Fakultät für Mathematik

Lehrstuhl I7, Theoretische Informatik

Workflow Nets: Reduction Rules and Games

Philipp Emanuel Hoffmann

Vollständiger Abdruck der von der Fakultät für Mathematik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Mathematik

genehmigten Dissertation.

Vorsitzender: Prof. Dr. Martin Brokate

Prüfer der Dissertation:

1. Prof. Dr. Francisco Javier Esparza Estaun
2. Prof. Dr. Jörg Desel

Die Dissertation wurde am 03.03.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Mathematik am 04.07.2017 angenommen.

I assure the single handed composition of this thesis only supported by declared resources.

Garching,

Zusammenfassung

Workflownetze sind eine Klasse von Petrinetzen, die als Formalismus für Business-Prozesse verbreitet Anwendung findet. Viele Modelle nutzen Workflownetze, um eine formale Semantik zu definieren und die dargestellten Prozesse zu analysieren.

In dieser Arbeit betrachten wir insbesondere eine Analysemethode, die regelbasierte Reduktion genannt wird. Dabei wird eine Menge von Regeln definiert; diese werden dann wiederholt angewendet, um die Größe des Netzes schrittweise zu reduzieren. Die Regeln sind dabei so gewählt, dass sie die zu studierende Eigenschaften wie "Wohlgeformtheit" des Netzes erhalten.

Wir präsentieren einen Algorithmus, der für eine bedeutende Klasse der Workflownetze, genannt Free-Choice Netze, vollständig ist. Dies bedeutet, dass er jedes wohlgeformte Netz in dieser Klasse (und nur die wohlgeformten Netze) zu einem trivialen Netz reduzieren kann. Durch eine Erweiterung der Regeln gelingt es ebenfalls, probabilistische Workflownetze zu reduzieren und Eigenschaften wie die erwartete Anzahl an Transitionen bis zum Erreichen des Endzustandes zu berechnen. Das präsentierte Verfahren benötigt höchstens polynomiell viele Regelanwendungen in der Größe des Netzes.

Im letzten Teil dieser Arbeit gehen wir auf Spiele ein, die auf Workflownetzen gespielt werden. Wir nehmen an, dass einer der Spieler einige Entscheidungen des Workflownetzes kontrolliert, andere jedoch nicht, und stellen die Frage, ob dieser Spieler dann das Erreichen des Endzustandes verhindern oder erzwingen kann. Wir zeigen, dass das Problem im Allgemeinen schwer zu lösen ist, im Falle eines wohlgeformten Free-Choice Netzes jedoch in polynomieller Zeit entscheidbar ist.

Abstract

Workflow nets are a class of Petri nets that are used as formalism for business processes. Multiple models use workflow nets to define a formal semantics and analyze the represented processes.

In this thesis we mainly focus on an analysis method called rule-based reduction. In this method, one defines a set of rules which are then repeatedly applied to the net to reduce it step by step. These rules are constructed in a way that they preserve important properties such as "well-formedness" of the net.

We present an algorithm for an important class of workflow nets, called free-choice nets, that is complete for that class. This means that the algorithm reduces every well-formed net (and only those) in that class to a trivial net. By extending the rules we can also reduce probabilistic workflow nets and compute properties like the expected number of transitions fired until termination. The presented algorithm needs at most a polynomial number of rule applications in the size of the net.

In the last part of this thesis we investigate games on workflow nets. We assume that one player controls some of the decision points of the net, but not all, and ask whether this is enough to force termination of the net. We show that in general this problem has a high complexity, but in the case of sound free choice workflow nets it is decidable in polynomial time.

Acknowledgments

First and foremost, I would like to express my special appreciation to my advisor Professor Dr. Javier Esparza who has been an outstanding mentor and provided guidance whenever needed. You have always taken the time to read or listen to the current status of my research, discuss difficulties or give advice on what to pursue next and I am very thankful for that. I would also like to thank my second committee member, Professor Jörg Desel, with whom I had multiple fruitful discussions about our work. Furthermore, I want to thank my committee for their comments on and suggestions for my thesis, thank you. I would also like to thank the whole chair I7, many of you have provided advice or discussions which have helped me a lot. In particular, I want to thank Philipp Meyer for his help with LOLA and for providing the benchmarks used.

A special thanks to my family. I am very grateful for your love and the unconditional support you all have provided during my time at university and even before that, and I want to thank you for the sacrifices you've made on my behalf. Your encouragement helped me when I doubted if I chose the right path. I also thank my friends, in particular Christian Müller and Teresa Zauner, who have provided ample feedback on all of my papers, found more typos than I can count and were always available for a discussion. Finally, I want to thank my girlfriend, Melanie Strauss, who supported me at all times during the creation of this thesis.

I would also like to thank all the anonymous reviewers of my published work for their advice and suggestions, they were greatly appreciated.

Contents

Chapter 1

Preliminaries

Contents

1.1 Introduction

Properly structuring work in larger organizations to be efficient and to function flawlessly has always been a major topic in industry. In recent years, the trend is to model business processes using "process-aware" information systems [14]. While there exist multiple business process models such as UML diagrams [13], Business Process Modeling Notation (BPMN) [45] and Event-driven Process Chains [2], most models do not have a rigorous semantic and formalization. Thus, Petri nets [38] are often used to define the semantics of a modeling language or to model the process directly and enable formal reasoning.

Petri nets are a formalism for concurrent systems that can model complex interplay of concurrency, causal dependencies and conflict. They have been widely used and a rich field of mathematical theory has been developed [37, 10, 38, 35]. Usually, a subclass of Petri nets called workflow nets is used [6, 1, 4, 36], for which an explicit start and end exists.

Once the business process is modeled in a modeling language or directly as a workflow net, it is desirable to check for errors in the modeled process. A crucial criterion for a "good" process is *soundness* [1], the property that processes will not get stuck and will always be able to finish. In this thesis, we study ways to decide whether a workflow net is sound efficiently. We then modify our approach to cover various extensions of workflow nets.

Example 1.1.

Consider Figure 1.1 which depicts two workflow nets. Petri nets are directed graphs with two types of nodes: *places* (depicted as circles) and *transitions* (depicted as squares). Edges may only connect one type of nodes to the other, but not two nodes of the same type. Every place of a net may contain *tokens*, graphically represented by black dots. The tokens present on the places form the *marking*, the current state of the Petri net.

Transitions represent the active part of the Petri net. If for a given transition all places with edges leading to that transition (the so called *input places*) contain a token, that transition is *enabled* and may *fire*. When a transition fires, it consumes one of the tokens on each of its input places and produces one token on each of its *output places*, the places at which an edge originating at the transition ends. In the net on the left of Figure 1.1, there is a single token on the place i which is the only input place to the transition t_1, thus that transition is enabled. Firing it removes the token from i and puts a token on s_1 and s_2 each. Then, t_2, t_3 and t_4 are all enabled.

Workflow nets have two special places, i and o, the input and output place. We begin with a single token on the input place, the workflow ends when there is a single token on the output place and no token elsewhere (called the *final marking*). Any other marking that does not enable a transition is considered a *deadlock*. For a workflow it is of course desirable that the workflow can always be completed, i.e. the final marking can always be reached. This property is called *soundness*.

The workflow net on the left of Figure 1.1 is sound, but the one on the

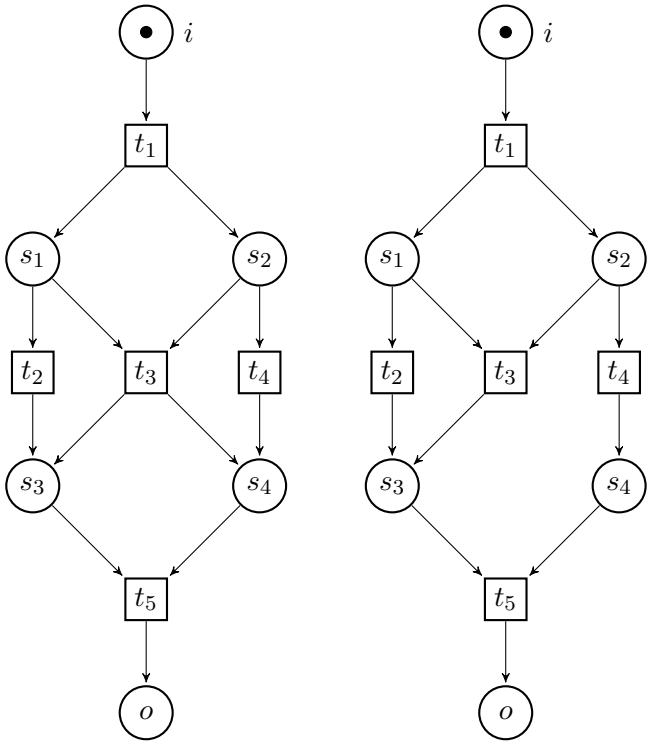


Figure 1.1: Two workflow nets.

right is not: after t_1 fires the places s_1 and s_2 are marked. If t_2 (t_4) fires, the workflow can continue with t_4 (t_2) and then finish with t_5. If however t_3 fires, a deadlock is reached.

A lot of work has already been devoted to checking soundness of workflow nets [1, 3, 6, 24, 20, 12] and weakening or strengthening the definition of soundness [24, 32, 23] and it is impossible to give a complete overview here. Unfortunately, due to the expressiveness of workflow nets, analysis of important properties such as soundness is often only possible with very high complexity [15]. However, for a subclass of workflow nets called free-choice workflow nets, polynomial decision procedures for many problems exist [6, 10]. Furthermore, free-choice workflow nets are still expressive enough to model most business processes [5, 21].

When modeling a business process, it is desirable to also include additional information besides the current processing state such as a piece of data that gets manipulated during the process. A workflow net cannot model such data and does not have something like an "output value". We will use different extensions of workflow nets such as colored workflow nets [27] to address these shortcomings.

Example 1.2.

In Figure 1.2 a fragment of a workflow net with data is depicted. Assume that the tokens carry data in the form of natural numbers. In our example, s_1 contains a token with value 3 and s_2 contains a token with value 5. Transition t_1 takes a token from s_1. The data carried by the token on s_1 is called x and t_1 produces two tokens, one on s_2 with data $f(x)$ for some function f, and one on s_3 with data $g(x)$ for some function g. For our example, let $f(x) = x^2$ and

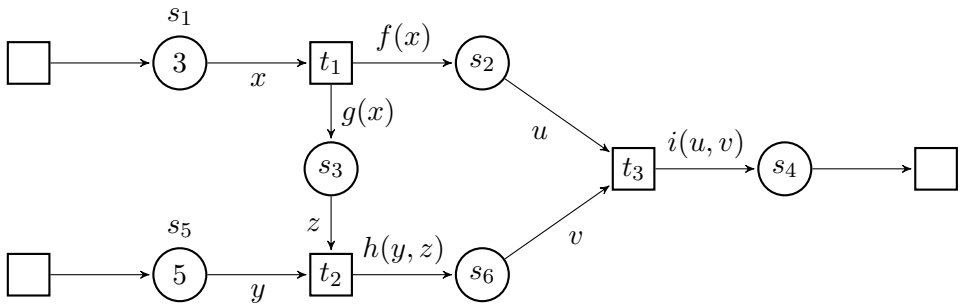


Figure 1.2: Fragment of a workflow net with data

$g(x) = x + 1$. Then the two tokens produced are one with value 9 on s_2 and one with value 4 on s_3. Thereafter t_2 is enabled and may consume the tokens on s_3 and s_5 with data $z = 4$ and $y = 5$ to produce a token on s_6 with data $h(y, z)$ for some function h.

There already exist a number of tools such as LoLA [47], an explicit tool which uses state exploration methods, and Woflan [43], which uses structural reduction and S-coverability analysis as well as explicit methods, which can check soundness of a workflow net. However, to our knowledge they do not allow for an extension with data. Furthermore, since they apply space exploration methods and the state space of a Petri net (i.e. the space of all reachable markings) is potentially exponential in size of the net (known as *state space explosion* [40]), they give no guarantee of a polynomial runtime.

Our work on workflow nets is closely related to earlier work on negotiations [16, 17, 11], a model that is quite similar to workflow nets with special constraints so that the number of tokens is a constant and each token has an identity and can be traced as it moves through the negotiation.

The algorithm presented in this thesis is based on *reduction*. In reduction, a set of rules is repeatedly applied to the net. Using reduction rules has already been proposed by other authors [35, 10]. These rules are chosen such that they preserve the properties one wishes to study so that after the net is reduced, it is easy to see whether the property holds. We initially concern ourselves only with the property of soundness. For free choice workflow nets our algorithm reduces all sound nets and only those to a trivial net, making it easy to decide whether the original net was sound. It furthermore takes at most polynomial time in the size of the net.

We then propose two extensions of workflow nets, data and probability, and show how to extend the rules such that properties like the input-output data relation or the expected number of transitions until termination can be computed.

Finally, we turn to a different topic, games on workflow nets. Games frequently arise in synthesis problems [7, 34, 39]. In synthesis, the target is to automatically create a system or controller that adheres to a given specification. This problem can be translated to a game between the system or controller that should be synthesized and the (malicious) environment, with the system winning if the specification is satisfied. Finding a winning strategy implies that there is a system that satisfies the specification against all possible environments.

For games on workflow nets, we assign control over clusters, which are the decision points of the net, to the players and study whether one player can force termination

of the net. Intuitively, one can think of a workflow involving different teams in a company. One of the teams is known to be quite lazy. The question we try to answer is whether this team alone (which only controls some of the decisions made during the execution of the workflow) has the power to prevent the workflow from reaching its designated end.

Similar to games played on pushdown automata [44], vector addition systems with states (VASS) [8], counter machines [29], or asynchronous automata [34], games on workflow nets can be translated into games played on the (reachable part of the) state space. However, due to the state space being possibly exponentially larger than the workflow net itself, an explicit computation is often undesirable. We study the general complexity of solving games in the size of the workflow net instead. This problem turns out to have rather high complexity, but in the case of sound free choice nets we give a polynomial decision algorithm.

The remainder of this chapter introduces basic notions of mathematics that we will use throughout this thesis. In Chapter 2 we introduce Petri nets and workflow nets and present general results regarding those nets. This chapter is largely based on the work of Desel and Esparza [10]. Chapter 3 then introduces our reduction algorithm and its extensions and is based on earlier work published in FASE 2016 [18] and QEST 2016 [19]. Finally, games on workflow nets are studied in Chapter 4, based on earlier work on negotiation games published in GandALF 2015 [25].

1.2 Math Basics/Used Symbols

The natural numbers are denoted by $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$. The reals are denoted by $\mathbb{R}$, the non-negative reals are $\mathbb{R}_{\geq 0}$, the positive reals are $\mathbb{R}_{>0}$.

For a set Σ, the set 2^{Σ} denotes the power set of Σ, or equivalently, the set of functions from Σ to $\{0, 1\}$ which each can be identified with a subset of Σ.

For a set Σ, the set Σ^* consists of all finite sequences of elements in Σ, the set Σ^ω consists of all infinite sequences of elements in Σ.

For a sequence σ, we denote the i-th element of σ by $\sigma(i)$, the prefix of σ ending at $\sigma(i)$ is denoted by σ^i and for finite sequences the last element is denoted by $last(\sigma)$.

A graph $G = (V, E)$ is a tuple consisting of nodes V and edges E. The edges may be directed or undirected, consequently the graph is either directed or undirected.

A path $\pi = v_1 v_2 \ldots v_k$ is a sequence of nodes of length $k \geq 1$ such that there is an edge between v_i and v_{i+1} for all $1 \leq i < k$. Undirected paths are paths in the undirected graph resulting by omitting edge directions. A cycle is a path $\pi = v_1 v_2 \ldots v_k$ of length $k \geq 2$ with $v_1 = v_k$.

We say that a (directed or undirected) graph is connected if for every two nodes v_1, v_2 there is an *undirected* path (of arbitrary length) starting in v_1 and ending in v_2. A directed graph is strongly connected if for every two nodes v_1, v_2 there is a *directed* path (of arbitrary length) starting in v_1 and ending in v_2.

1.2.1 Probability Theory

In this section we will introduce Markov chains and Markov decision processes (MDPs). Both are probabilistic models which we will use to give a semantics to our model.

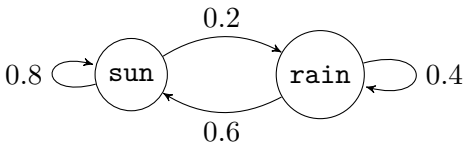


Figure 1.3: An easy Markov chain

Markov Chains

We begin by introducing Markov chains. A Markov chain is a probabilistic process characterized by its memoryless nature: From a certain state, the transition probabilities to other states are fixed and remain the same.

Example 1.3.

We use a well known example of a very simplified weather model, see for instance [46]. We assume that if today the sun shines, tomorrow with probability 0.8 it will be sunny and with probability 0.2 it will rain. If today it rains, tomorrow with probability 0.4 it will rain too and with probability 0.6 the sun will shine.

We can represent this Markov chain with two states (*rain* and *sun*) by its transition matrix:

$$\begin{array}{c} \\ sun \\ rain \end{array} \begin{array}{c} \begin{array}{cc} sun & rain \end{array} \\ \begin{bmatrix} 0.8 & 0.2 \\ 0.6 & 0.4 \end{bmatrix} \end{array}$$

Another representation which is typically used for Markov chains can be seen in Figure 1.3. Every state is drawn as a circle and the edges are labeled with the transition probabilities.

Markov chains with discrete time steps (days in the above example) are sometimes called discrete-time Markov chains (DTMCs).

Definition 1.1 (Discrete-Time Markov Chain (DTMC))**.** *A discrete-time Markov chain is a sequence of random variables $X_1, X_2, X_3 \ldots$ which all share the same countable set of outcomes Ω, called the states of the DTMC, and also fulfill the* Markov property*:*

$$P[X_{n+1} = c_{n+1} \mid X_1 = c_1 \wedge \ldots \wedge X_n = c_n] = P[X_{n+1} = c_{n+1} \mid X_n = c_n]$$

for any n and for any $c_1, \ldots, c_n$ such that

$$P[X_1 = c_1 \wedge \ldots \wedge X_n = c_n] > 0.$$

A DTMC may have an initial state $c \in \Omega$ which means that $X_1 = c$ is fixed, that is, $P[X_1 = c] = 1$.

In this thesis we are only concerned with discrete-time Markov chains and we always mean DTMCs when we write Markov chain.

The definition above exactly captures that the process is memoryless: the transition from the current state X_n to the next state X_{n+1} does solely depend on X_n and

not on and past event X_k where $k < n$. We therefore can describe a Markov chain by a series of transition probability matrices $(\Pi_n)_{n\in\mathbb{N}}$ where each Π_k describes the transition probabilities between the states in Ω in step k. If all those matrices are equal, the Markov chain is called *time-homogeneous*. We will only consider time-homogeneous Markov chains in this thesis. Such Markov chains can be described by a single transition matrix Π.

A path in a Markov chain is a finite or infinite non-empty sequence $\pi = c_1c_2c_3\ldots$ where c_i is the outcome of the i-th random variable X_i and $P[X_{n+1} = c_{n+1} \mid X_n = c_n] > 0$ for every $i \geq 1$. As for paths in any graph, we denote by $\pi(i)$ the i-th state along π (i.e., the state c_i), and by π^i the prefix of π ending at $\pi(i)$ (if it exists). For a finite path π, we denote by $last(\pi)$ the last state of π.

We define a probability measure on the Borel sets of infinite paths in a Markov chain with cylinder sets [28] of finite paths π as basic sets: $cyl(\pi) = \{\pi\pi' \mid \pi' \in \Omega^\omega\}$ are the infinite paths that begin with π. The probability of the cylinder of $\pi = c_1c_2c_3\ldots c_n$ is $P[cyl(\pi)] = P[X_1 = c_1] \cdot P[X_2 = c_2 \mid X_1 = c_1] \cdot \ldots \cdot P[X_n = c_n \mid X_{n-1} = c_{n-1}]$.

Example 1.4.

In our weather example, let $c_1 = sun$ be the initial state and consider the path *sun rain rain*. This path defines a cylinder set of all possible weather sequences starting with sun today, then two days of rain. The probability that this cylinder occurs is $1 \cdot 0.2 \cdot 0.4 = 0.08$.

Markov Decision Processes

Markov chains are governed entirely by probability. We now introduce Markov decision processes which also add an element of choice.

Example 1.5.

We extend the weather example by adding the possibility to buy an umbrella. There are now four possible states: Two where we have an umbrella and it either rains or the sun is shining, and two where we do not have an umbrella and it either rains or the sun is shining.

We can now describe a Markov decision process: Every day, as long as we do not have an umbrella, we decide whether we buy an umbrella or not. Depending on our decision, we either move to one of the two states where we have an umbrella, or to one of the two where we do not have one.

Figure 1.4 shows a graphical representation of this extended model. We see that the states without umbrella have two outgoing edges, buy an umbrella (u) or go without an umbrella (x). These edges lead to little circles where the probabilistic decision of the weather is taken. The two states where we already have an umbrella have only one outgoing edge because we already have an umbrella and do not want to buy another one.

We turn to the formal definition of this model. For a set Q, let $dist(Q)$ denote the set of probability distributions over Q.

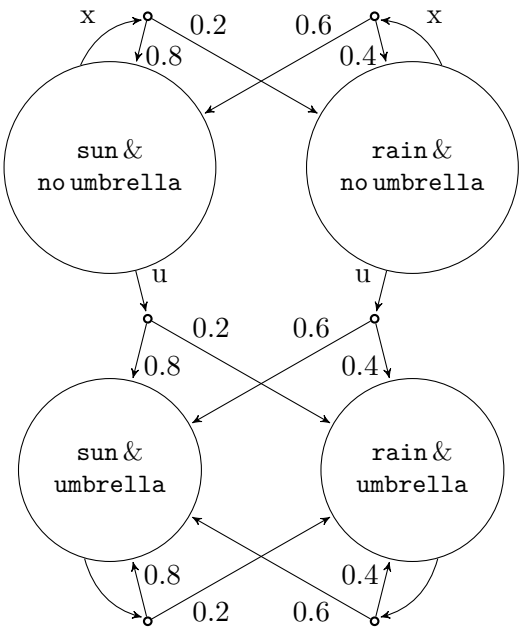


Figure 1.4: Extension to a Markov decision process

Definition 1.2 (Markov Decision Process)**.** *A Markov decision process (MDP) is a tuple* $\mathcal{M} = (Q, q_0, Steps)$ *where* Q *is a set of states,* $q_0 \in Q$ *is the initial state, and* $Steps\colon Q \to 2^{dist(Q)}$ *is the probability transition function.*

For a state q, a probabilistic transition is taken by first nondeterministically choosing a probability distribution $\mu \in Steps(q)$ and then choosing the successor state q' probabilistically according to μ.

A path in an MDP is a finite or infinite non-empty sequence $\pi = q_0 \xrightarrow{\mu_0} q_1 \xrightarrow{\mu_1} q_2 \dots$ where $\mu_i \in Steps(q_i)$ for every $i \geq 0$. Once again, we denote by $\pi(i)$ the i-th state along π (i.e., the state q_i), and by π^i the prefix of π ending at $\pi(i)$ (if it exists). For a finite path π, we denote by $last(\pi)$ the last state of π.

To resolve the decisions, we introduce a probabilistic scheduler.

Definition 1.3 (MDP-scheduler)**.** *An* MDP-scheduler *is a function that maps every finite path* π *of* $\mathcal{M}$ *to a distribution of* $Steps(last(\pi))$*.*

Notice that once we fix a scheduler, the MDP again collapses to a (possible infinite state) Markov chain. We first illustrate by example.

Example 1.6.

In our weather example, assume that we buy an umbrella the first time it rains. The scheduler thus is a function that maps every path in (`sun`&`no umbrella`)* to the distribution where x has weight 1 and every path in (`sun`&`no umbrella`)*`rain`&`no umbrella` to the distribution where u has weight 1. The resulting Markov chain is depicted in Figure 1.5.

In general, the decisions of the scheduler might differ on different visits to the same state of the MDP, so the Markov chain might have more states than the MDP.

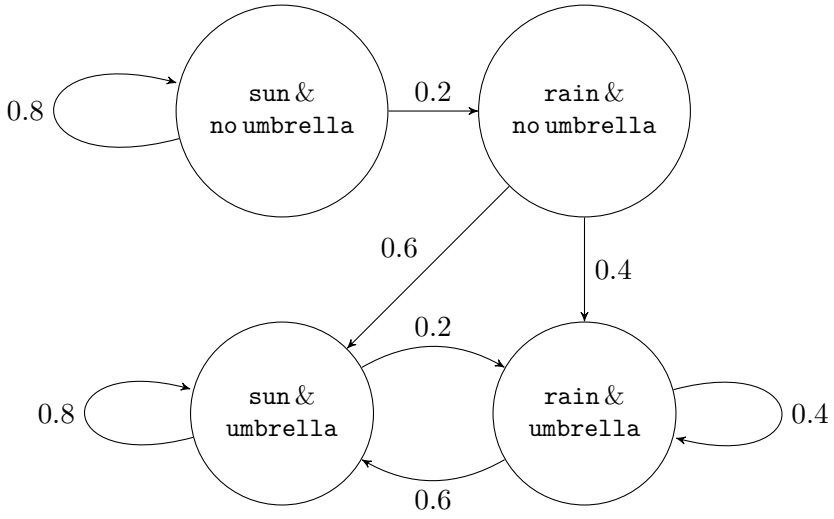


Figure 1.5: Collapse to a Markov chain

Definition 1.4 (Collapse to a Markov chain)**.** *For a given MDP-scheduler S, let ${Paths_{Fin}}^S$ denote all finite paths $\pi = q_0 \xrightarrow{\mu_0} q_1 \xrightarrow{\mu_1} q_2 \ldots$ starting in s_0 and satisfying $\mu_i = S(\pi^i)$ for every $i \geq 0$.*

We define a Markov chain with the set $\Omega = Q \times {Paths_{Fin}}^S$, initial state (q, q) and the transition probability $P[X_{i+1} = (q', \pi') \mid X_i = (q, \pi)] = S(\pi)(q')$ if $\pi' = \pi q'$ and 0 otherwise, i.e. the transition probability is exactly as in the distribution of Steps that the scheduler has chosen. We say that under the scheduler S, the MDP collapses to this Markov chain.

Let $Paths^S$ be the infinite paths for a given MDP-scheduler. We can use the Markov chain we just defined to fix a probability measure $Prob^S$ on $Paths^S$ by identifying paths in the MDP and paths in the Markov chain in the obvious way.

We introduce the notion of rewards for an MDP.

Definition 1.5 (Reward)**.** *A reward function for an MDP is a function $rew : Q \to \mathbb{R}$ together with a commutative binary operator $\oplus$ on $\mathbb{R}$. For a path π and a set of states F, the reward until F is reached is*

$$R(F, \pi) := \bigoplus_{i=0}^{\min\{j \mid \pi(j) \in F\}} rew(\pi(i))$$

if the minimum exists, and ∞ otherwise. Given an MDP-scheduler S, the expected reward to reach a set of states F is defined as

$$E^S(F) := \int_{\pi \in Paths^S} R(F, \pi) \mathrm{d}Prob^S$$

where the integral is a Lesbeque integral [22].

Example 1.7.

We continue with the umbrella example. Remember that the initial state is *sun*. As a reward function we choose $rew(\texttt{rain\&no umbrella}) = -1$ and set all other rewards to $+1$, the operator $\oplus$ is the minimum operator. That means we reach a reward of -1 for a path that contains the state `rain&no umbrella` and the reward of 1 for all other paths. For our scheduler that chooses to buy an umbrella on the first rainy day, we will eventually have rain and thus get a reward of -1 almost surely. To ensure a higher expected reward, a better Scheduler should buy an umbrella right away. This way, the state `rain&no umbrella` can never be reached and the expected reward will be 1.

1.2.2 Mazurkiewicz Equivalence

We will now turn to parallel computations and the subtleties thereof. In such computations, there may be steps which can be executed independently of each other and steps where one depends on the other. Such dependencies can be described by an independence relation. Using this relation, multiple observed sequential executions can be grouped together to an equivalence class of executions which intuitively describe different interleavings of independent actions, but the same computation in total. This equivalence relation was introduced by Mazurkiewicz [33] and is therefore called Mazurkiewicz equivalence.

Example 1.8.

We describe a production facility where every day there are three production steps A, B and C to be completed. Steps A and B can be done in parallel, but step C can only be executed after the other steps have been completed. Each step requires a single worker. The workers report to a supervisor when they begin or finish a production step and of course every worker begins a production step before he finishes it. We will denote by As and Af the reports that step A starts or has been finished, and similarly Bs, Bf, Cs, Cf.

The supervisor might get multiple different reports such as As, Bs, Bf, Af, Cs, Cf or As, Af, Bs, Bf, Cs, Cf, but the sequence As, Bs, Bf, Cs, Af, Cf cannot happen under normal circumstances. The first two executions will have the same meaning of "everything has worked", they are, in some sense, equivalent.

Formally, we define an alphabet Σ as a set of actions and an independence relation $I \subseteq \Sigma \times \Sigma$ as a symmetric relation on Σ. Together we call (Σ, I) an independence alphabet.

Definition 1.6 (Mazurkiewicz Equivalence)**.** *For an independence alphabet (Σ, I), the* Mazurkiewicz equivalence*, denoted by $\equiv$, is the smallest congruence such that $\sigma t_1 t_2 \sigma' \equiv \sigma t_2 t_1 \sigma'$ for every $\sigma, \sigma' \in \Sigma^*$ and for any t_1, t_2 where $(t_1, t_2) \in I$.*

Example 1.9.

For the above example of a production facility, the independence relation would contain the pairs (As, Bs),(As, Bf),(Af, Bs),(Af, Bf) (and their counterparts to make the relation symmetric) as the steps A and B are independent of each other. An example for an independence class would be $[As, Af, Bs, Bf, Cs, Cf]$ which contains all reports where

- As comes before Af
- Bs comes before Bf
- the report ends with Cs,Cf

which is exactly the class of "good" reports.

Chapter 2

Petri Nets and Workflow Nets

Contents

2.1 Petri Nets

2.1.1 Introduction

Petri nets are a formalism for concurrent systems which is widely used and has been studied extensively [35, 10, 38]. They provide an easy to understand graphical notation for stepwise processes which includes key features like conflict and parallelism. However, in contrast to similar notions like UML diagrams or event-driven process chains (EPCs) [2], Petri nets feature a precise mathematical definition and semantic that has given rise to a rich field of mathematical theory around those nets.

The analysis of Petri nets resulted in various algorithms to decide for different properties whether they hold for a given Petri net. While for most interesting properties such algorithms exist, they have high lower bounds (EXPSPACE-hard [15]) and much work has been devoted to finding suitable subclasses where analysis can be done more efficiently.

One particularly interesting subclass is the class of free choice Petri nets. For these nets and for important properties like liveness and boundedness, polynomial-time algorithms exist [10]. However, attempts to generalize these results to larger classes have been unsuccessful so far [15, 31].

The purpose of this section is to familiarize the reader with Petri nets. We first introduce Petri nets and give their formal definition. We also introduce the notions of liveness and boundedness. Thereafter we turn to a subclass, the aforementioned free choice Petri nets, and state one of the central theorems regarding liveness in free choice Petri nets, Commoner's theorem. Finally we briefly study the notion of S- and T-components, the coverability theorems and the relation of these components and S- and T- invariants.

All results presented in this section are taken from [10].

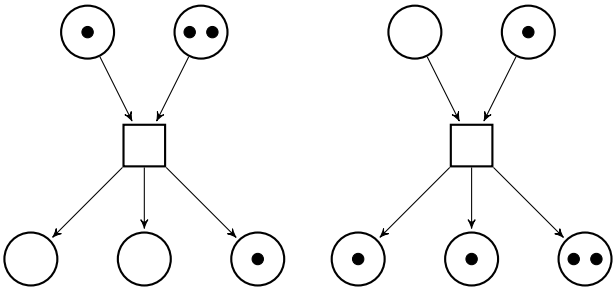
2.1.2 Syntax and Semantics

We begin with an informal introduction of Petri nets using the graphical notation. A Petri net consists of two components: a *net* and an *initial marking*. A net is a directed graph with two kinds of nodes: *places*, represented as circles, and *transitions*, represented as squares. Edges may only connect one type of nodes to the other, but not two nodes of the same type. Every place of a net may contain *tokens*, graphically represented by black dots. The tokens present on the places form the *marking*, the current state of the Petri net.

Transitions represent the active part of the Petri net. If for a given transition all places with edges leading to that transition (the so called *input places*) contain a token, that transition is *enabled* and may *fire*. When a transition fires, it consumes one of the tokens on each of its input places and produces one token on each of its *output places*, the places at which an edge originating at the transition ends.

Example 2.1.

Figure 2.1a shows a transition that is enabled. Firing this transition consumes one token on each input place and produces one token on each output place, leading to the marking shown on in Figure 2.1b.



(a) Before the firing (b) After the firing

Figure 2.1: An example of a transition firing

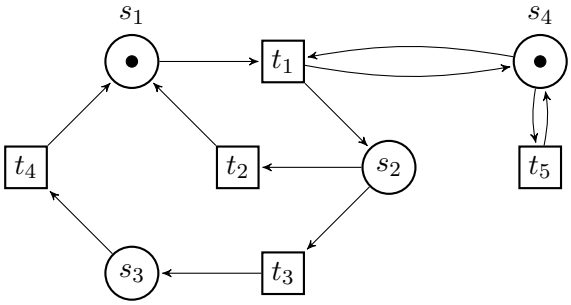


Figure 2.2: Modeling of different behaviors

By composing multiple transitions to form a net, various behaviors such as *casual dependencies*, *conflict*, *concurrency* and others can be simulated by a Petri net.

Example 2.2.

Figure 2.2 shows a Petri net in which transition t_2 can only fire after t_1 has fired (casual dependency). After t_1 has fired, t_2 and t_3 are both enabled and firing one of them will disable the other (conflict). Transition t_5 is enabled initially and will always stay enabled. Thus it may fire at any point (concurrency).

We now turn to the formal definitions.

Definition 2.1 (Net)**.** *A net is a tuple $\mathcal{N} = (S, T, F)$ where S and T are two finite, disjoint sets and $F \subseteq (S \times T) \cup (T \times S)$ is the flow relation.*

The elements in S are called *places*[1], the elements in T are called *transitions*. The elements in $S \cup T$ are called the *nodes* of the net.

Definition 2.2 (Pre-Set, Post-Set)**.** *For a node $n \in S \cup T$, its* pre-set $^\bullet n$ *consists of the nodes $\{n' : (n', n) \in F\}$. Similarly, its* post-set $n^\bullet$ *contains the nodes $\{n' : (n, n') \in F\}$.*

For a place s, its pre-set $^\bullet s$ contains the *input transitions* of s, its post-set contains the *output transitions* of s. Similarly, the pre-set of a transition t contains its *input places*, the post-set of t contains its *output places*.

[1]Petri nets were originally defined in German where places are called “Stellen”

Example 2.3.

Consider again Figure 2.2. The places are the set $\{s_1, s_2, s_3, s_4\}$, the transitions are the set $\{t_1, t_2, t_3, t_4, t_5\}$. The edge relation F contains for example (s_1, t_1), (t_1, s_2) and (t_2, s_1) but not (s_1, t_2). The post-set $t_4^\bullet$ contains the place s_1. The pre-set ${}^\bullet s_1$ contains the transition t_4 and also the transition t_2.

Since nets are basically graphs, the usual notion of paths, cycles and connectedness applies to them. We also formalize the notion of a marking.

Definition 2.3 (Marking)**.** *A* marking *of a net (S, T, F) is a function $M : S \to \mathbb{N}$.*

A marking can be represented by a vector $(M(s_1), M(s_2), \ldots)$ where $s_1, s_2, \ldots$ is an arbitrary but fixed order of the places.

A place s is *marked* by a marking if $M(s) > 0$. A set of places R is marked if some place of R is marked.

A marking *enables* a transition t if it marks every input place of t. An enabled transition may *fire* which changes the marking M to a marking M' defined as follows:

$$M'(s) = \begin{cases} M(s) - 1 & \text{if } s \in {}^\bullet t \text{ and } s \notin t^\bullet \\ M(s) + 1 & \text{if } s \notin {}^\bullet t \text{ and } s \in t^\bullet \\ M(s) & \text{if } s \in {}^\bullet t \text{ and } s \in t^\bullet, \text{ or } s \notin {}^\bullet t \text{ and } s \notin t^\bullet \end{cases}$$

We write $M \xrightarrow{t} M'$ to denote this marking change when t fires. For a sequence of transitions $\sigma = t_1 t_2 t_3 \ldots$, we say that M enables σ if $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_3} \ldots$. In particular, M enables t_1 and after t_1 fires, the marking M_1 enables t_2 and so on. For a finite sequence σ enabled at M, we write $M \xrightarrow{\sigma} M'$ to denote that after all transitions in σ have fired, the marking M' was reached.

Example 2.4.

The marking in Figure 2.2 marks the places s_1 and s_4 and is given by the vector $(1, 0, 0, 1)$. At this marking, t_1 and t_5 are enabled. After t_1 fires, the new marking is $(0, 1, 0, 1)$. We also write $(1, 0, 0, 1) \xrightarrow{t_1} (0, 1, 0, 1)$. At this new marking, for example the sequence $\sigma = t_2, t_1, t_5, t_3$ is enabled and $(0, 1, 0, 1) \xrightarrow{\sigma} (0, 0, 1, 1)$.

We call a marking M' *reachable* from M if there is a transition sequence σ enabled at M such that $M \xrightarrow{\sigma} M'$

Definition 2.4 (Petri Net)**.** *A Petri net is a pair $\mathcal{P} = (\mathcal{N}, M_0)$ where $\mathcal{N}$ is a connected net with at least one place and one transition and M_0 is a marking of $\mathcal{N}$.*

For a Petri net, we call M_0 the *initial marking*. We call the markings reachable from M_0 the reachable markings.

We introduce two properties of interest related to Petri nets, *liveness* and *boundedness*.

Definition 2.5 (Liveness)**.** *A Petri net is* live*, if for every transition t and every reachable marking M, there is a marking reachable from M that enables t.*

Liveness implies that the system can never reach a position where no transition is enabled, a so-called *deadlock*.

Definition 2.6 (Deadlock)**.** *A marking that enables no transition is called a* deadlock.

Proposition 2.7. *In a live Petri net, no deadlock is reachable.*

Proof. By Definition 2.4, a Petri net has at least one transition and from every reachable marking a marking can be reached that enables this transition. □

The second property, *boundedness*, limits the number of tokens on a place.

Definition 2.8 (Boundedness)**.** *A Petri net is* bounded *if there is a bound k such that no reachable marking puts more than k tokens on one place. A Petri net is called* safe *or* 1-safe *if that bound is* 1*.*

The Petri net in Figure 2.2 is live, bounded and even 1-safe.

Liveness and boundedness are defined as *behavioral* properties (in contrast to structural properties like connectedness) via the reachable markings. Such properties are in general difficult to check (EXPSPACE-hard, see [15] for a general study) because the set of reachable markings may be very large or even infinite (if the net is unbounded). We now turn to a subclass of Petri nets, free choice nets, for which there exists an efficient procedure to check whether the net is live and bounded.

2.1.3 Free Choice Nets

As seen in the previous section, Petri nets allow one to model different behaviors like conflict and concurrency. In free choice Petri nets, we do not allow certain combinations of behavior by syntactically restricting the net. In particular, we do not allow a combination of conflict and synchronization, the latter being a transition where the system waits for concurrent parts of the net to have reached a certain point.

Example 2.5.

Consider the Petri net shown in Figure 2.3. There are two tokens initially, on s_1 and s_2. The transitions t_1 and t_2 are enabled. After t_2 fires, the transition t_3 aims to synchronize the two tokens on s_1 and s_3. The transition t_1 however, is in conflict with t_3 but its pre-set does not contain s_3. This combination of synchronization and conflict is not allowed in free choice Petri nets.

Another way to look at the restriction is that initially the only transition enabled involving the token on s_1 is t_1. After t_2 fires, t_3 is enabled as well. This means that the token on s_1 has different choices depending on whether there is a token on s_3, thus the choice is not "free".

There are many equivalent definitions of free choice Petri nets [10], we give just one of them here.

Definition 2.9 (Free Choice)**.** *A net is* free choice *if for every two places $s_1, s_2 \in S$ either $s_1^\bullet \cap s_2^\bullet = \emptyset$ or $s_1^\bullet = s_2^\bullet$. A Petri net is free choice if its net is free choice.*

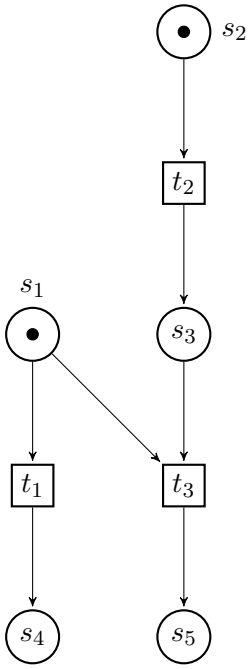


Figure 2.3: Synchronization and conflict

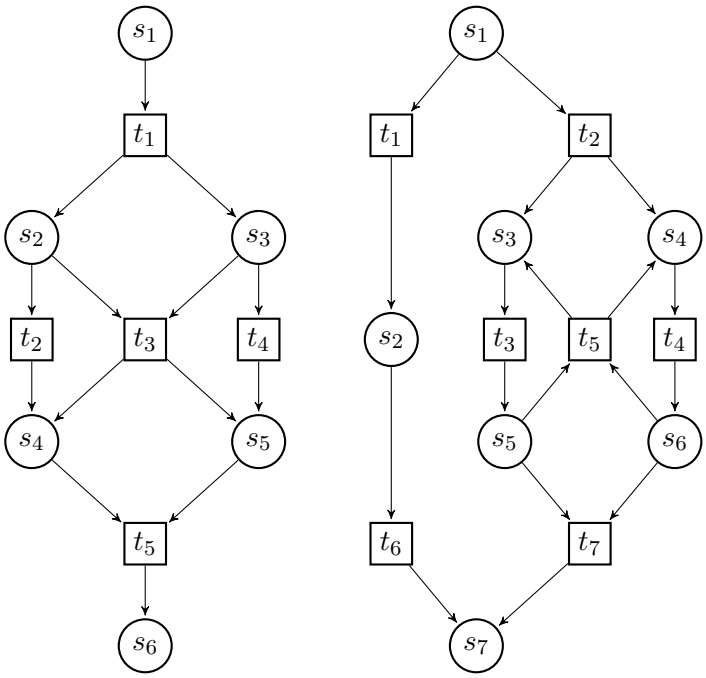


Figure 2.4: Two example nets

The net on the left of Figure 2.4 is not free choice, for example because of the places s_2 and s_3. The Petri net on the right of Figure 2.4 however is free choice.

We introduce the notion of clusters.

Definition 2.10 (Cluster)**.** *Let $\mathcal{N} = (S, T, F, i, o)$ be a net, $x \in S \cup T$ a node of $\mathcal{N}$. The cluster of x is the unique smallest set $[x] \subseteq S \cup T$ such that:*

- $x \in [x]$,
- *if $s \in S \cap [x]$ then $s^\bullet \subseteq [x]$, and*
- *if $t \in T \cap [x]$, then ${}^\bullet t \subseteq [x]$.*

A set $X \subseteq S \cup T$ is a cluster if $X = [x]$ for some node x.

Example 2.6.

The net in Figure 2.4 on the left contains four clusters: $\{s_1, t_1\}$, $\{s_2, s_3, t_2, t_3, t_4\}$, $\{s_4, s_5, t_5\}$ and $\{s_6\}$. Observe that some clusters may contain a place s and a transition t such that s is not a pre-place of t, for example the place s_2 is not a pre-place of t_4.

The clusters $[x]$ in free choice nets have a special structure: Every place $s \in [x]$ is a pre-place for every transition $t \in [x]$. This can easily be inferred from the definition of free choice net.

Example 2.7.

The net in Figure 2.4 on the right consists of six clusters: $\{s_1, t_1, t_2\}$, $\{s_2, t_6\}$, $\{s_3, t_3\}$, $\{s_4, t_4\}$, $\{s_5, s_6, t_5, t_7\}$ and $\{s_7\}$. In every cluster all places are pre-places of every transition in that cluster (and exactly those transitions).

It is easy to see that clusters, in free choice nets as well as in general nets, are either equal or disjoint, and thus the clusters of a net form a partition of that net. We say that a marking M marks a cluster if it marks every place in that cluster. Notice that this differs from our definition of M marks a set, where only some place of the set has to be marked.

We introduce the notion of *free choice clusters*, clusters of a net that offer free choice even if the net is not a free choice net.

Definition 2.11 (Free Choice Cluster)**.** *A cluster c is a* free choice cluster *if*

- *$s^\bullet = c \cap T$ for every $s \in C \cap S$, or equivalently*
- *${}^\bullet t = c \cap S$ for every $T \in C \cap T$.*

Notice that in free choice nets, every cluster is a free choice cluster, and the converse also holds as the following theorem states.

Theorem 2.12. *A net is free choice iff all of its clusters are free choice clusters.*

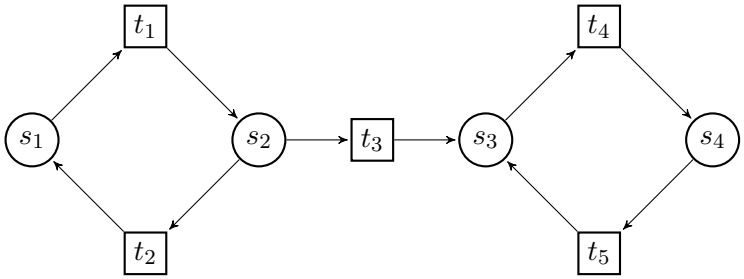


Figure 2.5: Siphons and Traps

2.1.4 Live and Bounded Free Choice Nets

We state some important results of [10] which we will use in the remainder of this thesis. The first result regards the complexity of deciding whether a net is live and bounded.

Theorem 2.13. *It can be decided in polynomial time whether a given Petri net $\mathcal{P}$ is live and bounded.*

For the second result regarding liveness in free choice Petri nets, we need the notion of *siphons* and *traps*.

Example 2.8.

Consider the net in Figure 2.5. If a marking M does not mark the set $\{s_1, s_2\}$, then no marking reachable from M will ever mark that set because the transitions putting a token on that set require a token already in the set. We call such a set a siphon.

Similarly, if a marking M marks the set $\{s_3, s_4\}$, then any marking reachable from M will mark that set because every transition that removes a token from the set $\{s_3, s_4\}$ also puts one token back into the set. Such a set is called a trap.

Definition 2.14 (Siphon, Trap)**.**
A non-empty set R of places of a net is a siphon *if* ${}^\bullet R \subseteq R^\bullet$.
A non-empty set R of places of a net is a trap *if* $R^\bullet \subseteq {}^\bullet R$.

Using siphons and traps, the following theorem characterizes liveness in free choice Petri nets in a structural way.

Theorem 2.15 (Commoner's Theorem)**.** *A free choice Petri net is live if and only if every siphon contains an initially marked trap.*

The remainder of this chapter focuses on *S-* and *T-components* as well as *S-* and *T-invariants*. We mostly collect important known results that we will later use. This part can be skipped and read when referred to later.

We first illustrate the concept of *S-components* by example.

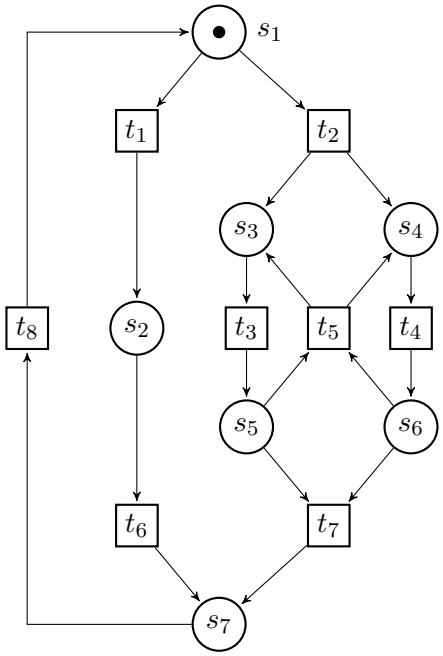


Figure 2.6: A live and bounded free choice Petri net

Example 2.9.

Consider the Petri net shown in Figure 2.6. It is a live and bounded free choice Petri net and was obtained from the free choice example on the right of Figure 2.4 by adding the transition t_8.

From the initial marking, there are two choices: t_1 and t_2 are enabled. If t_1 fires, there is neither choice nor concurrency until the token is back at s_1. If however t_2 fires, there are two tokens on s_3 and s_4 and the transitions t_3 and t_4 can fire concurrently.

This behavior is captured by the decomposition in Figure 2.7. Notice that some places like s_1 are present in both nets of the decomposition, but those places where there is true parallelism, s_3 to s_6, are only present in one of them. Furthermore, we have not "removed a choice": For every place, the input and output transitions are the same as in the original net.

One interpretation of this decomposition is that the Petri net is composed of multiple processes which sometimes take their steps synchronously and sometimes independently of each other. We now formalize of this kind of decomposition.

Definition 2.16 (S-Component)**.** *Let $\mathcal{N} = (S, T, F)$ be a net. An S-Component of $\mathcal{N}$ is a net $\mathcal{N}' = (S', T', F')$ such that*

- $S' \subseteq S$.
- $T' = \bigcup_{s \in S'} {}^\bullet s \cup s^\bullet$ *where* ${}^\bullet s$ *and* $s^\bullet$ *are the pre- and post-set in the original net* $\mathcal{N}$.
- *For all* $t \in T'$: $|{}^\bullet t| = 1 = |t^\bullet|$ *where* ${}^\bullet t$ *and* $t^\bullet$ *are the pre- and post-set in* $\mathcal{N}'$.
- $\mathcal{N}'$ *is strongly connected*

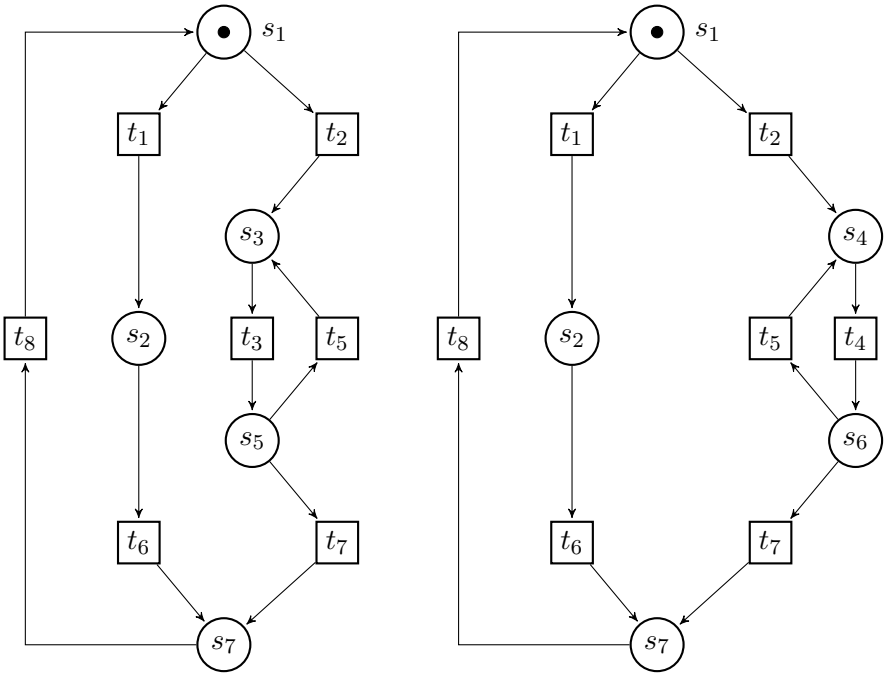


Figure 2.7: Decomposition of the net in Figure 2.6

We investigate some properties of S-components (and forget about the rest of the net for a moment). Due to the third condition in the above definition, the number of tokens in an S-component is constant: Every transition fired will consume and produce exactly one token. Due to the fourth condition that $\mathcal{N}'$ is strongly connected, a token can move from any place to any other place. Thus if there is a token in the S-component, every transition in this component can fire and the component is live.

It turns out that for free choice Petri nets, there is a strong connection between S-components which are always live and bounded, and liveness and boundedness of the Petri net as a whole.

Definition 2.17. *Let C be a set of S-components of a net. C is an S-cover if every place of the net belongs to an S-component of C. A net is covered by S-components if it has an S-cover.*

Theorem 2.18 (S-Coverability [10])**.** *The net of a live and bounded free choice Petri net is covered by S-components.*

Similar to S-components, we can define the notion of T-components as follows:

Definition 2.19 (T-component)**.** *Let $\mathcal{N} = (S, T, F)$ be a net. A T-component of $\mathcal{N}$ is a net $\mathcal{N}' = (S', T', F')$ such that*

- $T' \subseteq T$.
- $S' = \bigcup_{t \in T'} {}^\bullet t \cup t^\bullet$ *where* ${}^\bullet t$ *and* $t^\bullet$ *are the pre- and post-set in the original net* $\mathcal{N}$.
- *For all* $s \in S'$: $|{}^\bullet s| = 1 = |s^\bullet|$ *where* ${}^\bullet s$ *and* $s^\bullet$ *are the pre- and post-set in* $\mathcal{N}'$.
- $\mathcal{N}'$ *is strongly connected*

This is exactly dual to the definition of an S-component and just as for S-components, there is a coverability result for live and bounded free choice Petri nets.

Theorem 2.20 (T-Coverability)**.** *The net of a live and bounded free choice Petri net is covered by T-components.*

For T-components, we present another result related to activation of T-components.

Definition 2.21 (Activation of T-components)**.** *Let $\mathcal{N}'$ be a T-component of a net $\mathcal{N}$. A marking M of N activates $\mathcal{N}'$ if $\mathcal{N}'$ with the marking M restricted to $\mathcal{N}'$ is live.*

Theorem 2.22. *For every T-component $\mathcal{N}'$ in a live and bounded free choice workflow net there is a reachable marking M which activates $\mathcal{N}'$.*

We furthermore introduce the concept of S-invariants and T-invariants, an algebraic concept that for live and bounded free choice workflow nets turns out to be closely related to S-components and T-components.

Definition 2.23 (S-invariant, T-invariant)**.** *Let $\mathcal{N} = (S, T, F)$ be a net, I the incidence matrix, i.e. a matrix of size $|T| \times |S|$ where $I_{i,j} = 1$ iff $(t_i, s_j) \in F$. A solution v_s to the equation*

$$v_s \cdot I = 0 \tag{2.1}$$
$$v_s \neq 0 \tag{2.2}$$

is called an S-invariant. A solution v_t to the equation

$$I \cdot v_t = 0 \tag{2.3}$$
$$v_t \neq 0 \tag{2.4}$$

is called a T-invariant.

Definition 2.24 (Minimality of Invariants)**.** *A non-negative (S- or T-, resp.) invariant is minimal if it is not the sum of two non-negative (S- or T-, resp.) invariants.*

Theorem 2.25. *Every non-negative (S- or T-, resp.) invariant is the sum of minimal non-negative (S- or T-, resp.) invariants.*

We now state some results that relate invariants and components.

Proposition 2.26. *Every S-component of a net $\mathcal{N}$ induces an S-invariant v where $v(s) = 1$ iff s is in the S-component.*

Proposition 2.27. *Let v be a non-negative minimal S-invariant of a live and bounded free choice net. Then the places s such that $v(s) > 0$ together with their pre- and post-transitions form an S-component.*

Proposition 2.28. *Every T-component of a net $\mathcal{N}$ induces a T-invariant v where $v(t) = 1$ iff t is in the T-component.*

Proposition 2.29. *Let v be a non-negative minimal T-invariant of a live and bounded free choice net. Then the transitions t such that $v(t) > 0$ together with their pre- and post-places form a T-component.*

2.2 Workflow Nets

2.2.1 Introduction

Petri nets are a versatile and expressive tool to model various scenarios. One particular application for Petri nets is modeling of workflows. While many different industry standards like UML diagrams, Event-driven Process Chains (EPCs) [2] and Business Process Modeling Notation (BPMN) [45] exist, those usually do not feature a precise mathematical definition. Instead Petri nets can be used to define the semantics of those notations.

We will introduce *workflow nets*, a subclass of Petri nets with a distinguished beginning and end. Using Petri nets allows us to use the rich field of theory already developed and apply it to workflows. In particular, we will use results about liveness and boundedness, about S- and T-components and invariants in free choice systems.

One of the most important properties regarding workflow nets is *soundness*, a notion of well-formedness of the net. It captures the idea that a workflow net should never "get stuck" and should always be able to terminate. Since an analysis of the reachable states is impractical due to an exponential state space, we use an analysis technique known as *reduction*. This technique aims to reduce the state space of the net by applying *reduction rules* while preserving the properties we wish to study.

We also address an important shortcoming of workflow nets, the abstraction from any data. When modeling a business process, it might be desirable to also include additional information besides the current processing state. A workflow net cannot model such data and does not have something like an "output value" that depends on the processing and possibly an initial state. We will address this by introducing an extension to workflow nets called *colored workflow nets* [27].

The remainder of this section is devoted to the definition of workflow nets and soundness as well as extensions of workflow nets with data or probabilities.

2.2.2 Workflow Nets and Soundness

As mentioned in the introduction, workflow nets are a subclass of Petri nets which have two distinguished places, one where the workflow starts and one where it ends.

Definition 2.30 (Workflow Net)**.** *A* workflow net *is a tuple* $\mathcal{W} = (S, T, F, i, o)$ *where*

- (S, T, F) *is a net*
- $i, o \in S$ *are places with* ${}^\bullet i = o^\bullet = \emptyset$
- *The net* $(S, T, F \cup (o, i))$ *is strongly connected.*

The initial marking $\boldsymbol{i}$ *is the marking that only marks* i*. The final marking* $\boldsymbol{o}$ *is the marking that only marks* o*.*

We will call sequences enabled at $\boldsymbol{i}$ firing sequences. $Fin_{\mathcal{W}}$ is the set of all firing sequences of $\mathcal{W}$ that end in the final marking.

We introduce one of the most studied properties of workflow nets, *soundness*. Intuitively, a well-formed workflow net should never get stuck and always be able to terminate with the final marking.

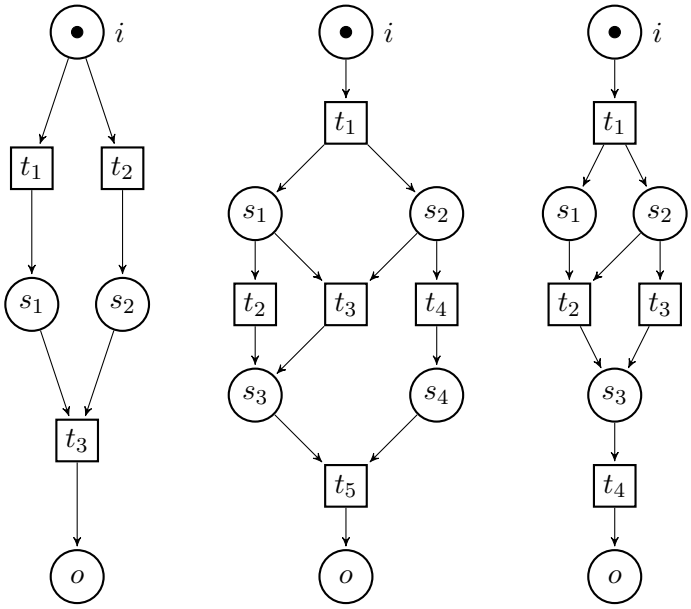


Figure 2.8: Three unsound workflow nets

Example 2.10.

Consider the three workflow nets in Figure 2.8. In the leftmost workflow net, after either t_1 or t_2 fires, no more transition is enabled and a deadlock is reached. In the second workflow net, after t_1 fires the places s_1 and s_2 are marked. If t_2 (t_4) fires, the workflow can continue with t_4 (t_2) and then finish with t_5. If however t_3 fires, a deadlock is reached. In the rightmost workflow net, after t_1 has fired either t_2 or t_3 can fire and then t_4 is enabled. If t_2 was fired, the final marking is reached after t_4, but if t_3 fired, a token on s_1 remains there and the final marking $\boldsymbol{o}$ can never be reached.

Formally, we define soundness as follows.

Definition 2.31 (Soundness)**.** *A workflow net is sound iff*

- *the final marking is reachable from any reachable marking, and*
- *every transition occurs in some firing sequence.*

The above semantic characterization can be expressed via liveness and boundedness as has been shown by van der Aalst [1].

Definition 2.32 (Extended Net)**.** *For a workflow net $\mathcal{W} = (S, T, F, i, o)$ we define the extended net $\overline{\mathcal{W}} = (S, T, F \cup (o, i))$.*

Theorem 2.33 ([1])**.** *A workflow net $\mathcal{W} = (S, T, F, i, o)$ is sound iff the Petri net $\mathcal{P} = (\overline{\mathcal{W}}, \boldsymbol{i})$ with the extended net and initial marking $\boldsymbol{i}$ is live and bounded.*

Together with Theorem 2.13 from the previous section, it follows that soundness for free choice workflow nets can be decided in polynomial time. In general this is unlikely according to the following theorem.

Theorem 2.34 ([31])**.** *The soundness problem is PSPACE-hard for general workflow nets and PSPACE-complete for bounded workflow nets.*

There exist alternative notions of soundness as introduced in [24] (see also [6]): k-soundness and generalized soundness. We show that for free choice workflow nets they coincide with the standard notion.

Definition 2.35. *Let $\mathcal{W} = (S, T, F, i, o)$ be a workflow net. For every $k \geq 1$, let i^k (o^k) denote the marking that puts k tokens on i (on o), and no tokens elsewhere. $\mathcal{W}$ is* k-sound *if o^k is reachable from every marking reachable from i^k. $\mathcal{W}$ is* generalized sound *if it is k-sound for every $k \geq 1$.*

Theorem 2.36. *Let $\mathcal{W}$ be a free choice workflow net. The following statements are equivalent:*

(1) W is sound

(2) W is k-sound for some $k \geq 1$

(3) W is generalized sound.

Before we prove this theorem, we recall the following theorem from [10].

Theorem 2.37. *Let $\mathcal{W}$ be a sound free choice workflow net, M a reachable marking of $\mathcal{W}$. Then $\overline{\mathcal{W}}$ with the marking M is live iff every S-component of $\overline{\mathcal{W}}$ is marked by M.*

We now prove Theorem 2.36

Proof. (1) $\Rightarrow$ (3). Assume $\mathcal{W}$ is sound which is the same as being 1-sound. Fix $k > 1$, and let $i^k \xrightarrow{\sigma} M$ be an arbitrary occurrence sequence of $\mathcal{W}$. We prove that there exists an occurrence sequence τ such that $M \xrightarrow{\tau} o^k$.

Consider the Petri net $\mathcal{P} = (\overline{\mathcal{W}}, \boldsymbol{i})$ as used in Theorem 2.33. We denote the transition connecting o and i by t^*. Since $\mathcal{W}$ is sound, $\mathcal{P}$ is live by Theorem 2.33. By Commoner's Theorem, adding tokens to a live and bounded marking of a free choice net preserves liveness, therefore the Petri net $(\overline{\mathcal{W}}, i^k)$ with i^k as initial marking is live as well.

We construct the occurrence sequence τ iteratively. Since $(\overline{\mathcal{W}}, i^k)$ is live and M was reached from i^k, it is a live marking. Thus there is a sequence that ends with t^* being fired and that contains t^* only once. We fire this sequence except for the occurrence of t^* which leads to one token being placed on o. We claim that if we remove this token, the resulting marking is still live and we can repeat the above argument until all tokens have reached o without firing t^*.

Indeed, by Theorem 2.37, a marking is live if it marks every S-component. Since all S-components were initially marked by $k > 1$ tokens, removing a token from o will still leave $k - 1 > 0$ tokens in each S-component, thus the marking obtained is live. By repeating the above argument until all tokens are on o, we can construct the sequence τ such that $M \xrightarrow{\tau} o^k$.

(2) $\Rightarrow$ (1). Assume W is k-sound for some $k \geq 1$. By definition of k-soundness, the net $(\mathcal{N}, i^k)$ is bounded and deadlock-free. Theorem 4.31 of [10] states that in bounded and strongly connected free choice nets, deadlock freedom is equivalent to liveness, thus $(\mathcal{N}, i^k)$ is live. By Commoner's Theorem, $(\mathcal{N}, i)$ is also live, and by

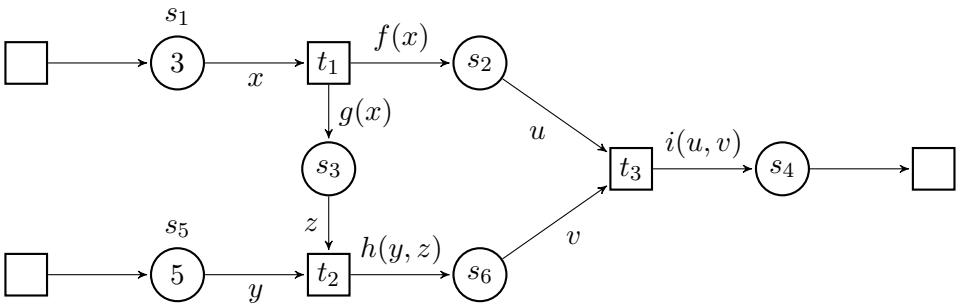


Figure 2.9: Fragment of a workflow net with data

the S-coverability Theorem it is also bounded. Therefore, the workflow net W is sound by Theorem 2.33.

(3) ⇒ (2). Obvious from Definition 2.35.

□

Another direct result from Theorem 2.37 is the following:

Corollary 2.38. *Let $\mathcal{W}$ be a sound free choice workflow net. Then every S-component of $\overline{W}$ contains i and o.*

2.2.3 Colored Workflow Nets

We introduce a variant of workflow nets with data, *colored workflow nets*. In this model, the tokens carry data that is transformed by transitions.

Example 2.11.

In Figure 2.9 a fragment of a workflow net with data is depicted. Assume that the tokens carry data in the form of natural numbers. In our example, s_1 contains a token with value 3 and s_2 contains a token with value 5. Transition t_1 takes a token from s_1. The data carried by the token on s_1 is called x and t_1 produces two tokens, one on s_2 with data $f(x)$ for some function f, and one on s_3 with data $g(x)$ for some function g. For our example, let $f(x) = x^2$ and $g(x) = x + 1$. Then the two tokens produced are one with value 9 on s_2 and one with value 4 on s_3. Thereafter t_2 is enabled and may consume the tokens on s_3 and s_5 with data $z = 4$ and $y = 5$ to produce a token on s_6 with data $h(y, z)$ for some function h.

Formally, the data is reflected in the marking which is now a *colored marking*.

Definition 2.39 (Colored Workflow Net [27])**.** *A* colored workflow net *(CWN) is a tuple $\mathcal{W} = (S, T, F, i, o, V, \lambda)$ where (S, T, F, i, o) is a workflow net, V is a function that assigns to every place $s \in S$ a* color set *C_s and λ is a function that assigns to each transition $t \in T$ a left-total relation $\lambda(t) \subseteq \prod_{s \in {}^\bullet t} C_s \times \prod_{s \in t^\bullet} C_s$ between the values of the input places and those of the output places of t.*

A colored marking *M of $\mathcal{W}$ is a function that assigns to each place s a multiset $M(s)$ over C_s, interpreted as a multiset of* colored tokens *currently on s. A colored marking is* initial *(*final*) if it puts one token on place i (on place o), of any color in C_i (C_o), and no tokens elsewhere.*

Observe that there are now multiple initial and final markings. We call $\lambda(t)$ the *transformer* associated with t. We address elements of a tuple $\lambda \in \lambda(t)$ by λ_s^{in} for an input place s of t, and by λ_s^{out} for an output-place s of t. When a transition t fires, the colored marking M changes to a marking M' in the expected way [27]:

For some color tuple $c \in \prod_{s \in {}^\bullet t} M(s)$, let $\lambda \in \lambda(t)$ such that $c_s = \lambda_s^{in}$ for all $s \in {}^\bullet t$. Then for all places s

$$M'(s) = M(s) - \lambda_s^{in} + \lambda_s^{out}.$$

The addition and subtraction here are the usual operations on the multiset $M(s)$, adding an element and removing one element.

Intuitively, the following happens:

- remove a token from each input place of t;
- choose an element of $\lambda(t)$ whose projection onto the input places matches the tuple of removed tokens;
- add tokens whose colors are the projection of $\lambda(t)$ onto the output places to the output places of t.

Example 2.12.

We again inspect Figure 2.9 with concrete examples. Take $f(x) = 2 \cdot x$, $g(x) = x + 1$ and $h(y, z) = y \cdot z$. Consider the marking $M = (\{3\}, \emptyset, \emptyset, \emptyset, \{2, 4\}, \emptyset)$, which puts a token with data 3 on s_1 and two tokens with data 2 and 4 on s_5. Then the following sequence is possible:

$$(\{3\}, \emptyset, \emptyset, \emptyset, \{2, 4\}, \emptyset) \xrightarrow{t_1} (\emptyset, \{6\}, \{4\}, \emptyset, \{2, 4\}, \emptyset) \xrightarrow{t_2} (\emptyset, \{6\}, \emptyset, \emptyset, \{4\}, \{8\})$$

One interesting problem related to CWNs is to determine the input-output relation or *summary* of the CWN.

Definition 2.40 (Summary)**.** *Let $\mathcal{W}$ be a colored workflow net. Let $\mathcal{M}_i$ and $\mathcal{M}_o$ be the sets of initial and final colored markings of $\mathcal{W}$. The* summary *of $\mathcal{W}$ is the relation $S \subseteq \mathcal{M}_i \times \mathcal{M}_o$ given by: $(M_i, M_o) \in S$ iff M_o is reachable from M_i.*

We define equivalence of CWNs.

Definition 2.41 (Equivalence of CWNs)**.** *Two CWNs $\mathcal{W}_1$ and $\mathcal{W}_2$ are called equivalent, denoted by $\mathcal{W}_1 \equiv \mathcal{W}_2$, if they have the same summary and are either both sound or both unsound.*

In Chapter 3 we will study how to compute the summary via *reduction rules* that partially reduce the net until just one transition remains that contains the whole input output relation.

2.2.4 Probabilistic Workflow Nets

A second extension of workflow nets we will study are *probabilistic workflow nets* (PWNs). Our aim is to decide conflict by assigning probabilities to transitions. However, concurrency should not be resolved by probability, but nondeterministically (or equivalently, by a scheduler). We will see that in general conflict and

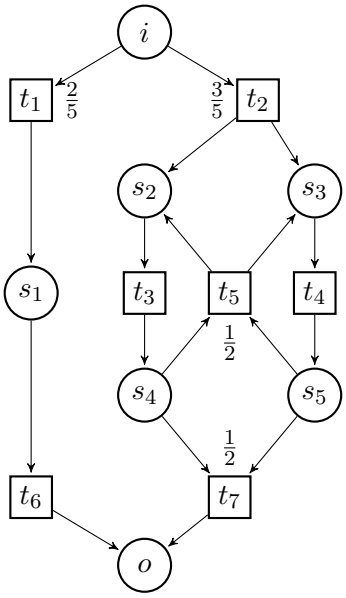


Figure 2.10: A probabilistic workflow net

concurrency may interfere which each other, resulting in the scheduler resolving not only concurrency but also deciding for some transitions whether they can fire at all. We will therefore restrict ourselves to a class of workflow nets where this does not happen.

We first introduce the notion of conflict in a formal setting.

Definition 2.42 (Independent Transitions, Conflict, Conflict Set)**.** *Two transitions t_1, t_2 are* independent *if ${}^\bullet t_1 \cap {}^\bullet t_2 = \emptyset$. Two transitions are* in conflict *at a marking M if M enables both transitions and they are not independent. The set of transitions in conflict with a transition t is called the* conflict set *of t at M.*

The following example gives an intuition how an execution of our probabilistic workflow nets will look like.

Example 2.13.

Consider the workflow net in Figure 2.10. Starting from the initial marking, only one conflict set is enabled which contains t_1 and t_2. The numbers besides those transitions are the probabilities with which they fire and with probability $\frac{3}{5}$ transition t_2 occurs. Thereafter t_3 and t_4 are both enabled concurrently, each is their own conflict set, and one of the two is chosen by the scheduler. Say t_4 fires first. Then only t_3 is enabled and fires. Now the only conflict set enabled contains t_7 and t_5 and the transition that fires is chosen probabilistically between those two.

We now give the definition and semantics of our net. We also add a reward function that assigns a reward to each transition.

Definition 2.43 (Probabilistic Workflow Nets)**.** *A* probabilistic workflow net *(PWN) is a tuple $\mathcal{W} = (S, T, F, i, o, w, r, \oplus)$ where (S, T, F, i, o) is a workflow net, $w : T \to \mathbb{R}_{>0}$ is a* weight function*, $r : T \to \mathbb{R}$ is a* reward function*, and $\oplus$ is a commutative binary operator on $\mathbb{R}$.*

We will use MDPs to give a semantic for PWNs. Remember that an MDP was defined as a tuple $\mathcal{M} = (Q, q_0, Steps)$ where Q is a set of states and $Steps$ assigns to each state a set of probability distributions over Q. One of those probability distributions is chosen nondeterministically by a scheduler and then the step is taken probabilistically. We also defined a reward function for an MDP as function $rew : S \to \mathbb{R}$ together with a commutative binary operator $\oplus$ on $\mathbb{R}$, the binary operator allowed us to accumulate the reward along a path.

As for MDPs, the reward function r for PWNs together with the sum operator $\oplus$ extend to transition sequences in the natural way by summing up over all transitions. In our examples we will use standard addition as well as the maximum operator for $\oplus$.

For PWNs, we would like the scheduler to pick a conflict set from which one transition is chosen probabilistically and fired. Since a PWN is not necessarily free choice, conflict sets for a transition may change depending on the marking (at points where the choice is not free) and we cannot give probabilities for transitions directly. We instead assign weights to the transitions which are then normalized.

Definition 2.44 (Probability distribution)**.** *Let $\mathcal{W} = (S, T, F, i, o, w, r, \oplus)$ be a PWN, let M be a marking of $\mathcal{W}$ enabling at least one transition, and let C be a conflict set enabled at M. The probability distribution $P_{M,C}$ over T is obtained by normalizing the weights of the transitions in C so that they add up to 1, and assigning probability 0 to all other transitions.*

We now define the MDP of a PWN which defines the semantic of the PWN. To represent the state of the PWN in an MDP, it would be sufficient to have one place per reachable marking. As we also want to translate the reward function (which is defined on transitions in the PWN and on states in the MDP), we include the transition used to reach the current marking in the state of the MDP we construct.

Definition 2.45 (MDP of a PWN)**.** *Let $\mathcal{W} = (S, T, F, i, o, w, \oplus)$ be a PWN. The MDP $M_{\mathcal{W}} = (Q, q_0, Steps)$ of $\mathcal{W}$ is defined as follows:*

- *$Q = (\mathcal{M} \times T) \cup \{i, o\}$ where $\mathcal{M}$ are the reachable markings of $\mathcal{W}$, and $q_0 = i$.*
- *For every transition t:*
  - *$Steps((\boldsymbol{o}, t))$ contains exactly one distribution, which assigns probability 1 to state o, and probability 0 to all other states.*
  - *For every marking $M \neq \boldsymbol{o}$ enabling no transitions, $Steps((M, t))$ contains exactly one distribution, which assigns probability 1 to (M, t), and probability 0 to all other states.*
  - *For every marking M enabling at least one transition: $Steps((M, t))$ contains a distribution μ_C for each conflict set C enabled at M. The distribution μ_C is defined as follows.*
    - *For the states i, o: $\mu_C(i) = 0 = \mu_C(o)$.*
    - *For each state (M', t') such that $t' \in C$ and $M \xrightarrow{t'} M'$: $\mu_C((M', t')) = P_{M,C}(t')$.*
    - *For all other states (M', t'): $\mu_C((M', t')) = 0$.*

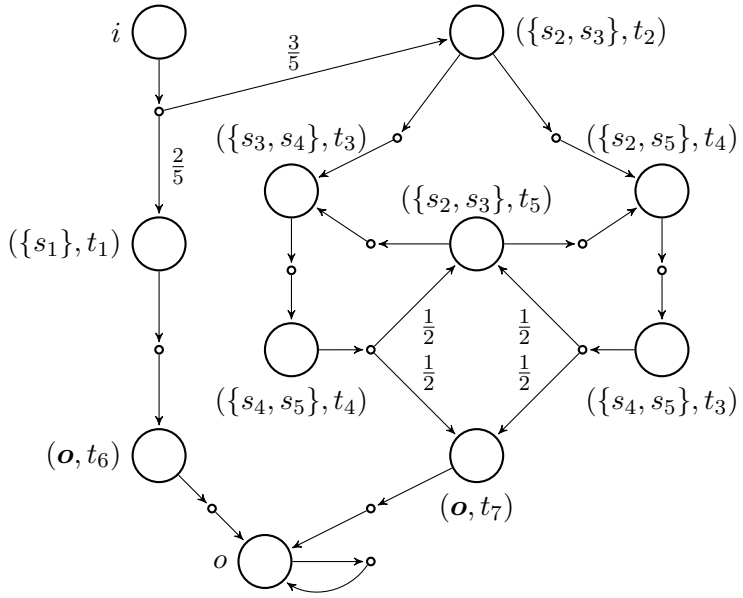


Figure 2.11: The MDP of the PWN in Figure 2.10

- $Steps(i) = Steps((\boldsymbol{i}, t))$ *for any transition* t.
- $Steps(o) = Steps((\boldsymbol{o}, t))$ *for any transition* t.

The reward function $rew_{\mathcal{W}}$ *of* $\mathcal{W}$ *is defined by:* $rew_{\mathcal{W}}(i) = 0 = rew_{\mathcal{W}}(o)$, *and* $rew_{\mathcal{W}}((M, t)) = r(t)$, *the reward operator* $\oplus$ *is used for the MDP as well.*

Example 2.14.

Figure 2.11 shows the MDP of the PWN in Figure 2.10. From the initial marking $\boldsymbol{i}$ there is just one outgoing transition because only one conflict set is enabled. The choice between the transitions t_1 and t_2 is made probabilistically and this is reflected in the MDP. If t_2 is fired, thereafter two transitions t_3 and t_4 are enabled and not in conflict because they do not share an input place. Thus in the MDP the state $(\{s_2, s_3\}, t_2)$ has two outgoing edges, one per transition. Note that o is an absorbing state (i.e. there is a single transition that leads back to o with probability 1) and is in fact the only absorbing state in this MDP.

In general, conflict sets are not necessarily equivalence relations. We will see in the following example that this results in the scheduler having the power to decide which transitions fire, thus resolving not only concurrency. To prevent this, we introduce a class of workflow nets for which conflict sets are an equivalence relation.

Example 2.15.

Consider the net in Figure 2.12. The conflict set of transition t_1 is $\{t_1, t_2\}$, but the conflict set of transition t_2 is $\{t_1, t_2, t_3\}$. By choosing one of those conflict sets, the scheduler already makes a decision whether it is possible for t_3 to fire or not. This contradicts the idea that the scheduler only resolves concurrency.

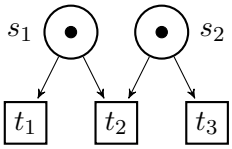


Figure 2.12: A confused marking

It is clear that for free choice nets the conflict relation is an equivalence relation as the conflict set of a transition is exactly its cluster. However, we can define a broader class with a definition similar to free choice but in a semantic way.

Definition 2.46 (Confusion-Free)**.** *A marking M of a workflow net is* confused *if there are two independent transitions t_1, t_2 enabled at M such that $M \xrightarrow{t_1} M'$ and the conflict sets of t_2 at M and at M' are different.*

A workflow net is confusion-free *if no reachable marking is confused.*

Proposition 2.47. *Every free choice workflow net is confusion free.*

Proof. The conflict sets in a free choice nets are exactly the clusters. □

We now work towards showing that the expected reward of PWNs is independent of the scheduler. Our work is based on an unpublished paper by Varacca and Nielsen [42] which we adapted to workflow nets. All definitions, lemmas and theorems that are adaptations of this paper are annotated with [42], the proofs are mostly the same with some changes due to the adaptations or for improved readability.

Lemma 2.48 ([42])**.** *Let W be a 1-safe, confusion-free workflow net. For every reachable marking of W the conflict relation on the transitions enabled at M is an equivalence relation.*

Proof. Consider a workflow net $\mathcal{W}$ and a reachable marking M of $\mathcal{W}$ for which the conflict relation is not an equivalence relation. The conflict relation is always reflexive and symmetric, therefore it must be non-transitive. Then there are three transitions t_1, t_2, t_3 so that t_1 and t_3 are in the conflict set of t_2 but t_1 and t_3 are not in conflict. However firing t_1 disables t_2 and thus modifies the conflict set of t_3 which is independent of t_1. Therefore the marking M of $\mathcal{W}$ is confused and $\mathcal{W}$ cannot be confusion-free. □

In our studies, we will focus on the *expected value* of a PWN under a scheduler. Intuitively, an execution σ of the net has an associated reward $r(\sigma)$. Summing up over all executions permitted by a scheduler, weighted by their respective probability, gives an expected reward. Since the reward function is an arbitrary function to the reals and the operator $\oplus$ can be any commutative binary operator, we can compute many attributes of a PWN that way. For example, setting the reward of each transition to one and using standard addition for $\oplus$ results in $r(\sigma)$ being equal to the number of transitions in σ, which allows us to compute the expected number of transitions fired during an execution. If on the other hand we set the reward of a single transition t to one and all others to zero and use the maximum operator for $\oplus$, then $r(\sigma)$ is one if t occurred and zero otherwise. The expected reward is then the probability that t occurs in an execution of the PWN.

We begin by formally defining the expected value via the MDP and proceed to show that there is an equivalent definition directly on the net.

Definition 2.49 (Expected Reward of a PWN under a Scheduler)**.** *Let $\mathcal{W}$ be a PWN, and let S be a scheduler of its MDP. The expected reward $V^S(\mathcal{W})$ of $\mathcal{W}$ under S is defined as the expected reward $E^S(o)$ to reach the final state o in the MDP $M_{\mathcal{W}}$ of $\mathcal{W}$.*

The remainder of this section has two main targets. First, we show that for free choice PWNs the expected reward can be computed without constructing the MDP by using the firing sequences of the net. This is an important step towards using reduction rules: When arguing about the rules, we can inspect the changes they cause on the net and do not have to concern ourselves with the changes caused to the underlying MDP. Second, we show that the value of a free choice PWN is independent of the scheduler. This is again necessary to argue about the rules as it allows us to freely choose a scheduler of our liking to compute the value.

We now establish correspondences between the MDP and the net itself, beginning with schedulers. In the MDP the MDP-scheduler chooses the probability distribution which determines the next state. By construction, those probability distributions correspond to conflict sets in the net. We give the definition of a *net-scheduler* which chooses the conflict set.

Definition 2.50 (Net-Scheduler)**.** *Let $\mathcal{W}$ be a PWN. A* net-scheduler *of $\mathcal{W}$ is a function that assigns to each finite firing sequence $\boldsymbol{i} \xrightarrow{\sigma} M$ of $\mathcal{W}$ one of the conflict sets of transitions enabled at M, or the empty set if M enables no transitions.,*

The correspondence between net-schedulers and MDP-schedulers is obvious. For a net-scheduler S_1 the corresponding MDP-scheduler S_2 always picks according to the choice C of S_1: S_2 chooses the probability distribution that was obtained from the conflict set C. Because of this simple 1-to-1 correspondence, we will mostly not distinguish between net-schedulers and MDP-schedulers and simply use the term scheduler. Moreover, we will convert from one type of schedulers to the other as necessary without explicit notice.

For a net-scheduler, we can define its *probabilistic language*. This language is a function that assigns each finite firing sequence a probability. Intuitively, this is the probability that this firing sequence will happen. More formally, it is the probability of the cylinder of all paths that "follow" that firing sequence.

Definition 2.51 (Probabilistic Language of a Scheduler [42])**.** *The* probabilistic language *ν_S of a scheduler S is the function $\nu_S \colon T^* \to \mathbb{R}_{\geq 0}$ defined by $\nu_S(\sigma) = Prob^S(cyl^S(\pi_\sigma))$. A transition sequence σ is* produced *by S if $\nu_S(\sigma) > 0$.*

Example 2.16.

We return to our example PWN again depicted in Figure 2.13. After t_2 fires, t_3 and t_4 are concurrently enabled and not in conflict. A very simple scheduler might always pick $\{t_3\}$ as conflict set and thus fire t_3 before t_4.

Its probabilistic language ν_S then assigns the following values: $\nu(t_2) = \frac{3}{5}$, $\nu(t_2t_3) = \frac{3}{5}$, $\nu(t_2t_4) = 0$.

Observe that for any transition sequence σ that is not a firing sequence, any scheduler S will have $\nu_S(\sigma) = 0$.

Using the definition of ν_S we are now able to compute the expected value $V^S(\mathcal{W})$ using transition sequences instead of paths in the MDP. Remember that $Fin_{\mathcal{W}}$ are

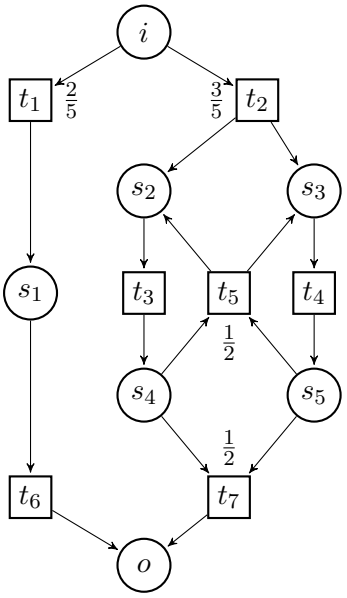


Figure 2.13: A probabilistic workflow net

the firing sequences in $\mathcal{W}$ ending in the final marking, r is the reward function of $\mathcal{W}$ and for an MDP we defined $R(F, \pi)$ as the reward along a path π until a set of states F is reached. We write $R(q, \pi)$ instead of $R(\{q\}, \pi)$ for a set F that only consists of a single state q.

Lemma 2.52. *Let $\mathcal{W}$ be a 1-safe confusion-free PWN, and let S be a scheduler. If $\mathcal{W}$ is sound, $V^S(\mathcal{W})$ is finite and*

$$V^S(\mathcal{W}) = \sum_{\pi \in \Pi} R(o, \pi) \cdot Prob^S(cyl^S(\pi)) = \sum_{\sigma \in Fin_{\mathcal{W}}} r(\sigma) \cdot \nu_S(\sigma)$$

where Π are the paths in the MDP $M_{\mathcal{W}}$ leading from the initial state i to the final state o.
If $\mathcal{W}$ is unsound, either $V^S(\mathcal{W})$ is finite and the above equation holds or $V^S(\mathcal{W})$ is infinite and so is $V^R(\mathcal{W})$ for any other scheduler R.

Proof. We repeat the original definition: $V^S(\mathcal{W}) = E^S(o) = \int_{\pi \in Paths^S} R(o, \pi) \mathrm{d}Prob^S$.

We begin with the sound case.

We first show that for some number k the probability to reach the final marking from any given marking in at most k steps can be bounded away from zero.

Since $\mathcal{W}$ is sound, the final marking is reachable from every marking. Furthermore, the reachability graph is finite because $\mathcal{W}$ is bounded. Let k be the size of the reachability graph. Let M be some marking, σ a shortest sequence to reach the final marking from M. Observe that the length of σ can be at most k.

We let the scheduler pick some conflict set C. We claim that in σ, some transition in C must occur. Let t' be some transition in C. Indeed, since t' is enabled and $\mathcal{W}$ is confusion-free, the conflict set C of t' may not change until some transition in C is fired. Since after σ the final marking is reached, this has to happen at some point.

Let t be the first transition of C that appears in σ. Then all transitions before t in σ must be independent of t and t must already be enabled at M. Let M' be the marking such that $M \xrightarrow{t} M'$.

We remove the first occurrence of t from σ to obtain σ' and claim that $M' \xrightarrow{\sigma'} \boldsymbol{o}$. This is obvious since we only switched the order of independent transitions.

In this way we modify the order of the transitions in σ to conform with the choices of the scheduler and obtain a transition sequence where the conflict sets were chosen by the scheduler that ends in the final marking. Since the weights are all positive, this sequence occurs with a positive probability.

So far we have shown that the set of reachable markings is finite and for each of those markings, there is a transition sequence that leads to the final marking and occurs with positive probability. It follows that the probability to eventually reach the final marking is equal to 1. Recall that Π are the paths in the MDP $M_{\mathcal{W}}$ leading from the initial state i to the final state o as defined in the statement of this lemma.

Since the probability to eventually reach the final marking is equal to 1, it holds that

$$\int_{\pi \in Paths^S} R(o,\pi) \mathrm{d}Prob^S = \int_{\pi \in cyl^S(\Pi)} R(o,\pi) \mathrm{d}Prob^S.$$

Furthermore, for a path $\pi \in \Pi$, it holds that $R(o,\pi) = R(o,\pi')$ for all $\pi' \in cyl^S(\pi)$ because $last(\pi) = o$. We obtain

$$\int_{\pi \in cyl^S(\Pi)} R(o,\pi) \mathrm{d}Prob^S = \sum_{\pi \in \Pi} R(o,\pi) \cdot Prob^S(cyl^S(\pi))$$

and therefore the first equality. Together with $r(\sigma) = R(o,\pi_\sigma)$, the obvious correspondence between paths in Π and sequences in Fin and the definition of ν_S, the second equality follows.

We now focus on the unsound case.

Let $\mathcal{W}$ be an unsound PWN. Then $\overline{\mathcal{W}}$ is either unbounded or not live by Theorem 2.33. Since per the lemma statement $\mathcal{W}$ is 1-safe, $\overline{\mathcal{W}}$ is as well and thus $\overline{\mathcal{W}}$ must be non-live. This means there is a marking M reachable from the initial marking such that some transition t can never occur in any transition sequence starting from M. We distinguish two cases.

(1) $\boldsymbol{i}$ is not such a marking M, i.e. every transition occurs in some occurrence sequence starting in the initial marking. Then it is clear that from M the initial marking cannot be reachable, and therefore the same holds for the final marking. Let σ be a sequence that ends in M starting from the initial marking. Using a similar reasoning as in the sound case, we can show that there is a sequence τ that is Mazurkiewicz equivalent to $\sigma\sigma'$ for some σ' and that occurs with positive probability under S. We show this by rearranging the transitions in $\sigma\sigma'$ so that they are in the order that S chooses. τ furthermore has infinite reward because no sequence starting with τ ever reaches the final marking, thus the PWN has an infinite expected value.

(2) $M = i$. That means some transition t does not occur in any occurrence sequence starting from the initial marking. By definition of the MDP of $\mathcal{W}$ and the reward of an MDP, if t never occurs in any occurrence sequence it has no effect on the expected reward. If we remove all such transitions t and the net is still unsound, the expected reward is infinite by (1). If the net is sound after the removal, we can use the argumentation for the sound case to show that the equalities hold.

Note that the argument made in (1) holds independently of the scheduler S, i.e. if the value is infinite for some scheduler S, it is infinite for every scheduler. □

We now turn to the task of proving that the expected reward is independent of the scheduler. First, we extend the notion of schedulers to *partial schedulers*

Definition 2.53 (Partial Scheduler)**.** *A* partial scheduler *of length n is the restriction of a scheduler to firing sequences of length less than n. Given two partial schedulers R, S on lengths n_R, n_S, we say that R* extends *S if $n_R \geq n_S$ and S is the restriction of R to firing sequences of length less than n_S.*

We extend the notion of Mazurkiewicz equivalence to schedulers. As the notion of independent transitions is symmetric, the transitions form an independence alphabet and therefore Mazurkiewicz equivalence on transition sequences is defined by Definition 1.6 and will be denoted by $\equiv$.

Definition 2.54 (Mazurkiewicz Equivalence of Partial Schedulers)**.** *Given a partial scheduler S of length n, we denote by F_S the set of firing sequences σ of $\mathcal{W}$ produced by S such that either $|\sigma| = n$ or σ leads to a marking that enables no transitions.*

Two partial schedulers R, S with probabilistic languages ν_R, ν_S are Mazurkiewicz equivalent[2]*, denoted by $R \equiv S$, if they have the same length and there is a bijection $\phi\colon F_R \to F_S$ such that $\sigma \equiv \phi(\sigma)$ and $\nu_R(\sigma) = \nu_S(\phi(\sigma))$ for every $\sigma \in F_n$.*

Example 2.17.

For our running example in Figure 2.13 there are two partial schedulers of length 2, the one that chooses t_3 after t_2 and the one that chooses t_4:

$$\begin{array}{lllll} R\colon & \epsilon \mapsto \{t_1, t_2\} & t_1 \mapsto \{t_6\} & t_2 \mapsto \{t_3\} \\ S\colon & \epsilon \mapsto \{t_1, t_2\} & t_1 \mapsto \{t_6\} & t_2 \mapsto \{t_4\} \end{array}$$

Those schedulers are not Mazurkiewicz equivalent since R produces the sequences $F_R = \{t_1t_6, t_2t_3\}$ while S produces the sequences $F_S = \{t_1t_6, t_2t_4\}$ and no bijection ϕ can satisfy $\sigma \equiv \phi(\sigma)$ for every $\sigma \in F_R$.

We now present the main result of [42] in our own terminology.

Theorem 2.55 (Equivalent extension of schedulers [42][3])**.** *Let R, S be two partial schedulers of a confusion-free PWN $\mathcal{W}$. There exist two partial schedulers R', S' of $\mathcal{W}$ such that R' extends R, S' extends S and $R' \equiv S'$.*

Example 2.18.

We can extend the schedulers R and S from the above example as follows:

$$\begin{array}{lllll} R'\colon & \ldots & t_1t_6 \mapsto \emptyset & t_2t_3 \mapsto t_4 \\ S'\colon & \ldots & t_1t_6 \mapsto \emptyset & t_2t_4 \mapsto t_3 \end{array}$$

Now we have $F_{R'} = \{t_1t_6, t_2t_3t_4\}$ and $F_{S'} = \{t_1t_6, t_2t_4t_3\}$. We choose the obvious bijection, which works because we have $t_2t_3t_4 \equiv t_2t_4t_3$ and $\nu_{R'}(t_2t_3t_4) = 3/5 = \nu_S(t_2t_4t_3)$.

[2] In [42], this is called strongly Mazurkiewicz equivalent

[3] Stated as Theorem 2, the original paper gives this theorem with R' and S' being (non-partial) schedulers. However, in the paper equivalence is only defined for partial schedulers and the schedulers constructed in the proof are also partial.

We prove Theorem 2.55 by induction over the length of the partial schedulers. Trivially, for two partial schedulers of length 0 the theorem holds. For the induction step, we need an intermediate result.

Lemma 2.56 ([42])**.** *Let R, S be two partial schedulers that are equivalent. Then for every partial scheduler R' that extends R there is a partial scheduler S' that extends S such that $R' \equiv S'$.*

Proof. Let R, S be two equivalent partial schedulers of length n. Clearly it is enough to show the lemma for extensions of length $n+1$. Let R' be such an extension of R. We construct an extension S' of S (also of length $n+1$) that is equivalent to R'.

Let $\phi : F_R \to F_S$ be a witness of $R \equiv S$ as in 2.54. Remember that F_R are the sequences σ produced by R which are either of length n or lead to a marking that enables no transition. We build a partial scheduler S' as follows:

$$S'(\sigma) = \begin{cases} S(\sigma) & \text{if } |\sigma| < n \\ R'(\phi^{-1}(\sigma)) & \text{if } |\sigma| = n \text{ and } \nu_S(\sigma) > 0 \\ \text{arbitrary} & \text{otherwise} \end{cases}$$

We first check that S' is a well defined scheduler that extends S. It is clear that the restriction of S' to sequences of length less than n is S. We furthermore have to show that $R'(\phi^{-1}(\sigma))$ is a conflict set enabled after σ. As $\sigma \equiv \phi^{-1}(\sigma)$ and equivalent transition sequences fire the same transitions and thus lead to the same marking, the conflict sets are the same and therefore $R'(\phi^{-1}(\sigma))$ is a conflict set enabled after σ.

To prove equivalence, we define $\phi' : F_{R'} \to F_{S'}$ as follows:

$$\phi'(\sigma) = \begin{cases} \phi(\sigma) & \text{if } |\sigma| <= n \\ \phi(\sigma')t & \text{if } \sigma = \sigma' t \text{ with } |\sigma'| = n \end{cases}$$

The function ϕ' is injective as ϕ is a bijection by definition, and it is also surjective due to our construction of S. The corresponding sequences are equivalent by construction. As for the probabilities:

$$\nu_{S'}(\sigma' t) = Prob^S(cyl(\sigma' t)) = Prob^S(cyl(\sigma')) \cdot P_{[t]} t =$$

$$Prob^R(cyl(\phi^{-1}(\sigma'))) \cdot P_{[t]} = \nu_{R'}(\phi^{-1}(\sigma')t) = \nu_{R'}(\phi'^{-1}(\sigma' t))$$

□

To prove Theorem 2.55 by induction, let R and S be two schedulers with $|R| = n$ and $|S| = m$. By induction hypothesis there exist equivalent schedulers R' and S' extending R and S, respectively. Let R_{n+1} be a scheduler extending R with $|R_{n+1}| = n+1$ We prove that there are two equivalent schedulers extending R_{n+1} and S, respectively.

Let k be the length of R' and S'. Our idea is depicted in Figure 2.14. We start by building two equivalent schedulers R'' and R_{k+1} that extend R' and R_{n+1}, respectively. Then we apply Lemma 2.56. Since R'' extends R' and R' is equivalent to S', we get a scheduler S'' which is equivalent to R'' (and thus also to R_{k+1}) and is also an extension to S' and thus to S. Therefore R_{k+1} and S'' are the schedulers we are looking for.

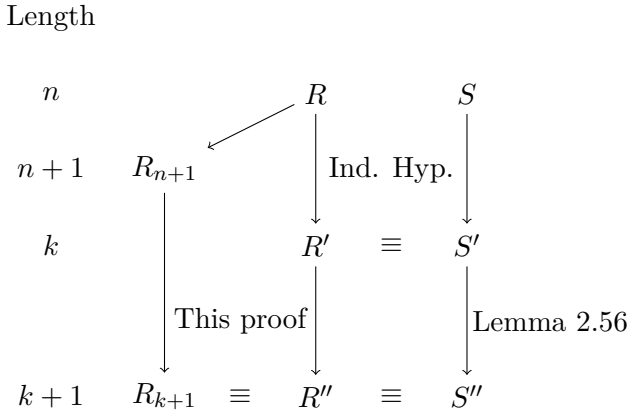


Figure 2.14: Illustration of the proof of Theorem 2.55, edges are extensions

As a first step we build R'' that extends R'. Intuitively, we need to replicate the additional behavior of R_{n+1} if it has not already happened. This behavior consisted of one conflict set chosen in the $n+1$-th step.

Let k be the length of R' and S'. Let $\sigma\tau$ be a sequence of length k with $|\sigma| = n$ produced by R'.

$$R''(\sigma\tau) = \begin{cases} \emptyset & \text{if } \sigma\tau \text{ enables no transitions} \\ R_{n+1}(\sigma) & \text{if there is no } i \in 1, \ldots, |\tau| \text{ with } R'(\sigma\tau^i) = R_{n+1}(\sigma) \\ \text{any enabled} & \\ \text{conflict set} & \text{otherwise} \end{cases}$$

Since the net is confusion free, if the conflict set $R_{n+1}(\sigma)$ was never picked by R', then it must still be enabled after $\sigma\tau$ which is why we can define R'' as above.

As a second step, we build R_{k+1} by extending R_{n+1} step by step, construction intermediate schedulers $R_{n+2}, \ldots, R_k$ along the way. We intuitively follow the actions of R' until R' chooses the conflict set that R_{n+1} chose in the $n+1$-th step. Since R_{n+1} already chose that conflict set, we skip it and continue to follow the actions of R'. If we never find such a point, we choose that conflict set in the $k+1$-th step.

While we define the extensions R_{n+i} of R_{n+1} we also define an additional function sfp_{n+i} (for shift point) which keeps track of when (if at all) R' behaved like R_{n+1}. This function can produce two kinds of values:

- an integer h, if position h is the first point up to $n+i$ when R' behaved like R_{n+1}
- $\perp$ if R' has not behaved like R_{n+1} until position $n+i$.

We need the following function on sequences:

$$shift_{i,j}(t_1 \ldots t_i \ldots t_j \ldots t_n) = t_1 \ldots t_{i-1} t_{i+1} \ldots t_{j-1} t_j t_i t_{j+1} \ldots t_n$$

We also define $shift_{i,i}$ as the identity. Notice that if t_i is independent of $t_{i+1} \ldots t_j$ then $shift_{i,j}(t_1 \ldots t_n) \equiv t_1 \ldots t_n$.

Let σt_{n+1} be produced by R_{n+1} with $|\sigma| = n$. In the following, we construct the schedulers $R_{n+2}, \ldots, R_{k+1}$ where each R_{i+1} will be an extension of R_i for $n+2 \leq i \leq k+1$. We first define sfp_{n+2} and R_{n+2}:

$$sfp_{n+2}(\sigma t_{n+1}) = \begin{cases} n+1 & \text{if } t_{n+1} \in R'(\sigma) \\ \bot & \text{otherwise} \end{cases}$$

$$R_{n+2}(\sigma t_{n+1}) = \begin{cases} R'(\sigma t_{n+1}) & \text{if } sfp_{n+2}(\sigma t_{n+1}) = n+1 \\ R'(\sigma) & \text{otherwise} \end{cases}$$

For $2 \leq i \leq k-n$ and a sequence $\sigma t_{n+1} \dots t_{n+i}$ produced by R_{n+i} we define

$$sfp_{n+i+1}(\sigma t_{n+1} \dots t_{n+i}) = \begin{cases} h & \text{if } sfp_{n+i}(\sigma t_{n+1} \dots t_{n+i-1}) = h \\ n+i & \text{if } \begin{cases} sfp_{n+i}(\sigma t_{n+1} \dots t_{n+i-1}) = \bot & \text{and} \\ t_{n+i} \in R'(\sigma) \end{cases} \\ \bot & \text{if } \begin{cases} sfp_{n+i}(\sigma t_{n+1} \dots t_{n+i-1}) = \bot & \text{and} \\ t_{n+i} \notin R'(\sigma) \end{cases} \end{cases}$$

Then for $2 \leq i < k-n$:

$$R_{n+i+1}(\sigma t_{n+1} \dots t_{n+i}) = \begin{cases} R'(shift_{n+1,h}(\sigma t_{n+1} \dots t_{n+i})) & \text{if } sfp_{n+i+1}(\sigma t_{n+1} \dots t_{n+i}) = h \in \mathbb{N} \\ R'(\sigma t_{n+2} \dots t_{n+i}) & \text{otherwise} \end{cases}$$

And for $i = k-n$:

$$R_{k+1}(\sigma t_{n+1} \dots t_k) = \begin{cases} R'(shift_{n+1,h}(\sigma t_{n+1} \dots t_k)) & \text{if } sfp_{k+1}(\sigma t_{n+1} \dots t_k) = h \in \mathbb{N} \\ R'(\sigma t_{n+2} \dots t_{k-1}) & \text{otherwise} \end{cases}$$

We show that each R_{n+i} is a well-defined scheduler, i.e. it picks a conflict set that is enabled. This follows from confusion freeness and the definitions of *sfp*:

First assume that $sfp_{n+i+1}(\sigma t_{n+1} \dots t_{n+i}) = \bot$. Then R' has not picked the conflict set $R_{n+1}(\sigma)$ yet, and thus a) this conflict set is still enabled and b) all transitions after σ were independent of the transitions in that conflict set. That means that we can keep following R' for another step.

Now assume that $sfp_{n+i+1}(\sigma t_{n+1} \dots t_{n+i}) = h$. Then R' has picked the conflict set $R_{n+1}(\sigma)$ in position h. Before that, only conflict sets independent of this set have been picked, thus the sequence $shift_{n+1,h}(\sigma t_{n+1} \dots t_k)$ is a sequence produced by R' and we follow R' from there.

It is easy to see that R'' and R_{k+1} will produce equivalent sequences with the same probabilities: In the construction of R_{k+1} we have mimicked every choice of R'' with the only change that the set $R_{n+1}(\sigma)$ has been picked earlier, but it has been swapped only with independent conflict sets. Since the net is confusion-free this is possible without influencing the probabilities that a specific sequence (or an equivalent one) arises. This concludes our proof of Theorem 2.55. □

We need one more observation about Mazurkiewicz equivalence before we arrive at our main theorem. Recall that PWNs have an associated reward function r that extends to transition sequences via the commutative reward operator $\oplus$.

Proposition 2.57. *Let $\mathcal{W}$ be a confusion-free PWN. Then for any two firing sequences σ and τ that are Mazurkiewicz equivalent, it holds that $r(\sigma) = r(\tau)$.*

This follows by the definition of Mazurkiewicz equivalence and the commutativity of the reward operator. We can now show one of our main results.

Theorem 2.58. *Let $\mathcal{W}$ be a 1-safe confusion-free PWN. Then the expected reward $V^S(\mathcal{W})$ does not depend on the scheduler S.*

Proof. Pick any two schedulers R, S. We show that there is a bijection between Mazurkiewicz equivalent firing sequences that end in the final marking and that are produced by those schedulers.

By Theorem 2.55, any two partial schedulers can be extended to two equivalent partial schedulers, in particular the partial schedulers R^k, S^k that are the restrictions of R and S to firing sequences of length less than k.

Let R' be a partial scheduler extending R^k, S' a partial scheduler extending S^k such that $R' \equiv S'$. Let σ be a firing sequence of length k produced by R that ends in the final marking. By definition of equivalence, there is a firing sequence τ such that $\sigma \equiv \tau$ and $\nu_{R'}(\sigma) = \nu_{S'}(\tau)$. Since σ and τ are Mazurkiewicz equivalent, τ also ends in the final marking and also has length k. Since σ is of length k, it was already produced by R^k and thus by R, and τ was already produced by S.

Repeating this for every k, we can construct a bijection ϕ that maps every firing sequence σ produced by R that ends in the final marking to a Mazurkiewicz equivalent firing sequence $\phi(\sigma)$ of the same length produced by S that ends in the final marking.

Using Proposition 2.57, we know that $r(\sigma) = r(\phi(\sigma))$.

We use Lemma 2.52. In the case that for some scheduler the value of $\mathcal{W}$ is infinite, this lemma already states that the value is infinite for all schedulers. Otherwise, we get:

$$V^R(\mathcal{W}) = \sum_{\sigma \in \Sigma} r(\sigma) \cdot \nu_R(\sigma) = \sum_{\sigma \in \Sigma} r(\phi(\sigma)) \cdot \nu_S(\phi(\sigma)) = \sum_{\sigma \in \Sigma} r(\sigma) \cdot \nu_S(\sigma) = V^S(\mathcal{W})$$

where the third equality is just a reordering of the sum. □

Example 2.19.

We continue with our running example. For convenience we have repeated the PWN in Figure 2.15. We have picked two partial schedulers R and S and extended them to two equivalent partial schedulers R' and S' by setting

$$\begin{array}{lllllll} R': & \epsilon \mapsto \{t_1, t_2\} & t_1 \mapsto \{t_6\} & t_2 \mapsto \{t_3\} & t_1 t_6 \mapsto \emptyset & t_2 t_3 \mapsto t_4 \\ S': & \epsilon \mapsto \{t_1, t_2\} & t_1 \mapsto \{t_6\} & t_2 \mapsto \{t_4\} & t_1 t_6 \mapsto \emptyset & t_2 t_4 \mapsto t_3 \end{array}$$

We extend those partial schedulers to (non-partial) schedulers R'' and S'' in the following way: R'' will continue to pick $\{t_3\}$ first if t_3 and t_4 are enabled, S'' will continue to pick $\{t_4\}$ first.

The sequences produced by R'' have the form $t_1 t_6$ or $t_2 t_3 t_4 (t_5 t_3 t_4)^* t_7$. We compute $\nu_{R''}$.

$$\begin{aligned} \nu_{R''}(t_1 t_6) &= \frac{2}{5} \\ \nu_{R''}(t_2 t_3 t_4 (t_5 t_3 t_4)^k t_7) &= \frac{3}{5} \cdot \frac{1}{2^k} \cdot \frac{1}{2} \end{aligned}$$

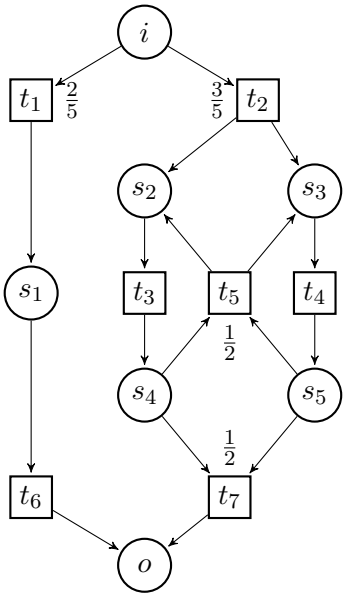


Figure 2.15: A probabilistic workflow net

If we set all rewards to 1, the reward $r(\sigma)$ for a single transition sequence σ is equal to its length. We compute the expected reward $V^{R''}$ under R''.

$$V^{R''} = \sum_{\sigma \in Fin_{\mathcal{W}}} r(\sigma) \cdot \nu_{R''}(\sigma) = \frac{2}{5} \cdot 2 + \sum_{k=0}^{\infty} \frac{3}{5} \cdot \frac{1}{2^k} \cdot \frac{1}{2} \cdot (3 \cdot k + 4) = \frac{113}{10}$$

We do the same for S'' which will produce sequences of the form $t_1 t_6$ or $t_2 t_4 t_3 (t_5 t_4 t_3)^* t_7$.

$$\begin{aligned} \nu_{S''}(t_1 t_6) = & \quad \frac{2}{5} \\ \nu_{S''}(t_2 t_4 t_3 (t_5 t_4 t_3)^k t_7) = & \quad \frac{3}{5} \cdot \frac{1}{2^k} \cdot \frac{1}{2} \end{aligned}$$

It is obvious that the expected value will the same as there is a one-to-one correspondence of equal length sequences with equal probability.

Theorem 2.58 thus allows us to speak of *the* expected reward of a confusion-free PWN and also enables us to define equivalence on confusion-free PWNs.

Definition 2.59 (Equivalence of PWNs)**.** *Two 1-safe confusion-free PWNs $\mathcal{W}_1$ and $\mathcal{W}_2$ are called equivalent, denoted by $\mathcal{W}_1 \equiv \mathcal{W}_2$, if they have the same expected value and are either both sound or both unsound.*

To see that Theorem 2.58 no longer holds on nets that are not confusion-free, consider the following example.

Example 2.20.

Consider the confused workflow net shown in Figure 2.16. After t_1 fires, all three remaining transitions are enabled. However, there are three different conflict sets: The conflict set of t_2 is $\{t_2, t_3\}$, the conflict set of t_3 is $\{t_2, t_3, t_4\}$

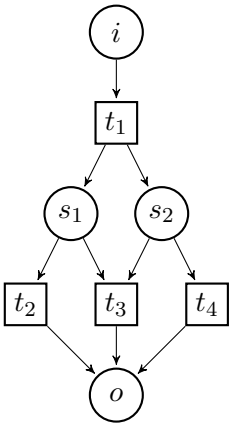


Figure 2.16: A confused workflow net

and the conflict set of t_4 is $\{t_3, t_4\}$. After the initial choice of a conflict set, there is always one or no conflict set enabled. Consequently, there are three possible schedulers, one per conflict set.

We set all rewards to 1 and all weights to 1. We now compute the expected value for the scheduler R that always chooses the conflict set of t_2 and the expected value for the scheduler S that always chooses the conflict set of t_3.

After R chooses $\{t_2, t_3\}$, both transitions appear with probability $\frac{1}{2}$. Thus the transition sequence $\sigma = t_1 t_3$ has $\nu(\sigma) = \frac{1}{2}$ and $r(\sigma) = 2$, the transition sequence $\tau = t_1 t_2 t_4$ has $\nu(\tau) = \frac{1}{2}$ and $r(\tau) = 3$ and in total we get an expected value $V^R(\mathcal{W}) = \frac{5}{2}$.

After S chooses $\{t_2, t_3, t_4\}$, all three transitions appear with probability $\frac{1}{3}$. Thus the transition sequences $t_1 t_3$, $t_1 t_2 t_4$ and $t_1 t_4 t_2$ are all equally likely, their rewards are equal to their lengths and we get an expected value $V^S(\mathcal{W}) = \frac{8}{3}$.

Chapter 3

Rule-based Reduction

Contents

3.1 Reduction Rules

In the previous chapter we have defined soundness of workflow nets, the summary of a colored workflow net and the expected value of a probabilistic workflow net. We now focus on a reduction procedure that reduces sound workflow nets to a trivial net. By extending the reduction rules, it will also be possible to compute the summary or expected value of a CWN or PWN, respectively, and we will do so towards the end of this chapter.

For a first idea, we look at the labeled transition graph of a given workflow net. This graph is defined as the (uncolored) marking graph, but labeling the edges corresponding to a transition $M \xrightarrow{t} M'$ with t. As there may be multiple transitions which connect the same two markings, this graph is actually a multigraph. Figure 3.1 shows an example workflow net and its labeled transition graph.

For every bounded workflow net the labeled transition graph is finite and we can apply a reduction procedure similar the procedure of transforming finite automata to regular expressions [26]. However, the labeled transition graph may be exponential in the size of the workflow net and thus the algorithm will have exponential running time. In this section, we present an idea that helps to overcome this state-space explosion: We create a set of rules that is inspired by the rules for labeled transition graphs, but our rules work directly on the workflow net and therefore avoid the exponential blowup.

This chapter has been published in parts in [18] and [19]. It is based on earlier work on negotiations [16, 17, 11] where these reduction rules were defined and the algorithm has been introduced and proven to be polynomial. Negotiations are a model close to workflow nets, but instead of tokens the moving parts are called agents and can be thought of as tokens with an identity. It is thus possible to follow an agent as it moves through the net. The proofs presented here have been adapted to workflow nets where "following a token" is closely related to the S-components of a net. In particular, the shortcut and merge rule have been introduced in [16] and have been shown to be complete for acyclic negotiations (together with an additional useless-arc rule which is not important for our rule on workflow nets). The proofs of Theorems 3.9 and 3.11 are translations of these proofs, the main difference is the addition of S-components which replaces the concept of agents for negotiations. Cyclic negotiations have been subject of [17], but the algorithm presented there is unfortunately wrong and has been corrected in [11]. The adaptation to workflow nets [18] concludes with Theorem 3.29 and is once again more involved because, intuitively, it is not easily possible to "follow a token" in a workflow net and we need to resort to S-components for the definition of synchronizers.

3.1.1 Inspiration: Finite Automata

We use the standard notation where $|$ denotes the choice operator and * denotes the Kleene star operator.

The reduction procedure for finite automata based on state elimination proceeds by iteratively applying one of the following three rules:

(1) Replace two distinct edges $M \xrightarrow{t_1} M'$ and $M \xrightarrow{t_2} M'$ by a new edge $M \xrightarrow{t_1|t_2} M'$.

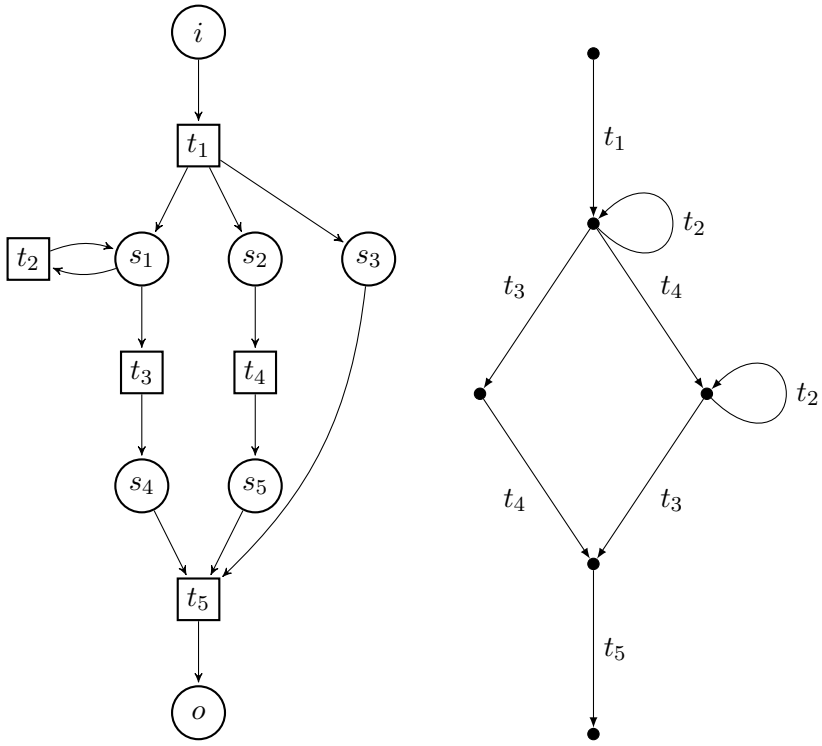


Figure 3.1: A workflow net and its labeled transition graph

(2) Given a self-loop $M \xrightarrow{t} M$, replace every edge $M \xrightarrow{t'} M'$ by an edge $M \xrightarrow{t^*t'}$ and then remove the self-loop.

(3) Given an edge $M \xrightarrow{t} M'$ such that $M \neq M'$ and M' has at least one successor, add for every edge $M' \xrightarrow{t'} M''$ a *shortcut edge* $M \xrightarrow{tt'} M''$, then remove the edge $M \xrightarrow{t} M'$. Furthermore, if M' has no more incoming edges, remove M' and all its outgoing edges.

The algorithm applies these rules in a certain order until no rule can be applied any more:

- Apply rule (1) as long as possible.
- Apply rule (2) as long as possible.
- When neither rule (1) nor rule (2) are applicable, choose a marking M that has both incoming and outgoing edges and apply rule (3) to all its incoming edges. Observe that this leads to the removal of M.

Every time we begin applying rule (3), we will keep applying this rule until we remove a node in the graph. We therefore reduce the number of nodes until this is no longer possible, that is, until every node either has no incoming or no outgoing edges.

We now explore the effects of this algorithm on a labeled transition graph of a workflow net. The only node without incoming edges is by definition the initial marking. If the workflow net was sound, only the final marking has no outgoing

edges and the resulting graph will have two nodes, the initial and the final marking, connected by a single edge.

For a sound workflow net, every node in the labeled transition graph lies on a path from the node $\boldsymbol{i}$ to the node $\boldsymbol{o}$ and after the algorithm is finished, only these two nodes will remain, connected by a single edge. In the unsound case, if from some marking the final marking is not reachable, we will end up with additional nodes. If the workflow net is unsound because some transition can never be fired, we will end up with two nodes and a single edge as in the sound case, but the transition in question will not appear in the regular expression.

Figure 3.2 shows the intermediate steps in the reduction of the graph shown in Figure 3.1. As the workflow net from which we obtained the labeled transition graph is sound, the reduction ends with a single transition remaining.

3.1.2 Reduction Rules for Workflow Nets

As mentioned earlier, the labeled transition graph of a workflow net may be exponential in the size of the workflow net, thus we want to work directly on the net instead. We begin with a formal definition of reduction rules and the terms *correctness* and *completeness*.

Definition 3.1 (Rules, Correctness, and Completeness)**.** *A* reduction rule*, or just a rule, is a binary relation on the set of workflow nets. Given a rule R, we write $\mathcal{W}_1 \xrightarrow{R} \mathcal{W}_2$ for $(\mathcal{W}_1, \mathcal{W}_2) \in R$.*

A rule R is correct *if its application to a workflow net preserves soundness, that is $\mathcal{W}_1 \xrightarrow{R} \mathcal{W}_2$ implies that $\mathcal{W}_1$ is sound iff $\mathcal{W}_2$ is sound.*

Given a correct set of rules $\mathcal{R} = \{R_1, \ldots, R_n\}$, we denote by $\mathcal{R}^*$ the reflexive and transitive closure of $R_1 \cup \ldots \cup R_n$. We say that $\mathcal{R}$ is complete with respect to a class of workflow nets *if for every sound workflow net $\mathcal{W}$ in that class there is a workflow net $\mathcal{W}'$ consisting of two places and a single transition between them such that $\mathcal{W} \xrightarrow{\mathcal{R}^*} \mathcal{W}'$.*

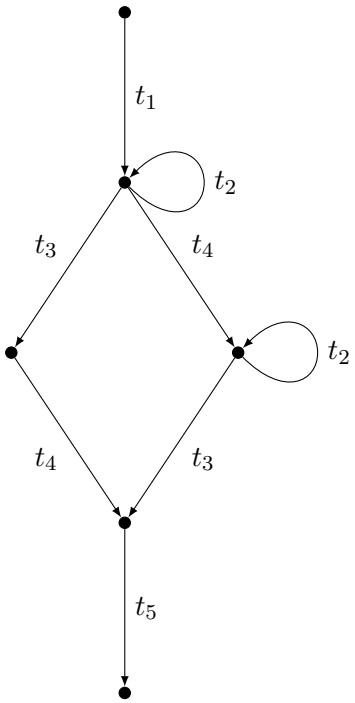
The *trivial net* consisting only of the places i and o and a single transition between them is always a sound workflow net. Thus for a correct set of rules, only the sound workflow nets can be completely reduced to a trivial net as correctness implies that the rules preserve soundness. We describe our rules as pairs of a *guard* and an *action*: $\mathcal{W}_1 \xrightarrow{R} \mathcal{W}_2$ holds if $\mathcal{W}_1$ satisfies the guard and $\mathcal{W}_2$ is a possible result of applying R to $\mathcal{W}_1$.

3.1.3 Merge Rule

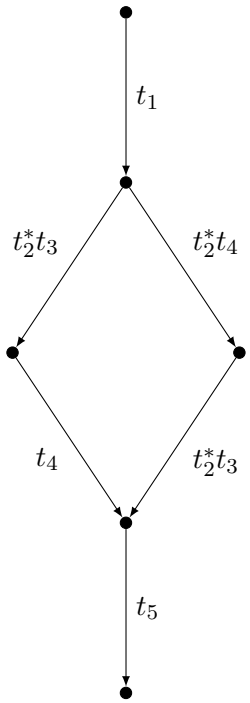
Rule (1) for the labeled transition graph was the following:

(1) Replace two distinct edges $M \xrightarrow{t_1} M'$ and $M \xrightarrow{t_2} M'$ by a new edge $M \xrightarrow{t_1|t_2} M'$.

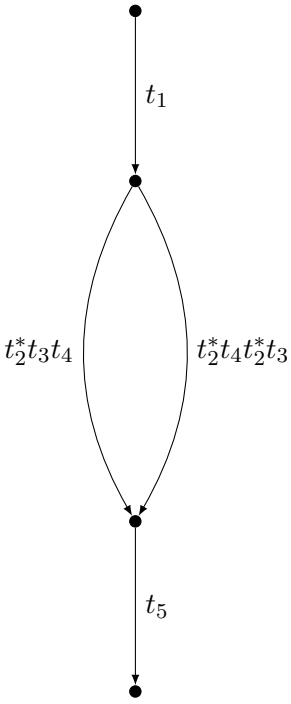
This rule can be easily lifted to the workflow net: If two transitions t_1, t_2 have the same pre-set and post-set, then $M \xrightarrow{t_1} M'$ iff $M \xrightarrow{t_2} M'$ and we can merge the two transitions.



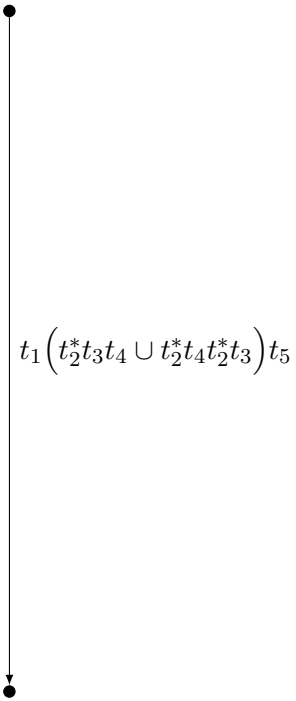
(a) Initial graph



(b) After applying rule (2)



(c) After applying rule (3) twice



(d) After applying rule (1) and then rule (3) twice

Figure 3.2: Example of the simple reduction procedure

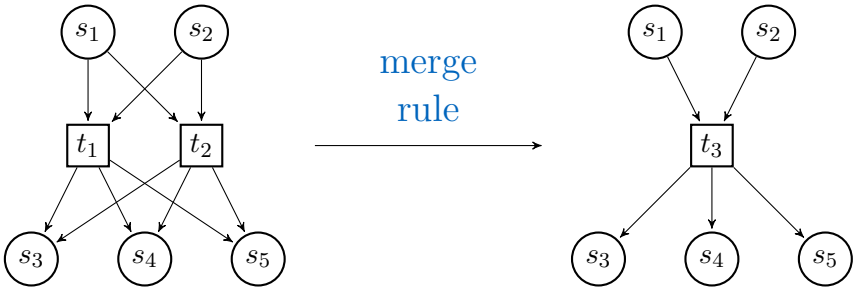


Figure 3.3: Example merge rule application

Definition 3.2. *Merge Rule*

Guard: *$\mathcal{W}$ contains two distinct transitions $t_1, t_2 \in T$ such that ${}^\bullet t_1 = {}^\bullet t_2$ and $t_1^\bullet = t_2^\bullet$.*

Action: *(1) $T := (T \setminus \{t_1, t_2\}) \cup \{t_m\}$, where t_m is a fresh name.*

(2) $t_m^\bullet := t_1^\bullet$ and ${}^\bullet t_m := {}^\bullet t_1$.

Figure 3.3 depicts a fragment of a workflow net with an example application of the merge rule.

Theorem 3.3. *The merge rule is correct.*

Proof. We have to show that an application of the merge rule preserves soundness or unsoundness. Let $\mathcal{W}_1 \xrightarrow{merge} \mathcal{W}_2$. By Theorem 2.33 it is equivalent to show that $\overline{\mathcal{W}_2}$ is live and bounded iff $\overline{\mathcal{W}_1}$ is live and bounded. This however is an immediate consequence of the definition: The reachable markings are unchanged, thus boundedness is preserved. The transition t_m is enabled at the same marking that t_1 and t_2 were enabled, thus liveness is preserved. □

3.1.4 Iteration Rule

The following was rule (2) for the labeled transition graph:

(2) Given a self-loop $M \xrightarrow{t} M$, replace every edge $M \xrightarrow{t'} M'$ by an edge $M \xrightarrow{t^*t'}$ and then remove the self-loop.

Again, this rule can be lifted to workflow nets: A transition does not change the marking if its input-places and output-places coincide. If preserving soundness is the only issue, we can simple remove all such self-loops. However, our aim is to later extend the rules to work for CWNs/PWNs. We therefore restrict ourselves to transitions of free choice clusters where it is possible to "preserve the effect" of the removed transition by shifting it to the other transitions in that cluster, similar to the modification of rule (2) which changes t' to t^*t'.

Definition 3.4. *Iteration Rule*

Guard: *$\mathcal{W}$ contains a free choice cluster c with a transition $t \in c$ such that $t^\bullet = {}^\bullet t$.*

Action: *$T := (T \setminus \{t\})$.*

In Figure 3.4 a fragment of a workflow net with an example application of the iteration rule is shown.

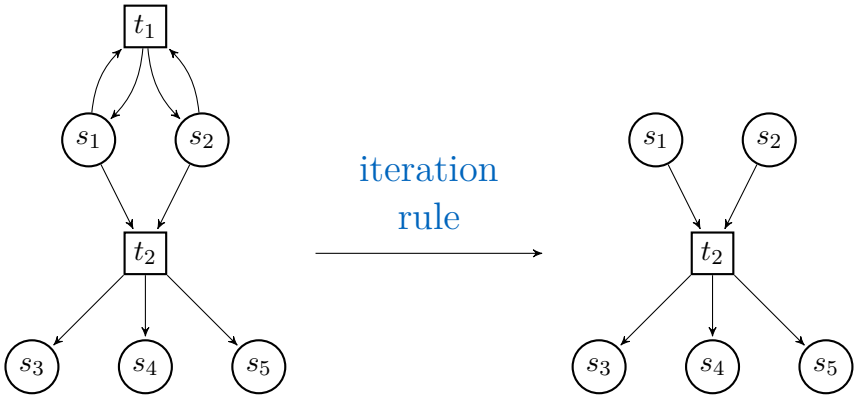


Figure 3.4: Example iteration rule application

Theorem 3.5. *The iteration rule is correct*

Proof. We have to show that an application of the iteration rule preserves soundness or unsoundness. Let $\mathcal{W}_1 \xrightarrow{iteration} \mathcal{W}_2$. As for the merge rule, we show that $\overline{\mathcal{W}_2}$ is live and bounded iff $\overline{\mathcal{W}_1}$ is live and bounded. The application of the iteration rule removes a transition but does not change the reachable markings, thus boundedness is preserved. Since the net is strongly connected, t cannot be the only transition in the free-choice cluster c, therefore liveness is also preserved. □

3.1.5 Shortcut Rule

The following was the third reduction rule for finite automata:

(3) Given an edge $M \xrightarrow{t} M'$ such that $M \neq M'$ and M' has at least one successor, add for every edge $M' \xrightarrow{t'} M''$ a *shortcut edge* $M \xrightarrow{tt'} M''$, then remove the edge $M \xrightarrow{t} M'$. Furthermore, if M' has no more incoming edges, remove M' and all its outgoing edges.

This rule is much more complicated to transfer to workflow nets. The intention of the rule is to make the edge $M \xrightarrow{t} M'$ redundant by creating shortcut edges. Moreover, sometimes the rule removes a node from the graph.

However, when we consider a workflow net, it is much harder to define shortcuts so that a transition is redundant.

Example 3.1.

Consider the workflow net in Figure 3.5 on the left. The net has two occurrence sequences, $\sigma_1 = t_1t_4t_5$ and $\sigma_2 = t_2t_3t_4t_5$. We can therefore conclude that after t_3 fires, it will always be the case that t_4 fires next. We want to create a new transition that is enabled whenever t_3 is enabled and that combines t_3t_4 into one transition.

This however is not that easy as t_4 can only fire when the place $s_2 \in t_3^\bullet$ is marked and additionally s_3 is marked. If the shortcut transition ignores the token on s_3 and puts a token on s_4 and s_5, that token on s_3 will never be removed resulting in an unsound net. Our result might look like the net shown in Figure 3.5 on the right.

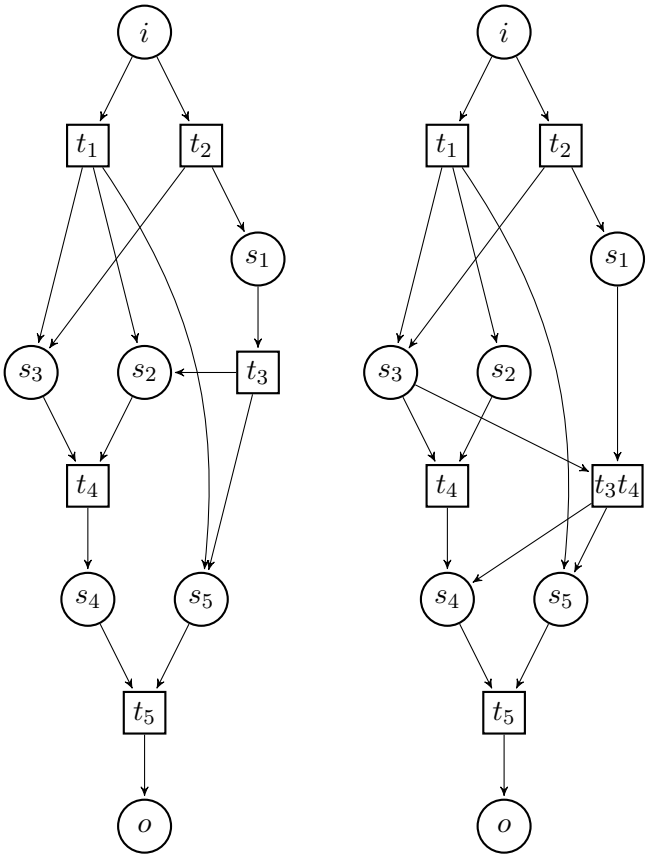


Figure 3.5: Trying to shortcut t_3

We try to generalize this approach : Take two transitions t and t' such that $t^\bullet \cap {}^\bullet t' \neq \emptyset$. Create a shortcut transition t_s with ${}^\bullet t_s = {}^\bullet t \cup {}^\bullet t' \backslash t^\bullet$ and $t_s^\bullet = t'^\bullet \cup t^\bullet \backslash {}^\bullet t'$.

Figure 3.6 shows the problems of this approach: What if some tokens needed by t' can only be produced after t has fired? In this case, we try to shortcut t_1 and t_3. The transition t_3 consumes a token from s_3 which is only produced after t_1 and then t_2 has fired. Creating the transition $t_1 t_3$ which has s_3 as pre-place creates a dead transition.

For this reason, we restrict possible shortcuts to transitions t such that after executing them from any marking that enables t, a cluster c is enabled. In this case, we know that some transition in c will eventually be fired as a consequence of t. For example, in Figure 3.5 we know that after t_1 fires, t_4 will be enabled and we can apply a shortcut there.

Definition 3.6 ([18])**.** *A transition* t unconditionally enables *a cluster* c *if* $c \cap S \subseteq t^\bullet$.

Observe that if t unconditionally enables c then any marking reached by firing t enables every transition in c.

We now turn to the second aspect: We need to remove places from the net if we ever want to arrive at a net consisting of only two places, and removing places (free choice clusters, actually) is also necessary to ensure that soundness is preserved: Removing the transition t may have the effect that the cluster c can never be enabled again and thus c must be removed.

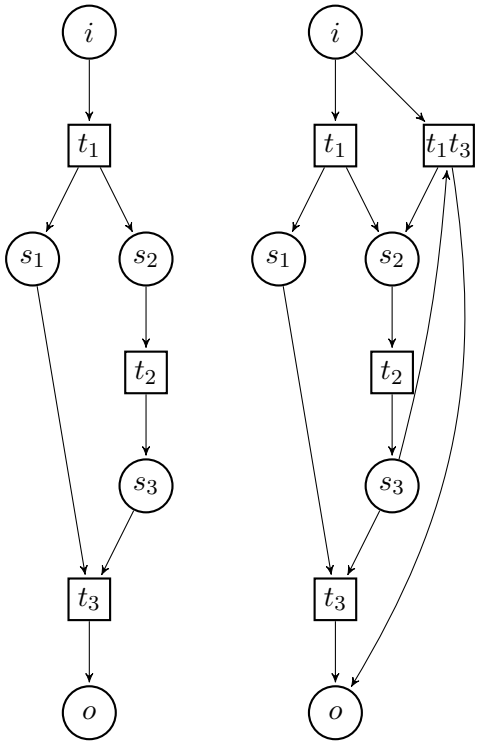


Figure 3.6: Problems with the approach

Combining the above observations, we arrive at the following formulation of the shortcut rule.

Definition 3.7. *Shortcut Rule*

Guard: *$\mathcal{W}$ contains a transition t and a free choice cluster $c \notin \{[t], [o]\}$ such that t unconditionally enables c.*

Action: *(1) $T := (T \setminus \{t\}) \cup \{t'_s \mid t' \in c\}$, where t'_s are fresh names.*

(2) For all $t' \in c \cap T$: ${}^\bullet t'_s := {}^\bullet t$ and ${t'_s}^\bullet := (t^\bullet \setminus {}^\bullet t') \cup {t'}^\bullet$.

(3) If ${}^\bullet s = \emptyset$ for all $s \in c \cap S$, then remove c from $\mathcal{W}$.

An example of a shortcut rule application is shown in Figure 3.7. Transition t_1 unconditionally enables the cluster $\{s_3, s_4, t_2, t_3\}$. This cluster contains two transitions, thus there are two new shortcut transitions t_{2f} and t_{3f} generated by the shortcut rule. As the cluster had no other incoming transitions, it is removed.

Theorem 3.8. *The shortcut rule is correct.*

Proof. Assume the transition t and cluster c are as in Definition 3.7. We say that c occurs in a firing sequence σ if some transition $t \in c \cap T$ occurs in σ.

Let $\mathcal{W}_2$ be the result of applying the shortcut rule to $\mathcal{W}_1$. The proof is divided into four parts.

(1) For every initial firing sequence σ_2 of $\mathcal{W}_2$, there is an initial firing sequence σ_1 of $\mathcal{W}_1$ leading to the same marking.

Let σ_1 be the result of replacing all occurrences of t'_s (as in Definition 3.7) in σ_2 by the sequence $t\,t'$. Clearly this yields an initial firing sequence σ_1

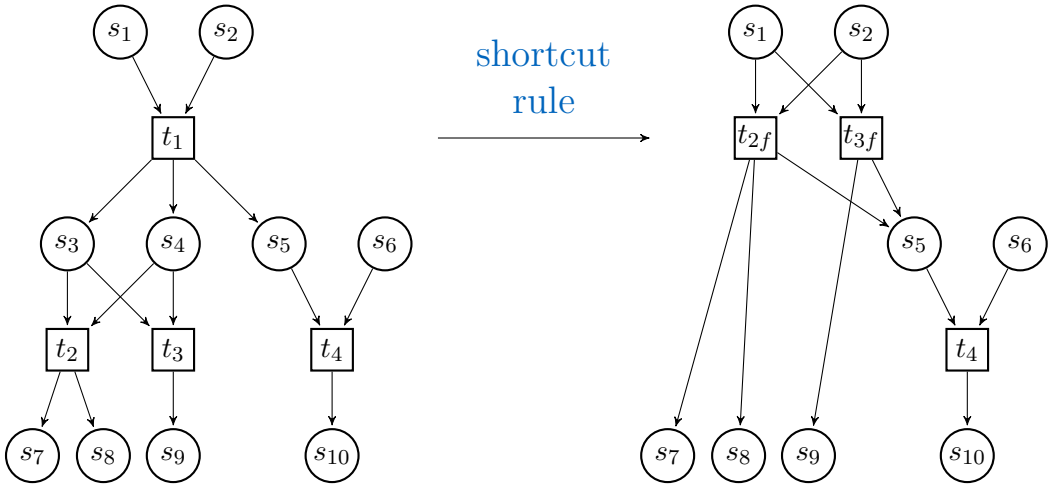


Figure 3.7: Example shortcut rule application

of $\mathcal{W}_1$. The marking reached by these two sequences is the same. We call σ_1 the corresponding sequence of σ_2 in $\mathcal{W}_1$.

(2) Let $\mathcal{W}_1$ be sound (thus bounded by Theorem 2.33), $K \in \mathbb{N}$ such that $\mathcal{W}_1$ is K-bounded. Then for each initial firing sequence σ_1 of $\mathcal{W}_1$, there is an initial firing sequence σ_2 of $\mathcal{W}_2$ and a $k \in \mathbb{N}$ with $0 \leq k \leq K$ such that σ_2 leads to the same marking as $\sigma_1 t^{*k}$ for some $t^* \in c$.

Since t unconditionally enables c, after t has fired all places of c are marked. We define σ_2 as follows: First, we extend σ_1 by as many occurrences of t^* as possible. Notice that this can be at most K many. Let k be the number of occurrences of t^* we have added.

Then we iteratively apply the following step to σ_1:

(a) Find the first occurrence of t. If there is none, stop.

(b) Find the first occurrence of $t' \in c \cap T$ after the occurrence of t found in step (1). If there is none, stop.

(c) If there was an occurrence of t' found in step (2) remove it and replace the occurrence of t found in step (1) by t'_s.

We claim that this yields a firing sequence σ_2 in $\mathcal{W}_2$ which leads to the same marking as $\sigma_1 t^{*k}$. Indeed, since we have added as many occurrences of t^* as possible and t unconditionally enables c, there must be at least as many occurrences of some $t' \in c \cap T$ as there are occurrences of t. Thus step (b) will always find a transition t' and σ_2 will not contain any occurrences of t. By definition of the shortcut rule the sequences lead to the same marking. We call σ_2 the corresponding sequence of σ_1 in $\mathcal{W}_2$.

(3) If $\mathcal{W}_1$ is sound then $\mathcal{W}_2$ is sound.

We first show that every transition in $\mathcal{W}_2$ can be enabled by some initial firing sequence. Since every transition in $\mathcal{W}_1$ can be enabled by some initial firing sequence, using the corresponding sequence in $\mathcal{W}_2$, which exists by (2), we are done for all transitions but those in c. If c still exists in $\mathcal{W}_2$, there must be some $t'' \in T$ such that such that $t'' \neq t$ and $s \in t''^{\bullet}$ for some $s \in c \cap S$. Since $t'' \neq t$,

the transition t'' is unchanged in $\mathcal{W}_2$. By soundness of W_1, some initial firing sequence σ_1 enables t''. We extend it by an occurrence of t'' and then to a firing sequence $\sigma_1 t'' \rho_1$ leading to the final marking in $\mathcal{W}_1$, which is possible since $\mathcal{W}_1$ is sound. This sequence contains an occurrence of c which is not matched by a prior occurrence of t, and so does the corresponding sequence in $\mathcal{W}_2$. Together with the fact that c is a free choice cluster, it follows that all transitions in c can be enabled in $\mathcal{W}_2$.

We now prove that every initial firing sequence σ_2 in $\mathcal{W}_2$ can be extended to a sequence that ends with the final marking. Take the corresponding occurrence sequence σ_1 in $\mathcal{W}_1$, and extend it to a sequence $\tau_1 = \sigma_1 \rho_1$ that ends with the final marking in $\mathcal{W}_1$ (possible by soundness of $\mathcal{W}_1$). The corresponding sequence in $\mathcal{W}_2$ is $\tau_2 = \sigma_2 \rho_2$, which is the extension of σ_2 (by construction of corresponding sequences) that ends with the final marking.

(4) If $\mathcal{W}_2$ is sound then $\mathcal{W}_1$ is sound.

Since every transition in $\mathcal{W}_2$ can be enabled by some initial occurrence sequence, using the corresponding sequence in $\mathcal{W}_1$ we see that the same is true for all transitions but those in c. However, it is then easy to show that the transitions in c can also be enabled in $\mathcal{W}_1$: take the initial firing sequence that enables t and extend it by t which unconditionally enables c.

For an initial occurrence sequence σ_1 in $\mathcal{W}_1$, the corresponding occurrence sequence σ_2 in $\mathcal{W}_2$ can be extended to a sequence $\tau_2 = \sigma_2 \rho_2$ that ends with the final marking in $\mathcal{W}_2$. The corresponding sequence τ_1 in $\mathcal{W}_1$ is either $\sigma_1 \rho_1$ or $\sigma_1 t' \rho_1$ for some $t' \in c \cap T$, an extension of σ_1 that ends with the final marking.

By (3) and (4) the shortcut rule is correct. □

We have defined three correct rules for workflow nets: The shortcut, merge and iteration rule. The remainder of this chapter is structured as follows: First we discuss two syntactic special cases of workflow nets and study how our rules can be used to reduce these cases. Then we extend our approach to cover the general case and we show that our three rules are *complete for free choice workflow nets*. In Sections 3.4 and 3.5 we present adaptions of our rules to colored workflow nets and probabilistic workflow nets and show that these adaptions are correct for those classes (with slightly different versions of correctness that include the summary or expected value). Finally we present an example implementation and give experimental results.

3.2 Reduction of Simple Cases

In this section, we explore ways to reduce two special subclasses of workflow nets using our rules. We start by considering nets which are acyclic and then turn to S-nets, nets that consist of a single S-component. In the latter the number of tokens remains constant, as in any S-component, and we have a single token that moves through the net. The solution to these two cases will be used in the next section to create a general algorithm.

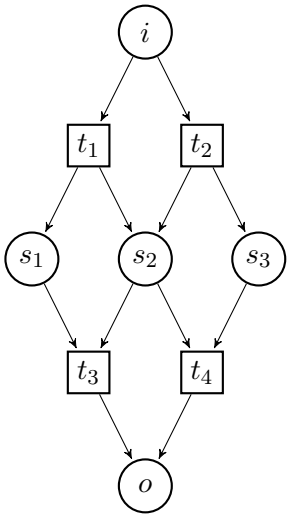


Figure 3.8: No rule is applicable

3.2.1 Acyclic Nets

Even for acyclic nets, the three rules we presented are not complete.

Example 3.2.

Figure 3.8 shows an acyclic sound workflow net where no rule is applicable. Obviously the merge rule is not applicable as there are no two transitions with the same input- and output-places. The iteration rule is not applicable either. For the shortcut rule, notice that the clusters are $\{i, t_1, t_2\}$, $\{s_1, s_2, s_3, t_3, t_4\}$ and $\{o\}$. No transition unconditionally enables any of those clusters except $\{o\}$ and therefore the shortcut rule is not applicable.

However, if we focus on free choice nets, we get a completeness result.

Theorem 3.9. *The shortcut and merge rule are complete for acyclic free choice workflow nets.*

Proof. Since the correctness of the rules has already been proven, we have to show that the rules completely reduce every sound acyclic free choice workflow net. Let $\mathcal{W}$ be a sound acyclic free choice workflow net. The proof has three parts:

(1) If $\mathcal{W}$ has more than two places, then the shortcut rule is applicable.

Since $\mathcal{W}$ is acyclic, its graph induces a partial order $<$ on the places. We define a partial order on the clusters by setting $c_1 < c_2$ if for some places $s_1 \in c_1$ and $s_2 \in c_2$ we have $s_1 < s_2$. We claim that this partial order is well defined for sound acyclic free choice nets.

Assume the contrary: There are two clusters c_1, c_2 and four places $s_1, s_1' \in c_1$ and $s_2, s_2' \in c_2$ such that $s_1 < s_2$ and $s_2' < s_1'$. This means in the graph of the workflow net there is a path π_1 from s_1 to s_2 and also a path π_2 from s_2' to s_1'. By soundness, any transition in c_1 can be enabled, in particular the first transition t_1 of π_1. We start with a sequence σ that enables t_1 and extend it by firing t_1. By soundness, the final marking must be reachable after σ. Therefore

some transition in the cluster containing the next transition along π_1 can be enabled (because there is a token on at least one pre-place of that transition) and by the free choice condition, all transitions in that cluster can be enabled, in particular the next transition of π_1. Continuing with this argument yields a transition sequence that enables c_2. We argue in the same way following the path π_2 and then eventually c_1 can be enabled again, thus there is a cycle in $\mathcal{W}$.

We now consider an arbitrary linearization of that partial order. Let c_1 be the first cluster after $[i]$. Since c_1 can be enabled by soundness and its only incoming edges are from the cluster $[i]$, there must be a transition in that cluster that unconditionally enables c_1 and therefore the shortcut rule is applicable.

(2) The shortcut and the merge rule cannot be applied infinitely often.

Let $\mathcal{W}_2$ be the result of applying any of the rules to a workflow net $\mathcal{W}_1$. For every firing sequence σ_2 in $\mathcal{W}_2$ leading to the final marking, let $\phi(\sigma_2)$ be defined as follows: If the merge rule was applied with t_1, t_2, t_m as in Definition 3.2, replace every occurrence of t_m in σ_2 by t_1. If the shortcut rule was applied with t, t' and t'_s as in Definition 3.7, replace every occurrence of t'_s in σ_2 by tt'. Then we have $|\sigma_2| \leq |\phi(\sigma_2)|$ for all such sequences σ_2 and for at least one of them we have $|\sigma_2| < |\phi(\sigma_2)|$ if the shortcut rule was applied. Since the set of firing sequences leading to the final marking in an acyclic workflow net is finite, the shortcut rule cannot be applied infinitely often. Since the merge rule reduces the number of transitions by one, it also cannot be applied infinitely often.

(3) If $\mathcal{W}$ has exactly two places, the merge rule can be applied until there is exactly one transition left.

In this case the only two places are i and o. Since $\mathcal{W}$ is sound, all transitions have as pre-place i and as post-place o and the merge rule can be applied.

□

We have shown that acyclic nets can be completely reduced by applying the shortcut and merge rule. However, applying the shortcut rule has one important shortcoming: It generates additional transitions, potentially exponentially many. This leads to an exponential number of merge rule applications.

Example 3.3.

Figure 3.9 shows a workflow net in which it is possible to create exponentially many transitions using the shortcut rule. Transition t_1 unconditionally enables all k clusters from $[s_{1a}]$ to $[s_{ka}]$. If we shortcut all of those, 2^k transitions will be created.

This exponential blowup can however be avoided by first applying the shortcut rule to t_{1b} and the cluster $[s_{1b}]$, also to t_{1c} and the cluster $[s_{1c}]$, and then merging the two transitions of the cluster $[s_{1a}]$ which then both have as post-set s_{1d}.

We therefore introduce a variant of the shortcut rule with the following addition to the guard: the cluster c unconditionally enabled by t contains only one transition.

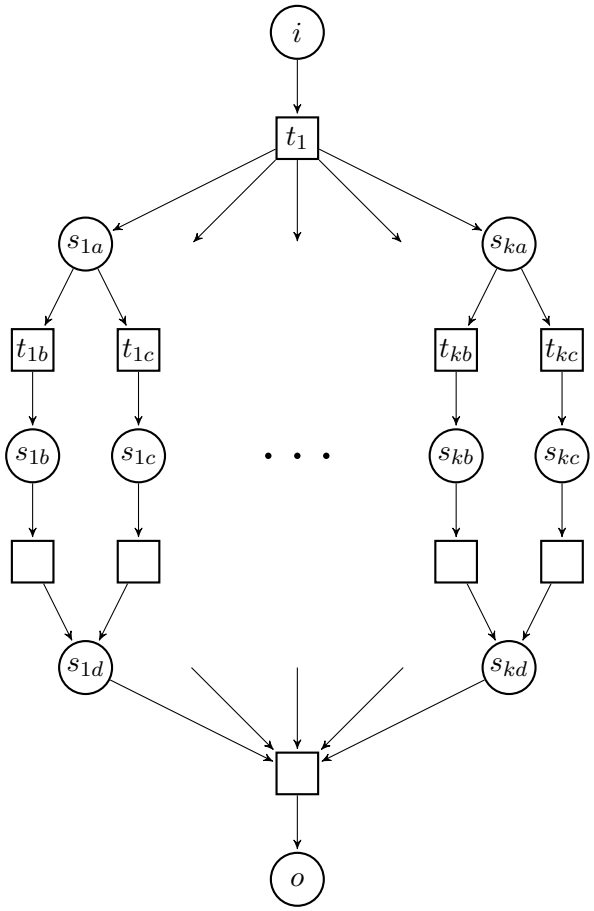


Figure 3.9: Exponentially many shortcuts possible

Definition 3.10. *D-Shortcut rule*[1]

Guard: *$\mathcal{W}$ contains a transition t and a free choice cluster $c \notin \{[t], [o]\}$ such that t unconditionally enables c and $|c \cap T| = 1$.*

Action: *(1) $T := (T \setminus \{t\}) \cup \{t'_s \mid t' \in c\}$, where t'_s are fresh names.*

(2) For all $t' \in c \cap T$: ${}^\bullet t'_s := {}^\bullet t$ and $t'^\bullet_s := (t^\bullet \setminus {}^\bullet t') \cup t'^\bullet$.

(3) If ${}^\bullet s = \emptyset$ for all $s \in c \cap S$, then remove c from $\mathcal{W}$.

The proof that the d-shortcut rule and the merge rule are still complete is much more involved, but will result in a polynomial number of rule applications.

Theorem 3.11. *The d-shortcut and merge rule are complete for acyclic free choice workflow nets.*

An additional term will be used during the proof of this theorem.

Definition 3.12 (Irreducible)**.** *We call a net* irreducible *if neither the d-shortcut nor the merge rule is applicable.*

We split the proof of Theorem 3.11 into three lemmas from which the result follows. Recall that by Theorem 2.13 for a sound free choice workflow net $\mathcal{W}$ the extended net $\overline{\mathcal{W}}$ is live and bounded and by Theorem 2.18 $\overline{\mathcal{W}}$ is covered by S-components.

Lemma 3.13. *Let $\mathcal{W}$ be an acyclic sound free choice workflow net that is irreducible and let $s \in S$ be a place of $\mathcal{W}$ with $|s^\bullet| > 1$. Then every S-component of $\overline{\mathcal{W}}$ contains an element in $[s] \cap S$.*

Proof. We proceed in two steps.

(a) There is a transition t in $s^\bullet$ such that: either $t^\bullet = \{o\}$ or t unconditionally enables some cluster c with $|c \cap T| > 1$.

This is the core of the proof. We first claim: if t unconditionally enables some cluster c, then (a) holds. Indeed: if t enables some cluster c, then either $c = [o]$ or $|c \cap T| > 1$, because otherwise the d-shortcut rule can be applied to t and c, contradicting the irreducibility of $\mathcal{W}$. This proves the claim.

It remains to prove that t unconditionally enables some cluster c. For this, we assume the contrary, and prove that $\mathcal{W}$ contains a cycle, contradicting the hypothesis.

Since the merge rule is not applicable to $\mathcal{W}$, the set $s^\bullet$ contains two transitions t_1, t_2 such that $t_1^\bullet \neq t_2^\bullet$. We proceed in three steps.

(a1) For every reachable marking M that marks $[s]$ there is a sequence σ such that $M \xrightarrow{t_1\sigma} M_1$ and $M \xrightarrow{t_2\sigma} M_2$ for some markings M_1, M_2, and the sets C_1 and C_2 of clusters marked by M_1, M_2 are disjoint.

Let σ be a longest occurrence sequence such that $M \xrightarrow{t_1\sigma} M_1$ and $M \xrightarrow{t_2\sigma} M_2$ for some markings M_1, M_2 (notice that σ exists, because all occurrence sequences of $\mathcal{W}$ are finite by acyclicity). We have $C_1 \cap C_2 = \emptyset$, because otherwise we can extend σ by firing any transition of any cluster marked by both markings. In particular it must be that $M_1 \neq \boldsymbol{o} \neq M_2$ which also follows from the fact that $t_1^\bullet \neq t_2^\bullet$ and therefore the sequences $t_1\sigma$ and $t_2\sigma$ cannot end in the same marking.

[1] This rule was first defined for negotiations [16] where it was used to summarize deterministic negotiations, thus d-shortcut rule or deterministic shortcut rule

(a2) For every $c_1 \in C_1$ there is a path leading from some $c_2 \in C_2$ to c_1, and for every $c_2 \in C_2$ there is a path leading from some $c_1 \in C_1$ to c_2.

By symmetry it suffices to prove the first part. Since C_1 and C_2 are disjoint, c_1 is marked by M_1 but not by M_2. Thus there is a place s_1 in c_1 that is not marked by M_2. Since the sequences $t_1\sigma$ and $t_2\sigma$ only differ in their first element, it must hold that $s_1 \in t_1^\bullet$ and $s_1 \notin t_2^\bullet$. Let R be an S-component of $\overline{\mathcal{W}}$ that contains s_1. Then R contains ${}^\bullet s_1$ and thus t_1. It also contains some place in $c_1 = {}^\bullet t_1$ and therefore also each transition in c_1 and in particular t_2. In R, let s_2 be the post-place of t_2. It is clear that $s_2 \neq s_1$ and even $s_2 \notin c_1$ as an S-component can only contain one place per cluster.

Comparing the markings after t_1 fired and after t_2 fired, the token of R is in s_1 in the first case and in s_2 in the second case. As argued above, the places s_1 or s_2 will remain marked during the sequence σ and M_2 marks s_2 while M_1 marks s_1 (see Figure 3.10).

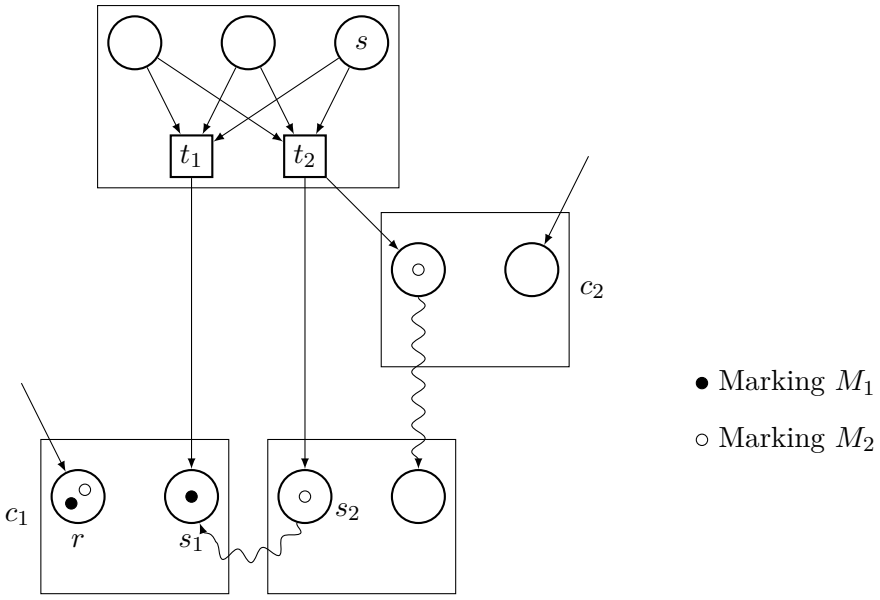


Figure 3.10: Illustration of the proof of Lemma 3.13.

We first show that there is a path from $[s_2]$ to $[s_1] = c_1$. By assumption, there is no transition t of $[s]$ such that t unconditionally enables some cluster c, and in particular t_1 does not unconditionally enable c_1. Thus c_1 contains a place $r \neq s_1$ such that $r \notin t_1^\bullet$, and since M_1 marks c_1 (and therefore r), it holds that either some transition in σ marked r or r was already marked by M. Therefore M_2 must also mark r.

Since M_2 marks r, and $\mathcal{W}$ is sound, there is a sequence of transitions τ such that $M_2 \xrightarrow{\tau} M_2'$ and M_2' marks $[r] = c_1$. Since in M_2 the token in the S-component of s_1 is on s_2, there is a path from s_2 to s_1 and thus from $[s_2]$ to c_1.

We now prove that there is a path from some $c_2 \in C_2$ to $[s_2]$. If $[s_2]$ is marked by M_2, then $[s_2] \in C_2$ and we are done. If $[s_2]$ is not enabled at M_2 (as in the figure) then, since M_2 marks s_2 and $\mathcal{W}$ is sound, there is a sequence of transitions τ such that $M_2 \xrightarrow{\tau} M_2'$ and M_2' enables $[s_2]$. We claim that for

every transition t fired along τ either $[t] \in C_2$ or there is a path from some cluster in C_2 to $[t]$. This again easily follows since C_2 is exactly the set of enabled clusters. As this holds for all t along τ, it also holds for s_2.

(a3) $\mathcal{N}$ contains a cycle.

Follows immediately from (a2) and the finiteness of N_1 and N_2.

(b) Every S-component of $\overline{\mathcal{W}}$ contains a place of $[s]$.
By repeated application of (a) we find a sequence of clusters and transitions $(a_1, t_1) \dots (a_k, t_k)$ such that $a_1 = [s]$, $a_k = o$, $t_i \in a_i \cap T$ and t_i unconditionally enables a_{i+1} for every $1 \leq i \leq k-1$. Since every S-component contains o by Corollary 2.38, every S-component must contain t_{n-1} and also a place of a_{n-1}, and also a place of $a_{n-2}, \dots, a_1$. □

Lemma 3.14. *Let $\mathcal{W}$ be an acyclic sound free choice workflow net that is irreducible. Every S-component of $\overline{\mathcal{W}}$ contains a place of every cluster, and for every transition t there is a cluster c satisfying $t^\bullet = c \cap S$.*

Proof. We first show that every S-component of $\overline{\mathcal{W}}$ contains a place of every cluster. By Lemma 3.13, it suffices to prove that every cluster $c \neq [o]$ contains more than one transition. Assume the contrary, i.e., some cluster different from $[o]$ contains only one transition. Since, by soundness, every transition can occur, there is an occurrence sequence $t_1 \dots t_k$ such that $[t_k]$ contains only one transition and all of $[t_1], \dots, [t_{k-1}]$ contain more than one transition. We denote by c_i the cluster $[t_i]$ for each $1 \leq i \leq k$. By Lemma 3.13, every S-component of $\overline{\mathcal{W}}$ contains a place in all of $c_1, \dots, c_k$. It follows that t_i unconditionally enables c_{i+1} for every $1 \leq i \leq k-1$. In particular, t_{k-1} unconditionally enables c_k. But then, since c_k only has one transition, the d-shortcut rule can be applied, contradicting the hypothesis that $\mathcal{W}$ is irreducible.

For the second part, assume there is a transition t and two clusters c_1, c_2 such that $c_1 \cap t^\bullet \neq \emptyset \neq c_2 \cap t^\bullet$. Let s_1 be some place in $c_1 \cap t^\bullet$ and s_2 some place in $c_2 \cap t^\bullet$. By the first part, every S-component contains a place in $[t]$, c_1 and c_2. Since $\mathcal{W}$ is sound, some reachable marking M marks $[t]$. Moreover, since all S-components contain a place in $[t]$, and every S-component contains exactly one token, the marking M marks exactly $[t]$. Let M' be the marking given by $M \xrightarrow{t} M'$. Since the S-component of s_1 contains a place in every cluster, no cluster different from c_1 can be marked at M'. Symmetrically, no cluster different from c_2 can be enabled at M'. So M' does not mark any cluster, contradicting that $\mathcal{W}$ is sound.

Therefore we obtain that for every transition t there is a cluster c such that $t^\bullet \subseteq c$. To show equality, again assume the contrary. Then following the same reasoning as above, after an occurrence sequence that ends with c, only places in c are marked but c is not marked. Thus the marking does not mark any cluster, again contradicting soundness. □

Lemma 3.15. *Let $\mathcal{W}$ be an irreducible sound acyclic free choice workflow net. Then $\mathcal{W}$ contains only two clusters $[i], [o]$.*

Proof. Assume $\mathcal{W}$ contains more than two clusters. For every cluster $c \neq [o]$, let $l(c)$ be the length of the longest path from c to o in the graph of $\mathcal{W}$. Let $c_{\min}$ be any cluster such that $l(c_{\min})$ is minimal, and let t be an arbitrary transition of $c_{\min}$ (notice that c_{min} cannot be $[i]$).

Algorithm 1 Reduction procedure for acyclic workflow nets $\mathcal{W}$

1: **while** $\mathcal{W}$ is not reduced completely **do**
2: apply the merge rule exhaustively
3: apply the d-shortcut rule
4: **end while**

By Lemma 3.14 there is a cluster c' such that t unconditionally enables c'. If $c' \neq [o]$ then by acyclicity we have $l(c') < l(c_{\min})$, contradicting the minimality of $c_{\min}$. So we have $t^\bullet = [o]$ for every transition t of $c_{\min}$.

If $c_{\min}$ has more than one transition, then the merge rule is applicable. Otherwise, since $\mathcal{W}$ is strongly connected some transition t' must have an output-place in $c \cap S$ and thus by Lemma 3.14 it must hold that $t'^\bullet = c \cap S$. Then the d-shortcut rule is applicable to t' and c_{min}. In both cases we get a contradiction to irreducibility, therefore the lemma holds. □

The above lemmas prove that as long as the free choice workflow net $\mathcal{W}$ consists of more than two clusters and one transition, one of the rules is applicable. We now show that an application of the rules actually completely reduces the net in polynomial time. The algorithm can be found as Algorithm 1.

Theorem 3.16. *Every sound free choice workflow net $\mathcal{W} = (S, T, F, i, o)$ can be completely reduced by means of $|T|$ applications of the merge rule and $|T| \cdot |C|$ applications of the d-shortcut rule where C is the set of clusters of $\mathcal{W}$.*

Proof. Since $\mathcal{W}$ is acyclic the transitions induce a partial order $<$ on the places and as we have already shown in the proof of Theorem 3.9, also on the clusters. We choose an arbitrary linearization $[i] < c_1 < c_2 < \ldots < [o]$.

The merge rule removes a transition, and since the d-shortcut rule does not change the number of transitions, the number of merge rule applications is bound by $|T|$.

For the d-shortcut rule, we associate a transition t with the regular expression $r(t)$ that expresses the relation between this transition and the original net, i.e., if the merge rule is applied to t_1 and t_2 we associate the regular expression $r(t_1)|r(t_2)$ with the new transition, for the shortcut rule and transition t and t', we associate the regular expression $r(t)r(t')$ with the new transition. Notice that every transition sequence that can be formed according to $r(t)$ corresponds to a transition sequence in the original net that can be fired from any marking that enables t by definition of the shortcut and merge rule.

Since the net is acyclic, no transition sequence can ever contain the same transition twice. (This is an easy consequence of the fact that $\overline{\mathcal{W}}$ is covered by S-components by Theorem 2.18 and every S-component without the additional transition in $\overline{\mathcal{W}}$ is acyclic if $\mathcal{W}$ is acyclic.) Therefore, if the regular expression associated with a transition t contains some transition t', it cannot happen that the shortcut rule with t and $[t']$ is applicable. We identify the newly created and the removed transition of the d-shortcut rule and infer that the d-shortcut rule can be applied at most once to each pair of a transition and a cluster, limiting the number of shortcut rule applications to $|T| \cdot |C|$. □

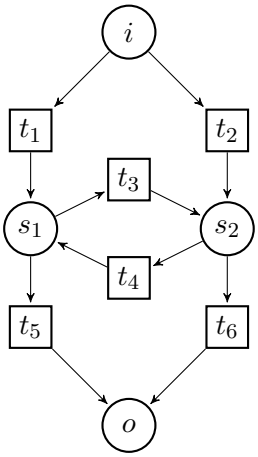


Figure 3.11: A workflow net where every cluster except $[o]$ contains two transitions

3.2.2 S-Nets

We now turn to workflow nets $\mathcal{W}$ where the extended net $\overline{\mathcal{W}}$ is an S-net.

Definition 3.17 (S-Net)**.** *A Petri net is called an S-net if the net is an S-component of itself.*

In S-nets the number of tokens is constant. Thus, as we start with a single token on i, we will only ever have to deal with one token. We describe a procedure that will reduce all free choice workflow nets where the extended net is an S-net. Notice that in an S-net, the shortcut rule can be applied to every transition as it always unconditionally enables the cluster of its post-set.

However, there are two pitfalls: first, the shortcut potentially creates more edges than it removes. Thus, we have to be careful not to create exponentially many edges. Previously, we have addressed this by using the d-shortcut rule, but this is no longer possible as there are cyclic nets where each cluster except $[o]$ contains at least two transitions, for example the one in Figure 3.11. We will show later on that it suffices to always prioritize the merge and iteration rule over the shortcut rule to keep the number of rule applications polynomial.

The second problem is that it is possible to shortcut without making progress, as the following example demonstrates.

Example 3.4.

In Figure 3.12 on the left the transition t_1 unconditionally enables the cluster $\{s_1, t_2\}$ and we can apply the shortcut rule to obtain the net on the right. The new transition t_1' unconditionally enables the cluster $\{s_2, t_3, t_4\}$. We apply the shortcut rule again and merge one of the new transitions with t_5 to arrive at a net with the same structure as the original net.

To address the second pitfall, we fix an arbitrary total order $<$ of the clusters with two restrictions: $[i]$ is the first (smallest) and $[o]$ is the last (largest) cluster in that order. Using this total order $[i] < c_1 < c_2 < \ldots < [o]$, we categorize transitions into two sets: forward transitions, those which lead from a smaller to a larger cluster, and backward transitions, the remaining ones. Our procedure will aim to remove

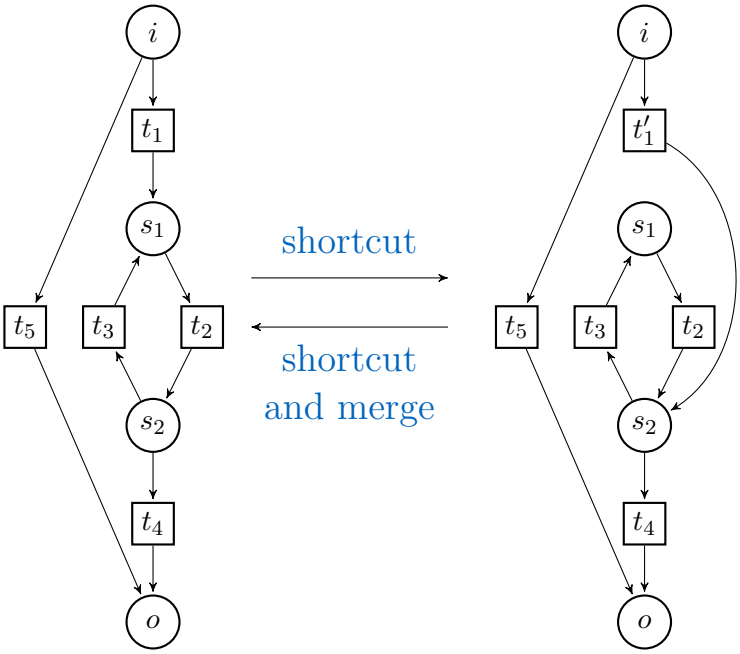


Figure 3.12: Shortcut rule and cycles

all backward transitions which will create an acyclic net to which we then apply the d-shortcut and merge rule until it is completely reduced.

We start with an important observation:

Corollary 3.18. *Let $\mathcal{W}$ be a workflow net such that $\overline{\mathcal{W}}$ is an S-net. Let $<$ be any total order on the clusters and let the transitions be categorized into forward and backward transitions where forward transitions are the transitions leading from a smaller to a larger cluster and backward transitions are the remaining ones. Then every cycle in $\mathcal{W}$ contains a backward transition.*

This is a straightforward observation as at some point during the cycle, there must be a decrease along a transition. By contraposition, Corollary 3.18 implies that if there are no backward transitions, $\mathcal{W}$ is acyclic.

From now on, we always assume that we have chosen a total order $<$ and categorized the transitions.

We introduce a new notion for S-nets.

Definition 3.19 (Output Cluster)**.** *Let $\mathcal{W}$ be a workflow net such that $\overline{\mathcal{W}}$ is an S-net. For a transition t, we call the cluster of its* unique *output place the* output cluster *of t.*

Our algorithm will always prioritize the merge rule, then the iteration rule, and if none of those two rules is applicable it will apply the shortcut rule in a special order: To reduce the number of backward transitions, we always choose a transition where the output cluster is minimal according to our total order. If we apply the shortcut rule to such a transition, we claim that any newly created backward or forward transition will have a larger output cluster than the chosen (and removed) transition.

Algorithm 2 Reduction procedure for workflow nets $\mathcal{W}$ where $\overline{\mathcal{W}}$ is an S-net

1: fix a total order on the clusters with $[i]$ as smallest and $[o]$ as largest element
2: **while** $\mathcal{W}$ is cyclic **do**
3: apply the merge rule exhaustively
4: apply the iteration rule exhaustively
5: apply the shortcut rule to a backward edge whose output cluster is minimal
6: **end while**
7: apply Algorithm 1

Lemma 3.20. *Let $\mathcal{W}$ be a workflow net such that $\overline{\mathcal{W}}$ is an S-net. Let t be a backwards transition with minimal output cluster c. Assume the iteration and merge rule have been applied exhaustively. Then any transition t'_s created by applying the shortcut rule to t and c will have a larger output cluster than c.*

Proof. Assume this is not the case and the cluster of the post-place of t' is smaller than c. Let $t' \in c$ be the transition from which t'_s was created. As $t' \in c$ and t' ends at a smaller cluster than c, it must have been a backwards transition. This however contradicts that we have chosen t such that its output cluster is minimal. □

Due to Lemma 3.20, we know that the output clusters of backwards transitions get larger until eventually they become forward transitions.

The algorithm can be found as Algorithm 2.

Example 3.5.

Consider the S-net in Figure 3.13 on the upper left. The numbers beside the places are the ordering we choose. The transition t_7 leads from s_4 to s_1 and is the only backward transition. We apply the shortcut rule and obtain the net shown in Figure 3.13 on the upper right. The transitions t_7t_2 and t_7t_3 are created, both backward edges. We start with t_7t_3 as s_3 is smaller than s_2 in our order. After one shortcut the resulting transition can be merged with t_5 (which we skip in the drawing to show the resulting transition). After we shortcut t_7t_2, the result is a self-loop in s_4 (shown on the lower left) and we apply the iteration rule to obtain an acyclic net shown on the lower right of Figure 3.13.

To obtain a runtime bound, consider the vector $(back([i]), back(c_1), \ldots, back([o]))$ where $back(c)$ is the number of backward transitions with output cluster c. With every application of the shortcut rule as described above, the first non-zero entry of this vector decreases by one, and all zero entries before that entry stay zero by Lemma 3.20. After exhaustive applications of the merge rule, every entry $back(c)$ is bounded by the number of clusters. Thus the number of shortcut rule applications is bounded by $|C|^2$ where C is the set of clusters of the net.

After each application of the shortcut rule, we apply the merge rule and the iteration rule exhaustively. Since the cluster c in the definition of the shortcut rule has at most $|C|$ outgoing transitions (one per possible output cluster) when the shortcut rule is applied, at most $|C|$ new transitions are created. The shortcut rule is applied at most $|C|^2$ times, thus at most $|C|^3$ transitions are created. Both the

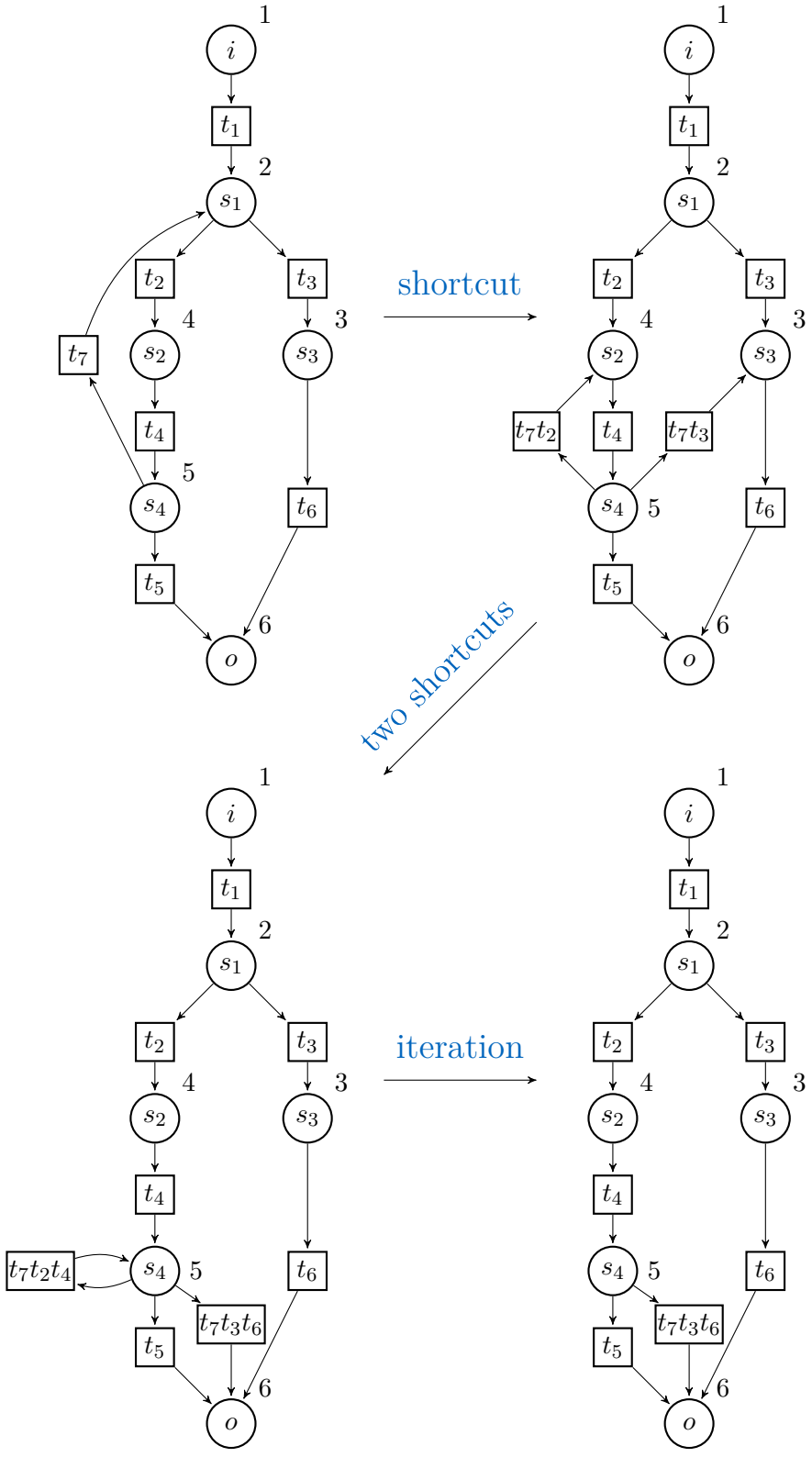


Figure 3.13: Reducing an S-net

merge and the iteration rule reduce the number of transitions by 1, thus the number of applications of the merge or iteration rule is bound by $|C|^3 + |T|$.

We obtain the following bound:

Corollary 3.21. *Let $\mathcal{W}$ be a free choice workflow net where the extended net $\overline{\mathcal{W}}$ is an S-net, and let C be the set of clusters of $\mathcal{W}$. Then $\mathcal{W}$ can be reduced to an acyclic negotiation with at most $|S|$ places and $|C|^2$ transitions by $|C|^2$ applications of the shortcut rule and $|C|^3 + |T|$ applications of the merge or iteration rule.*

3.3 A Complete Reduction Algorithm

3.3.1 The Algorithm

We have seen how to reduce acyclic nets and S-nets by means of the merge, iteration and (d-)shortcut rule. In this section, we extend the approach to all free choice workflow nets.

In the case of S-nets, we used a total order on the clusters to classify transitions as forward or backward. This was possible because each transition has exactly one pre-place and one post-place. In the general case this no longer holds true. We approach this problem by considering *fragments*, cyclic parts of the workflow net, and show that we can choose such a part and reduce it using the d-shortcut and merge rules until it consists only of clusters where for every transition, the post-set contains exactly the places of some cluster. This reduced fragment is then effectively the same as an S-component and can be reduced by the algorithm for S-nets.

We start with some definitions concerning cyclic workflow nets.

Definition 3.22 (Loop)**.** *Let $\mathcal{W}$ be a workflow net. A non-empty transition sequence σ is a* loop *of $\mathcal{W}$ if $M \xrightarrow{\sigma} M$ for some reachable marking M.*

Definition 3.23 (Synchronizer)**.** *Let $\mathcal{W}$ be a workflow net. A free choice transition t* synchronizes *a loop σ if t appears in σ and for every reachable marking M: if M enables t, then $M(p) = 0$ for every $p \in (\bigcup_{t' \in \sigma, [t'] \neq [t]} {}^\bullet t')$. A free choice transition is a* synchronizer *if it synchronizes some loop.*

The above definition describes that when a synchronizer t is enabled, of all places that are pre-places of transitions that appear in the loop, exactly the pre-places of t are marked (thus the name).

Definition 3.24 (Fragment)**.** *Let $\mathcal{W}$ be a workflow net and let t be a synchronizer of $\mathcal{W}$. The fragment $\mathcal{W}_t$ is the net that contains every transition appearing in some loop synchronized by t, together with its input and output places.*

Example 3.6.

Figure 3.14a depicts an example workflow net. An example of a loop is the sequence $t_2t_3t_4t_6t_7t_8$. This loop is synchronized by t_2, t_7 and t_8. The fragment of these synchronizers is shown in Figure 3.14b and contains the mentioned transitions as well as t_5, and all pre-places of those transitions. The whole fragment is also synchronized by t_2, t_7 and t_8.

While t_5 is not a synchronizer of the first fragment, there is a loop that is synchronized by t_5: the loop $t_5t_3t_4$. The fragment consists of those three transitions and their pre-places and t_5 is the only synchronizer and is shown in Figure 3.14c.

The fragments also induce an ordering on the synchronizers via inclusion.

Definition 3.25 (Minimal Synchronizer)**.** *A synchronizer is minimal if its fragment does not contain a smaller fragment of another synchronizer.*

We now proceed to show a crucial point concerning cyclic free choice workflow nets. It is easy to construct a cyclic workflow net without a synchronizer.

Example 3.7.

Consider the workflow net shown in Figure 3.15. After firing t_1, there are tokens on s_1, s_2 and s_4. The transitions t_2 and t_3 simply move the token from s_2 to s_3 and back.

The net is sound and cyclic and has a loop, t_2t_3, but neither of those transitions is a synchronizer.

However, for sound free choice workflow nets, it is always possible to find a synchronizer if the net is cyclic, as the following lemma states.

Lemma 3.26. *Every sound cyclic free choice workflow net has at least one synchronizer.*

Proof. We first show that every sound cyclic free choice workflow net has a loop.

Let π be a cycle of the graph of the net $\mathcal{W}$. Let t_1 be an arbitrary transition occurring in π, and let t_2 be its successor in π. Then $t_1^\bullet \neq \{o\}$ because o has no outgoing transitions, and hence no cycle contains o.

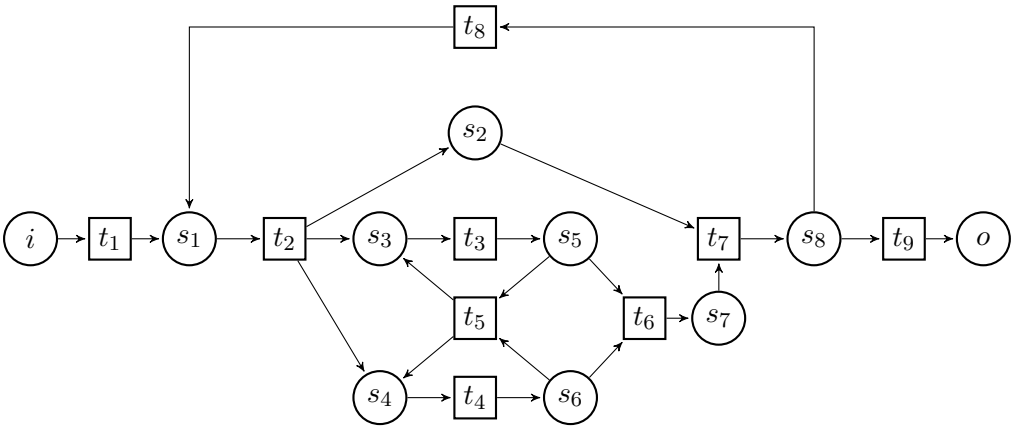
By soundness some reachable marking M_1 enables t_1. Furthermore it holds that $t_1^\bullet \cap {}^\bullet t_2 \neq \emptyset$. Let M_1' be the marking reached after firing t_1 from M_1. Again by soundness, there is an occurrence sequence from M_1 that leads to the final marking. This sequence has to contain an occurrence of a transition of the cluster $[t_2]$ because there is a token on at least one place of this cluster. In particular, some prefix of this sequence leads to a marking M_2 that enables t_2.

Repeating this argument arbitrarily for the transitions t_1, t_2, t_3, $\ldots$, $t_k = t_1$ of the cycle π, we conclude that there is an infinite occurrence sequence, containing infinitely many occurrences of transitions of the cycle π. Since the set of reachable markings is finite, this sequence contains a loop.

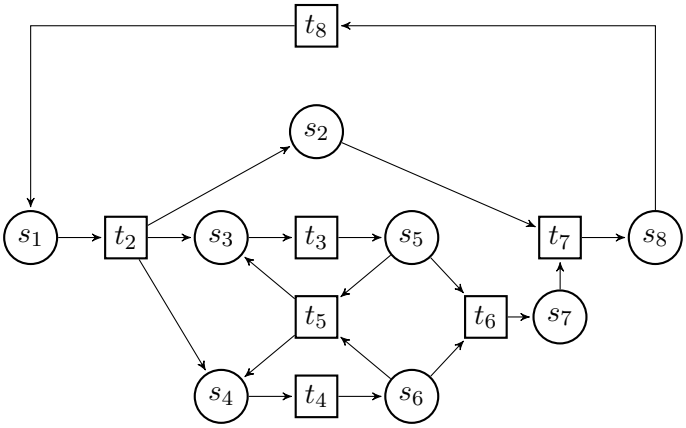
Let now σ be a minimal loop, i.e. a loop containing the least amount of different transitions possible. We show that σ contains a synchronizer.

Let M be a marking where some transition in σ is enabled. By soundness, there is some occurrence sequence τ enabled at M which leads to the final marking. Let t be the last transition in τ such that $[t] = [t^*]$ for some transition t^* that is contained in σ. We claim that t^* is a synchronizer of σ.

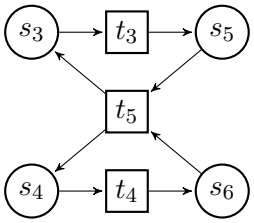
Assume that this is not the case, i.e. there is a marking M^* that enables t^* (and since the net is free choice, also t) and also marks some other place $p^* \in (\bigcup_{t' \in \sigma, [t'] \neq [t^*]} {}^\bullet t')$. We construct an occurrence sequence that is a loop and uses only transitions in σ but not t^* contradicting the minimality of σ.



(a) A cyclic workflow net



(b) The first fragment



(c) The second fragment

Figure 3.14: Example net and its fragments

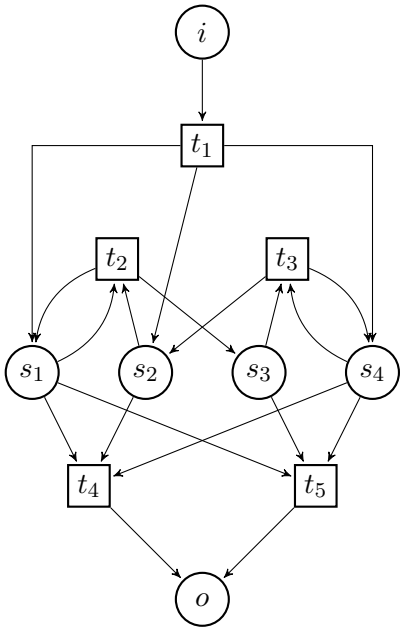


Figure 3.15: A cyclic net without a synchronizer

We use a minimal cover of S-components of $\overline{\mathcal{W}}$ which exists by Theorem 2.18 since $\mathcal{W}$ is sound and free choice. Let R be an S-component that contains t (and thus also t^*). We pick a subsequence $\tau' = t_1 t_2 t_3 \ldots$ of τ which begins with $t_1 = t$ and contains all subsequent transitions in τ that are present in R. Intuitively, this subsequence pushes the token of R along its path toward the final marking.

We now construct an occurrence sequence ρ starting from the marking M^* as follows:

1. ρ begins with t.
2. Whenever a transition in σ is enabled, extend ρ by firing that transition.
3. If no transition of σ is enabled, but a transition in τ' is enabled, extend ρ by firing that transition.
4. Otherwise, extend ρ by a minimal transition sequence that either ends with the final marking or enables some transition in σ or τ'.

First observe that there will always be some marked place in the set $\bigcup\{{}^\bullet t' \mid t' \in \sigma\}$. Indeed, initially p^* is marked by assumption that t^* is not a synchronizer. Since $[t] = [t^*]$, after t is fired in step (1) p^* is still marked. Whenever we fire a transition in σ in step (2), this transition will again mark some place in this set. If no transition in σ is enabled, the marking of the whole set $\bigcup\{{}^\bullet t' \mid t' \in \sigma\}$ cannot change since the net is free choice.

We argue that step (3) and (4) can only appear finitely often. Since τ' is finite, we can only finitely often add transition sequences that enable a transition in τ' and subsequently fire it, i.e. step (3) appears only finitely often and therefore step (4) only finitely often ends with a transition in τ' being enabled. Whenever step (4) ends with a transition in σ being enabled, it must be the case that for some S-component of our chosen cover, the token of that component is now on $\bigcup\{{}^\bullet t' \mid t' \in \sigma\}$ and was

not in that set before. This however can also only happen finitely often since the minimal cover consists of finitely many S-components. Thus it has to be the case that transitions of σ appear infinitely often in ρ. (Because of soundness and because the final marking is not reached, ρ has to be infinite, otherwise there would be a deadlock.)

However, t and t^* cannot be enabled ever again: the token of R, an S-component that contains t, cannot be in the set $\bigcup\{{}^\bullet t' \mid t' \in \sigma\}$ by construction of τ'. Thus there must be a loop which uses only a subset of transitions in σ contradicting the minimality of σ.

Thus our assumption that t^* is not a synchronizer must be false and we have shown that σ contains a synchronizer. □

Thanks to Lemma 3.26, we know that as long as the net is cyclic, we can pick a synchronizer. The following lemma shows that fragments of minimal synchronizers have a very special structure.

Lemma 3.27. *Let t^* be a minimal synchronizer of a cyclic sound free choice workflow net. Then all cycles in the fragment $\mathcal{W}_{t^*}$ contain a synchronizer.*

Proof. Let $\pi = \{t_1, t_2, \ldots, t_k\}$ be a cycle in $\mathcal{W}_{t^*}$ and let $\{s_1, \ldots, s_k\}$ be a set of places such that $s_1 \in t_1^\bullet \cap {}^\bullet t_2, \ldots, s_k \in t_k^\bullet \cap {}^\bullet t_1$.

We use the same argument as in the proof of Lemma 3.26 to show that there is an infinite sequence τ that is enabled at the initial marking and that contains infinitely many occurrences of all transitions in π. We start with a sequence τ_0 that enables t_1 in π (exists by soundness). We then extend this sequence according to the following rules:

(1) If the next transition in π is enabled, fire that transition.

(2) Otherwise, pick a shortest sequence that leads to the final marking and fire transitions until the next transition in π is enabled.

By soundness, (2) is always possible. Since the net is free choice and there is always a token on some pre-place of a transition of π, (1) will occur infinitely often.

The constructed sequence τ must contain a loop since the number of markings is finite. In particular, it must contain a loop that contains every transition of π. We call this loop σ. This loop corresponds to a T-invariant v. By Theorem 2.25, v is the sum of minimal T-invariants. Since $v(t_1) > 0$, some minimal invariant v^* must also have $v^*(t_1) > 0$. But then it must also have $v^*(t') > 0$ for some transition $t' \in [s_1]$ and since the only transition in $[s_1]$ that we fired is t_2, it must be that $t' = t_2$. Repeating this reasoning, v^* must have $v^*(t) > 0$ for all $t \in \pi$.

According to Proposition 2.29, v^* corresponds to a minimal T-component and by Theorem 2.22 this T-component can be activated. Thus this T-component contains a minimal loop σ^* and according to the proof of Lemma 3.26 a synchronizer. By construction, if this component is activated there is some token on the pre-place of a transition in π and during σ^* this will always be the case, thus the synchronizer by definition must be one of the transitions in π. □

We now show that due to the special structure of minimal fragments shown in Lemma 3.27, it is possible to use the d-shortcut and merge rule to remove all non-synchronizers from such a minimal fragment.

Lemma 3.28. *Let t^* be a minimal synchronizer of a cyclic sound free choice workflow net. Then all non-synchronizers of $\mathcal{W}_{t^*}$ can be removed by means of applications of the d-shortcut and merge rules.*

Proof. Let $\mathcal{W} = (S, T, F, i, o)$ be a cyclic sound free choice workflow net, t^* a minimal synchronizer, $\mathcal{W}_{t^*}$ its fragment, and U the set of synchronizers in $\mathcal{W}_{t^*}$. Remember that $\mathcal{W}_{t^*}$ is defined as the set of all transitions t such that t is a part of some loop synchronized by t^*, together with all input and output places of those transitions t. We construct an auxiliary net which will be acyclic.

Every cycle in $\mathcal{W}_{t^*}$ must contain some transition in U by minimality of t^*. We now describe our auxiliary net:

$\mathcal{W}' = (S', T', F', i', o')$ such that

- $S' = \{s \in W_t\} \cup \{i', o'\} \cup \{s' \mid s \in {}^\bullet t^*\}$.
- $T' = \{t \in W_t\} \cup \{t', t''\}$.
- For each $t \in W_t$:
  - ${}^\bullet t$ in $\mathcal{W}'$ is the same as in $\mathcal{W}$.
  - $t^\bullet$ in $\mathcal{W}'$ is $\begin{cases} \{o'\} & \text{if } t \in U \text{ and } t \neq t^*. \\ \phi(t^\bullet) & \text{otherwise, where } \phi \text{ replaces every place } s \in {}^\bullet t^* \\ & \text{by its copy } s' \in S'. \end{cases}$
- $(i, t') \in F'$.
- $(t', s) \in F$ for all $s \in {}^\bullet t^*$.
- $(s', t'') \in F'$ for each $s \in {}^\bullet t^*\}$.
- $(t'', o') \in F$.

Finally, we remove all unreachable clusters.

We describe the construction above informally. We take the fragment of t, add a copy of the places ${}^\bullet t^*$ and also an additional start place i' and end place o'.

We redirect each synchronizer except t^* directly to the output place o'. We also redirect all transitions that have as post-place some place in ${}^\bullet t^*$ to the copy of ${}^\bullet t^*$.

We add a transition t' that takes a token from i' and places one on each place of ${}^\bullet t^*$. We furthermore add a transition t'' that has as pre-set the copy of ${}^\bullet t^*$ and as post-set the output place o'.

Intuitively, we start by enabling the cluster of t^* and whenever we hit a synchronizer, we stop and go to the final marking.

To ensure soundness, we remove all non-reachable clusters. Clusters could be unreachable because they do not appear on a path from $[t^*]$ to another synchronizer's cluster, but e.g. on a path from another synchronizer's cluster to $[t^*]$.

We claim that $\mathcal{W}'$ is an acyclic sound free choice workflow net. $\mathcal{W}'$ contains all paths in the fragment W_{t^*} leading from $[t^*]$ to any synchronizer in S. The extended net $\overline{\mathcal{W}'}$ is strongly connected because we have removed all unreachable clusters. Since all cycles in the fragment W_{t^*} contain a transition in U, the net $\mathcal{W}'$ must be acyclic by construction. Any deadlock in $\mathcal{W}'$ would mean that in $\mathcal{W}$ the tokens get stuck, thus $\mathcal{W}'$ is deadlock-free and therefore sound: the final marking is always reachable because the net is acyclic and deadlock-free. Every transition occurs in

some sequence because the net is strongly connected and free choice which means that for a given path to a transition, we can fire exactly the transitions along the path (possibly with other transitions interleaved) until the transition in question is fired.

It follows that $\mathcal{W}'$ can be reduced by means of the d-shortcut and merge rules. While the d-shortcut rule may be applicable to t' and $[t^*]$ at some point, we defer this application to the end of the reduction.

Applying the same rule sequence in the original net $\mathcal{W}$ (we do not formally define what it means to apply the sequence to the original net, but it should be clear from Example 3.8) results in the transition t^* in $\mathcal{W}_{t^*}$ being shortcut to directly enable some $t \in U$. We repeat this for the other synchronizers in S and thus reduce the fragment to synchronizers only. □

Example 3.8.

We continue with the net shown in Figure 3.14a. We start with the smallest fragment, again depicted in Figure 3.16a. This fragment contains non-synchronizers t_3 and t_4. The auxiliary net $\mathcal{W}'$ is depicted in Figure 3.16b. The places s_5 and s_6 have been duplicated, the transitions t_3 and t_4 redirected to the copies.

The d-shortcut rule is applicable to t_5 and $[t_3]$ and also to t_5 and $[t_4]$ and we apply it in $\mathcal{W}'$. We also apply the d-shortcut rule to t_5 and $[t_3]$ and also to t_5 and $[t_4]$ in the fragment. Figures 3.16c and 3.16d show the resulting fragment and auxiliary net.

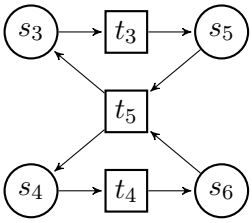
Notice that while in the auxiliary net and in the fragment the transitions t_3 and t_4 are removed by the application of the d-shortcut rule, this is not the case if we apply the d-shortcut rule to the same transitions in the complete net shown in Figure 3.14a. The result of these two shortcuts is depicted in Figure 3.17. However, the transitions t_3 and t_4 are no longer part of the fragment synchronized by t_5. As a result, t_5 is now a self-loop with pre-set and post-set $\{s_5, s_6\}$ and can be removed via the iteration rule. The effect on the original net is depicted in Figure 3.18a.

We continue with the second fragment, depicted in Figure 3.18b. This fragment contains non-synchronizers t_3, t_4 and t_6 which can be removed by repeated application of the d-shortcut rule with t_2. The result is the fragment shown in Figure 3.18c. This fragment contains only synchronizers.

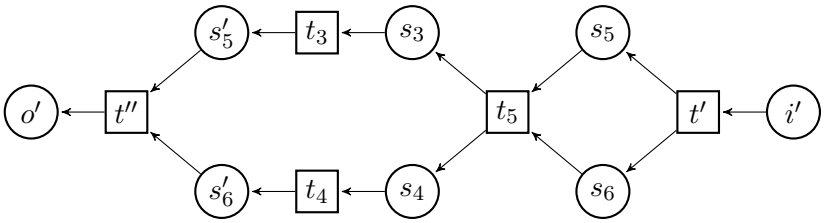
Due to Lemma 3.28, we can reduce the fragment of a minimal synchronizer until it contains only synchronizers. Thereafter, the structure of the fragment is very similar to an S-component: every transition has as post-set exactly one other cluster or its post-set lies completely outside of the fragment. Intuitively, the synchronizer-only fragment is an S-component where some places were duplicated. We describe an auxiliary workflow net which has a similar structure as the fragment and is an S-net.

Let $\mathcal{W}$ be a free choice workflow net, t_* a minimal synchronizer, $\mathcal{W}_{t_*} = (S, T, F)$ its fragment containing only synchronizers. We define the net $\mathcal{W}' = (S', T', F', i', o')$ where

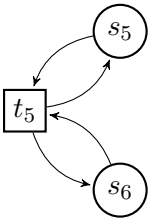
- $S' = \{c \mid c \text{ cluster in } \mathcal{W}_{t_*}\} \cup o'$



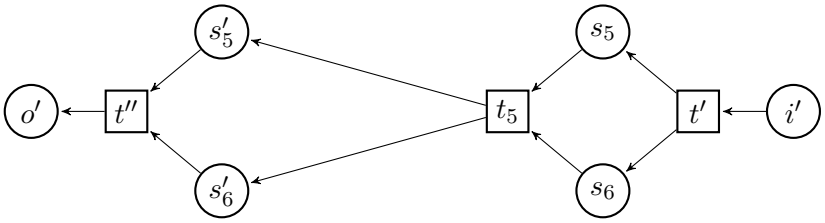
(a) The smallest fragment



(b) The auxiliary net



(c) Fragment after two shortcuts



(d) Auxiliary net after two shortcuts

Figure 3.16: Reducing the smallest fragment

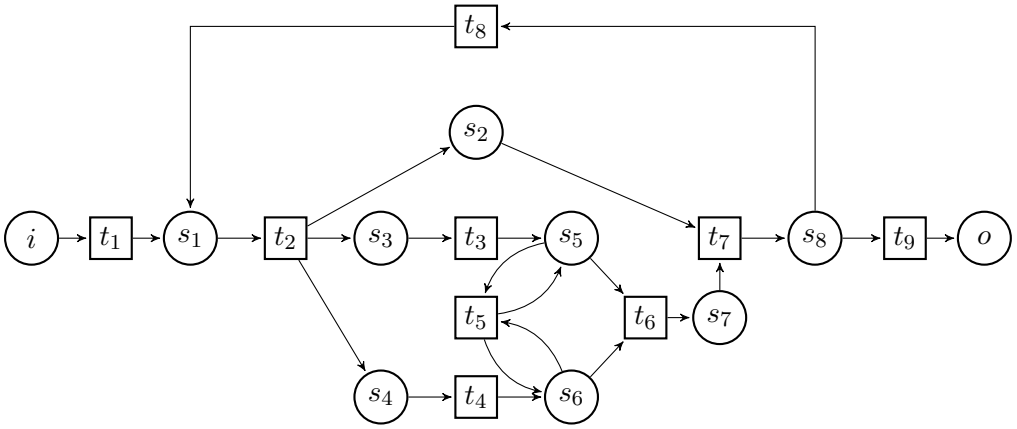


Figure 3.17: Original net after two shortcuts

- $T' = \{t' \mid t \in T\}$
- $i' = {}^\bullet t'_*$
- $(c, t') \in F'$ if ${}^\bullet t = c \cap S$
- $(t', c) \in F'$ if $t^\bullet = c \cap S$ and $c \neq i'$
- $(t', o') \in F'$ if $t^\bullet = c \cap S$ and $c = i'$.

This auxiliary net combines the places of each cluster into a single place and also splits the cluster of t_* into two, the starting point i' with the transitions and the end point o' to which all transitions that would lead into $[t_*]$ are redirected.

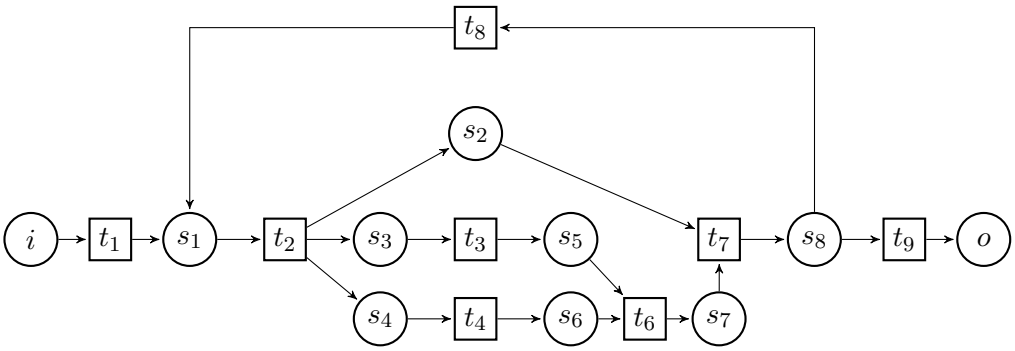
The net we constructed is an S-net and we can apply Algorithm 2 to reduce the net. We apply the same rules to the original net and reduce the fragment. When the auxiliary net is reduced completely, the fragment will consist of a single self-loop in $[t_*]$ which can be removed with the iteration rule. Due to additional transitions which originate in the fragment but to not belong to the fragment, it may be necessary to substitute the d-shortcut rule with the shortcut rule during the reduction.

Example 3.9.

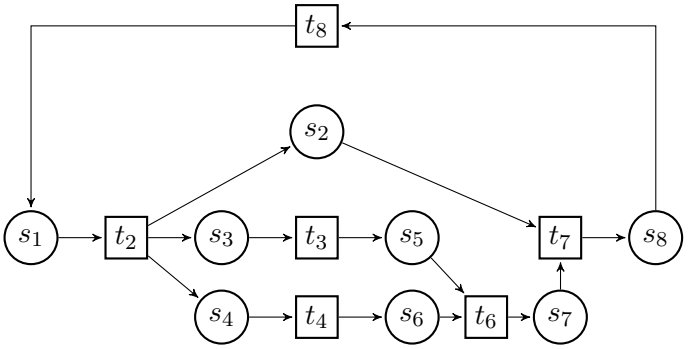
We continue with the synchronizer-only fragment shown in Figure 3.18c. The auxiliary net is depicted in Figure 3.18d and can be reduced by three applications of the d-shortcut rule. We apply the same rules to the original net and obtain a self-loop in s_1 which we remove with the iteration rule. Observe that when we apply the d-shortcut rule to $[t_8]$, in the original net the cluster contains another transition, t_9, and so we have to apply the shortcut rule there. The remaining net is acyclic.

We have now established the theoretical foundation and can present our reduction algorithm for sound free choice workflow nets. The algorithm is given as Algorithm 3, including some comments regarding the unsound case that we will discuss later.

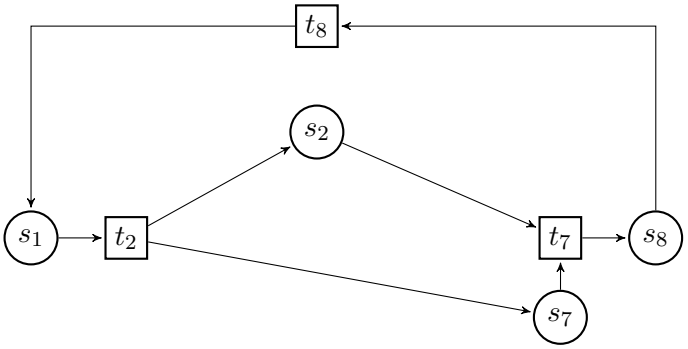
We begin with a possibly cyclic workflow net $\mathcal{W}$. While this workflow net is cyclic, Lemma 3.26 states that there is a synchronizer and so in line 2 we can pick a minimal synchronizer and compute its fragment. Computing a synchronizer and its fragment



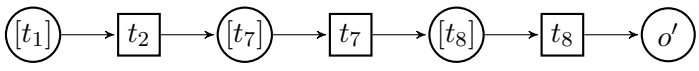
(a) After reduction of the smallest fragment



(b) The remaining fragment



(c) Synchronizer-only fragment



(d) Auxiliary net

Figure 3.18: Reducing the fragments

is quite complicated, thus we describe how to compute them in detail in the next subsection.

By Lemma 3.28 we know that in a fragment of a sound free choice workflow net, all non-synchronizers can be removed by means of the d-shortcut and merge rule. We thus apply these rules in lines 4-8, prioritizing the merge rule and also adding the iteration rule as the shortcuts might create a self-loop.

After removing all non-synchronizers, the fragment F is reduced by applying the algorithm for S-nets as described above. In line 15, we need to apply the iteration rule to remove self-loops created by shortcutting the last backwards transition inside F. Thereafter, F is acyclic and we can continue, either with another fragment if $\mathcal{W}$ is still cyclic, or with the algorithm for acyclic nets if $\mathcal{W}$ is no longer cyclic.

We now turn to the case of unsound nets. We discuss the lines where in the case that the net is unsound, errors can occur.

In line 2, we pick a minimal synchronizer of the cyclic workflow net $\mathcal{W}$. If the net is unsound, it may happen that there is no synchronizer and we stop here reporting that the net is unsound.

In line 3, we construct the fragment of c. From the above observations, we know that in the sound case, every transition that originates in the fragment has a post-set that either lies completely inside or completely outside the fragment. Furthermore, transitions of the second type must originate from a synchronizer. If one of those conditions is violated, we immediately stop and report that the net is unsound.

We then start reducing the fragment starting in line 4. In every iteration, we apply the shortcut rule at least once. By Lemma 3.28 this must be possible as long as there are non-synchronizers, otherwise the net cannot be sound and we can abort the procedure. Furthermore, this while loop must terminate after a polynomial amount of rule applications: The merge and iteration rule both remove a transition and the d-shortcut rule does not increase the number of transitions, so the first two rules can only be applied at most $|T|$ times. Assume the d-shortcut rule can be applied $|T|^2$ times. If we identify the removed and added transition, there must be some transition to which the d-shortcut rule was applied at least $|T|$ times. This however would imply that there is a cycle of clusters which all have only one outgoing transition, the transition that belongs to the cycle, which is a contradiction to strong connectedness of the extended net.

We continue by fixing a total order on F. By now, F can only consist of synchronizers and for every transition, the post-set is a single cluster, so F must be a sound part of the net. We continue by reducing F.

After we have removed all fragments, $\mathcal{W}$ is no longer cyclic and we apply the reduction algorithm for acyclic nets. Again, if no rule application was possible, the net must be unsound due to Theorem 3.11.

We arrive at the following theorem:

Theorem 3.29. *Every sound free choice workflow net can be summarized by Algorithm 3. Any unsound free choice workflow net can be recognized as unsound by Algorithm 3.*

Before we turn to the computation of synchronizers and fragments as well as a runtime analysis, we illustrate our algorithm by example.

Algorithm 3 Reduction procedure for cyclic free choice workflow nets $\mathcal{W}$

```
1: while 𝒲 is cyclic do
2:     c ← a minimal synchronizer of 𝒲                ▷ If there is none, return
3:     F ← the fragment of c                          ▷ If fragment is malformed, return
4:     while F contains non-synchronizers do
5:         apply the merge rule exhaustively
6:         apply the iteration rule exhaustively
7:         apply the d-shortcut rule to F             ▷ If not possible, return
8:     end while
9:     fix a total order on F
10:    while F is cyclic do
11:        apply the merge rule exhaustively
12:        apply the iteration rule exhaustively
13:        apply the shortcut rule to the backward transition which ends at
           a minimal cluster
14:    end while
15:    apply the iteration rule exhaustively
16: end while
17: while 𝒲 is not reduced completely do
18:    apply the merge rule exhaustively
19:    apply the d-shortcut rule to F                 ▷ If neither was possible, return
20: end while
```

Example 3.10.

We reduce the example net shown in Figure 3.19a. Initially, t_1 unconditionally enables $[t_6]$ and we apply the shortcut rule. Since $[t_6] \cap T = \{t_6\}$, exactly one new transition t_8 is created. Furthermore t_1, s_1 and t_6 are removed (Figure 3.19b).

Next, t_5 unconditionally enables $[t_3]$ and $[t_4]$. We apply the shortcut rule twice and call the result t_9 (Figure 3.19c).

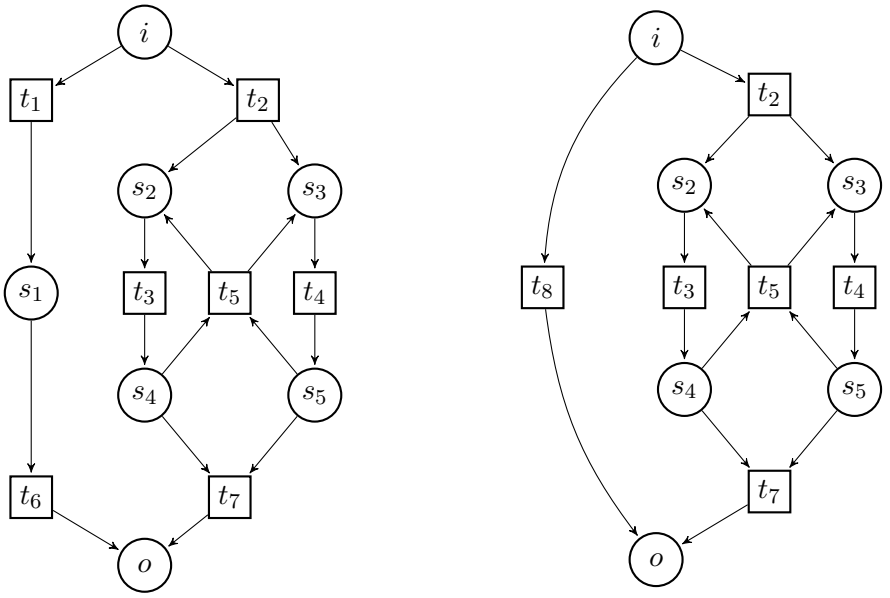
Transition t_9 now satisfies the guard of the iteration rule and can be removed (Figure 3.19d).

Since t_2 unconditionally enables $[t_3]$ and $[t_4]$, we apply the shortcut rule twice and call the result t_{10}. (Figure 3.19e)

After applying the shortcut rule to t_{10}, we apply the merge rule to the two remaining transitions, which yields a net with one single transition (Figure 3.19f).

3.3.2 Computing Synchronizers and Fragments

We now focus on how to find minimal synchronizers and their fragments for a cyclic free choice workflow net. We will split this task into three steps: First we compute a minimal cover of S-components of $\overline{W}$ which by Theorem 2.18 exists if the workflow net is sound. We then use this cover to formulate a linear program for each transition t that has a solution exactly if t is a synchronizer. Finally, we show how to compute the fragment for a given synchronizer again using the minimal S-cover.



(a) A workflow net

(b) The first shortcut

Figure 3.19: Example of reduction

Throughout this subsection, whenever we write $|v|$ for a vector, we mean the 1-norm, i.e. the sum of the entries in the vector.

Computing a minimal S-cover

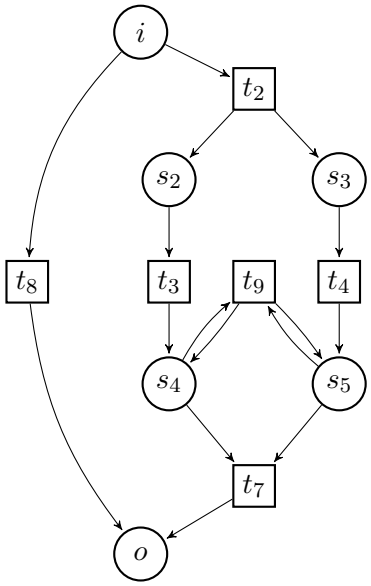
We first compute a minimal cover of S-components of $\overline{W}$ which by Theorem 2.18 exists if the workflow net is sound. A single S-component containing a given place s can be found by solving the following linear program where I is the incidence matrix of the extended net $\overline{W}$ as in Definition 2.23 and v is a vector of length $|S|$:

$$\begin{aligned} &\min |v| \\ &v \cdot I = 0 \\ &v \geq 0 \\ &v_s = 1 \end{aligned}$$

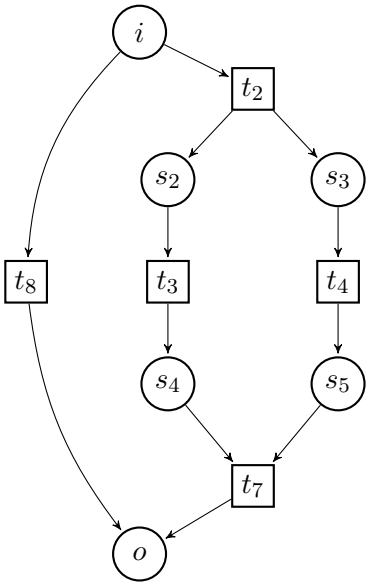
An optimal (minimal) solution v to this linear program is a non-negative minimal S-invariant. Since an S-cover of $\overline{W}$ exists and by Proposition 2.26 an S-component induces a non-negative S-invariant, there must be a solution to the above linear program for every $s \in S$.

According to Proposition 2.27 for a solution v, the net that contains every place s where $v(s) > 0$ together with its pre- and post-transitions is an S-component of $\mathcal{W}$ if $\mathcal{W}$ is a sound free choice net. We simply check whether the result is indeed an S-component, if not we can report that the net is unsound.

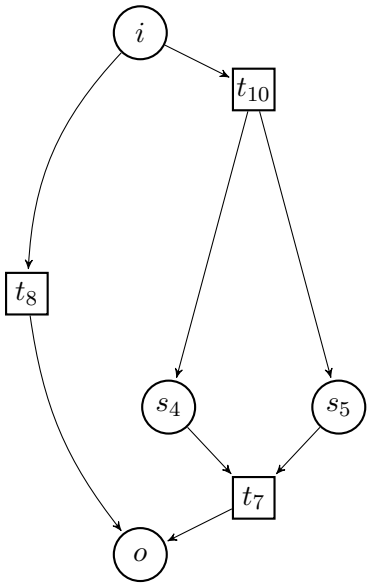
We can now compute a minimal cover of S-components in the following way: While not all places are covered by some S-component, pick a place s that is not yet covered and compute a component containing s. Add this component to the cover. Finally, check if any component in the cover is redundant, i.e. every place in that



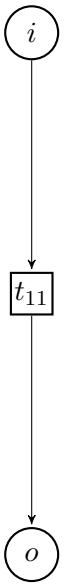
(c) After two more shortcuts



(d) After iteration rule



(e) After two more shortcuts



(f) Final net

Figure 3.19: Example of reduction (continued)

component is also covered by another component. If so, remove that component from the cover.

Finding Synchronizers

Before we give a characterization of synchronizers by means of a linear program, we need an observation regarding synchronizers and S-components.

Lemma 3.30. *Let $\mathcal{W}$ be a sound free choice workflow net, t a synchronizer, σ a loop synchronized by t and C an S-component of $\overline{W}$. If C contains a transition in σ, it also contains t.*

Proof. As σ is a loop, there is a reachable marking M such that $M \xrightarrow{\sigma} M$. Recall that since t is a synchronizer, whenever t is enabled, out of all pre-places of transitions in σ exactly the pre-places of t are marked. Also note that there is exactly one token in C at all times by the definition of S-component and Corollary 2.38. Finally note that the token in C cannot "exit" the loop, i.e. end up on some place that is not a pre-place of a transition in σ as otherwise it would not be possible that $M \xrightarrow{\sigma} M$ because the token would stay there from that point on. Thus when t is enabled, the token of C must be on some pre-place of t and therefore C contains t. □

We now show how to compute synchronizers.

Theorem 3.31. *Let $\mathcal{W}$ be a sound free choice workflow net, $\mathcal{C}$ a minimal cover of S-components of $\overline{W}$. For a transition t, let $U_t = \{t' \mid t' \text{ is contained in an S-component of } \mathcal{C} \text{ which does not contain } t\}$.*

The synchronizers of $\mathcal{W}$ are exactly the transitions for which the following linear program has a solution:

$$\begin{aligned} &\min |v| \\ &I \cdot v = 0 \\ &v \geq 0 \\ &v_t = 1 \\ &v_{t'} = 0 \text{ for all } t' \in U_t \end{aligned}$$

Proof. We first show that for every synchronizer t the linear program above has a solution. Let σ be a minimal loop synchronized by t. Then the vector v that counts for each transition t' the number of occurrences of t' in σ is a (not necessarily optimal) solution to the linear program above. To see that v fulfills the last condition, apply Lemma 3.30.

We now show that if the above linear program has a solution for some t, then t is a synchronizer. A solution v to the linear program is a minimal T-invariant. By Proposition 2.29 this invariant induces a T-component that can be activated by Theorem 2.22. Let M be a marking that activates this T-component. By liveness of the T-component, there is a sequence τ of transitions in the T-component after which t is activated. We fire this sequence and then t. Repeating this argument yields an infinite transition sequence and thus (since the reachable markings are finite) a loop σ containing t and only transitions in the T-component. Since t is contained in all S-components that intersect the T-component, whenever t is activated exactly the pre-places of t are marked inside the T-component, thus σ is synchronized by t, therefore t is a synchronizer. □

For our algorithm we need to find a minimal synchronizer. By Lemma 3.30 we know that for two synchronizers t, t', if the fragment of t contains the fragment of t' then t must be contained in all the S-components that contain t'. Thus if we pick a synchronizer that is contained in as few S-components as possible, this must be a minimal synchronizer.

Computing a fragment

Previously we have computed a minimal cover of S-components and a synchronizer t together with a T-component that contains t. We have also seen that t synchronizes all loops in the T-component, thus that T-component is a part of the fragment of t. We now extend this T-component to obtain the complete fragment of t. We call this T-component our preliminary fragment and proceed as follows:

- Pick some transition t' such that ${}^\bullet t'$ is in the preliminary fragment but t' is not.
- If there is a loop containing t' and only transitions contained in (a subset of) the S-components that contain t, add this loop to the fragment
- Otherwise, exclude t' from further computations.

We can check for loops containing t' using the same linear program as above, but setting $v_{t'} = 1$ instead of v_t.

$$\begin{aligned} &\min |v| \\ &I \cdot v = 0 \\ &0 \leq v \leq 1 \\ &v_{t'} = 1 \\ &v_{t''} = 0 \text{ if } t'' \in U_t \end{aligned}$$

Excluding t' from further computations amounts to setting $v_{t'} = 0$ from that point. If we find another T-invariant, again we know that the T-component corresponding to it can be activated and all transitions will belong to loops synchronized by t. We continue this computation until the preliminary fragment has no more outgoing transitions we have not considered. The result is the fragment synchronized by t.

3.3.3 Runtime Analysis

Due to careful analysis, we can give a polynomial bound on the number of rule applications for Algorithm 3.

Theorem 3.32. *Every sound free choice workflow net can be summarized in at most $\mathcal{O}(|C|^4 + |C|^3 \cdot |T|)$ shortcut rule applications and $\mathcal{O}(|C|^4 + |C|^2 \cdot |T|)$ applications of the merge or iteration rule where C is the set of clusters of the net. Any unsound free choice workflow net can be recognized as unsound in the same number of rule applications.*

Proof. We start by bounding the number of clusters and transitions that may arise during the reduction. Since none of our rules create places or clusters, the number

of clusters in any fragment or the net itself will always be bounded by $|C|$. The number of transitions only increases if we apply the shortcut rule to a cluster with more than one transition. This happens during the reduction of the synchronizer-only fragment. However, the total number of unique transitions (i.e. different pre-set or post-set from every other transition) produced in this way during the whole reduction procedure is bounded by $|C|^2 + |C| \cdot |T|$: at most $|C|^2$ transitions can be produced inside the synchronizer-only fragment, one for each pair of clusters. For transitions leading outside of the fragment, each such transition must originate from some cluster, and since we reduce a synchronizer only fragment, the new transition's post places are exactly the post places of an existing transition, thus at most $|C| \cdot |T|$ unique transitions can be created.

Table 3.1 lists the number of rule applications in the different phases of the algorithm and is explained below.

We first bound the number of rule applications it takes to reduce a minimal fragment with clusters C_F and transitions T_F to a synchronizer only fragment. We already know that $|C_F|$ is bounded by $|C|$ and $|T_F|$ is bounded by $|C|^2 + |C| \cdot |T|$. During this reduction, only the d-shortcut and merge rule are applied. Using the auxiliary net construction from Lemma 3.28 we see that for each synchronizer, an acyclic net with $\mathcal{O}(C_F)$ clusters and $\mathcal{O}(T_F)$ transitions is reduced. By Theorem 3.16 this reduction takes at most $\mathcal{O}(|C_F| \cdot |T_F|) = \mathcal{O}(|C|^3 + |C|^2 \cdot |T|)$ applications of the d-shortcut rule and at most $\mathcal{O}(|T_F|) = \mathcal{O}(|C|^2 + |C| \cdot |T|)$ applications of the merge rule. We also apply the iteration rule to any self-loops formed during this reduction, the number of iteration rule applications is as well bounded by $\mathcal{O}(|T_F|) = \mathcal{O}(|C|^2 + |C| \cdot |T|)$. This reduction takes place once for each synchronizer in $\mathcal{W}$.

Next we bound the time to reduce a synchronizer only fragment with clusters C_F and transitions T_F. Again we bound $|C_F|$ by $|C|$ and $|T_F|$ by $|C|^2 + |C| \cdot |T|$. The auxiliary net defined above is a S-net with $\mathcal{O}(|C_F|)$ clusters. By Corollary 3.21 the reduction reduces the fragment in $\mathcal{O}(|C_F|^2) = \mathcal{O}(|C|^2)$ applications of the shortcut rule and $\mathcal{O}(|C_F|^3 + |T_F|) = \mathcal{O}(|C^3| + |C| \cdot |T|)$ applications of the merge or iteration rule. This reduction takes place once for each fragment in $\mathcal{W}$.

Finally, applying the reduction procedure to an acyclic net $\mathcal{W}_A$ with C_A clusters and T_A transitions takes again $\mathcal{O}(|C_A| \cdot |T_A|) = \mathcal{O}(|C|^3 + |C|^2 \cdot |T|)$ applications of the d-shortcut rule and $\mathcal{O}(|T_A|) = \mathcal{O}(|C|^2 + |C| \cdot |T|)$ applications of the merge rule by Theorem 3.16.

Summing up, we obtain for the shortcut rule

$$\mathcal{O}(|C| \cdot (|C|^3 + |C|^2 \cdot |T|) + |C| \cdot |C|^2 + |C|^3 + |C|^2 \cdot |T|) = \mathcal{O}(|C|^4 + |C|^3 \cdot |T|)$$

rule applications and for the merge or iteration rule

$$\mathcal{O}(|C| \cdot (|C|^2 + |C| \cdot |T|) + |C| \cdot (|C^3| + |C| \cdot |T|) + |C|^2 + |C| \cdot |T|) = \mathcal{O}(|C|^4 + |C|^2 \cdot |T|)$$

rule applications suffice.

□

3.4 Colored Workflow Nets

We now show how to extend the reduction rules defined in the previous sections to colored workflow nets. We aim to completely reduce a CWN and during the

	shortcut	merge/iteration	how often
min. fragment to sync. only, lines 4-8	$\mathcal{O}(\|C\|^3 + \|C\|^2 \cdot \|T\|)$	$\mathcal{O}(\|C\|^2 + \|C\| \cdot \|T\|)$	$\mathcal{O}(\|C\|)$
reduce sync. only, lines 10-14	$\mathcal{O}(\|C\|^2)$	$\mathcal{O}(\|C^3\| + \|C\| \cdot \|T\|)$	$\mathcal{O}(\|C\|)$
acyclic net, lines 17-20	$\mathcal{O}(\|C\|^3 + \|C\|^2 \cdot \|T\|)$	$\mathcal{O}(\|C\|^2 + \|C\| \cdot \|T\|)$	1

Table 3.1: Number of rule applications during Algorithm 3

reduction also compute the summary.

Recall that for colored workflow nets, each token has a color and transitions t have associated transformers $\lambda(t)$ that determine the colors of the tokens produced by the transition depending on the colors of the consumed tokens. The summary of a CWN is the relation that contains for each initial marking M_i the pair (M_i, M_o) if M_o is a final marking reachable from M_i.

First we define the concatenation, union and Kleene star of the transformers $\lambda(t)$ that occur in CWNs. For that, we extend each transformer $\lambda(t)$ to a relation between two colored markings where all tokens except those on ${}^\bullet t \cup t^\bullet$ are unchanged and call $\mathcal{M}$ the set of reachable markings.

$$\begin{aligned}
\lambda(t_1)\lambda(t_2) =& \{(M, M') \in \mathcal{M} \times \mathcal{M} \mid (M, M'') \in \lambda(t_1) \text{ and} \\
& (M'', M') \in \lambda(t_2) \text{ for some marking } M'' \in \mathcal{M}\} \\
\lambda(t_1) \cup \lambda(t_2) =& \{(M, M') \in \mathcal{M} \times \mathcal{M} \mid (M, M') \in \lambda(t_1) \text{ or } (M, M') \in \lambda(t_2)\} \\
\lambda(t)^0 =& \{(M, M) \in \mathcal{M} \times \mathcal{M} \mid M \in \mathcal{M}\} \\
\lambda(t)^{i+1} =& \lambda(t)\lambda(t)^i \text{ for every } i \geq 0 \\
\lambda(t)^* =& \bigcup_{i \geq 0} \lambda(t)^i
\end{aligned}$$

For a transition sequence σ, its transformer is the concatenation of the individual transformers. By definition the summary is the union of all transformers of the firing sequences ending in the final marking.

We state the extended version of the rules. Observe that the only additions to the rules concern the transformer, otherwise the rules remain unchanged.

Definition 3.33. *Merge Rule for CWNs*

Guard: *$\mathcal{W}$ contains two distinct transitions $t_1, t_2 \in T$ such that ${}^\bullet t_1 = {}^\bullet t_2$ and $t_1^\bullet = t_2^\bullet$.*

Action: *(1) $T := (T \setminus \{t_1, t_2\}) \cup \{t_m\}$, where t_m is a fresh name.*

(2) $t_m^\bullet := t_1^\bullet$ and ${}^\bullet t_m := {}^\bullet t_1$.

(3) $\lambda(t_m) := \lambda(t_1) \cup \lambda(t_2)$.

Definition 3.34. *Iteration Rule for CWNs*

Guard: *$\mathcal{W}$ contains a free choice cluster c with a transition $t \in c$ such that $t^\bullet = {}^\bullet t$.*

Action: *(1) $T := (T \setminus \{t\})$.*

(2) For all $t' \in c \cap T$: $\lambda(t') := \lambda(t)^* \lambda(t')$.

Definition 3.35. *Shortcut Rule for CWNs*

Guard: *$\mathcal{W}$ contains a transition t and a free choice cluster $c \notin \{[t], [o]\}$ such that t unconditionally enables c.*

Action: *(1) $T := (T \setminus \{t\}) \cup \{t'_s \mid t' \in c\}$, where t'_s are fresh names.*

(2) For all $t' \in c \cap T$: ${}^\bullet t'_s := {}^\bullet t$ and $t'^\bullet_s := (t^\bullet \setminus {}^\bullet t') \cup t'^\bullet$.

(3) For all $t' \in c \cap T$: $\lambda(t'_s) := \lambda(t)\lambda(t')$.

(4) If ${}^\bullet s = \emptyset$ for all $s \in c \cap S$, then remove c from $\mathcal{W}$.

We extend the definition of correctness to reduction rules for CWNs.

Definition 3.36 (Correctness for CWNs)**.** *A rule R is* correct for CWNs *if its application to a CWN produces an equivalent CWN, that is $\mathcal{W}_1 \xrightarrow{R} \mathcal{W}_2$ implies that $\mathcal{W}_1 \equiv \mathcal{W}_2$.*

Theorem 3.37. *The merge, iteration and shortcut rule for CWNs are correct for CWNs.*

Proof. It was already shown in Theorems 3.3, 3.5 and 3.8 that the rules without the changes to the transformers preserve soundness. We therefore only have to show that they also preserve the summary. For the merge and iteration rule, this is quite obvious from the definition.

For the shortcut rule, we have shown in the proof of Theorem 3.8 that if $\mathcal{W}_1 \xrightarrow{shortcut} \mathcal{W}_2$, then for every initial occurrence sequence that leads to the final marking in $\mathcal{W}_1$, there is a corresponding initial occurrence sequence that leads to the final marking in $\mathcal{W}_2$ and also the other way round. Corresponding sequences were constructed in a way that they contain the same transitions except for the transitions modified by the application of the shortcut rule. For these transitions, every occurrence of the new transition t'_s is replaced by the sequence $t\,t'$ to obtain the corresponding sequence of a sequence in $\mathcal{W}_2$ (the corresponding sequence of a sequence in $\mathcal{W}_1$ is constructed similarly but more complicated for sequences in $\mathcal{W}_1$). Corresponding sequences by construction have the same transformer, thus the summary of $\mathcal{W}_1$ and $\mathcal{W}_2$ must be equal. □

3.5 Probabilistic Nets

Our final aim will be the computation of the expected reward using our reduction rules. To that end, we manipulate the weights and rewards of the transitions when we apply rules so that the reduced net has the same expected value. We then

apply our algorithm and compute the expected value for sound free choice PWNs in polynomial time.

Recall that in PWNs each transition t has an associated weight $w(t)$ and a reward $r(t)$. We will assume that for every free choice cluster (where the conflict set is exactly the cluster), the weights of the transitions in that cluster have been normalized to sum to one. We state the adapted rules:

Definition 3.38. *Merge Rule for PWNs*

Guard: *$\mathcal{W}$ contains two distinct transitions $t_1, t_2 \in T$ such that ${}^\bullet t_1 = {}^\bullet t_2$ and $t_1^\bullet = t_2^\bullet$.*

Action: *(1) $T := (T \setminus \{t_1, t_2\}) \cup \{t_m\}$, where t_m is a fresh name.*

(2) $t_m^\bullet := t_1^\bullet$ and ${}^\bullet t_m := {}^\bullet t_1$.

(3) $r(t_m) := \frac{w(t_1)}{w(t_1)+w(t_2)} \cdot r(t_1) + \frac{w(t_2)}{w(t_1)+w(t_2)} \cdot r(t_2)$.

(4) $w(t_m) = w(t_1) + w(t_2)$.

Definition 3.39. *Iteration Rule for PWNs*

Guard: *$\mathcal{W}$ contains a free-choice cluster c with a transition $t \in c$ such that $t^\bullet = {}^\bullet t$.*

Action: *(1) $T := (T \setminus \{t\})$.*

(2) For all $t' \in c \setminus \{t\}$: $r(t') := \frac{w(t)}{1-w(t)} \cdot r(t) + r(t')$

(3) For all $t' \in c \setminus \{t\}$: $w(t') := \frac{w(t')}{1-w(t)}$

Definition 3.40. *Shortcut Rule for PWNs*

Guard: *$\mathcal{W}$ contains a transition t and a free-choice cluster $c \neq [t]$ such that t unconditionally enables c.*

Action: *(1) $T := (T \setminus \{t\}) \cup \{t'_s \mid t' \in c\}$, where t'_s are fresh names.*

(2) For all $t' \in c$: ${}^\bullet t'_s := {}^\bullet t$ and $t'^\bullet_s := (t^\bullet \setminus {}^\bullet t') \cup t'^\bullet$.

(3) For all $t' \in c$: $r(t'_s) := r(t) + r(t')$.

(4) For all $t' \in c$: $w(t'_s) = w(t) \cdot w(t')$.

(5) If ${}^\bullet p = \emptyset$ for all $p \in c$, then remove c from $\mathcal{W}$.

Definition 3.41 (Correctness for PWNs)**.** *A rule R is* correct for 1-safe confusion-free PWNs *if its application to a 1-safe confusion-free PWN produces an equivalent 1-safe confusion-free PWN, that is $\mathcal{W}_1 \xrightarrow{R} \mathcal{W}_2$ implies that $\mathcal{W}_1 \equiv \mathcal{W}_2$.*

Theorem 3.42. *The merge, shortcut and iteration rules for PWNs are correct for 1-safe confusion-free PWNs.*

Proof. It was already shown that the rules preserve soundness for workflow nets. We thus only have to show that the rules preserve the expected reward of the net.

We first focus on the unsound case.

Let $\mathcal{W}$ be an unsound 1-safe confusion-free PWN. Then $\overline{\mathcal{W}}$ is either unbounded or not live by Theorem 2.33. Since $\mathcal{W}$ is 1-safe, $\overline{\mathcal{W}}$ must be non-live. This means

there is a marking M reachable from the initial marking such that some transition t can never occur in any transition sequence starting from M. We distinguish two cases.

(1) $M \neq \boldsymbol{i}$, i.e. every transition occurs in some occurrence sequence starting in the initial marking. Then it is clear that from M the initial marking cannot be reachable, and therefore the same holds for the final marking. Let σ be a sequence that ends in M starting from the initial marking. Since all weights are positive, the cylinder of paths that extend the path π_σ has positive probability. It furthermore has infinite reward because no path ever reaches the final marking, thus the PWN has an infinite expected value. By inspection of our rules, the merge and iteration rule do not change the reachable markings and while the shortcut may remove the marking M from the marking graph, some marking M' reachable from M will still be reachable. Therefore the expected reward of the resulting net after the rule application is still infinite and was preserved by the rules.

(2) $M = i$. That means some transition t does not occur in any occurrence sequence starting from the initial marking. By definition of the MDP of $\mathcal{W}$ and the reward of an MDP, if t never occurs in any occurrence sequence it has no effect on the expected reward. If we remove all such transitions t and the net is still unsound, the expected reward is infinite by (1) and will remain infinite. If the net is sound after the removal, the expected reward is the same for $\mathcal{W}$ and we argue below why the rules preserve the expected reward in that case.

We now focus on the sound case.

By Theorem 2.58 the expected reward of the net does not depend on the scheduler. We use this fact in the following way: For each rule, we pick two schedulers, one for the net before the rule application and one for the net after the rule was applied. These schedulers will be such that it is easy to show that their expected rewards are equal. We begin with the shortcut rule.

Shortcut rule. Let $\mathcal{W}_1, \mathcal{W}_2$ be such that $\mathcal{W}_1 \overset{\text{shortcut}}{\longrightarrow} \mathcal{W}_2$. Let c, t be as in Definition 3.40. Let S_1 be a scheduler for W_1 such that $S_1(\sigma_1) = c$ if σ_1 ends with t. Since t unconditionally enables c, this is a valid scheduler.

We define a mapping ϕ that maps firing sequences in $\mathcal{W}_2$ to firing sequences in $\mathcal{W}_1$ by replacing every occurrence of t'_s by $t\,t'$. Next we define a scheduler S_2 for $\mathcal{W}_2$ by $S_2(\sigma_2) = S_1(\phi(\sigma_2))$.

Observe that ϕ is a bijection between sequences produced by S_1 that do not end with t and sequences produced by S_2. In particular ϕ is a bijection between sequences produced by S_1 and S_2 that end with the final marking.

Let now σ_2 be a firing sequence in $\mathcal{W}_2$ and let $\sigma_1 = \phi(\sigma_2)$. We claim that σ_1 and σ_2 have the same reward and also $\nu_{S_1}(\sigma_1) = \nu_{S_2}(\sigma_2)$. Indeed, since the only difference is that every occurrence of t'_s is replaced by $t\,t'$ and $r(t'_s) = r(t) + r(t')$ and $w(t'_s) = w(t)w(t')$ by the definition of the shortcut rule, the reward must be equal and $\nu_{S_1}(\sigma_1) = \nu_{S_2}(\sigma_2)$.

We now use these equalities, the fact that ϕ is a bijection between firing sequences that end with the final marking, and Lemma 2.52:

$$
\begin{aligned}
V(\mathcal{W}_2) &\overset{Lemma\ 2.52}{=} \sum_{\sigma_2 \in Fin_{\mathcal{W}_2}} r(\sigma_2) \cdot \nu_{S_2}(\sigma) \overset{\text{Claim}}{=} \sum_{\sigma_2 \in Fin_{\mathcal{W}_2}} r(\phi(\sigma_2)) \cdot \nu_{S_1}(\phi(\sigma_2)) \\
&\overset{\phi \text{ bij.}}{=} \sum_{\sigma_1 \in Fin_{\mathcal{W}_1}} r(\sigma_1) \cdot \nu_{S_1}(\sigma_1) = V(\mathcal{W}_1)\ .
\end{aligned}
$$

Iteration rule. Let $\mathcal{W}_1$, $\mathcal{W}_2$ be such that $\mathcal{W}_1 \stackrel{\text{iteration}}{\longrightarrow} \mathcal{W}_2$. Let c, t be as in Definition 3.39. Let S_2 be a scheduler for W_2 such that $S_2(\sigma_2) = c$ if c is enabled after σ_2.

We define a mapping ϕ that maps firing sequences in $\mathcal{W}_1$ to firing sequences in $\mathcal{W}_2$ by removing all occurrences of t. Next we define a scheduler S_1 for $\mathcal{W}_1$ by $S_1(\sigma_1) = S_2(\phi(\sigma_1))$. Note that ϕ is not a bijection but it is surjective.

Let r_1 and r_2 be the reward functions of $\mathcal{W}_1$ and $\mathcal{W}_2$. For a sequence σ_2 in $\mathcal{W}_2$, we claim:

$$r_2(\sigma_2) \cdot \nu_{S_2}(\sigma_2) = \sum_{\sigma_1 \in \phi^{-1}(\sigma_2)} r_1(\sigma_1) \cdot \nu_{S_1}(\sigma_1) \ .$$

Let k be the number of times c is enabled during σ_2. We only consider the case $k = 1$, the general case being similar. We observe that σ_2 is also a sequence in $\mathcal{W}_1$. We have

$$\nu_{S_1}(\sigma_2) = \nu_{S_2}(\sigma_2) \cdot (1 - w(t)) \tag{3.1}$$

$$r_1(\sigma_2) = r_2(\sigma_2) - \frac{w(t)}{1 - w(t)} \cdot c(t) \tag{3.2}$$

because the probabilistic choice must pick something other than t, and because the iteration rule adds $\frac{w(t)}{1-w(t)} \cdot c(t)$ to the reward of every transition in c in $\mathcal{W}_2$.

We now insert l occurrences of t in σ_2, at the position at which c is enabled, and call the new sequence τ_l. We have $\phi^{-1}(\sigma_2) = \{\tau_l \mid l \geq 0\}$. Further $r_1(\tau_l) = r_1(\sigma_2) + l \cdot c(t)$ and $\nu_{S_1}(\tau_l) = \nu_{S_1}(\tau) \cdot w(t)^l$, and so summing over all l we get:

$$\begin{aligned}
\sum_{\sigma_1 \in \phi^{-1}(\sigma_2)} r_1(\sigma_1) \cdot \nu_{S_1}(\sigma_1) &= \sum_{l=0}^{\infty} r_1(\tau_l) \cdot \nu_{S_1}(\tau_l) \\
&= \nu_{S_1}(\sigma_2) \cdot \sum_{l=0}^{\infty} (r_1(\sigma_2) + l \cdot c(t)) \cdot w(t)^l \\
&= \nu_{S_1}(\sigma_2) \cdot \left(\frac{r_1(\sigma_2)}{1 - w(t)} + \frac{c(t) \cdot w(t)}{(1 - w(t))^2} \right) \\
&= \nu_{S_2}(\sigma_2) \cdot \left(r_1(\sigma_2) + \frac{c(t) \cdot w(t)}{1 - w(t)} \right) \qquad \text{(by 3.1)} \\
&= \nu_{S_2}(\sigma_2) \cdot r_2(\sigma_2) \qquad \text{(by 3.2)}
\end{aligned}$$

and the claim is proven.

Now, using the claim we obtain:

$$\begin{aligned}
V(\mathcal{W}_2) &= \sum_{\sigma_2 \in Fin_{\mathcal{W}_2}} r_2(\sigma_2) \cdot \nu_{S_2}(\sigma) = \sum_{\sigma_2 \in Fin_{\mathcal{W}_2}} \sum_{\sigma_1 \in \phi^{-1}(\sigma_2)} r_1(\sigma_1) \cdot \nu_{S_1}(\sigma_1) \\
&= \sum_{\sigma_1 \in Fin_{\mathcal{W}_1}} r_1(\sigma_1) \cdot \nu_{S_1}(\sigma_1) = V(\mathcal{W}_1)
\end{aligned}$$

where the third equality follows from the fact that ϕ is defined on all sequences of $\mathcal{W}_1$ and thus ϕ^{-1} hits every sequence in $\mathcal{W}_1$ exactly once.

Merge rule. Let $\mathcal{W}_1$, $\mathcal{W}_2$ be such that $\mathcal{W}_1 \stackrel{\text{merge}}{\longrightarrow} \mathcal{W}_2$. Let t_1, t_2 be as in Definition 3.38. Let S_2 be a scheduler for $\mathcal{W}_2$.

We define a mapping ϕ that maps firing sequences in $\mathcal{W}_1$ to firing sequences in $\mathcal{W}_2$ by replacing all occurrences of t_1 and t_2 by t_m. We define a scheduler S_1 for $\mathcal{W}_1$ by $S_1(\sigma_1) = S_2(\phi(\sigma_1))$.

Once again, ϕ is a surjective function. For a sequence σ_2 in $\mathcal{W}_2$, we claim that

$$r(\sigma_2) \cdot \nu_{S_2}(\sigma_2) = \sum_{\sigma_1 \in \phi^{-1}(\sigma_2)} r(\sigma_1) \cdot \nu_{S_1}(\sigma_1).$$

Indeed, every sequence σ_1 the set $\phi^{-1}(\sigma_2)$ can be obtained by replacing t_m by either t_1 or t_2. So, by Definition 3.38, the sums are equal.

As for the iteration rule, this equality and the fact that ϕ is defined for every sequence in $\mathcal{W}_1$ imply that the expected rewards of $\mathcal{W}_1$ and $\mathcal{W}_2$ are equal. □

As the rules are correct for 1-safe confusion-free PWNs and complete for free choice workflow nets, we can compute the expected value of a 1-safe free choice PWN using Algorithm 3.

Example 3.11.

We illustrate a complete reduction by reducing the example PWN shown in Figure 3.20a. We set the reward for each transition to 1, so the expected reward of the net is the expected number of transition firings until the final marking is reached. In the next figures, transitions will have labels (p, r) which indicate their weight p and reward r.

Initially, t_1 unconditionally enables $[t_6]$ and we apply the shortcut rule. Since $[t_6] = \{s_1, t_6\}$, exactly one new transition t_8 is created. The new transition has reward 2 and weight $\frac{2}{5}$. Furthermore t_1, s_1 and t_6 are removed (Figure 3.20b).

Next, t_5 unconditionally enables $[t_3]$ and $[t_4]$. We apply the shortcut rule twice and call the result t_9 (Figure 3.20c) which has weight $\frac{1}{2}$ and reward 3.

Transition t_9 now satisfies the guard of the iteration rule and can be removed, changing the label of t_7 to $(1, 4)$ (Figure 3.20d).

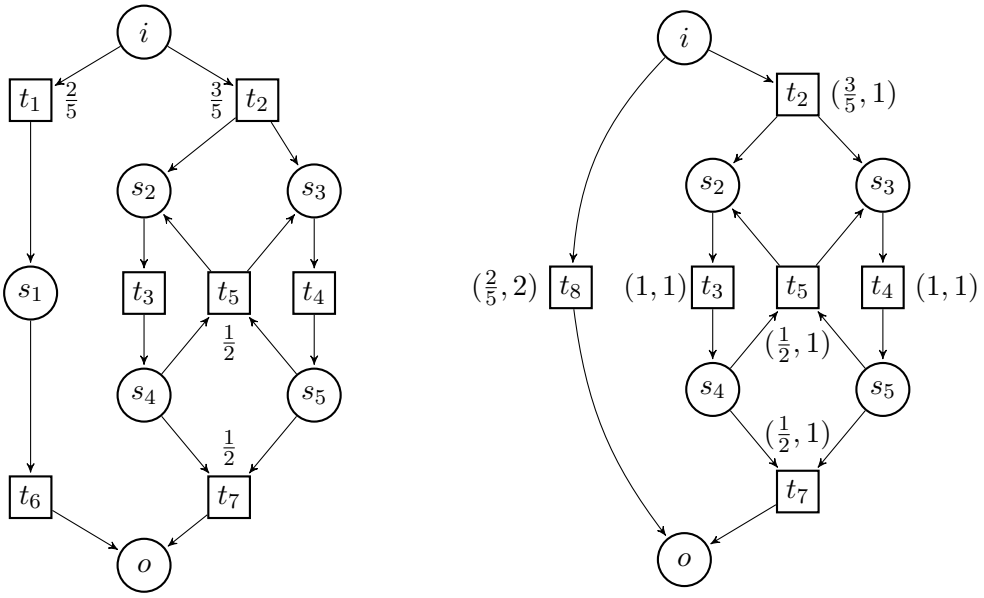
Since t_2 unconditionally enables $[t_3]$ and $[t_4]$, we apply the shortcut rule twice and call the result t_{10}. (Figure 3.20e)

After applying the shortcut rule to t_{10}, we apply the merge rule to the two remaining transitions, which yields a net with one single transition labeled by $(1, 5)$ (Figure 3.20f). So the net terminates with probability 1 after firing 5 transitions on average.

3.6 Implementation and Experimental Results

The algorithm described above has been implemented including the extension for probabilistic workflow nets. In this section, we report on our findings regarding the performance of our algorithm. We use four existing tool as comparison to our tool: LoLA [47], an explicit tool which uses state exploration methods, Woflan [43], which uses structural reduction and S-coverability analysis as well as explicit methods, SESE [41], an approach based on structural reduction and heuristics combined with state exploration and PRISM [30], an explicit state space exploration tool for Markov chains and MDPs.

In a first step, we apply the algorithm purely to check soundness and compare our performance with the performance reported in [20]. We find that the performance



(a) A PWN (b) The first shortcut

Figure 3.20: Example of reduction

of our algorithm is comparable to the reported performance of Woflan, LoLA and SESE.

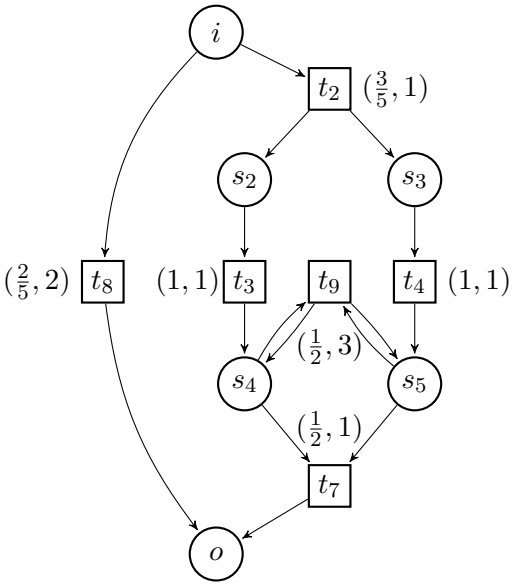
In a second step, we add probabilities to the benchmark nets and evaluate the performance of our algorithm in comparison with PRISM. As expected, our polynomial algorithm for free-choice workflow nets outperforms PRISM's exponential, but more generally applicable algorithm.

All experiments were carried out on an 3.60 GHz i7-3820 CPU using 1 GB of memory.

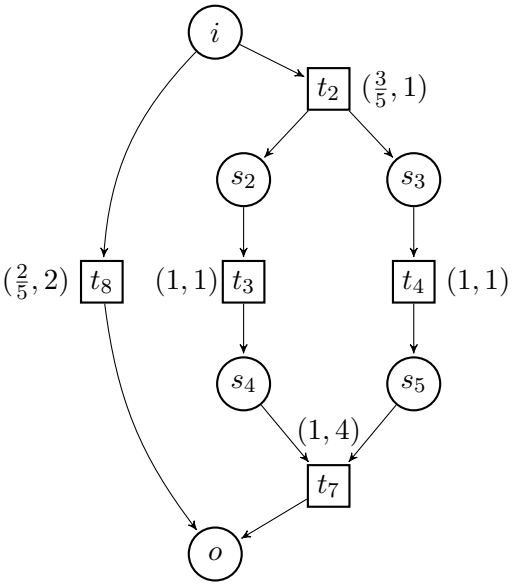
Industrial Benchmarks. We used 1385 free-choice workflow nets previously studied in [20], of which 642 nets are sound. The workflows were obtained from industrial business models designed at IBM. In [20] it was reported that checking soundness for each net in the library takes around 6.5 seconds with Woflan on a 1.66 GHz CPU, around 12 seconds with LoLA on a 2.16 GHz CPU with 11 nets timing out, and around 6.5 seconds with SESE on a 2GHz CPU. Our algorithm took 3.3 seconds to analyze the library.

As the workflow nets provided do not contain probabilistic information, we assigned to each transition t the probability $\frac{1}{|[t]|\cap T}$ (i.e., the probability is distributed uniformly among the transitions of a cluster). We study the following questions, which can be answered by both our algorithm and PRISM: Is the probability to reach the final marking equal to one (equivalent to "is the net sound?"). And if so, how many transitions must be fired on average to reach the final marking? (This corresponds to a reward function assigning reward 1 to each transition.)

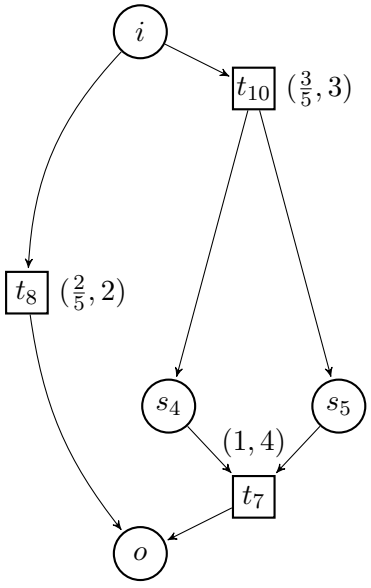
PRISM has three different analysis engines able to compute expected rewards: explicit, sparse and symbolic (bdd). In a preliminary experiment with a timeout of 30 seconds, we observed that the explicit engine clearly outperforms the other two: It solved 1309 cases, while the bdd and sparse engines only solved 636 and 638 cases, respectively. Moreover, 418 and 423 of the unsolved cases were due to



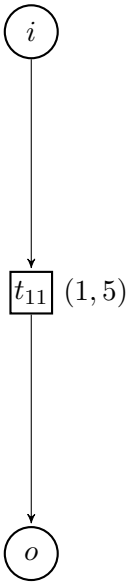
(c) After two more shortcuts



(d) After iteration rule



(e) After two more shortcuts



(f) Final net

Figure 3.20: Example of reduction (continued)

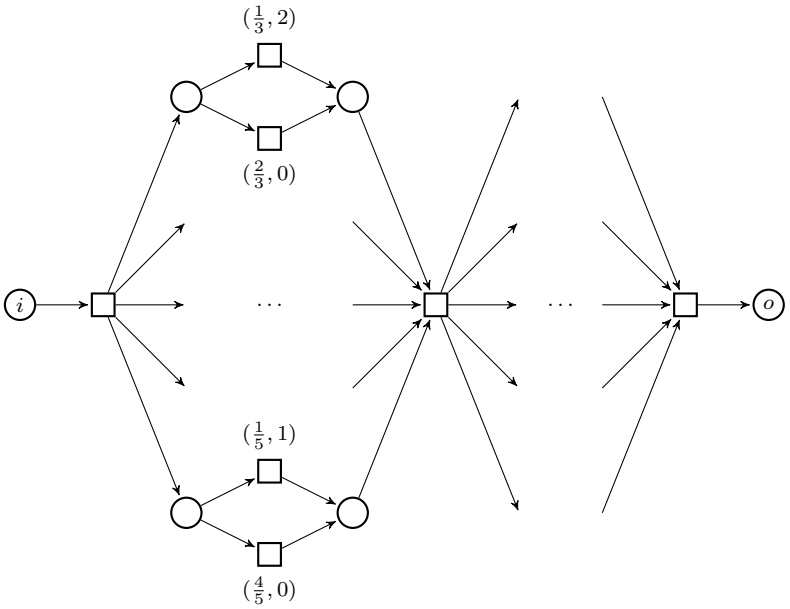


Figure 3.21: The academic benchmark

memory overflow, so even with a larger timeout the explicit engine is still leading. For this reason, in the comparison we only used the explicit engine.

After increasing the timeout to 10 minutes, the explicit engine did not solve any further case, leaving 76 cases unsolved. This was due to the large state space of the nets: 69 out of the 76 cases have over 10^6 reachable states.

The 1309 cases were solved by the explicit engine in 353 seconds, with about 10 seconds for the larger nets. Our implementation solved all 1385 cases in 3.3 seconds combined. It never needs more than 50 ms for a single net, even for those with more than 10^7 states (for these nets we do not know the exact number of reachable states).

In the unsound case, our implementation still reduces the reachable state space by at least an order of magnitude which makes it easier to apply state exploration tools for other problems than the expected reward, like the distribution of the rewards. After reduction, the 69 nets that initially had at least 10^6 states were reduced to an average of 5950 states, with the largest at 313443 reachable states.

An academic benchmark. Many workflows in our suite have a large state space because of fragments modeling the following situation: multiple processes do a computation step in parallel, after which they synchronize. Process i may execute its step normally with probability p_i, or a failure may occur with probability $1 - p_i$, which requires to take a recovery action and therefore has a higher cost. Such a scenario is modeled by the free-choice PWNs net of Figure 3.21, where the probabilities and costs are chosen at random. The scenario can also be easily modeled in Prism. Figure 3.22 shows the time needed by the three Prism engines and by our implementation for computing the expected reward using a time limit of 10 minutes. The number of reachable states grows exponentially in the number processes, and the explicit engine runs out of memory for 15 processes. Since the failure probabilities vary between the processes, there is little structure that the symbolic engine can exploit, and it times out for 13 processes. The sparse engine reaches the time limit

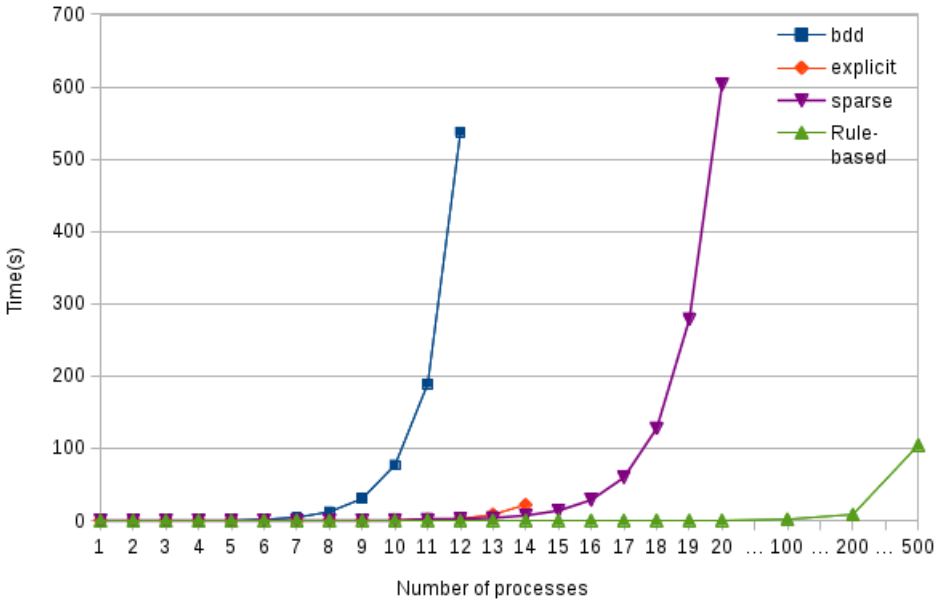


Figure 3.22: Runtimes for the academic benchmark

at 20 processes. However, since the rule-based approach does not need to construct the state space, we can easily solve the problem with up to 500 processes.

Chapter 4

Games on Workflow Nets

Contents

4.1 Introduction

In this chapter we study games on workflow nets. We introduce two players with conflicting goals, one wanting the workflow to reach the final marking while the other tries to prevent the final marking. We study games where control over the clusters is divided between the players, each deciding which transition inside a cluster is fired when a scheduler picks that cluster. Intuitively, one can think of a workflow involving different teams in a company. One of the teams has a lot of inexperienced members and is likely to fail tasks it is involved in. The question we try to answer is whether this team alone (which only controls some of the decisions made during the execution of the workflow) has the power to prevent the workflow from reaching its designated end.

We investigate the general complexity of solving games in the size of the workflow net and aim to find algorithms that decide the winner in polynomial time. Our first result is that these games are EXPTIME-complete in general, even if the workflow net is free choice, and it seems that a polynomial algorithm is out of reach. However, in the case that the workflow net is sound and free choice, we are able to show that the winner can be decided in polynomial time.

This chapter is based on earlier work regarding Negotiation Games [25].

4.2 Analysis Problems

We study a setting of games played on *workflow arenas*, a workflow net where the clusters have been partitioned into two sets $\mathcal{C}_1$ and $\mathcal{C}_2$. We consider a concurrent game with three players, Player 1, Player 2 and Scheduler. In each step, Scheduler chooses a subset of the clusters enabled at the current marking. For each cluster chosen, the player who was assigned that cluster chooses one enabled transition of that cluster which is fired. This choice is done simultaneously and independently of the choices the other player makes in this step. The game terminates if a marking is reached that enables no transition, otherwise it continues forever.

Formally, a partial play is a sequence of tuples $(S_i, F_{1,i}, F_{2,i})$ where S_i is a set of clusters and $F_{1,i}$, $F_{2,i}$ are functions that assign to each cluster c in $S_i \cap \mathcal{C}_1$ and $S_i \cap \mathcal{C}_2$, respectively, a transition in c. Furthermore it must hold that each cluster in S_i contains an enabled transition after the transitions chosen by $F_{1,j}$, $F_{2,j}$ have occurred for all $j < i$, and the transitions chosen by $F_{1,i}$ and $F_{2,i}$ each must be one of those enabled transitions.

A partial play is a play if it is either infinite or ends in a marking that enables no transitions. For a play π, we denote by π_i the prefix of π of length i. In the *termination game*, Player 1 wins if the play ends in the final marking, otherwise Player 2 wins.

A strategy σ_j for Player j, $j \in \{1, 2\}$, is a partial function that, given a partial play $(S_i, F_{1,i}, F_{2,i}), \ldots, (S_i, F_{1,i}, F_{2,i})$ and a set S_{i+1} returns an assignment $F_{j,i+1}$ according to the specifications above. A play is said to be played according to σ_j if $F_{j,i} = \sigma_j(\pi_{i-1}, S_i)$ for all i. A strategy σ_j is winning for Player j if he wins every play that is played according to σ_j. Player j is said to win the game if he has a

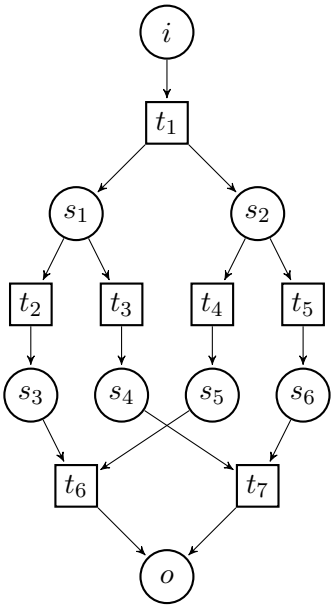


Figure 4.1: Scheduler influences who wins this game

winning strategy. Notice that if Player j has a winning strategy then he wins against every pair of strategies of Scheduler and the other player.

Definition 4.1. *Let $\mathcal{W}$ be a workflow net, $\mathcal{C}_1$ and $\mathcal{C}_2$ the partition of the clusters. The* termination problem *is the problem of deciding whether Player 1 has a winning strategy for the termination game. The* non-termination problem *is the problem of deciding whether Player 2 has a winning strategy for the termination game.*

Notice that the termination problem and the non-termination problem are not dual as Scheduler may influence the game by his choice of the clusters.

Example 4.1.

Consider Figure 4.1. We partition the clusters so that $[t_2]$ is controlled by Player 1, all other clusters are controlled by Player 2. The game ends if either t_2 and t_4 are fired, or t_3 and t_5 are fired. Thus Player 1 needs to "mirror the choice" of Player 2 in a certain sense. However, if Scheduler picks $[t_2]$ before $[t_4]$, Player 1 has no chance to win. Conversely, if Scheduler picks $[t_4]$ first, Player 1 can always win. If Scheduler picks both clusters together, neither of the players has a deterministic winning strategy but picking randomly gives a winning probability of 0.5 to both players.

We first investigate the general complexity of the problem.

Theorem 4.2. *The termination problem and the non-termination problem are in EXPTIME.*

For the proof we need some definitions and results from [7].
A concurrent game structure is a tuple $G = (k, Q, \Pi, \pi, d, \delta)$ where

- $k \geq 1$ is a natural number, the number of players.

- Q is a finite set of states.
- Π is a finite set of propositions.
- π assigns every state $q \in Q$ a set of propositions that are true in q.
- d assigns every player $a \in \{1, ..., k\}$ and every state $q \in Q$ a natural number $d_a(q) \geq 1$ of possible moves. We identify these moves with natural numbers $1, ..., d_a(q)$. For each state q, we write $D(q)$ for the set $\prod_{i=1}^{k} \{1, ..., d_i(q)\}$ of move vectors.
- δ is the transition function that assigns a state $q \in Q$ and a move vector $(j_1, ..., j_k) \in D(q)$ a state $\delta(q, j_1, ..., j_k)$ that results from state q if every player $a \in \{1, ..., k\}$ chooses move j_a.

The following is one of the results from [7]:

Theorem 4.3. *Determining whether a player can force reaching a certain set of states or prevent the game from reaching a certain set of states on a concurrent game structure is possible in time linear in the number of transitions.*

We now give the proof of Theorem 4.2.

Proof. We construct a concurrent reachability game of single exponential size such that Player 1 (Player 2) wins if Player 1 (Player 2) has a winning strategy in the termination game. By Theorem 4.3, our result follows.

The states of the game are either markings M of the workflow net, or pairs (M, C_M), where M is a marking and C_M is a set of clusters enabled at M. Nodes M belong to Scheduler, who chooses a set C_M, after which the play moves to (M, C_M). At nodes (M, C_M) Players 1 and 2 concurrently select transitions for the clusters C_M, and depending on their choice the play moves to a new marking.

The states of the concurrent game structure are the following:

$$\begin{aligned} Q = \quad & \{M : M \text{ a reachable marking}\} \cup \{(M, C_M) : M \text{ a reachable marking}, \\ & C_M \text{ a set of clusters with at least one enabled transition } M\} \end{aligned}$$

Only one proposition is needed to mark the final state:

$$\Pi = \{\text{final}\}$$

$$\pi(M) = \begin{cases} \text{final} & \text{if } M = \boldsymbol{o} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\pi((M, C_M)) = \emptyset$$

The move vectors, given by the Cartesian product of the possible moves of each player, are defined as follows:

- $D(M) = \{1\} \times \{1\} \times 2^S$ is the set of move vectors from state M, where S is the set of all clusters containing a transition enabled at M.
- $D((M, C_M)) = \{F_1\} \times \{F_2\} \times \{1\}$ is the set of move vectors from state (M, C_M), where F_i is the set of all functions assigning each $c \in C_M \cap \mathcal{C}_i$ an enabled transition $t \in c$.

The transition function δ is defined by $\delta(M, (1, 1, C_M)) = (M, C_M)$, and $\delta((M, C_M), (f_1, f_2, 1)) = M'$, where M' is the marking reached from M after all clusters of C_M occur with the transitions specified by f_1, f_2. Notice that M' does not depend on the order in which they occur.

Player 1 wins if the play reaches the final marking M_f. Player 2 wins if the game never reaches the final marking. Using Theorem 4.3, the result follows. □

While Theorem 4.2 gives only an upper bound for the complexity, unfortunately the lower bound matches the upper bound and makes both the termination and the non-termination problem EXPTIME-complete.

Theorem 4.4. *The termination problem and the non-termination problem are EXPTIME-hard even for free choice workflow arenas, and even for arenas where Scheduler never has any choice.*

Proof. We reduce the acceptance problem of linearly bounded alternating Turing machines (TM) [9] to a workflow game. Recall that an alternating Turing machine has four types of states, existential and universal, accepting and rejecting. Acceptance for existential and universal states can then be defined as follows: An existential state is accepting if there is a transition that leads to an accepting state, a universal state is accepting if all transitions lead to an accepting state. The TM accepts if its initial state is accepting.

In the workflow game we construct, there will always be at most one cluster that is enabled. Therefore Scheduler never has any choice and the termination problem and non-termination problem are dual to each other and EXPTIME-hard.

We are given an alternating TM A with transition relation δ, and an input x of length n. We assume that A always halts in one of two designated states q_{accept} or q_{reject}, and does so immediately after reaching one of those states.

To ease understanding, we split the construction into two parts: First we construct a workflow net which is not free choice, but contains the core idea of how to simulate the TM nevertheless. However, due to the construction, a large part of the workflow net consists of a single cluster and it will not be possible to model universal and existential quantifiers. In the second part we modify the construction so that the resulting net is free choice and show how to assign the clusters to the players.

We first describe the initial construction informally. We define a workflow net which will, aside from the initial and final marking, use two tokens to model the head position and internal state of M, and one token for each cell describing its content. Aside from the input place i and the output place o, there will be

- one place per internal state of M
- one place per possible head position, i.e. per cell on the tape
- one place for each alphabet symbol and cell.

There will be one transition in the workflow net for each transition $(q, \alpha) \to (q', \alpha', D)$ of M and each possible head position, where q, q' are control states, α, α' are letters and D is the direction that the head moves, either "R", "L" or "N" (right, left, no movement). The construction is shown in Figure 4.2. Intuitively, we move the three tokens that represent the head, the current cell content and the internal state according to the transition of the Turing machine.

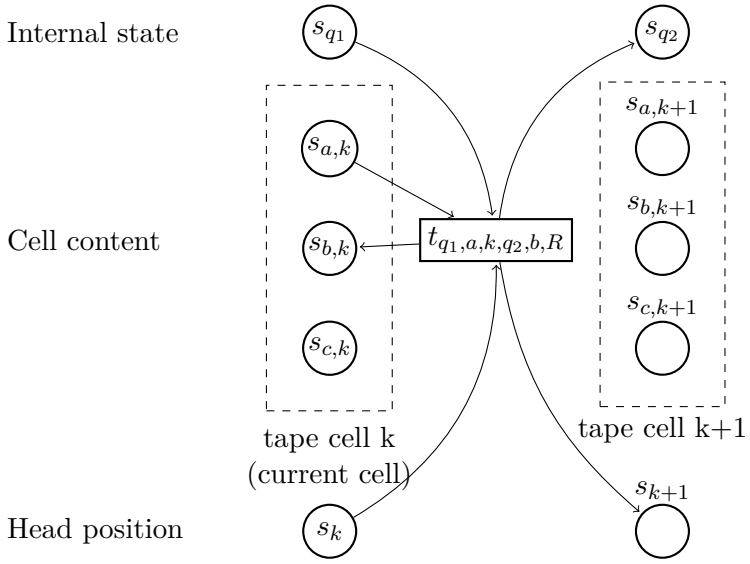


Figure 4.2: Part of the workflow net representing a TM, the transition drawn represents the transition $(q_1, a) \to (q_2, b, R)$

Finally, the workflow net also has an initial transition that, loosely speaking, takes care of modeling the initial configuration, and a block of final transitions that "clean up" after q_{accept} has been reached as internal state.

Since every transition that actually simulates the TM will have the internal state in its pre-set (and there is only one internal state at all times), all enabled transitions will be in conflict with each other. However, there is only a single cluster containing all transitions simulating the TM.

Formally, we are given an alternating TM $M = (Q, \Gamma, \delta, q_0, g)$ with tape alphabet Γ and a function $g\colon Q \to \{\wedge, \vee, \text{accept}, \text{reject}\}$ that determines the state type, and an input x of length n. We assume that M always halts, does so in designated states q_{accept} or q_{reject}, and does so immediately when reaching one of those. We define a workflow game $\mathcal{W}_{M,x}$ as follows:

- The set of places is

$$\begin{aligned} S = &\{i, o\} \cup \{s_q \mid q \in Q\} \cup \{s_k \mid 1 \le k \le n\} \cup \\ &\{s_{\alpha,k} \mid \alpha \in \Gamma, 1 \le k \le n\} \cup \{s_{clean,k} \mid 1 \le k \le n\} \end{aligned}$$

  s_q will model the current state, s_k the head position, $s_{\alpha,k}$ the content of cell k and $s_{clean,k}$ will be used to clean up.

- The set of transitions is

$$\begin{aligned} T = &\{t_i\} \cup \{t_{clean,k} \mid 1 \le k \le n\} \cup \{t_{clean,\alpha,k} \mid \alpha \in \Gamma, 1 \le k \le n\} \cup \\ &\{t_{q,\alpha,k,q',\alpha',R} \mid (q, \alpha) \to (q', \alpha', R) \in \delta, 1 \le k < n\} \cup \\ &\{t_{q,\alpha,k,q',\alpha',L} \mid (q, \alpha) \to (q', \alpha', L) \in \delta, 1 < k \le n\} \cup \\ &\{t_{q,\alpha,k,q',\alpha',N} \mid (q, \alpha) \to (q', \alpha', N) \in \delta, 1 \le k \le n\} \end{aligned}$$

  t_i initializes all cells, $t_{q,\alpha,k,q',\alpha',D}$ models a transition of the TM, $t_{clean,k}$ and $t_{clean,\alpha,k}$ are used to clean up.

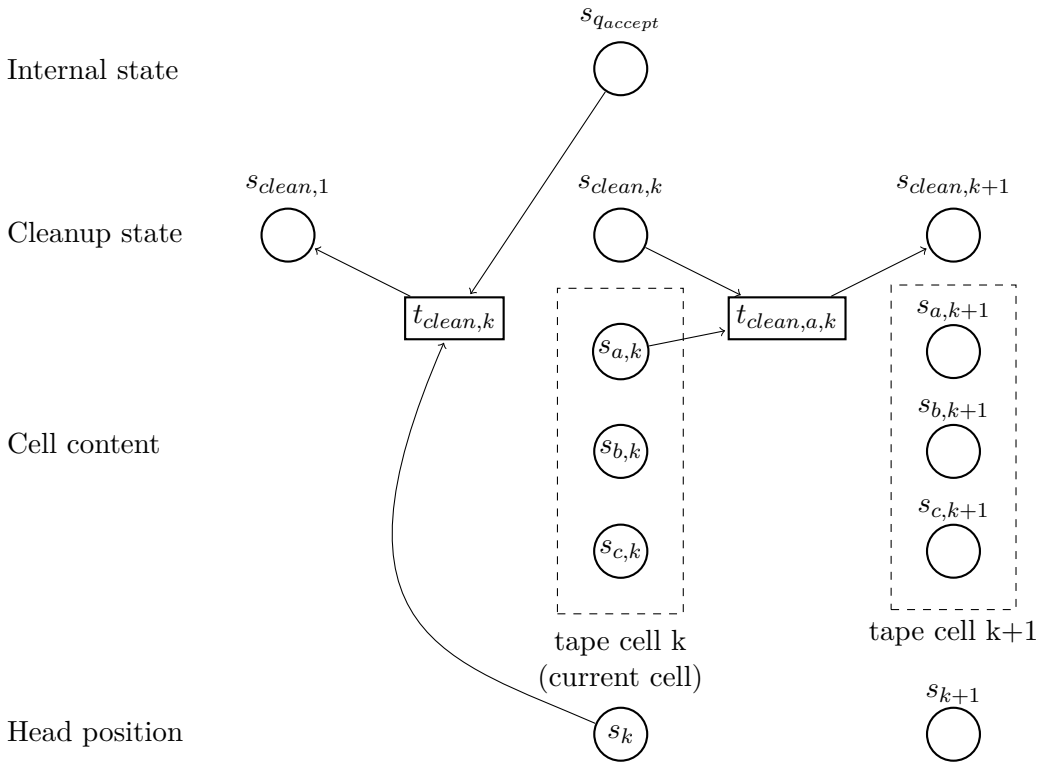


Figure 4.3: Two transitions of the cleanup procedure

- The pre- and post-sets are

$$
\begin{aligned}
{}^\bullet t_i &= \{i\} \\
t_i^\bullet &= \{s_{q_0}, s_1\} \cup \{s_{\alpha,k} \mid \text{ cell } k \text{ initially contains } \alpha\} \\
{}^\bullet t_{q,\alpha,k,q',\alpha',D} &= \{s_q, s_{\alpha,k}, s_k\} \\
t^\bullet_{q,\alpha,k,q',\alpha',D} &= \{s_{q'}, s_{\alpha',k}\} \cup \begin{cases} \{s_{k+1}\} & \text{if } D = R \\ \{s_{k-1}\} & \text{if } D = L \\ \{s_k\} & \text{if } D = N \end{cases} \\
{}^\bullet t_{clean,k} &= \{s_{q_{accept}}, s_k\} \\
t^\bullet_{clean,k} &= \{s_{clean,1}\} \\
{}^\bullet t_{clean,\alpha,k} &= \{s_{clean,k}, s_{\alpha,k}\} \\
t^\bullet_{clean,\alpha,k} &= \begin{cases} \{s_{clean,k+1}\} & \text{if } k < n \\ \{o\} & \text{otherwise} \end{cases}
\end{aligned}
$$

- The set C_1 of conflict sets owned by Player 1 contains t_i, all conflict sets containing some $t_{clean,k}$ or $t_{clean,\alpha,k}$ and all conflict sets containing some $t_{q,\alpha,k,q',\alpha',D}$ such that $g(q) \neq \wedge$. The set C_2 contains all other conflict sets.

We briefly discuss the cleanup procedure. Two of the cleanup transitions are depicted in Figure 4.3. After the TM has reached the accepting state q_{accept}, we need to collect all the tokens to reach the final marking. First we collect the head token on s_k and the state token on $s_{q_{accept}}$ with the transition $t_{clean,k}$. Then we

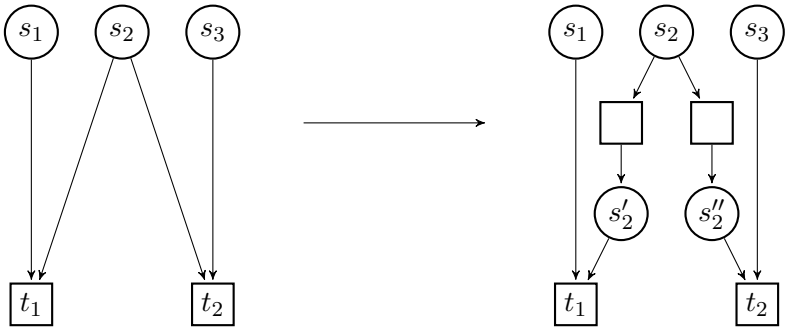


Figure 4.4: Creating free choice

collect the tokens in the cells, one by one, via the places $s_{clean,k}$ and transitions $t_{clean,\alpha,k}$. The last cell is cleared by transitions $t_{clean,\alpha,n}$ for some α, this transition puts a token on the output place o.

Observe that, in general, the workflow net $\mathcal{W}_{M,x}$ is not free choice and also unsound: if the TM does not accept, the state q_{accept} is not reachable and therefore the cleanup procedure is never executed, making the final marking unreachable.

Initially, only t_i is enabled. Thereafter, at any point in time, exactly one conflict set is enabled until either $s_{q_{accept}}$ or $s_{q_{reject}}$ is marked. If $s_{q_{accept}}$ is marked, after the cleanup the final marking is reached. To model universal and existential quantifiers, we would like to assign conflict sets to the players instead of clusters. In general however this is unpractical as there may be exponentially many conflict sets. Instead, we focus on modifying the above construction to yield a free choice arena.

We apply the following idea to transform the above construction to a free choice net: assume that t_1 and t_2 are two transitions that share an input place, but not all input places. That means t_1 and t_2 directly conflict with the free choice property. However, we can separate the two transitions by creating additional places and two "choice transitions" that choose between t_1 and t_2. This construction is depicted in Figure 4.4. While this construction generally produces unsound behavior (if the wrong choice is taken, the result may be a deadlock), we can neglect this as our construction is already unsound.

Our altered construction is detailed in the remainder of the proof.

We remove the places s_q and s_k and instead add the following places:

- $s_{q,k,head}$ and $s_{q,k,state}$ for all $q \in Q, 1 \leq k \leq n$
- $s_{\alpha,k,head}$ for all $\alpha \in \Gamma, 1 \leq k \leq n$
- $s_{q,\alpha,k,head}, s_{q,\alpha,k,state}$ and $s_{q,\alpha,k,cell}$ for all $q \in Q, \alpha \in \Gamma, 1 \leq k \leq n$

We modify the existing transitions as follows

$$
\begin{aligned}
{}^\bullet t_i &= \{i\} \\
t_i^\bullet &= \{s_{q_0,1,head}, s_{q_0,1,state}\} \cup \{s_{\alpha,k} \mid \text{ cell } k \text{ initially contains } \alpha\} \\
{}^\bullet t_{q,\alpha,k,q',\alpha',D} &= \{s_{q,\alpha,k,head}, s_{q,\alpha,k,state}, s_{q,\alpha,k,cell}\} \\
t^\bullet_{q,\alpha,k,q',\alpha',D} &= \{s_{\alpha',k}\} \cup \begin{cases} \{s_{q',k+1,head}, s_{q',k+1,state}\} & \text{if } D = R \\ \{s_{q',k-1,head}, s_{q',k-1,state}\} & \text{if } D = L \\ \{s_{q',k,head}, s_{q',k,state}\} & \text{if } D = N \end{cases}
\end{aligned}
$$

We add new transitions $t_{q,\alpha,k,cell}$ and $t_{q,\alpha,k,state}$ with the following pre- and post-sets:

$$\begin{aligned}
{}^\bullet t_{q,\alpha,k,cell} &= \{s_{q,k,head}, s_{q,k,state}\} \\
t_{q,\alpha,k,cell}^\bullet &= \{s_{\alpha,k,head}, s_{q,\alpha,k,state}\} \\
{}^\bullet t_{q,\alpha,k,state} &= \{s_{\alpha,k}, s_{\alpha,k,head}\} \\
t_{q,\alpha,k,state}^\bullet &= \{s_{q,\alpha,k,head}, s_{q,\alpha,k,cell}\}
\end{aligned}$$

This construction is shown in Figure 4.5. Intuitively, the cell content token in $s_{\alpha,k}$ waits for the head to arrive in $s_{\alpha,k,head}$. The head and the state are in $s_{q,k,head}$ and $s_{q,k,state}$. There they "guess" via a transition $t_{q,\alpha,k,cell}$ the letter α of the cell k. The state then moves to $s_{q,\alpha,k,state}$ while the head moves to $s_{\alpha,k,head}$ to collect the cell content. From there, they again "guess", this time the internal state, by choice of the transition $t_{q,\alpha,k,state}$, and move to $s_{q,\alpha,k,head}$ and $s_{q,\alpha,k,cell}$, respectively.

We also need to improve the process of cleaning up. We remove the places $s_{clean,k}$ as well as the transitions $t_{clean,k}$ and $t_{clean,\alpha,k}$ and instead add new transitions $t_{clean,k,\alpha}$ and $t_{clean,\alpha,k,\alpha'}$ for all $1 \leq k \leq n$, $\alpha, \alpha' \in \Gamma$ and set

$$\begin{aligned}
{}^\bullet t_{clean,k,\alpha} &= \{s_{q_{accept},k,state}, s_{q_{accept},k,head}\} \\
t_{clean,k,\alpha}^\bullet &= \{s_{\alpha,1,head}\} \\
{}^\bullet t_{clean,\alpha,k,\alpha'} &= \{s_{\alpha,k}, s_{\alpha,k,head}\} \\
t_{clean,\alpha,k,\alpha'}^\bullet &= \begin{cases} \{s_{\alpha',k+1,head}\} & \text{if } k < n \\ \{o\} & \text{otherwise} \end{cases}
\end{aligned}$$

Again, during the clean-up we need to guess for each of the cells the state it is in. Two transitions of the construction are shown in Figure 4.6. Notice that the cleanup transitions $t_{clean,\alpha,k,\alpha'}$ have the same pre-set as $t_{q,\alpha,k,state}$ where head and cell content guess the current state together. This will however not be a problem: Any guessing we have added in the above construction will be done by Player 1, so "guessing right" will be a part of his winning strategy. Guessing correctly is possible as Player 1 can simply keep track of the tape contents and internal state by simulating the TM.

Thus the partition of the clusters is: The set containing t_i and all conflict sets containing some $t_{q,\alpha,k,q',\alpha',D}$ such that $g(q) \neq \wedge$ belong to Player 1. Additionally, all conflict sets containing a transition $t_{q,\alpha,k,cell}$ or $t_{q,\alpha,k,state}$ belong to Player 1. Furthermore, the cleanup conflict set containing the transitions $t_{clean,k,\alpha}$ belongs to player 1. All remaining conflict sets belong to Player 2.

By this partition of the clusters, Player 1 decides all existentially quantified choices of the Turing machine while Player 2 resolves the universal quantifiers. Player 1 does all the guessing, but guessing correctly is easily possible. Therefore Player 1 has a winning strategy iff M accepts x. □

4.3 Free Choice Games

As we have seen in the previous chapters, checking for soundness of a workflow net, a PSPACE-hard problem, is achievable in polynomial time if the workflow net is free choice. Theorem 4.4 suggests that for games, there is no such tractability result. We provide a result that shows the contrary.

Well-designed workflow nets should have neither deadlocks nor livelocks and therefore should be sound. We prove that for sound free choice workflow nets, the termination and non-termination problem can be solved in polynomial time. To this end, we modify the well-known attractor computation.

Definition 4.5 (Attractor)**.** *Let $\mathcal{W}$ be a workflow arena, $\mathcal{C}_1$ and $\mathcal{C}_2$ the partition of the clusters. The attractor of the output place o is $\mathcal{A} = \bigcup_{k=0}^{\infty} \mathcal{A}_k$, where $\mathcal{A}_0 = \{[o]\}$ and*

$$\begin{aligned} \mathcal{A}_{k+1} = \mathcal{A}_k \cup & \quad \{c \in \mathcal{C}_1 \mid \exists t \in c\, \forall s \in t^\bullet :\ [s] \in \mathcal{A}_k\} \\ \cup & \quad \{c \in \mathcal{C}_2 \mid \forall t \in c\, \forall s \in t^\bullet :\ [s] \in \mathcal{A}_k\} \end{aligned}$$

The attractor position of a cluster c is the smallest k such that $c \in \mathcal{A}_k$. The attractor position of a place is that of its cluster. We say that a place is in the attractor if its cluster is in the attractor.

Theorem 4.6. *Let $\mathcal{N}$ be a sound free choice workflow arena. Player 1 has a winning strategy in the termination game iff $[i] \in \mathcal{A}$.*

Proof. ($\Leftarrow$): Assume that $[i] \in \mathcal{A}$. We fix the *attractor strategy* for Player 1. The strategy for a cluster $c \in \mathcal{C}_1 \cap \mathcal{A}$ is to choose any transition t such that all places $s \in t^\bullet$ have smaller attractor position.

Such a transition exists by construction of $\mathcal{A}$. For clusters $c \in \mathcal{C}_1 \setminus \mathcal{A}$ we choose an arbitrary transition. Notice that this strategy is not only memoryless, but also independent of the current marking.

We show that the attractor strategy is winning. By definition of the game, we have to prove that every play following the strategy ends with the final marking.

Assume the contrary: there is a play π where Player 1 plays according to the attractor strategy, which never reaches the final marking. We claim that for all markings reached along π, all marked places are in the attractor. We first observe that initially all marked places are in the attractor since $[i] \in \mathcal{A}$ by assumption. Now, assume that in some marking reached along π all marked places are in the attractor $\mathcal{A}$. Then of course all enabled clusters are in the attractor, and therefore also the clusters chosen by Scheduler. By definition of $\mathcal{A}$, after a cluster of $\mathcal{C}_2 \cap \mathcal{A}$ occurs, the tokens on $t^\bullet$ are in $\mathcal{A}$ for the transition $t \in c$ that was fired; by definition of the attractor strategy, the same holds for clusters of $\mathcal{C}_1 \cap \mathcal{A}$. This concludes the proof of the claim.

We now make use of Theorem 2.18: Since the workflow arena is sound and free choice, the extended net $\overline{\mathcal{W}}$ obtained by adding a transition which removes a token from o and adds one to i is safe and live by Theorem 2.33 and therefore covered by S-components. We choose a minimal cover and order these S-components arbitrarily. All of these components are marked initially by Corollary 2.38 and thus stay marked with exactly one token.

Since all markings M reached along π satisfy that all marked places are in $\mathcal{A}$, for each marking there is an associated attractor position vector whose components are natural numbers, the attractor positions of the marked places of each S-component. Let P_k denote the position vector of the marking reached after $k \geq 0$ steps in π. Initially only i is marked, and so $P_0 = (k_0, k_0, \ldots, k_0)$, where k_0 is the attractor position of i. We have $k_0 < |S|$ as the number of places is an upper bound for the number of clusters. Given two position vectors $P = (p_1, ..., p_m)$ and $P' = (p'_1, ..., p'_m)$, we say $P \prec P'$ if $p_i \leq p'_i$ for every $1 \leq i \leq m$, and $p_i < p'_i$ for at least one $1 \leq i \leq m$.

By definition of the attractor strategy, the sequence $P_0, P_1, ...$ of attractor positions satisfies $P_{i+1} \prec P_i$ for every i. Since $\prec$ is a well-founded order, the sequence is finite, i.e., the game terminates. By definition of the game, it terminates at a marking that does not enable any transition. Since, by assumption, the play never reaches the final marking, this marking is a deadlock, which contradicts the soundness of $\mathcal{W}$.
($\Rightarrow$): Let $\mathcal{B} = \mathcal{W} \setminus \mathcal{A}$, and assume $[i] \in \mathcal{B}$. We give a winning strategy for Player 2. The strategy for a cluster $c \in \mathcal{C}_2 \cap \mathcal{B}$ is to choose any transition t such that at least one place $s \in t^\bullet$ is not in $\mathcal{A}$. Such a transition exists because, since $c \in \mathcal{B}$, we have $c \notin \mathcal{A}_k$ for every k, and so by definition and monotonicity of $\mathcal{A}_k$ there exists some $t \in c$ such that $[s] \notin \mathcal{A}_k$ for some $s \in t^\bullet$ and every k. For atoms in $\mathcal{C}_2 \setminus \mathcal{B}$ we chose an arbitrary transition.

We show that this strategy is winning for Player 2. Once again, because the workflow net is sound, no play played according to this strategy ends in a deadlock. So we have to prove that every game played following the strategy never ends. Since $[o] \in \mathcal{A}$, it suffices to prove that the play never reaches a marking at which every marked place is in the attractor.

Initially all tokens are on i, and $[i] \in \mathcal{B}$. Now, assume that at some marking M reached along π there is a marked place s that satisfies $[s] \in \mathcal{B}$. We prove that the same holds for the marking M' reached after one step of the play. If $[s]$ is not enabled at M, then we have $M'(s) = M(s)$ and we are done. The same holds if $[s]$ is enabled at M, but is not selected by Scheduler. If $[s]$ is enabled and selected by Scheduler, there are two possible cases. If $[s] \in \mathcal{C}_1$ then by the definition of $\mathcal{A}$ for every transition $t \in [s]$ there is a place $s' \in t^\bullet$ such that $[s'] \in \mathcal{B}$. If $[s] \in \mathcal{C}_2$, then by definition the strategy chooses a transition t for which there is a place $s' \in t^\bullet$ such that $[s'] \in \mathcal{B}$. □

The above proof not only yields a strategy for Player 1 if $[i] \in \mathcal{A}$, but also gives us a strategy for Player 2 if $[i]$ is not in the attractor. Thus the choices made by Scheduler cannot influence the result of the game in the sound free choice case.

Corollary 4.7. *For the termination game over sound free choice workflow arenas, the following holds:*

(a) The game collapses to a two-player game.

(b) Memoryless strategies suffice for both players.

(c) The winner and the winning strategy can be computed in $O(|T| \cdot |S|)$ time.

Proof. (a) and (b): The attractor computation and the strategies used in the proof of Theorem 4.6 are independent of the choices of Scheduler; the strategies are memoryless.
(c): We describe an algorithm that computes the set $\mathcal{A}$ and the winning strategy for Player 1, listed as Algorithm 4 on page 107.

Algorithm 4 Attractor computation

```
1: array[] count
2: array[] transitionCount
3: array[] strategy
4: set border
5: set 𝒜
6: for every cluster c do
7:     transitionCount[c] ← |c ∩ T|
8:     for every transition t ∈ c do
9:         count[c][t] ← |t•|
10:    end for
11: end for
12: 𝒜 ← {[o]}
13: border ← {[o]}
14: transitionCount[[o]] ← 0
15: while border ≠ ∅ do
16:    choose and remove a cluster c from border
17:    for every transition t' ∈ T where t'• ∩ c ≠ ∅ do
18:        count[t'] ← count[t'] − |t'• ∩ c|
19:        c' ← [t']
20:        if count[t'] = 0 and c' ∈ 𝒞1 then
21:            transitionCount[c'] ← 0
22:            strategy[c'] ← t
23:            border ← border ∪ {c'}
24:        end if
25:        if count[t'] = 0 and c' ∈ 𝒞2 then
26:            transitionCount[c'] ← transitionCount[c'] − 1
27:            if transitionCount[c'] = 0 then
28:                border ← border ∪ {c'}
29:            end if
30:        end if
31:    end for
32: end while
33: return 𝒜
```

Intuitively, $count[t]$ counts the output places of t that do not reach the attractor, $transitionCount[c]$ counts the transitions of c for which $count[t] \neq 0$ or is zero if c is in the attractor.

This algorithm terminates and computes $\mathcal{A}$, the check $[i] \in \mathcal{A}$ is done by checking whether $transitionCount[[i]]$ is zero. Furthermore, the strategy for Player 1 is computed in $strategy[]$. Every transition is inspected at most once per cluster, thus the running time is at most $O(|T| \cdot |S|)$. This leads to a total running time of at most $O(|T| \cdot |S| + |S|) \in O(|T| \cdot |S|)$. Computing the strategy for Player 2 can be done in a straightforward loop over all transitions choosing any transition for which $count[t] \neq 0$ in running time $O(|T|)$. □

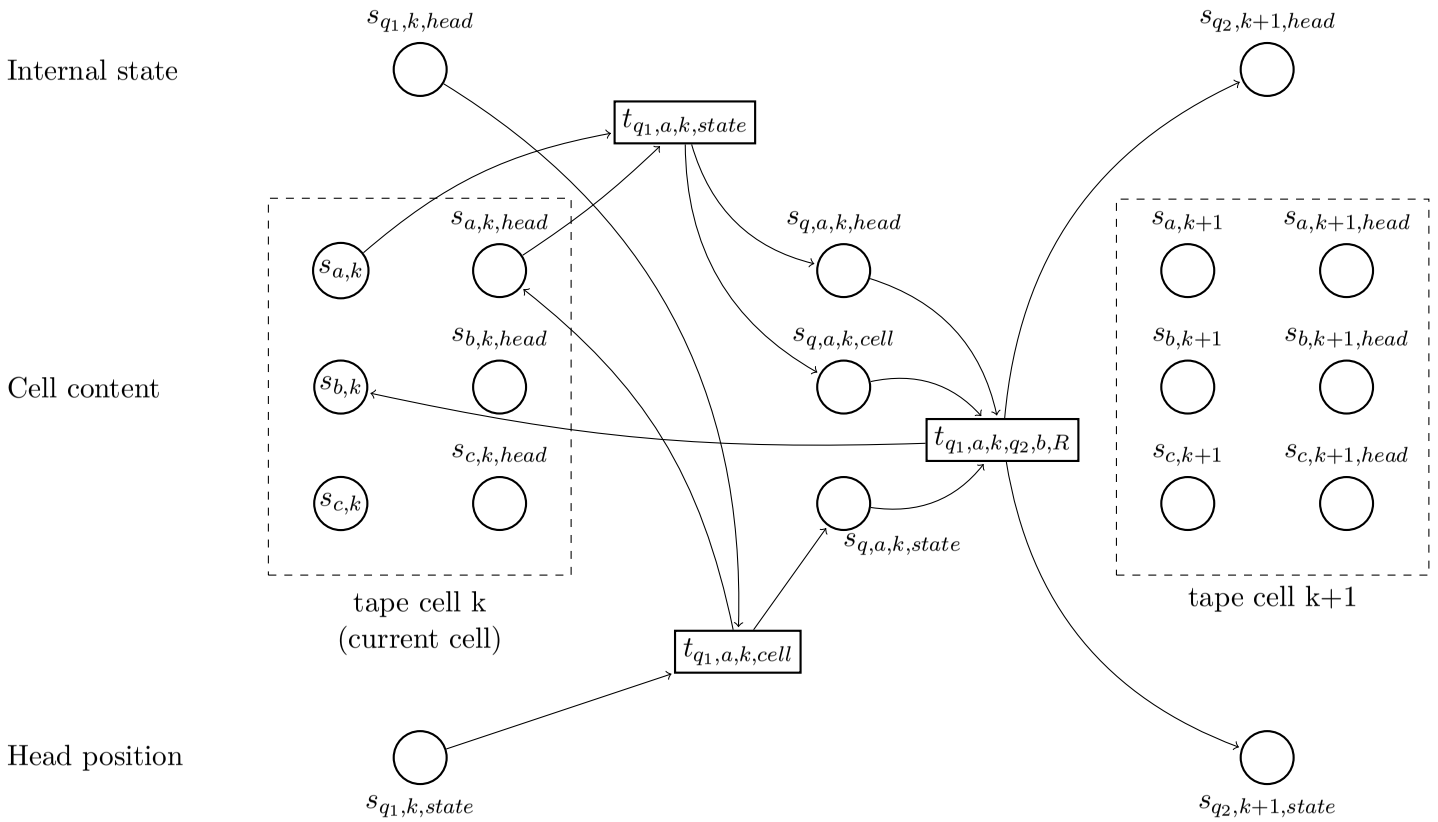


Figure 4.5: The free choice construction

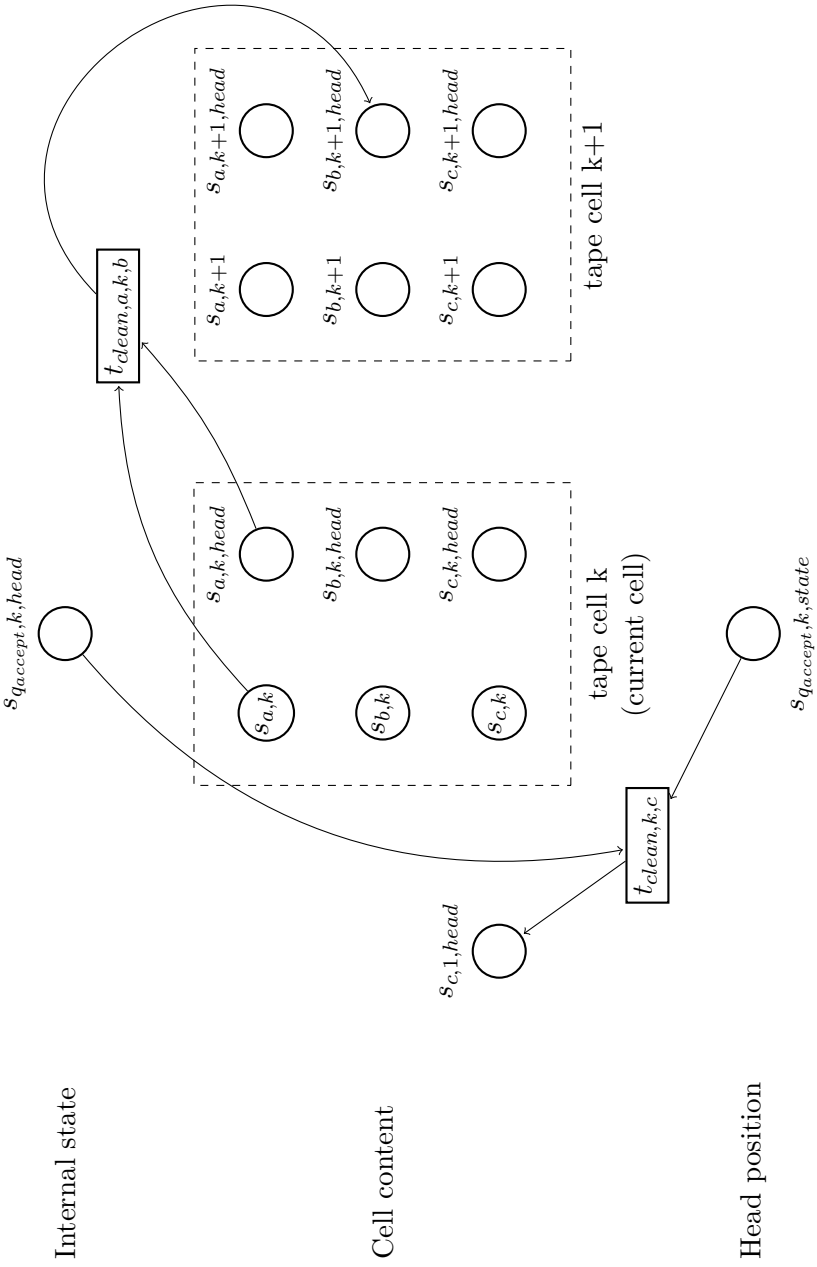


Figure 4.6: Part of the cleanup procedure in the free choice construction

Chapter 5

Conclusion and Future Work

Contents

5.1 Conclusion

In this thesis, we have introduced the reader to Petri nets and workflow nets. We have described properties such as liveness, boundedness and free choice of Petri nets as well as soundness of workflow nets and described the connection between them. We have also introduced extensions of colored workflow nets and defined the summary of a workflow net. In another extension we have defined probabilistic workflow nets and the expected reward.

Using the reduction procedure for finite automata as inspiration, we have defined reduction rules for workflow nets that are correct for workflow nets and summarize all sound free choice workflow nets, and only the sound nets, to a trivial net. With slight modifications to the rules it is also possible to compute the summary of a CWN or the expected value of a PWN. Furthermore, the algorithm needs at most a polynomial number of rule applications.

In the last chapter, we have introduced games on workflow nets. An initial study on the complexity of computing a winner and winning strategy yielded disappointing complexity bounds for the general case and even for the free choice case. However, a restriction to sound free choice workflow nets enabled us to give a polynomial algorithm and we were able to show that the game collapses to a two-player game with memoryless strategies.

5.2 Future Work

Both extensions of workflow nets presented in this thesis, colored workflow nets and probabilistic workflow nets, can be interpreted as workflow nets where the transitions are labeled with an element of a given semiring. The summary/expected value can then be defined by also associating a semiring element with each *Mazurkiewicz trace* via the multiplication of the semiring, then summing up over all traces to obtain the summary/expected value.

This more general setting would also allow us to show that our rules are correct for workflow nets with timed transitions where each transition fires according to an exponential distribution and the task is to compute the expected time until the final marking is reached. Thus we would like to prove that the rules are also correct in this setting.

For games, an interesting extension might be to also consider probability, for example as an additional player that controls some of the clusters. The task is then to compute the probabilities with which the players can achieve their goals.

Bibliography

[1] W. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[2] W. van der Aalst. Formalization and Verification of Event-Driven Process Chains. *Information and Software Technology*, 41(10):639–650, 1999.

[3] W. van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri-Net-based Techniques. In *Business Process Management*, pages 161–183. Springer, 2000.

[4] W. van der Aalst, J. Desel, and A. Oberweis. *Business Process Management: Models, Techniques, and Empirical Studies*. Springer, 2003.

[5] W. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, 2004.

[6] W. van der Aalst, K. van Hee, A. ter Hofstede, N. Sidorova, H. Verbeek, M. Voorhoeve, and M. T. Wynn. Soundness of Workflow Nets: Classification, Decidability, and Analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011.

[7] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *J. ACM*, 49(5):672–713, 2002.

[8] T. Brázdil, P. Jancar, and A. Kucera. Reachability Games on Extended Vector Addition Systems with States. In *ICALP (2)*, volume 6199 of *LNCS*, pages 478–489. Springer, 2010.

[9] A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1): 114–133, 1981.

[10] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40. Cambridge university press, 2005.

[11] J. Desel, J. Esparza, and P. Hoffmann. Negotiation as concurrency primitive. *Computing Research Repository*, abs/1612.07912, 2016.

[12] B. van Dongen, M. H. Jansen-Vullers, H. Verbeek, and W. van der Aalst. Verification of the SAP Reference Models using EPC Reduction, State-Space Analysis, and Invariants. *Computers in Industry*, 58(6):578–601, 2007.

[13] M. Dumas and A. ter Hofstede. UML Activity Diagrams as a Workflow Specification Language. In *International Conference on the Unified Modeling Language*, pages 76–90. Springer, 2001.

[14] M. Dumas, W. van der Aalst, and A. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software through Process Technology*. John Wiley & Sons, 2005.

[15] J. Esparza. Decidability and Complexity of Petri Net Problems - An Introduction. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the Volumes Are Based on the Advanced Course on Petri Nets*, pages 374–428. Springer-Verlag, 1998.

[16] J. Esparza and J. Desel. On Negotiation as Concurrency Primitive. In *CONCUR*, volume 8052 of *LNCS*, pages 440–454. Springer, 2013.

[17] J. Esparza and J. Desel. On negotiation as concurrency primitive II: Deterministic cyclic negotiations. In A. Muscholl, editor, *FoSSaCS*, volume 8412 of *LNCS*, pages 258–273. Springer, 2014.

[18] J. Esparza and P. Hoffmann. Reduction Rules for Colored Workflow Nets. In *FASE 2016*, volume 9633 of *LNCS*, pages 342–358. Springer, 2016.

[19] J. Esparza, P. Hoffmann, and R. Saha. Polynomial Analysis Algorithms for Free Choice Probabilistic Workflow Nets. In *QEST 2016*, volume 9826 of *LNCS*, pages 89–104. Springer, 2016.

[20] D. Fahland, C. Favre, B. Jobstmann, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf. Instantaneous Soundness Checking of Industrial Business Process Models. In *Business Process Management*, volume 5701 of *LNCS*, pages 278–293. Springer, 2009.

[21] C. Favre, D. Fahland, and H. Völzer. The Relationship between Workflow Graphs and Free-Choice Workflow Nets. *Information Systems*, 47:197–219, 2015.

[22] R. A. Gordon. *The Integrals of Lebesgue, Denjoy, Perron, and Henstock*. Number 4. American Mathematical Soc., 1994.

[23] K. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In *ICATPN*, volume 2679, pages 337–356. Springer, 2003.

[24] K. van Hee, N. Sidorova, and M. Voorhoeve. Generalised Soundness of Workflow Nets is Decidable. In *International Conference on Application and Theory of Petri Nets*, pages 197–215. Springer, 2004.

[25] P. Hoffmann. Negotiation Games. In *GandALF 2015*, volume 193 of *EPTCS*, pages 31–42, 2015.

[26] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[27] K. Jensen and L. M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer Science & Business Media, 2009.

[28] J. G. Kemeny, J. L. Snell, and A. W. Knapp. *Denumerable Markov Chains: with a chapter of Markov Random Fields by David Griffeath*, volume 40. Springer Science & Business Media, 2012.

[29] A. Kucera. Playing Games with Counter Automata. In *Reachability Problems*, volume 7550 of *LNCS*, pages 29–41. Springer, 2012.

[30] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *CAV 2011*, LNCS, vol. 6806, pages 585–591, 2011.

[31] G. Liu, J. Sun, Y. Liu, and J. Dong. Complexity of the Soundness Problem of Workflow Nets. *Fundamenta Informaticae*, 131(1):81–101, 2014.

[32] A. Martens. Analyzing Web Service Based Business Processes. In *International Conference on Fundamental Approaches to Software Engineering*, pages 19–33. Springer, 2005.

[33] A. Mazurkiewicz. Trace Theory. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, pages 278–324. Springer, 1986.

[34] S. Mohalik and I. Walukiewicz. Distributed Games. In *FSTTCS*, volume 2914 of *LNCS*, pages 338–351. Springer, 2003.

[35] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[36] A. Oberweis. *Modellierung und Ausführung von Workflows mit Petri-Netzen*. Springer-Verlag, 2013.

[37] J. L. Peterson. Petri Nets. *ACM Computing Surveys (CSUR)*, 9(3):223–252, 1977.

[38] W. Reisig. *Petri Nets: An Introduction*, volume 4. Springer Science & Business Media, 2012.

[39] W. Thomas, T. Wilke, et al. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500. Springer Science & Business Media, 2002.

[40] A. Valmari. The State Explosion Problem. In *Lectures on Petri Nets I: Basic Models*, pages 429–528. Springer, 1998.

[41] J. Vanhatalo, H. Völzer, and F. Leymann. Faster and more focused control-flow analysis for business process models through SESE decomposition. In *Service-Oriented Computing - ICSOC 2007*, volume 4749 of *LNCS*, pages 43–55. Springer, 2007.

[42] D. Varacca and M. Nielsen. Probabilistic Petri Nets and Mazurkiewicz Equivalence. 2003. Unpublished Manuscript. Available online at `http://www.lacl.fr/~dvaracca/works.html`. Last retrieved on May 27, 2016.

[43] H. M. Verbeek, T. Basten, and W. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.

[44] I. Walukiewicz. Pushdown Processes: Games and Model-Checking. *Inf. Comput.*, 164(2):234–263, 2001.

[45] S. A. White. Introduction to BPMN. *IBM Cooperation*, 2(0):0, 2004.

[46] Wikipedia. Markov Chains — Wikipedia, the Free Encyclopedia, 2016. Available online at `https://en.wikipedia.org/wiki/Markov_chain`. Last retrieved on Dec 01, 2016.

[47] K. Wolf. Generating Petri Net State Spaces. In *International Conference on Application and Theory of Petri Nets*, pages 29–42. Springer, 2007.