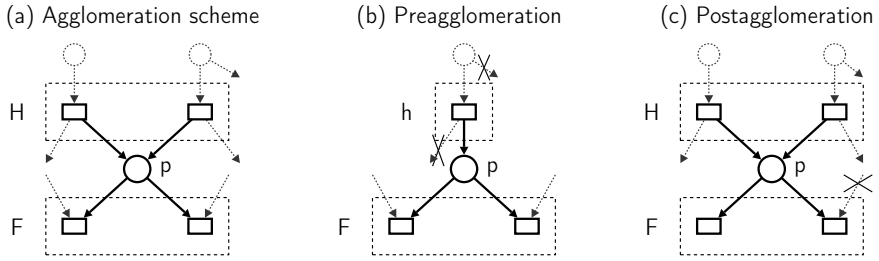


Figure 3.1: The Agglomerations: Preagglomeration is illustrated in (b) and postagglomeration in (c). In grey dashed lines are parts that rule's preconditions refer to.

Place p can be removed from the net and transitions in H, F are merged, i.e. H, F are removed from the net and new transitions $(hf) \in H \times F$ are introduced. A transition (hf) has all input places of F and H except the eliminated p and (hf) has all output places of F and H except the eliminated p. In [85] it has been shown that agglomerations of transition sets F or H preserve an $\text{LTL}_{-\mathsf{X}}$ property φ, given F is preagglomerateable and h does not effect the places referred to by φ, or given H is postagglomerateable and transitions in F do not effect any places φ refers to.

3.3 Alliance Against State Space Explosion

In Sect. 3.1 we gave a coarse overview of approaches to tackle the state space explosion problem. However, it is very difficult to know which method

yields the best reduction rate. So several verification tools combine different approaches to gain a synergetic effect, like SPIN, PROD or NuSMV. There is also ongoing research on how to develop elaborate combinations of the different approaches, e.g. [95, 111, 12]. Petri net reductions—or model reductions in general—conveniently allow to be daisy chained with other methods. So our methods can be used as preprocessing step before applying other methods.

When using Petri net reductions for preprocessing, one has to be aware of the side effects on the succeeding methods. We discussed in Sect. 3.1 that for instance symbolic model checking uses heuristics to select a variable ordering for its state space encoding. Similarly, partial order reductions build a condensed state space by heuristically choosing representatives for a class of equivalent interleavings (cf. Sect. 3.3.1). When Petri net reductions are applied first to simplify a net and thereby decrease the size of its state space, the heuristics may perform differently, that is better *or* worse. But for most techniques its worst case performance is bounded by the size of its state space, so that a reduced net with smaller state space guarantees a better worst case behaviour.

In Sect. 6 we empirically study the effects of using our methods as a preprocessor for partial order reductions. As we will see, partial order reductions exhibit some conceptual similarities to both slicing and cutvertex reductions, but both techniques bring in complementary ideas to further repel state space explosion. In the following we therefore introduce partial order reductions.

3.3.1 Partial Order Reductions

One reason of state space explosion is that the interleaving semantics represents concurrency of actions by interleaving them in all possible ways, whereas the actions' total effect is independent of their ordering. PORs (Partial order reduction s) condense state spaces by decreasing the number of equivalent interleavings in the model's state space TS_M.

In the following we present *stubborn-set-type methods*[2] following mainly the presentation of Valmari in [102] but focusing on Petri nets. In [102] the term *stubborn-set-type method* is a generic term referring to ample, persistent or stubborn set methods.

Stubborn-set-type methods build a reduced state space by constructing representative interleavings postponing independent transitions. Starting at the initial state, a set of transitions $\mathcal{T}(s)$ is computed for each state s that a stubborn-set-type method encounters during the state space construction, and only successors reachable via transitions in $\mathcal{T}(s)$ are explored.

Valmari introduces the notion of *dynamic* stubborn sets to specify the characteristics a stubborn-set-type method has to guarantee for its stubborn sets.

Definition 3.3.1 *A set $\mathcal{T}(M_0) \subseteq T$ of transitions is* dynamically stubborn *at state $M_0 \in [M_{\mathsf{init}}\rangle$, if and only if the following hold:*

D1 *If $t_s \in \mathcal{T}(M_0)$, $t_1, ..., t_n \notin \mathcal{T}(M_0)$, $M_0[t_1...t_n\rangle M_n$ and $M_n[t_s\rangle \hat{M}_n$, then there is $\hat{M}_0 \in [M_{\mathsf{init}}\rangle$ such that $M_0[t_s\rangle \hat{M}_0$ and $\hat{M}_0[t_1...t_n\rangle \hat{M}_n$.*

D2 *There is at least one $t_k \in \mathcal{T}(M_0)$, such that if $t_1, ..., t_n \notin \mathcal{T}(M_0)$ and $M_0[t_1...t_n\rangle M_n$, then $M_n[t_k\rangle$.*

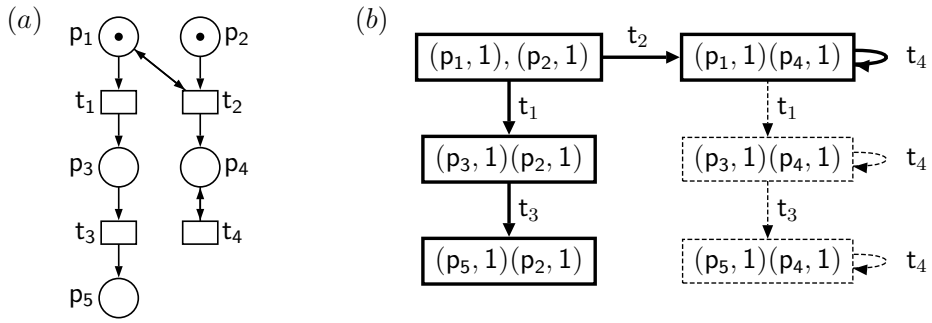


Figure 3.2: State space condensation by stubborn set type methods: The condensed state space is the boldly printed part of (b). This state space condensation complies with *D1* and *D2*.

[2]We present here only the strongly dynamic stubborn sets.

Fig. 3.2 shows a condensed state space complying with Def. 3.3.1. As the definition refers to states in the full state space, to implement stubborn set methods, strategies are necessary to guarantee the independence of transitions without referring to the full state space. Such strategies define sufficient criteria based on the modelling formalism to imply that stubborn sets are dynamic. Different such strategies can be defined depending on how much effort is spent on analysing the dependencies between transitions. The following is a simple definition of static stubborn sets for Petri nets guaranteeing *D1* and *D2*.

1. *If* $t_s \in \mathcal{T}(M_0)$ *and* $\neg M_0[t_s\rangle$*, then there is* $p \in {}^\bullet t_s$ *such that* $M(p) < W(p,t_s)$ *and* ${}^\bullet p \subseteq \mathcal{T}(M_0)$.
2. *If* $t_s \in \mathcal{T}(M_0)$ *and* $M_0[t_s\rangle$*, then* $({}^\bullet t_s)^\bullet \subseteq \mathcal{T}(M_0)$.
3. $\mathcal{T}(M_0)$ *contains a transition* t_s *such that* $M_0[t_s\rangle$.

A more refined definition of stubborn sets is given by:

1. *If* $t_s \in \mathcal{T}(M_0)$ *and* $\neg M_0[t_s\rangle$*, then there is* $p \in {}^\bullet t_s$ *such that* $M_0(p) < W(p,t_s)$ *and* $\{\hat{t} \mid W(p,\hat{t}) < W(\hat{t},p) \wedge W(p,\hat{t}) \leq M_0(p)\} \subseteq \mathcal{T}(M_0)$.
2. *If* $t_s \in \mathcal{T}(M_0)$ *and* $M_0[t_S\rangle$*, then for every* $p \in {}^\bullet t_s$*,* $\{\hat{t} \mid min(W(t_s,p), W(\hat{t},p)) < min(W(p,t_s), W(p,\hat{t}))\} \subseteq \mathcal{T}(M_0)$.
3. $\mathcal{T}(M_0)$ *contains a transition* t_s *such that* $M_0[t_s\rangle$.

These two definitions of structural stubborn sets are nondeterministic. Depending on the start transition different stubborn sets are constructed. Several (or all) this stubborn sets can be computed and based on a heuristics one is chosen. Usually smaller stubborn sets are preferred.

Preserved Properties Various stubborn-set-type methods have been defined preserving a variety of different properties like termination, safety, $\text{LTL}_{\text{-x}}$ or $\text{CTL}^*_{\text{-x}}$ properties. The stubborn sets as defined by *D1* and *D2* generate a condensed state space that contains all final markings of a net reachable from its initial marking. All final states in the condensed state space are also final

markings of the net. Furthermore, the reduced state space contains an infinite execution if and only if the full state space contains an infinite execution. To preserve more complex properties, additional conditions on stubborn sets are necessary.

The example in Fig. 3.2 demonstrates that *D1* and *D2* are not sufficient to guarantee preservation of LTL$_{-\mathsf{X}}$. The LTL$_{-\mathsf{X}}$ property $\mathsf{G}((p_3, 1) \Rightarrow \mathsf{F}(p_5, 1))$ holds on the reduced state space but not on the full state space. This is due to the so called *ignoring problem*, i.e. in the reduced state space is a path on which some transitions can be infinitely postponed. In our example the transition t_1 is ignored on the path corresponding to $t_2 t_4 t_4 \ldots$.

The conditions V and L, given below, guarantee the preservation of an LTL$_{-\mathsf{X}}$ property φ where *effect*(φ) can be any overapproximation of the set of observable transitions, i.e. it has to hold that $t \in$ *effect*(φ), if there are reachable markings $M_1, M_2 \in [M_{\mathsf{init}}\rangle$ and an atomic proposition (p, x) of φ such that $(M_1[t\rangle M_2) \wedge ((p, x) \in L(M_1) \Leftrightarrow (p, x) \notin L(M_2))$.

V *If the stubborn set $\mathcal{T}(M_0)$ contains a transition t_v such that $M_0[t_v\rangle$ and $t_v \in$ effect(φ), then $\mathcal{T}(M_0) = T$.*

L *If $M_1[t_1\rangle M_2[t_2\rangle M_3 \ldots$ is an infinite execution in the reduced state space starting at a marking M_1, then for each $t_v \in$ effect(φ) there is an index $i \geq 1$ such that $t_v \in \mathcal{T}(M_i)$.*

Conditions V and L are a sufficient proviso for preserving liveness properties.

3.4 Summary

In this chapter we gave a coarse overview of methods to tackle the state space explosion problem of model checking. As pigeonholes of our approaches we introduced decompositional methods and Petri net reductions. We outlined that the combination of different methods promises further improvement for model checking. As two examples for other approaches fighting state space

explosion, we presented in more detail agglomerations and partial order reductions.

Chapter 4

Slicing Petri Nets

Contents

4.1 Introduction

In this chapter we introduce the approach of *Petri net slicing*. Slicing is a technique to syntactically reduce a model in such a way that at best the

reduced model contains only those parts that may influence the property the model is analysed for. It originated as a method for program debugging but has found applications in many other domains. We introduce slicing as a means to alleviate the state space explosion problem for model checking Petri nets. Tailoring slicing for model checking Petri nets, allows to better exploit the Petri net graph and to fine-tune the slicing algorithms to preserve relevant classes of properties.

We develop two flavours of Petri net slicing, *CTL$^*_{-\mathsf{X}}$ slicing* and *safety slicing*. As means for alleviating the state space explosion problem for model checking, they determine what parts of the Petri net Σ can be sliced away (i.e. discarded) so that the remaining net is equivalent to the original w.r.t. ψ. The remaining net is called *slice* Σ' and is built for a so called slicing criterion *Crit*. We use as *Crit* the set of places referred to by the examined CTL$^*_{-\mathsf{X}}$ property, $scope(\psi)$.

We will show that CTL$^*_{-\mathsf{X}}$ slices allow to verify and falsify CTL$^*_{-\mathsf{X}}$ properties assuming relative fairness on the original net and safety slices allow for verification and falsification of safety properties (without fairness constraints on the original net). Slices of both algorithms can be used to falsify $\forall$CTL*. We will see that safety properties allow for more aggressive slicing, while the CTL$^*_{-\mathsf{X}}$ preserving algorithm will usually produce bigger slices.

Outline In the remainder of the section we will survey the history of Petri net slicing. The slicing algorithm preserving CTL$^*_{-\mathsf{X}}$ properties is defined in Sect. 4.2. In Sect. 4.3 we present the safety slicing algorithm. We discuss related work in Sect. 4.4, before drawing the conclusions in Sect. 4.6 and outlining possibilities for future work in Sect. 4.5.

4.1.1 The History of Petri Net Slicing

Program Slicing The term *slicing* was coined by Mark Weiser in his original publication on program slicing [112], where he introduced slicing as a formalisation of an abstraction technique that experienced programmers (unconsciously) use during debugging to minimise the program by "slicing

away" bits that are not relevant for the current analysis.

The relevant part of the program, the *slice*, is determined with respect to a *slicing criterion* that specifies which aspect of the program is of interest. Depending on the actual slicing algorithm, the slicing criterion usually is a line number within the program code and a set of variables plus additional information the slicing algorithm may use, like an input value.

Let us consider the small example program in Fig. 4.1 to explain the basic idea of Weiser's slicing algorithm. As in [112] we take as slicing criterion a line number and a set of variables, C=(`line 9,{sum}`). The slice is built by tracing backwards possible influences on the variables: In `line 6` `sum` is increased by `i`, so we also need to know the value of `i` at `line 6`. Hence `i` becomes a relevant variable. Whether `line 6` is executed depends on the control statement at `line 5`, which refers to variable `n`. Hence `n` is also relevant. Tracing backwards we see that the relevant variable `sum` is set to value zero in `line 3`. From now on `sum` is not relevant anymore, because earlier changes are overwritten in `line 3`. Analogously, `i` ceases to be relevant at `line 2`. To determine a program slice, Weiser's algorithm computed such sets of relevant variables according to data dependencies (e.g. `sum` depends on the value of `i`) and control dependencies (e.g. `n` determines how often `sum` is increased).

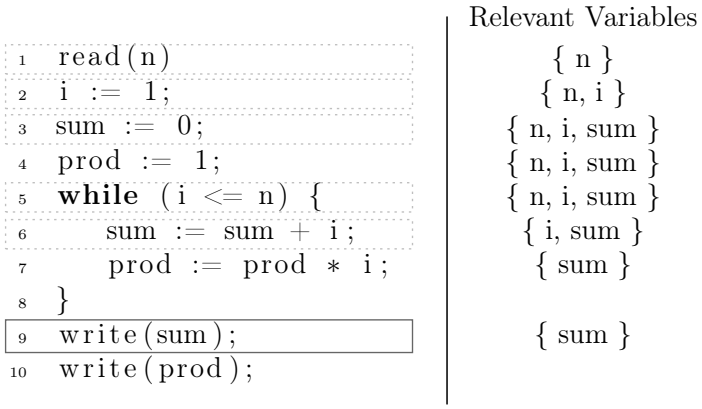


Figure 4.1: A program slice for slicing criterion (9,{sum}).

Since the original publication of Weiser in 1981, a variety of slicing approaches have been developed and program slicing has successfully been ap-

plied to support software developers in tasks like program understanding, integration, maintenance, testing and software measurement [99, 13]. One major challenge in program slicing is to appropriately capture the relevant dependencies of high level programming languages.

There are two main classifications of slices: (i) By the direction of slicing and (ii) by knowledge on the program's input used by the slicing algorithm. In the above example we have built a *backward slice*. Starting from the slicing criterion we traced backwards the possible influences on the variables. A backward slice contains the statements which may affect the slicing criterion. A *forward slice* contains the statements that are affected by the slicing criterion.

Classified by the amount of knowledge on the program's input, slices are called *static, dynamic* or *conditioned*. The slice in Fig. 4.1 is static. Static slices are built without any knowledge of the program's input, whereas dynamic slices are built for exact input values of the program. Suppose `n=0` is the input to the program in Fig. 4.1. A dynamic slicing algorithm could generate a slice consisting of `lines 3` and `9` only. Conditioned slices are built without knowing the exact input but sets of initial values usually given by a first-order logic formula on the input variables [28].

Slicing Formal Specifications & Cone of Influence Reduction Research was also undertaken to apply slicing on formal specifications. Without claiming completeness we representatively survey works on slicing formal specifications in order to illustrate how wide spread research on this field developed. In [97] the concept of slicing has been applied to attribute grammars. Heimdahl and Whalen defined slices of hierarchical state machines in [54]. J. Chang and D. J. Richardson and also Brückner and Wehrheim published works on slicing Z- and CSP-OZ-Specifications [15, 14, 17]. An slicing approach for VHDL is described in [21].

Cone of influence reduction (COI)[8] is a related approach used in hardware verification. To the author's opinion it is hard to differentiate when a method classifies as either slicing or COI technique. Both techniques build a reduced model by analysing dependencies and omitting independent parts.

Historically the focus for slicing applications is wider and mainly on debugging and testing. Slicing techniques are usually applied to complex high level languages and most slicing research discusses how to extract various dependencies necessary for the desired analysis. In contrast, COI was studied for simplifying models for verification right from the start. Historically COI is applied on synchronous systems, for which it has been shown to preserve CTL$^*_{-\mathsf{X}}$[22]. COI usually is applied on simpler modelling formalisms, e.g. boolean equations describing an asynchronous circuit. Whereas slicing encompasses a variety of approaches, COI usually refers to the backward tracing of dependencies only. So the slicing algorithm implemented in the NuSMV model checker [20] as well as the algorithm for fFSM models as used in the PeaCE hardware/ software codesign environment[79] are referred to as COI implementations, whereas Clarke et. al. describe in [21] a slicing technique for a hardware description language.

4.2 CTL$^*_{-\mathsf{X}}$ Slicing

The basic idea for our slicing algorithm is to define dependencies based on the locality property of Petri nets: The token count of a place p is determined by the firings of incoming and outgoing transitions of p. Whether such a transition can fire, depends on the token count of its input places.

If we want to observe the marking on a set of places *Crit*, we can iteratively construct a subnet $\hat{\Sigma} = (\hat{P}, \hat{T}, \hat{W}, \hat{M}_{\mathsf{init}})$ of Σ by taking all incoming and outgoing transitions of a place $p \in \hat{P}$ together with their input places, starting with $\hat{P} = \mathit{Crit}$. The subnet $\hat{\Sigma}$ certainly captures every token flow of Σ that influences the token count of a place $p \in \mathit{Crit}$.

We refine the above construction by distinguishing between *reading* and *non-reading* transitions. A reading transition of places R cannot change the token count of any place in R. We formally define t to be a reading transition of $R \subseteq P$ iff $\forall p \in R : W(p,t) = W(t,p)$. If t is not a reading transition of R, we call t a non-reading transition of R. Let us now iteratively build a subnet $\Sigma' = (P', T', W', M'_{\mathsf{init}})$ by taking all non-reading transitions of a place $p \in P'$ together with their input places, starting with $P' = \mathit{Crit}$.

Definition 4.2.1 (*slice*(Σ, Crit)) *Let Σ be a marked Petri net and $\mathit{Crit} \subseteq P$ a non-empty set, called* slicing criterion. *The following algorithm constructs* *slice*(Σ, Crit) *of Σ for the slicing criterion Crit.*

```
1   generateSlice(Σ, Crit){
2     T', P_done := ∅ ;
3     P' := Crit ;
4     while ( ∃p ∈ (P' \ P_done) ) {
5       while ( ∃t ∈ ((•p ∪ p•) \ T') : W(p,t) ≠ W(t,p) ) {
6         P' := P' ∪ •t ;
7         T' := T' ∪ {t} ; }
8       P_done := P_done ∪ {p} ; }
9     return (P', T', W|_(P',T'), M_init|_P') ; }
```

The algorithm always terminates and always determines a subnet *slice*(Σ, Crit) for any given slicing criterion *Crit*. Though, the slice may equal the original net Σ. If $\mathit{Crit} \subseteq \mathit{Crit}'$, *slice*$(\Sigma, \mathit{Crit})$ is a subnet of *slice*(Σ, Crit'). Figure 4.2 illustrates the effect of `generateSlice`.

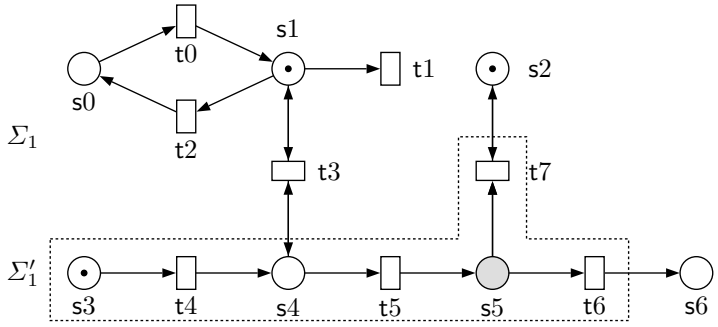


Figure 4.2: Slicing a Petri net. The original net Σ_1 and its slice $\Sigma_1' = \mathit{slice}(\Sigma_1, \{s5\})$.

The slice *slice*(Σ, Crit) may be smaller than $\hat{\Sigma}$, the subnet constructed without considering reading transitions. Even for certain strongly connected nets the algorithm `generateSlice` might produce a slice Σ' that is smaller than Σ, whereas $\hat{\Sigma}$ for a strongly connected net is always equals to Σ. As illustrated in Fig. 4.3, such a subnet evolves without tokens being generated

by the remaining net: Reading transitions are the only incoming transitions.

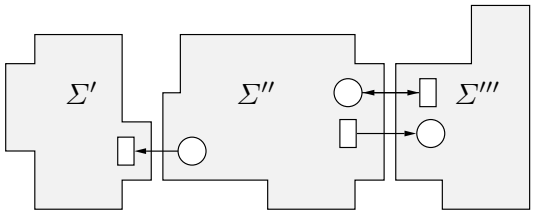


Figure 4.3: Slices of a Petri net. The net has (at least) four possible slices Σ'', $\Sigma'\Sigma''$, $\Sigma''\Sigma'''$ and the original net itself.

Slicing Effects The effect of our slicing algorithm can be classified into trivial, proper and effective. As already noted $slice(\Sigma, Crit)$ may equal the original net Σ, in which case we refer to $slice(\Sigma, P)$ as *trivial*. We call a slice *proper*, if it is a proper subnet of the original net. A slice is *effective* if it is proper and its state space is less than the state space of the original system. We will show that the slice's states are a subset of the (projected) states of the original system and the state transitions are a subset of the state transitions of original system. So a slice is either (i) trivial, (ii) proper and ineffective or (iii) effective.

Figure 4.4 shows a proper slice that is ineffective. In Fig. 4.4 (a) the original Petri net is displayed and its slice for $\{p5\}$ is marked by a dashed frame. The reachability graphs of the original system and $slice(\Sigma, \{p5\})$ are displayed in Fig. 4.4 (b) and (c), respectively.

If we generate the slice of the same system but for place $p4$ we have an effective slice. This case is illustrated in Fig. 4.5.

4.2.1 Nets, Slices and Fairness

In this section we want develop an intuition on how precisely a slice captures the behaviour of the original net with respect to the places mentioned in the slicing criterion *Crit*.

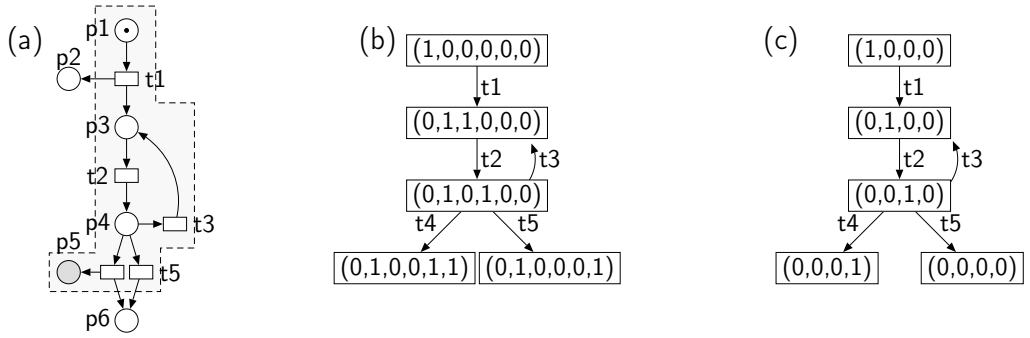


Figure 4.4: Example of a proper but ineffective slice. (a) shows the original net Σ and $slice(\Sigma, \{p5\})$, (b) the state space of Σ and (c) the state space of $slice(\Sigma, \{p5\})$.

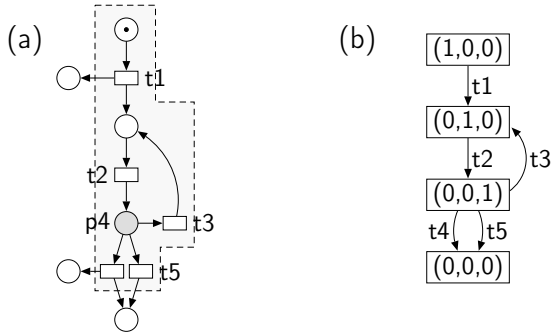


Figure 4.5: Example of an effective slice. (a) shows the original net Σ and $slice(\Sigma, \{p4\})$ and (b) the state space of $slice(\Sigma, \{p4\})$.

Firing Sequences on the Original and its Slices So let us consider the firing sequence $\sigma = t_2\, t_0\, t_4\, t_3\, t_5$ of Σ_1 in Fig. 4.2. Firing σ generates the marking with a token on s_1, s_2, s_5 only. σ is certainly not executable on Σ_1' as such, since t_0, t_2 and t_3 are not transitions of Σ_1'. Omitting these transitions, $proj_{T_1'}(\sigma) = t_4 t_5$ remains. $t_4 t_5$ is a firing sequence of Σ_1' and generates the marking with only a token on s_5. So the markings resulting from firing σ on Σ_1 and $proj_{T_1'}(\sigma)$ on Σ_1' coincide on the places in P_1'.

Actually, it is always the case that for a firing sequence σ of Σ, $proj_{T'}(\sigma)$ is a firing sequence of its slice Σ' and that σ and $proj_{T'}(\sigma)$ change the token count on P' in the same way. In section 4.2.2 we will formally show that every firing sequence σ' of a slice Σ' is also a firing sequence of Σ, and the projection $proj_{T'}(\sigma)$ of a firing sequence σ of Σ is a firing sequence of Σ'. But since we are interested in the preservation of temporal properties, reachability of (sub)markings is not enough.

Temporal Properties and Maximal Firing Sequences According to Chapter 2 the satisfiability of path formulas is determined by maximal firing sequences (c.f. Def. 2.3.2 and Def. 2.4.1). Since we would like to preserve LTL or CTL, we would hence like a correspondence of maximal firing sequences of Σ and Σ'. Unfortunately this is not the case and we cannot verify CTL or LTL using the slice right away, as the following example illustrates.

Let us consider the formula $\varphi = \mathsf{AF}(s_5, 1)$, which is an LTL$_{\text{-x}}$ and CTL$_{\text{-x}}$ property. The slice Σ_1' satisfies this property: Σ_1' has two maximal firing sequences $t_4\, t_5\, t_6$ and $t_4\, t_5\, t_7$. Both sequences fire t_5 and hence mark eventually s_5 with a token. Σ does not satisfy φ, since the maximal firing sequence $\sigma = t_4\, t_3\, t_3\, t_3 \ldots$ never generates a token on s_5. The projection of σ on T_1', $proj_{T_1'}(\sigma) = t_4$, is not maximal on Σ'.

Intuitively, the reason for the non-correspondence between maximal firing sequences of Σ and Σ' is that the net *discard*—the bit that is sliced away—exposes a divergence that is not reflected within the slice. As a consequence maximal firing sequences on Σ are projected onto non-maximal firing sequences on Σ'. One way to fix this problem is to rule out divergencies outside the slice by means of a fairness assumption on the original net.

Fairness Rules out Divergencies In the following we will use a very weak fairness assumption to rule out divergencies within the net discard. In Def. 2.4.3 we introduced a firing sequence to be relatively fair with respect to a fairness constraint $F \subseteq T$ if in case a transition $t \in F$ is eventually permanently enabled, some transition of F is fired infinitely often. In the following we will set F to the set of transitions of the slice. This fairness assumption guarantees progress within the slice. As long as there are transitions in T' permanently enabled, transitions in T' will be fired. Our example firing sequence $\sigma = t_4\, t_3\, t_3\, t_3 \ldots$ is not fair with respect to T', as t_5 is permanently enabled, but no transition in T' is fired.

Note that we do not always need to make this fairness assumption to verify a property using the slice. Guaranteeing progress is only necessary when studying liveness properties. Also if we want to falsify a property using the slice it is not necessary to assume that Σ is fair w.r.t. T', as we will see.

Fairness and Maximality We will show that any firing sequence of Σ that is fair with respect to T' is projected onto a maximal firing sequence of Σ' and that any maximal firing sequence of Σ' corresponds to a fair firing sequence on Σ.

Other fairness assumptions could be made to guarantee progress on T' and hence to rule out divergencies within the net discard. We may for instance assume stronger fairness notions like weak fairness or strong fairness.

Stuttering Marking Sequences As we consider state based logics we are not primarily interested in correspondences of firing sequences but in the induced changes of the states (=markings). Since a firing sequence of Σ and its projection $proj_{T'}(\sigma)$ fired on Σ' change the token count on places in P' in the same way, the generated marking sequences are quite similar.

Consider again the net Σ_1 in Fig. 4.2 and its slice Σ_1'. $\sigma_1 = t_4\, t_1\, t_5\, t_6$ and $\sigma_2 = t_4\, t_3\, t_1\, t_5\, t_6$ are both maximal firing sequences of Σ generating the marking M with a token on s_2 and s_6 only. $\sigma' = t_4\, t_5\, t_6$ is the projection onto T_1' of both σ_1 and σ_2. Since σ', σ_1 and σ_2 are from the slice's point of view the same,

we want $\mathcal{M}(M'_{\mathsf{init}}, \sigma')$ correspond to $\mathcal{M}(M_{\mathsf{init}}, \sigma_1)$ and $\mathcal{M}(M_{\mathsf{init}}, \sigma_2)$. Figure 4.6 illustrates that when we merely restrict the markings of $\mathcal{M}(M_{\mathsf{init}}, \sigma_1)$ and $\mathcal{M}(M_{\mathsf{init}}, \sigma_2)$ to P'_1, the result is the same as $\mathcal{M}(M'_{\mathsf{init}}, \sigma')$ except for stuttering, that is finite repetitions of (sub)markings (c.f. Sect. 2.1). This stuttering is due to the firing of transitions outside the slice. σ_1 fires t_1 between the slice's transitions t_4 and t_5, and σ_2 fires $t_3\,t_1$ between t_4 and t_5.

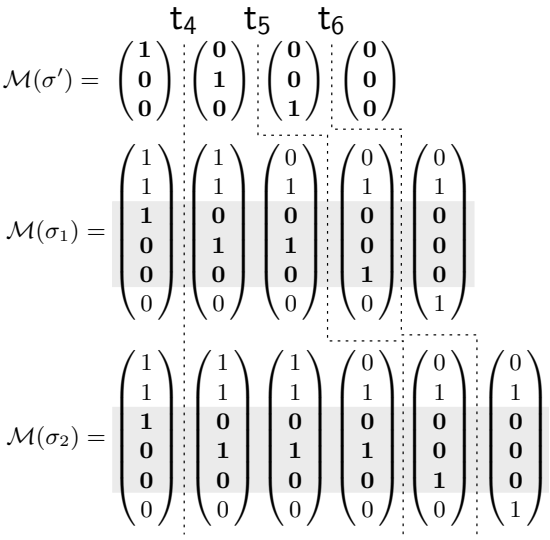


Figure 4.6: Correspondence of marking sequences. Marking sequences $\mathcal{M}(M_{\mathsf{init}}, \sigma_1)$, $\mathcal{M}(M_{\mathsf{init}}, \sigma_2)$ on Σ_1 are both generated from firing sequences corresponding to σ' on Σ'_1 (cf. Fig. 4.2).

Stuttering and Next-Time When studying the previous example it becomes obvious that by considering the slice we cannot say how many steps (= transition firings) the original net will make to reach a certain submarking. But the next-time operator X counts steps. Let us examine what this means for CTL formulas using X.

In Fig. 4.2 the CTL formulas $\varphi_1 = \mathsf{EX}\ \mathsf{EX}\ \mathsf{EX}\ (s_5, 1)$ and $\varphi_2 = \mathsf{EX}\ \mathsf{EX}\ \mathsf{EX}\ \mathsf{EX}\ (s_5, 1)$ are valid on Σ_1. $\mathcal{M}(M_{\mathsf{init}}, \sigma_1)$ and $\mathcal{M}(M_{\mathsf{init}}, \sigma_2)$ represent such marking sequences. But there is no marking sequence on Σ'_1 satisfying either φ_1 or φ_2. The CTL formula $\varphi_3 = \mathsf{AX}\,\mathsf{AX}\,(s_5, 1)$ holds for Σ'_1, but obviously not for Σ_1. Hence the slice can neither be used for verification or falsification of CTL formulas using X.

Let us now examine LTL properties using X: The LTL property $\psi = \mathsf{AXX}\ (s_5, 1)$ is satisfied by Σ_1', but not by Σ_1. Hence in general Σ' cannot be used for verification of LTL formulas using X, as $\Sigma' \models \varphi \not\Rightarrow \Sigma \models \varphi$. But we will show later in this section that indeed Σ' can be used for falsification of $\forall\text{CTL}^*$ and hence LTL properties using X.

4.2.2 Proving $\text{CTL}^*_{-\mathsf{X}}$-Equivalence

In Sect. 4.2.2.1 we concentrate on the correspondences between Σ and Σ'. As outlined in the previous section, we consider two transition sequences σ of Σ and σ' of its slice Σ' as correspondent iff $proj_{T'}(\sigma) = \sigma'$ and two markings M of Σ and M' of Σ' are correspondent iff $M|_{P'} = M'$. We first show that the (sets of) firing sequences of Σ and Σ' correspond. Then we show that maximal firing sequences of Σ' correspond to firing sequences of Σ that are fair w.r.t. T'.

In Sect. 4.2.2.2 we examine what the discovered correspondences mean for the set of formulas that Σ and Σ' satisfy. We show that Σ and Σ' satisfy the same $\text{CTL}^*_{-\mathsf{X}}$ formulas, by proving that the two nets have stuttering fair bisimilar transition systems. We also show that $\forall\text{CTL}^*$ using the next-time operator X can be falsified via the slice, which implies that LTL using X can be falsified.

Convention In what follows we denote with Σ' the slice of a given net Σ for a slicing criterion $Crit \subseteq P$. If we interpret a temporal logic formula φ on a net Σ we assume that the formula refers to places of the slice only, that is we assume that $scope(\varphi) \subseteq P'$ holds.

4.2.2.1 Firing Sequences and Slicing

We start with two simple observations:

(i) The occurrence of a transition $t \in T \setminus T'$ cannot change the token count of any place in P' whereas the occurrence of a transition $t \in T'$ changes the token count of at least one place in P'.

(ii) A marking M of Σ enables a transition $t \in T'$ if and only if a marking

$M' = M|_{P'}$ of Σ' enables t, since transitions in T' have the same input places in Σ and Σ'.

It follows by induction on the length of the firing sequences, that whatever one of the nets can do to a marking on P', the other can do as well by firing the same transitions in T' in the same order. So for firing sequences there is a correspondence between Σ and Σ', i.e. $proj_{T'}(\mathsf{Fs}_N(M_{\mathsf{init}})) = \mathsf{Fs}_{N'}(M'_{\mathsf{init}})$:

Proposition 4.2.2 *Let σ be a firing sequence and M be a marking of Σ.*
(i) $M_{\mathsf{init}}\,[\sigma\rangle\, M \Rightarrow M'_{\mathsf{init}}\,[proj_{T'}(\sigma)\rangle\, M|_{P'}$.
Let σ' be a firing sequence and M' a marking of Σ'.
(ii) $M'_{\mathsf{init}}[\sigma'\rangle M' \;\Rightarrow\; \exists M \in \mathbb{N}^{|P|} : M' = M|_{P'} \wedge M_{\mathsf{init}}[\sigma'\rangle M$.

Proof We show Prop. 4.2.2 by induction on the length l of σ and σ', respectively.

$l = 0$: The initial marking of Σ and Σ' is generated by firing the empty firing sequence ε. By Def. 4.2.1, $M'_{\mathsf{init}} = M_{\mathsf{init}}|_{P'}$.

$l \to l{+}1$: First we show (i). Let σt be a firing sequence of Σ of length $l{+}1$. Let σ' be $proj_{T'}(\sigma)$. By the induction hypothesis, σ' is a firing sequence of Σ' and the markings generated by firing σ and σ' coincide on P', $M'_{\sigma'} = M_\sigma|_{P'}$. If t is an element of T', it follows from $M_\sigma[t\rangle$ that $M'_{\sigma'}$ enables t. By the firing rule and since M_σ and $M'_{\sigma'}$ coincide on P', it follows that also $M_{\sigma t}$ and $M'_{\sigma' t}$ coincide on P', $M_{\sigma t}|_{P'} = M'_{\sigma' t}$. If $t \in T \setminus T'$, $proj_{T'}(\sigma t) = \sigma'$, which is a firing sequence of Σ' by the induction hypothesis. A transition in $T \setminus T'$ cannot change the token count of any place $p \in P'$, thus $M_{\sigma t}|_{P'} = M_{\sigma'}|_{P'}$.

For (ii) let $\sigma' t$ be a firing sequence of Σ' with length $l{+}1$. By the induction hypothesis σ' is a firing sequence of Σ and induces the same changes on P', $M_{\sigma'}|_{P'} = M'_{\sigma'}$. Since $M'_{\sigma'}$ enables t and t has only input places in P', also $M_{\sigma'}$ enables t. Again by the firing rule, $M_{\sigma' t}|_{P'} = M'_{\sigma' t}$. □

By the next propositions, it holds that whatever maximal firing sequence the slice Σ' may fire, Σ can fire a corresponding maximal firing sequence. The converse does not hold. In the previous section we have already seen a counterexample. t_4 is not a maximal firing sequence of $slice(\Sigma_1, \{s_5\})$ in Fig. 4.2, but it is the projection of Σ_1's maximal firing sequence $t_4\, t_3\, t_3\, t_3\, t_3\, \ldots$. So for maximal firing sequences $proj_{T'}(\mathsf{Fs}_{N,max}(M_{\mathsf{init}})) \supset \mathsf{Fs}_{N',max}(M'_{\mathsf{init}})$.

Proposition 4.2.3 *Let σ'_m be a maximal firing sequence of Σ'.*

There is a maximal firing sequence σ_m of Σ that starts with σ'_m and for which $proj_{T'}(\sigma_m) = \sigma'_m$ holds.

Proof By Prop. 4.2.2 (ii), σ'_m is a firing sequence of Σ. In case σ'_m is infinite, it is also a maximal firing sequence of Σ. So let σ'_m be finite. Let σ_m be a maximal firing sequence of Σ with $\sigma_m = \sigma'_m\sigma$ where $\sigma \in T^\infty$. Let σ' be the transition sequence with $\sigma' = proj_{T'}(\sigma_m) = \sigma'_m proj_{T'}(\sigma)$. By Prop. 4.2.2 (i), σ' is a firing sequence of Σ'. Since σ'_m is maximal, it follows that $proj_{T'}(\sigma) = \varepsilon$. □

If we assume that Σ is fair w.r.t. T', we get a two way correspondence between sets of fair firing sequences of the original net and maximal firing sequences of the slice, i.e. $proj_{T'}(\mathsf{Fs}_{N,\{T'\}}(M_{\mathsf{init}})) = \mathsf{Fs}_{N',max}(M'_{\mathsf{init}})$, as the following propositions states.

Proposition 4.2.4 *Let σ' be a maximal firing sequence of Σ'.*

(i) There is a firing sequence σ of Σ that is fair w.r.t. T', starts with σ' and $proj_{T'}(\sigma) = \sigma'$.

Let σ be a firing sequence of Σ, that is fair w.r.t. T'.

(ii) $proj_{T'}(\sigma)$ is a maximal firing sequence of Σ'.

Proof We first show (i). Let σ' be a maximal firing sequence of Σ'. By Prop. 4.2.2 (ii), σ' is a firing sequence of Σ. If σ' is infinite, it is fair w.r.t. T'. So let σ' be finite. As σ' is maximal, $M'_{\sigma'}$ does not enable transitions of T'. Since $M'_{\sigma'}$ and $M_{\sigma'}$ coincide on P', $M_{\sigma'}$ does not either. Let $\sigma_2 \in (T \setminus T')^\infty$ be such that $\sigma := \sigma'\sigma_2$ is a maximal firing sequence of Σ, which exists by Prop. 4.2.3. Transitions of σ_2 cannot change the token count of places in P'. Thus σ is fair with respect to T'.

We now show (ii) by contraposition. Assume that $\sigma' := proj_{T'}(\sigma)$ is not a maximal firing sequence of Σ. By Prop. 4.2.2 (i), σ' is a firing sequence of Σ'. Since we assume that σ' is not maximal, there is a $t' \in T'$ enabled after firing σ', $M'_{\sigma'}[t'\rangle$. Let σ_{pr} be the minimal prefix of σ with $proj_{T'}(\sigma_{pr})$ equals σ'. By Prop. 4.2.2 (i), σ' and σ_{pr} generate corresponding markings,

$M_{\sigma_{pr}}|_{P'} = M'_{\sigma'}$. Hence $M_{\sigma_{pr}}$ enables t'. After firing σ_{pr}, σ does not fire any $t \in T'$ and thus the token count on P' is not changed and t' stays enabled. Hence σ is not fair with respect to T'. □

We conclude with a summary of the main results regarding the correspondence between Σ and its slice Σ':

- Corresponding firing sequences σ of Σ and σ' of Σ' generate corresponding markings $M_\sigma|_{P'} = M'_{\sigma'}$ (Prop. 4.2.2).
- $proj_{T'}(\mathsf{Fs}_N(M_{\mathsf{init}})) = \mathsf{Fs}_{N'}(M'_{\mathsf{init}})$ (Prop. 4.2.2).
- $proj_{T'}(\mathsf{Fs}_{N,max}(M_{\mathsf{init}})) \supset \mathsf{Fs}_{N',max}(M'_{\mathsf{init}})$ (Prop. 4.2.3), but in general $proj_{T'}(\mathsf{Fs}_{N,max}(M_{\mathsf{init}})) \subset \mathsf{Fs}_{N',max}(M'_{\mathsf{init}})$ does not hold.
- $proj_{T'}(\mathsf{Fs}_{N,\{T'\}}(M_{\mathsf{init}})) = \mathsf{Fs}_{N',max}(M'_{\mathsf{init}})$, (Prop. 4.2.4).

4.2.2.2 Verification and Falsification Results

In this section we present the main results concerning the preservation of CTL$^*_{-\mathsf{X}}$ properties: For Σ that is fair w.r.t. T' and a CTL$^*_{-\mathsf{X}}$ formula ψ, we can derive whether or not $\Sigma \models \psi$ by examining Σ'. For next-time, we can derive from $\Sigma' \not\models \psi$ that $\Sigma \not\models \psi$ fairly w.r.t. T' for $\forall$CTL* formulas (and hence LTL). We show its contraposition in Theorem 4.2.6.

Theorem 4.2.5 (CTL$^*_{-\mathsf{X}}$Equivalence) *Let Σ be a Petri net and $Crit \subseteq P$ a non-empty set of places. Let Σ' be slice$(\Sigma, Crit)$. Let φ be a CTL$^*_{-\mathsf{X}}$ formula with scope$(\varphi) \subseteq P'$.*

$\Sigma \models \varphi$ fairly w.r.t. $T' \Leftrightarrow \Sigma' \models \varphi$

Proof To show that Σ and Σ' satisfy the same CTL$^*_{-\mathsf{X}}$ formulas, we use Theorem 2.6.4 and show that TS_Σ and $TS_{\Sigma'}$ are stuttering bisimilar assuming that Σ is fair w.r.t T'.

In a first step we define the bisimulation relation $\mathcal{B} \subseteq [M_{\mathsf{init}}\rangle \times [M'_{\mathsf{init}}\rangle$ and then show according to Def. 2.6.2 that $\forall (M, M') \in \mathcal{B}$,

(L) $L(M) = L'(M')$

(SF1) $\forall \mu \in \Pi_{TS_\Sigma,\{T'\}}(M) : \exists \mu' \in \Pi_{TS_{\Sigma'},\mathsf{inf}}(M') : \mathsf{match}(\mathcal{B}, \mu, \mu')$

(SF2) $\forall \mu' \in \Pi_{TS_{\Sigma'},\mathsf{inf}}(M') : \exists \mu \in \Pi_{TS_{\Sigma},\{T'\}}(M) : \mathsf{match}(\mathcal{B}, \mu, \mu')$

hold.

The bisimulation relation $\mathcal{B}$ is simply defined as $(M, M') \in \mathcal{B}$ if and only if $M' = M|_{P'}$. Hence by definition $(M_{\mathsf{init}}, M'_{\mathsf{init}}) \in \mathcal{B}$.

Since we assume that $AP \subseteq P' \times \mathbb{N}$, the definition of $\mathcal{B}$ implies $L(M') = L(M)$.

We now prove that (SF1) holds. Let (M_0, M'_0) be in $\mathcal{B}$. Let μ be a path in TS_Σ from M_0 that is fair w.r.t. T'. Let σ be a fair firing sequence corresponding to μ, that is $\mu = \mathcal{M}(M_0, \sigma)$. By Prop. 4.2.4 (ii) $proj_{T'}(\sigma)$ is a maximal firing sequence of Σ' from M'_0. Hence the marking sequence $\mu' := \mathcal{M}(M'_0, proj_{T'}(\sigma))$ generated by $proj_{T'}(\sigma)$ is a path in $TS_{\Sigma'}$. Note that both μ and μ' are infinite since they are generated by maximal firing sequences.

To show that $\mathsf{match}(\mathcal{B}, \mu, \mu')$ holds, we have to show that there are partitions θ and θ' such that $\forall i \geq 0 : \forall M \in seg_{\theta,\mu}(i) : \forall M' \in seg_{\theta',\mu'}(i) : \mathcal{B}(M, M')$ holds. Let l be $|proj_{T'}(\sigma)|$, that is the number of changes in markings on P'. We partition μ and μ' so that a new segment starts with every change in the marking of P' in μ and μ', respectively. With other words, the first segment contains the initial marking and all markings with the same token count on P', a new segment starts every time a transition in T' is fired. Note, that a transition $t \in T'$ changes the token count of at least one place in P' and a transition $t \in T \setminus T'$ does not change the token count on P'. In the case that $l \neq \infty$, we define the partition such that after the last change on P' all following segments contain one marking only: for $i \geq l$ segment i on μ' contains the final marking of $proj_{T'}(\sigma)$ and segment i on μ contains the next marking—either a final marking of σ or a marking generated by firing a $t \in T \setminus T'$.

Let us first consider segment i where $0 \leq i < l+1$: A marking in $seg_{\theta,\mu}(i)$ is generated by firing a prefix of σ with i transitions in T'. A marking in $seg_{\theta',\mu'}(i)$ is generated by firing the first i transitions of $proj_{T'}(\sigma)$. Let us now consider the case that $l \neq \infty$ and $i > l$: The marking in $seg_{\theta,\mu}(i)$ is generated by firing σ or a prefix of it that contains all transitions of σ in T'. The marking in $seg_{\theta',\mu'}(i)$ is generated by $proj_{T'}(\sigma)$. In both cases it follows

by Prop. 4.2.2 that $\forall i \geq 0 : \forall M \in seg_{\theta,\mu}(i) : \forall M' \in seg_{\theta',\mu'}(i) : \mathcal{B}(M, M')$ holds.

(SF2) follows analogously. □

We now show that we can falsify via a slice Σ' that an $\forall$CTL* property using X holds on Σ. Therefore we show that if the original net fairly satisfies an $\forall$CTL* formula ψ, then also the slice satisfies ψ.

Theorem 4.2.6 (Falsification of $\forall$CTL*) *Let Σ be a Petri net and $Crit \subseteq P$ a non-empty set of places. Let Σ' be slice$(\Sigma, Crit)$. Let ψ be an $\forall$CTL* formula with scope$(\psi) \subseteq P'$.*

If $\Sigma \models \psi$ fairly w.r.t. T', then $\Sigma' \models \psi$.

Proof We show that $(TS_\Sigma, M_{\mathsf{init}})$ simulates $(TS_{\Sigma'}, M'_{\mathsf{init}})_{\{T'\}}$, which implies that if $\Sigma \models \psi$ fairly w.r.t. T' then $\Sigma' \models \psi$ (c.f. Sect. 2.6). We define the simulation relation $\mathcal{S}$ by $(M, M') \in \mathcal{S}$ if and only if $M' = M|_{P'}$. Hence $(M_{\mathsf{init}}, M'_{\mathsf{init}}) \in \mathcal{S}$. We have to show that all states M of TS_Σ and states M' of $TS_{\Sigma'}$ with $(M, M') \in \mathcal{S}$ satisfy

(L) $L(M) = L'(M')$, and

(F) $\forall \mu' \in \Pi_{TS_{\Sigma'},\mathsf{inf}}(M') : \exists \mu \in \Pi_{TS_\Sigma,\{T'\}}(M) : (\mu(i), \mu'(i)) \in \mathcal{S}$.

As we assume that $AP \subseteq P' \times \mathbb{N}$, (L) holds by definition of $\mathcal{S}$.

Let us assume that $(M_0, M'_0) \in \mathcal{S}$. Let μ' be an infinite path from M'_0 in $TS_{\Sigma'}$. Let σ' be a maximal firing sequence corresponding to μ', that is $\mu' = \mathcal{M}(M'_0, \sigma')$. By Prop. 4.2.4 (i) there is a firing sequence σ of Σ that is fair w.r.t. T' and starts with σ'. Consequently, all markings generated on Σ and Σ' during the firing of σ' coincide on P'. Let us assume that σ' is finite. Hence $M'_{\sigma'}$ is a final marking and all succeeding markings in μ' equal $M'_{\sigma'}$. Let us consider a marking M of $\mu := \mathcal{M}(M_0, \sigma)$ succeeding $M_{\sigma'}$. After firing σ', σ did not fire any transition of T'. Since only transitions in T' can change the token count on P', it follows that $M|_{P'} = M_{\sigma'}|_{P'} = M'_{\sigma'}$. □

The above result can be used for falsification: By contraposition $\Sigma' \not\models \varphi$ implies that $\Sigma \not\models \varphi$ fairly w.r.t. T'.

It is not necessary for the above result to assume that Σ is fair w.r.t. T', since if "$\Sigma \not\models \psi$ fairly w.r.t. T'" holds, then there is a firing sequence σ that does not satisfy ψ, is fair w.r.t. T' and hence also maximal. So "$\Sigma' \not\models \psi$" implies also that "$\Sigma \not\models \psi$".

Again we shortly summarise: Using next-time, we can falsify $\forall$CTL* formulas, which include LTL formulas. We can verify and falsify CTL$^*_{\text{-X}}$, if the modelled system behaves fairly w.r.t. T'. In all scenarios it suffices to examine Σ' without fairness assumptions.

4.3 Safety Slicing

In this section we will develop a more aggressive slicing algorithm. In exchange for building smaller slices, we are not able to verify every CTL$^*_{\text{-X}}$ but only (stutter-invariant) safety properties—still a very relevant class of properties.

The Ease of Slicing for Safety Properties The reason why the slicing algorithm can be more aggressive for safety properties is due to fact that satisfiability of safety properties can already be determined inspecting finite prefixes of traces of TS_Σ. A transition system satisfies a safety property $\mathcal{P}_{safe}$ iff its set of finite traces does not have a bad prefix (c.f. Prop. 2.3.8). As in the previous section we will discard parts of the net possibly exposing divergencies when building the safety slice. Thus we aim for the preservation of stutter-invariant safety-properties, i.e. safety properties build without using X. Two transition systems satisfy the same stutter-invariant safety-properties if their sets of finite paths are stutter-equivalent:

Proposition 4.3.1 *Let TS and TS_2 be two transition systems with the same set of atomic propositions, $AP = AP_2$. Let $\mathcal{P}_{safe} \subseteq (2^{AP})^\omega$ a stutter-invariant safety property.*

If $\mathit{unstutter}(\mathsf{Traces}_{TS,\mathsf{fin}}(s_{\mathsf{init}})) = \mathit{unstutter}(\mathsf{Traces}_{TS_2,\mathsf{fin}}(s_{\mathsf{init2}}))$,
then $TS, s_{\mathsf{init}} \models \mathcal{P}_{safe}$ *if and only if* $TS_2, s_{\mathsf{init2}} \models \mathcal{P}_{safe}$.

Proof We first show that $TS, s_{\mathsf{init}} \models \mathcal{P}_{safe}$ implies $TS_2, s_{\mathsf{init2}} \models \mathcal{P}_{safe}$. Let us assume that $TS, s_{\mathsf{init}} \models \mathcal{P}_{safe}$. Let ϑ_2 be a finite trace of TS_2 from s_{init2}. By assumption, TS has a stutter-equivalent trace ϑ, $\mathit{unstutter}(\vartheta) = \mathit{unstutter}(\vartheta_2)$. Since $TS, s_{\mathsf{init}} \models \mathcal{P}_{safe}$, ϑ is the prefix of an infinite trace $\vartheta\vartheta_{suf}$ that satisfies $\mathcal{P}_{safe}$. Since ϑ and ϑ_2 are stutter-equivalent, $\vartheta_2\vartheta_{suf} \models \mathcal{P}_{safe}$. This implies that $\vartheta_2 \notin \mathit{BadPref}(\mathcal{P}_{safe})$. By Prop. 2.3.8 it follows that $TS_2, s_{\mathsf{init2}} \models \mathcal{P}_{safe}$.

It follows analogously that $TS_2, s_{\mathsf{init2}} \models \mathcal{P}_{safe} \Rightarrow TS, s_{\mathsf{init}} \models \psi_{safe}$. □

Two Petri nets satisfy the same safety properties if their (sets of) finite firing sequences generate (sets of) stutter-equivalent marking sequences. Thus we have now a more relaxed notion of correctness. Whereas previously a *fair firing sequence* of the original net had to correspond to a *maximal firing sequence* of the slice and vice versa, now the slice has only to *execute* a corresponding firing sequence for every *finite firing sequence* of the original net.

Building the Safety Slice The basic idea for constructing such a *safety slice* is to build a slice for a set of places *Crit* by taking all non-reading transitions connected to *Crit* and all their input places, so that we get the exact token count on *Crit*. But for all other places we are more relaxed: We iteratively take only transitions that increase the token count on places in P' and their input places (c.f. Def. 4.3.2).

Definition 4.3.2 (safety slice, slice_S) *Let Σ be a marked Petri net and let $\mathit{Crit} \subseteq P$ be the slicing criterion. The* safety slice *of Σ for slicing criterion Crit, $\mathit{slice}_S(\Sigma, \mathit{Crit})$, is the subnet generated by the following algorithm.*

```
1   generateSafetySlice(Σ, Crit){
2     T' := {t ∈ T | ∃p ∈ Crit : W(p,t) ≠ W(t,p)} ;
3     P' := •T' ∪ Crit ;
4     P_done := Crit ;
5     while ( ∃p ∈ (P' \ P_done) ) {
6       while ( ∃t ∈ (•p \ T') : W(p,t) < W(t,p) ) {
7         P' := P' ∪ •t ;
8         T' := T' ∪ {t} ; }
9       P_done := P_done ∪ {p} ; }
10    return (P', T', W|_(P',T'), M_init|_P') ; }
```

This safety slice allows to verify and falsify linear-time stutter-invariant safety properties. We will also show that the safety slice can even be used to falsify ∀CTL* and hence LTL properties using X.

Figure 4.7 illustrates the effect of `generateSafetySlice`. Whereas $slice_S(\Sigma_2, \{s_6\})$ does not contain the transition t_4, the CTL* slice $slice(\Sigma_2, \{s6\})$ includes it, as t_4 can decrease the token count on s_7.

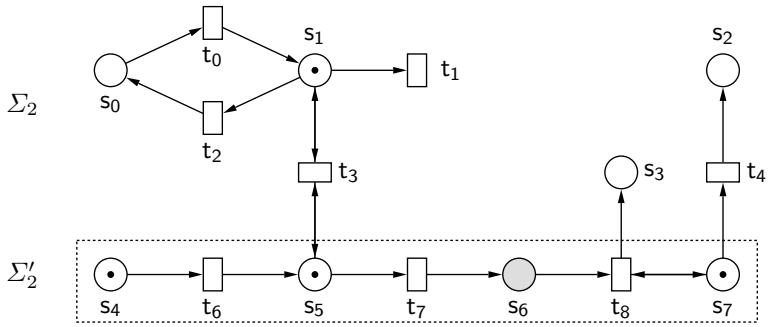


Figure 4.7: Slicing a Petri net for safety. The original net Σ_2 and its slice $\Sigma_2' = slice(\Sigma_2, \{s6\})$.

4.3.1 Proving Safety Slice's Properties

We first show that the safety slice preserves indeed stutter-invariant safety properties. We have seen that it suffices to show that the sets of finite firing sequences of Σ and Σ' generate stutter-equivalent traces.

Correspondence of Firing Sequences We first show the correspondence of firing sequences. We will show that for a given firing sequence σ of Σ we can fire the projected firing sequence $proj_{T'}(\sigma)$ on the safety slice Σ'. We can omit transitions in $T \setminus T'$, since they do not increase the token count of any place in P', so the token count on all places will be at least as high as it is on Σ firing σ. Further, every firing sequence of a safety slice Σ' is a firing sequence of Σ.

Firing Sequences, Marking Sequences, Traces We then show that corresponding firing sequences σ and σ' generate corresponding markings, $M_\sigma|_{Crit} = M'_{\sigma'}|_{Crit}$. We now consider markings M of Σ and M' of Σ' as correspondent iff they coincide on *Crit*, because we assume that $scope(\varphi) \subseteq Crit$ (not $scope(\varphi) \subseteq P'$ as before). It thus follows that two marking sequences that are stutter-equivalent w.r.t. their submarkings on *Crit* represent stutter-equivalent traces, which concludes our proof.

The second result proved is that the safety slice can be used to falsify $\forall$CTL* properties—including properties using next-time.

Note that in contrast to the previous results for $slice(\Sigma, Crit)$, we now assume that $scope(\varphi) \subseteq Crit$, since in a safety slice places in $P' \setminus Crit$ can be changed by transitions outside the slice.

Convention For the following let $Crit \subseteq P$ be a set of places and $\Sigma' = slice_S(\Sigma, Crit)$ be the safety slice of Σ. If we interpret a temporal logic formula φ on a net Σ we assume that $scope(\varphi) \subseteq Crit$.

4.3.1.1 Preservation of Safety Properties

We start with three simple observations: A transition sequence σ of Σ generates at most as many tokens on P' as its projection to T', $proj_{T'}(\sigma)$, because in Σ' a place p' is connected to all transitions $t \in T$ that can potentially increase its token count.

As W' is the restriction of W to P' and T', a transition sequence in T' has the same effect on P' in Σ and Σ'.

The effect on *Crit* of a transition sequence σ of Σ is the same as of $proj_{T'}(\sigma)$, because all transitions that may change the token count on *Crit*

are in T'. For the following equations let $\sigma \in T^\infty$ be a transition sequence of Σ and $\sigma' \in T'^\infty$ be a transition sequence of Σ'.

$$\forall p \in P' : \Delta_\Sigma(\sigma, p) \leq \Delta_{\Sigma'}(proj_{T'}(\sigma), p). \tag{4.1a}$$

$$\forall p \in P' : \Delta_\Sigma(\sigma', p) = \Delta_{\Sigma'}(\sigma', p). \tag{4.1b}$$

$$\forall p \in Crit : \Delta_\Sigma(\sigma, p) = \Delta_{\Sigma'}(proj_{T'}(\sigma), p). \tag{4.1c}$$

By the next proposition the sets of firing sequences of Σ and Σ' correspond, i.e. $\mathsf{Fs}_N(M_{\mathsf{init}}) = \mathsf{Fs}_{N'}(M'_{\mathsf{init}})$.

Proposition 4.3.3 *Let σ be a firing sequence and M be a marking of Σ.*

(i) $M_{\mathsf{init}}\,[\sigma\rangle\, M \Rightarrow \exists M' \in [M'_{\mathsf{init}}\rangle : M'_{\mathsf{init}}[proj_{T'}(\sigma)\rangle M'$ *with*

$M(p) \leq M'(p), \forall p \in P'$.

Let σ' be a firing sequence and M' a marking of Σ'.

(ii) $M'_{\mathsf{init}}[\sigma'\rangle M' \Rightarrow \exists M \in [M_{\mathsf{init}}\rangle : M' = M|_{P'} \wedge M_{\mathsf{init}}\,[\sigma'\rangle\, M$.

Proof We show Prop. 4.3.3 by induction on the length l of σ and σ', respectively. For the induction base $l = 0$ its enough to note that by Def. 4.3.2, $M'_{\mathsf{init}} = M_{\mathsf{init}}|_{P'}$.

$l \to l + 1$: First we show (i). Let σt be a firing sequence of Σ of length $l + 1$. By the induction hypothesis, $\sigma' := proj_{T'}(\sigma)$ is a firing sequence of Σ' and generates a marking $M'_{\sigma'}$ with at least as many tokens on P' as M_σ, $M_\sigma(p) \leq M'_{\sigma'}(p)$, $\forall p \in P'$. If t is an element of T', it follows from $M_\sigma[t\rangle$ that M'_σ enables t. By Eq. 4.1a, it follows that $M_{\sigma t}(p) \leq M'_{\sigma' t}(p)$, $\forall p \in P'$. If $t \in T \setminus T'$, $proj_{T'}(\sigma) = proj_{T'}(\sigma t)$ which is a firing sequence of Σ' by the induction hypothesis. A transition in $T \setminus T'$ can only decrease the token count on P', thus $M_{\sigma t}(p) \leq M_\sigma(p) \leq M'_{\sigma'}(p)$, $\forall p \in P'$.

For (ii) let $\sigma' t$ be a firing sequence of Σ' with length $l + 1$. Since $M'_{\sigma'}$ enables t and by Eq. 4.1b, also $M_{\sigma'}$ enables t and the generated markings coincide on P', $M_{\sigma' t}|_{P'} = M'_{\sigma' t}$. □

The following proposition implies in combination with Prop. 4.3.3 that the sets of finite traces of TS_Σ and $TS_{\Sigma'}$ are stutter-equivalent. It states,

that given two marking sequences μ, μ' generated by corresponding firing sequences, we can find for any finite prefix of μ' a stutter-equivalent corresponding finite prefix of μ and vice versa. As we are now assuming that $scope(\varphi) \subseteq Crit$, we restrict markings to $Crit$.

At the first glance, Prop. 4.3.4 may seem overly complicated by talking about prefixes. But note, $unstutter(\mathcal{M}(M_{\mathsf{init}}, \sigma)|_{Crit}) = unstutter(\mathcal{M}(M'_{\mathsf{init}}, \sigma')|_{Crit})$ does not necessarily hold, since either just σ or σ' may be maximal and hence one marking sequence would be finite whereas the other would be infinite.

Proposition 4.3.4 *Let $\sigma \in T^*$ be a firing sequence of Σ such that $\sigma' := proj_{T'}(\sigma)$ is a firing sequence of Σ'.*

(i) If μ is a finite prefix of $\mathcal{M}(M_{\mathsf{init}}, \sigma)$, then there is a finite prefix μ' of $\mathcal{M}(M'_{\mathsf{init}}, \sigma')$ with $unstutter(\mu|_{Crit}) = unstutter(\mu'|_{Crit})$.

Let $\sigma' \in T'^*$ be a firing sequence of Σ'.

(ii) If μ' is a finite prefix of $\mathcal{M}(M'_{\mathsf{init}}, \sigma')$, then there is a finite prefix μ of $\mathcal{M}(M_{\mathsf{init}}, \sigma')$ with $unstutter(\mu|_{Crit}) = unstutter(\mu'|_{Crit})$.

Proof We only prove (i), since (ii) follows analogously. So we show that $\mathcal{M}(M'_{\mathsf{init}}, \sigma')|_{Crit}$ starts with a stutter-equivalent version of $\mu|_{Crit}$. The proof is by induction on the length l of μ.

First note that the initial markings M_{init} and M'_{init} coincide on $Crit$ and hence for a prefix of length 1 the above holds.

$l \to l+1$: Let μM be a prefix of $\mathcal{M}(M_{\mathsf{init}}, \sigma)$ of length $l+1$. Let $\sigma_\mu t$ be the firing sequence generating μM. Let σ'_μ be the projection of σ_μ to T', $proj_{T'}(\sigma_\mu)$. By the induction hypothesis $\mathcal{M}(M'_{\mathsf{init}}, \sigma'_\mu)$ has a prefix μ' such that $\mu|_{Crit}$ and $\mu'|_{Crit}$ are stutter-equivalent. The case that $\mu|_{Crit}$ and $\mu M|_{Crit}$ are stutter-equivalent follows trivially. Otherwise, t changes the submarking on $Crit$ and hence t is an element of T'. Let M' be the marking generated by $\sigma'_\mu t$. So $\mathcal{M}(M'_{\mathsf{init}}, \sigma'_\mu t)$ has a prefix that starts with μ' and ends with M', $\mu'\mu'_2 M'$. By Eq. 4.1c, M coincides with M' on $Crit$. Since μ' reflects all changes on $Crit$ caused by σ_μ, there cannot be a change on the submarking of $Crit$ within μ'_2. So $\mu'\mu'_2 M'$ is stutter-equivalent to $\mu' M'$ and hence stutter-equivalent to μM. □

Theorem 4.3.5 (Preservation of Safety Properties) *Let Σ be a Petri net and $Crit \subseteq P$ be a set of places. Let Σ' be $slice_S(\Sigma, Crit)$ and φ a stutter-invariant linear-time safety property with $scope(\varphi) \subseteq Crit$.*

$\Sigma \models \varphi$ if and only if $\Sigma' \models \varphi$.

Proof By Prop. 4.3.1 it is sufficient to show that $unstutter(\mathsf{Traces}_{TS_\Sigma,\mathsf{fin}}(M_{\mathsf{init}})) = unstutter(\mathsf{Traces}_{TS_{\Sigma'},\mathsf{fin}}(M'_{\mathsf{init}}))$. Let ϑ be a finite trace of TS_Σ. Let σ be a corresponding firing sequence of Σ, i.e. σ corresponds to a path μ with $L(\mu) = \vartheta$. By Prop. 4.3.3, $\sigma' = proj_{T'}(\sigma)$ is also a firing sequence of Σ'. Hence it follows by Prop. 4.3.4, that there is a finite path μ' in TS'_Σ such that $\mu'|_{Crit}$ and $\mu|_{Crit}$ are stutter-equivalent. Since $scope(\varphi) \subseteq Crit$, it follows that μ' generates a trace ϑ' that is stutter-equivalent to ϑ.

Analogously follows that for a finite trace ϑ' of $TS_{\Sigma'}$ there is stutter equivalent trace ϑ of TS_Σ. □

4.3.1.2 Falsification of ∀CTL*

The small Petri net in Fig. 4.8 illustrates that a safety slice cannot be used to verify liveness properties. The slice satisfies the LTL (and CTL) liveness property $\psi = \mathsf{AF}(p_3, 1)$ which does not hold on the original net.

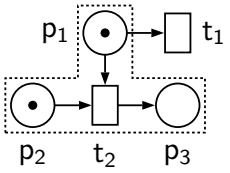


Figure 4.8: Liveness is not preserved by Safety Slicing. The safety slice for $\{p_3\}$ is depicted within dashed borders. It satisfies $\mathsf{AF}(p_3, 1)$ but the original net does not.

In the following we will show that the safety slice can be used to falsify ∀CTL* properties. The next two propositions show that a maximal firing sequence corresponds to the projection of a fair firing sequence. With these results we are ready to show that a fair TS_Σ simulates $TS_{\Sigma'}$.

Proposition 4.3.6 *Let σ'_m be a maximal finite firing sequence of Σ'.*

There is a maximal firing sequence σ_m of Σ that starts with σ'_m and for which $proj_{T'}(\sigma_m) = \sigma'_m$ holds.

Proof By Prop. 4.3.3 (ii), σ'_m is a firing sequence of Σ. Let σ_m be a maximal firing sequence of Σ with $\sigma_m = \sigma'_m\sigma$ where $\sigma \in T^\infty$. Let σ' be the transition sequence with $\sigma' = proj_{T'}(\sigma_m) = \sigma'_m proj_{T'}(\sigma)$. By Prop. 4.3.3 (i), σ' is a firing sequence of Σ'. Since σ'_m is maximal, it follows that $proj_{T'}(\sigma) = \varepsilon$. □

Proposition 4.3.7 *Let σ' be a maximal firing sequence of Σ'.*

There is a firing sequence σ of Σ, (i) that is fair w.r.t. T', (ii) that starts with σ' and (iii) for which $proj_{T'}(\sigma) = \sigma'$ holds.

Proof Let σ' be a maximal firing sequence of Σ'. By Prop. 4.3.3 (ii), σ' is a firing sequence of Σ. If σ' is infinite, it is fair w.r.t. T'. So let σ' be finite. Let $\sigma_2 \in (T \setminus T')^\infty$ be such that $\sigma = \sigma'\sigma_2$ is a maximal firing sequence of Σ, which exists by Prop. 4.3.6. As σ' is maximal, $M'_{\sigma'}$ does not enable transitions of T' and by Eq. 4.1b, $M_{\sigma'}$ does not either. Transitions of σ_2 cannot increase the token count of places in P' and hence they cannot enable transitions in T'. Consequently, σ is fair with respect to T'. □

Theorem 4.3.8 (Falsification of $\forall$CTL*) *Let Σ be a Petri net and $Crit \subseteq P$ a set of places. Let Σ' be $slice_S(\Sigma, Crit)$. Let ψ be an $\forall CTL^*$ formula with $scope(\psi) \subseteq Crit$.*

If $\Sigma \models \psi$ fairly w.r.t. T', then $\Sigma' \models \psi$.

Proof We show that TS_Σ fairly simulates $TS_{\Sigma'\{T'\}}$, which implies that if $\Sigma \models \psi$ fairly w.r.t. T', then $\Sigma' \models \psi$ holds.

We define the simulation relation $\mathcal{S}$ inspired by the construction of the fair firing sequence in Prop. 4.3.7. The pair (M, M') is in $\mathcal{S}$ if $M' = M|_{P'}$ and M' is not a final marking, but in case M' is a final marking (M, M') is in $\mathcal{S}$, if $M'|_{Crit} = M|_{Crit}$ and $M(p) \leq M'(p), \forall p \in P'$.

$(M_{\mathsf{init}}, M'_{\mathsf{init}})$ is in $\mathcal{S}$ because $M_{\mathsf{init}}|_{P'} = M'_{\mathsf{init}}$.

We show that all states M of TS_Σ and states M' of $TS_{\Sigma'}$ with $(M, M') \in \mathcal{S}$ satisfy (L) $L(M) = L'(M')$, and (F) $\forall \mu' \in \Pi_{TS_{\Sigma'},\mathsf{inf}}(M') : \exists \mu \in \Pi_{TS_\Sigma,\{T'\}}(M) : (\mu(i), \mu'(i)) \in \mathcal{S}$.

(L) holds, because we assume that $AP \subseteq \mathit{Crit} \times \mathbb{N}$ and if $(M, M') \in \mathcal{S}$, then $M|_{\mathit{Crit}} = M'|_{\mathit{Crit}}$ holds.

Let us assume that $(M_0, M_0') \in \mathcal{S}$ for two states $M_0 \in [M_{\mathsf{init}}\rangle$, $M_0' \in [M_{\mathsf{init}}'\rangle$. Let μ' be an infinite path from M_0' in $TS_{\Sigma'}$. Let σ' be the maximal firing sequence corresponding to μ', that is $\mu' = \mathcal{M}(M_0', \sigma')$.

If M_0' is a final marking, μ' is the infinite sequence $M_0'M_0'\ldots$. Since $M_0(p) \leq M_0'(p), \forall p \in P'$, M_0 does not enable any transition in T'. Since transitions in $T \setminus T'$ cannot increase the token count on P', it follows that any reachable marking M_j from M_0 satisfies $M_j(p) \leq M_0'(p)$ and this implies that $M_j|_{\mathit{Crit}} = M_0|_{\mathit{Crit}}$, as transitions in T' stay disabled. So any firing sequence from M_0 fires only transition in $T \setminus T'$ but also does not enable any transition in T' and hence is fair w.r.t. T'.

If M_0' is not a final marking, M_0 and M_0' coincide on all places in P', $M_0|_{P'} = M_0'$. By Prop. 4.3.7 there is a firing sequence σ of Σ that is fair w.r.t. $\{T'\}$ and starts with σ'. Consequently, the same markings on P' are generated by during the firing σ' on Σ and Σ'. If σ' is infinite, the pair $(\mu(i), \mu'(i))$ is hence in $\mathcal{S}$. So let us assume that σ' is finite. So $M'_{|\sigma'|}$ is a final marking and all succeeding markings in μ' equal $M'_{\sigma'}$, which brings us back to case one. □

As for Theorem 4.2.6, the slightly weaker result holds as well: $\Sigma' \not\models \varphi$ implies that $\Sigma \not\models \varphi$.

We summarise the results of this section: The safety slice can be used to verify and falsify stutter-invariant linear-time safety properties of Σ (Theorem 4.3.5). The safety slice can be used to falsify $\forall\mathrm{CTL}^*$ formulas using X, which include LTL formulas. For both results it is required that $\mathit{scope}(\varphi) \subseteq \mathit{Crit}$, whereas in Sect. 4.2.2 $\mathit{scope}(\varphi) \subseteq P'$ was required.

4.4 Related Work

The slicing and other reduction approaches are relatively old research areas and have received much attention.

In this section we highlight differences and similarities to the most relevant works.

4.4.1 Petri Net Slicing

In [16] C. K. Chang and H. Wang presented a first slicing algorithm on Petri nets for testing. For a given set of communication transitions CS, their algorithm determines the sets of paths in the Petri net graph, called concurrency sets, such that all paths within the same set should be executed concurrently to allow for the execution of all transitions in CS.

Whereas the approach of Chang and Wang does not yield a reduced net, Llorens et. al. developed an algorithm to generate a reduced Petri net [68]. They showed how to use Petri net slicing for reachability analysis and debugging presenting a forward and backward algorithm for Petri nets with maximal arc weight 1, as shown in Fig. 4.9. A forward slice is computed for all initially marked places, which makes their approach a dynamic slicing technique. They presented a second algorithm to compute a backward slice for a slicing criterion *Crit* based on our $\mathrm{CTL}^{*}_{\text{-x}}$ slicing algorithm `generateSlice` as presented in [91, 90]. Their (combined) slice is defined by $\Sigma' = (P', T', W|_{(P',T')}, M_{\mathsf{init}}|_{P'})$ with $(P', T') = \text{forwardSlice}(\Sigma) \cap \text{backwardSlice}(\Sigma, P)$.

```
1  forwardSlice(Σ){
2     T′:={t ∈ T | M_init[t⟩} ;
3     P′:= {p ∈ P | M_init(p) > 0} ∪ T′• ;
4     T_do:= {t ∈ T \ T′ | •t ⊆ P′} ;
5     while ( T_do ≠ ∅ ) {
6        P′:= P′ ∪ T•_do ;
7        T′:= T′ ∪ T_do ;
8        T_do:= {t ∈ T \ (T′) | •t ⊆ P′}
9     }
10    return (P′, T′)
11 }
```

```
1  backwardSlice(Σ, C){
2     T′:= ∅ ;
3     P′:= C ;
4     while ( •P′ ≠ T′ ) {
5        T′:= T′ ∪ •P′ ;
6        P′:= P′ ∪ •T′ ;
7     }
8     return (P′, T′)
9  }
```

Figure 4.9: Llorens' forward and backward slice according to [68]

Obviously the forward slice can also be used as a preprocessing step to model checking and removes dead transitions only. Although they defined their (combined) slice to find erroneous submarkings, their slice was considered correct iff for every firing sequence σ of the original net Σ it holds that the restriction $\sigma' = proj_{T'}(\sigma)$ can be performed on Σ' and for every place p' of the slice it holds that firing σ' generates at least as many tokens as σ. We infer that their slice allows falsification but no verification of lower bounds, and their slice allows verification and falsification of upper bounds, but no decision whether a certain submarking is reachable.

The principal difference between the `backwardSlice` of Llorens et. al. and our $\text{CTL}^*_{-\mathsf{X}}$ slicing algorithm is that `backwardSlice` includes only those transitions that increase the token count on slice places whereas $\text{CTL}^*_{-\mathsf{X}}$ slicing also includes transitions that decrease the token count. Now our *safety* slicing algorithm combines the two approaches. It uses $\text{CTL}^*_{-\mathsf{X}}$ slicing on *Crit* and a refined version of `backwardSlice` on $P' \setminus Crit$. By exploiting read arcs and considering arc weights, `line 5` in the `backwardSlice` algorithm (c.f. Fig. 4.9) can be replaced by

$T' := T' \cup \{t \mid t \in {}^\bullet P' \wedge \exists p \in P' : W(t,p) > W(p,t)\}$;

Now the backward algorithm adds new transitions only if they might produce *additional* tokens on interesting places. This principle is used in the safety slicing algorithm of Def. 4.3.2.

Let us compare the three algorithms—the algorithm of Llorens for examining bounds, our algorithm preserving $\text{CTL}^*_{-\mathsf{X}}$ properties and our algorithm preserving safety properties. The idea of forward slicing can be used for our algorithms as well. It can be seen as a preprocessing step applied before the backward slicing. The idea to use read arcs and to extend the algorithm to weighted Petri nets is also applicable to the algorithm of Llorens et al. So let us compare the algorithms for backward slicing considering the version of Llorens et al. extended for weighted Petri nets as discussed above. Our algorithm for slicing of $\text{CTL}^*_{-\mathsf{X}}$ properties is the least aggressive but most conservative algorithm, that is its slices are bigger or as big as slices generated by the other algorithms but preserves the most properties. The algorithm for slicing of safety properties is more aggressive than that

preserving $\text{CTL}^*_{\text{-x}}$ but less aggressive than the algorithm preserving bounds. The algorithm of Llorens is the most aggressive algorithm and is also the least conservative. Note, that all three variants produce the same results on strongly-connected nets.

4.4.2 Slicing for Verification

In the context of the Bandera project [7]—a project for building model checking tools for Java programs—Hattcliff et al. showed in [53, 52] that a program P and its program slice P' either both satisfy an $\text{LTL}_{\text{-x}}$ formula ψ or do not satisfy $\text{LTL}_{\text{-x}}$ formula ψ given the slicing criterion is the set of atomic propositions of ψ. Since they focus on verification of a Java program executed on a real computer, they assume that no process has to starve.

Brückner developed in his dissertation [14] a method for slicing CSP-OZ-DC specifications.

He showed that slicing preserves formulas of state/event interval logic SE-IL. To derive this result he assumed that the slice and the original system satisfy an unconditional fairness constraint that guarantees some progress within the slice.

For our approach only the original system has to satisfy a weak fairness assumption, which allows for model checking the reduced net without fairness constraints.

Clarke et. al presented a language independent slicing algorithm applied on the hardware description language VHDL [21]. They illustrated the state space reductions that can be achieved by applying slicing to some hardware circuits and planned to develop a theoretical basis for slicing w.r.t CTL specifications.

Milett and Teitelbaum discussed in [74] applications of their slicing approach for Promela, the specification language of the SPIN Model Checker [98]. They stated that slicing can be useful for model checking by helping understand the system behaviour but does not preserve global properties like deadlocks. The SPIN tool applies program slicing for so-called *selective data hiding* [48] to identify statements that can be omitted from the model. Con-

ditions statements can be mapped to `true` and other statements to the empty statement `skip`. The method preserves LTL properties. For the preservation of liveness no execution cycle can be added or skipped.

The model checker NuSMV [78] also implements slicing—or rather COI. NuSMV allows for the representation of synchronous and asynchronous finite state systems.

Slicing Petri Net Encodings The above works provide a way to slice Petri nets: In principal it is possible to encode a Petri net into a Java program. Petri nets can be encoded into Promela [42] and the PEP tool [84] already provides a mechanism to encode Petri nets into the input language of NuSMV.

We examined the slicing effects using SPIN and NuSMV. Their slicing implementations do not yield good results. SPIN's slicing algorithm [48] truncates only the very chain ends. NuSMV's COI implementation is not able to reduce a Petri net at all, because NuSMV is a tool focusing on synchronous systems [8] and the asynchronous behaviour of a Petri net cannot be adequately encoded to allow effective slicing. In both cases the slicing algorithms were neither able to use reading transitions to build the slice as in slicing algorithm of Def. 4.2.1 nor arc weight as in Def. 4.3.2.

This underlines why tailoring slicing to Petri nets is important. When using these algorithms for more powerful models, the characteristics of Petri nets cannot be exploited like e.g. reading transitions.

4.4.3 Related Approaches

In the following we compare the theoretical concepts of related approaches and our slicing approach. In Chap. 6 we will also examine these empirically.

4.4.3.1 Petri Net Reductions

Petri net slicing is a structural reduction technique (cf. Chap. 3), as slicing constructs a smaller net based on the model structure, i.e. the Petri net graph. As outlined in Chap. 3 there are only a few Petri net reductions

that preserve temporal properties. Pre- and postagglomeration are two very powerful structural reduction rules and probably also the most established [9, 85]. In the sequel we contrast the $\mathrm{LTL}_{-\mathsf{X}}$ preserving pre- and postagglomeration on the one hand and the $\mathrm{CTL}^{*}_{-\mathsf{X}}$ preserving slicing algorithm on the other. Similar aspects are relevant in a comparison between agglomerations and safety slicing.

As we have seen in Sect. 3.2.2, pre- and postagglomerations merge two transition sets $H := {}^\bullet p$ and $F := p^\bullet$ around a place p into a new one, HF. Applying these rules changes the net structure. So when model checking the reduced net a counterexample needs a translation first to be executable on the original net. Whereas slicing preserves the net structure by taking every place and transition the places in $scope(\varphi)$ causally depends on, agglomerations can also be applied in between to shorten causal dependencies. But agglomerations are not applicable in the following scenarios: (1) Transition sets $H := {}^\bullet p$ and $F := p^\bullet$ are not agglomerateable, if place p is marked. (2) Given a place p with more than one input and output transition, if any transition in $F := p^\bullet$ has an input place other than p, $H := {}^\bullet p$ is not postagglomerateable. (3) Given a place p with ${}^\bullet p = \{h\}$, $h \in T$, if h has an output place other than p, $F := p^\bullet$ is not preagglomerateable and (4) if other transitions consume tokens from the input places of h, h is not agglomerateable at all. It is easy to build a net that exposes a lot of these constructs but is nicely sliceable for a given property.

An important conceptual difference is that agglomerations are not able to eliminate infinite behaviour, as follows from the proof in [85]. Slicing may eliminate infinite behaviour, which allows additional reductions and necessitates the fairness assumption to preserve liveness properties.

4.4.3.2 Partial Order Reductions

Partial order reductions have already been introduced in Sect. 3.3.1. The key idea of partial order reductions is to reduce the state space by taking one interleaving as representative of a set of interleavings that represent the same concurrent behaviour. Partial order methods reduce the state space whereas

slicing reduces the model description. But both approaches use a notion of independence as basis of their reductions. Our first slicing algorithm for instance includes all places and transitions the places in $scope(\varphi)$ causally dependent on. Partial order methods aim for an efficient way to structurally define a sufficient condition to guarantee dynamic independence, i.e. independence of transitions in a state. Therefrom usually arises a greater degree of independence. Slicing on the other hand has to capture dependencies for every (structurally) possible behaviour. Slicing trims causal dependencies, so that some transitions do not appear at all in the reduced state space. Only by trimming causal dependencies, slicing also reduces the concurrent behaviours. Partial order reductions do not aim to trim causal dependencies. Valmari states in [102], that every Petri net transition has an occurrence as (label of) a state transition in a state space reduced by stubborn sets to preserve safety properties. So slicing can complement partial order reductions.

4.4.3.3 Summary

Our slicing approach conceptually complements partial order reductions and pre- and postagglomeration. All three methods may reduce concurrent behaviours. Pre- and postagglomerations as well as slicing pare causal dependencies. It has been highlighted that also their impact may be complementary. An empirical study on these aspects is presented in Chap. 6.

4.5 Future Work

We presented two flavours of slicing, $\mathrm{CTL}^*_{-\mathsf{X}}$ slicing and safety slicing. Whereas $\mathrm{CTL}^*_{-\mathsf{X}}$ slicing preserves all properties expressible as $\mathrm{CTL}^*_{-\mathsf{X}}$ formulas, safety slicing allows greater reductions but preserves stutter-invariant safety properties only. Like we have done for safety properties, it seems worthwhile to develop refined slicing algorithms for certain (classes of) properties. A good starting point seems antecedent slicing [108, 109], a form of conditional slicing where information about system input is encoded as antecedent of an LTL formula. If we study a formula of the form $\psi := \mathsf{G}\,(\varphi_1 \Rightarrow \mathsf{F}\,\varphi_2)$, we only

need to include transitions that make the antecedent φ_1 true, we do not need to include transitions that are fired when φ_1 cannot become true [109]. We conjecture that in this setting a safety slicing like algorithm can be used for the antecedent places, $scope(\varphi_1)$, whereas $\text{CTL}^*_{\text{-X}}$ slicing has to be applied to places in $scope(\varphi_2)$. Since both, safety slicing and $\text{CTL}^*_{\text{-X}}$ slicing, are not able to reduce strongly connected nets, it seems worthwhile to explore whether the antecedent can be used to eliminate transitions when their firing implies that the antecedent cannot become true.

4.6 Conclusions

In this chapter we introduced two slicing algorithms to reduce the size of a Petri net in order to alleviate the state space explosion problem for model checking Petri nets. We formally proved that $\text{CTL}^*_{\text{-X}}$ slicing allows falsification and verification of a CTL^* formula given $scope(\psi)$ *refers to the slice places* only and if we can assume fairness w.r.t. T' for the original net. We also showed that $\forall\text{CTL}^*$ formulas can be falsified by $slice(\Sigma, Crit)$. Safety slicing promises greater reductions but sacrifices the preservation of liveness properties. A safety slice $slice_S(\Sigma, Crit)$ satisfies the same stutter-invariant safety properties φ as the original net, given that φ *refers to the places of the slicing criterion Crit* only. A safety slice can also be used to falsify $\forall\text{CTL}^*$ formulas using next-time.

We outlined related work, in particular slicing approaches, and discussed competitive (and complementary) approaches. An empirical evaluation of the two slicing algorithms will be given in Chap. 6.

Chapter 5

Cutvertex Reductions

Contents

5.1 Introduction

Compositional methods try to bypass the combinatorial blow-up of the state space by examining the system componentwise. *Compositional reduction* techniques generate a reduced state space for each component and compose the component state spaces to a global reduced state space, on which the verification task is performed. *Compositional verification* methods divide the verification task into local verification tasks on the system components. The global result is then derived from the results of the local verifications. At both approaches a similar problem arises: A component as part of a global system is constrained by its surrounding environment. When a component is examined in isolation, usually an overapproximation of the environment is assumed and hence the component may expose behaviour it does not have within the global system. For compositional reduction techniques this means that the state space of a single component may in isolation be bigger than the state space of the overall system; for compositional verification techniques, the spurious behaviour may imply false local verification results.

In this chapter we present a decomposition approach for monolithic Petri nets that generates very narrow interfaces to the environment and therefore reduces the risk of spurious behaviour. For a given LTL$_{-\mathsf{X}}$ property we decompose a net Σ into a kernel net Σ_{k} and environment nets $\Sigma_{\mathrm{e}_1}, ..., \Sigma_{\mathrm{e}_n}$ such that the kernel subnet contains all places mentioned by the LTL$_{-\mathsf{X}}$ property

φ and shares with an environment net a single 1-safe place q only. If the set

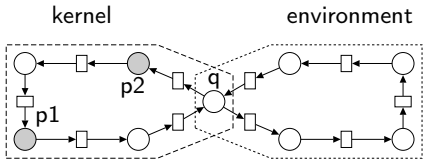


Figure 5.1: Decomposition into kernel and environment.

of 1-safe places of Σ is given, the decomposition of a monolithic Petri net can be determined in linear time. Hence we can decompose 1-safe nets in linear time.

Based on this decomposition, we minimise the environments. Every environment net Σ_{e_i} is replaced by a small summary net $S(\Sigma_{e_i})$, which captures Σ_{e_i}'s influence on the kernel. Five fixed and distinct summary nets of at most four nodes (=places+transitions) suffice to describe the influence of any environment net. To determine the appropriate summary net $S(\Sigma_{e_i})$, Σ_{e_i} is model checked independently to characterise its influence on the kernel. For this up to three out of five local and fixed LTL$_{-X}$ properties are checked on Σ_{e_i}.

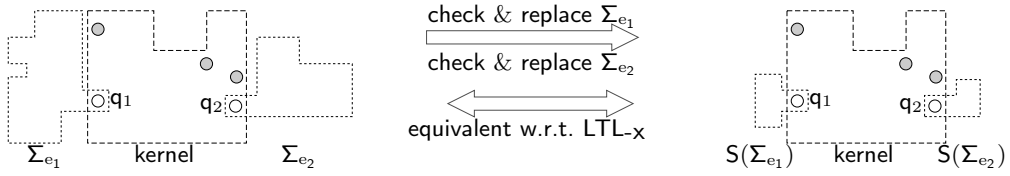


Figure 5.2: Replacement of environment nets.

We establish whether φ holds on Σ by model checking whether φ holds on the reduced net. The characterisation of environment nets leads straight forwardly to structural reduction rules for the smallest environment nets and to structural optimisations to accelerate the identification of the appropriate summary net.

We further point out criteria when spurious behaviour is possible and how it can be identified early on.

Outline In Sect. 5.2 we introduce six reduction rules that define how Σ_{e_i} in Σ is replaced by the summary net $S(\Sigma_{e_i})$. Provided we can assume fairness w.r.t. T_k (the kernel's transitions) on Σ, all reductions guarantee that the reduced net satisfies an LTL$_{\text{-X}}$ property φ if and only if Σ satisfies φ. For some reduction rules even stronger results haven been shown. The detailed correctness results are given together with their proofs in Sect. 5.3. We will show in Sect. 5.4 that our set of reductions is sufficient to reduce any environment and that five of six reductions are necessary to reduce any environment. The *Dead End* rule is not necessary—though it is useful. Section 5.5 illustrates a decomposition algorithm and discusses the computational expense of determining a decomposition. In Sect. 5.6 we discuss costs and benefits for cutvertex reductions. We present optimisations in Sect. 5.7. Before we conclude in Sect. 5.10, we survey related work in Sect. 5.8 and outline ideas for future work in Sect. 5.9.

5.2 The Reduction Rules

In the following we show how to reduce a net Σ composed of a kernel net Σ_k and an environment net Σ_e. An algorithm to determine an appropriate decomposition is presented in Sect. 5.5.

All of our reductions preserve LTL$_{\text{-X}}$ properties and some reductions even CTL$^*_{\text{-X}}$. Therefore the following definitions refer more generally to CTL$^*_{\text{-X}}$ formulas.

Reducible Nets Any subnet Σ_e of Σ that is free of places the considered property refers to and shares just a 1-safe place q with the remainder is reducible by our approach. We will show how such an environment net can be summarised by a simple net $S(\Sigma_e)$ and that the environment net can be examined independently to determine its summary $S(\Sigma_e)$. Before we show how to reduce a Σ that is composed of Σ_k and Σ_e, we formally define how these two nets compose Σ. Recall that $\mathit{scope}(\varphi)$ denotes the set of places referred to by a temporal logic formula φ.

Definition 5.2.1 (reducible, kernel, environment, contact place) *Let Σ be a marked Petri net and φ be an CTL$^*_{\text{-X}}$ formula.*

Σ *is* reducible *for φ by N_e iff*
there is a 1-safe place $q \in P$ and a subnet N_k, such that

- $N = (P_\mathrm{k} \uplus (P_\mathrm{e} \setminus \{q\}), T_\mathrm{k} \uplus T_\mathrm{e}, W|_{(P_\mathrm{k},T_\mathrm{k})} \uplus W|_{(P_\mathrm{e},T_\mathrm{e})})$,
- $\mathit{scope}(\varphi) \subseteq (P_\mathrm{k} \setminus \{q\})$ *and*
- $q \in P_\mathrm{k} \cap P_\mathrm{e}$.

Σ *is* reducible by $\Sigma_\mathrm{e} = (N_\mathrm{e}, M_{\mathsf{init}}|_{P_\mathrm{e}})$ *iff Σ is reducible by N_e. We call $\Sigma_\mathrm{k} = (N_\mathrm{k}, M_{\mathsf{init}}|_{P_\mathrm{k}})$ the* kernel *subnet and Σ_e the* environment *subnet. A place q is the* contact place *of kernel and environment, if q is the single common place of kernel and environment.*

So Σ is reducible by an environment net N_e iff Σ is composed of an environment net N_e and a kernel N_k, such that (i) φ does not refer to the environment N_e and (ii) kernel N_k and environment N_e have only a 1-safe place q in common, so that the transitions of kernel and environment are disjoint and they have neither input- nor output places in the other net with exception of q. If we would remove q (and connected arcs), N_e and N_k will not be connected anymore. Figure 5.1 shows an example of a reducible net that is decomposed into kernel and environment subnet for $\mathit{scope}(\varphi) = \{p1, p2\}$.

Convention In the sequel let φ be an CTL$^*_{\text{-X}}$ formula, let $\Sigma_\mathrm{k} = (N_\mathrm{k}, M_{\mathsf{init}}|_{P_\mathrm{k}})$ be the kernel and $\Sigma_\mathrm{e} = (N_\mathrm{e}, M_{\mathsf{init}}|_{P_\mathrm{e}})$ be the environment subnet of a net Σ, such that Σ is reducible by Σ_e according to Def. 5.2.1. Let q be the contact place shared by Σ_k and Σ_e.

Next we first intuitively, then formally introduce the six reduction rules to reduce Σ by Σ_e. The reductions are applied to example nets in Fig. 5.3.

We assume that Σ is fair w.r.t. T_k. This guarantees progress on Σ_k and is a prerequisite for preserving liveness properties and hence for preserving LTL$_{\text{-X}}$ or CTL$_{\text{-X}}$ as we have demonstrated in Sect. 4.2.1 for the slicing approach. To characterise how Σ_e may affect the given property φ we study Σ_e's effect on the 1-safe place q at the two scenarios, Σ_e *with a token on q* and Σ_e *without a token on q*.

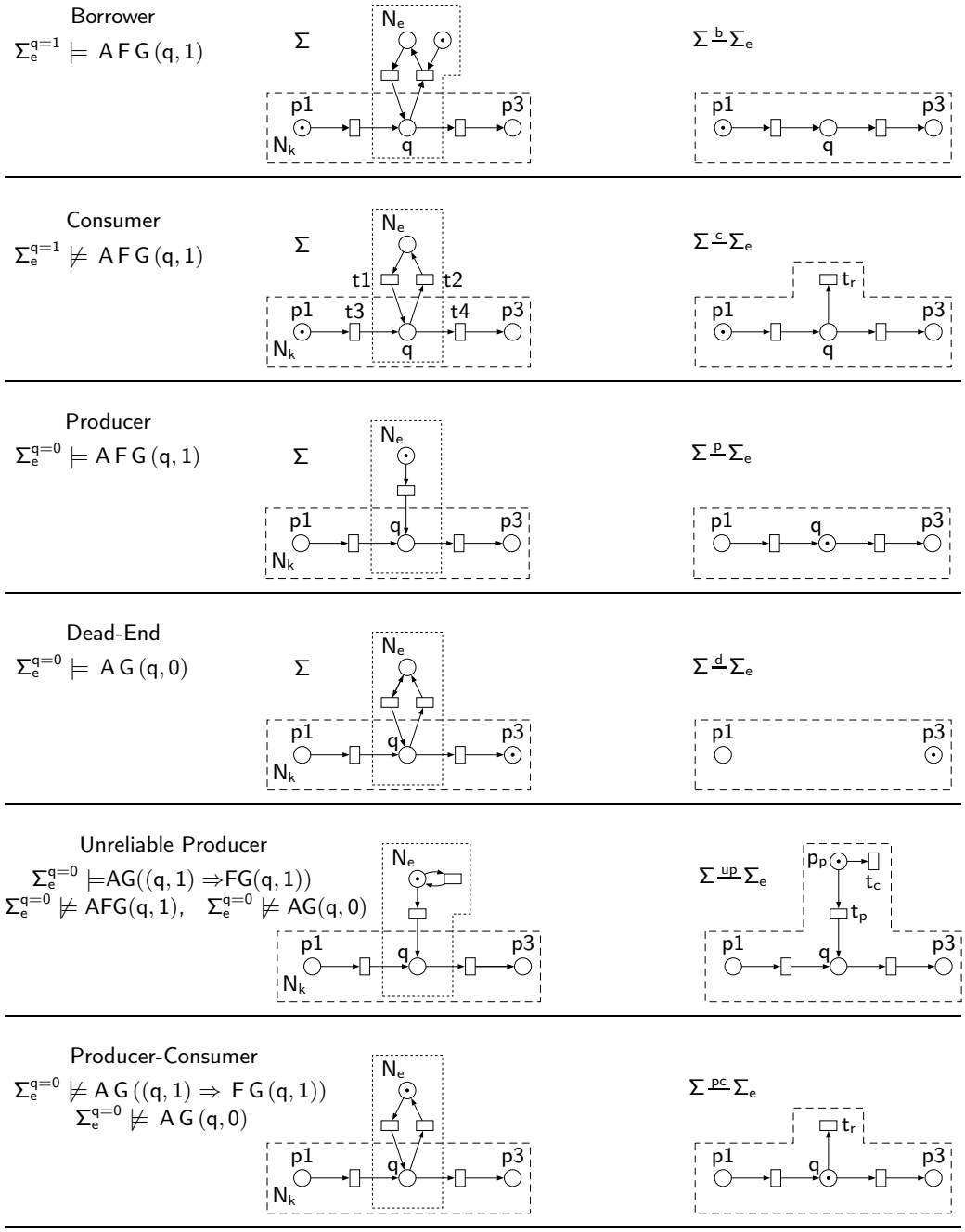


Figure 5.3: The reductions

Notation We denote Σ_e with a token on q as ${\Sigma_e}^{q=1} = (N_e, M_{\mathsf{init}}^{q=1}|_{P_e})$, and Σ_e without a token on q is denoted as ${\Sigma_e}^{q=0} = (N_e, M_{\mathsf{init}}^{q=0}|_{P_e})$.

Environments and Reductions An environment subnet Σ_e is called a *Borrower* if it may take a token from q—one or several times—but eventually permanently marks q. In other words, given q gets a token, Σ_e can only temporarily borrow the token. As we study stuttering-invariant properties, which do not count execution steps [65], Borrower subnets can be omitted without changing the behaviour on the kernel.

An environment subnet Σ_e is a *Consumer*, if Σ_e may not return the token from q, i.e. ${\Sigma_e}^{q=1}$ has at least one execution that does not eventually *permanently* mark q. Due to our weak fairness notion, progress in Σ_k is only guaranteed, if a transition is eventually permanently enabled, i.e. its preset is permanently (sufficiently) marked. So "permanently borrowing", i.e. taking without eventually returning the token permanently, is considered equivalent to (permanently) removing the token. For an example of a Consumer net that can permanently borrow see Fig. 5.3. The Consumer environment with a token on q, ${\Sigma_e}^{q=1}$, does not eventually permanently mark q. So after firing t_3 on Σ, t_4 might never be fired, since the token may get lost in Σ_e by firing infinitely often $t_2 t_1$. Therefore a Consumer net can be replaced by just one transition that may remove the token from q, just like the Consumer may remove the token from q or keep the token for ever.

Σ_e is called a *Producer* environment, if ${\Sigma_e}^{q=0}$ eventually permanently marks the initially unmarked q. In case of a Producer environment, it is enough to place a token on q, as stuttering-invariant properties do not count the number of steps to generate the token.

We apply a *Dead End* reduction, if the place q is never marked in Σ. In case of a Dead End environment we can omit Σ_e and also the transitions of Σ_k that are connected to it. Transitions in ${}^\bullet q$ are never fired because otherwise q would be marked and since q is never marked, transitions in $q^\bullet$ are never enabled. The Dead End reduction is not necessary to be able to reduce environments, as we will see in Sect. 5.4. But as the Dead End reduction usually indicates a design error within the net—there usually is

no reason to include dead transitions—it is a useful reduction rule to have. Note also, that the Dead End reduction is the only reduction changing the kernel as well.

Σ_e is an *Unreliable Producer*, if ${\Sigma_\mathrm{e}}^{q=0}$ eventually permanently marks q at some executions and never marks q at the others. An Unreliable Producer subnet is replaced by a net that can do the same, i.e. produce a token on q or never mark q.

An environment subnet Σ_e is called a *Producer-Consumer*, if some executions of ${\Sigma_\mathrm{e}}^{q=0}$ generate a token on q but do not eventually permanently mark q.

We now formally define the reduction rules motivated above.

Definition 5.2.2 (Reduction Rules) *Let Σ be reducible by an environment Σ_e for a CTL^*_{-X} formula φ. Let $\Sigma_\mathrm{k} = (N_k, M_{\mathsf{init},\mathrm{k}})$ be the kernel and q be the 1-safe contact place, $q \in (P_\mathrm{k} \cap P_\mathrm{e})$.*

Σ_e *is a* Borrower
iff q is a 1-safe place of ${\Sigma_\mathrm{e}}^{q=1}$ and ${\Sigma_\mathrm{e}}^{q=1} \models \mathsf{AFG}(q, 1)$.
The Borrower-reduced *of Σ by Σ_e, $\Sigma \underline{b} \Sigma_\mathrm{e}$, is the net Σ_k.*

Σ_e *is a* Consumer
iff q is a 1-safe place of ${\Sigma_\mathrm{e}}^{q=1}$ and ${\Sigma_\mathrm{e}}^{q=1} \not\models \mathsf{AFG}(q, 1)$.
The Consumer-reduced *of Σ by Σ_e, $\Sigma \underline{c} \Sigma_\mathrm{e}$, is the net $(P_\mathrm{k}, T_\mathrm{k} \uplus \{t_r\}, W_k \uplus \{(q, t_r) \mapsto 1\}, M_{\mathsf{init},\mathrm{k}})$.*

Σ_e *is a* Dead End
iff q is not 1-safe in ${\Sigma_\mathrm{e}}^{q=1}$ and ${\Sigma_\mathrm{e}}^{q=0} \models \mathsf{AG}(q, 0)$.
The Dead End-reduced *of Σ by Σ_e, $\Sigma \underline{d} \Sigma_\mathrm{e}$, is $(P', T', W|_{(P',T')}, M_{\mathsf{init},\mathrm{k}}|_{P'})$ with $P' = P_\mathrm{k} \setminus \{q\}$ and $T' = T_\mathrm{k} \setminus ({}^\bullet q \cup q^\bullet)$.*

Σ_e *is a* Producer
iff ${\Sigma_\mathrm{e}}^{q=0} \models \mathsf{AFG}(q, 1)$.
The Producer-reduced *of Σ by Σ_e, $\Sigma \underline{p} \Sigma_\mathrm{e}$, is $(P_\mathrm{k}, T_\mathrm{k}, W_k, M^{q=1}_{\mathsf{init},\mathrm{k}})$.*

Σ_e *is an* Unreliable Producer
iff ${\Sigma_\mathrm{e}}^{q=0} \not\models \mathsf{AG}(q,0)$, ${\Sigma_\mathrm{e}}^{q=0} \not\models \mathsf{AFG}(q,1)$ *and* ${\Sigma_\mathrm{e}}^{q=0} \models \mathsf{AG}((q,1) \Rightarrow \mathsf{FG}(q,1))$.

The Unreliable Producer-reduced *of* Σ *by* Σ_e, $\Sigma \underline{^{up}} \Sigma_\mathrm{e}$, *is the net* $\Sigma \underline{^{up}} \Sigma_\mathrm{e} = (P_\mathrm{k} \uplus \{p_p\}, T_\mathrm{k} \uplus \{t_c, t_p\}, W_k \uplus \{(p_p, t_p) \mapsto 1, (t_p, q) \mapsto 1,), (p_p, t_c) \mapsto 1\}, M_{\mathsf{init},\mathrm{k}} \uplus \{p_p \mapsto 1\})$.

Σ_e *is a* Producer-Consumer
iff ${\Sigma_\mathrm{e}}^{q=0} \not\models \mathsf{AG}(q,0)$ *and* ${\Sigma_\mathrm{e}}^{q=0} \not\models \mathsf{AG}((q,1) \Rightarrow \mathsf{FG}(q,1))$.

The Producer-Consumer-reduced *of* Σ *by* Σ_e, $\Sigma \underline{^{pc}} \Sigma_\mathrm{e}$, *is the net* $\Sigma \underline{^{pc}} \Sigma_\mathrm{e} = (P_\mathrm{k}, T_\mathrm{k} \uplus \{t_r\}, W_k \uplus \{(q, t_r) \mapsto 1\}, M^{q=1}_{\mathsf{init},\mathrm{k}})$.

Each of these reduction rules preserves LTL$_{-\mathsf{X}}$, i.e., if φ does not refer to the environment, Σ satisfies an LTL$_{-\mathsf{X}}$ property φ fairly w.r.t. T_k if and only if its reduced net Σ' satisfies φ. For some reduction rules even stronger results hold, as will be shown in the following section.

Figure 5.4 illustrates how the appropriate reduction rule to replace an environment net Σ_e can be determined.

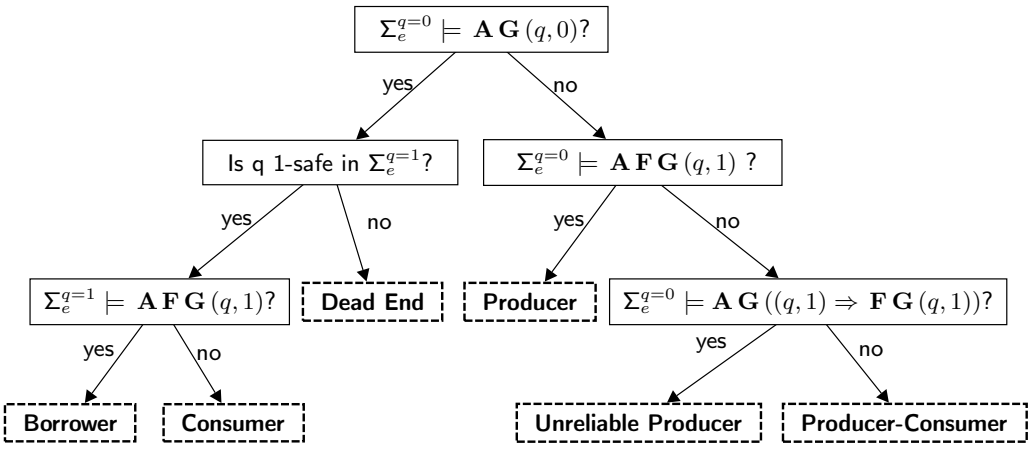


Figure 5.4: Decision tree with rule preconditions. Leafs of the decision tree classify Σ_e.

1-Safeness and Spurious Behaviour To identify the appropriate summary $S(\Sigma_\mathrm{e})$ for an environment Σ_e we examine Σ_e at the two scenarios, Σ_e with a token on q and Σ_e without a token on q. If q is never marked within

Σ, placing a token on q might enable spurious behaviour. We hence only risk to encounter spurious behaviour for the non-producing environments. ${\Sigma_e}^{q=1}$ may even become unbounded, whereas Σ_e within Σ is bounded. During the evaluation of this method we never encountered such a case, though it is theoretically possible.

To avoid examining a possibly enlarged state space of ${\Sigma_e}^{q=1}$, additional knowledge about Σ can be used to identify spurious behaviour early on. For instance knowing that place p has bound $\mathsf{b}(p)$, spurious behaviour is encountered if ${\Sigma_e}^{q=1}$ has a marking with more than $\mathsf{b}(p)$ tokens on p. In this case we can apply the Dead End reduction. Also, if earlier simulation showed that q can be marked, Σ_e is consequently not a Dead End. So one could only use contact places that are known to get marked, to prevent the risk of encountering spurious behaviour.

Nevertheless, all reductions guarantee that q remains 1-safe in the reduced net. And the state space of a reduced net is never bigger than the state space of the original net.

The Dead End rule is not necessary to preserve LTL$_{-\mathsf{X}}$. As we will see in Sect. 5.3.5, we could as well replace a Dead End environment by a Borrower or Consumer summary. But it is convenient to single out Dead Ends, since a Dead End usually indicates a design error, as it implies that q in Σ is never marked and the transitions in ${}^\bullet q \cup q^\bullet$ are dead.

Summary To summarise, so far we have defined when a net Σ is reducible and six reduction rules have been introduced to replace environments Σ_e by their summaries $S(\Sigma_e)$. We discussed scenarios where cutvertex reductions cause an overhead and sketched countermeasures.

5.3 Preservation of Temporal Properties

In this section we will examine in detail which temporal properties are preserved by the reduction rules. All reductions preserve LTL$_{-\mathsf{X}}$ and as we will see the Borrower, Producer and Dead End reductions preserve even CTL$^*_{-\mathsf{X}}$.

5.3.1 Outline and Common Results

The reduction rule proofs follow the same general outline as the slicing proofs: To proof that the reduced net Σ' preserves satisfiability of a $\mathrm{CTL}^*_{\text{-}\mathsf{X}}$ or $\mathrm{LTL}_{\text{-}\mathsf{X}}$ formula φ, we first show that any firing sequence of Σ that is fair w.r.t. T_k corresponds to a maximal firing sequence of the reduced net Σ'. Also, any maximal firing sequence σ' of Σ' corresponds to a firing sequence of Σ that is fair w.r.t. T_k.[1] We show that corresponding firing sequences generate corresponding markings.

Firing sequences σ of Σ and σ' of Σ' correspond if σ and σ' fire the same transitions of T_k in the same order. Two markings M of Σ and M' of Σ' correspond if they coincide on the places in $P_k \setminus \{q\}$, $M|_{P_k \setminus \{q\}} = M'|_{P_k \setminus \{q\}}$.

To show that a reduction preserves $\mathrm{CTL}^*_{\text{-}\mathsf{X}}$ properties, a bisimulation relation is defined based on the correspondence of firing sequences on Σ and Σ'. To show that $\mathrm{LTL}_{\text{-}\mathsf{X}}$ properties are preserved we show that marking sequences of corresponding firing sequences satisfy the same $\mathrm{LTL}_{\text{-}\mathsf{X}}$ formulas.

Some reduction rules also allow falsification of $\forall\mathrm{CTL}^*$ formulas using X. Therefore we need a stronger, that is stepwise, correspondence between firing sequences. We construct for every firing sequence σ' of Σ' a corresponding fair firing sequence σ of Σ such that σ mimics σ' step by step. So when σ' fires a transition in T_k, σ fires a transition in T_k, and when σ' fires a transition in $T' \setminus T_k$, σ fires a transition in T_e. We then show that Σ fairly simulates Σ'.

We prove every reduction rule separately, though we make use of earlier results established at other reductions. Since any two rules have distinct preconditions, we strengthen our assumptions step by step to finally match the full precondition. In a rule's proof, we highlight the assumptions in framed boxes heading the inferred results.

Convention As in the previous sections we denote the original net as $\Sigma = (N, M_{\mathsf{init}})$ and assume that Σ is reducible by an environment net $\Sigma_{\mathrm{e}} = (N_e, M_{\mathsf{init}}|_{P_e})$ and further that Σ_{e} shares only the place q with the kernel $\Sigma_{\mathrm{k}} = (P_k, M_{\mathsf{init}}|_{P_k})$. Σ' refers to the reduced net.

[1] For some rules we can even show that every σ' has a corresponding firing sequence σ that is fair w.r.t. T_k and T_e.

Fairness Rules out Divergencies As discussed in Sect. 4.2.1 for slicing, we also use here fairness to rule out divergencies outside of the kernel. This might seem counterintuitive, as the reduction rules characterise the behaviour of the environment. But the rules' precondition only constrain behaviour w.r.t. the contact place. Let us consider the example of a Borrower reduction in Fig. 5.5. The place q is 1-safe in Σ and ${\Sigma_e}^{q=1}$, and ${\Sigma_e}^{q=1} \models \mathsf{AFG}(q, 1)$, so Σ_e is a Borrower indeed. The reduced net satisfies the LTL$_{\text{-X}}$ property $\varphi = \mathsf{AF}(p3, 1)$ but the original net does not, because after firing t_1 the transition t_ω could be fired infinitely often retaining the token on q. Fairness w.r.t T_k rules out $\sigma = t_1 t_\omega t_\omega ...$ as being unfair, and $\Sigma \models \varphi$ fairly w.r.t. $\{T_k\}$ holds.

Figure 5.5: $proj_{T_k}(\mathsf{Fs}_{N,\mathsf{max}}(M_{\mathsf{init}})) \not\subseteq \mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}})$. $\sigma = t_1 t_\omega t_\omega ...$ is a maximal firing sequence of Σ but $proj_{T_k}(\sigma) = t_1$ is not maximal on $\Sigma \triangleq \Sigma_e$.

5.3.1.1 Common Results

We now give some results that are valid for all six reductions and will be used throughout the rest of this section. Most of the results presented here are straightforward but nevertheless necessary for the formal proof.

The token count on $P_k \setminus \{q\}$ is only affected by transitions in T_k (Eq. 5.1a) and analogously the token count on $P_e \setminus \{q\}$ is only affected by transitions in T_e (Eq. 5.1b). For the following equations let $\sigma \in T^*$ and let $\sigma' \in T'^*$ be a transition sequence with $proj_{T_k}(\sigma) = proj_{T_k}(\sigma')$.

$$\forall p \in (P_k \setminus \{q\}) : \quad \Delta(\sigma, p) = \Delta(proj_{T_k}(\sigma), p) = \Delta(\sigma', p). \tag{5.1a}$$

$$\forall p \in (P_e \setminus \{q\}) : \quad \Delta(\sigma, p) = \Delta(proj_{T_e}(\sigma), p). \tag{5.1b}$$

The Eq. 5.1a holds, since $proj_{T_k}(\sigma)$ omits transitions of σ in T_e only and these do not have input or output places in $P_k \setminus \{q\}$. As the newly introduced

transitions of σ' in $T' \setminus T_k$ do not have input or output places in $P_k \setminus \{q\}$ (q is the only contact), σ' affects the token count on places in $P_k \setminus \{q\}$ in the same way as $proj_{T_k}(\sigma)$. Equation 5.1b follows analogously.

If we have two subnets Σ_1 and Σ_2 of Σ with only one place in common, it depends on the temporal logic formula φ which one will be called kernel. The decomposition itself is symmetrical. The next proposition abstracts now from the roles of kernel and environment, and studies only the two subnets.

A firing sequence from marking M on the original net Σ firing only transitions of one subnet Σ_i, $i \in \{e, k\}$, is also a firing sequence of Σ_i from marking $M|_{P_i}$. Also, a firing sequence from M_i on subnet Σ_i is also a firing sequence of Σ from any marking M that coincides with M_i on P_i.

Proposition 5.3.1 *Let $\tilde{\Sigma}$ be either Σ_e or Σ_k. Let $\tilde{\sigma} \in \tilde{T}^\infty$ be a transition sequence. Let $M \in \mathbb{N}^{|P|}$ be a marking of Σ, and $\tilde{M} \in \mathbb{N}^{|\tilde{P}|}$ a marking of $\tilde{\Sigma}$.*

(i) If $M[\tilde{\sigma}\rangle_\Sigma$ and $M|_{\tilde{P}} = \tilde{M}$, then $\tilde{M}[\tilde{\sigma}\rangle_{\tilde{\Sigma}}$, and

(ii) If $\tilde{M}[\tilde{\sigma}\rangle_{\tilde{\Sigma}}$ and $M|_{\tilde{P}} = \tilde{M}$, then $M\,[\tilde{\sigma}\rangle_\Sigma$.

Proof Every $t \in \tilde{T}$ has the same input and output places in Σ and $\tilde{\Sigma}$. Thus a transition sequence $\tilde{\sigma} \in \tilde{T}^\infty$ is either a firing sequence of both Σ and $\tilde{\Sigma}$ or cannot be fired on either of them. □

As direct consequence follows that a k-bounded place in Σ is also k-bounded in kernel and environment subnets.

Proposition 5.3.2 *Let $\tilde{\Sigma}$ be either Σ_e or Σ_k.*

If p is a k-bounded place of Σ, then p is a k-bounded place of $\tilde{\Sigma}$ and of $\tilde{\Sigma}^{q=0}$.

Proof Suppose p is not k-bounded in $\tilde{\Sigma}$. Hence there is a firing sequence $\tilde{\sigma}$ with $M_{\mathsf{init}}|_{\tilde{P}}[\tilde{\sigma}\rangle\tilde{M}$ and $\tilde{M}(q) > k$. By Prop. 5.3.1, $\tilde{\sigma}$ is a firing sequence of Σ. But then p is not k-bounded in Σ. Analogously follows that p is k-bounded in $\tilde{\Sigma}^{q=0}$, since if $\tilde{\sigma}$ is a firing sequence of $\tilde{\Sigma}^{q=0}$, then also of Σ. □

The next three propositions are auxiliaries used for each reduction rule. As outlined, we show that (i) every maximal firing sequence of reduced net

Σ' corresponds to a fair firing sequence of the original net Σ, and (ii) every fair firing sequence of Σ corresponds to a maximal firing sequence of Σ'. To prove that this correspondence holds, we use that the summary net $S(\Sigma_e)$ as defined by our reductions equivalently captures the effect of maximal firing sequences of Σ_e on q.

For (i) we show that a maximal firing sequence of σ' can be executed on Σ by emulating the reduction's effect by behaviour of Σ_e. For (ii) we show that a (certain) fair firing sequence σ of Σ contains a maximal firing sequence σ^e of Σ^e and the effect of σ^e is emulated by the reduction. These proofs are done by contradiction. They assume that a maximal firing sequence does not correspond to a fair firing sequence and then derive a contradiction. Of course the behaviour is only equivalent w.r.t. the interface place q and the contradiction proofs need only to refer to this behaviour. Before we can argue about maximality on Σ^e, we first have to show that the projection of a firing sequence σ of Σ to T_e is a firing sequence of Σ.

Proposition 5.3.3 *Let $\tilde{\Sigma}$ be either ${\Sigma_e}^{q=1}$, ${\Sigma_e}^{q=0}$, $\Sigma_k^{q=0}$ or $\Sigma_k^{q=1}$.*

If σ is a firing sequence of Σ but $\tilde{\sigma} = proj_{\tilde{T}}(\sigma)$ is not a firing sequence of $\tilde{\Sigma}$, then there are prefixes σ_p of σ and $\tilde{\sigma}_p$ of $\tilde{\sigma}$ such that $proj_{\tilde{T}}(\sigma_p) = \tilde{\sigma}_p$ and $M_{\sigma_p}(q) > \tilde{M}_{\tilde{\sigma}_p}(q)$ holds.

Proof If σ is a firing sequence of Σ but $\tilde{\sigma} = proj_{\tilde{T}}(\sigma)$ is not a firing sequence of $\tilde{\Sigma}$, then $\tilde{\sigma}$ has a fireable prefix $\tilde{\sigma}_p$, which might be empty, a transition t and a suffix $\tilde{\sigma}_s$ such that $\tilde{\sigma} = \tilde{\sigma}_p t \tilde{\sigma}_s$, so that t is the first disabled transition of $\tilde{\sigma}$. Let σ_p be the prefix of σ corresponding to $\tilde{\sigma}_p$, $proj_{\tilde{T}}(\sigma_p) = \tilde{\sigma}_p$. By the Effect Equations 5.1a and 5.1b, respectively, it follows that $M_{\sigma_p}|_{\tilde{P}\setminus\{q\}} = M_{\tilde{\sigma}_p}|_{\tilde{P}\setminus\{q\}}$. So t has q as an input place and $M_{\sigma_p}(q) > M_{\tilde{\sigma}_p}(q)$. □

Proposition 5.3.4 *Let $\tilde{\Sigma}$ be either ${\Sigma_e}^{q=1}$, ${\Sigma_e}^{q=0}$, $\Sigma_k^{q=0}$ or $\Sigma_k^{q=1}$. Let σ be a firing sequence of Σ and let $\tilde{\sigma}$ be a firing sequence of $\tilde{\Sigma}$ with $proj_{\tilde{T}}(\sigma) = \tilde{\sigma}$.*

(i) If q is 1-safe in $\tilde{\Sigma}$ and σ is fair w.r.t. $\tilde{T}$ but $\tilde{\sigma}$ is not maximal, then $\tilde{\sigma}$ is finite and $\tilde{M}_{\tilde{\sigma}}(q) = 1$ and σ does not eventually permanently mark q.

(ii) If $\tilde{\sigma}$ is maximal but σ is not fair w.r.t. $\tilde{T}$, then
$\tilde{\sigma}$ is finite and $\tilde{M}_{\tilde{\sigma}}(q) = 0$ and σ eventually permanently marks q.

Proof (i) If $\tilde{\sigma}$ is not maximal, then it has to be finite and there is a transition $\tilde{t} \in \tilde{T}$ that is enabled after firing $\tilde{\sigma}$, $\tilde{M}_{\tilde{\sigma}}[\tilde{t}\rangle$. If also σ is fair w.r.t. $\tilde{T}$, it follows that σ does not eventually permanently enable $\tilde{t}$. Hence there is a finite prefix σ_p of σ with $proj_{\tilde{T}}(\sigma_p) = \tilde{\sigma}$ and M_{σ_p} does not enable $\tilde{t}$. By Eq. 5.1a or 5.1b, respectively, it follows that $M_{\sigma_p}|_{\tilde{P}\setminus\{q\}} = \tilde{M}_{\tilde{\sigma}}|_{\tilde{P}\setminus\{q\}}$. So $\tilde{t}$ is enabled at $\tilde{M}_{\tilde{\sigma}}$ and not at M_{σ_p}, because $\tilde{M}_{\tilde{\sigma}}(q) > M_{\sigma_p}(q)$. Since q is 1-safe in $\tilde{\Sigma}$, $\tilde{M}_{\tilde{\sigma}}(q) = 1$ and $M_{\sigma_p}(q) = 0$. σ does not eventually permanently mark q, because otherwise it would eventually permanently enable $\tilde{t}$.

(ii) If σ is not fair w.r.t. $\tilde{T}$, then it eventually permanently enables a transition $\tilde{t} \in \tilde{T}$ but fires only finitely many transitions in $\tilde{T}$. Since $proj_{\tilde{T}}(\sigma) = \tilde{\sigma}$, $\tilde{\sigma}$ is hence finite. Since $\tilde{\sigma}$ is maximal, $\tilde{M}_{\tilde{\sigma}}$ does not enable $\tilde{t}$. Let σ_p be a finite prefix of σ that enables $\tilde{t}$ and contains $\tilde{\sigma}$, i.e. $proj_{\tilde{T}}(\sigma_p) = \tilde{\sigma}$ and $M_{\sigma_p}[\tilde{t}\rangle$. By Eq. 5.1a or 5.1b, respectively, it follows that $M_{\sigma_p}|_{\tilde{P}\setminus\{q\}} = \tilde{M}_{\tilde{\sigma}}|_{\tilde{P}\setminus\{q\}}$. So $\tilde{t}$ is enabled at M_{σ_p} and not at $\tilde{M}_{\tilde{\sigma}}$, because $\tilde{M}_{\tilde{\sigma}}(q) < M_{\sigma_p}(q)$. Since q is 1-safe, $\tilde{M}_{\tilde{\sigma}}(q) = 0$ and $M_{\sigma_p}(q) = 1$. As σ eventually permanently enables $\tilde{t}$, it eventually permanently marks q. □

The following proposition is very similar to the previous, but now refers to a reduced net. Since the reduced net may have transitions additional to T_k (as e.g. a Consumer-reduced net has t_r), the result (i) of the following proposition varies from the result (i) of the previous proposition.

Proposition 5.3.5 *Let Σ' be the reduced of Σ by an arbitrary reduction of Def. 5.2.2. Let σ be a firing sequence of Σ and let σ' be a firing sequence of Σ' with $proj_{T_k}(\sigma') = proj_{T_k}(\sigma)$.*

(i) If q is 1-safe in Σ', σ is fair w.r.t. T_k, σ' is not maximal and a transition in $T_k \subseteq T'$ is enabled, then
$M'_{\sigma'}(q) = 1$ while σ does not eventually permanently mark q.

(ii) If σ' is maximal but σ is not fair w.r.t. T_k, then
σ' is finite and $M'_{\sigma'}(q) = 0$ and σ eventually permanently marks q.

Proof In Σ and Σ' the token count on $P_k \setminus \{q\}$ initially coincides and is only effected by transitions in T_k. So the proof is in its main parts analogous to the proof of Prop. 5.3.4.

(i) If σ' is not maximal, then it and also $proj_{T_k}(\sigma) = proj_{T_k}(\sigma')$ have to be finite and there is a transition $t' \in T'$ enabled after firing σ'. This transitions may be in $T' \setminus T_k$ or in T_k. We are interested in the latter case.

There is a prefix σ_p of σ with $proj_{T_k}(\sigma_p) = proj_{T_k}(\sigma')$. It holds that $M_{\sigma_p}|_{P_k\setminus\{q\}} = M'_{\sigma'}|_{P_k\setminus\{q\}}$. Since we assume that q is 1-safe in Σ' and σ is fair w.r.t. T_k, it follows analogously to Prop. 5.3.4 that $M'_{\sigma'}(q) = 1$ and σ does not eventually permanently mark q.

(ii) Every firing sequence of any reduced net can fire only finitely many transitions in $T' \setminus T_k$ by construction. Hence (ii) follows analogously to (ii) of Prop. 5.3.4. □

After we have established the correspondence between maximal firing sequences of Σ' and fair firing sequences of Σ, we can show by the following proposition that Σ and Σ' are equivalent w.r.t. any $\text{LTL}_{-\mathsf{X}}$ formula ψ that does not refer to the environment Σ_e. The proposition states that marking sequences generated by corresponding transition sequences are equivalent w.r.t. ψ.

Proposition 5.3.6 *Let ψ be an $LTL_{-\mathsf{X}}$ formula with $scope(\psi) \subseteq P_k \setminus \{q\}$ and Σ' the reduced of Σ by an arbitrary reduction of Def. 5.2.2.*
Let M_0 be a marking of N and M'_0 a marking of N' with $M'_0|_{P_k\setminus\{q\}} = M_0|_{P_k\setminus\{q\}}$.
Let σ be a transition sequence in T^∞ and σ' a transition sequence in T'^∞ such that $proj_{T_k}(\sigma) = proj_{T_k}(\sigma')$ and such that $\mathcal{M}(M'_0, \sigma')$ and $\mathcal{M}(M_0, \sigma)$ are infinite marking sequences.

$$\mathcal{M}(M_0, \sigma) \models \psi \Leftrightarrow \mathcal{M}(M'_0, \sigma') \models \psi$$

Proof The proof is by induction on the structure of ψ.

$\psi = (p, x)$: As the satisfiability depends on marking of $p \in P_k \setminus \{q\}$ under M_0 and M'_0 only and since $M_0|_{P_k\setminus\{q\}} = M'_0|_{P'\setminus\{q\}}$, both directions hold.

The cases $\psi = \neg\psi_1$, $\psi = \psi_1 \wedge \psi_2$ follow directly by the induction hypothesis.

$\psi = \psi_1 \mathsf{U} \psi_2$: Let us assume $\mathcal{M}(M'_0, \sigma') = M'_0 M'_1 M'_2 ... \models \psi_1 \mathsf{U} \psi_2$. Let $\mathcal{M}(M_0, \sigma)$ be $M_0 M_1 M_2 ...$. We can find a prefix σ'_1 of σ' such that $M'_{|\sigma'_1|} M'_{|\sigma'_1|+1} ... \models \psi_2$ and $\forall i,\, 0 \leq i < |\sigma'_1| : M'_i M'_{i+1} ... \models \psi_1$. We hence can find a prefix σ_1 of σ corresponding to σ'_1, i.e. $proj_{T_k}(\sigma_1) = proj_{T_k}(\sigma'_1)$, and σ_1 does not end with a transition $t \in T_e$. By the induction hypothesis and Eq. 5.1a $M_{|\sigma_1|} M_{|\sigma_1|+1} ... \models \psi_2$. For the case that $|\sigma_1| > 0$, let σ_p be a prefix of σ such that $|\sigma_p| < |\sigma_1|$. Let σ'_p be a corresponding prefix of σ', i.e. $proj_{T_k}(\sigma_p) = proj_{T_k}(\sigma'_p)$. Since σ_p truncates at least one transition in T_k, $|\sigma'_p| < |\sigma'_1|$. From $M'_{|\sigma'_p|} M'_{|\sigma'_p|+1} ... \models \psi_1$, it follows by the induction hypothesis that $M_{|\sigma_p|} M_{|\sigma_p|+1} ... \models \psi_1$.

Analogously, it can be shown that $\mathcal{M}(M_0, \sigma) \models \psi_1 \mathsf{U} \psi_2 \Rightarrow \mathcal{M}(M'_0, \sigma') \models \psi_1 \mathsf{U} \psi_2$ holds. □

The correspondence of firing sequences is also cental for establishing preservation of $\mathrm{CTL}^*_{-\mathsf{X}}$ properties. We proof that $\mathrm{CTL}^*_{-\mathsf{X}}$ is preserved by showing that Σ and the reduced net Σ' are bisimilar. We define the bisimulation relation based on the firing of transitions in T_k. But for bisimulation a further ingredient besides correspondence of firing sequences is needed. For the Borrower, Producer and Dead End reductions we have a kind of Uniqueness Lemma stating that a marking on Σ corresponds to just one marking on Σ'.

5.3.2 Borrower Reduction

In this section we examine Borrower-reducible environments, i.e. environments Σ_e where q is a 1-safe place of ${\Sigma_\mathrm{e}}^{q=1}$ and ${\Sigma_\mathrm{e}}^{q=1} \models \mathsf{AFG}(q, 1)$. We prove that if Σ is reducible by a Borrower subnet N_e and if the given $\mathrm{CTL}^*_{-\mathsf{X}}$ formula φ does not refer to N_e, $\Sigma \underline{^b} \Sigma_\mathrm{e}$ satisfies φ if and only if Σ satisfies φ fairly w.r.t. T_k.

Convention In the following we denote $\Sigma \underline{^b} \Sigma_\mathrm{e} = \Sigma_\mathrm{k}$ also as $\Sigma' = (N', M'_{\mathsf{init}}) = (P', T', W', M'_{\mathsf{init}})$.

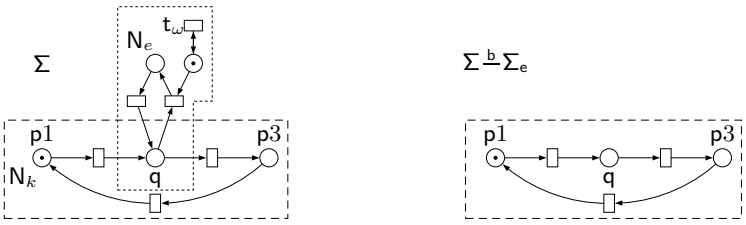


Figure 5.6: Example of a Borrower reduction

For the following we assume: (1) q is 1-safe in Σ.

The next propositions show that (T1) for any firing sequence σ of Σ it holds that its projections to T_k or T_e are also firing sequences of Σ_{k} or ${\Sigma_{\mathrm{e}}}^{q=1}$, respectively. We use this to show that (T2) for any firing sequence σ of Σ that is fair w.r.t. T_k it holds that its projection to T_k is a maximal firing sequence of the reduced net Σ'.

Proposition 5.3.7 *Let σ be a firing sequence of Σ such that $proj_{T_k}(\sigma)$ is a firing sequence of Σ_{k}.*

$proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_{\mathrm{e}}}^{q=1}$.

Proof Let M^e_{init} be $M^e_{\mathsf{init}} = (M^{q=1}_{\mathsf{init}}|_{P_e})$. If σ is a firing sequence of Σ but $\sigma^e := proj_{T_e}(\sigma)$ is not a firing sequence of ${\Sigma_{\mathrm{e}}}^{q=1}$, then by Prop. 5.3.3, there are prefixes σ_p of σ and σ^e_p of σ^e such that $M_{\sigma_p}(q) > M^e_{\sigma^e_p}(q)$. Since q is 1-safe in Σ, it follows that $M_{\sigma_p}(q) = 1$ and $M^e_{\sigma^e_p}(q) = 0$. So σ^e_p consumes a token from q because q is initially marked on ${\Sigma_{\mathrm{e}}}^{q=1}$. Hence with $\Delta(\sigma_p, q) = \Delta(\sigma^e_p, q) + \Delta(\sigma^k_p, q)$ where $\sigma^k_p := proj_{T_k}(\sigma_p)$ and from $1 = M_{\mathsf{init}}(q) + \Delta(\sigma_p, q)$, it follows that $2 = M_{\mathsf{init}}(q) + \Delta(\sigma^k_p, q)$. But since σ^k_p is a firing sequence of Σ_{k} by assumption, this contradicts the 1-safeness of q in Σ_{k} by Prop. 5.3.2. □

For the following we assume: (1) and (2) q is 1-safe in ${\Sigma_{\mathrm{e}}}^{q=1}$.

Intuitively the next proposition says, given a firing sequence σ that is fair w.r.t. T_e, in case $proj_{T_k}(\sigma)$ eventually permanently marks q, then $proj_{T_e}(\sigma)$ is a maximal firing sequence of ${\Sigma_{\mathrm{e}}}^{q=1}$.

Proposition 5.3.8 *Let σ be a firing sequence from M_{init} that is fair w.r.t. T_e. Let $\sigma_1^k, \sigma_2^k \in T_k^\infty$ and σ_1^k be finite. Let M_{init}^k be $M_{\mathsf{init}}|_{P_k}$.*

If $proj_{T_k}(\sigma) = \sigma_1^k\sigma_2^k$ is a firing sequence of Σ_{k} and $M_{\sigma_1^k}^k(q) = 1$ and $\forall i, 1 \leq i \leq |\sigma_2^k| : \Delta(\sigma_2^k(i), q) = 0$, then $proj_{T_e}(\sigma)$ is a maximal firing sequence of ${\Sigma_{\mathrm{e}}}^{q=1}$.

Proof Let M_{init}^e be $M_{\mathsf{init}}^{q=1}|_{P_e}$ and let $proj_{T_k}(\sigma)$ be σ^k. By Prop. 5.3.7, $\sigma^e := proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_{\mathrm{e}}}^{q=1}$. Assume σ^e is not maximal on ${\Sigma_{\mathrm{e}}}^{q=1}$. Hence by Prop. 5.3.4, σ^e is finite, $M_{\sigma^e}^e(q) = 1$ and σ does not eventually permanently mark q. So there is a finite prefix σ_p of σ with $M_{\sigma_p}(q) = 0$ and σ_p contains σ^e and σ_1^k. From $M_{\mathsf{init}}^e(q) = 1$ and $M_{\sigma^e}^e(q) = 1$ it follows that $\Delta(\sigma^e, q) = 0$. It thus follows from $M_{\sigma_p}(q) = 0 = M_{\mathsf{init}}(q) + \Delta(\sigma_p, q)$ that $0 = M_{\mathsf{init}}(q) + \Delta(proj_{T_k}(\sigma_p), q)$. Since we assume that σ_2^k does not affect q, it follows $0 = M_{\mathsf{init}}(q) + \Delta(\sigma_1^k, q)$. But this contradicts the assumption that firing σ_1^k places a token on q, that is $M_{\sigma_1^k}^k(q) = 1$. □

Proposition 5.3.9 ${\Sigma_{\mathrm{e}}}^{q=0} \models \mathsf{AG}(q, 0)$

Proof Let M_{init}^e be $M_{\mathsf{init}}^{q=0}|_{P_e}$. Suppose that ${\Sigma_{\mathrm{e}}}^{q=0} \not\models \mathsf{AG}(q, 0)$. Thus there is a firing sequence σ^e with $M_{\sigma^e}^e(q) \geq 1$. σ^e is also a firing sequence of ${\Sigma_{\mathrm{e}}}^{q=1}$ generating two tokens on q, which contradicts the 1-safeness of q in ${\Sigma_{\mathrm{e}}}^{q=1}$. □

Now we can show (T1) of the targeted results: By the next proposition we can fire $proj_{T_k}(\sigma)$ on Σ_{k} for any firing sequence σ of Σ. And by Prop. 5.3.7 also $proj_{T_e}(\sigma)$ is a firing sequence of $\Sigma_e^{q=1}$.

Proposition 5.3.10 *Let σ be a firing sequence of Σ.*

$proj_{T_k}(\sigma)$ is a firing sequence of Σ_{k}.

Proof We denote the initial marking $M_{\mathsf{init}}|_{P_k}$ of Σ_{k} as M_{init}^k. If the above does not hold, then by Prop. 5.3.3, there are prefixes σ_p of σ and σ_p^k of $proj_{T_k}(\sigma)$ such that $M_{\sigma_p}(q) > M_{\sigma_p^k}^k(q)$. Since $M_{\mathsf{init}}^k(q) = M_{\mathsf{init}}(q)$, it follows that $\Delta(\sigma_p^k, q) < \Delta(\sigma_p, q)$, which implies that $0 < \Delta(\sigma_p^e, q)$ where $\sigma_p^e := proj_{T_e}(\sigma_p)$. But since σ_p^e is a firing sequence of ${\Sigma_{\mathrm{e}}}^{q=1}$ by Prop. 5.3.7, this contradicts assumption (2), i.e. 1-safeness of q in ${\Sigma_{\mathrm{e}}}^{q=1}$. □

For the following we assume: (1), (2) and (3) ${\Sigma_e}^{q=1} \models \mathsf{AFG}(q,1)$.

The following lemma is the Uniqueness Lemma: If σ_1 and σ_2 generate the same marking M then $proj_{T_k}(\sigma_1)$ and $proj_{T_k}(\sigma_2)$ will also generate the same marking. With other words the corresponding marking of M is unique. The Uniqueness Lemma will be used to show that Σ and the reduced net Σ' are bisimilar.

Lemma 5.3.11 (Uniqueness Lemma) *Let σ_1, σ_2 be firing sequences of Σ.*
If $\Delta(\sigma_1, p) = \Delta(\sigma_2, p), \forall p \in P$,
then $\Delta(proj_{T_k}(\sigma_1), p) = \Delta(proj_{T_k}(\sigma_2), p), \forall p \in P_k$.

Proof Let σ_1^k denote $proj_{T_k}(\sigma_1)$ and σ_2^k denote $proj_{T_k}(\sigma_2)$. Similarly, let σ_1^e be $proj_{T_e}(\sigma_1)$ and σ_2^e be $proj_{T_e}(\sigma_2)$. As $\Delta(\sigma_1, p) = \Delta(\sigma_2, p)$, $\forall p \in P$, and transitions in T_e cannot change the token count on places in P_k, it follows that $\Delta(\sigma_1^k, p_k) = \Delta(\sigma_2^k, p_k)$, $\forall p_k \in P_k \setminus \{q\}$ holds. Let us assume that $\Delta(\sigma_1^k, q) \neq \Delta(\sigma_2^k, q)$. It follows that also $\Delta(\sigma_1^e, q) \neq \Delta(\sigma_2^e, q)$, because $\Delta(\sigma_1^e, q) + \Delta(\sigma_1^k, q) = \Delta(\sigma_2^e, q) + \Delta(\sigma_2^k, q)$ by assumption. Since σ_1^e and σ_2^e are both firing sequences of ${\Sigma_e}^{q=1}$ by Prop. 5.3.7 and q is 1-safe in ${\Sigma_e}^{q=1}$, it follows that $\Delta(\sigma_1^e), \Delta(\sigma_2^e) \in \{-1, 0\}$. Without loss of generality let $\Delta(\sigma_1^e, q) = -1$ and $\Delta(\sigma_2^e, q) = 0$. Since Σ_e is a Borrower and thus ${\Sigma_e}^{q=1}$ satisfies $\mathsf{AFG}(q,1)$, all maximal firing sequences are non-consuming. Hence a firing sequence σ_g^e is enabled after firing σ_1^e that eventually marks q. But σ_g^e is also a firing sequence from $M_{\sigma_2^e}^e$ and generates two tokens on q. This contradicts 1-safeness of q in ${\Sigma_e}^{q=1}$. □

Remark 5.3.12 *For the Borrower reduction, the reduced net Σ' is equal to Σ^k. So Prop. 5.3.10 and the Uniqueness Lemma 5.3.11 also hold for Σ' instead of Σ^k.*

With the next two propositions we are ready to show that $\mathrm{CTL}^*_{-\mathsf{X}}$ is preserved. According to the next propositions it holds that

- $\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}}) \subseteq proj_{T_k}(\mathsf{Fs}_{N,\{T_k,T_e\}}(M_{\mathsf{init}}))$ and
- $proj_{T_k}(\mathsf{Fs}_{N,\{T_k\}}(M_{\mathsf{init}})) \subseteq \mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}})$.

Note that $\mathsf{Fs}_{N,\{T_k,T_e\}}(M_{\mathsf{init}}) \subseteq \mathsf{Fs}_{N,\{T_k\}}(M_{\mathsf{init}})$. So it also follows that

- $\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}}) \subseteq proj_{T_k}(\mathsf{Fs}_{N,\{T_k\}}(M_{\mathsf{init}}))$ and
- $proj_{T_k}(\mathsf{Fs}_{N,\{T_k,T_e\}}(M_{\mathsf{init}})) \subseteq \mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}})$.

In summary, we can derive that
$\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}}) = proj_{T_k}(\mathsf{Fs}_{N,\{T_k\}}(M_{\mathsf{init}})) = proj_{T_k}(\mathsf{Fs}_{N,\{T_k,T_e\}}(M_{\mathsf{init}}))$ holds.

Proposition 5.3.13 *Let σ be a firing sequence of Σ that is fair w.r.t. T_k. $proj_{T_k}(\sigma)$ is a maximal firing sequence of Σ'.*

Proof Note that $\Sigma' = \Sigma_{\mathrm{k}}$. By Prop. 5.3.10, $\sigma' := proj_{T_k}(\sigma)$ is a firing sequence of Σ'. Let us assume that σ' is not maximal. By Prop. 5.3.4 σ' is finite and $M'_{\sigma'}(q) = 1$ and σ does not eventually permanently mark q. Let σ_p be a prefix of σ that contains σ' and does not mark q, i.e. $proj_{T_k}(\sigma_p) = \sigma'$ and $M_{\sigma_p}(q) = 0$. Hence $0 = M_{\mathsf{init}}(q) + \Delta(\sigma', q) + \Delta(proj_{T_e}(\sigma_p), q)$ and $1 = M'_{\sigma'}(q) = M_{\mathsf{init}}(q) + \Delta(\sigma', q)$ holds. It follows that $\Delta(proj_{T_e}(\sigma_p), q) = -1$. Since σ' is finite and σ is maximal, it follows that σ is fair w.r.t. T_e. By Prop. 5.3.8, $proj_{T_e}(\sigma)$ is a maximal firing sequence of ${\Sigma_{\mathrm{e}}}^{q=1}$. As ${\Sigma_{\mathrm{e}}}^{q=1} \models \mathsf{AGF}(q, 1)$, σ eventually permanently enables q (contradiction). □

A stronger version of Prop. 5.3.13, that requires maximality of σ only, instead of fairness w.r.t T_k, does not hold, as we have seen while discussing the need of fairness at the begin of this section. The maximal firing sequence $\sigma = t_1 t_\omega t_\omega ...$ of the net in Fig. 5.5 has the $proj_{T_k}(\sigma) = t_1$, which is not maximal on $\Sigma \underline{b} \Sigma_{\mathrm{e}}$.

By the next proposition it follows $\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}}) \subseteq proj_{T_k}(\mathsf{Fs}_{N,\{T_k,T_e\}}(M_{\mathsf{init}}))$. But more than that, it says that for a maximal firing sequence σ' from M' and for a pair of markings $(M, M') = (M, M|_{P_k})$ of respectively Σ and Σ', we can find a fair firing sequence σ that visits M and corresponds to σ'. This extra is necessary to establish bisimulation. Proposition 5.3.13 has a simpler form, as for a marking M we can pinpoint M' (Uniqueness Lemma), whereas, vice versa, we can find for a marking M' more than one corresponding marking of Σ.

Proposition 5.3.14 *Let $\sigma' = \sigma'_1\sigma'_2$ be a maximal firing sequence of the Borrower-reduced Σ'. Let σ_1 be a firing sequence of Σ with $proj_{T_k}(\sigma_1) = \sigma'_1$.*

If σ_1 is finite, then there is a firing sequence σ_2 of Σ such that $\sigma_1\sigma_2$ is fair w.r.t. T_k and T_e, and $proj_{T_k}(\sigma_2) = \sigma'_2$.

Proof The algorithm to construct σ_2 can be sketched as follows. First, σ is extended to sufficiently mark q (`line 10-16`). Then, as long as σ' fires transitions whose firing decreases q's token count, transitions in T_e are fired that do not have q as an input place (`line 17-25`). Then, if σ' fires transitions that do not change q's token count but have q as an input place (`line 29-41`), first a prefix of σ' is fired that places a token on q (`line 30`). In this case a maximal firing sequence σ^e of Σ_e is enabled and we know that σ^e behaves like a Borrower sequence. We fire the "borrowing" prefix of a σ^e, i.e. the prefix of σ^e up to the point when the token from q is not removed any more. Then (line `37-40` or `42-46`) we fire in turn transitions in T_k (the suffix of σ') and T_e (the suffix of σ^e), since they do not disable each other. In the following M^e_{init} denotes $M_{\mathsf{init}}|_{P_e}$.

```
1  /* The algorithm's input is the original net Σ, the
2   kernel Σk, the environment Σe, the firing sequence
3   σ1 of Σ and firing sequences σ'1 and σ'2 of the reduced
4   net Σ', such that σ'1σ'2 is maximal. Its output is a
5   firing sequence of Σ that is fair w.r.t. Te and Tk.*/
6  Input: Σ, Σk, Σe, σ1, σ'1, σ'2
7  Output: σ2
8  σ := σ1
9  σ' := σ'2        /* not yet part of σ */
10 if (Mσ(q) < Mσ'1(q)){
11     Let σ^e be a maximal firing sequence of Σe^{q=1} with
12     prefix proj_Te(σ) and M(M^e_init, σ^e) ⊨ FG(q, 1).
13     Let σ^e_1 be a transition sequence where
14     proj_Te(σ)σ^e_1 is a prefix of σ^e and M^e_{proj_Te(σ)σ^e_1}(q) = 1.
15     σ := σσ^e_1
16 }
```

17 **if** (σ' contains a $t' \in q^\bullet$ with $W(q,t') > W(t',q)$) {
18 Let σ'_p be the minimal prefix of σ' containing all
19 transitions t' with $W(q,t') > W(t',q)$.
20 **for** ($i := 1$; $i < |\sigma'_p| + 1$; $i := i + 1$) {
21 $\sigma := \sigma\sigma'_p(i)$
22 **if** ($\exists t_e \in (T_e \setminus q^\bullet) : M_\sigma[t_e\rangle$) $\sigma := \sigma t_e$
23 }
24 $\sigma' := \sigma'^{(|\sigma'_p|)}$ /* *truncate by prefix* σ'_p */
25 }
26 /* *From now on holds that* $W(q,\sigma'(i)) \leq W(\sigma'(i),q)$,
27 $\forall i, 1 \leq i < |\sigma'| + 1$. */
28 **if** (σ' contains a $t' \in q^\bullet$) {
29 Let σ'_p be σ''s minimal prefix that includes a $t' \in q^\bullet$.
30 $\sigma := \sigma\sigma'_p$
31 $\sigma' := \sigma'^{(|\sigma'_p|)}$ /* *truncate by prefix* σ'_p */
32 Let σ^e be a maximal firing sequence of ${\Sigma_e}^{q=1}$ with
33 prefix $proj_{T_e}(\sigma)$ and $\mathcal{M}(M^e_{\mathsf{init}}, \sigma^e) \models \mathsf{FG}(q,1)$.
34 Let σ^e_1 and σ^e_2 be transition sequences with
35 $\sigma^e = proj_{T_e}(\sigma)\sigma^e_1\sigma^e_2$ and $\mathcal{M}(M^e_{proj_{T_e}(\sigma)\sigma^e_1}, \sigma^e_2) \models \mathsf{G}(q,1)$.
36 $\sigma := \sigma\sigma^e_1$
37 **for** ($i := 1$; $i < |\sigma^e_2| + 1$ or $i < |\sigma'| + 1$; $i := i + 1$) {
38 **if** ($i < |\sigma^e_2| + 1$) $\sigma := \sigma\sigma^e_2(i)$
39 **if** ($i < |\sigma'| + 1$) $\sigma := \sigma\sigma'(i)$
40 }
41 } **else** { /* $q \notin {}^\bullet\sigma'(i)$, $\forall i, 1 \leq i \leq |\sigma'|$ */
42 **for** ($i := 1$; $i < |\sigma'| + 1$; $i := i + 1$) {
43 $\sigma := \sigma\sigma'(i)$
44 **if** ($\exists t_e \in T_e : M_\sigma[t_e\rangle$) $\sigma := \sigma t_e$
45 }
46 **while** ($\exists t_e \in T_e : M_\sigma[t_e\rangle$) $\sigma := \sigma t_e$
47 }

48 return $\sigma := \sigma^{(|\sigma_1|)}$

Listing 5.1: Generating a firing sequence fair w.r.t. T_e and T_k.

The transition sequence σ, constructed in the first if-block, (`line 10-16`), is a firing sequence of Σ: Since $0 = M_\sigma(q) < M_{\sigma_1'}(q) = 1$, it follows that $proj_{T_e}(\sigma)$ consumed a token from q. But since ${\Sigma_e}^{q=1} \models \mathsf{AFG}(q,1)$, there is a firing sequence σ_e that (re)generates the token.

The transition sequence σ, constructed in the second if-block, (`line 17-25`), is a firing sequence of Σ, since $M_{\mathsf{init}}[\sigma_1\rangle$, since by construction $M_\sigma(q) = M'_{\sigma_1'}(q)$ at `line 16` and since transitions in $t \in T_e \setminus q^\bullet$ do not disable transitions in T_k. Note that at the end of `line 25` $M_\sigma(q) = 0 = M'_{proj_{T'}(\sigma)}(q)$, since σ_p' decreases q's token count. In case $\sigma_p' = \sigma'$ is infinite, σ is by construction fair w.r.t. T_k and obviously fair w.r.t. $T_e \setminus q^\bullet$. As σ_p' infinitely often removes the token from q, the place q is not permanently marked, and thus transitions in $q^\bullet \cap T_e$ are not permanently enabled. Hence σ is fair w.r.t. T_e.

Next we show that σ constructed at the third if-block (`line 28-41`), is a firing sequence of Σ. Since $M_\sigma[\sigma'\rangle$ at the end of `line 24` , σ of `line 30` is a firing sequence. Let t' be the last transition of σ_p'. As t' has q as an input place and does not decrease the token count on q, q has to be marked at the end of `line 30`. Also the firing sequence of `line 32-35` exists: As $proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_e}^{q=1}$ by Prop. 5.3.7 and since ${\Sigma_e}^{q=1} \models \mathsf{AFG}(q,1)$, $proj_{T_e}(\sigma)$ can be extended to a maximal firing sequence σ^e whose marking sequence satisfies $\mathsf{FG}(q,1)$. It follows that σ^e can be divided into a finite prefix σ_p^e that contains $proj_{T_e}(\sigma)$ and restores the token on q and into a suffix σ_2^e that does not remove the token on q, $\mathcal{M}(M^e_{\sigma_p^e}, \sigma_2^e) \models \mathsf{G}(q,1)$. Let σ_1^e be the transition sequence such that $\sigma_p^e = proj_{T_e}(\sigma)\sigma_1^e$. The transition sequence σ of `line 36` is a firing sequence of Σ, since by Eq. 5.1b $M_\sigma|_{P_e\setminus\{q\}} = M_{proj_{T_e}(\sigma))}|_{P_e\setminus\{q\}}$ holds at the end of `line 30`, and by construction $M_\sigma(q) = 1 \geq M^e_{proj_{T_e}(\sigma)}(q)$ at the begin of `line 36`. σ of `line 38` and `39` are firing sequences: σ_1^e does not change the token count of places $P_k\setminus\{q\}$ and $M_\sigma(q) = 1$ holds at the end of `line 36`. It follows that $M_\sigma[\sigma'\rangle$ and $M_\sigma[\sigma_2^e\rangle$. Neither the transitions of σ_2^e nor the transitions of σ' remove the token from q .

We now show that σ is fair w.r.t. T_k and T_e. We first show that σ is fair

w.r.t. T_k. If σ' is infinite, σ is fair w.r.t. T_k. Suppose σ' is finite. Let σ'_p be the already considered prefix of σ', that is $proj_{T_k}(\sigma) =: \sigma'_p$. We show that for each σ at `line 36, 38` and `39` the generated marking M_σ satisfies $M_\sigma|_{P_k} = M'_{\sigma'_p}$, since at these lines σ may become maximal or the algorithm may infinitely loop. As σ' is maximal, $M_\sigma|_{P_k} = M'_{\sigma'} = M'_{\sigma'_p}$ implies that $\neg M_\sigma[t\rangle, \forall t \in T_k$. By Eq. 5.1a it follows that at all times $M_\sigma|_{P_k\setminus\{q\}} = M'_{\sigma'_p}|_{P_k\setminus\{q\}}$. $M_\sigma(q) = 1 = M'_{\sigma'_p}(q)$ holds at the end of `line 36`, which has been shown above. Neither transitions of σ^e_2 nor transitions of σ' may change the token count on q afterwards, thus $M_\sigma|_{P_k} = M'_{\sigma'_p}$ holds at `line 38` and `39`.

Next we show that σ is fair w.r.t. T_e. Transitions of σ' do not change the token count of q from line `32 to 40`. So by construction $proj_{T_e}(\sigma)$ is a maximal firing sequence of ${\Sigma_e}^{q=1}$. At `line 36-40` it holds that $M_\sigma(q) = 1$, so it follows that $M_\sigma(q)|_{P_e} = M^e_{proj_{T_e}(\sigma)}$. Hence it follows that σ is fair w.r.t. T_e.

The σ constructed at the else-block (`line 41 to 47`) is a firing sequence: By Eq. 5.1a, $M_{|\sigma|}|_{P\setminus\{q\}} = M'_{|\sigma'_p|}|_{P_k\setminus\{q\}}$. As the enabledness of transitions in σ' does not depend on the token count on q, any enabled transition of T_e can be fired in between two transitions of σ'. By construction σ is fair w.r.t. T_e, as transitions in T_e are fired as long as there are any enabled. σ is fair w.r.t T_k because σ' is maximal and transitions of σ' do not depend on q. □

The following proposition establishes that every maximal firing sequence of Σ' corresponds to the projection of a firing sequence of Σ that is fair w.r.t. T_k (not also fair w.r.t. T_e), i.e. $\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}}) \subseteq proj_{T_k}(\mathsf{Fs}_{N,\{T_k\}}(M_{\mathsf{init}}))$. The proposition also guarantees that σ starts with σ'. We need this form to show that we can use the reduced net for falsification of ∀CTL* properties using the next-time operator X.

Proposition 5.3.15 *Let σ' be a maximal firing sequence of Σ'.*

There is a firing sequence σ of Σ such that $proj_{T_k}(\sigma) = \sigma'$ and σ starts with σ' and σ is fair w.r.t. T_k.

Proof By Prop. 5.3.1 σ' is a firing sequence of Σ. If σ' is finite, we fire transitions of T_e as long as there are any enabled. Let us assume that σ is

not fair w.r.t. T_k. By Prop. 5.3.4 σ' is finite and $M'_{\sigma'}(q) = 0$ and σ eventually permanently marks q. Hence $proj_{T_e}(\sigma)$ generates a token, which contradicts assumption (2). □

We are now in the position to show our main verification and falsification results:

Theorem 5.3.16 *Let Σ_e a Borrower environment net and Σ be reducible by Σ_e. Let φ be a CTL$^*_{-X}$ formula referring to $P \setminus P_e$ only.*

$\Sigma \underline{^b} \Sigma_e \models \varphi \Leftrightarrow \Sigma \models \varphi$ *fairly w.r.t.* T_k *and*

$\Sigma \underline{^b} \Sigma_e \models \varphi \Leftrightarrow \Sigma \models \varphi$ *fairly w.r.t.* T_k *and* T_e.

Proof We show that (i) TS_Σ and $TS_{\Sigma'}$ are stuttering bisimilar assuming that Σ is fair w.r.t T_k and (ii) TS_Σ and $TS_{\Sigma'}$ are stuttering bisimilar assuming Σ is fair w.r.t. T_k and T_e. In a first step we define the relation $\mathcal{B} \subseteq [M_{\mathsf{init}}\rangle \times [M'_{\mathsf{init}}\rangle$ and then show that $\forall (M, M') \in \mathcal{B}$:

(L) $L(M) = L'(M')$

(SF1) $\forall \mu \in \Pi_{TS_\Sigma,\{T_k\}}(M) : \exists \mu' \in \Pi_{TS_{\Sigma'},\mathsf{inf}}(M') : \mathsf{match}(\mathcal{B}, \mu, \mu')$

(SF2) $\forall \mu' \in \Pi_{TS_{\Sigma'},\mathsf{inf}}(M') : \exists \mu \in \Pi_{TS_\Sigma,\{T_k,T_e\}}(M) : \mathsf{match}(\mathcal{B}, \mu, \mu')$

Note, (SF1) implies $\forall \mu \in \Pi_{TS_\Sigma,\{T_k,T_e\}}(M) : \exists \mu' \in \Pi_{TS_{\Sigma'},\mathsf{inf}}(M') : \mathsf{match}(\mathcal{B}, \mu, \mu')$ and (SF2) implies that $\forall \mu' \in \Pi_{TS_{\Sigma'},\mathsf{inf}}(M') : \exists \mu \in \Pi_{TS_\Sigma,\{T_k\}}(M) : \mathsf{match}(\mathcal{B}, \mu, \mu)$ holds, since $\Pi_{TS_\Sigma,\{T_k,T_e\}}(M) \subseteq \Pi_{TS_\Sigma,\{T_k\}}(M)$. Thus, it follows from (L), (SF1), (SF2) that $\mathcal{B}$ is a stuttering fair bisimulation assuming that Σ is fair w.r.t. T_k, and also assuming that Σ is fair w.r.t. T_k and T_e.

We now define $\mathcal{B} \subseteq [M_{\mathsf{init}}\rangle \times [M'_{\mathsf{init}}\rangle$. The basic idea is that two markings M and M' are bisimilar iff they correspond, $M|_{P\setminus\{q\}} = M'|_{P\setminus\{q\}}$, but then for one marking M there might be several markings M'_i and we have to be careful with the token count on q. So let M be a reachable marking of Σ. Hence there is a firing sequence σ that generates M, $M_{\mathsf{init}}[\sigma\rangle M$. $proj_{T_k}(\sigma)$ is a firing sequence of Σ' by Prop. 5.3.10. Let M' be the marking of Σ' generated by firing $proj_{T_k}(\sigma)$, $M'_{\mathsf{init}}[proj_{T_k}(\sigma)\rangle M'$. We define that $(M, M') \in \mathcal{B}$. Defining $\mathcal{B}$ this way, it follows that $(M_{\mathsf{init}}, M'_{\mathsf{init}}) \in \mathcal{B}$ and since $AP \subseteq (P' \setminus \{q\}) \times \mathbb{N}$, it also follows that $\forall (M, M') \in \mathcal{B} : L(M) = L(M')$ holds. Note that by Prop.

5.3.11, M' is uniquely defined, i.e. $(M, M'_1) \in \mathcal{B}$ and $(M, M'_2) \in \mathcal{B}$ it follows that $M'_1 = M'_2$.

To show (SF1), consider $(M, M') \in \mathcal{B}$ and a path μ_2 from M that is fair w.r.t. T_k. Let σ_2 a the corresponding firing sequence, that is $\mathcal{M}(M, \sigma_2) = \mu_2$. Let σ_1 be a firing sequence that generates M and μ_1 the corresponding path, that is $\mu_1 = \mathcal{M}(M_{\mathsf{init}}, \sigma_1)$. Hence $\mu := \mu_1\mu_2$ is a path from M_{init} in TS_Σ that is fair w.r.t. T_k. Let $\sigma := \sigma_1\sigma_2$ be the corresponding firing sequence. By Prop. 5.3.13, $\sigma' := proj_{T_k}(\sigma)$ is a maximal firing sequence of Σ'. By definition of $\mathcal{B}$ and by the Uniqueness Lemma 5.3.11 firing $proj_{T_k}(\sigma_1)$ generates M'. It follows that the marking sequence μ'_2 generated by $proj_{T_k}(\sigma_2)$ is a path from M' in $\Pi_{TS_{\Sigma'},\mathsf{inf}}(M')$.

Based on σ and σ' we define partitions θ of μ_2 and θ' of μ'_2. Let the partition θ' be the identity mapping and let θ be the partition defined as $\theta(1) = 1$ and $\forall i,\ 1 < i < |\sigma'_2| + 2$, $\theta(i) = j + 1$ where j is such that $\sigma(j) \in T_k$ and $\sigma(1)...\sigma(j)$ has exactly $i-1$ transitions in T_k, $\theta(i) = \theta(i-1)+1$ otherwise. Fig. 5.7 illustrates the partitioning of μ_2 and μ'_2.

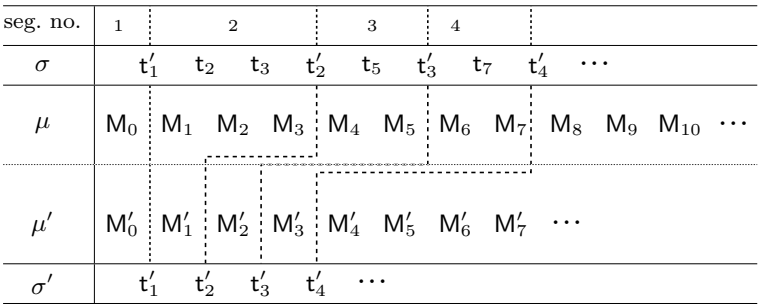


Figure 5.7: Partitioning of corresponding marking sequences. μ is divided into segments according to occurrences of transitions in T_k.

θ partitions μ_2 so that segment 1 contains markings without a change on the token count of $P_k \setminus \{q\}$. Segment i, $1 < i < |\sigma'_2| + 1$, starts with the marking generated by firing its first $(i-1)$ transitions in T'. Note that this defines θ already well, if $\sigma'_2 = proj_{T_k}(\sigma_2)$ is infinite. In case σ'_2 is finite, segment i, $|\sigma'_2| + 1 < i$, consists of only one marking, the next marking in μ_2.

Segment i of μ'_2, $1 \leq i \leq |\sigma'_2| + 1$, contains only one marking M'_i, which is generated by firing the first $i - 1$ transitions of σ'_2.

We now show that the partitions of μ and μ' generate segments of bisimilar markings. Let us consider a segment i, $1 \leq i < |\sigma_2'| + 2$. Let M_2 be a marking in segment i on μ_2. M_2 is generated by firing a prefix σ_p with $i-1$ transitions in T_k. The marking M_2' in segment i on μ_2' is generated by firing $i-1$ transitions in T'. Hence it follows that $proj_{T_k}(\sigma_p) = \sigma_2'(1)...\sigma_2'(i-1)$ holds and hence $(M_2, M_2') \in \mathcal{B}$. Let us assume that σ_2' is finite. Segment i on μ_2', $i > |\sigma'| + 1$, contains the final marking generated by firing σ_2'. Segment i on μ, $i > |\sigma'| + 1$, contains the marking generated by firing a prefix σ_p of σ_2 that contains $|\sigma_2'|$ transitions of T'. So it follows that $proj_{T_k}(\sigma_p) = \sigma_2'$ holds and hence also $(M_2, M_2') \in \mathcal{B}$.

To show (SF2), let us consider $(M, M') \in \mathcal{B}$ and an infinite path μ_2' from M' in $\Pi_{TS_{\Sigma'},\mathsf{inf}}(M')$. Again let σ_1 be a firing sequence generating M and $\sigma_1' = proj_{T_k}(\sigma_1)$ generating M'. So $\mu_1 := \mathcal{M}(M, \sigma_1)$ and $\mu_1' := \mathcal{M}(M', \sigma_1')$ are the corresponding paths. It follows that $\mu' = \mu_1'\mu_2'$ is an infinite path in $TS_{\Sigma'}$. Let $\sigma' := \sigma_1'\sigma_2'$ be the corresponding maximal firing sequence of Σ' where σ_2' generates μ_2'. By Prop. 5.3.14 there is a firing sequence σ that is fair w.r.t. T_k and T_e and corresponds to σ'. Let σ_2 be the suffix of σ that corresponds to σ_2'. $\mu_2 := \mathcal{M}(M, \sigma_2)$ is thus a fair path from M. As above it follows that μ_2' and μ_2 are partitioned into segments of bisimilar markings. □

We cannot verify LTL or CTL properties using X. Consider the LTL property $\psi = \mathsf{AXX}(p_3, 1)$ and the CTL property $\varphi = \mathsf{AXAX}(p_3, 1)$. The net $\Sigma \underline{^b} \Sigma_\mathrm{e}$ in Fig. 5.6 satisfies both ψ and φ but Σ satisfies neither of them. As $\neg\varphi := \mathsf{EXEX}(p_3, 0)$ is a valid CTL property on Σ but not on $\Sigma \underline{^b} \Sigma_\mathrm{e}$, CTL properties using X can also not be falsified, but we can falsify $\forall$CTL* assuming fairness of Σ w.r.t. T_k.

Theorem 5.3.17 *Let Σ_e be a Borrower environment net and Σ be reducible by Σ_e. Let ψ be an $\forall CTL^*$ formula referring to $P \setminus P_e$ only.*

$\Sigma \models \psi$ *fairly w.r.t.* $T_k \quad \Rightarrow \quad \Sigma \underline{^b} \Sigma_\mathrm{e} \models \psi$.

Proof We show that $(TS_\Sigma, M_{\mathsf{init}})_{\{T_k\}}$ simulates $(TS_{\Sigma'}, M'_{\mathsf{init}})$. This implies that if $\Sigma \models \psi$ fairly w.r.t. T_k then $\Sigma \underline{^b} \Sigma_\mathrm{e} \models \psi$.

To define $\mathcal{S}$, we mimic the construction of σ in Prop. 5.3.15. Let $M' \in [M'_{\mathsf{init}}\rangle$ be a marking of Σ'. Let σ' be a firing sequence generating M'. If M' is not a final marking, only $(M_{\sigma'}, M') \in \mathcal{S}$. But if M' is final, we pick several markings M to correspond to M'. Let σ be any finite firing sequence starting with σ' and $proj_{T_k}(\sigma) = \sigma'$. We set $(M_\sigma, M') \in \mathcal{S}$ where M_σ is generated by such a σ. In both cases it holds that $(M_{\mathsf{init}}, M'_{\mathsf{init}}) \in \mathcal{S}$.

We have to show that all $(M, M') \in \mathcal{S}$ satisfy (L) $L(M) = L'(M')$, and (F) $\forall \mu' \in \Pi_{TS_{\Sigma'},\mathsf{inf}}(M') : \exists \mu \in \Pi_{TS_\Sigma,\{T_k\}}(M) : (\mu(i), \mu'(i)) \in \mathcal{S}$. As we require that $scope(\psi) \subseteq P_k \setminus \{q\}$, (L) holds by Eq. 5.1a. Let (M, M') be in $\mathcal{S}$, let μ' be an infinite path from M' and let σ' be the corresponding maximal firing sequence of Σ', that is $\mu' = \mathcal{M}(M', \sigma')$.

By definition of $\mathcal{S}$, there is a firing sequence σ_g that generates M on Σ and $\sigma'_g := proj_{T_k}(\sigma_g)$ generates M'. By Prop. 5.3.7 $\sigma^e_g := proj_{T_e}(\sigma_g)$ is a firing sequence of ${\Sigma_e}^{q=1}$ and hence does not generate additional tokens on q.

In case M' is final, μ' is the infinite marking sequence $M'M'\ldots$. As σ^e_g does not generate additional tokens, M enables only transitions in T_e. So for any marking M_i reachable from M consequently $(M_i, M') \in \mathcal{S}$ holds.

Let us now consider the case that M' is not final. By Prop. 5.3.15 there is a firing sequence σ with $proj_{T_k}(\sigma) = \sigma'_g\sigma'$ that is fair w.r.t. T_k. M is generated by σ'_g. Let σ_s be the suffix of σ with $proj_{T_k}(\sigma_s) = \sigma'$, which is also fair w.r.t. T_k and starts with σ'. $\mu = \mathcal{M}(M, \sigma_s)$ is hence a fair path of TS_Σ starting with markings corresponding to μ'. So the case that σ' is infinite follows trivially. In case σ' is finite, the markings generated along σ' correspond. The case that we reach a final marking has been discussed above. □

Theorem 5.3.17 implies that the weaker version "$\Sigma \models \psi \Rightarrow \Sigma' \models \psi$" hold. To see that a stronger version of the theorem assuming fairness of Σ w.r.t. T_k and T_e does not hold, consider the two nets in Fig. 5.6 and the LTL ($\forall$CTL*) property $\psi = \mathsf{AF}((p_3, 1) \wedge \mathsf{XXX}(p_3, 0))$. ψ expresses that all paths eventually mark p_3 and then do not mark $p3$ after three transition firings. Obviously, ψ does not hold on $\Sigma \underline{b} \Sigma_e$, but ψ holds on Σ, assuming Σ is fair w.r.t. T_e. Σ has to fire a transition in T_e, since otherwise t_ω is permanently enabled while

no transition of T_e occurs.

5.3.3 Consumer Reduction

A Consumer environment satisfies that q is a 1-safe place of ${\Sigma_e}^{q=1}$ and ${\Sigma_e}^{q=1} \not\models \mathsf{AFG}(q, 1)$. In the following we denote $\Sigma \stackrel{c}{-} \Sigma_e = (P_k, T_k \uplus \{t_r\}, W_k \uplus \{(q, t_r) \mapsto 1\}, M_{\mathsf{init,k}})$ also as Σ'.

Figure 5.8: Example of a Consumer reduction

For the following we assume: (1) q is 1-safe in Σ.

Proposition 5.3.18 *Let σ' be a firing sequence of Σ'.*
$proj_{T_k}(\sigma')$ is a firing sequence of Σ and Σ_k.

Proof Since t_r decreases only the token count of q, we can omit occurrences of t_k without enabling any transitions of σ'. Since $M'_{\mathsf{init}} = M_{\mathsf{init,k}}$, it follows that $proj_{T_k}(\sigma')$ is also a firing sequence of Σ' and hence of Σ_k. By Prop. 5.3.1 $proj_{T_k}(\sigma')$ is a firing sequence of Σ. □

The following proposition is a direct consequence of Prop. 5.3.18.

Proposition 5.3.19 *Let $p \in P_k$ be a k-bounded place in Σ.*
p is a k-bounded place in Σ'.

Proof Suppose p is not k-bounded in Σ'. Hence there is a firing sequence σ' with $M'_{\mathsf{init}}[\sigma'\rangle M'$ and $M'(p) > k$. By Prop. 5.3.18, $proj_{T'}(\sigma')$ is a firing sequence of Σ. But since $proj_{T'}(\sigma')$ may only omit t_r, then p is not k-bounded in Σ. □

Intuitively, the next proposition says that if we can remove a token from q right at the start of a firing sequence σ' of Σ', the token is forever gone and all successive transitions do not depend on q.

Proposition 5.3.20 *Let $\sigma' \in T'^{\infty}$ be a transition sequence $t_1 t_2 ...$ such that $t_r\sigma'$ is a firing sequence of Σ' from $M' \in \mathbb{N}^{|P'|}$.*

$\forall i, 1 \leq i < |\sigma'| + 1 : t_i \notin {}^\bullet q \cup q^\bullet$.

Proof Assume σ' fires a transition $t \in {}^\bullet q \cup q^\bullet$. Since q is 1-safe, t_r removes the only token from q. So a transition in $q^\bullet$ is enabled only after a $t \in {}^\bullet q$ is fired. Hence there is a prefix σ'_p of σ' that ends with a transition $t \in {}^\bullet q \setminus q^\bullet$ and does not contain a $t' \in q^\bullet$. Since $M'[t_r\sigma'_p\rangle$, it follows that $M'[\sigma'_p\rangle$. As q is marked after firing $t_r\sigma'_p$, firing only σ'_p places two tokens on q, which contradicts Prop. 5.3.19. □

For the following we assume:
(1) and (2) q is 1-safe in ${\Sigma_e}^{q=1}$ and (3) ${\Sigma_e}^{q=1} \not\models \mathsf{AFG}(q, 1)$.

We come already to the three central propositions that establish

- $proj_{T_k}(\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}})) \subseteq proj_{T_k}(\mathsf{Fs}_{N,\{T_k\}}(M_{\mathsf{init}}))$ and
- $proj_{T_k}(\mathsf{Fs}_{N,\{T_k\}}(M_{\mathsf{init}})) \subseteq proj_{T_k}(\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}}))$ hold.
- For every maximal firing sequence σ' of Σ', we can find a corresponding fair firing sequence that mimics σ' step by step.

For the first two results we have to project the firing sequences of the Consumer-reduced net to T_k, since it has the additional transition t_r. We start with the last result:

Proposition 5.3.21 *Let σ' be a maximal firing sequence on Σ'.*

There is a firing sequence σ of Σ that is fair w.r.t. T_k and $proj_{T_k}(\sigma') = proj_{T_k}(\sigma)$.

Also, σ has a prefix σ_p of length $|\sigma'|$, σ_p contains $proj_{T_k}(\sigma')$ and instead of t_r it fires a transition in T_e ($\sigma(i) \in T_e$ if $\sigma'(i) = t_r$, $1 \leq i < |\sigma| + 1$).

Proof Let M^e_{init} be $M_0^{q=1}|_{P_e}$ and σ^e be a firing sequence of ${\Sigma_e}^{q=1}$ that does not eventually permanently mark q. Such a firing sequence exists by assumption (3). By Prop. 5.3.18 $\sigma'_k := proj_{T_k}(\sigma')$ is a firing sequence of Σ. As σ we fire first σ' but instead of t_r we fire the first transition of σ^e. If σ' has an

occurrence of t_r and is finite, we fire the remainder of σ^e at the end. We can replace t_r by the first transition of σ^e, because to fire t_r q has been marked. Similarly follows that we can fire the remainder of σ^e at the end. In case σ' does not fire t_r and is finite, after firing σ' we fire transitions in T_e as long as there are any enabled.

Suppose that σ is not fair w.r.t. T_k. Hence σ' is finite and $M'_{\sigma'}(q) = 0$ and σ eventually permanently marks q (Prop. 5.3.5). In case σ' does not fire t_r, it follows that $proj_{T_e}(\sigma)$ generates a token, which contradicts 1-safeness of q on ${\Sigma_e}^{q=1}$. If σ' fires t_r then σ has as suffix the remainder of σ^e which by assumption does not eventually permanently mark q. □

Fig. 5.9 shows that for a given maximal firing sequence σ' of Σ', there is not always a corresponding firing sequence σ of Σ that is fair w.r.t. T_k and T_e. So the stronger version of Prop. 5.3.21 —that guarantees the existence of a σ that is fair w.r.t. T_k and T_e—does not hold.

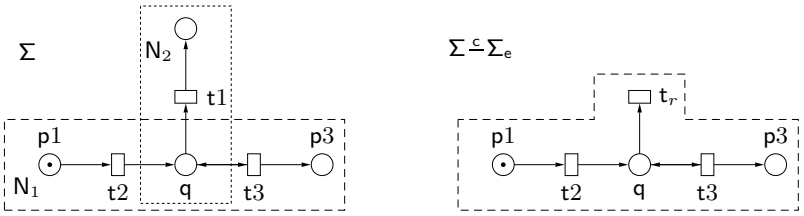


Figure 5.9: Consumer: $proj_{T_k}(\mathsf{Fs}_{N',\mathsf{max}}(M_{\mathsf{init}})) \not\subseteq proj_{T_k}(\mathsf{Fs}_{N,\{T_1,T_2\}}(M_{\mathsf{init}}))$. $\sigma' = t_2t_3t_3...$ is maximal on Σ' but there is no σ with $proj_{T_k}(\sigma) = \sigma'$ that is fair w.r.t T_k and T_e, since q is permanently enabled by σ' after firing t_2 and t_1 has to be fired eventually to achieve fairness w.r.t. T_e.

Proposition 5.3.22 *Let σ be a firing sequence of Σ that is fair w.r.t. T_k.*

There is a maximal firing sequence σ' of Σ' with $proj_{T_k}(\sigma') = proj_{T_k}(\sigma)$.

Proof We show that if $\sigma^k := proj_{T_k}(\sigma)$ is not a maximal firing sequence of Σ' then $\sigma^k t_r$ is. Firstly, σ^k is a firing sequence of Σ_k by Prop. 5.3.10 and thus a firing sequence of Σ'. Suppose σ^k is not a maximal firing sequence of Σ'. Thus σ^k is finite and $M'_{\sigma^k}(q) = 1$ (Prop. 5.3.5). So $\sigma^k t_r$ is a maximal firing sequence of Σ'. □

We now come to the verification and falsification results for the Consumer reduction. Again we cannot verify LTL or CTL properties using the next-time operator X. Consider the LTL property $\psi = \mathsf{A}(\mathsf{G}(p_3, 0) \vee \mathsf{XX}(p_3, 1)$ and the CTL property $\varphi = \mathsf{AX}(\mathsf{AX}(\,(p_3, 1) \vee \mathsf{AG}(p_3, 0)\,)$ on the two nets in Fig. 5.8. Intuitively, both formulas say that after two steps p_3 is marked or p_3 stays unmarked. Both formulas are satisfied by $\Sigma \stackrel{c}{-} \Sigma_\mathrm{e}$, as it first fires t_3, then either t_r is fired and p_3 is never marked, or firing t_4 marks p_3. Σ satisfies neither ψ nor φ, because after firing $t_3 t_2$ the place p_3 is unmarked but it is still possible to marked p_3.

Theorem 5.3.23 *Let Σ_e be a Consumer environment net and Σ be reducible by Σ_e. Let ψ be an $\forall CTL^*$ formula referring to $P \setminus P_e$ only.*

$\Sigma \models \psi$ *fairly w.r.t.* $T_k \quad \Rightarrow \quad \Sigma \stackrel{c}{-} \Sigma_\mathrm{e} \models \psi$.

Proof We show that $(TS_\Sigma, M_{\mathsf{init}})_{\{T_k\}}$ simulates $(TS_{\Sigma'}, M'_{\mathsf{init}})$, which implies that if $\Sigma \models \psi$ fairly w.r.t. T_k then $\Sigma \stackrel{c}{-} \Sigma_\mathrm{e} \models \psi$. We first define $\mathcal{S}$ based on Prop. 5.3.21.

Let $M' \in [M'_{\mathsf{init}}\rangle$ be a marking of Σ'. Let σ'_g be a firing sequence generating M'. Let σ_g be a the firing sequence corresponding to σ'_g as constructed in Prop. 5.3.21, i.e. σ_g equals σ'_g but instead of t_r it fires the first transition of a firing sequence of ${\Sigma_\mathrm{e}}^{q=1}$ that does not eventually permanently mark q, and if σ'_g is finite σ_g may fire a finite suffix in T_e as constructed in Prop. 5.3.21. All such pairs (M_{σ_g}, M') are in $\mathcal{S}$.

We show that all $(M, M') \in \mathcal{S}$ satisfy (L) $L(M) = L'(M')$, and (F) $\forall \mu' \in \Pi_{TS_{\Sigma'}, \mathsf{inf}}(M') : \exists \mu \in \Pi_{TS_\Sigma, \{T_k\}}(M) : (\mu(i), \mu'(i)) \in \mathcal{S}$. As we require that $\mathit{scope}(\psi) \subseteq P_k \setminus \{q\}$, (L) holds by Eq. 5.1a. Let (M, M') be in $\mathcal{S}$, let μ' be an infinite path from M' and let σ' be the corresponding maximal firing sequence of Σ', that is $\mu' = \mathcal{M}(M', \sigma')$.

By definition of $\mathcal{S}$, M is generated by a firing sequence σ_g and there is a corresponding firing sequence σ'_g that generates M'. So $\sigma'_g \sigma'$ is a maximal firing sequence of Σ and σ_g can be extended by a suffix σ according to Prop. 5.3.21 such that $\sigma_g \sigma$ is a fair firing sequence from M_{init}. So $\mu := \mathcal{M}(M, \sigma)$ is a fair path from M. It immediately follows that any marking M_i generated

by prefix σ_p of σ simulates the marking M'_i, generated by a prefix σ'_p of σ' of the same length, i.e. $|\sigma_p| = |\sigma'_p|$. □

The Consumer reduction does not allow for verification of $\forall\text{CTL}^*_{\text{-x}}$ or verification and falsification of $\text{CTL}_{\text{-x}}$ or $\text{CTL}^*_{\text{-x}}$. Consider the example of an Consumer reduction as illustrated in Fig. 5.8 and the $\forall\text{CTL}^*$ (and CTL) formula $\varphi = \mathsf{AF}((p3,1) \vee \mathsf{AG}(p3,0))$. The CTL formula φ is violated—i.e. $\neg\varphi = \mathsf{EG}((p3,0) \wedge \mathsf{EF}(p3,1))$ is satisfied—if there is a path μ on which $p3$ remains unmarked and if a path μ_s from every visited state s of μ eventually leads to a state where $p3$ is marked. The original net in Fig. 5.8 violates φ: It first fires t_3. Firing then infinitely often t_1t_2 never marks $p3$ but from every generated state we can fire t_4 or t_1t_4, respectively, to mark $p3$. The reduced net satisfies φ, because the only maximal firing sequence that does not eventually mark $p3$ fires t_r and after firing t_r it is not possible to mark $p3$.

We can verify and falsify $\text{LTL}_{\text{-x}}$.

Theorem 5.3.24 *Let Σ_e be a Consumer environment net and Σ be reducible by Σ_e. Let ψ be an $\text{LTL}_{\text{-x}}$ formula referring to $P \setminus P_e$ only.*

$\Sigma \underline{^c} \Sigma_\text{e} \models \psi \;\Leftrightarrow\; \Sigma \models \psi$ *fairly w.r.t. T_k.*

Proof As $\Sigma \models \psi$ fairly w.r.t. $T_k \Rightarrow \Sigma \underline{^c} \Sigma_\text{e} \models \psi$ holds for any $\forall\text{CTL}^*$ formula, we only need to show $\Sigma \underline{^c} \Sigma_\text{e} \models \psi \Rightarrow \Sigma \models \psi$ fairly w.r.t. T_k. Let us assume that $\Sigma \not\models \psi$ fairly w.r.t. T_k. Hence there is a firing sequence σ of Σ that is fair w.r.t. T_k and $\mathcal{M}(M_{\mathsf{init}}, \sigma) \not\models \psi$. By Prop. 5.3.22, there is a maximal firing sequence σ' of Σ' with $proj_{T_k}(\sigma) = proj_{T_k}(\sigma')$. Since $\mathcal{M}(M_{\mathsf{init}}, \sigma) \not\models \psi$, it follows by Prop. 5.3.6 that $\mathcal{M}(M'_{\mathsf{init}}, \sigma') \not\models \psi$ and hence $\Sigma' \not\models \psi$.

□

5.3.4 Producer Reduction

To show that an reducible environment Σ_e is a Producer, we check that ${\Sigma_\text{e}}^{q=0} \models \mathsf{AFG}(q,1)$, i.e. ${\Sigma_\text{e}}^{q=0}$ is guaranteed to eventually permanently mark q. For the following we denote $\Sigma \underline{^p} \Sigma_\text{e} = (P_k, T_k, W_k, M^{q=1}_{\mathsf{init,k}})$ as Σ'.

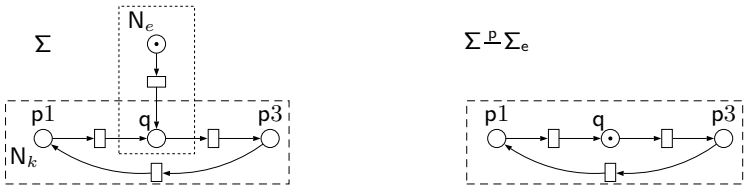


Figure 5.10: Example of a Producer reduction

For the following we assume:
(1) q is 1-safe in Σ and (2') ${\Sigma_e}^{q=0} \models \mathsf{EF}(q, 1)$.

Proposition 5.3.25 $M_{\mathsf{init}}(q) = 0$.

Proof Suppose the initial marking places a token on q, $M_{\mathsf{init}}(q) = 1$. Because we assume (2'), there is a firing sequence σ^e with $\Delta(\sigma^e, q) \geq 1$ in ${\Sigma_e}^{q=0}$. Since σ^e is also a firing sequence of Σ, this contradicts (1). □

Proposition 5.3.26 *q is 1-safe in Σ'.*

Proof Suppose q is not 1-safe in Σ'. Hence there is a firing sequence σ' with $\Delta(\sigma', q) \geq 1$. Since ${\Sigma_e}^{q=0} \models \mathsf{EFG}(q, 1)$, there is a firing sequence σ^e that can be fired at M_{init} and marks q. So we can fire $M_{\mathsf{init}}[\sigma^e\sigma'\rangle$ to generate more than one token on q in Σ, which contradicts (1), the 1-safeness of q in Σ. □

Proposition 5.3.27 *Let σ be a firing sequence of Σ such that $proj_{T_k}(\sigma)$ is a firing sequence of Σ'.*

$proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_e}^{q=0}$.

Proof Let us assume that σ is a firing sequence of Σ but $\sigma^e := proj_{T_e}(\sigma)$ is not a firing sequence of ${\Sigma_e}^{q=0}$. By Prop. 5.3.3, there are prefixes σ_p of σ and σ_p^e of σ^e such that $M_{\sigma_p}(q) > M^e_{\sigma^e_p}(q)$. Since $M^e_{\mathsf{init}}(q) = 0$ by definition and $M_{\mathsf{init}}(q) = 0$ by Prop. 5.3.25, it follows that $\Delta(\sigma_p, q) > \Delta(\sigma^e_p, q)$ and hence $\Delta(proj_{T_k}(\sigma_p), q) > 0$. But then q is not 1-safe in Σ', which contradicts Prop. 5.3.26. □

Proposition 5.3.28 *Let σ be a firing sequence of Σ.*
$proj_{T_k}(\sigma)$ is a firing sequence of Σ'.

Proof If the above does not hold, then there are prefixes σ_p of σ and σ_p^k of $proj_{T_k}(\sigma)$ such that $M_{\sigma_p}(q) > M'_{\sigma_p^k}(q)$ by Prop. 5.3.3. Since $M'_{\mathsf{init}}(q) = 1$, it follows that $1 + \Delta(\sigma_p^k, q) < 0 + \Delta(\sigma_p, q)$ holds. But then $\sigma_p^e := proj_{T_e}(\sigma_p)$ must have generated a token on q, $\Delta(\sigma_p^e, q) > 1$. By Prop. 5.3.27, σ_p^e is a firing sequence of ${\Sigma_e}^{q=0}$, which contradicts the 1-safeness of q in ${\Sigma_e}^{q=0}$ (cf. Prop. 5.3.2). □

The next proposition says that if σ is fair w.r.t. T_e and generates a token on q which Σ^k does not remove, $\sigma^e = proj_{T_e}(\sigma)$ is maximal on ${\Sigma_e}^{q=0}$. Intuitively, this holds because σ^e generates the token and then behaves undisturbed by transitions in T_k, as they do not remove the token from q.

Proposition 5.3.29 *Let $\sigma = \sigma_1\sigma_2$ be a firing sequence of Σ from M_{init} that is fair w.r.t. T_e. Let σ_1 be such that $M_{\sigma_1}(q) = 1$ and let σ_2 be such that for $\sigma_2^k := proj_{T_k}(\sigma_2)$ holds $\forall i, 1 \leq i < |\sigma_2^k| + 1 : \Delta(\sigma_2^k(i), q) = 0$.*
$proj_{T_e}(\sigma)$ is a maximal firing sequence of ${\Sigma_e}^{q=0}$.

Proof Let M^e_{init} be $M^{q=0}_{\mathsf{init}}|_{P_e}$. By Prop. 5.3.27 and Prop. 5.3.28 holds, that $\sigma^e := proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_e}^{q=0}$. Let us assume σ^e is not maximal. Thus σ^e is finite and $M^e_{\sigma^e}(q) = 1$ and σ does not eventually permanently mark q (Prop. 5.3.4). Since $M^e_{\sigma^e}(q) = 1$, it follows that $\Delta(\sigma^e, q) = 1$. Let σ_p be a prefix of σ_2 with $M_{\sigma_1\sigma_p}(q) = 0$ containing σ^e. Such a prefix exists because q is not permanently marked by σ. It follows that $0 = M_{\mathsf{init}}(q) + \Delta(\sigma_1\sigma_p, q)$. Since σ_2^k does not change the token count on q, $0 = \Delta(\sigma_e, q) + \Delta(proj_{T_k}(\sigma_1), q)$. It follows that $\Delta(proj_{T_k}(\sigma_1), q) = -1$. But since we assume that $M_{\sigma_1}(q) = 1 = \Delta(proj_{T_k}(\sigma_1), q) + \Delta(proj_{T_e}(\sigma_1), q)$, $proj_{T_e}(\sigma_1) = +2$, which contradicts 1-safeness of q in $\Sigma^{q=0}$. □

For the following we assume: (1) and (2) ${\Sigma_e}^{q=0} \models \mathsf{AFG}(q, 1)$.

The following lemma is the Uniqueness Lemma for the Producer reduction (cf. Lemma 5.3.11 for the Borrower reduction).

Lemma 5.3.30 (Uniqueness Lemma) *Let σ_1, σ_2 be firing sequences of Σ.*
If $\Delta(\sigma_1, p) = \Delta(\sigma_2, p), \forall p \in P$,
then $\Delta(proj_{T_k}(\sigma_1), p) = \Delta(proj_{T_k}(\sigma_2), p), \forall p \in P_k$.

Proof Let σ_1^k denote $proj_{T_k}(\sigma_1)$ and σ_2^k denote $proj_{T_k}(\sigma_2)$. Similarly, let σ_1^e be $proj_{T_e}(\sigma_1)$ and σ_2^e be $proj_{T_e}(\sigma_2)$. As $\Delta(\sigma_1, p) = \Delta(\sigma_2, p)$, $\forall p \in P$, and transitions in T_e cannot change the token count on places in P_k, it follows that $\Delta(\sigma_1^k, p_k) = \Delta(\sigma_2^k, p_k)$, $\forall p_k \in P_k \setminus \{q\}$ holds. Let us assume that $\Delta(\sigma_1^k, q) \neq \Delta(\sigma_2^k, q)$. It follows that also $\Delta(\sigma_1^e, q) \neq \Delta(\sigma_2^e, q)$, because $\Delta(\sigma_1^e, q) + \Delta(\sigma_1^k, q) = \Delta(\sigma_2^e, q) + \Delta(\sigma_2^k, q)$ by assumption. Since σ_1^k and σ_2^k are both firing sequences of Σ' by Prop. 5.3.28 and q is 1-safe in Σ', it follows that $\Delta(\sigma_1^k), \Delta(\sigma_2^k) \in \{-1, 0\}$. Without loss of generality let $\Delta(\sigma_1^e, q) = 1$ and $\Delta(\sigma_2^e, q) = 0$. Since Σ_e is a producer and thus satisfies $\mathsf{AFG}(q, 1)$, a firing sequence σ_g^e is enabled that eventually marks q after firing σ_2^e. σ_g^e is also a firing sequence from M_{σ_2}, as $M_{\sigma_2}|_{P_k \setminus \{q\}} = M^e_{\sigma_2^e}|_{P_k \setminus \{q\}}$ by Eq. 5.1b and $M_{\sigma_2}(q) \geq 0 = M^e_{\sigma_2^e}(q)$. But then σ_g^e is also a firing sequence from M_{σ_1} and hence $proj_{T_e}(\sigma_1 \sigma_g^e) = \sigma_1^e \sigma_g^e$ is a firing sequence of $\Sigma_e^{q=0}$ that generates two tokens on q. This contradicts 1-safeness of q in ${\Sigma_\mathrm{e}}^{q=0}$ (Prop. 5.3.2). □

We now show the two main propositions that imply

- $\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}}) \subseteq proj_{T_k}(\mathsf{Fs}_{N,\{T_k,T_e\}}(M_{\mathsf{init}}))$ and
- $proj_{T_k}(\mathsf{Fs}_{N,\{T_k\}}(M_{\mathsf{init}})) \subseteq \mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}})$.

Again this means that

$$\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}}) = proj_{T_k}(\mathsf{Fs}_{N,\{T_k\}}(M_{\mathsf{init}})) = proj_{T_k}(\mathsf{Fs}_{N,\{T_k,T_e\}}(M_{\mathsf{init}}))$$

holds (cf. page 94).

Proposition 5.3.31 *Let $\sigma' = \sigma'_1 \sigma'_2$ be a maximal firing sequence of Σ'. Let σ_1 be a firing sequence of Σ with $proj_{T_k}(\sigma_1) = \sigma'_1$.*

If σ_1 is finite, then there is a firing sequence σ_2 of Σ such that $\sigma_1 \sigma_2$ is fair w.r.t. T_k and T_e and $proj_{T_k}(\sigma_2) = \sigma'_2$.

Proof The proof is in most parts equal to the proof of Prop. 5.3.14 for the Borrower reduction on page 96.

In the first if-Block (`line 10-16`) we use a maximal firing sequence σ^e on ${\Sigma_e}^{q=0}$ instead of on ${\Sigma_e}^{q=1}$. We use that ${\Sigma_e}^{q=0} \models \mathsf{AFG}(q,1)$ instead of ${\Sigma_e}^{q=1} \models \mathsf{AFG}(q,1)$. Except for these changes the proof is the same. □

Proposition 5.3.32 *Let σ be a firing sequence of Σ that is fair w.r.t. T_k.*
$proj_{T_k}(\sigma)$ is a maximal firing sequence of Σ'.

Proof By Prop. 5.3.28, $\sigma' := proj_{T_k}(\sigma)$ is a firing sequence of Σ'. Let us assume that σ' is not maximal but σ fair w.r.t. T_k. Thus σ' is finite and $M'_{\sigma'}(q) = 1$ and σ does not eventually permanently mark q. Hence $\Delta(\sigma', q) = 0$. Because σ is maximal and only finitely many transitions of T_k are fired, it follows that σ is fair w.r.t. T_e. By Prop. 5.3.29, $\sigma^e = proj_{T_e}(\sigma)$ is a maximal firing sequence of ${\Sigma_e}^{q=0}$. Since ${\Sigma_e}^{q=0} \models \mathsf{AFG}(q,1)$, we can divide σ^e into a prefix σ^e_p that marks q and a suffix σ^e_s that does not change the token count on q ($\sigma^e = \sigma^e_p\sigma^e_s$ and $\mathcal{M}(M^e_{\sigma_p}, \sigma^e_s) \models \mathsf{G}(q,1)$). Let σ_p be a prefix of σ that contains σ' and σ^e_p. It follows that $\Delta(\sigma_p, q) = 1$ holds. But any prefix σ_{p_2} that extends σ_p also satisfies $\Delta(\sigma_{p_2}, q) = 1$, since σ^e_s never effects q's token count. Consequently, σ eventually permanently marks q, which is a contradiction. □

Theorem 5.3.33 *Let Σ_e a Producer environment net and Σ be reducible by Σ_e. Let φ be a CTL$^*_{-\mathsf{X}}$ formula referring to $P \setminus P_e$ only.*
$\Sigma \overset{p}{-} \Sigma_e \models \varphi \;\Leftrightarrow\; \Sigma \models \varphi$ *fairly w.r.t. T_k*
$\Sigma \overset{p}{-} \Sigma_e \models \varphi \;\Leftrightarrow\; \Sigma \models \varphi$ *fairly w.r.t. T_k and T_e.*

Proof This follows analogously to the proof of Theorem 5.3.16 on page 100 by Prop. 5.3.28, the Producer's Uniqueness Lemma 5.3.30, Prop. 5.3.31 and Prop. 5.3.32. □

The Producer reduction does not allow for verification or falsification of CTL or LTL using X. Consider the properties $\varphi_1 = \mathsf{AX}(p_3, 1)$, which is expressible in CTL and LTL, and $\psi = \mathsf{AXX}(p_3, 1)$, expressible in LTL. φ_1 holds on $\Sigma \overset{p}{-} \Sigma_e$ in Fig. 5.10 but not on Σ, as Σ has to fire two transitions to mark p_3. Hence

verification of LTL and CTL as well as falsification of CTL is not possible via $\Sigma \underline{^p} \Sigma_e$. ψ holds on Σ but not on $\Sigma \underline{^p} \Sigma_e$, as only p_1 is marked after two transition firings. This shows that LTL cannot be falsified via $\Sigma \underline{^p} \Sigma_e$.

5.3.5 Dead End Reduction

We defined an environment Σ_e to be a Dead End, if q is not 1-safe in ${\Sigma_e}^{q=1}$, but ${\Sigma_e}^{q=0}$'s behaviour does not change the token count on q (${\Sigma_e}^{q=0} \models \mathsf{AG}(q, 0)$). Technically the Dead End reduction is not necessary to preserve LTL$_{-\mathsf{X}}$ properties: Although we defined the Dead End-reduced in Def. 5.2.2 as $\Sigma \underline{^d} \Sigma_e = (\tilde{P}, \tilde{T}, W_k|_{(\tilde{P},\tilde{T})}, M_{\mathsf{init,k}}|_{\tilde{P}})$ where $\tilde{P} = P_k \setminus \{q\}$ and $\tilde{T} = T_k \setminus (q^\bullet \cup {}^\bullet q)$, we show here that Σ' can be either $\Sigma \underline{^b} \Sigma_e$ or also $\Sigma \underline{^c} \Sigma_e$. In a second step, we show that q is never marked in Σ and Σ', which justifies to remove $q^\bullet \cup {}^\bullet q$.

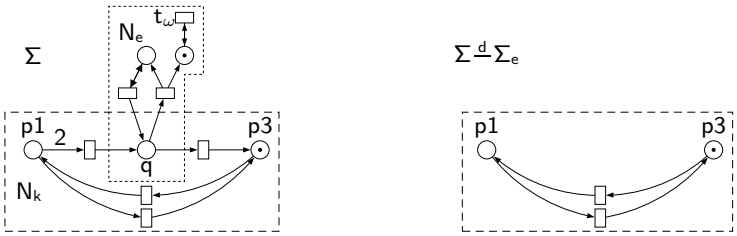


Figure 5.11: Example of a Dead End reduction

For this special case it is convenient to use another effect equation expressing that if a firing sequence σ has no occurrences of ${}^\bullet q \cup q^\bullet$, then its effect on all places in P_k is completely determined by transitions in T_k. Let σ be a transition sequence in $(T \setminus ({}^\bullet q \cup q^\bullet))^*$ and σ' be a transition sequence in T'^*.

$$\Delta(\sigma, p) = \Delta(\sigma', p),\ \forall p \in P_k. \tag{5.2}$$

The following proposition says, that if σ is a firing sequence of Σ and every prefix of $proj_{T_e}(\sigma)$ does not change the token count on q, then $proj_{T_k}(\sigma)$ is a firing sequence of Σ^k.

Proposition 5.3.34 *Let σ be a firing sequence of Σ.*

If every prefix σ_p^e of $proj_{T_e}(\sigma)$ satisfies $\Delta(\sigma_p^e, q) = 0$, then $proj_{T_k}(\sigma)$ is a firing sequence of Σ^k.

Proof As transitions in T_e do not change the token count of q, it follows by Eq. 5.1a, that the token count on P_k is completely determined by transitions in T_k. □

For the following we assume: (1) q is 1-safe in Σ and
(2) q is not 1-safe in ${\Sigma_\mathrm{e}}^{q=1}$ and (3) ${\Sigma_\mathrm{e}}^{q=0} \models \mathsf{AG}(q, 0)$.

The first observation, we make, is that the kernel Σ_k does not generate tokens on q.

Proposition 5.3.35 *Let σ be a finite firing sequence of Σ.*
$\Delta(proj_{T_k}(\sigma), q) \leq 0$

Proof Let us assume that $\sigma^k := proj_{T_k}(\sigma)$ generates a token on q, $\Delta(\sigma^k, q) > 0$. As q is 1-safe in Σ, $proj_{T_e}(\sigma)$ does not generate a token on q. So by Prop. 5.3.34, we can fire σ^k on Σ^k and hence on Σ. Since q is not 1-safe in ${\Sigma_\mathrm{e}}^{q=1}$, there is a firing sequence σ^e that generates more than one token, $\Delta(\sigma^e, q) \geq 1$. Hence the firing $\sigma^k\sigma^e$ from M_{init} generates a marking on Σ with more than one token on q. □

Proposition 5.3.36 *Let σ be a firing sequence of Σ.*
$proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_\mathrm{e}}^{q=0}$.

Proof If σ is a firing sequence of Σ but $\sigma^e := proj_{T_e}(\sigma)$ is not a firing sequence of ${\Sigma_\mathrm{e}}^{q=0}$, then there are prefixes σ_p of σ and σ_p^e of σ^e such that $M_{\sigma_p}(q) > M^e_{\sigma_p^e}(q)$ by Prop. 5.3.3. But then $\Delta(proj_{T_k}(\sigma_p), q) > 0$, which contradicts Prop. 5.3.35. □

Proposition 5.3.37 *Let σ be a firing sequence of Σ.*
σ does not fire a transition in ${}^\bullet q \cup q^\bullet$.

Proof By Prop. 5.3.35, $\Delta(proj_{T_k}(\sigma), q) \leq 0$. As $\sigma^e := proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_\mathrm{e}}^{q=0}$, it follows by assumption (3) that also σ^e does not generate a token on q. As consequence also a $t \in q^\bullet$ is never enabled. □

Proposition 5.3.38 *Let σ be a firing sequence of Σ.*
$proj_{T_k}(\sigma)$ is a firing sequence of Σ_k.

Proof By Prop. 5.3.35 it holds that $\Delta(proj_{T_k}(\sigma), q) \leq 0$. So $proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_e}^{q=0}$ by Prop. 5.3.36. Since ${\Sigma_e}^{q=0} \models \mathsf{AG}(q, 0)$, it follows by Prop. 5.3.34 that $proj_{T_k}(\sigma)$ is a firing sequence of Σ_k. □

We have shown in Prop. 5.3.37, that Σ never fires a transition in ${}^\bullet q \cup q^\bullet$. Now we show that also Σ' does not fire a transition in ${}^\bullet q \cup q^\bullet$. This justifies why we can replace Σ_e by the non-producing summary nets, i.e. the borrower or consumer summary. No transition in $q^\bullet$ is ever fired in Σ and in Σ', so it does not matter if there is a transition.

Proposition 5.3.39 *Let σ' be a firing sequence of Σ'.*

σ' does not fire transitions in ${}^\bullet q \cup q^\bullet$.

Proof If Σ' is $\Sigma \stackrel{b}{-} \Sigma_e$, every firing sequence σ' is also a firing sequence of Σ by Prop. 5.3.1. If Σ' is $\Sigma \stackrel{c}{-} \Sigma_e$, $proj_{T_k}(\sigma')$ is a firing sequence of Σ by Prop. 5.3.18. By Prop. 5.3.37 no firing sequence of Σ fires a transition in ${}^\bullet q \cup q^\bullet$. Hence it also follows that also $\Sigma \stackrel{c}{-} \Sigma_e$ never fires t_r. □

Propositions 5.3.37 and 5.3.39 justify why the Dead End reductions remove ${}^\bullet q \cup q^\bullet$.

We now show correspondences between maximal firing sequences of Σ' and fair firing sequences of Σ.

- We show $\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}}) \subseteq proj_{T_k}(\mathsf{Fs}_{N,\{T_k,T_e\}}(M_{\mathsf{init}}))$ and
- $proj_{T_k}(\mathsf{Fs}_{N,\{T_k\}}(M_{\mathsf{init}})) \subseteq \mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}})$.

Again this means that

$$\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}}) = proj_{T_k}(\mathsf{Fs}_{N,\{T_k\}}(M_{\mathsf{init}})) = proj_{T_k}(\mathsf{Fs}_{N,\{T_k,T_e\}}(M_{\mathsf{init}}))$$

holds (cf. page 94).

- We show a stepwise correspondence: For every maximal firing sequence σ' of Σ' there is a fair firing sequence σ that starts with σ'.

Proposition 5.3.40 *Let σ' be a maximal firing sequence of Σ'.*

There is a firing sequence σ of Σ that is fair w.r.t. T_k and T_e and $proj_{T_k}(\sigma) = \sigma'$.

Proof By Prop. 5.3.1, σ' is a firing sequence of Σ. We construct σ by firing in turn a transition of σ' and an enabled transition of T_e, if it exists, until all of σ' has been fired and no transitions of T_e are enabled. We now show that σ is indeed a firing sequence. M_{init} does not mark q, since q is 1-safe in Σ but not in ${\Sigma_{\mathrm{e}}}^{q=1}$. As the enabled transitions in T_e do not change the token count on q by assumption (3) and Prop. 5.3.36, σ is a firing sequence of Σ. By construction, σ is fair w.r.t. T_e. From maximality of σ' it follows that σ is fair w.r.t. T_k. □

As we will show that Σ' allows for falsification of $\forall\mathrm{CTL}^*$ formulas assuming that Σ is fair w.r.t. T_k, we need in addition to Prop. 5.3.40 the following proposition.

Proposition 5.3.41 *Let σ' be a maximal firing sequence of Σ'.*

There is a firing sequence σ of Σ that is fair w.r.t. T_k, starts with σ' and $proj_{T_k}(\sigma) = \sigma'$.

Proof By Prop. 5.3.1, σ' is a firing sequence of Σ. After firing σ' on Σ we fire enabled transitions of T_e as long as there are any. As transitions of T_e do not change the token count on T_k (Prop. 5.3.37), maximality of σ' implies that σ is fair w.r.t. T_k. □

Proposition 5.3.42 *Let σ be a firing sequence of Σ that is fair w.r.t. T_k.*

$proj_{T_k}(\sigma)$ is a maximal firing sequence of Σ'.

Proof By Prop. 5.3.38, $\sigma' := proj_{T_k}(\sigma)$ is a firing sequence of Σ'. Let us assume σ' is not maximal. Hence σ' is finite and can be extended by a transition $t \in T'$, i.e. $M'_{\sigma'}[t\rangle$. Let σ_p be the smallest prefix of σ that contains σ'. By Eq. 5.2 and Prop. 5.3.37 it follows that σ permanently enables t and hence σ is not fair w.r.t. T_k. □

Now we present our main result for the Dead End reduction. $\Sigma \underline{\overset{d}{-}} \Sigma_{\mathrm{e}}$ can be used for verification and falsification of a $\mathrm{CTL}^*_{\text{-}\mathsf{X}}$ formula φ.

Theorem 5.3.43 *Let Σ_{e} be a Dead End environment and Σ be reducible by Σ_{e}. Let φ be a $CTL^*_{\text{-}\mathsf{X}}$ formula referring to $P \setminus P_e$ only.*

$\Sigma \underline{\overline{d}} \Sigma_{\mathrm{e}} \models \varphi \;\Leftrightarrow\; \Sigma \models \varphi$ *fairly w.r.t.* T_k *and* T_e.
$\Sigma \underline{\overline{d}} \Sigma_{\mathrm{e}} \models \varphi \;\Leftrightarrow\; \Sigma \models \varphi$ *fairly w.r.t.* T_k.

Proof This follows analogously to the proof of Theorem 5.3.16 on page 100 by Prop. 5.3.38, Prop. 5.3.40 and Prop. 5.3.42. For the Dead End reduction the markings on Σ and Σ' are uniquely corresponds according to Eq. 5.2 and Prop. 5.3.37 and Prop. 5.3.39 and Eq. 5.2. □

A Dead End reduced cannot be used to verify LTL or CTL, or to falsify CTL properties using the next-time operator. Consider the two nets in Fig. 5.11. The property $\varphi = \mathsf{AX}(p_1, 1)$ is expressible as CTL and LTL formula. φ is satisfied by $\Sigma \underline{\overline{d}} \Sigma_{\mathrm{e}}$ but Σ does not satisfy φ because of firings of t_ω in Σ_{e}. We also cannot falsify LTL properties assuming fairness w.r.t. T_k and T_e. To see this, consider the LTL property $\psi = \mathsf{AF}((p_3, 1) \wedge \mathsf{XX}(p3, 0))$. ψ does not hold on $\Sigma \underline{\overline{d}} \Sigma_{\mathrm{e}}$ but it does hold on Σ assuming fairness w.r.t. T_e, as t_ω is permanently enabled it has to be fired eventually. We can falsify that Σ satisfies an $\forall$CTL* property fairly w.r.t T_k using $\Sigma \underline{\overline{d}} \Sigma_{\mathrm{e}}$ as the following theorem states.

Theorem 5.3.44 *Let* Σ_{e} *be a Dead End environment net and* Σ *be reducible by* Σ_{e}. *Let* ψ *be an* $\forall CTL^*$ *formula referring to* $P \setminus P_e$ *only.*
$\Sigma \models \psi$ *fairly w.r.t.* $T_k \;\Rightarrow\; \Sigma \underline{\overline{d}} \Sigma_{\mathrm{e}} \models \psi$.

Proof The proof is analogously done to the proof of Theorem 5.3.17. We use Prop. 5.3.36 instead of Prop. 5.3.7 and Prop. 5.3.41 instead of Prop. 5.3.15. □

5.3.6 Unreliable Producer Reduction

Σ_{e} is an Unreliable Producer if ${\Sigma_{\mathrm{e}}}^{q=0}$ eventually permanently marks q at some executions, otherwise never marks q. Formally, we expressed this behaviour in Def. 5.2.2 as ${\Sigma_{\mathrm{e}}}^{q=0} \not\models \mathsf{AG}(q, 0)$, ${\Sigma_{\mathrm{e}}}^{q=0} \not\models \mathsf{AFG}(q, 1)$ and ${\Sigma_{\mathrm{e}}}^{q=0} \models \mathsf{AG}((q, 1) \Rightarrow \mathsf{FG}(q, 1))$. In the following we denote $\Sigma \underline{\overline{up}} \Sigma_{\mathrm{e}} = (P_k \uplus \{p_p\}, T_k \uplus \{t_c, t_p\}, W_k \uplus \{(p_p, t_p) \mapsto 1, (t_p, q) \mapsto 1, (p_p, t_c) \mapsto 1\}, M_{\mathsf{init,k}} \uplus \{p_p \mapsto 1\})$ also as Σ'.

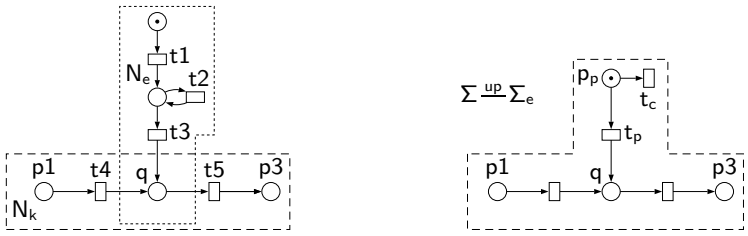


Figure 5.12: Example of an Unreliable Producer reduction

An Unreliable Producer either does not generate any token on q or it behaves as a Producer and eventually permanently marks q. A Producer-reduced net places just a token on q. If we fire t_p in the Unreliable Producer-reduced net, we can emulate this. In the following proof of $\mathrm{CTL}^*_{-\mathsf{X}}$ preservation, we can therefore reuse parts shown for the Producer reduction.

For the following we assume:
(1) q is 1-safe in Σ and (2) ${\Sigma_e}^{q=0} \not\models \mathsf{AG}(q, 0)$.

Basically the following proposition says, that the kernel cannot produce a token on q. The reason simply is that the environment can produce a token on q and q is 1-safe in Σ. The constraint that $t_p proj_{T_k}(\sigma)$ has to be a firing sequence of Σ', can later be dropped as this is the case for every firing sequence of Σ (Prop. 5.3.47).

Proposition 5.3.45 *Let σ be a finite firing sequence of Σ such that $t_p proj_{T_k}(\sigma)$ is a firing sequence of Σ'.*

$\Delta(proj_{T_k}(\sigma), q) \leq 0$

Proof Let σ^k denote $proj_{T_k}(\sigma)$. Since ${\Sigma_e}^{q=0} \not\models \mathsf{AG}(q, 0)$, there is a firing sequence σ^e of ${\Sigma_e}^{q=0}$ that generates a token on q. We can fire σ^e from M_{init} to mark q. By Eq. 5.1a and since $t_p\sigma^k$ is a firing sequence of Σ', we can hence fire $\sigma^e\sigma^k$ from M_{init}. If we assume that σ^k generates a token, $\Delta(\sigma^k, q) \geq 1$, q is not 1-safe in Σ (contradiction to (1)). □

Proposition 5.3.46 *Let σ be a firing sequence of Σ such that $t_p proj_{T_k}(\sigma)$ is a firing sequence of Σ'.*

$proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_e}^{q=0}$.

Proof Let M^e_{init} be $M_{\mathsf{init}}|_{P_e}$. If Prop. 5.3.46 does not hold, there are prefixes σ_p of σ and σ^e_p of $proj_{T_e}(\sigma)$ such that $M_{\sigma_p}(q) > M^e_{\sigma^e_p}(q)$ by Prop. 5.3.3. Since $M^e_{\mathsf{init}}(q) = 0$ and $M_{\mathsf{init}}(q) = 0$ by Prop. 5.3.25, it follows that $\Delta(proj_{T_k}(\sigma), q) > 0$. This contradicts Prop. 5.3.45. □

Given we produce a token on q by firing t_p, the projection onto T_k of any firing sequence of Σ can be executed on Σ'.

Proposition 5.3.47 *Let σ be a firing sequence of Σ.*

$t_p proj_{T_k}(\sigma)$ is a firing sequence of Σ'.

Proof The proof is by induction on the length l of σ. The case $l = 0$ follows trivially.

$l \to l+1$: Let σt be a firing sequence of length $l+1$. We denote $proj_{T_k}(\sigma)$ as σ^k. The case $t \in T_e$ follows directly by the induction hypothesis. Let us assume that $t \in T_k$ and t is not enabled on Σ' after firing $t_p\sigma^k$, $\neg M'_{t_p\sigma^k}[t\rangle$. By Eq. 5.1a it follows that $t \in q^\bullet$ and $M'_{t_p\sigma^k}(q) = 0$ and $M_\sigma(q) = 1$. Since $M'_{\mathsf{init}}(q) = 0$ it follows that $\Delta(t_p\sigma^k, q) = 0$ and hence $\Delta(\sigma^k, q) = -1$. According to Prop. 5.3.25, $M_{\mathsf{init}}(q) = 0$. Since $M_\sigma(q) = 1$, it thus follows that $\Delta(\sigma, q) = +1$ and $\Delta(proj_{T_e}(\sigma), q) = +2$. By Prop. 5.3.46, $proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_e}^{q=0}$. This contradicts 1-safeness of q in ${\Sigma_e}^{q=0}$. □

The next proposition is a rather technical result stating that if a firing sequence on Σ contains only finite kernel behaviour and if the kernel does only temporarily change the token count on q, σ contains maximal behaviour of Σ_e, since Σ_e can basically behave undisturbed by Σ_k. This result will be used to show that σ has a corresponding maximal firing sequence on Σ'.

Proposition 5.3.48 *Let σ be a firing sequence that is fair w.r.t. T_k.*

If $proj_{T_k}(\sigma)$ is finite and $\Delta(proj_{T_k}(\sigma), q) = 0$, then $proj_{T_e}(\sigma)$ is a maximal firing sequence of ${\Sigma_e}^{q=0}$.

Proof Let M^e_{init} be ${M^{q=0}_{\mathsf{init}}}|_{P_e} = M_{\mathsf{init}}|_{P_e}$. By Prop. 5.3.46 and Prop. 5.3.47, $\sigma^e := proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_e}^{q=0}$. As $proj_{T_k}(\sigma)$ is finite, it follows

that σ is fair w.r.t. T_e. Let us assume that σ^e is not maximal but σ is fair w.r.t. T_k. Consequently, σ^e and thus σ are finite and $M^e_{\sigma^e}(q) = 1$. σ does not mark q, $M_\sigma(q) = 0$, according to Prop. 5.3.4. As we assume that $\Delta(proj_{T_k}(\sigma), q) = 0$, it follows that $\Delta(\sigma^e, q) = 0$. But this contradicts $M^e_{\sigma^e}(q) = 1$. □

Again a technical result follows. We have already seen a version of this proposition on page 110 for the Producer. It says that if σ is fair w.r.t. T_e and generates a token which Σ^k does not remove, $\sigma^e := proj_{T_e}(\sigma)$ is maximal on ${\Sigma_e}^{q=0}$. Intuitively, this holds because σ^e generates the token and then behaves undisturbed by transitions in T_k, as they do not remove the token from q.

Proposition 5.3.49 *Let $\sigma = \sigma_1\sigma_2$ be a firing sequence of Σ from M_{init} that is fair w.r.t. T_e. Let σ_1 be such that $M_{\sigma_1}(q) = 1$ and let σ_2 be such that for $\sigma_2^k := proj_{T_k}(\sigma_2)$ holds $\forall i, 1 \leq i < |\sigma_2^k| + 1 : \Delta(\sigma_2^k(i), q) = 0$.*

$proj_{T_e}(\sigma)$ is a maximal firing sequence of ${\Sigma_e}^{q=0}$.

Proof This follows analogously to Prop. 5.3.29 on page 110 by Prop. 5.3.46 and Prop. 5.3.47. □

For the following we assume: (1), (2) and
(3) ${\Sigma_e}^{q=0} \models \mathsf{AG}((q,1) \Rightarrow \mathsf{FG}(q,1))$.

We now show the two main propositions for Unreliable Producers. They imply

- $proj_{T_k}(\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}})) \subseteq proj_{T_k}(\mathsf{Fs}_{N,\{T_k,T_e\}}(M_{\mathsf{init}}))$ and
- $proj_{T_k}(\mathsf{Fs}_{N,\{T_k\}}(M_{\mathsf{init}})) \subseteq proj_{T_k}(\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}}))$.

Again this means that

$$\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}}) = proj_{T_k}(\mathsf{Fs}_{N,\{T_k\}}(M_{\mathsf{init}})) = proj_{T_k}(\mathsf{Fs}_{N,\{T_k,T_e\}}(M_{\mathsf{init}}))$$

holds (cf. page 94).

Proposition 5.3.50 *Let σ be a firing sequence of Σ that is fair w.r.t. T_k.*

There is a maximal firing sequence σ' of Σ' with $proj_{T_k}(\sigma') = proj_{T_k}(\sigma)$.

Proof We denote $proj_{T_k}(\sigma)$ as σ^k. We show that $t_c\sigma^k$ or $t_p\sigma^k$ is a maximal firing sequence of Σ'. By Prop. 5.3.47, $t_p\sigma^k$ is a firing sequence of Σ' and it holds $proj_{T_k}(t_p\sigma^k) = proj_{T_k}(\sigma)$.

Suppose $t_p\sigma^k$ is not maximal but σ is fair w.r.t. T_k. By Prop. 5.3.5 σ^k is finite and $M'_{t_p\sigma^k}(q) = 1$ and σ does not eventually permanently mark q. Since $M'_{\mathsf{init}}(q) = 0$, $t_p\sigma^k$ generates a token on q. From $\Delta(t_p\sigma^k, q) = 1$ it follows $\Delta(\sigma^k, q) = 0$.

We replace t_p by t_c and show that $t_c\sigma^k$ is still a firing sequence of Σ'. Let us assume that $t_c\sigma^k$ is not a firing sequence. Hence σ^k fires a transition $t \in T_k$ that consumes from q. Let $\sigma_p^k t$ be the minimal prefix of σ^k that contains a transition $t \in q^\bullet$. Let σ_p be the minimal prefix of σ with $proj_{T_k}(\sigma_p t) = \sigma_p^k t$. It follows that after firing σ_p the place q is marked and since $M_{\mathsf{init}}(q) = 0$, that $\Delta(\sigma_p, q) = 1$. As we assume that q is not marked after firing $t_c\sigma_p^k$, $M'_{t_c\sigma_p^k}(q) = 0$, it follows with $M'_{\mathsf{init}}(q) = 0$, that $\Delta(\sigma_p^k, q) = 0$. Since $\Delta(\sigma_p, q) = \Delta(proj_{T_e}(\sigma_p), q) + \Delta(\sigma_p^k, q)$, it holds that $\Delta(proj_{T_e}(\sigma_p), q) = 1$. So $proj_{T_e}(\sigma_p)$ generates a token on q. But then q is eventually permanently enabled as $\Delta(\sigma^k, q) = 0$ and since $proj_{T_e}(\sigma)$ is a maximal firing sequence of ${\Sigma_e}^{q=0}$ according to Prop. 5.3.48, which satisfies $\mathsf{AG}((q,1) \Rightarrow \mathsf{FG}(q,1))$. □

For the following we assume: (1) - (3) and
(4) ${\Sigma_e}^{q=0} \not\models \mathsf{AG}(q,1)$.

Proposition 5.3.51 *Let σ' be a maximal firing sequence of Σ'.*

There is a firing sequence σ on Σ that is fair w.r.t. T_k and T_e and $proj_{T_k}(\sigma') = proj_{T_k}(\sigma)$.

Proof First note, that if $proj_{T_k}(\sigma') \neq \sigma'$, σ' has one occurrence of either t_p or t_c. Let σ_1^k and σ_2^k be transition sequences such that $\sigma' = \sigma_1^k t_p \sigma_2^k$ or $\sigma' = \sigma_1^k t_c \sigma_2^k$ and $proj_{T_k}(\sigma') = \sigma_1^k \sigma_2^k$.

In case $proj_{T_k}(\sigma') = \sigma'$, also $t_c\sigma'$ is a maximal firing sequence of Σ', so that this case can be considered as a special case of $\sigma_1^k t_c \sigma_2^k$.

Let us assume that σ' fires t_p. Since t_p decreases the token count of p_p only and thus only disables t_c, $t_p\sigma_1^k\sigma_2^k$ is a firing sequence of Σ' with

$proj_{T_k}(\sigma') = \sigma_1^k \sigma_2^k$. Analogously to the proof of Prop. 5.3.31 on page 111 or Prop. 5.3.14 on page 96, respectively, it is shown that there is a corresponding firing sequence σ of Σ that is fair w.r.t. T_k and T_e. Since we assume that ${\Sigma_\mathrm{e}} \not\models \mathsf{AG}(q, 0)$, there is a finite firing sequence σ_1^e of ${\Sigma_\mathrm{e}}^{q=0}$ that marks q. Since ${\Sigma_\mathrm{e}}^{q=0} \models \mathsf{AG}((q,1) \Rightarrow \mathsf{FG}(q,1))$, any maximal firing sequence σ^e of ${\Sigma_\mathrm{e}}^{q=0}$ with prefix σ_1^e satisfies $\mathcal{M}(M^e_{\mathsf{init}}, \sigma^e) \models \mathsf{FG}(q,1)$. Since $M_{\mathsf{init}}|_{P_e} = M^e_{\mathsf{init}}$, it follows that σ of `line 13` is a firing sequence of Σ.

```
1  /* The algorithm's input is the original net Σ, the
2   kernel Σ_k, the environment Σ_e, the firing sequence
3   σ_1 of Σ and firing sequence σ' = t_p σ_1^k σ_2^k of the
4   reduced net Σ'. Its output is a firing sequence of Σ
5   that is fair w.r.t. T_e and T_k. */
6  Input: Σ, Σ_k, Σ_e, σ'
7  Output: σ
8  Let σ^e be a maximal firing sequence of Σ_e^{q=0} with
9  M(M^e_init, σ^e) ⊨ FG(q,1).
10 Let σ_1^e, σ_2^e be a transition sequences where σ^e = σ_1^e σ_2^e
11 and M^e_{σ_1^e}(q) = 1.
12 σ' := σ_1^k σ_2^k /* truncate t_p */
13 σ := σ_1^e
14 if(σ' contains a t' ∈ q• with W(q,t') > W(t',q)){
15    Let σ'_p be the minimal prefix of σ' containing all
16    transitions t' with W(q,t') > W(t',q).
17    if(σ'_p is infinite){
18       while(true){
19          σ := σ σ'_p(i)
20          if(∃t_e ∈ (T_e \ q•) : M_σ[t_e⟩) σ := σ t_e }
21    } else { /* σ'_p is finite */
22       σ := σ σ'_p
23       σ' := σ'^{(|σ'_p|)} /* truncate by prefix σ'_p */
24 } }
25 /* From now on holds that W(q,σ'(i)) ≤ W(σ'(i),q),
```

```
26      ∀i, 1 ≤ i < |σ'| + 1 . */
27  if (σ' contains a t' ∈ q•){
28      Let σ'_p be σ''s minimal prefix that includes a t' ∈ q•.
29      σ := σσ'_p
30      σ' := σ'^(|σ'_p|) /* truncate by prefix σ'_p */
31      for (i := 1; i < |σ^e_2| + 1 or i < |σ'| + 1; i := i + 1){
32          if (i < |σ^e_2| + 1) σ := σσ^e_2(i)
33          if (i < |σ'| + 1) σ := σσ'(i)
34      }
35  } else { /* q ∉ •σ'(i), ∀i, 1 ≤ i ≤ |σ'| */
36      for (i := 1; i < |σ'| + 1; i := i + 1){
37          σ := σσ'(i)
38          if (∃t_e ∈ T_e : M_σ[t_e⟩) σ := σt_e
39      }
40      while (∃t_e ∈ T_e : M_σ[t_e⟩) σ := σt_e
41  }
42  return σ
```

Listing 5.2: Generating a firing sequence fair w.r.t. T_e and T_k.

As the proof is analogous to the proof of Prop. 5.3.14, we refrain from reproducing it here. The algorithm in Listing 5.2 deviates from the algorithm in Listing 5.1 in its initialisation and the if-block from `line 14-24`. Here we fix in the initialisation one firing sequence σ^e that satisfies $\mathsf{AFG}(q, 1)$. But not all firing sequences of ${\Sigma_e}^{q=0}$ satisfy $\mathsf{AFG}(q, 1)$. Therefore we fire transitions in $T_e \setminus q^\bullet$ only in case σ'_p is infinite (`line 18-21`), since then we need not continue firing σ^e. In case σ'_p is finite (`line 21-24`), we can fire it without jeopardising fairness w.r.t. T_e.

Let us now consider the case that σ' fires t_c, $\sigma' = \sigma'^k_1 t_c \sigma^k_2$. The transition sequences σ^k_1 and σ^k_2 fire transitions in $T_k \setminus ({}^\bullet q \cup q^\bullet)$ only, because q is 1-safe in Σ' according to Prop. 5.3.26. There is also a maximal firing sequence σ^e of ${\Sigma_e}^{q=0}$ that does not fire transitions in ${}^\bullet q \cup q^\bullet$, as there is a maximal firing sequence of ${\Sigma_e}^{q=0}$ that never marks q by assumptions (2+3), i.e. ${\Sigma_e}^{q=0} \not\models \mathsf{AFG}(q, 1)$ and ${\Sigma_e}^{q=0} \models \mathsf{AG}((q, 1) \Rightarrow \mathsf{FG}(q, 1))$. We construct σ by firing in

turn a transition of $\sigma_1^k\sigma_2^k$ and σ^e until both firing sequences have been fired. Since $\sigma_1^k\sigma_2^k$ and σ^e do not change the token count of q, the maximality of σ' and σ^e implies that σ is fair w.r.t. T_k and T_e. □

Theorem 5.3.52 *Let Σ_e be an Unreliable Producer environment net and Σ be reducible by Σ_e. Let ψ be an LTL$_{\text{-}\mathsf{X}}$ formula referring to $P \setminus P_e$ only.*

$\Sigma \underline{^{up}} \Sigma_\mathrm{e} \models \psi \;\Rightarrow\; \Sigma \models \psi$ *fairly w.r.t.* T_k.

$\Sigma \models \psi$ *fairly w.r.t.* T_k *and* $T_e \;\Rightarrow\; \Sigma \underline{^{up}} \Sigma_\mathrm{e} \models \psi$.

Proof We first show $\Sigma \underline{^{up}} \Sigma_\mathrm{e} \models \psi \Rightarrow \Sigma \models \psi$ fairly w.r.t. T_k . Let us assume that $\Sigma \not\models \psi$ fairly w.r.t. T_k. Hence there is a firing sequence σ of Σ that is fair w.r.t. T_k and $\mathcal{M}(M_{\mathsf{init}}, \sigma) \not\models \psi$. By Prop. 5.3.50, there is a maximal firing sequence σ' of Σ' with $proj_{T_k}(\sigma) = proj_{T_k}(\sigma')$. Since $\mathcal{M}(M_{\mathsf{init}}, \sigma) \not\models \psi$, it follows by Prop. 5.3.6 that $\mathcal{M}(M'_{\mathsf{init}}, \sigma') \not\models \psi$ and hence $\Sigma' \not\models \psi$.

Analogously follows $\Sigma \models \psi$ fairly w.r.t. T_k and $T_\mathrm{e} \Rightarrow \Sigma \underline{^{up}} \Sigma_\mathrm{e} \models \psi$ by Prop. 5.3.51. □

Theorem 5.3.52 implies that $\Sigma \underline{^{up}} \Sigma_\mathrm{e} \models \psi \;\Rightarrow\; \Sigma \models \psi$ fairly w.r.t. T_k and T_e holds and that $\Sigma \models \psi$ fairly w.r.t. $T_k \Rightarrow \Sigma \underline{^{up}} \Sigma_\mathrm{e} \models \psi$, since $\Sigma \models \psi$ fairly w.r.t. T_k implies $\Sigma \models \psi$ fairly w.r.t. T_k and T_e. With other words, according to Theorem 5.3.52 $\Sigma \underline{^{up}} \Sigma_\mathrm{e} \models \psi \;\Leftrightarrow\; \Sigma \models \psi$ fairly w.r.t. T_k and $T_\mathrm{e} \Leftrightarrow \Sigma \models \psi$ fairly w.r.t. T_k.

The Unreliable Producer reduction does not preserve CTL$_{\text{-}\mathsf{X}}$. Consider the example of an Unreliable Producer reduction as illustrated in Fig. 5.12 at the start of this section. As for the Consumer reduction on page 108 the CTL formula $\varphi = \mathsf{AF}((p_3, 1) \vee \mathsf{AG}(p_3, 0))$ distinguishes the reduced and the original net, $\Sigma' \models \varphi$ and $\Sigma \not\models \varphi$: Firing t_1 and then infinitely often t_2 never marks p_3 but from every generated state we can fire t_3t_5 to mark p_3. The reduced net satisfies φ, because the only maximal firing sequence that does not eventually mark p_3 fires t_c and after firing t_c it is not possible to mark p_3.

We cannot verify or falsify CTL or LTL using X via $\Sigma \underline{^{up}} \Sigma_\mathrm{e}$. Consider the LTL properties $\psi_1 = \mathsf{A}(\mathsf{XX}(p_3, 1) \vee \mathsf{G}(p_3, 0))$ and $\psi_2 = \mathsf{A}(\mathsf{XX}(p_3, 0))$ and the two nets in Fig. 5.12. Whereas ψ_1 holds on $\Sigma \underline{^{up}} \Sigma_\mathrm{e}$, it does not hold on Σ.

ψ_2 holds on Σ, since Σ has to fire at least three transitions to mark p_3, but ψ_2 does not hold on $\Sigma \underline{\overline{up}} \Sigma_e$.

To see that we cannot verify or falsify CTL properties consider the net in Fig. 5.13 and $\varphi = \mathsf{EX}(\mathsf{EX}(p_3, 0) \wedge \mathsf{EX}(p_3, 1))$, which means that after one step we can take a second step and p_3 is unmarked or we can mark p_3 taking a different second step. So φ does not hold on Σ as we cannot mark p_3 after two steps but does hold on $\Sigma \underline{\overline{up}} \Sigma_e$.

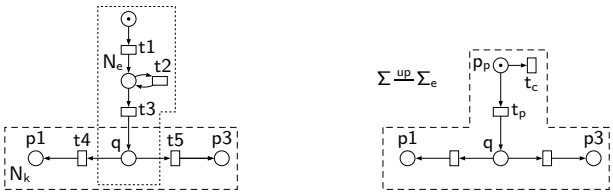


Figure 5.13: Unreliable Producer Reduction does not preserve X for CTL

5.3.7 Producer-Consumer Reduction

An environment net Σ_e is a Producer-Consumer if it has at least one firing sequence σ^e that generates a token on q and also consumes the token it generated, i.e. ${\Sigma_e}^{q=0} \not\models \mathsf{AG}(q, 0)$ and ${\Sigma_e}^{q=0} \not\models \mathsf{AG}((q, 1) \Rightarrow \mathsf{FG}(q, 1))$. In the following we denote $\Sigma \underline{\overline{pc}} \Sigma_e = (P_k, T_k \uplus \{t_r\}, W_k \uplus \{(q, t_r) \mapsto 1\}, M^{q=1}_{\mathsf{init},k})$ also as Σ'.

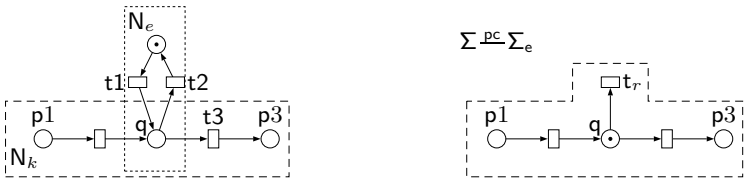


Figure 5.14: Example of a Producer-Consumer reduction

For the following we assume:
(1) q is 1-safe in Σ and (2) ${\Sigma_e}^{q=0} \not\models \mathsf{AG}(q, 0)$.

Proposition 5.3.53 ${\Sigma_e}^{q=0} \models \mathsf{EF}(q, 1)$

Proof Σ_e has a firing sequence σ_g that marks q due to assumption (2). Since q is 1-safe in Σ_e by Prop. 5.3.2, it gets at most 1 token. □

Proposition 5.3.54 *Let σ be a firing sequence of Σ that is fair w.r.t. T_k.*
There is a maximal firing sequence σ' of Σ' with $proj_{T_k}(\sigma') = proj_{T_k}(\sigma)$.

Proof We will show that either $\sigma^k := proj_{T_k}(\sigma)$ or $\sigma^k t_r$ is a maximal firing sequence of Σ'. σ^k is a firing sequence of $(N_k, M_{\mathsf{init}}^{q=1}|_{P_k})$ by Prop. 5.3.28 and by Prop. 5.3.53. Hence σ^k is also a firing sequence of Σ'. Suppose σ^k is not a maximal firing sequence of Σ'. Thus by Prop. 5.3.5 and Prop. 5.3.26 σ^k is finite, $M_{\sigma^k}(q) = 1$, and after firing σ^k a transition $t' \in T'$ is enabled. If $t' \in T_k$, then σ^k does not eventually permanently mark q, but then $\sigma' t_r$ is a maximal firing sequence of Σ'. □

For the following we assume:
(1), (2) and (3) ${\Sigma_e}^{q=0} \not\models \mathsf{AG}((q,1) \Rightarrow \mathsf{FG}(q,1))$.

We now establish the two central propositions.

- $proj_{T_k}(\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}})) \subseteq proj_{T_k}(\mathsf{Fs}_{N,\{T_k\}}(M_{\mathsf{init}}))$ and
- $proj_{T_k}(\mathsf{Fs}_{N,\{T_k\}}(M_{\mathsf{init}})) \subseteq proj_{T_k}(\mathsf{Fs}_{N',\mathsf{max}}(M'_{\mathsf{init}}))$ hold.

Proposition 5.3.55 *Let σ' be a maximal firing sequence of Σ'.*
There is a firing sequence σ on Σ with $proj_{T_k}(\sigma') = proj_{T_k}(\sigma)$ and σ is fair w.r.t. T_k.

Proof Let M^e_{init} be $M_{\mathsf{init}}^{q=0}|_{P_e}$ and let σ^e be a firing sequence of ${\Sigma_e}^{q=0}$ that eventually marks q but does not eventually permanently mark q. σ^e exists by Prop. 5.3.53 and by assumption (3) ${\Sigma_e}^{q=0} \not\models \mathsf{AG}((q,1) \Rightarrow \mathsf{FG}(q,1))$. Let σ^e be divided into prefix σ^e_g and suffix σ^e_s such that $\sigma^e = \sigma^e_g \sigma^e_s$ and σ^e_g marks q, $M^e_{\sigma^e_g}(q) = 1$. We fire a transition sequence σ consisting of up to three parts, $\sigma = \sigma^e_g \sigma^k \sigma^e_t$. We only fire a tail σ^e_t in T_e^∞, if σ^k is finite. If $\sigma^e_g \sigma^k$ marks q, we fire σ^e_s as tail. Otherwise we fire as σ^e_t transitions in T_e as long as there are any enabled. Let us call this tail σ^e_{s2}. Because firing σ^e_g on Σ enables $\sigma^k := proj_{T_k}(\sigma')$ and because we fire σ^e_s only if $M_{\sigma^e_g \sigma^k}(q) = 1$, σ is a firing sequence of Σ.

Suppose σ is not fair w.r.t. T_k. By Prop. 5.3.5, σ' is finite and σ eventually permanently marks q and $M'_{\sigma'}(q) = 0$. Since q is initially marked at Σ' but q is not marked after firing σ', σ' consumes a token from q, $\Delta(\sigma', q) = -1$. Its projection to T^k, σ^k, might omit t_r, hence $\Delta(\sigma^k, q) \leq 0$. In case we fired $\sigma = \sigma_g^e \sigma^k \sigma_s^e$, σ does not eventually permanently mark q, because $\sigma^e = \sigma_g^e \sigma_s^e$ does not eventually permanently mark q by assumption and since σ^k does not generate a token. Let us consider the case we fired $\sigma = \sigma_g^e \sigma^k \sigma_{s2}^e$. Since we fired σ_{s2}^e from a marking where q has no token, it follows that also $\sigma_g^e \sigma_{s2}^e$ is a firing sequence of Σ. Since q is 1-safe in Σ and σ_g^e generates a token on q, σ_{s2}^e does not generate a token on q. Thus σ does not eventually permanently mark q. □

Figure 5.15 shows that a stronger version of Prop. 5.3.55 for fairness w.r.t. T_k and T_e does not hold.

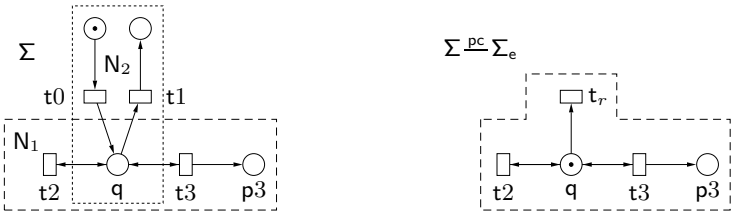


Figure 5.15: Producer-Consumer: $proj_{T_k}(\mathsf{Fs}_{N',\mathsf{max}}(M_{\mathsf{init}})) \not\subseteq proj_{T_k}(\mathsf{Fs}_{N,\{T_k,T_e\}})$. $\sigma' = t_2 t_3 t_3 ...$ is maximal on Σ' but there is no σ of Σ with $proj_{T_k}(\sigma) = proj_{T_k}(\sigma')$ that is fair w.r.t T_k and T_e.

Theorem 5.3.56 *Let Σ_e be a Producer-Consumer environment net and Σ be reducible by Σ_e. Let ψ be an LTL$_{-X}$ formula referring to $P \setminus P_e$ only.*

$\Sigma \underline{pc} \Sigma_e \models \psi \Leftrightarrow \Sigma \models \psi$ *fairly w.r.t. T_k.*

Proof This follows analogously to the Consumer's Theorem 5.3.24 at page 108 by Prop. 5.3.55, Prop. 5.3.54. □

Like the Consumer and the Unreliable Producer reduction, the Producer-Consumer reduction does not preserve CTL. Consider the Producer-Consumer example at the start of this section in Fig. 5.14. Yet again the CTL formula

$\varphi = \mathsf{AF}((p_3, 1) \vee \mathsf{AG}(p_3, 0))$ (cf. page 108) distinguishes the reduced and the original net: Firing infinitely often $t_1 t_2$ never marks p_3 but from every generated state we can fire t_3 or $t_1 t_3$ to mark p_3. So the original net does not satisfy φ, whereas the reduced net Σ' satisfies φ, because the only maximal firing sequence of Σ' that does not eventually mark p_3 fires t_r and after firing t_r it is not possible to mark p_3.

$\Sigma \underline{^{pc}} \Sigma_\mathrm{e}$ does not allows for verification or falsification of CTL or LTL using X. Consider the properties $\psi = \mathsf{A}(\mathsf{X}(p_3, 1) \vee \mathsf{G}(p_3, 0))$, expressible in LTL, and $\varphi = \mathsf{A}(\mathsf{X}(p_3, 0))$, which is expressible in CTL and LTL. $\Sigma \underline{^{pc}} \Sigma_\mathrm{e}$ satisfies ψ but Σ does not. φ does not hold on $\Sigma \underline{^{pc}} \Sigma_\mathrm{e}$ but holds on Σ, has Σ first fires t_1 which does not mark q.

5.3.8 Summary

In this section we have studied in detail what temporal properties each reduction preserves and under which fairness constraints. As summary we list here all results proved. Again Σ_e refers to the reducible environment of Σ and Σ' to the net reduced by the appropriate reduction rule. We first list the results for stutter-invariant properties.

If Σ_e is a Borrower, Producer or Dead End and φ a $\mathrm{CTL}^*_{\text{-}\mathsf{X}}$ formula referring to $P \setminus P_e$ only, then $\Sigma' \models \varphi \Leftrightarrow \Sigma \models \varphi$ *fairly w.r.t.* T_k and $\Sigma' \models \varphi \Leftrightarrow \Sigma \models \varphi$ *fairly w.r.t.* T_k *and* T_e holds (cf. Theorem 5.3.16, 5.3.33, 5.3.43).

If Σ_e is an Unreliable Producer and ψ and $\mathrm{LTL}_{\text{-}\mathsf{X}}$ formula referring to $P \setminus P_e$ only, then it holds that $\Sigma \underline{^{up}} \Sigma_\mathrm{e} \models \psi \Leftrightarrow \Sigma \models \psi$ *fairly w.r.t.* $T_k \Leftrightarrow \Sigma \models \psi$ *fairly w.r.t.* T_k *and* T_e (cf. Theorem 5.3.52).

If Σ_e is a Consumer or a Producer-Consumer and ψ is an $\mathrm{LTL}_{\text{-}\mathsf{X}}$ formula referring to $P \setminus P_e$ only, then $\Sigma' \models \psi \Leftrightarrow \Sigma \models \psi$ *fairly w.r.t.* T_k holds (cf. Theorem 5.3.24, 5.3.56).

Only the Borrower, Consumer and Dead-End reduced can be used for falsification of properties using X: Given an $\forall\mathrm{CTL}^*$ formula ψ, $\Sigma \models \psi$ fairly w.r.t. $T_k \Rightarrow \Sigma' \models \psi$ (cf. Theorem 5.3.17, 5.3.23,5.3.44).

Note, that in case there are several environment nets $\Sigma_{e_1}, ..., \Sigma_{e_n}$, the

results of this section justify that all nets are checked simultaneously, since which reduction rule is applicable depends on Σ_{e_i} only.

5.4 Necessity and Sufficiency

In this section we will first examine whether the set of reduction rules is sufficient to reduce any environment and whether all six reductions are necessary to replace any environment while preserving LTL$_{-\mathsf{X}}$. Then we discuss sufficiency for CTL$_{-\mathsf{X}}$ preserving reductions. Of course, our reductions are not sufficient to replace every environment equivalently w.r.t. CTL$_{-\mathsf{X}}$, as the results of the previous section show. We will show that any set of reductions that is sufficient to replace every environment net by an CTL$_{-\mathsf{X}}$ equivalent summary has far more reduction rules.

Necessity and Sufficiency for LTL$_{-\mathsf{X}}$Preservation The reduction rules' preconditions can be arranged in a full binary decision tree, where every node except a leaf has two children. Such a binary tree is given in Fig. 5.4 on page 83. Since this tree defines a route for every outcome of each inner node's decision, our set of reduction rules is sufficient to classify all environment nets.

The results for the Dead End reduction (c.f. Sect. 5.3.5) show that a Dead End environment can also be replaced by a Borrower or a Consumer summary. Hence the Dead End reduction is not necessary and we can build a decision tree like in Fig. 5.16. Although the Dead End reduction is not necessary to reduce environment nets, it is convenient to single out Dead Ends since they imply dead transitions and hence indicate a design error of the net.

We will now show, that all reduction rules of Fig. 5.16 are necessary by proving that we cannot find another set of reductions preserving LTL$_{-\mathsf{X}}$ with less rules.

We give example nets Σ_A and Σ_B for every two distinct environment net classes such that they are distinguishable by an LTL$_{-\mathsf{X}}$ formula only due to their environment nets. Any set of LTL$_{-\mathsf{X}}$ preserving reduction rules has to distinguish these.

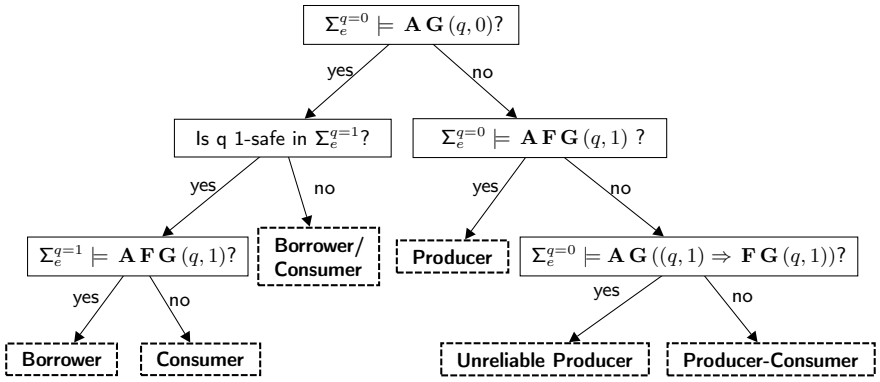


Figure 5.16: Decision tree without Dead End environment.

In Fig. 5.17 we give two nets with the same kernel where the right one has a Borrower environment whereas the left one has a Consumer environment instead. The LTL$_{\text{-X}}$ property $\psi = \mathsf{AF}(p_3, 1)$ holds for the Borrower net, because the Borrower environment can fire its transitions at most once and eventually places the token on q permanently. So the kernel transition has to mark p_3. In contrast, the Consumer net may fire t_r and in this case p_3 is not eventually marked.

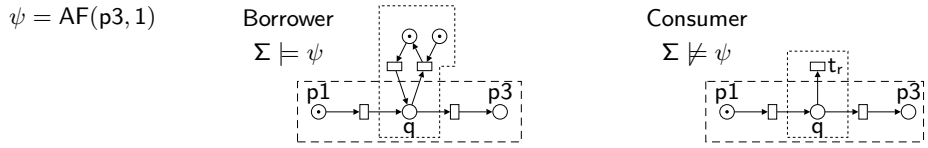


Figure 5.17: Borrower versus Consumer

Figure 5.18 demonstrates how a non-producing environment can be distinguished from a producing environment. In the Borrower net the place p_3 is never marked, whereas p_3 can eventually be marked if the environment is producing. It is shown analogously that a net with a Borrower or Consumer environment is distinguishable from a net with a Producer, Producer-Consumer or Unreliable Producer environment.

We are left to show that the producing environments are also distinguishable from each other by LTL$_{\text{-X}}$ formulas. A net with Producer environment can be distinguished from a net with Producer-Consumer environment, because the Producer is guaranteed to place the token on q. Hence p_3 is even-

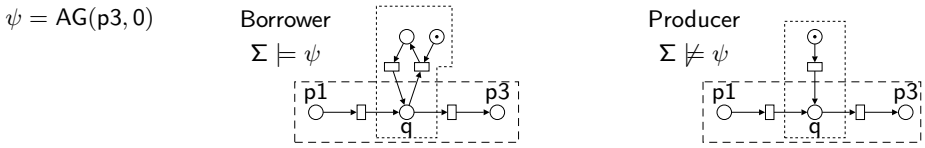


Figure 5.18: Borrower versus Producer

tually marked in the Producer net of Fig. 5.19. A Producer-Consumer environment can decide remove the token from q and an Unreliable Producer environment can decide not mark q. Hence there is an execution that does not eventually mark p_3.

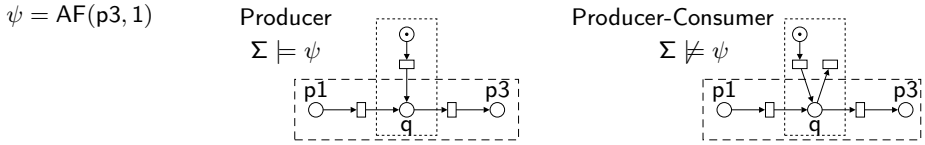


Figure 5.19: Producer versus Producer-Consumer

Finally, we show that Producer-Consumer environments must be distinguished from Unreliable Producer environments in Fig. 5.20. The net with an Unreliable Producer environment satisfies ψ, i.e. if p_3 eventually gets marked, then it always will eventually be marked again, as the token will cycle within the kernel. In the net with Producer-Consumer environment the token may also cycle within the kernel but every time the token is placed on q the environment may decide to remove the token.

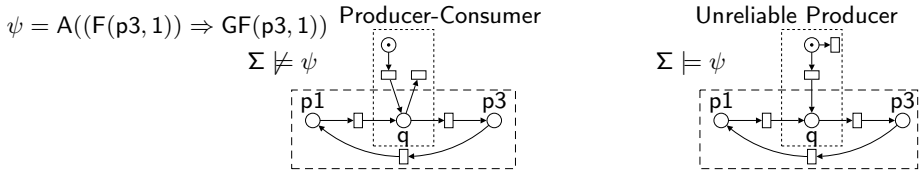


Figure 5.20: Producer-Consumer versus Unreliable Producer

So we have now shown that the set of rules is sufficient to replace any environment net and that all reductions are necessary to preserve $\text{LTL}_{-\text{X}}$.

Sufficiency for $\text{CTL}_{-\text{X}}$ Preservation Only the Borrower, Producer and Dead End reductions preserve $\text{CTL}^*_{-\text{X}}$ and hence $\text{CTL}_{-\text{X}}$, as we have seen in

Sect. 5.3. We will show that any sufficient set of CTL$_{-\mathsf{X}}$ preserving rules is much larger than our set of LTL$_{-\mathsf{X}}$ preserving rules.

Using CTL we can also distinguish the environment nets discussed above and hence a set of CTL preserving reductions has to distinguish at least all these five environments. Above we used LTL properties that are also CTL properties for all but the last case. The CTL formula $\varphi = \mathsf{EF}((p_3, 1) \wedge \mathsf{EF}(\mathsf{EG}(p_3, 0)))$ distinguishes the Producer-Consumer from the Unreliable Producer in Fig. 5.20. φ holds on the Producer-Consumer as p_3 can be marked and the token can be removed every time it is placed on q. φ does not hold on the Unreliable Producer net, because after q is marked the token circles within the kernel forever.

Since not all reductions preserve CTL we have to extend the set of reductions. Fig. 5.21 shows four nets with the same kernel but different Producer-Consumer environments that can be distinguished from each other by CTL$_{-\mathsf{X}}$ formulas. In the following we present and explain the distinguishing formulas.

The formula $\varphi_A = \mathsf{AGEF}(p_3, 1)$ holds for the net in (A) but not for the nets in (B)-(D), because in (B)-(D) the token can be removed from the contact place q and then p_3 is never marked again. The formula $\varphi_B = \mathsf{EG}((p_3, 0) \wedge \mathsf{EF}(p_3, 1))$ is satisfied if there is a path where every visited state does not mark p_3 and from every visited state a path leads to a state where p_3 is eventually marked. φ_B does not hold on (B) as the only firing sequence that never marks p_3 fires t_1t_2 and then it is not possible to mark p_3. φ_B holds on (C) and (D) since the token can circle within the environment net. We can distinguish (C) from (D) by $\varphi_C = \mathsf{EF}((p_3, 1) \wedge \mathsf{EFEG}((p_3, 0) \wedge \mathsf{EF}(p_3, 1)))$. We omit some of the quantifiers and motivate the slightly simpler formula $\varphi'_C = \mathsf{EF}((p_3, 1) \wedge \mathsf{FG}((p_3, 0) \wedge \mathsf{EF}(p_3, 1)))$ instead. φ'_C is satisfied if there is a path that leads to a state where p_3 is marked and also eventually only visits states where p_3 is not marked and where a path starts that leads to a state where p_3 is marked. So φ'_C and φ_C hold on (C), since t_0 can fire, then the token can finitely often circle within the kernel and finally the token circles forever within the environment net. While circling within the environment net, the token can always be placed onto p_3 by firing t_4 or t_0t_4, respectively. The net in (D) does not satisfy φ_C because p_3 can only be marked after t_1

has been fired and then eventually t_2 has to be fired to guarantee that p_3 is never marked again, but then no possibility remains to mark p_3.

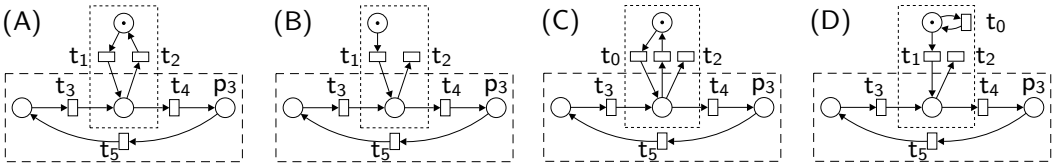


Figure 5.21: $\text{CTL}_{-\mathsf{X}}$ distinguishable nets

This demonstrates that at least four reduction rules have to be introduced just to replace the Producer-Consumer. Additional reductions have to be introduced to also replace the Consumer and Unreliable Producer reductions. So a sufficient set of $\text{CTL}_{-\mathsf{X}}$ preserving rules has to have a lot more reduction rules than our set of reductions. To have more reduction rules usually means that the appropriate reduction is less efficiently determined in the worst case.

5.5 Decomposing Monolithic Petri Nets

So far our results justify the replacement of a reducible environment net Σ_e by its summary net $S(\Sigma_\text{e})$. In this section we show that there is an efficient way to determine environment nets for a given monolithic Petri net. For a connected net Σ with a given set of 1-safe places $P_{\mathsf{safe}} \subseteq P$, we present an linear-time algorithm that decomposes Σ into (i) a kernel net Σ_k that contains all places mentioned by the temporal logic property φ and (ii) possibly several reducible environment nets $\Sigma_{\text{e}_1}, ..., \Sigma_{e_n}$. The connectedness assumption simplifies the following but does not impose a strong restriction, as we can consider every connected subnet of Σ on its own.

If we know a set of 1-safe places in Σ, the task of finding environment nets of Σ is basically the task of finding contact places $q \in (P_{\mathsf{safe}} \setminus \mathit{scope}(\varphi))$, i.e. after removing q from Σ, Σ is not connected anymore, since kernel and environment are connected by a single contact place only. We will show that with a simple modification of Σ, finding contact places is the graph theoretic problem of finding articulation points $q \in (P_{\mathsf{safe}} \setminus \mathit{scope}(\varphi))$. Since the set of articulation points can be determined in linear time by a depth-first

search (DFS) algorithm, we can decompose a net Σ with 1-safe places P_{safe} in linear time. In the next section we show how articulation points relate to contact places. In Sect. 5.5.2 we discuss the 1-safeness constraint of contact places. A decomposition algorithm is described in Sect. 5.5.3 and different decompositions strategies are outlined.

5.5.1 Articulation Points and Contact Places

Before we discuss how articulation points can be used to decompose a given monolithic Petri net, we formally introduce the graph theoretic terms connected component and biconnected component mainly following [32].

Biconnected Components and Articulation points A *graph* is a pair $G = (V, E)$ where V is a set of *vertices* (or *nodes*) and $E \subseteq (V \times V)$ is a set of *edges* (or *lines*). An edge $e = \{v, w\}$ is indicated by a line between vertices v and w. We denote a vertex as a small circle. The graph $G_\emptyset = (\emptyset, \emptyset)$ is called the *empty graph*. A graph $\tilde{G} = (\tilde{V}, \tilde{E})$ is a *subgraph* of graph $G = (V, E)$, if $\tilde{V} \subseteq V$ and $\tilde{E} \subseteq E$.

A *path* $P = (V_P, E_P)$ is a non-empty graph of the form $V_P = \{v_0, v_1, ..., v_k\}$ and $E = \{v_0v_1, v_1v_2, ..., v_{k-1}v_k\}$ where all the v_i are distinct. The vertices v_0 and v_k are linked by P and $v_1, ..., v_{k-1}$ are the *inner vertices* of P. Two or more paths are *independent* if none of them contains an inner vertex of another.

A non-empty graph is said to be *connected* if there is a path between any two nodes. A subgraph $\tilde{G}$ is a *connected component* of G, if $\tilde{G}$ is a maximal connected subgraph of G, that is $\tilde{G}$ is not contained in any other connected subgraph of G.

If $A, B, C \subseteq V$ are such that every path between a node $a \in A$ and a node $b \in B$ contains a vertex in C, then C *separates* the sets A and B.

Now we introduce the two main notions, articulation points and biconnected components.

A vertex $a \in V$ is an *articulation point* (or *cutvertex*), if there are vertices v and w of the same component such that v, w and a are distinct, and $\{a\}$

separates $\{v\}$ and $\{w\}$. Alternatively, a is said to be an articulation point if the deletion of a increases the number of connected components in the graph.

A graph G is said to be *biconnected* if and only if it has no articulation points. A graph $\tilde{G}$ is a *biconnected component* of G iff $\tilde{G}$ is a maximal biconnected subgraph of G.

Any two nodes of a biconnected graph (or component) are joined by two independent paths—except for the two examples given in Fig. 5.22.

Figure 5.22: Smallest biconnected graphs

As illustrated in Fig. 5.23, articulation points divide the graph into biconnected components, so that neighbouring biconnected components share an articulation point. The biconnected components of G define a partition of G's edge set.

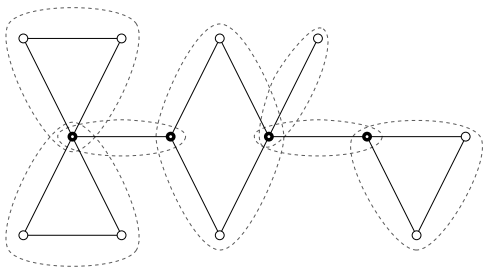


Figure 5.23: A graph and its biconnected components. A biconnected component is the subgraph within a dashed bubble. Articulation points are boldly bordered.

Articulation Points as Contact Places The key idea for the decomposition algorithm is to use a DFS algorithm for determining articulation points. A good presentation of a DFS algorithm to determine articulation points can be found in [64]. Formally articulation points are vertices of a graph. Contact places are special places of a Petri net. We define the graph G_Σ of a Petri net Σ as the graph (V, E) with vertices $V = P \cup T$ and edges $E = \{\{x, y\} \mid (x, y) \in ((P \times T) \cup (T \times P)) : W(x, y) \neq 0\}$. So G_Σ basically

forgets that the bipartite Petri net graph has two kinds of vertices and that arcs have an direction and weight.

We say that $p \in P$ is an *articulation place* of Σ, iff p is an articulation point of G_Σ. An articulation place q is a contact place iff (i) it is 1-safe and (ii) it does not separate any two places in $scope(\varphi)$, as the kernel contains all places of $scope(\varphi)$. To guarantee (ii), we extend G_Σ by a vertex v_φ and edges connecting all nodes of $scope(\varphi)$ to v_φ. With this extension all nodes in $scope(\varphi)$ are connected via two independent paths, since we assume that they are initially connected and a second path exists via v_φ.

Now every articulation point $a \in P_{\mathsf{safe}} \setminus scope(\varphi)$ corresponds to a contact place and the biconnected component containing $scope(\varphi)$ corresponds to the kernel.

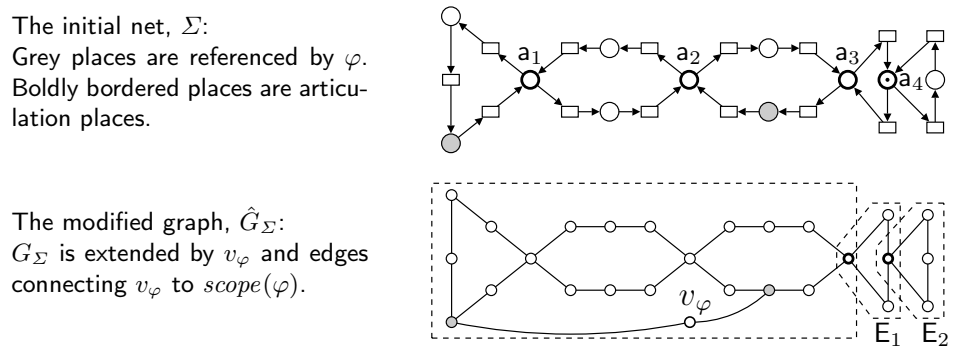


Figure 5.24: Extension of G_Σ.

Complexity Let us now consider the complexity of decomposing a monolithic Petri net Σ by a DFS algorithm for determining articulation points when a set of 1-safe places P_{safe} is given. A good presentation of such an algorithm is given in [64].

To build the graph G_Σ takes $\mathcal{O}(|P| + |T| + |W|)$, where $|W| = |\{(x, y) \in ((P \times T) \cup (T \times P)) \mid W(x, y) \neq 0\}|$. To extend this graph by v_φ and connect it to $scope(\varphi)$ takes time of $\mathcal{O}(|scope(\varphi)| + 1)$. Determining articulation points via DFS, takes again time $\mathcal{O}(|P| + 1 + |T| + |W| + |scope(\varphi)|)$. Hence the algorithm determines contact places in $\mathcal{O}(|P| + |T| + |W|)$ time.

5.5.2 1-Safeness of Contact Places

Given we know which places of Σ are 1-safe, we can very efficiently determine contact places, but to determine whether a given P/T net is 1-safe, is known to be PSPACE complete [18]. Hence determining whether a single place is 1-safe is at least PSPACE hard. So determining whether an articulation place is 1-safe is as difficult as LTL model checking itself. But often nets are known to be 1-safe or it may for instance be possible to determine a structural bound by linear programming techniques, which can be done in polynomial time [25].

In the sequel we will prove that we can also modularly determine whether the (candidate) contact place q is 1-safe: Instead of checking whether q is 1-safe in the whole net Σ, we can examine the environment Σ_e and the reduced net Σ'. This allows us to "guess" which (candidate) contact places are 1-safe, replace the environment and check afterwards, whether we did guess correctly. But if we guessed wrongly, we have to undo the replacement.

5.5.2.1 Inducement of 1-Safeness

We will now prove that if we assume additional constraints on the environment, we can guarantee that q is 1-safe in the original net. Although we have to check the constraints additionally on the environments, they do not restrict the sufficiency of our reductions. If we replace an environment at a 1-safe contact place q, these constraints are implied by 1-safeness of q in Σ (cf. Sect. 5.3).

For the sequel let Σ consist of the two subnets Σ_e and Σ_k with contact place q.

Outline We prove for each reduction that the additional constraints we make suffice to guarantee that q is 1-safe in Σ. We show that any firing sequence σ of Σ generates at most one token, by showing that $proj_{T_k}(\sigma)$ and $proj_{T_e}(\sigma)$ are constrained to generate together at most one token. Therefore it is shown that $proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_e}^{q=0}$ or ${\Sigma_e}^{q=1}$, respectively, and that $proj_{T_k}(\sigma)$ corresponds to a firing sequence of Σ'. As we assumed

in Sect. 5.3 that q is 1-safe in Σ, we cannot reuse the results accomplished there.

Borrower and Consumer Reduction

For the following we assume:
(1) q is 1-safe in Σ' and (2) Σ_{e} is a Borrower or Consumer.

Proposition 5.5.1 *Let σ be a firing sequence of Σ such that $proj_{T_k}(\sigma)$ is a firing of Σ_{k}.*

$proj_{T_e}(\sigma)$ is a firing sequence of $\Sigma_e^{q=1}$.

Proof Let M^e_{init} be $M^{q=1}_{\mathsf{init}}|_{P_e}$. If the above does not hold, then by Prop. 5.3.3 there is a prefix σ_p of σ such that for $\sigma_p^e := proj_{T_e}(\sigma_p)$ it holds that $M_{\sigma_p}(q) > M^e_{\sigma_p^e}(q)$. Hence it follows that $M_{\mathsf{init}}(q) + \Delta(proj_{T_k}(\sigma_p), q) > 1$. As we assume that $proj_{T_k}(\sigma_p)$ is a firing sequence of Σ^k and hence of Σ', this contradicts to 1-safeness of q in Σ'. □

Proposition 5.5.2 *Let σ be a firing sequence of Σ.*

$proj_{T_k}(\sigma)$ is a firing sequence of Σ_{k}.

Proof This follows with Prop. 5.5.1 as for Prop. 5.3.10 □

Theorem 5.5.3 *Let Σ_{e} be a Borrower or a Consumer environment and let Σ' be respectively the Borrower- or Consumer-reduced of Σ.*

If the contact place q is 1-safe in Σ', then q is 1-safe in Σ.

Proof Let σ be a finite firing sequence of Σ. Since $proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_{\mathrm{e}}}^{q=1}$ by Prop. 5.5.1 and q is 1-safe in ${\Sigma_{\mathrm{e}}}^{q=1}$, $\Delta(proj_{T_e}(\sigma), q) < 1$. Since $proj_{T_k}(\sigma)$ is a firing sequence of Σ' by Prop. 5.5.2 and q is 1-safe in Σ', $\Delta(proj_{T_k}(\sigma), q) \leq 1$. Hence σ generates at most one token on q. □

Dead End Reduction

As the Dead End-reduced net we study $\Sigma' := \Sigma^k$, just like we have done in Sect. 5.3.5. There we have shown that the reduced net never fires transitions in ${}^\bullet q \cup q^\bullet$ and that they can hence be removed. For the following result

it is more convenient though to consider again Σ^k as the reduced net, because we can formulate a simpler and more intuitive constraint. Later on we will show how to formulate the constraint in case we removed ${}^\bullet q \cup q^\bullet$ from Σ'.

For the following we assume:
(1) q is never marked in Σ' and (2) Σ_e is a Dead End.

Proposition 5.5.4 *Let σ be a firing sequence of Σ such that $proj_{T_k}(\sigma)$ is a firing sequence of Σ'.*

$proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_\mathrm{e}}^{q=0}$.

Proof Let M^e_{init} be $M^{q=0}_{\mathsf{init}}|_{P_e}$. If the above does not hold, then by Prop. 5.3.3 there is a prefix σ_p of σ such that for $\sigma^e_p := proj_{T_e}(\sigma_p)$ it holds that $M_{\sigma_p}(q) > M^e_{\sigma^e_p}(q)$. But then $M'_{\mathsf{init}}(q) + \Delta(proj_{T_k}(\sigma), q) > 0$, which contradicts our assumption that q is never marked in Σ'. □

Proposition 5.5.5 *Let σ be a firing sequence of Σ.*

$proj_{T_k}(\sigma)$ is a firing sequence of Σ'.

Proof Let M^e_{init} be $M^{q=0}_{\mathsf{init}}|_{P_e}$. If the above does not hold, then by Prop. 5.3.3 there is a prefix σ_p of σ such that for $\sigma^k_p := proj_{T_k}(\sigma_p)$ it holds that $M_{\sigma_p}(q) > M^k_{\sigma^k_p}(q)$. Hence $\sigma^e_p := proj_{T_e}(\sigma_p)$ generates a token on q. But σ^e_p cannot generate a token, since σ^e_p is a firing sequence of ${\Sigma_\mathrm{e}}^{q=0}$ and ${\Sigma_\mathrm{e}}^{q=0} \models \mathsf{AG}(q, 0)$. □

Theorem 5.5.6 *Let Σ_e be a Dead End environment of Σ.*

If the contact place q is never marked in $\Sigma \underline{d} \Sigma_\mathrm{e}$, then q is never marked in Σ.

Proof Let σ be a finite firing sequence of Σ. Since $\sigma^e := proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_\mathrm{e}}^{q=0}$ and ${\Sigma_\mathrm{e}}^{q=0} \models \mathsf{AG}(q, 0)$, σ^e does not generate tokens on q, $\Delta(\sigma^e, q) = 0$. Since $\sigma^k := proj_{T_k}(\sigma)$ is a firing sequence of Σ' and since is q never marked in Σ' by assumption, σ^k also does not generate tokens on q, $\Delta(proj_{T_k}(\sigma), q) \leq 0$. □

An analogous result can be established for a Dead End-reduced, where the transitions in ${}^\bullet q \cup q^\bullet$ have been deleted. Instead of disallowing a token on q, we disallow markings that would have enabled transitions in ${}^\bullet q$. Let $\mathcal{M}_t$ be the minimal enabling marking of t in Σ, i.e. $\mathcal{M}_t(p) = W(p,t)$, $\forall p \in P$. For the case that transitions ${}^\bullet q \cup q^\bullet$ have been removed, the analogous result can then be formulated as:

If for every transition $t \in {}^\bullet q \setminus q^\bullet$ in Σ' it holds that $(\forall M' \in [M'_{\mathsf{init}}\rangle : \exists p \in P' : M'(p) < \mathcal{M}_t(p))$, then q is never marked in Σ.

To check this result when the set of transitions $({}^\bullet q \cup q^\bullet)$ has been removed, it would hence be necessary to store all forbidden markings that might enable (the removed) transitions in ${}^\bullet q \setminus q^\bullet$.

Producer and Producer-Consumer Reduction

For the following we assume:
(1) $M_{\mathsf{init}}(q) = 0$, (2) q is 1-safe in ${\Sigma_{\mathrm{e}}}^{q=0}$ and
(3) q is 1-safe in Σ' and (4) Σ_{e} is a Producer or a Consumer.

Proposition 5.5.7 *Let σ be a firing sequence of Σ such that $proj_{T_k}(\sigma)$ is a firing of Σ'.*

$proj_{T_e}(\sigma)$ is a firing sequence of $\Sigma_e^{q=0}$.

Proof Let M^e_{init} be $M^{q=0}_{\mathsf{init}}|_{P_e}$. If σ is a firing sequence of Σ but $\sigma^e := proj_{T_e}(\sigma)$ is not a firing sequence of ${\Sigma_{\mathrm{e}}}^{q=0}$, then by Prop. 5.3.3, there is a prefix σ_p of σ such that for the prefix $\sigma^e_p := proj_{T_e}(\sigma_p)$ of σ^e it holds that $M_{\sigma_p}(q) > M^e_{\sigma^e_p}(q)$. But then $\Delta(proj_{T_k}(\sigma_p), q) > 0$, which contradicts assumption (3). □

Proposition 5.5.8 *Let σ be a firing sequence of Σ.*

$proj_{T_k}(\sigma)$ is a firing sequence of Σ'.

Proof If the above does not hold, then by Prop. 5.3.3, there is a prefix σ_p of σ such that for $\sigma'_p := proj_{T_k}(\sigma)$ it holds that $M_{\sigma_p}(q) > M'_{\sigma'_p}(q)$. Since $M'_{\mathsf{init}}(q) = 1$ by definition and $M_{\mathsf{init}}(q) = 0$ by assumption (1), $\Delta(proj_{T_e}(\sigma), q) > 1$, which contradicts assumption (2). □

Theorem 5.5.9 *Let Σ_e be a Producer or Producer-Consumer environment of Σ and Σ' be the Producer- or Producer-Consumer-reduced. Let q be the contact place.*

If q is 1-safe in ${\Sigma_e}^{q=0}$, $M_{\mathsf{init}}(q) = 0$ and q is 1-safe in Σ', then q is 1-safe in Σ.

Proof Let σ be a finite firing sequence of Σ. Since $\sigma^e := proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_e}^{q=0}$ by Prop. 5.5.7 and q is 1-safe in ${\Sigma_e}^{q=0}$, σ^e generates at most one token on q. $\sigma^k := proj_{T_k}(\sigma)$ is a firing sequence of Σ' by Prop. 5.5.8. Since q is 1-safe and initially marked in Σ', σ^k does not generate a token on q, $\Delta(\sigma^k, q) \leq 0$. □

Unreliable Producer Reduction

For the following we assume:
(1) $M_{\mathsf{init}}(q) = 0$, (2) q is 1-safe in ${\Sigma_e}^{q=0}$ and
(3) q is 1-safe in Σ' and (4) Σ_e is an Unreliable Producer.

Proposition 5.5.10 *Let σ be a firing sequence of Σ such that $t_p proj_{T_k}(\sigma)$ is a firing sequence of Σ'.*

$proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_e}^{q=0}$.

Proof Let M^e_{init} be $M^{q=0}_{\mathsf{init}}|_{P_e}$. If the above does not hold, then by Prop. 5.3.3, there is a prefix σ_p of σ such that for $\sigma^e_p := proj_{T_e}(\sigma_p)$ it holds that $M_{\sigma_p}(q) > M^e_{\sigma^e_p}(q)$. Since $M^e_{\mathsf{init}}(q) = 0 = M_{\mathsf{init}}(q)$, it follows that $\Delta(proj_{T_k}(\sigma_p), q) > 0$. But since $t_p proj_{T_k}(\sigma_p)$ is a firing sequence of Σ', this contradicts 1-safeness of q in Σ'. □

Proposition 5.5.11 *Let σ be a firing sequence of Σ.*

$t_p proj_{T_k}(\sigma)$ is a firing sequence of Σ'.

Proof The proof is by induction on the length l of σ. The case $l = 0$ follows trivially.

$l \to l+1$: Let σt be a firing sequence of length $l+1$. We denote $proj_{T_k}(\sigma)$ as σ^k. The case $t \in T_e$, follows directly by the induction hypothesis. If

$t \in T_k$ and t is not enabled after firing $t_p\sigma^k$ on Σ', then it follows that $M'_{t_p\sigma^k}(q) < M_\sigma(q)$. Since $M'_{\mathsf{init}}(q) = M_{\mathsf{init}}(q) = 0$, $\Delta(t_p\sigma^k, q) < \Delta(\sigma, q)$. It follows that $\Delta(proj_{T_e}(\sigma), q) > 1$. But this contradicts 1-safeness of q in ${\Sigma_\mathrm{e}}^{q=0}$, since $proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_\mathrm{e}}^{q=0}$ by Prop. 5.5.10. □

Theorem 5.5.12 *Let Σ_e be an Unreliable Producer. Let q be the contact place.*

If q is 1-safe in ${\Sigma_\mathrm{e}}^{q=0}$, $M_{\mathsf{init}}(q) = 0$ and q is 1-safe in $\Sigma \underline{^{up}} \Sigma_\mathrm{e}$, then q is 1-safe in Σ.

Proof Let σ be a finite firing sequence of Σ. Since $\sigma^e := proj_{T_e}(\sigma)$ is a firing sequence of ${\Sigma_\mathrm{e}}^{q=0}$ by Prop. 5.5.10 and q is 1-safe in ${\Sigma_\mathrm{e}}^{q=0}$, $\Delta(proj_{T_\mathrm{e}}(\sigma), q) \leq 1$. $t_p proj_{T_\mathrm{k}}(\sigma)$ is a firing sequence of Σ' by Prop. 5.5.11. Since q is 1-safe in Σ' by assumption, $\Delta(proj_{T_k}(\sigma), q) \leq 0$. □

Conclusion We have shown that if the contact place q is 1-safe in a Borrower- or Consumer-reduced net ${\Sigma_\mathrm{e}}^{q=0}$, then q is also 1-safe in the original net Σ. If q is never marked in a Dead End-reduced, then q is also never marked in the original net Σ. If q is initially unmarked in Σ, q is 1-safe in ${\Sigma_\mathrm{e}}^{q=0}$ and q is 1-safe in Σ', then q is 1-safe in the original net Σ, where Σ' is the Producer- or Unreliable Producer- or Producer-Consumer-reduced of Σ.

As observed at the outset, we have shown here that Σ_e satisfying additional constraints induces that q is 1-safe in Σ and we have shown in Sect. 5.3 that if q is 1-safe in Σ these additional constraints on Σ_e are implied.

The results of this section also justify the replacement of an environment $\Sigma_{\mathrm{e}1}$ by another environment $\Sigma_{\mathrm{e}2}$ satisfying the same constraints. To see this suppose we have two nets Σ_1 and Σ_2 both with the same kernel but Σ_1 has environment Σ_{e_1} and Σ_2 has environment Σ_{e_2}. If we can replace $\Sigma_{\mathrm{e}1}$ and $\Sigma_{\mathrm{e}2}$ both by the same summary $S(\Sigma_{\mathrm{e}_1}) = S(\Sigma_{\mathrm{e}_2})$, then the reduced nets $\Sigma'_1 = \Sigma'_2$ are equivalent w.r.t. LTL$_{\text{-x}}$ properties referring to $P_k \setminus \{q\}$ from Σ_1 and Σ_2 (assuming fairness) by the results of Sect. 5.3. Hence also Σ_1 and Σ_2 are indistinguishable by LTL$_{\text{-x}}$ properties referring to $P_k \setminus \{q\}$.

5.5.3 Applying Reductions and DFS

In the following we describe how we use a DFS-algorithm determining biconnected components [64] in combination with our reduction rules.

Input of the algorithm is a given net Σ, a set of places $scope(\varphi)$, as well as a set of 1-safe places of Σ, P_{safe}. The algorithm reduces the net following the DFS, such that contact places are in $P_{\mathsf{safe}} \setminus scope(\varphi)$ and the kernel contains all of $scope(\varphi)$.

1. Generate the modified G_Σ, $G_{\tilde{\Sigma}}$.

   Initialise $G_\Sigma = (V_\Sigma, E_\Sigma) := (P \cup T, \{\{x, y\} \mid (x, y) \in (P \times T) \cup (T \times P) : W(x, y) \neq 0\})$. Then, as illustrated in Fig. 5.24, connect every vertex in $scope(\varphi)$ with v_φ to build $G_{\tilde{\Sigma}} := (V_\Sigma \cup \{v_\varphi\}, E_\Sigma \cup \{\{v_\varphi, p\} \mid p \in scope(\varphi)\})$.

2. Perform a DFS to determine biconnected components of $G_{\tilde{\Sigma}}$.

   The DFS uses a stack to keep track of the currently traversed component. Visited edges are put onto the stack. If an articulation point $a \in P_{\mathsf{safe}} \setminus scope(\varphi)$ is found, edges are removed from the stack's top down to the first encounter of a.

   (a) The search starts at a place $p \in scope(\varphi)$, so that the first and thus the last component on the stack is the biconnected component containing all places in $scope(\varphi)$. This component corresponds to the kernel of Σ.

   (b) When an articulation point is found, the removed edges define a reducible environment E. Replace the subnet Σ_E corresponding to E by its summary $S(\Sigma_E)$.

Replacement Strategies By implementing the algorithm as described above, the smallest environment subnets are successively replaced. Figure 5.25 contrasts this with a strategy to replace maximal environment subnets. There are many possible replacement strategies in between these two extremes. To avoid the combinatorial blow-up it is theoretically best to replace

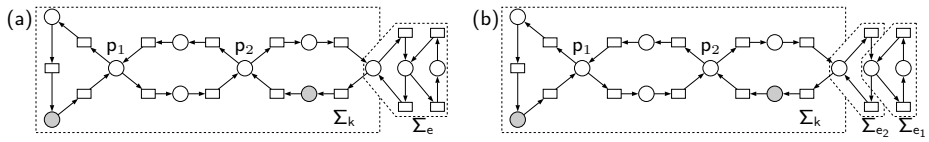


Figure 5.25: Decomposition into kernel and (a) maximal and (b) minimal environments.

the smallest possible environment. But to determine the appropriate reduction, we have to model check every environment at most three times (cf. Fig. 5.4) and in the end we might replace the environment by a summary net with as many states. If we replace smallest environments that risk increases. This inspired us to optimise our algorithm by *micro reduction rules*. The appropriate reduction rule for an environment consisting of an articulation point, a transition and up to one additional place is determined by matching the net structure instead of model checking. Micro reductions will be presented in Sect. 5.7.

5.6 Cost-Benefit Analysis

As shown in Sect. 5.5.1 the decomposition of Σ can be determined in time linear in the size of Σ given we know whether cutvertex places are 1-safe. We discussed in Sect. 5.5.2 approaches that allow to use our technique even if the set of 1-safe places is not known in advance.

In the following we make a cost-benefit analysis of a net like the one in Fig. 5.26 by replacing minimal environment nets.

Convention We refer to the net Σ of Fig. 5.26 as $\Sigma_\mathrm{k} \circ \Sigma_{\mathrm{e}_1} \circ \Sigma_{\mathrm{e}_2}$ to stress that Σ chains up $\Sigma_\mathrm{k}, \Sigma_{\mathrm{e}_1}$ and Σ_{e_2}. Analogously we denote the reduced net Σ' as $\Sigma_k \circ S(\Sigma_{\mathrm{e}_1} \circ S(\Sigma_{\mathrm{e}_2}))$. We use $|TS_{\Sigma_1}|_\Sigma$ to denote the number of states and state transitions of a subnet Σ_1 has within Σ.

If Σ mainly evolves sequentially, the size of the state space of Σ can be approximated as the sum of the state space sizes of its subsystems $|TS_\Sigma| = |TS_{\Sigma_\mathrm{k}}|_\Sigma + |TS_{\Sigma_{\mathrm{e}_1}}|_\Sigma + |TS_{\Sigma_{\mathrm{e}_2}}|_\Sigma$, but if kernel and environment evolve concurrently the state space size is better approximated as $|TS_\Sigma| = |TS_{\Sigma_\mathrm{k}}|_\Sigma \cdot |TS_{\Sigma_{\mathrm{e}_1}}|_\Sigma \cdot |TS_{\Sigma_{\mathrm{e}_2}}|_\Sigma$. The state space size of the reduced net is analog-

ously approximated as $|TS_{\Sigma'}| = |TS_{\Sigma_k}|_{\Sigma'} + |TS_{S(\Sigma_{e_1}+S(\Sigma_{e_2}))}|_{\Sigma'}$ or $|TS_{\Sigma'}| = |TS_{\Sigma_k}|_{\Sigma'} \cdot |TS_{S(\Sigma_{e_1}+S(\Sigma_{e_2}))}|_{\Sigma'}$ where $|TS_{S(\Sigma_{e_1}+S(\Sigma_{e_2}))}|_{\Sigma'}$ is at most 3 and it is guaranteed that $|TS_{\Sigma'}| \leq |TS_{\Sigma}|$. Consequently the reduced net saves the more the more concurrent Σ_k, Σ_{e_1} and Σ_{e_2} evolve and the bigger their state spaces. As LTL model checking is in $\mathcal{O}(|TS_{\Sigma}| \cdot 2^{|\psi|})$ the balance is even better when complex formulas are examined. But to determine the reduction we also have expenses.

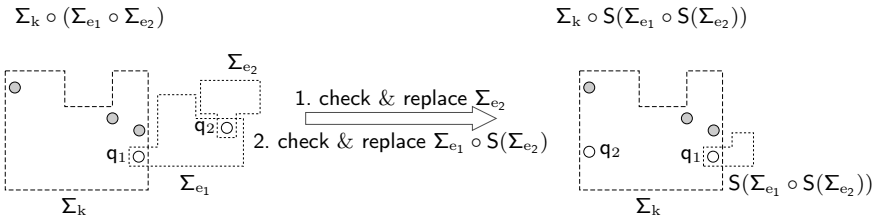


Figure 5.26: Replacement of minimal environment nets.

To replace Σ_{e_2} we have to identify the appropriate summary net, $S(\Sigma_{e_2})$. In the worst case this entails checking up to three LTL formulas (cf. Fig. 5.4), either *(i)* $\mathsf{AG}(q,0)$, $\mathsf{AFG}(q,1)$ and $\mathsf{AG}((q,1) \Rightarrow \mathsf{FG}(q,1))$ on $\Sigma_{e_2}^{q=0}$, or *(ii)* $\mathsf{AG}(q,0)$ on ${\Sigma_e}^{q=0}$ and $\mathsf{AG}((q,0) \vee (q,1))$ and $\mathsf{AFG}(q,1)$ on $\Sigma_{e_2}^{q=1}$. The states on $\Sigma_{e_2}^{q=0}$ are also reachable within Σ, so that $|TS_{{\Sigma_{e_2}}^{q=0}}| \leq |TS_{\Sigma_{e_2}}|_{\Sigma}$ follows. If q is eventually marked on Σ, all states of $\Sigma_{e_2}^{q=1}$ are also reachable on Σ_{e_2} within Σ , so that $|TS_{{\Sigma_{e_2}}^{q=1}}| \leq |TS_{\Sigma_{e_2}}|_{\Sigma}$ is guaranteed. But if q is never marked on Σ, then spurious behaviour in ${\Sigma_{e_2}}^{q=1}$ is possible, since the token on q might enable additional transitions. In this case it is not possible to give an upper bound on the expense, as $\Sigma_{e_2}^{q=1}$ might become unbounded. In Sect. 5.2 we discussed ways to deal with these cases.

So suppose we do not encounter spurious behaviour. Determining the appropriate summary net for Σ_{e_2} is in $\mathcal{O}(|TS_{\Sigma_{e_2}}| \cdot 2^8)$, since the longest formula $\mathsf{AG}((q,1) \Rightarrow \mathsf{FG}(q,1))$ has eight operators when expressed as via $\neg, \wedge, \mathsf{X}$ and U. Determining the replacement for $\Sigma_{e_1} \circ S(\Sigma_{e_2})$ is hence in $\mathcal{O}(|TS_{\Sigma_{e_1} \circ S(\Sigma_{e_2})}| \cdot 2^8)$. $TS_{\Sigma_{e_1} \circ S(\Sigma_{e_2})}$ might be much smaller than $|TS_{\Sigma_{e_1} \circ \Sigma_{e_2}}|$, especially if Σ_{e_1} and Σ_{e_2} have many states and evolve concurrently. But in the worst case Σ_{e_2} has as many states as $S(\Sigma_{e_2})$, which means that Σ_{e_2} has

a very small state space of at most 3 states.

To summarise, our method saves $\Delta_{TS} := |TS_{\Sigma_k \circ \Sigma_{e_2} \circ \Sigma_{e_1}}| - |TS_{\Sigma_k \circ S(\Sigma_{e_2} \circ S(\Sigma_{e_1}))}|$ states. For each replacement we risk to spend $\mathcal{O}(|TS_{\Sigma_e} \cdot 2^8|)$, where Σ_e is the environment without any prior replacements. As a rule of thumb, our method pays off when state spaces of the component nets are bigger than the state spaces of their summaries and if the component nets evolve concurrently. As LTL model checking is in $\mathcal{O}(|TS_\Sigma| \cdot 2^{|\varphi|})$ the method usually also pays off when formulas more complex than $\mathsf{AG}((q,1) \Rightarrow \mathsf{FG}(q,1))$ are model checked. We might get an overhead when we apply our method to a net where its components evolve sequentially and a simple formula is checked.

Of course, the above analysis is a worst case analysis. For instance we only have to check the three formulas under *(ii)* if they all evaluate to false. But usually it is not necessary to explore the full state space to find a counterexample. Also—as outlined in the following section—it is straight forward to implement optimisations based on the structure of the environments. These optimisations do not rule out the worst case behaviour but make it less likely.

5.7 Optimisations

In the sequel we will introduce two structural optimisations, namely micro reductions and pre/postset optimisations for connected environment nets. We then examine optimising the order of formulas in the decision tree classifying the summary nets. Last we discuss optimising the replacement time by parallelising the identification of the appropriate reduction.

5.7.1 Micro Reductions

Micro reductions determine the appropriate summary net for the smallest environment nets by just inspecting the net structure. We implemented two such replacement algorithms for connected environment nets. The summary net for an environment net Σ_e with just one transition and upto two places is determined by the algorithm in Listing 5.3. Since the environment has just one transition, it either behaves as a producer, a consumer or does not

change the token count on q, in which case we can replace it by a borrower net. If the transition t is connected to q via arcs with arc weight greater one, then t can never be fired in Σ since q is 1-safe (`line 7`). Σ_e is a consumer if t can consume a token from q. We have to only check that p is sufficiently marked to allow at least one firing of a consuming transition t—the contact place q might get marked later on (`line 9`).

```
1  let q be the contact place and t be the transition;
2  let p be the other place if it exists;
3  int Δq := W(t,q) − W(q,t);
4  bool enabled_p := true;
5  if (p exists) enabled_p := W(p,t) ≤ M_init(p);
6  bool enabled := W(q,t) ≤ M_init(q) ∧ enabled_p;
7  if(W(t,q) > 1 ∨ W(q,t) > 1) borrower;
8  elseif(Δq == 1 ∧ enabled) producer;
9  elseif(Δq == −1 ∧ enabled_p) consumer;
10 else borrower;
```

Listing 5.3: Micro reductions for one transition and up to two places.

The summary net for an environment net Σ_e with just two transitions, contact place q and upto one additional place p and maximal arc weight 1 is determined by the algorithm in Listing 5.4.

We replace Σ_e with a borrower if q is read only by t_1 and t_2. If $|q^\bullet| < |{}^\bullet q|$ holds, then either both transitions have q as output place and one of them may have q as input place or only one transition has q as output. We check whether one transition can produce. A transition $t_1 \in {}^\bullet q \setminus q^\bullet$ has a second input place p as input place, since q is 1-safe. If $M_{\mathsf{init}}(p) == 1$ then t_1 can generate a token on q. If t_2 may consume the token from p, then Σ_e is an unreliable producer, else a producer (`line 8-9`). If $M_{\mathsf{init}}(p) == 0$ then neither t_1 nor t_2 can change the token count on q and we can replace Σ_e by a borrower net.

If $|q^\bullet| > |{}^\bullet q|$ holds, then either both transitions have q as input place and one of them may also have q as output place or only one transition has q as input and q has no input transitions. A transition $t_1 \in q^\bullet \setminus {}^\bullet q$ can consume

a token from q, if it either has no further or its other input p is marked or can get marked by firing t_2. If t_1 cannot consume, then because of p being not sufficiently marked. We then check whether t_2 can consume. If neither t_1 nor t_2 are enabled, the token count on q cannot be changed and we replace Σ_e by a borrower net.

If $|{}^\bullet q| = |q^\bullet|$ but not ${}^\bullet q = q^\bullet$, then q has an input transition t_1 and an output transition t_2. Again p has to be input place of t_1, since q is 1-safe in Σ. `Line 20` handles the case that t_1 has output q and reads p. The place p can never get marked, since q is 1-safe in Σ. Hence if t_2 has p only as output, q can never be marked since this would enable t_2. We replace such an environment by a dead end. If p is initially marked, t_1 can produce. If t_2 can also fire, then Σ_e is a producer-consumer else a producer. Transition t_2 can fire, if does not depend on the token count on p. As p will be unmarked after t_1 fired. If p is initially unmarked, then t_1 cannot produce (but regenerate). If t_2 has p as input also t_2 is disabled and Σ_e does not change the token count of q. Hence we replace Σ_e by a borrower. If t_2 has no further input place, then Σ_e is a consumer.

```
1  let q be the contact place;
2  let p be the other place if it exists;
3  if( ●q == q●) borrower;
4  else if (|q●| < |●q|) {
5     let t1 be the transition in ●q \ q●;
6     let t2 be the other transition;
7     if(M_init(p)==1) {
8        if(t2 ∈ p● ∧ q ∉ t2●) unreliable_producer;
9        else producer;
10    } else borrower;
11 } elseif (|q●| > |●q|) {
12    let t1 be a transition in q● \ ●q;
13    let t2 be the other transition;
14    if((|●t1| == 2) ⇒ (M_init(p) > 0 ∨ (t2 ∈ ●p \ p●))) consumer;
15    else if(t2 ∈ q● \ ●q ∧ |●t2| == 1) consumer;
```

```
16      else borrower;
17  } else {
18      let t_1 be the transition in •q;
19      let t_2 be the transition in q•;
20      if(p ∈ (•t_1 ∩ t_1•) ∧ p ∈ t_2• ∧ ¬(p ∈ •t_2) ) deadend;
21      if(M_init(p) == 1) {
22          if(p ∈ •t_2) producer;
23          else producer−consumer;
24      else {
25          if(p ∈ •t_2) borrower;
26          else consumer;
27      }
28  }
```

Listing 5.4: Micro reductions for two transitions, contact place, upto one additional place and arc weights in $\{0, 1\}$.

By using micro reductions we can efficiently replace the smallest environment nets and do not have to model check this small nets (upto three times) exploring the full state space. Also we do not risk to explore spurious behaviour when examining the graph structure only.

5.7.2 Pre-/Postset Optimisation

A further means to decrease the costs of determining the appropriate summary net is to inspect the initial marking of the contact place q and its pre-/postsets within the environment. Obviously, if the cutvertex has an empty postset within the environment, the environment net cannot consume a token, and if the cutvertex has an empty preset or if the cutvertex is initially marked, the environment net cannot produce a token. We can be even more precise, if we take reading transitions into account. Let $\texttt{read}(q)$ denote the set of transitions reading q and let $q_e^\bullet$ a short hand for the postset of q within the environment, $q_e^\bullet = q^\bullet \cap T_e$, and analogously ${}^\bullet q_e = {}^\bullet q \cap T_e$. An environment Σ_e cannot consume if $q_e^\bullet \setminus \texttt{read}(q) = \emptyset$, and Σ_e cannot produce

$\texttt{may\_produce} := {}^\bullet q_e \setminus \texttt{read}(q) \neq \emptyset \wedge M_{\mathsf{init}}(q) == 0$
$\texttt{may\_consume} := q_e^\bullet \setminus \texttt{read}(q) \neq \emptyset$

environment type	prerequisite
producer	may_produce
producer-consumer	may_consume $\wedge$ may_produce
unreliable producer	may_produce
dead end	may_produce $\wedge$ $(q_e^\bullet \neq \emptyset)$
consumer	may_consume

Table 5.1: Structural prerequisites for the environment types

if ${}^\bullet q_e \setminus \texttt{read}(q) = \emptyset$. It is easy to see, that an environment that can neither produce nor consume can be replaced by a Borrower net. Tabular 5.1 lists prerequisites for the different environment types. A Producer environment needs to be able to produce, hence there has to be a transition that can put a token onto the contact place q. Similarly, the producing environments Producer-Consumer, Unreliable Producer and Dead End need to be able to place a token on q. The Producer-Consumer also has to be able to consume the token. A Dead End environment has to consume a token from q or at least read q, since otherwise q could not be 1-safe in the original net. Using these criteria, the traversal of the decision tree (cf. Fig. 5.4) to determine the appropriate summary may be considerably shortened.

5.7.3 Order of Formulas

How many states are explored to determine the appropriate summary net, depends also on the order in which the formulas are checked on Σ_e. For instance if an oracle tells us that the environments are in majority producing, then we start by checking $\mathsf{AFG}(q, 1)$ first and postpone the check of $\mathsf{AG}(q, 0)$ (cf. Fig. 5.4). So if we have an environment net and its contact place q such that $|q^\bullet| = 0$, then we start by checking $\mathsf{AFG}(q, 1)$. Also $|q^\bullet| = 0$ implies that Σ_e cannot be a Producer-Consumer environment. If $|{}^\bullet q| = 0$, then Σ_e cannot generate a token on q. So we do not have to check $\mathsf{AG}(q, 0)$ on ${\Sigma_\mathrm{e}}^{q=0}$ or whether ${\Sigma_\mathrm{e}}^{q=1}$ is 1-safe.

5.7.4 Parallel Model Checking

In order to reduce the reduction time, our approach conveniently allows parallelisation of model checking: To identify the appropriate reduction rule for an environment, the five LTL$_{-\mathsf{X}}$ properties making up the rules' preconditions (cf. Fig. 5.4) can be checked in parallel. The outcome is then combined to determine the appropriate summary net.

5.8 Related Work

Since the first works on compositional reasoning in the beginning of 1980s, many works on compositional reasoning have been published. In the following we compare our works with the works we believe are most relevant to ours.

Decompositional Methods tailored to Petri Nets The work of Vogler and Wollowski et al. focuses on decomposition for logic synthesis. In [110, 93] an approach is presented to decompose Signal Transition Graphs—a version of Petri nets for the specification of asynchronous circuit behaviour—into components that together implement the global behaviour. The decomposition process starts from an initial partition of the output signals, which determines initial components. These components are then stepwise transformed by place deletion or transition contraction until a valid final decomposition is found. Logic synthesis is then applied to yield one circuit per component.

Concerned with verification is the work of Lee et al. [67], where a net is decomposed based on minimal linear invariants to check boundedness and liveness compositionally. The generated components may overlap, that is share places and transitions. The components' reachability graphs are reduced and (re)composed to analyse the global behaviour, where shared transitions have to be synchronised.

An iterative approach to decompose a monolithic Petri net for checking LTL$_{-\mathsf{X}}$ properties[2] is presented by Klai et al. in [62]. The generated com-

[2]There action-based properties are studied, i.e. the LTL$_{-\mathsf{X}}$ formulas refer to transition

ponents approximate the global behaviour. To reduce the risk of spurious behaviour, abstraction places are added representing the place invariants of the global net—and thereby representing an abstraction of the component's environment. If a component does not satisfy the property φ under consideration, the validity of the counterexample has to be checked by means of a so-called non-constraining relation, which represents the environment's constraints. If this relation is not satisfied, the net is reexamined under a coarser partition. In case φ holds on all components, it is checked on a reduced synchronised product.

In the approach of Lee et al. spurious behaviour is ruled out by synchronising the transitions shared among components. In the approach of Klai et al. spurious behaviour may lead to repartition such that the global behaviour is captured more accurately with each iteration. Our decomposition allows us to accurately characterise the influence of the environment net on the kernel net, so that neither local components need to be synchronised as in the approach of [67] nor an iterative refinement as in [62] is necessary.

Whereas the decomposition approach of Signal Transition Graphs [110, 93] is driven by an initial partition of output signals, the idea to use structural characteristics of a Petri net to derive a decomposition underlies our and the decomposition approaches of [67] and [62]. In our approach a small interface of just one place is chosen. Additionally this place has to be 1-safe. The decomposition approach of [67] is based on linear invariants and [62] rather mechanically derives components based on pre- and postsets but adds abstractions places representing linear invariants to reduce spurious behaviour.

Compositional Reduction/Minimisation of Petri Nets Valmari et al. and Juan et al. have published approaches to compositional reduction for Petri nets, that is they assume an initial decomposition of a net and minimise parts of the net preserving certain properties. Both present failure-based semantics for asynchronously communicating systems and state based information is preserved. Also, both works introduce algorithms to reduce

occurrences rather than to the token count.

the components' transition systems.

Valmari et Kaivola introduced in [105] two behavioural semantics, *chaos-free failure divergencies* (CFFD) and *nondivergent failure divergencies* (NDFD) semantics. They showed that CFFD is the weakest compositional equivalence with respect to parallel composition and hiding operators in CSP for synchronous communication systems preserving $\text{LTL}_{-\mathsf{X}}$ with an extra operator distinguishing deadlocks from divergencies. NDFD-equivalence is the weakest compositional equivalence preserving standard $\text{LTL}_{-\mathsf{X}}$. Although these results do not directly apply to asynchronous systems and place fusion, the semantics are the weakest known $\text{LTL}_{-\mathsf{X}}$ preserving semantics according to [102].

In [103] Valmari applied the above approach for state-based properties to (already decomposed) Petri nets whose subnets share a set of places only. It is assumed that the given net is divided into an environment component and an interesting component, i.e. a kernel. A labelled transition system representing the environment net's behaviour is condensed by a CFFD-semantics (or NDFD) preserving algorithm. The environment net is then replaced by a net corresponding to the condensed labelled transition system.

Juan et al. introduce in [59, 58] the condensation theories IOT-failure, IOT-state equivalence and firing-dependency theory. Their technique can analyse several state-based properties: boundedness, reachable markings, reachable submarkings and deadlock states. They develop condensation rules to minimise the components' transition systems.

We list the main differences to our work:

- Valmari et al. give an approach to preserve $\text{LTL}_{-\mathsf{X}}$ and divergencies/deadlocks and just $\text{LTL}_{-\mathsf{X}}$, respectively. Juan et. al. preserve boundedness, reachable (sub)markings and deadlock states. In our work we preserve at least $\text{LTL}_{-\mathsf{X}}$ but assuming fairness w.r.t. the kernel. A deadlock in a cutvertex reduct may not be a deadlock in the original net. We also preserve reachable (sub)markings.

- Whereas in [103] and [59] the replacement net is the result of a condensation, we identify six fixed summary nets that suffice to describe

the influence of any environment net.

- We use model checking to determine the replacement and thus can directly make use of the various methods that have been invented to speed up model checking. In contrast, Juan et al. use condensation rules on the components' transition systems and Valmari et al. transform the transition system into a kind of deterministic automaton and minimise this automaton using the functional coarsest partition algorithm [105].
- Our approach be conveniently be parallelised (cf. Sect. 5.7). The approaches of Valmari et al. and Juan et al. do not explore the potential of parallelisation.
- The works of Valmari et al. and Juan et al. do not consider the Petri net structure in their condensation procedures. Micro reductions can easily be combined with their condensation approaches, whereas pre-/postset optimisations are tailored to our classification of environment nets (cf. Fig. 5.4).
- We characterise cases when spurious behaviour may be encountered: Spurious behaviour may only be encountered when examining ${\Sigma_e}^{q=1}$ and this is necessary only when we replace a Borrower, Consumer or Dead End environment. Our approach also allows to disable these rules, so that no spurious behaviour may occur. Valmari acknowledges that spurious behaviour may be encountered and examines in [102] the use of interface processes as a means to decrease the risk of spurious behaviour. Juan et al. suggest the use a coverability graphs to at least identify components whose state space are infinite when examined in isolation. Both works lack a characterisation of "risky" components.

5.9 Future Work

Cutvertex reductions are able to replace any subnet connected to the kernel via only a 1-safe place by a summary net. This allows a successive replacement of subnets from the chain ends.

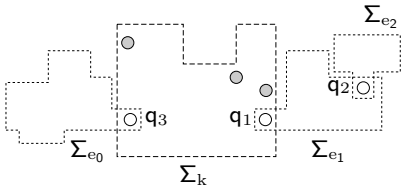


Figure 5.27: Decomposition by cutvertices. The chain of environments is tackled from the outside: First Σ_{e_2} is replaced and then Σ_{e_1}.

An interesting extension of the current approach are reduction rules that allow replacements from within a net, so that nets with a higher connectivity can also be reduced. Considering subnets with two contact places—we call them bridge subnets in the sequel—would allow replacements from within a net and can be combined with cutvertex reductions to yield a more fine grained decomposition, as illustrated in Fig. 5.28. Bridge subnets correspond to triconnected components, which can be determined in linear time [50].

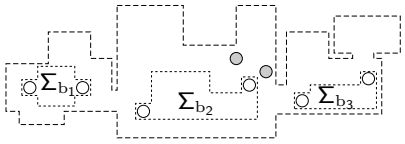


Figure 5.28: Decomposition into bridge nets. Bridge nets allow replacements from within a net and are an addition to cutvertex reductions.

In our benchmark set (cf. Sect. 6) 61 of 73 nets have bridge subnets.

Decompositions into triconnected components have already proven to be useful in the area of business process modelling, where certain bridge nets are replaced by a single transition. In [107] Vanhatalo et al. presented an approach to check *soundness* of workflow nets[3]. Their approach decomposed IBM WebSphere Business models, which are basically extended free choice nets, into single-entry-single-exit (SESE) fragments, that are certain bridge nets. Soundness of a workflow can be checked by checking soundness of each fragment in isolation. They also gave structural heuristics to show soundness or unsoundness of some fragments in linear time.

[3]Intuitively, a workflow net is sound iff every task is executable and the workflow process can properly terminate. For more information see Sect. 6.2

The above results characterise only when a free-choice SESE bridge net can equivalently be replaced by a transition. The extension of our approach is more general. Structural heuristics similar to micro reductions or as for SESE fragments as in [107] will certainly play an important role to efficiently determine the appropriate reductions. Also, an important aspect of the extension to bridge nets will be the identification of replacements that are risky in terms of encountering spurious behaviour when identifying the appropriate reduction.

5.10 Conclusion

We presented a decompositional approach to alleviate the state explosion problem of model checking an $\mathrm{LTL}_{\text{-}\mathsf{X}}$ formula φ. We suggested to decompose the Petri net into a kernel net containing $\mathit{scope}(\varphi)$ and environment nets so that kernel and environment share a 1-safe place only. For a 1-safe net the decomposition can be determined in linear time and every environment net can be replaced. To determine the applicable reduction an environment net is checked in isolation, thus avoiding the combinatorial blow-up. The empirical evaluation of this approach is presented in the next chapter.

Chapter 6

Evaluation

Contents

Many approaches exist to combat state space explosion, as outlined in Chapter 3. In this work we developed two further techniques for Petri nets, slicing and cutvertex reductions. When introducing a new method an important question is how the new method compares with existing techniques. In the Chapters 4 and 5 we discussed conceptual differences and similarities to existing approaches, in particular to agglomerations, compositional reductions and POR. This section is dedicated to empirically evaluating, how our methods perform in comparison to and in combination with this three approaches.

In Sect. 6.1 we will first compare the effects on the full state space of cutvertex reductions, safety slicing and $\mathrm{CTL}^{*}_{-\mathsf{X}}$ slicing with agglomerations

method	references	used tool
$CTL^{*}_{-\mathsf{X}}$ slicing	[89]	own implementation
safety slicing	Sect. 4.3	own implementation
cutvertex reduction	[88]	own implementation
agglomerations	[9, 51]	implementation following [51]
CFFD reduction	[105, 103]	TVT [100]
stubborn set reductions	[104, 102]	PROD [83]

Table 6.1: Methods in the evaluation. The first three methods have been developed by the author. Their performance is compared to agglomerations, stubborn set reduction and compositional reduction based on CFFD semantics (cf. Sect. 3).

and compositional reduction based on CFFD semantics (cf. Sect. 6.1.4). We then analyse the performance of the methods on state spaces condensed by stubborn set reductions (Sect. 6.1.5). Table 6.1 lists the original references for the considered methods.

Section 6.2 illustrates the usefulness of our reductions for business process management. Therefore we first give a brief introduction to business process management and the role of Petri nets therein and then study the effects of our techniques on a small case study.

We believe that rather than presenting cases where our methods work best, it is equally interesting to see where other methods excel, so that we will present a comparative evaluation on a benchmark set.

In oder to compare our techniques to agglomerations, CFFD and partial order reductions, we developed a generic and fully automatic evaluation procedure. This evaluation procedure works on any benchmark set and does not require that temporal properties are specified for the benchmark nets. The evaluation procedure is introduced in Sect. 6.1.1.

To compare the different techniques we defined key indicators that summarise the performance of a technique on the benchmark set and thereby allow a comparison to key indicators of other methods on the same benchmark set. In order to give an impression on the absolute effect of the methods on the benchmark set, we describe their impact in terms of percental state

space savings.

In Sect. 6.1.2 we introduce and briefly characterise our benchmark set. In Sect. 6.1.4 we examine $\mathrm{CTL}^*_{-\mathsf{X}}$ slicing, safety slicing, cutvertex reductions, agglomerations and CFFD reduction on the full (=uncondensed) state space. We start Sect. 6.1.5 by analysing the effect of combining slicing and cutvertex reductions. We then examine the combination of the five methods with partial order reductions to analyse whether these methods yield (further) reductions on condensed state spaces.

6.1 Comparative Evaluation on a Benchmark Set

In this section we introduce our generic evaluation procedure that was employed to compute our key indicators. The evaluation procedure does not rely on given temporal properties and works on any benchmark set. Key indicators summarise the impact of the reductions and thereby allow a comparison of different techniques. The key indicators are discussed in Sect. 6.1.1.1.

6.1.1 A Generic Evaluation Procedure

An ideal evaluation procedure to collect data for a comparison of different methods may be sketched as follows:

```
1  forall nets Σ in the benchmark set {
2     forall relevant temporal properties φ of Σ {
3        model check φ on Σ;
4        forall methods m of the comparison {
5           use method m when model checking φ;
6           assess/quantify the effect of m;
7  } } }
```

Every net of a benchmark set is examined for all relevant properties. The impact of the examined method is analysed by comparing its effect to the

results without reductions. Based on the collected data, a comparison or further analysis can be applied.

Unfortunately, it is usually infeasible to determine all relevant properties and it is not possible to determine *automatically* which sets of places of a net correspond to *interesting* properties. But as all methods examined in this section take care of the places referred to in the temporal property, the choice of the property can be an important influence on the reduction effects.

As we use a rather large benchmark set, the evaluation procedure has to be automatic. So we chose to apply the methods for every single place of the net. This seemed to be a reasonable choice, because all examined methods work the better the fewer places are used and because applying the methods for every one-elementary place set already requires extensive computational resources.

We refrained from measuring the model checking performance for automatically built temporal properties referring to the single place, as we cannot tell which properties are interesting. Instead we measured the reductions' effect in terms of state space decrease (=the decrease in the number of states and state transitions).

Our generic evaluation procedure is sketched below:

```
1  forall nets Σ in the benchmark set {
2     generate Σ's state space;
3     forall places p of Σ {
4        forall methods m of the comparison {
5           generate Σ's state space using m and
6           observing p;
7           assess/quantify the effect of m;
8  } } }
```

Quantifying Reduction Effects by State Space Decrease Using the state space decrease to quantify the reduction effect enables us to run the benchmark on different machines, and also to examine nets with small state spaces where measuring time becomes problematic for technical reasons.

Undoubtedly the size of the state space has a strong influence on time and space needed for model checking. LTL model checking can be performed in $\mathcal{O}(|TS_{\mathcal{M}}| \cdot 2^{|\psi|})$ space and time, and CTL model checking is in $\mathcal{O}(|TS_{\mathcal{M}}| \cdot |\psi|)$. By using the state space as a measure, we neglect the influence of the temporal property. So state space savings may be multiplied when model checking temporal properties on the reduced system.

Filtering Not every place necessarily corresponds to a meaningful temporal logic formula. For a fair comparison we have to take into account how this decision influences the different methods of the comparison.

Slicing starts with the slicing criterion place and iteratively includes other relevant parts of the net. Slicing produces especially small reducts when the slicing criterion is within an *initialisation subnet* as illustrated in Fig. 6.1. In this case slicing discards everything but this initialisation subnet. The places of such a small initialisation subnet probably do not represent any interesting property.

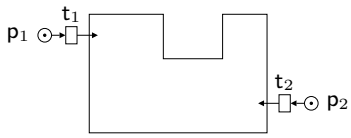


Figure 6.1: Filtering out smallest slices is necessary: The slice $\mathit{slice}(\Sigma, p_1)$ for place p_1 consists of place p_1 and transition t_1 only. Similarly $\mathit{slice}(\Sigma, p_2)$ consists only of p2 and t2 only.

Cutvertex reductions are similarly able to fold away anything but this initialisation subnet. Agglomerations on the other hand may generate small reducts by compressing subnets between two places. So that a small agglomeration reduct may summarise behaviour of a big subnet.

We hence applied a filter eliminating the smallest reducts of slicing and cutvertex reductions. Based on inspections of the smallest reducts, a reduct is considered as meaningful, if it has at least 20 states and in case it has less than 3 % of places, it has to have at least 5 transitions and places.

6.1.1.1 Key Indicators

Our evaluation procedure generates a data flood: We take a measurement for every place of every net of the benchmark set for every examined technique. *Key indicators summarise the measurements to quantify the performance of the examined technique on the benchmark set.* The key indicators have to be chosen carefully, to sufficiently reflect the impact of a technique.

One difficulty is that the effect of our reductions on the nets is very heterogeneous. Table 6.2 illustrates the effect of $\text{CTL}^*_{-\mathsf{X}}$ slicing on four very different nets. Whereas the net *dac_15* (modelling a divide and conquer computation) has many slices of small to medium size, *elevator_4* (modelling a controller for 4 elevators) has only very small and very big slices. Another extreme example is *furnace_4* (modelling temperature management of 4 furnaces) where slicing only removes reading transitions, which results in substantial reduction in the number of state transitions but in no reduction of the number of states.

Table 6.2: Four exemplary nets of the benchmark. For every net (a) gives net size, its state space size and the number of properly effective slices, and (b) gives the percental size of its smallest and biggest properly effective slice and percentage of places covered by some properly effective slice.

	system	$\lvert\Sigma\rvert$ (places,trans.)	$\lvert TS_\Sigma\rvert$ (states,state trans.)	#properly effective reducts
(a)	dac_15	105,73	114685, 794642	44
	elevator_4	736, 1938	47436, 150066	10
	furnace_4	66, 139	221041, 1757106	1
	q_1	163, 194	123596, 584896	14

The large number of reduced nets (e.g. 1388 $\text{CTL}^*_{-\mathsf{X}}$ slices have been built) paired with the heterogeneity of the reduction effects make a concise summary difficult.

As a first step we categorise the reduction effects into *trivial, proper, effective, properly effective* and *limited effective*. If the reduced net equals

(b)

system	smallest reduct [%] (states, state trans.)	greatest reduct [%](states,state trans.)	covered [%] places
dac_15	0.11, 0.03	49.99, 46.39	90.48
elevator_4	0.05, 0.03	100, 98.52	99.86
furnace_4	100, 89.24	100, 89.24	100
q_1	75.97, 75.26	95.19, 95.05	96.93

the original, the reduced net (or reduct) is called *trivial*. If the reduced net differs from the original, we call the reduced net *proper*. A reduced net is called *effective*, if its state space is smaller than the original's state space. Our approaches guarantee that the state space of a reduct is at most as big as the original's state space. When applying cutvertex reductions, an overhead may be caused when the appropriate summary for an environment is determined. A reduced net is called *limited effective* if its state space together with the states and state transitions inspected to determine the appropriate summary net is bigger than the original's state space. A reduced net is called *properly effective* if its state space together with the states and state transitions inspected to determine the appropriate summary is less than the original's state space.

Our benchmark set (cf. Sect. 6.1.2) contains several scaled up instances. We say that nets generated from the same system blue print belong to the same *family*. These nets are of a similar structure and hence the reductions' effects on the net graph are similar.

To allow a succinct comparison for the different approaches we chose the following values as key indicators:

1. the number of families with properly effective/limited effective reduced nets,

   The number of properly effective families gives a rough estimate of the scope of a technique. The number of limited effective reduced nets measures cases when the techniques cause an overhead. Of the here considered methods only cutvertex and CFFD reductions can cause limited effective reducts.

2. the mean size of the state space and of the Petri net graph of reduced nets,
   Some families are better reducible than others. To reflect the influence of each family equally, we built the mean over all families by first computing the mean over all reducts of a net (after applying the filter), and then the mean per family to then determine the mean over all families.

   As the mean values are computed considering all nets of the benchmark but not all nets are reducible by a given method, the untouched nets dilute the reduction effect and make the savings seem marginal. But for the comparison of different techniques we need to consider every net of the benchmark. These two mean values describe the total effect on the benchmark set.

3. the mean (place) coverage for a state space saving of y.
   A mean coverage of x% expresses that in average for x% percent of the places in the original net there is an effective reduced net saving at least y.

We say that we have a *saving of* x of the states (state transitions / state space) per reduct, if the reduced net has factor x less states (state transitions / state space size) than the original net. Analogously, we say that we have an *expense or cost of* x of states (state transitions / state space), if factor x of the states (state transitions / state space) of the original's state space has been inspected to determine the summary. We refer to the difference of saving minus expense as *benefit*. If the difference of saving minus expense is negative, we also refer to the benefit as *overhead*. The *mean saving per net* is the average build over the reducts for every place of the net. The *mean saving per family* is the average build over mean savings per net of all reducts of the family. Analogously, we use *mean expense, mean overhead* and *mean benefit* per net and per family. We say that a net Σ has a saving of x, if there is a reduct of Σ with a saving of x.

The key indicators listed above are generated to allow a succinct comparison. To show how great the reduction effect of a method can be, we give the greatest percental savings for every benchmark net and the coverage for

a percental saving of at least 10%.

Before we present the results of our evaluation (cf. Sect. 6.1.4 and 6.1.5), we first introduce the set of examples we use as a benchmark, and then briefly discuss tool specific issues for the evaluation.

6.1.2 The Benchmark Set

To evaluate our approach we used a set of case studies of James C. Corbett [27, 26]. Originally this set of systems was compiled to study the methodical issues of empirically comparing different deadlock detection techniques on Ada tasking programs. The set of benchmark examples consists of real Ada tasking programs as well as standard benchmark examples from the concurrency analysis literature as Corbett states in [26]. The aim was to collect as many examples as possible to cover a wide range of systems with different characteristics. In the benchmark set there are five non-scalable systems and seventeen systems were scaled and are present in four different sizes.

In the original publication the systems were used in various formats, among others in the input formalism of the SPIN model checker [98]. Petri net encodings of several systems of this benchmark have been used in other publications e.g. in [72, 37, 35, 55, 56, 57, 61, 60]. To make all systems of the benchmark available, we implemented a translation from the SPIN encodings to Petri nets —which is done straightforwardly and has been validated by using SPIN.

Table 6.2 characterises the benchmark nets. Column *state space* gives the state space of the biggest system instance; in case of the non-scalable examples, which are at the bottom of the table, there is thus one instance only. The column titled m gives the step sizes for the scalable families. In the sequel we frequently refer to the nets by their names as in the original benchmark set, which are mnemonics for the description as given in Table 6.2.

system	description	state space	m
cyclic(m)	cyclic scheduler	77822, 501760	$3, 6, 9, 12$
dac(m)	divide and conquer computation	114685, 794642	$6, 9, 12, 15$
dp(m)	dining philosopher	531440, 4251516	$6, 8, 10, 12$
dpd(m)	phil. with dictionary	111444, 674139	$4, 5, 6, 7$
dpfm(m)	phil. with fork manager	200, 1211	$2, 5, 8, 11$
dph(m)	phil. with host	104680, 615875	$4, 5, 6, 7$
elevator(m)	controller for m elevators	47436, 150066	$1, 2, 3, 4$
furnace(m)	temperature data management of m furnaces	221041, 1757106	$1, 2, 3, 4$
gasnq(m)	non queueing self-service gas station	115743, 430703	$2, 3, 4, 5$
gasq(m)	queueing self-service gas station	15430, 46681	$1, 2, 3, 4$
hartstone(m)	program which starts and stops worker tasks	202, 202	$25, 50, 75, 100$
key(m)	keyboard/screen interaction management	398760, 1041364	$2, 3, 4, 5$
mmgt(m)	memory management scheme	66308, 218955	$1, 2, 3, 4$
over(m)	highway overtake protocol	33506, 163618	$2, 3, 4, 5$
ring(m)	token ring mutual exclusion protocol	211684, 1188828	$3, 5, 7, 9$
rw(m)	reader writers on a database	32784, 491551	$6, 9, 12, 15$
sentest(m)	sensor test program	381, 777	$25, 50, 75, 100$
abp	alternating bit protocol	112, 167	1
bds	border defence system	36096, 263302	1
ftp	file transfer protocol	113927, 805043	1
q	RPC based client/server user interface	123596, 584896	1
speed	program monitoring and regulating speed	8689, 30369	1

Figure 6.2: Characteristics of the benchmark set

6.1.3 Tools in the Evaluation

In the sequel we compare our methods—CTL^*_{-X} slicing, safety slicing, cutvertex reductions—with agglomerations, CFFD reductions and also examine whether the methods further reduce state space condensed by POR (=partial order reduction).

Agglomerations/Slicing As listed in Table 6.1 we implemented the pre- and postagglomerations as described in [51]. The costs of applying agglomerations (and slicing) is linear to the size of net graph, so that we neglect their application costs.

The two decompositional reduction methods—cutvertex and CFFD reductions—both examine the state space of subnets to determine the appropriate replacement.

Cutvertex Reductions For cutvertex reductions we can exactly determine the costs, as the employed model checkers output the number of inspected states and state transitions.

CFFD Reductions The tool TVT [100] is the only current tool known to us that allows to reduce a given labelled transition system (LTS) based on the CFFD semantics. To apply CFFD reductions on Petri nets as described in [103], we used the same decomposition as for cutvertex reductions and generated the LTS of the resulting environment nets. In [103] an LTS is generated assuming that the contact place has arbitrarily many tokens, so that environment transitions are never disabled because of the marking on the contact place. We instead generated the LTSs by adding the knowledge about 1-safeness of the contact places, so that our generated LTS is at most as big as in [103]. We used TVT to reduce the LTS. The reduced LTS is then translated back to a Petri net and recomposed with the kernel as described in [103].

For CFFD reduction, TVT runs a script on the LTS, that transforms the LTS into a so-called acceptance graph, normalises and reduces it and

transforms the acceptance graph back to an LTS. We roughly approximate the cost for the LTS condensation as the sum of the sizes of the input LTS plus twice the size of the output LTS. This underestimates the actual costs, as the script applies three times reductions that are in O(N log N) [76] plus transforms the LTS and normalises the initial acceptance graph.

Partial Order Reductions We used several partial order tools on the given examples, namely SPIN, Tina and PROD. PROD was chosen for the comparison as its stubborn set reductions yielded by far the greatest reductions on the benchmark set.

6.1.4 Effect on the Full State Space

In this section we present the results of the five methods—CTL$^*_{-\mathsf{X}}$ slicing, safety slicing, agglomerations, cutvertex and CFFD reductions—on the full state space. In Sect. 6.1.4.1 we will examine the effect of the structural optimisations implemented for cutvertex reductions. In Sect. 6.1.4.3 we discuss how concurrency and model size influence the effectiveness of our methods. In Sect. 6.1.4.2 the role of limited effective reducts is examined.

We start by inspecting the key indicators summarised for the five methods in the tables 6.3-6.5. If we consider the savings in terms of the state space, CTL$^*_{-\mathsf{X}}$ slicing and cutvertex reductions have about the same impact on this benchmark set. By far, safety slicing gains the greatest savings and CFFD reductions the least savings. Agglomerations affect the fewest families, whereas cutvertex reductions affect the most families. But on closer inspection, we see that about one third of the nets affected by cutvertex reductions, is only marginally affected.

The mean savings in tables 6.3 and 6.4 may seem marginal, but keep in mind that the mean values are computed over all nets of the benchmark and unreduced nets dilute the effect on very effectively reduced nets. Figure 6.3 illustrates the state space savings the five techniques are capable of by clustering the nets according to their greatest savings.

	$\frac{\#\text{properly red. families}}{\#\text{familes}}$	# properly effect. reducts	mean state space size *(states, state trans.)*
safety slicing	10/23	714	(33528.43, 202306.84)
$\text{CTL}^*_{-\mathsf{X}}$ slicing	9/23	625	(37152.91, 211526.34)
cutvertex reds.	17/23	183	(37063.17, 212210.59)
CFFD reds.	8/23	64	(39772.73, 232651.90)
agglomerations	7/23	124	(38155.49, 222034.95)

Table 6.3: Mean savings on the full state space I

	mean state space savings *(states, state trans.)*	mean net graph savings *(places, trans.)*
safety slicing	(0.16, 0.13)	(0.10, 0.03)
$\text{CTL}^*_{-\mathsf{X}}$ slicing	(0.07, 0.09)	(0.09, 0.02)
cutvertex reds.	(0.07, 0.09)	(0.01, 0.01)
CFFD reds.	∼(0,0)	∼(0,0)
agglomerations	(0.04, 0.05)	(0.01, 0.01)

Table 6.4: Mean savings on the full state space II

	$\frac{\#\text{properly red. families}}{\#\text{familes}}$	coverage on reducible *places* [%]	coverage on all nets *places* [%]
safety slicing	9/23	85.36	35.39
$\text{CTL}^*_{-\mathsf{X}}$ slicing	6/23	48.38	18.58
cutvertex reds.	6/23	99.48	21.48
CFFD reds.	5/23	9.48	1.64
agglomerations	3/23	99.33	11.27

Table 6.5: Reducts with a state space saving of 10%

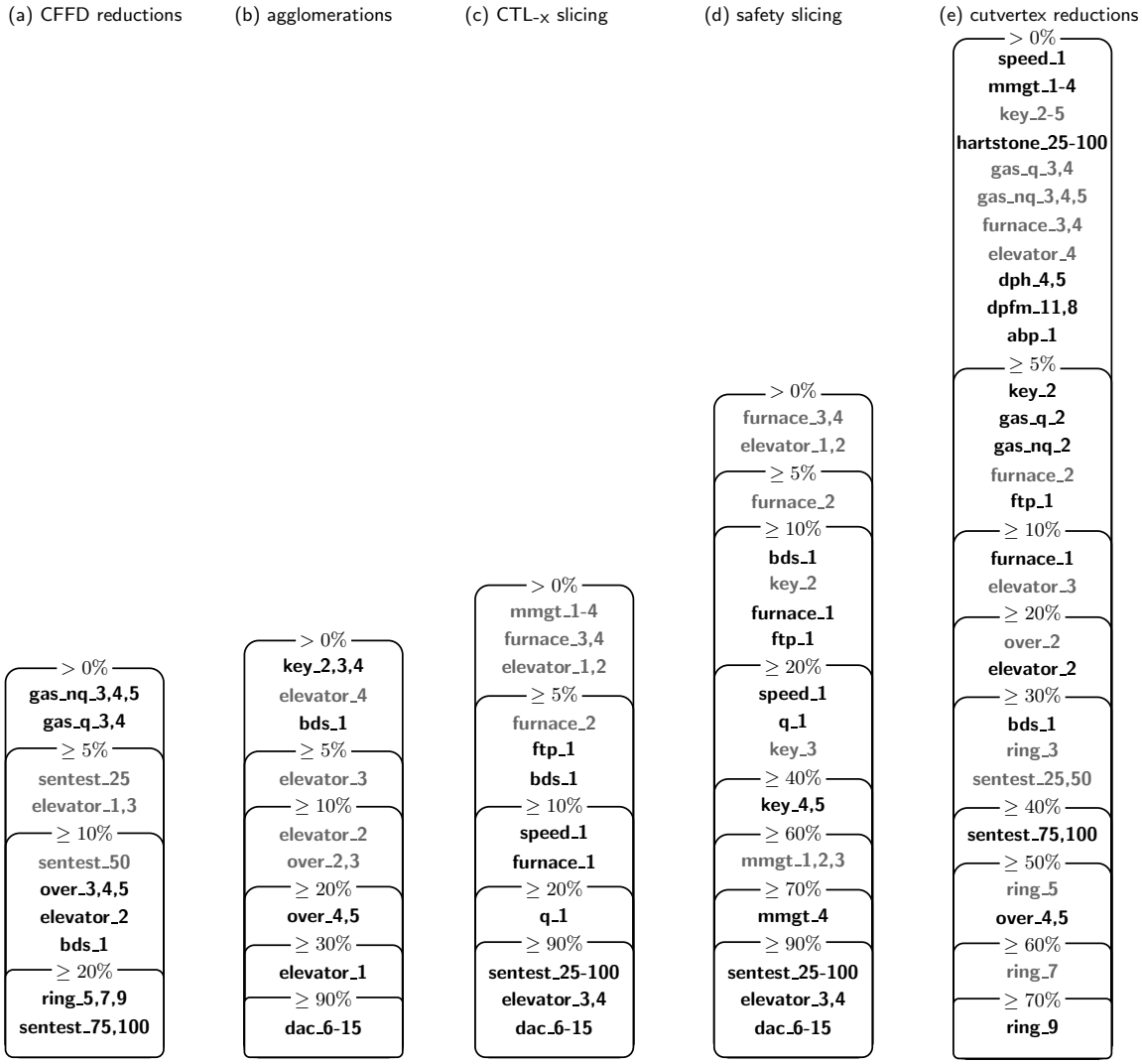


Figure 6.3: Properly effective reduced nets clustered by their state space savings. Each column displays the savings of the respective reduction technique. A net name appears within a cluster when a net has a properly effective reduct with a saving within the cluster's range. The earliest occurrence of a family is marked in black.

Figure 6.3 indicates that the five techniques have different capabilities, i.e. some nets are very effectively reduced by one method but not by the other. For instance nets of the *dac* family are effectively reduced by agglomerations and slicing but not by cutvertex and CFFD reductions, and nets of the *over* family are effectively reduced by agglomerations and cutvertex reductions but not by slicing. The *ring* nets are only reducible by cutvertex and CFFD reductions and only safety slicing has a real reduction impact on *mmgt*.

We already know from the Table 6.4 that safety slicing gains the greatest savings on this benchmark set. Comparing the results of $\mathrm{CTL}^{*}_{-\mathsf{X}}$ slicing (c) and safety slicing (d) in Fig. 6.3 shows that safety slicing is also able to reduce more nets more effectively than $\mathrm{CTL}^{*}_{-\mathsf{X}}$ slicing.

We noticed in Table 6.3 that cutvertex reductions affect a large number of nets. Figure 6.3 (e) shows that cutvertex reductions also affect the widest range of families. But only for 10 of 16 properly effective reduced families the savings are at least 5% of the state space. Cutvertex and CFFD reductions are the only methods examined here that can cause limited effective reducts. For cutvertex reduction the *rw* family had limited effective reducts only and only the *rw* family had limited effective reducts. In total there were nine limited effective reducts. In average 0.69 states and 1.2 state transitions were inspected to determine the appropriate summaries.

6.1.4.1 Structural Optimisations for Cutvertex Reductions

We have already analysed the effect of cutvertex reductions using micro reductions and pre-/postset optimisations. In this section we analyse the effect of these optimisations. Therefore we apply cutvertex reductions *(i)* without any structural optimisations, *(ii)* using pre-/postset optimisations, and *(iii)* using pre-/postset optimisations and micro reductions.

The optimisations do not change the generated reducts but reduce the costs of determining the summary. As it turned out the application of the optimisations did not change the average benefit. Even without any optimisations the costs were small enough not to decrease the benefits of the overall benchmark set, but Table 6.6 shows that the optimisations decrease

the mean cost of determining the appropriate summary and further increase the number effective reducts.

	mean cost *(states, state trans.)*	# effective reducts	# limited effect. reducts
no optimisations	14.73, 15.24	171	29
pre-/postset	6.05, 6.56	172	26
micro & pre-/postset	0.69, 1.2	183	9

Table 6.6: Cutvertex reductions and structural optimisations

As we have seen cutvertex reductions generated nine limited effective reducts even with both optimisations—all reducts are from the *rw* family. Without optimisations 29 limited effective reducts are generated from seven different families. In the following we will analyse limited effective reducts, in particular we will discuss the meaning of limited effective reducts for model checking and demonstrate that a limited effective reduct may in certain cases accelerate model checking nevertheless.

6.1.4.2 Limited Effective Reductions

In Sect. 6.1.1.1 we chose the number of limited effective reducts as an indicator for cases when cutvertex reductions do not pay off and we defined a reduct to be limited effective iff the state space of the original unreduced net is smaller than the state space of the reduct plus states and state transitions inspected to determine the appropriate summary. We believe that counting limited effective reducts is a good enough indicator to study general effects, but it is not accurate. A limited effective reduced net does not necessarily mean an overhead when it comes to model checking temporal properties. When a complex formula is verified, the reductions' savings may pay off the reduction costs. Let us consider as an example the reducts of the *rw* family illustrated in Table 6.7. As already mentioned, all reducts of the *rw* family are limited effective.

By means of the net rw_12 Table 6.8 illustrates that limited effective reducts may accelerate model checking. For each of its limited effective reducts

	$\|TS\|$ (states, state trans.)	#insp. (states, state trans.)	$\|\Sigma\|$ (places, trans.)	overhead [%]
rw_15	32784,491551	–	78,481	–
rw_15_red1	17,31	32927,491717	32,31	0.07
rw_15_red2	32769,491521	35, 63	46,451	0.01
rw_12	4109, 49177	–	63,313	–
rw_12_red1	14,25	4215,49298	26,25	0.5
rw_12_red2	4097,49153	29,51	37,289	0.08
rw_9	522,4627	–	48,181	–
rw_9_red1	11,19	533,4637	20,19	0.99
rw_9_red2	513,4609	23,39	28,163	0.68
rw_6	71,397	–	33,85	–
rw_6_red1	3,2	114,455	2,2	22.65
rw_6_red2	8,13	97,428	14,13	16.67
rw_6_red3	65,385	17,27	19,37	5.56

Table 6.7: Cutvertex reductions on the *rw* family.

a formula is given that is checked more effectively on the reduct taking into account the costs of determining the summary. Note, that CFFD reductions also generated limited effective reducts for the *rw* nets, but no other method examined here (including stubborn set reduction) was able to properly effectively reduce nets of the *rw* family!

# insp. on rw_12 (states, state trans.)	reduced net	# insp. on red. (states, state trans.)	benefit [#] (states, state trans.)
$\mathsf{AG}((\mathsf{state\_2\_17}, 0) \Rightarrow (\mathsf{F}(\mathsf{state\_15\_0}, 0)))$			
8230,147554	rw_12_red1	42,128	3973,98128
$\mathsf{A\,G}\,((\mathsf{state\_2\_2}, 0) \Rightarrow ((\mathsf{F}\,(\mathsf{state\_14\_0}, 0)) \wedge (\mathsf{F}\,(\mathsf{state\_10\_0}, 0))))$			
17426,478298	rw_12_red2	17382,478112	15,135

Table 6.8: Model checking *rw_12* and its reducts.

We analysed the cases when cutvertex reduction without optimisations cause an overhead. In all but one case an overhead was caused by replacing very small environments. The *rw* family is the only family whose reduction caused an overhead while not having very small environments.

6.1.4.3 Scalability, Concurrency and Model Size

We now briefly discuss for slicing and cutvertex reductions whether the gained state space reductions scale with the system/model size.

How much of the model in terms of the net graph is discarded depends on the model structure for both approaches, whereas the effect of discarding model parts depends on the system dynamics.

So there are systems like *sentest* (Table 6.9) for which the savings decrease with increasing system size but also like *dac* (Table 6.10) where a system is more effectively sliced with increasing model size.

system	state space savings [%] (states, state trans.)	net graph savings [%] (places,trans.)	covered [%]
sentest_25	85.57, 91.02	62.97, 77.95	93.27
sentest_50	84.75, 91.35	64.64, 56.52	96.09
sentest_75	83.65, 90.95	65.27, 55.33	97.24
sentest_100	82.7, 90.44	65.61, 54.48	97.87

Table 6.9: Mean savings and coverage of the sentest family as percentage.

When measuring the reductions' effects as the savings of the original's state space, a strong influence on the savings is the degree of concurrency between the remainder (=kernel,slice) and the net discard (or environment, respectively). This effect is illustrated in Figures 6.4 and 6.5 for CTL* slicing. Slicing saves one-third of states for the sequential system given in Fig. 6.4 but for the concurrent system it saves two-thirds (cf. Fig. 6.5) although as many places and transitions are discarded.

system	state space savings [%] (states, state trans.)	net graph savings [%] (places,trans.)	covered [%]
dac_6	75.17, 78.93	61.12, 50.07	88.1
dac_9	76.03, 78.08	48.78, 35.88	92.06
dac_12	82.12, 83.23	51.17, 38.49	94.05
dac_15	85.72, 86.41	53.7, 41.77	90.48

Table 6.10: Mean savings and coverage of the dac family as percentage.

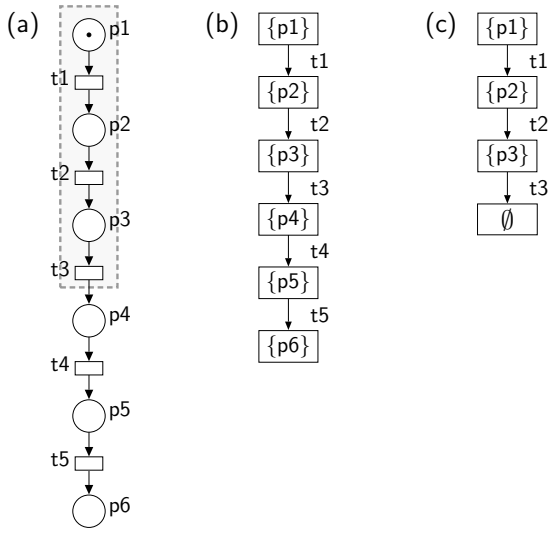


Figure 6.4: Slicing sequential systems. (a) a sequential net system Σ_s with $slice(\Sigma_s, p3)$ within the dashed area, (b) state space of the original, (c) state space of $slice(\Sigma_s, p3)$

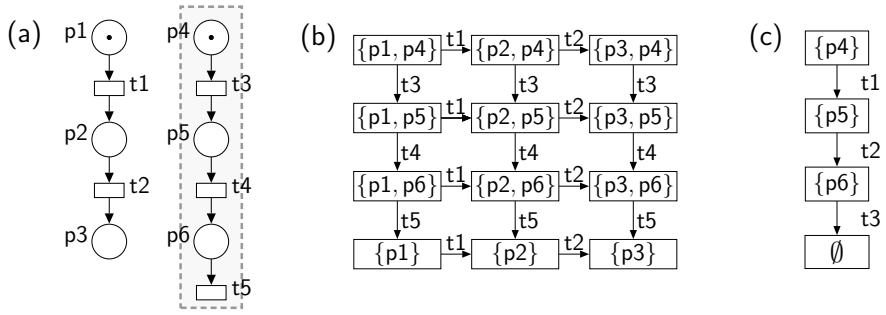


Figure 6.5: Slicing concurrent systems. (a) a concurrent net system Σ_c with slice $slice(\Sigma_c, p5)$ within the dashed area, (b) state space of the original, (c) state space of $slice(\Sigma_c, p5)$

Both approaches, slicing and cutvertex reductions, may gain greater savings by eliminating concurrency: If a *slice* is properly effective, then causal dependencies have been truncated or concurrent behaviours have been omitted. A net is properly effectively reduced by cutvertex reductions (without optimisations), if an environment has been replaced by a summary net and the cost of determining the appropriate summary is less than the state space reduction gained by the replacement.

So, whether a net is reducible depends on the model structure, whereas the impact of the reductions depends on the system dynamics. Both methods profit from concurrency. As the dynamics is difficult to predict by just studying the model structure, the impact of the reductions is difficult to predict as well.

6.1.4.4 Summary and Conclusions

The results of this section show that slicing and cutvertex reductions can efficiently speed up model checking. On the benchmark set $\mathrm{CTL}^{*}_{-\mathsf{X}}$ slicing and cutvertex reductions save about the same percentage, whereas agglomerations gain less savings on the state space. All three methods differ in the range of effected nets.

Safety slicing is the most effective method. It gains the greatest reductions on the state space and the net graph.

6.1.5 Alliance Against State Space Explosion

In this section we will evaluate combinations of different methods.

In the first subsection we examine cutvertex reductions on slices. As we will see, slicing may lead to additional articulation points. Thus we want to evaluate whether the combination of slicing and cutvertex reductions has a synergetic effect on the benchmark set.

We pointed out that cutvertex reductions yield savings primarily by avoiding the state space blow-up caused by concurrency and that slicing profits from concurrency within the original system. With POR a class of methods exists that has been especially developed to avoid the blow-up caused by con-

current behaviours. In the following we hence empirically examine, whether our methods combined with POR contribute to further reductions or whether their effect is subsumed by POR or even adversary to POR. In Sect. 6.1.5.2 we use condensed state spaces as reference state spaces to analyse the impact of our reductions.

6.1.5.1 Cutvertex Reductions on Slices

In this section we evaluate the effect of slicing followed by cutvertex reductions on the benchmark set.

The results of the previous section indicate that slicing and cutvertex reductions affect a different range of nets, so the effects of the one method are not entirely subsumed by the other's. Slicing can generate further articulation points, as Figure 6.6 (from [11]) illustrates. Hence slicing a net before applying cutvertex reductions can lead to further savings.

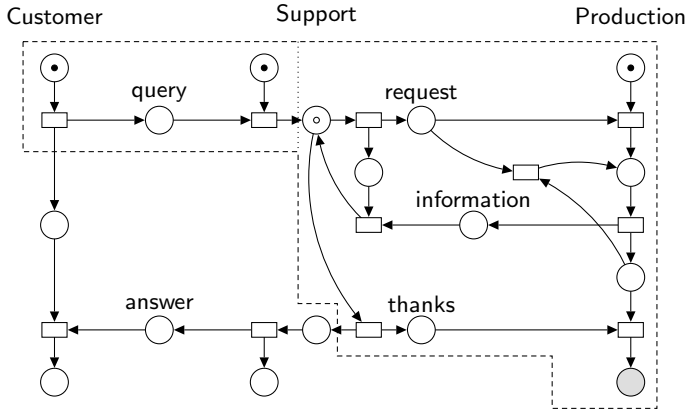


Figure 6.6: A customer/support/production system.
Slicing makes cutvertex reductions possible. The original net has no articulation point, its slice has (i.e. the place marked with the hollow token). Cutvertex reductions on the slice (within dashed lines) yield the reduct to the right of the dotted line and with the hollow token.

But —as Fig. 6.7 shows— the effect of slicing and cutvertex reductions is for certain nets the same.

In the following we present the results of applying *(i)* $\mathrm{CTL}^{*}_{-\mathsf{X}}$ slicing followed by cutvertex reductions and *(ii)* safety slicing followed by cutvertex

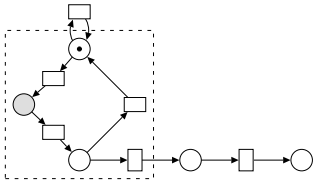


Figure 6.7: Truncation of chain ends by slicing and cutvertex reductions. The reduced/sliced net is displayed within the dashed area.

	# properly effect. reducts	mean state space size (states,state trans.)	mean # insp. (states,state trans.)	mean state space savings (states,state trans.)
CTL^*_{-X} slicing	270	34723.63, 197289.74	2.98, 3.78	0.13, 0.15
safety slicing	594	30833.83, 186874.7	0.72, 1.23	0.22, 0.2

Table 6.11: Mean values of cutvertex reductions on slices

reductions.

Table 6.11 summarises the key indicators for cutvertex reductions on CTL^*_{-X} and safety slices. In both cases 18 families had properly effective reducts.

CTL^*_{-X} Slicing and Cutvertex Reductions The results of Table 6.11 show that the combination of cutvertex reductions with slicing increases the state space savings, i.e. applying both methods yields greater savings than just applying one method. Comparing the results with those in tables 6.3 and 6.4, we see that the state savings of slicing and cutvertex reductions approximately add up (even better if no filtering is applied) whereas the reductions in state transitions are slightly less than the sum.

In general, applying cutvertex reductions *after slicing* bears an increased risk of an overhead w.r.t. the state space *of the slice*, since slicing might have reduced the concurrency within the net so much that cutvertex reductions do not pay off any more (cf. Fig. 6.8). When analysing the combined application, we are interested in the total effect. This means for example, *(i)* if slicing very efficiently reduces the original system—*lets say it saves* 80%—

but cutvertex reductions on the slice are limited effective—*lets say it has an overhead of* 20% *on the slice*—, the total effect is beneficial—*saving 79.6*%, but also *(ii)* if slicing has no effect, but cutvertex reductions very efficiently reduces a net, the total effect equals the effect of cutvertex reductions.

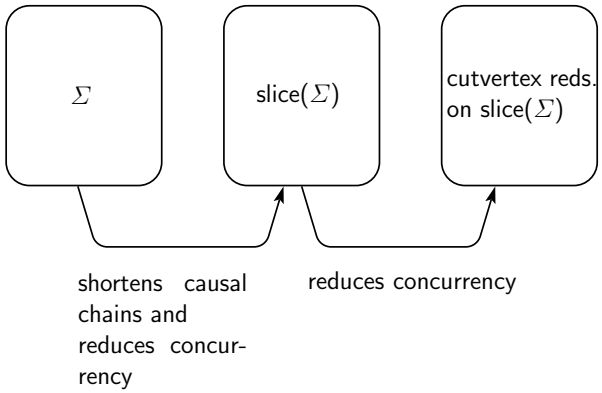


Figure 6.8: Daisy chaining slicing and cutvertex reductions

The coupled application generated more limited effective slices. This is possible as slicing generates additional cutvertices. Whereas applying only cutvertex reductions resulted in only the *rw* family having limited effective reduced nets, the combined application generated two additional limited effective reduced nets for *dac_6*. In both cases an environment as shown in Fig. 6.9 is replaced by an Unreliable Producer environment. 9 states and 10 state transitions were inspected to determine this replacement.

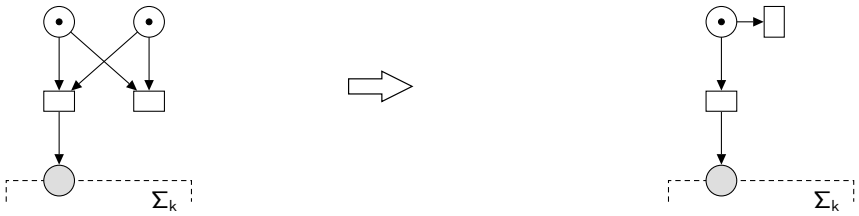


Figure 6.9: Limited effective reduction of dac_6.

For nets of the *dac* family applying cutvertex reductions after CTL^{*}_{-X} slicing caused an overhead with respect to the state space solely reduced by slicing —but in total the combined application was beneficial for all but the *dac_6* instances mentioned above. Additionally to the replacement described

in Fig. 6.9 an overhead was caused when cutvertex reductions were applied to purely sequentially evolving slices. For instance the greatest costs—125 states and 140 state transitions—were caused when cutvertex reductions were applied on a slice of *dac_15*, which had 24 places and transitions with a state space of 24 states and state transitions. Cutvertex reductions reduced this slice to a net of 3 places and 5 transitions with a state space of 4 states and 4 state transitions. For this case the benefit with respect to the original net was still about 99.97%.

The coupled application led to further state space savings for 5 families. The net *q_1* is even only properly effective reducible by cutvertex reductions if it is sliced first.

Safety Slicing and Cutvertex Reductions Like the coupled application with $\mathrm{CTL}^{*}_{-\mathsf{X}}$ slicing, applying safety slicing before cutvertex reductions gained further state space savings.

Whereas the filtered state savings roughly add up, the unfiltered state saving of the coupled application even exceeds the sum of the single reductions. The reductions in state transitions are slightly less than the sum of savings by safety slicing and cutvertex reductions.

Of the 18 families with properly effective reduced nets, the state spaces of 8 families were further reduced by the combined application of safety slicing and cutvertex reductions. Only the *rw* family had limited effective reduced nets.

Again *q_1* was the only net, for which slicing was necessary to properly effectively apply cutvertex reductions.

Summary and Conclusions In both cases the combination was beneficial and the effects approximately add up. The combined application of $\mathrm{CTL}^{*}_{-\mathsf{X}}$ and cutvertex reductions stresses the risk of an overhead when cutvertex reductions are applied to small or sequentially evolving nets. The combination also has synergetic effects for instance on the net *q_1* illustrated in Table 6.12.

	states [%]	state trans. [%]	places [%]	trans. [%]
CTL^*_{-X} slicing and cutvertex reductions				
slice_4_16	24.3	24.96	16.57	5.16
cutvertex reduced slice_4_16	0.36	0.29	2.86	1.08
safety slicing and cutvertex reductions				
slice_7_11	9.61	9.9	9.82	2.06
cutvertex reduced slice_7_11	5.05	5.21	1.35	0.53

Table 6.12: Exemplary savings of slicing plus cutvertex reductions on *q_1*. The values on the *slice* line describe the savings gained by slicing. The values on the *cutvertex reduced* line give the savings gained by cutvertex reductions relative to the slice's state space.

6.1.5.2 Stubborn Sets

In this section we measure the results with respect to state spaces that are condensed by the stubborn set technique. As for the previous results we filter out the smallest reducts *with respect to the full state space*. The condensed state space was generated by PROD's implementation of the stubborn set method. PROD [83] is an analysis tool for Predicate/Transitions nets (PrT-nets). We encoded P/T-nets as special case of PrT-nets like it is described in [43].

Results with respect to Condensed State Spaces Since we now measure the results with respect to the condensed state space (=state space reduced by POR), we say that we have a *saving of* x of states (state transitions), if the reduct has factor x less states (state transitions) than the original net has in its condensed state space. Analogously, we use *overhead*, *benefit* and *cost* with respect to the condensed state space.

Of course, the *condensed* state space of a *reduced* net generated by the stubborn set technique is smaller than (or equals) the *full* state space of the reduced net and hence also smaller than the full state space of the original, but the *condensed* state space of a reduced net may not be smaller than the *condensed* state space of the original net if the stubborn set performs worse on the reduced net (cf. Fig. 6.10). Consequently the overhead may have

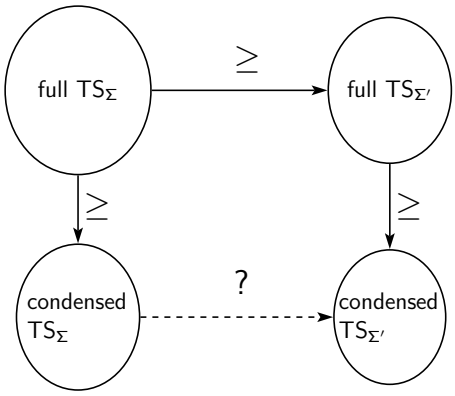


Figure 6.10: Condensed and reduced state spaces. TS_Σ refers to the state space of the original system and $TS_{\Sigma'}$ to the state space of a reduct.

values greater one, whereas the saving ranges between 0 and 1.

It may be counterintuitive that the *condensed* state space of the *reduced* net can be bigger than the *condensed* state space of the *unreduced* net even when the full state space of the *reduced* net is substantially smaller than the full state space of the *unreduced* net. But as PORs usually implement a heuristic to determine which transitions can be considered as independent, such a heuristic can work for one net better than for the other so that the stubborn set condensation on the original may be more effective than the condensations on the reduced net.

Using Condensed State Spaces as Reference State Space The order of transitions in the specification of the input net significantly influences the state space reductions gained by PROD's partial order implementation.

PORs heuristically choose the set of transitions that have to be executed at each state, if several candidate sets exist. A common heuristic is to use the smallest such set. In case there are sets of the same size this nondeterminism has to be resolved. We conjecture that this resolution is influenced by the order of transitions within the net description.

The fact that the result is influenced by the order of transitions hinders a direct comparison, since structural Petri net reductions change the Petri net graph and the same order is not reproducible since it is not the same

	only properly # families	prop. & limited # families	only limited # families	effective # families
safety slicing	6	3	2	11
CTL^*_{-X} slicing	5	4	2	11
cutvertex reductions	11	1	5	17
CFFD reductions	0	3	11	14
agglomerations	5	1	0	6

Table 6.13: Properly and limited properly reduced families.

net. To compensate this effect we measured the reductions on six different permutations including the original order and built the mean over all six results.

Another difficulty when using the condensed state space as reference state space is the selection of observable places for the generation of the condensed state space. In a CTL^*_{-X} slice the temporal properties may refer to all places, and in a cutvertex reduct the temporal properties may refer to all places of the kernel but the cutvertices. Partial order techniques that preserve temporal logics have to preserve the order for observables. So if we would declare all places as observable, the condensed state space would equal the full state space. But usually temporal logic formulas refer to only a few places of the net. Keeping this in mind, a good choice would be to generate condensed state spaces for all place subsets upto a certain size. But even generating the condensed state spaces for the original and its slices for every single place would be intractable—even the more so as we compute the state spaces several times permuting the transitions' order. So in a sense a fair comparison is not possible.

We hence chose to condense the state space by deadlock preserving stubborn sets. Usually a state space condensed to preserve deadlocks is expected to be smaller than (or equal to) a state space condensed to preserve safety properties or e.g. LTL_{-X} properties [102]. We hence belief that the results presented in the following allow to study the general effects of combining Petri net graph reductions with stubborn sets.

Let us study the results summarised in Tables 6.13 and 6.14. Table 6.13 lists the numbers of families that have properly effective reducts only or

	mean state space (states,state trans.)	relative mean state space (states,state trans.)	# limited effective # nets	# properly effective # nets
safety slicing	7843.58, 25296.81	0.637, 0.735	14	714
CTL^*_{-X} slicing	12226.40, 34026.95	0.993, 0.988	41	631
cutvertex reds.	12526.33, 34709.27	1.017, 1.008	34	95
CFFD reds.	12328.4 , 34516.28	1.001, 1.002	96	18
agglomerations	12297.62, 34422.27	0.999, 1	2	118

Table 6.14: Mean values for a comparative evaluation.

properly and limited effective reducts, or limited effective reducts only. The sum of the three values gives the number of effectively reduced nets. Again cutvertex reductions affect the most net families, followed by CFFD reductions, followed by the two slicing methods. Agglomerations affect by far the least nets and also causes fewest limited effective reducts. CFFD reductions causes the most limited effective reducts. Cutvertex reductions and CTL^*_{-X} slicing cause limited effective reducts for six net families, whereas safety slicing causes limited effective reducts for five families. We think the main reason that agglomerations cause less families to have limited effective reducts, is that it affects less families and the other methods affect a wide range of nets only marginally, which just tips the stubborn set method off to take different representatives.

According to Table 6.14 applying CTL^*_{-X} slicing, agglomerations, cutvertex or CFFD reductions generates state spaces of similar sizes. In all four cases the mean state space of the reducts is about as big as the mean condensed state space. The only method that significantly decreases the mean state space is safety slicing, which yields a benefit of 36.3% of the states and 26.5% of the state transitions with respect to the condensed state space, which is about twice as much safety slicing could save on the full state space (cf. Table 6.4). This is mainly due to three families that are more effectively condensed by stubborn set reductions when sliced.

For the mean reduction effect it makes nearly no difference whether CTL^*_{-X} slicing, agglomerations, cutvertex or CFFD reductions are applied or not. Table 6.13 shows that there are nevertheless many instances where the ap-

plication of the reductions increases the state space savings and that the majority of reducts improves the state space savings. The number of properly effective reducts exceeds the number of limited effective reducts by many times—except for CFFD reductions.

According to Fig. 6.11 some nets were reduced so much by slicing and agglomerations that they now appear in a higher savings cluster while all other nets remain in the same savings cluster. Comparing the savings by only the stubborn set method (a) to the results of cutvertex reductions (e) in Fig. 6.11, we notice that when cutvertex reductions are applied, nets of the *sentest* family appear in clusters of less savings whereas only *elevator_1* appears in a cluster of greater savings. This seems to indicate that cutvertex reductions actually work against POR. So we inspected the reducts with the greatest overheads.

Reducts with the greatest overhead are of *bds_1, ftp_1*, *speed_1* and *sentest*. If we would ignore the reducts of the first two nets, applying cutvertex reductions would lead to an increase in the mean savings. To evaluate whether cutvertex reductions decrease the savings gained by the stubborn set reduction when verifying temporal logic formulas, we picked for each of these nets sample reducts with greatest overheads. Each reduct and its respective original were checked for a temporal property[1] referring to one place only but causing the model checker to examine the full state space and this was done for each of its (non-contact) places. The states and state transitions inspected were measured. Table 6.15 presents the results. It turns out that when we consider $\mathrm{LTL}_{-\mathsf{X}}$ preserving stubborn set reductions, the combination of cutvertex reductions and stubborn sets works quite well.

Summary and Conclusions In this section we examined the effect of combining POR with slicing, agglomerations, CFFD and cutvertex reductions. To examine the general effects of such a combination we used deadlock preserving stubborn sets of PROD.

[1] We checked $\mathsf{A}(\mathsf{F}(\mathsf{G}(p,0) \vee \mathsf{G}(\mathsf{F}(p,1))))$.

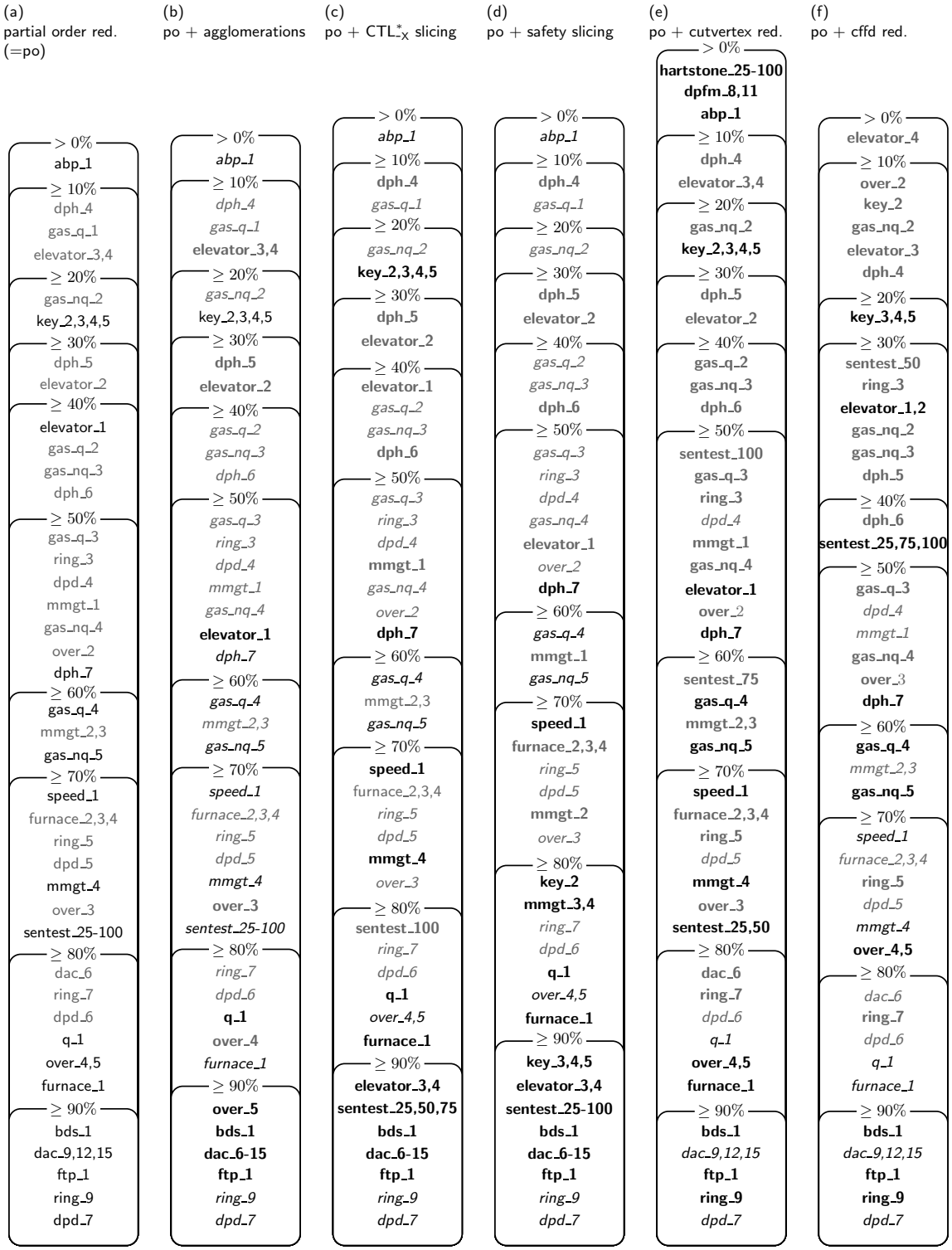


Figure 6.11: Properly effective reduced nets for condensed state spaces. (a) lists the savings gained by PROD's stubborn set reduction. (b) to (e) show the reductions on the condensed state space when the respective reduction technique is applied. The earliest occurrence of a family is marked in black. Nets reduced by the respective reduction are set in bold face. Nets left unchanged by the respective reductions are set in italics.

	inspected states, state trans. on the original	cutvertex savings (states, state trans.)	$\frac{\#\text{saving instances}}{\text{total}\#\text{observables}}$
bds_1	9610.39, 33957	0.378, 0.512	32/44
ftp_1	47481.41, 210594.78	0.204, 0.276	152/171
sentest_100	385.49, 632.71	-0.097, -0.113	10/326
speed_1	13131.93, 34949.12	0.008, 0.014	18/32

Table 6.15: Verification on reduced nets with condensed state spaces. The last column gives the number of places p for which cutvertex reductions increased the savings when the temporal property referred to p.

Safety slicing increased the state space savings considerably. $\text{CTL}^*_{-\mathsf{X}}$ slicing, agglomeration and cutvertex reductions had about no effect on the mean state space but for many concrete instances further state space savings were gained. For nets with the greatest overhead we examined the savings when model checking a temporal property: Even for those nets the reductions were beneficial in average for three out of four examined nets.

The benchmark set we used has been compiled trying to cover a wide range of systems. The results of this section show that some nets profit from one technique but not the other. This makes it difficult to predict the effectiveness of our techniques on other (sets of) examples. To evaluate the relevance of our techniques more real world case studies would have to be undertaken. A class of system models that seem very apt to our reductions are workflow nets.

6.2 Workflow Management

In this section we present a workflow net case study representing a class of system models that lately received a lot of attention in the Petri net research community because of their industrial relevance for business process modelling.

Business processes are marked-centred descriptions of an organisation's activities for a certain service or product. A *workflow* models a business process on the conceptual level either for understanding, evaluating and re-

designing the business process, or for describing process requirements [44].

Business process management involves the design and specification of business processes *(business process modelling)*, analysis and optimisation *(business process reengineering)*, definition *(workflow modelling)*, execution and administration *(workflow enactment)* and monitoring and evaluation. *Workflow management* means the IT-based support of business process management [69]. So a *workflow management system* is a "system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications" [113].

Typically workflows are *case*-based [1] and every case has a beginning and an end. A *workflow process* is designed to handle similar cases by defining a route of tasks to be executed for a specific case. The routing is also called *workflow process definition*. A task that needs to be executed for a specific case is called a *work item*. Most work items are executed by a resource, which may be a person or machine like a fax.

For example, the processing of insurance claims can be described by a workflow. A case of such a workflow process is a specific insurance claim, e.g. the insurance claim of Mr. J. Smith. An example of a work item is the execution of the task *send notification* for the case of the *insurance claim of Mr. J. Smith*.

Workflow specifications describe various perspectives: The *control flow perspective* (or *process perspective*) specifies tasks and their execution ordering. The *data perspective* deals with business and processing data. The *resource perspective* is concerned with roles of humans or devices executing tasks. The *operational perspective* describes the elementary actions that are executed by tasks and that map into underlying applications [47].

Because of their graphically intuitive notation and abundance of analysis techniques, Petri nets have been advocated by many as formalism for modelling business processes, for instance in [2, 4, 40]. Using the Petri net formalism, tasks are modelled by transitions, conditions, on which tasks depend, are modelled by places, and cases are modelled by tokens. Often, high-level

Petri nets (with coloured tokens, time and hierarchy) are used to model a workflow and classical P/T nets are studied when analysing the control flow.

One line of research is concerned with the analysis of workflow processes. As fixing errors later on is costly, a focus is to define correctness criteria that allow to detect errors as early as possible. Van der Aalst suggested *WorkFlow-nets (WF-nets)* [2] to define workflow processes. A WF-net has a distinct input place, p_{in}, and an output place, p_{out}, and has no dangling tasks, i.e. every transition is on a path from the input place p_{in} to the output place p_{out}. A prominent correctness criteria for WF-nets is *soundness*. Intuitively, a workflow process is sound iff every task is executable and the workflow process can properly terminate, that is it can always terminate and that after termination there are no objects left behind. More formally, a WF-net is sound if (1) if it is possible to fire any transition given a token on p_{in}, $(\forall t \in T : \exists M \in [M_{\mathsf{init}}\rangle : M[t\rangle)$, (2) the marking with only tokens on p_{out} is the only marking placing tokens on p_{out} $(\forall M \in [M_{\mathsf{init}}\rangle : M(p_{\text{out}}) \geq 1 \Rightarrow \forall p \in P \setminus \{p_{\mathsf{out}}\} : M(p) = 0)$, and (3) for every marking reachable from M_{init} there is a firing sequence placing a token on p_{out} $(\forall M \in [M_{\mathsf{init}}\rangle : \exists \sigma \in T^* : \exists M' \in [M_{\mathsf{init}}\rangle : M[\sigma\rangle M' \wedge M'(p_{\text{out}}) = 1)$ [2], where M_{init} marks p_{in} with one token and marks no other place. Van der Aalst showed that soundness for acyclic, free-choice WF-nets can be proven in polynomial time and argued that many workflow processes can be modelled as free-choice and acyclic nets.

Soundness is a minimal property any workflow process definition should satisfy. More intricate errors within the control flow can be found by model checking, as demonstrated for instance in [70, 63]. Our techniques are especially apt to reduce workflow nets. By definition workflow nets are not strongly connected and since they model work *flows* slicing can effectively reduce such nets. Mendling argued in [73] that the ratio articulation points per nodes in a process definition can be seen as a measure of separability. A high ratio implies a decrease in the error probability of the overall model. Hence reasonable workflow nets should have articulation points and cutvertex reductions promise further reductions. When we restrict cutvertex reductions to micro reductions and structural optimisations only, there is no risk of applying them to even small and sequentially evolving nets. Although

for instance *interorganisational workflows*, which model business processes where several organisations participate, quite naturally exhibit concurrency.

To illustrate the potential of both techniques for workflow nets, we analyse the Petri net of Fig. 6.12, which models the workflow of a business process for dealing with insurance claims like in [3]. An incoming claim is recorded first. A claim may be accepted or rejected, depending on the insurance cover. For a rejected claim, a rejection letter is written. If the claim is accepted, emergency measures, if necessary, are provided. After an assessment -possibly done by an expert- a settlement is offered to the customer, who may either accept or reject. A rejected offer may be followed by legal proceedings or a revision. If a settlement is agreed upon, money is paid [89].

We want to verify that every accepted claim is settled, i.e. $\varphi = \mathsf{AG}((ac, 1) \Rightarrow \mathsf{F}\,(cs, 1))$. The *slice* of $\Sigma_{\mathsf{ins.}}$ for $\{ac, cs\}$ is the subnet within the dashed borders. If we also apply cutvertex reductions, the slice is further reduced as illustrated in 6.12. So slicing can truncate chain/flow ends and cutvertex reductions can additionally summarise initial flows.

When model checking, we learn that φ does not hold due to the *offer/revise* loop. The counter example on the original is found inspecting 23 states and 24 state transitions; the counter example on the slice is found even faster by inspecting 12 states and state transitions. So we assume strong fairness and modify φ to express that *revise* is executed only finitely often, that is $\varphi' = \mathsf{A}(\mathsf{F}\,\mathsf{G}\,(as, 0)) \Rightarrow (\mathsf{G}((ac, 1) \Rightarrow \mathsf{F}\,(cs, 1)))$. To model check φ' on the original 218 states and 564 state transitions were inspected. The combination of slicing and cutvertex reductions reduced this to 68 states and 112 state transitions. The combination of slicing and cutvertex reductions also saved the most compared with agglomerations and when partial order reductions are applied.

Summary We gave a brief introduction to business process modelling and introduced workflow nets as a class of systems that have industrial relevance and are seemingly very apt to slicing and cutvertex reductions. We presented a small case study to illustrate the potential of our techniques for this class

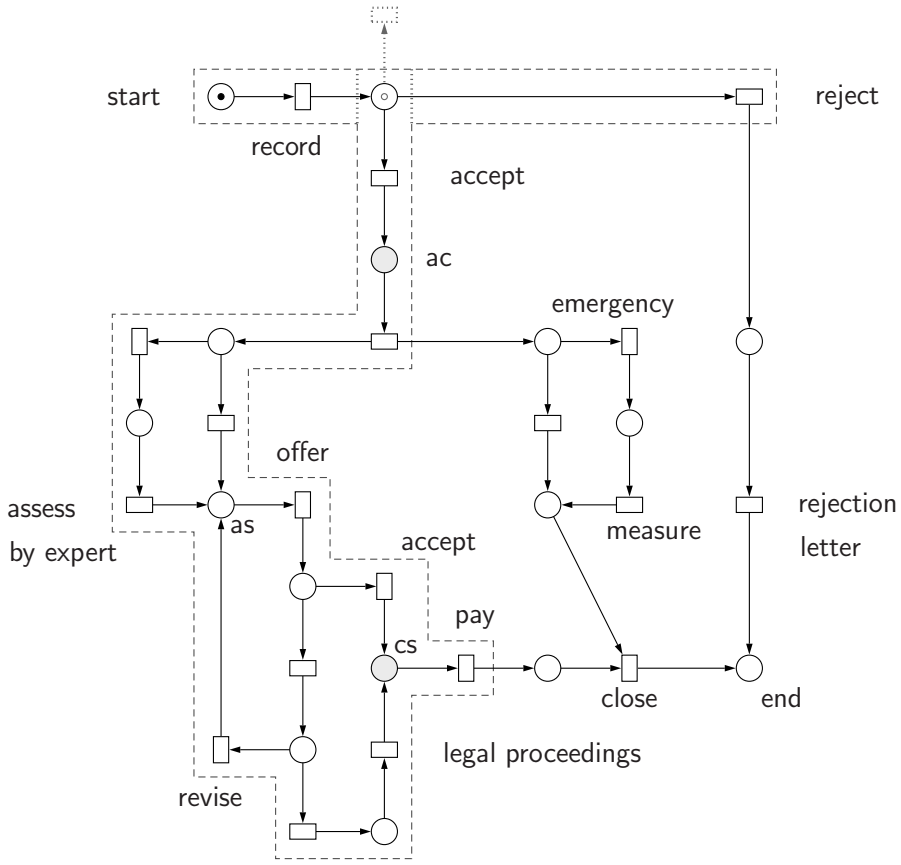


Figure 6.12: A WF-net, $\Sigma_{\text{ins.}}$ modelling an insurance claim process. The dashed border marks the slice for $\{ac, cs\}$. When cutvertex reductions are applied the place *start*, transitions *record* and *reject* are summarised as producer-consumer, that is they are eliminated and replaced by the hollow token and dotted transition).

of systems.

Chapter 7

Conclusions

Contents

We first give a summary of this work and refer to accompanying publications. Then we briefly recapitulate ideas for future work as given in detail in Sect. 4.5 and 5.9.

7.1 Summary

In this work we presented two Petri net reduction approaches as further means to combat the state space explosion problem: *slicing* and *cutvertex reductions*. For both approaches we examined the preservation of several relevant classes of temporal properties. For preservation of liveness properties, we introduced and examined a weak fairness notion, referred to as relative fairness. We demonstrated the effectiveness of our approaches on a workflow case study and in comparison to three prominent approaches for mitigating state space explosion on a benchmark set.

Relative Fairness We showed that although relative fairness is indeed weaker than fairness notions like weak and strong fairness, it suffices for

liveness preservation by $\mathrm{CTL}^*_{-\mathsf{X}}$ slicing as well as by cutvertex reductions.

Slicing We developed two flavours of slicing: safety slicing and $\mathrm{CTL}^*_{-\mathsf{X}}$ slicing. We proved that whereas the latter preserves all $\mathrm{CTL}^*_{-\mathsf{X}}$ properties under relative fairness, safety slicing preserves only stutter-invariant safety properties but reduces a net more aggressively. $\mathrm{CTL}^*_{-\mathsf{X}}$ slicing has been published in [90, 91] and safety slicing in [87].

Cutvertex Reductions We introduced $\mathrm{LTL}_{-\mathsf{X}}$ preserving cutvertex reductions as a decompositional approach to Petri net reductions where a monolithic net is decomposed into a kernel containing $scope(\varphi)$ and environments that are to be replaced by small summary nets. We identified six distinct behavioural classes of environment nets and determined their replacement summary. We gave a decomposition algorithm that runs in linear time when 1-safeness of contact places is known a priori. As structural optimisations we developed the so-called pre-/postset optimisations, which accelerate the identification of the behavioural class an environment belongs to. As structural reductions we implemented micro reductions which reduce the smallest environment nets directly. Cutvertex reductions are presented in [89]. Best and the author herself illustrate the effectiveness of $\mathrm{CTL}^*_{-\mathsf{X}}$ slicing and cutvertex reductions on a business process model in [11].

Evaluation The evaluation on the benchmark set showed that our approaches compare well with pre- and postagglomerations and CFFD reductions, and that our approaches can lead to further state space reductions even for state spaces condensed by partial order reductions.

7.2 Future Work

We have shown that structural Petri net reductions can further accelerate model checking of temporal logic properties. Especially safety slicing showed good results in combination with stubborn set reductions. So it seems worthwhile to develop refined slicing algorithms for special classes of properties

that allow for more aggressive slicing. As we argued in Sect. 4.5, antecedent slicing seems a good starting point. Cutvertex reductions are limited to environment nets with a single contact place only. Further research could lift the approach to environment nets with two contact places. We conjecture that the classification will be more complex, so that a focus should be on the development of structural reductions and optimisations for the classification of environments.

Index

List of Figures

List of Tables

Bibliography

[1] van der Aalst, W.M.P.; Loosely Coupled Interorganizational Workflows: Modelling and Analyzing Worklfows Crossing Organizational Boundaries In: Information and Management, 37 (2), 2000, Elsevier, 67 – 75.

[2] van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. In: The Journal of Circuits, Systems and Computers, 8 (1), 1998, World Scientific Publishing, 21–66.

[3] van der Aalst, W.M.P.; van Hee, K.: Workflow Management - Models, Methods, and Systems. The MIT Press, 2002.

[4] Adam, N. R.; Atluri, V.; Huang, W.-K.: Modeling and Analysis of Workflows Using Petri Nets. In: Journal of Intelligent Information Systems, 10 (2), Special issue on workflow management systems, Kluwer Academic Publishers 1998, 131–158.

[5] Aziz, A.; Singhal, V.; Balarin, F.; Brayton, R.; Sangiovanni-Vincentelli, A.-L.: Equivalences for fair kripke structures. In: Automata, Languages and Programming, Proceedings of 21st ICALP, Jerusalem, 1994, 364–375.

[6] Baier, C.; Katoen, J.-P.: Principles of Model Checking. The MIT Press, 2008, 112–120.

[7] Bandera. `http://bandera.projects.cis.ksu.edu`

[8] Berezin, S.; Campos, S.; Clarke, E.M.: Compositional reasoning in model checking. In: Revised Lectures of the International Symposium COMPOS'97, Lecture Notes in Computer Science 1536, 1998, 11–22.

[9] Berthelot, G.: Checking Properties of Nets Using Transformation. In: Advances in Petri Nets 1985, Lecture Notes in Computer Science 222, 1985, Springer Verlag, 19–40.

[10] Berthelot, G.: Verification de Réseaux der Petri. Université Paris VI, 1983.

[11] Best, E.; Rakow, A.: A Slicing Technique for Business Processes, In: Proc. of the 2nd International United Systems Conference on Information Systems and e-Business Technology, Klagenfurt, Austria, 2008, LNBIP 5, 2008, 45–51.

[12] Billington, J.; Gallasch, G.E.; Kristensen, L.M.; Mailund, T.: Exploiting equivalence reduction and the sweep-line method for detecting terminal states. In: IEEE Transactions on Systems, Man and Cybernectics, Part A: Systems and Humans 34 (1), 2004, IEEE, 23–37.

[13] Binkley, D. W.; Gallagher K. B.: Program Slicing. In Zelkowitz. M. V. ed.: Advances in Computers, 43, 1996, Academic Press San Diego, 2–52.

[14] Brückner, I.: Slicing Integrated Formal Specifications for Verification. PhD thesis. University of Paderborn, March 2008.

[15] Brückner, I.: Slicing CSP-OZ specifications. In: Nordic Workshop on Programming Theory (2004).

[16] Chang, C.K., Wang, H.: A slicing algorithm of concurrency modeling based on Petri nets. In Hwang, K., Jacobs, S.M., Swartzlander, E.E., eds.: Proc. of the 1986 Int. Conf. on Parallel Processing, Washington, IEEE Computer Society Press, 1987, 789–792.

[17] Chang, J., Richardson, D.J.: Static and dynamic specification slicing. In: Proceedings of the Fourth Irvine Software Symposium, 1994, .

[18] Cheng, A.; Esparza, J.; Palsberg, J.: Complexity Results for 1-safe nets. In: Foundations of Software Technology and Theoretical Computer Science 1993, Lecture Notes in Computer Science 761, 1993, Springer Verlag, 326–337.

[19] Cheng, Y.; Tsai, W.T.: An Algebraic Approach to Petri Net Reduction and Its Application to Protocol Analysis. Technical Report, University of Minnesota, 1990.

[20] Ciamatti, A.; Clarke, E.; Giunchiglia, E.; Giunchiglia, F.; Pistore, P.; Roveri, M.; Sebastiani, R.; Tacchella, A.: NuSMV2: An OpenSource Tool for Symbolic Model Checking. In: Proc. of Computer Aided Verification 2002 (CAV 02), Lecture Notes in Computer Science 2404, 2002, Springer Verlag, 359–364.

[21] Clarke, E. M.; Fujita, M.; Rajan, S. P:; Reps, T.; Shankar S.; Teitelbaum, T.: Program Slicing for VHDL. In: Software Tools for Technology Transfer (STTT) 2 (4), 2000, Springer Verlag, 343–349.

[22] Clarke, E. M.; Grumberg, O.; Peled, D. A.: Model Checking. The MIT Press, 1999, 171–176.

[23] Clarke, E. M.; Filkorn, T.; Jha, S.: Exploiting Symmetry in Temporal Logic Model Checking. In: Computer Aided Verification, Lecture Notes in Computer Science 697, Springer Verlag, 1993, 450–462.

[24] Clarke, E. M.; Emerson, E. A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Logic of Programs 1981, Lecture Notes in Computer Science 131, 1981, Springer Verlag, 428–437.

[25] Colom, J. M.; Teruel, E.; Silva, M.; Haddad, S.: Structural Methods. In Girault, C.; Valk, R. (eds.): Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications, Springer Verlag, 2003, 277–316.

[26] Corbett, J.C.: Evaluating Deadlock Detection Methods for Concurrent Software. In: IEEE Transactions on Software Engineering 22 (3), 1996, 161–180.

[27] Corbett, J.C.: An empirical evaluation of three methods for deadlock analysis of Ada tasking programs. In: Proceedings of the 1994 International Symposium on Software Testing and Analysis (ISSTA), 1994, 110–116.

[28] Danicic, S.; De Lucia, A; Harman, M.: Building Executable Union Slices using Conditioned Slicing. In: 12th IEEE International Workshop on Program Comprehension (IWPC'04), 2004, IEEE Computer Society, 89–99.

[29] Desel, J.: Basic Linear Algebraic Techniques for Place/Transition Nets. In Lectures on Petri Nets I: Basic Models, Lecture Notes in Computer Science 1492, 1998, Springer Verlag, 257–308.

[30] Desel, J.; Esparza, J.: Free choice Petri Nets. Cambridge University Press, New York, 1995.

[31] Desel, J.: Reduction and Design of Well-Behaved Concurrent Systems. In: Proc. of CONCUR'90, Lecture Notes in Computer Science 458, 1990, Springer Verlang, 166–181.

[32] Diestel R.: Graph Theory. Graduate Texts in Mathematics, Volume 173, Electronic Edition 2005, Springer Verlag, 2005.

[33] Emerson, E.; Sistla, A.: Symmetry and Model Checking. In: Computer Aided Verification, Lecture Notes in Computer Science 697, Springer Verlag, 1993, 450–462.

[34] Esparza, J.; Heljanko, K.: Unfoldings – A Partial-Order Approach to Model Checking. Monographs in Theoretical Computer Science, Springer Verlag, 2008, 172 pp.

[35] Esparza, J.; Heljanko, K.: Implementing LTL Model Checking with Net Unfoldings. Research Report A68, Laboratory for Theoretical Computer Science, Helsinki University of Technology, Espoo, Finland, March 2001, 29p.

[36] Esparza, J.; Nielsen, M.: Decidability Issues for Petri nets: A Survey. In: Journal of Information Processing and Cybernetics 30 **3**, 1994, 143–160.

[37] Esparza, J.; Römer, S.: An Unfolding Algorithm for Synchronous Products of Transition Systems. In: Proc. of CONCUR'99, Lecture Notes in Computer Science 1664, 1999, Springer Verlang, 2–20.

[38] Esparza, J.; Schröter, C.: Net Reductions for LTL Model-Checking. In: Proceedings of the 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods 2001, Lecture Notes in Computer Science 2144, 2001, Springer Verlag, 310–324.

[39] Esparza, J.; Silva, M.: On the analysis and sythesis of free choice systems. In: Advances in Petri Nets 1990, Lecture Notes in Computer Science 483, 1991, 243–286.

[40] Flores-Badillo, M.; López-Mellado, E.; Padilla-Duarte, M.; Modeling and Simulation of Workflow Processes Using Multi-level Petri Nets. In: Proceedings of the 4th International Workshop on Enterprise & Organizational Modeling and Simulation held in conjunction with the CAiSE'08 Conference, EOMAS'08, 2008, CEUR Workshop Proceedings, 338, 50–63.

[41] Furia, C. A.: A Compositional Word: A Survey of recent works on compositionality in formal methods. Technical Report 2005.22, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 2005.

[42] Gannod, G. C.; Gupta, S.: An Automated Tool for Analyzing Petri Nets Using Spin. In: Proceedings of the 16th International Conference on Automated Software Engineering, 2001, IEEE, 404–407.

[43] Genrich, H. J.: Predicate/Transition Nets. In Adavances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Lecture Notes in Computer Science 254, 1987, Springer Verlag, 207–247.

[44] Georgakopoulos, D.; Hornick, M.; Seth, A.: An Overview of Workflow Management: From Process Modelling to Worklfow Automation Infrastructure. In: Distributed and Parallel Databases, 3, Kluwer Academic Publishers, 1995, 119-153.

[45] Godefroid, P.: Partial-Order Methods for the Verification of Concurrent Systems Springer Verlag, 1996.

[46] Godefroid, P.: Using Partial Orders to Imporve Automatic Verification Methods. In: Computer Aided Verification 1990, Lecture Notes in Computer Science 531, 1990, Spinger Verlag, 321–340.

[47] ter Hofstede, A. H. M.; van der Aalst, W. M.: YAWL: yet another workflow language. In: Information Systems, 30 (4), 2005, Elsevier, 245–275.

[48] Holzmann, G. J.: The Spin Model Checker. Primer and Reference Manual. Addison Wesley, 2004, 231–235.

[49] Holzmann, G. J.; Peled, A.; Yannakakis, M.: On nested depth first search. In: The Spin Verification System, American Mathematical Society, 1996, 23–32.

[50] Hopcroft, J.E.; Trajan, R.E.: Dividing a Graph into Triconnected Components In: SIAM Journal on Computing 2 (3), 1975, Society for Industrial and Applied Mathematics, 135–158.

[51] Haddad, S.; Pradat-Peyre, J.-F.: New Efficient Petri Nets Reductions for Parallel Programs Verification. In: Parallel Processing Letters 16 (1), 2006, World Scientific Publishing Company, 101–116.

[52] Hatcliff, J.; Dwyer, M.; Zheng, H.: A Formal Study for Multi-threaded Programs with JVM Concurrency Primitives. In: Higher-Order and Symbolic Computation 13 (4), 2000, Springer-Verlag, 315–353.

[53] Hatcliff, J.; Corbett, J.; Dwyer, M.; Solowski, S.; Zheng, H.: Slicing software for model construction. In: Proceedings of the 6th. International Static Analysis Symposium (SAS'99), Lecture Notes in Computer Science (1694), 1999, Springer Verlag, 315–353.

[54] Heimdahl, M.P.E., Whalen, M.W.: Reduction and slicing of hierarchical state machines. In Jazayeri, M., Schauer, H., eds.: Proceedings of the Sixth European Software Engineering Conference (ESEC/FSE 97), 1997, Springer-Verlag, 450–467.

[55] Heljanko, K.: Minizing Finite Complete Prefixes. In Burkhard, H.-D., Czaja, H.-D., Nguyen, H.-S., Starke, P., eds.: Proc. of the Workshop Concurrency, Specification & Programming 1999, 1999, Warsaw University, 83–95.

[56] Heljanko, K.: Deadlock and reachability checking with finite complete prefixes. Research Report A56, Helsinki University of Technology, Department of Computer Science and Engineering, Laboratory for Theoretical Computer Science, Espoo, Finland, December 1999, 70p.

[57] Heljanko, K.; Khomenko, V.; Koutny, M.: Parallelisation of the Petri Net Unfolding Algorithm. In: Proc of TACAS 2002, Lecture Notes in Computer Science 2280, 2002, Springer Verlang, 371–385.

[58] Juan, E.Y.T.; Tsai, J.J.P.: Compositional Verification of Concurrent and Real-Time Systems. Kluwer Academic Publishers, 2002, 83–117.

[59] Juan, E.Y.T.; Tsai, J.J.P.: Compositional Verification of Concurrent Systems Using Petri-Net-Based Condensation Rules. In: ACM Transactions on Programming Languages and Systems 20 (5), 1998, 917–979.

[60] Khomenko, V.; Koutny, M.: Verification of bounded Petri nets using integer programming. In: Formal Methods in System Design 30 (2), 2007, Springer Netherlands, 143–176.

[61] Khomenko, V.; Koutny, M.: Towards An Efficient Algorithm for Unfolding Petri Nets. In : Proc. of CONCUR 2001, Lecture Notes in Computer Science 2154, 2001, Springer Verlag, 266–280.

[62] Klai, K.; Petrucci, L.; Reniers, M.: An Incremental and Modular Technique for Checking LTL\X Properties of Petri nets. In: Formal Techniques for Networked and Distributed Systems 2007, Lecture Notes in Computer Science 4574, 2007, Springer Verlag, 280–295.

[63] Köhler, J.; Tirenni, G.; Kumaran, S.; From Business Process Model to Consistent Implementation: A Case for Formal Verification Methods. In: Proc. of Sixth International Enterprise Distibuted Object Computing Conference (EDOC 2002), IEEE Computer Society Press, 2002, 96 – 106.

[64] Biconnected Components. `http://sparcs.kaist.ac.kr/~lacrimosa/algorithm/2003/CS300-09.ppt`

[65] Lamport, L.: What Good is Temporal Logic? In: Information Processing 1983: Proceedings of the IFIO 9th World Computer Congress, 1983, 657-668.

[66] Lamport, L.: Proving the Correctness of Multiprocess Programs In: IEEE Transactions on Software Engineering SE-3, 2, 1977, IEEE, 125–143.

[67] Lee, W.J.; Cha, S.D.; Kwon, Y.R.; Kim, H.N.: A Slicing-based Approach to Enhance Petri Net Reachability Analysis. In: Journal of Research and Practice in Information Technology 3, 2000, 131–143.

[68] Llorens, M.; Oliver, J.; Silva, J.; Tamarit, S.; Vidal, G.: Dynamic Slicing Techniques for Petri Nets. Second Workshop on Reachability Problems (RP'08), Liverpool (UK). In: Proceedings of the Second Workshop on Reachability Problems in Computational Models (RP 2008), Electronic Notes in Theoretical Computer Science 223, December 2008, 153–165.

[69] Martens, A.: Verteilte Geschäftsprozess: Modellierung und Verifikation mit Hilfe von Web Services. Institut für Informatik, Humboldt-Universität zu Berlin, Berlin, Germany, 2003.

[70] Matousek, P.: Verification of Business Process Models. PhD thesis, 2003, 100 pp. .

[71] McMillan, K. L.: Symbolic Model Checking: An approach to the State Explosion Problem. Kluwer academic Publishers, 1993.

[72] Melzer, S.; Römer, S.: Deadlock Checking Using Net Unfoldings In: Proc. of CAV'97, Lecture Notes in Computer Science 1254, 1997, Springer Verlag, 164–174.

[73] Mendling, J.: Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness. Lecture Notes in Business Information Processing 6, 2009, Springer Verlag, 193 pp.

[74] Millett, I. L.; Teitelbaum, T.: Issues in Slicing Promela and its Applications to Model Checking, Protocol Understanding and Simulation. In: Software Tools for Technology Transfer (STTT) 4 (2), 2002, Springer Verlag, 125–137.

[75] Murata, T.: Petri nets: Properties, Analysis and Applications. In: Proceedings, of the IEEE 77 (4), 1989, IEEE, 541–580

[76] Nieminen, J.; Kilamo, T.; Kivelä, T.; Geldenhuys, J.; Erkkilä, T.: Tampere Verification Tool - TVT tutorial. Tampere Univerity of Technology, Institute of Software Systems, 2003, 43 pp.

[77] Namjoshi, K.S.: A Simple Characterization of Stuttering Bisimulation. In: Foundations of Software Technology and Theoretical Computer Science, 1997, Springer-Verlag, 284–296.

[78] NuSMV. `http://nusmv.irst.itc.it`

[79] Park, S.; Kown, G.: Avoidance of State Explosion Using Dependency Analysis in Model Checking Control Flow Model. In: Proc. of Computational Science and its Applications (ICCSA06), Lecture Notes in Computer Science 3984, 2006, Springer Verlag, 905–911.

[80] Peled, D.; Wilke, T.: Stutter-Invariant Temporal Properties are Expressible Without the Next-time Operator. In: Information Processing Letters 63, 5, 1997, Elsevier, 243–246.

[81] Peng, H.; Tahar, S.: A Survey on Compositional Verification. Technical Report, Concordia University, Department of Electrical and Computer Engineering, December 1998.

[82] Pnueli, A.: A temporal logic of concurrent programs. In: Proc. of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977), IEEE Computer Society Press, 1977, 46–57.

[83] PROD. `http://www.tcs.hut.fi/Software/prod/`

[84] PEP. `http://peptool.sourceforge.net`

[85] Poitrenaud, D.; Pradat-Peyre, J.-F.: Pre- and Post-agglomerations for LTL Model Checking. In: International Conference on Application and Theory of Petri Nets 2000, Lecture Notes in Computer Science 1825, 2000, Springer Verlag, 387–408.

[86] Queille, J. P.; Sifakis, J.: Specification and Verification of Concurrent Systems in CEGAR. In: Proc. of the 5th international Symposium on Programming, Lecture Notes in Computer Science 137, 1982, Springer Verlag, 337–351.

[87] Rakow, A.: Safety Slicing Petri Nets. In: Proceedings of the International Conference on Application and Theory of Petri Nets 2012, Lecture Notes in Computer Science, 2012, Springer Verlag, to be published.

[88] Rakow, A.: Decompositional Petri Net Reductions. In: 7th Confenerence on Integrated Formal Methods (IFM 2009), Lecture Notes in Computer Science 5423, 2009, Springer Verlag, 352–366.

[89] Rakow, A.: Slicing Petri nets with an Application to Workflow Verification. In: Theory and Practice of Computer Science 2008, Lecture Notes in Computer Science 4910, 2008, Springer Verlag, 436–447.

[90] Rakow, A.: Slicing Petri Nets. In: Proceedings of the Workshop on FABPWS'07 (2007), Satellite Event of the ICATPN 2007, Siedlce, 56–70.

[91] Rakow, A.: Slicing petri nets. Technical Report, Carl von Ossietzky Universität Oldenburg, 20 pages, `parsys.informatik.uni-oldenburg.de/pubs/SlPN_tr.pdf`, 2007.

[92] de Roever, W.: The Need for Compositional Proof Systems: A Survey. In: Revised Lectures of the International Symposium COMPOS'97, Lecture Notes in Computer Science 1536, 1998, 11–22.

[93] Wist, D.; Wollowski, R.; Schaefer, M.; Vogler, W.: Avoiding Irreducible CSC Conflicts by Internal Communication. In: Application of Concurrency to System Design, Fundamenta Informaticae 95 (1), 2009, 1–29.

[94] Schnoebelen, Ph.: The Complexity of Temporal Logic Model Checking. In: Selected Papers from the 4th Workshop on Advances in Modal Logics (AiML'02), 2003, King's College Publication, 393–436.

[95] Shatz, S.M.; Tu, S.; Murata, T.; Duri, S.: An Application Of Petri Net Reduction For Ada Tasking Deadlock Analysis. In: IEEE Transactions on Parallel and Distributed Systems 7 (12), 1996, IEEE Press, 1307–1322.

[96] Silva, M.: Las redes de Petri: en la Automática y la Informaática. Editorial AC, Madrid, 1985.

[97] Sloane, A.M., Holdsworth, J.: Beyond traditional program slicing. In: International Symposium on Software Testing and Analysis, San Diego, CA, ACM Press, 1996, 180–186.

[98] SPIN. `http://spinroot.com/spin/whatisspin.html`

[99] Tip, F.: A survey of program slicing techniques. In: Journal of programming languages 3, 1995, 121–189.

[100] TVT. `http://www.cs.tut.fi/ohj/VARG/TVT/`

[101] Valmari, A.: Composition and Abstraction. In: Modeling and Verification of Parallel Processes (MOVEP 2000), Lecture Notes in Computer Science 2067, 2001, Springer Verlag, 58–98.

[102] Valmari, A.: The State Explosion Problem. In: Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, Lecture Notes in Computer Science 1491, 1996, Springer Verlag, 429–528.

[103] Valmari, A.: Compositional Analysis with Place-Bordered Subnets. In: International Conference on Application and Theory of Petri Nets 1994, Lecture Notes in Computer Science 815, 1994, Springer Verlag, 531–547.

[104] Valmari, A.: On-the-Fly Verification with Stubborn Sets. In: Computer Aided Verification 1993, Lecture Notes in Computer Science 697, 1993, Springer Verlag, 397–408.

[105] Kaivola, R.; Valmari, A.: The Weakest Compositional Semantic Equivalence Preserving Nexttime-less Linear Temporal Logic. In: CONCUR'92, Lecture Notes in Computer Science 630, 1992, Springer Verlag, 207–221.

[106] Valmari, A.: A Stubborn Attack on State Explosion. In: Computer Aided Verification 1990, Lecture Notes in Computer Science 531, 1990, Springer verlag, 156–165.

[107] Vanhatalo, J.; Völzer, H.; Leymann, F.: Faster and More focused control-Flow Analysis for Business Process Models Through SESE Decomposition. In: Service-Oriented Computing- Proceedings of the 5th ICSOC 2007, Lecture Notes in Computer Science 4749, 2010, Springer Verlag, 43–55.

[108] Vasudevan, S.; Emerson, E.A.; Abraham, J.A.: Efficient model checking of hardware using conditioned slicing. In: Proc. of the 4th International Workshop on Automated Verification of Critical Systems (AVOCS

2004), Electronic Notes in Theoretical Computer Science (ENTCS) 128 (6), 2005, Elsevier Science Publishers, 279–294.

[109] Vasudevan, S.; Emerson, E.A.; Abraham, J.A.: Improved Verification of Hardware Designs Through Antecedant Conditioned Slicing. In: International journal of Software Tools and Technology Transfer (STTT) 9, 1, 2007, Springer Verlag, 89-101.

[110] Vogler, W.; Wollowski, R.: Decomposition in Asynchronous Circuit Design. In: Concurreny and Hardware Design, Advances in Petri Nets, Lecture Notes in Computer Science 2549, 2002, Springer Verlag, 152–190.

[111] Wang, C.; Yang, Z.; Kahlon, V.; Gupta, A.: Peephole partial order reduction. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 2008), Lecture Notes In Computer Science , 2008, Springer Verlag, 283–396.

[112] Weiser, M.: Program slicing. In: Proceedings of the 5th international conference on Software engineering, IEEE Press Piscataway, NJ, USA, 1981, 439–449.

[113] The Workflow Management Coalition: Terminology & Glossary. `http://www.wfmc.org/reference-model.html`, 3, 1999.