# Computing Cut Sets for Petri Nets

Clara Scherbaum

August 12, 2016

This report introduces several algorithms to compute cut sets of Petri nets. A cut set is a set of places computed with respect to a certain Petri net and a *bad place* of that Petri net. The characteristic of a cut set is that if we never have a token on the places of the cut set then we never have a token on the bad place. A motivation to compute cut sets is to propose a therapy for Petri nets corresponding to biological networks where we would like to forbid certain behaviour. We present in total three methods. Two of those methods are complete and correct methods for computing cut sets and one is an under-approximation. Whereas the first method is based on the marking graph of the Petri net, the other methods are unfolding-based. The two exact methods result in a second step in an Answer Set Program. A comparison of the two methods shows off that they are suited to different properties of a Petri net. Furthermore, we introduce a notion of cut sets that also allows to preserve certain good behaviour and propose a method how to compute these sets.

## 1 Introduction

Petri nets are a widely used model. Among other contexts Petri nets can be used to represent the dynamics of biological systems [CHJ+14, SBW06, CP16]. In biological networks there might be undesired behaviour. To propose therapies we need to find other parts of the network that we can block and whose blocking results in an avoidance of the undesired behaviour. This gives rise to a formal definition of set of states/places of a network whose regulation forbids the undesired behaviour [Pau16, SvKK10, HK11, CP16].

We define *cut sets* of a Petri net with respect to a certain place of the net, the *bad place*. A cut set is a set of places such that if we never have a token on each of these places there will never be a token on the bad place. We will concentrate on the computation of *minimal* cut sets, cut sets minimal with respect to set inclusion. We aim to develop algorithms that find all minimal cut sets of a Petri net and a bad place. In a next step we extend the notions of cut sets to the one of control sets. The main difference is that

the aim is to find sets that do not only prohibit bad behaviour but also allow for good behaviour.

The computation of cut sets also with regard to the preservation of good behaviour in boolean networks was investigated in [SvKK10]. That algorithm is based on enumeration and aims to work on boolean networks. Furthermore, there exist an under-approximation for cut sets in automata networks [PAK13]. This approach is based on obtaining a structure representing causal dependencies in the automata network. Using this as a starting point several cut sets are computed. None of these algorithms is based on the semantics of the Petri net. The execution semantics and concurrency semantics of Petri nets have different advantages and disadvantages that suggest that for different Petri nets certain semantics provide a better starting point for algorithms. Given the size of the biological networks, runtime of algorithms is of importance, and therefore the need to investigate the difference the used semantics can make. Furthermore, we show several ways to generate an answer set program to obtain all cut sets. As ASP solvers have been intensively studied over the past years solvers have implemented good heuristics to approach the problem [GKK$^+$11].

This report presents in total three algorithms. The first two are complete and correct algorithms based on the different semantics of Petri nets and using answer set programming. The third one is an under-approximation. Our first two methods aim for suitable algorithms with regard to the Petri nets and instead of enumerating with own heuristics using heuristics of efficient and carefully-developed solvers. The first algorithm is based on the marking graph of the Petri net, the second one on the unfolding and one of its prefixes respectively. This gives also rise to a comparison of the two methods. The third method is an under-approximation. In contrast to the under-approximation developped in [PAK13] it is based on the Esparza-Römer-Vogler prefix [ERV96] of the Petri net. This gives rise to new observations of characteristics of this prefix.

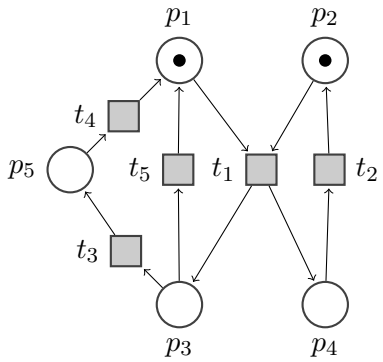
# 2 Petri Nets, their Unfoldings and Prefixes

In this section we introduce Petri nets, its execution semantics and its concurrency semantics. By execution semantics we mean the marking graph and by concurrency semantics the unfolding and underlying concepts as branching processes and occurrence nets. An general overview on Petri nets can be found in [Rei13].

## 2.1 Introduction to Petri Nets and their Execution Semantics

In this subsection we introduce the definitions of a Petri net and other basic concepts such as pre- and postsets and the marking graph.

A Petri net is a model for representing concurrent systems. Formally, it is a directed, bipartite graph where the nodes can be places or transitions. A place can hold an




Example 1: Petri net $N_1$

arbitrary number of tokens. A marking specifies the number of tokens on each place. A transition is enabled by a marking iff there is a token on each places that have an edge to the transition. If a transition is enabled by a marking, it can fire. Firing a transition removes one token from each place having an edge to the transition and adding one token to each place that the transition has an edge to. Hence, a Petri net consists of places, transitions, a flow relation which describes the edges between places and transitions and an initial marking. The initial marking specifies how many tokens are on each place before any transition has been fired.

**Definition 1** (Petri net)**.** A *Petri net* is a quadruple $N = (P, T, F, M_0)$ where

- $P$ is a set of places
- $T$ is a set of transitions
- $F \subseteq \{(p,t)|p \in P, t \in T\} \cup \{(t,p)|t \in T, p \in P\}$ the flow relation
- $M_0 \subseteq P$ is the initial marking referring to all places that initially hold a token.

To get a deeper insight into Petri nets and related concepts such as its marking graph and its unfolding we look at an example.

The Petri net of Example 1 is formalized by $N = (P, T, F, M_0)$ with $M_0 = \{p_1, p_2\}$. To be able to easily refer to the places relevant for infering a transition, we define the preset and the postset of a node in the Petri net. A preset of a node contains all nodes that have an edge to the node and in the postset are all nodes that can be reached by using one edge.

**Definition 2** (Pre- and postset)**.** The preset of any node $n \in P \cup T$ of the Petri net, ${}^\bullet n$, is defined as

$${}^\bullet n = \{n' | (n', n) \in F\}.$$

The postset is analogously defined as

$$n^\bullet = \{n' | (n, n') \in F\}.$$

Above, we explained when a transition is enabled by a marking and what marking is reached if it fires. We can now properly define this by referring to the pre- and postset of the transition. A transition is enabled by a marking iff there is a token on each place of its preset. When it fires we remove one token from each place in the preset and add one to each place in its postset and obtain this way a new marking.

**Definition 3** (Enabling and firing of a transition)**.** A transition $t \in T$ is *enabled* by a marking $M \subseteq P$ iff ${}^\bullet t \subseteq M$.

When firing $t \in T$, enabled by $M$, this leads to a new marking $M'$ with $M' = (M \backslash {}^\bullet t) \cup t^\bullet$. We also write for this $M \xrightarrow{t} M'$.

By being able to fire several transitions in a row we obtain a *transition sequence*.

**Definition 4** (Transition sequence)**.** $\sigma = t_1 \dots t_n$ with $\{t_1, \dots, t_n\} \subseteq T$ is a *transition sequence* iff there exist $M_1, \dots, M_n$ such that

$$M_0 \xrightarrow{t_1} M_1 \dots \xrightarrow{t_n} M_n$$

In the context of Example 1 we can obtain among others the following transition sequences:

$$\{p_1, p_2\} \xrightarrow{t_1} \{p_3, p_4\} \xrightarrow{t_2} \{p_2, p_3\} \xrightarrow{t_3} \{p_2, p_5\} \xrightarrow{t_4} \{p_1, p_2\}$$
$$\{p_1, p_2\} \xrightarrow{t_1} \{p_3, p_4\} \xrightarrow{t_3} \{p_4, p_5\} \xrightarrow{t_2} \{p_2, p_5\} \xrightarrow{t_4} \{p_1, p_2\}$$

We see that we can reach different markings by firing transition sequences such as the ones above. This leads to the definition of *reachable markings*.

**Definition 5** (Reachable marking)**.** Given a Petri net $N = (P, T, F, M_0)$ a marking $M$ is *reachable* iff there exists a transition sequence $\sigma = t_1 \dots t_n$ such that

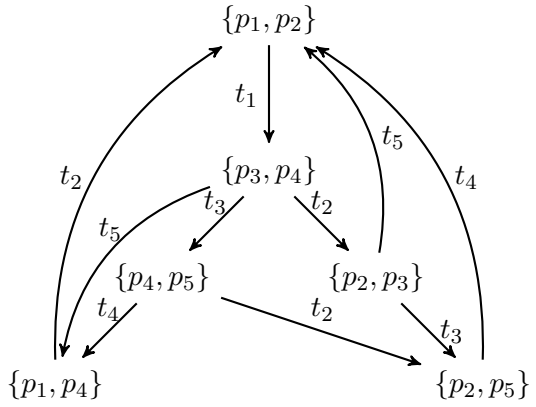
$$M_0 \xrightarrow{\sigma}{}^* M$$

In the following, we only work with Petri nets that cannot reach a marking in which we have more than one token in a place. We call these Petri nets *safe* Petri nets.

**Definition 6** (Safe Petri net)**.** A Petri net is *safe* iff there is no marking reachable where we have more than one token at one place.

From now on, when we use the term *Petri net* we mean *safe Petri net*.

To represent all reachable markings of a net we can construct the *marking graph*. The marking graph has as nodes reachable markings and two markings $M_1, M_2$ are connected




Example 2: Marking graph of Petri net $N_1$ from Example 1

by an edge iff there exists $t \in T$ such that $M_1 \xrightarrow{t} M_2$. We label the edge from $M_1$ to $M_2$ by all transitions that are enabled by $M_1$ and reach by firing $M_2$.

**Definition 7** (Marking graph)**.** A *marking graph* is a directed graph $(V, E)$ where each $V$ is the set of all reachable markings and for two reachable markings $M_1, M_2 \in V$ there is $(M_1, M_2) \in E \subseteq V \times V$ iff there exists a transition $t \in T$ such that $M_1 \xrightarrow{t} M_2$.

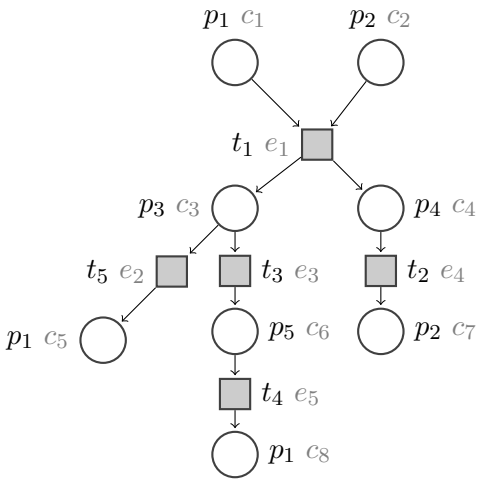
The marking graph of the Petri net of Example 1 can be seen in Example 2.

As we obtain the marking graph by executing enabled transitions one-by-one we also refer to it as the *execution semantics* of a Petri net.

## 2.2 Concurrency Semantics of Petri Nets

In the last subsection we introduced Petri nets in general and their execution semantics, the marking graph. In this subsection we show some of the weaknesses of this semantics and introduce another semantics, the concurrency semantics of Petri nets. As in the previous section, we use the term "Petri net" in the sense of "safe Petri net" (Definition 6). A formal introduction to unfoldings can be found in [EH08].

When we only want to know whether a certain marking is reachable by performing a marking sequence from the initial marking we can check whether there exists a path in the marking graph from the initial marking to the sought-after marking. Regarding Example 1 when we want to check whether there is a non-empty transition sequence from the initial marking that leads to the initial marking again we find three different




Example 3: A branching process of the Petri net $N_1$ from Example 1, $B_0$

acyclic paths.

$$\{p_1, p_2\} \xrightarrow{t_1} \{p_3, p_4\} \xrightarrow{t_2} \{p_2, p_3\} \xrightarrow{t_3} \{p_2, p_5\} \xrightarrow{t_4} \{p_1, p_2\}$$
$$\{p_1, p_2\} \xrightarrow{t_1} \{p_3, p_4\} \xrightarrow{t_3} \{p_4, p_5\} \xrightarrow{t_2} \{p_2, p_5\} \xrightarrow{t_4} \{p_1, p_2\}$$
$$\{p_1, p_2\} \xrightarrow{t_1} \{p_3, p_4\} \xrightarrow{t_3} \{p_4, p_5\} \xrightarrow{t_4} \{p_1, p_4\} \xrightarrow{t_2} \{p_1, p_2\}$$

Comparing these marking sequences we observe that the transitions that are used for the three transitions sequences are the same, only their schedules varies. This may indicate that the marking graph is not the most compact model. There are even more schedules if we have more transitions that can fire at the same time. If we have $n$ transitions that can fire at the same time, we can reach the marking that holds after firing all of these transitions by $n!$ different interleavings. This leads to a combinatorial explosion in the marking graph. It would be useful to have a representation that takes not only into account that a transition is enabled by a certain marking but also that certain transitions are concurrently enabled by a certain marking. Therefore, we will look at the concept of branching processes and unfoldings.

In Example 3 a branching process of the Petri net in Example 1 is shown.

A branching process is an occurrence net related by a homomorphism to another Petri net. We see that we have attached to each place and transition two designations. The one in gray is the name of the place/transition and the one in black is its label. We can have several places/transitions with the same label. So there are for example two places with the label $p_1$. We see that by performing marking sequences in this net we obtain marking sequences of the original net when looking at the labels of the places and transitions, and vice versa. We illustrate this with an example.

$$\{c_1, c_2\} \xrightarrow{e_1} \{c_3, c_4\} \xrightarrow{e_4} \{c_7, c_3\} \xrightarrow{e_3} \{c_7, c_6\} \xrightarrow{e_5} \{c_7, c_8\}$$

When looking at the labels of the places and transitions in the marking sequence above we obtain the following sequence:

$$\{p_1, p_2\} \xrightarrow{t_1} \{p_3, p_4\} \xrightarrow{t_2} \{p_2, p_3\} \xrightarrow{t_3} \{p_2, p_5\} \xrightarrow{t_4} \{p_1, p_2\}$$

This is also a sequence in the Petri net of Example 1.

To distinguish the places of this net from the places of the original net and the transitions respectively we denote the places and transitions of this net *conditions* and *events*, respectively. We see that the labels of the conditions and events correspond to the context these places and transitions of the original net had, ie the labels of the pre- and postset correspond to the original pre- and postsets, too.

We observe that for firing $e_2$ we need to fire $e_1$ (analogously for $e_3$ and $e_4$). Therefore, we say $e_1$ is in *causal relation* with $e_2$. Formally, we define the causal relation by saying that two nodes (so events or conditions) are in causal relation if there is a directed path from one to the other in the Petri net.

We see furthermore that if we fire $e_2$ we can no longer fire $e_3$. We denote that relation by saying $e_2$ and $e_3$ are *in conflict*. Formally two nodes are in conflict if we have two paths in the net from a condition and each path comes across one of these events and the paths directly diverge.

We also see that $e_2$ and $e_4$ are neither in conflict nor does one causally precede the other. We can deduce that firing $e_2$ has no effect on firing $e_4$ and the other way around. We therefore say, $e_2$ and $e_4$ are *concurrent*.

Of course, not only transitions but also places can be in such relations.

**Definition 8** (Causal, conflict and concurrency relation)**.** We have a Petri net $N = (P, T, F, M_0)$.

A node $n_1 \in T \cup P$ is *causally preceding* another node $n_2 \in T \cup P$ if there exists a path from $n_1$ to $n_2$. We denote that relation by writing $n_1 < n_2$.

A node $n_1 \in T \cup P$ is *in conflict* with another node $n_2 \in T \cup P$ if there exist $t_1, t_2$ with $t_1 < n_1$ and $t_2 < n_2$ and ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$. We denote this relation by writing $n_1 \# n_2$.

If two nodes $n_1, n_2 \in T \cup P$ are neither in conflict nor causally related they are *concurrent*. We denote this by saying $n_1$ co $n_2$.

Formally, a branching process is an *occurrence net* with a specific labelling function. A Petri net is an occurrence net if no place can hold more than one token, it is acyclic, it has initial conditions whose preset is empty and no node is in conflict with itself.

**Definition 9** (Occurrence net)**.** A Petri net $O = (C, E, F, M_0)$ is an *occurrence net* iff

- There are no two nodes $n_1, n_2 \in C \cup E$ such that $n_1 < n_2$ and $n_2 < n_1$ (The causality relation is acyclic).

- A condition holds at most one token and $M(c) = 1$ iff $|{}^\bullet c| = 0$.
- For every event $e \in E$, $e \# e$ does not hold and $\{n | n \leq e\}$ is finite.

As we said before, we use occurrence nets to illustrate reachable markings and transition sequences of another Petri net. Therefore, we have a labelling function establishing a connection between conditions and events of the occurrence net and places and transitions of the original Petri net. This labelling function from the conditions and events of the occurrence net to the places and transitions of the original Petri net should fulfil the following conditions:

- It should map conditions only to places and events only to transitions.
- It should preserve pre- and postsets of places and transitions.
- The initial marking of the occurrence net should be mapped to the initial marking of the original Petri net.

**Definition 10** (Net homomorphism)**.** Given an occurrence net $O = (C, E, F_O, M_{0_O})$ and $N = (P, T, F_N, M_{0_N})$ a function $\pi \colon C \cup E \mapsto P \cup T$ is a *net homomorphism* iff the following conditions hold

- $\pi(C) \subseteq P$ and $\pi(E) \subseteq T$
- For all $e \in E$ the restriction of $\pi$ to ${}^\bullet e$ is a bijection between ${}^\bullet e$ and ${}^\bullet \pi(e)$ and the restriction of $\pi$ to $e^\bullet$ is a bijection between $e^\bullet$ and $\pi(e)^\bullet$
- The restriction of $\pi$ to $M_{0_O}$ is a bijection between $M_{0_O}$ and $M_{0_N}$.

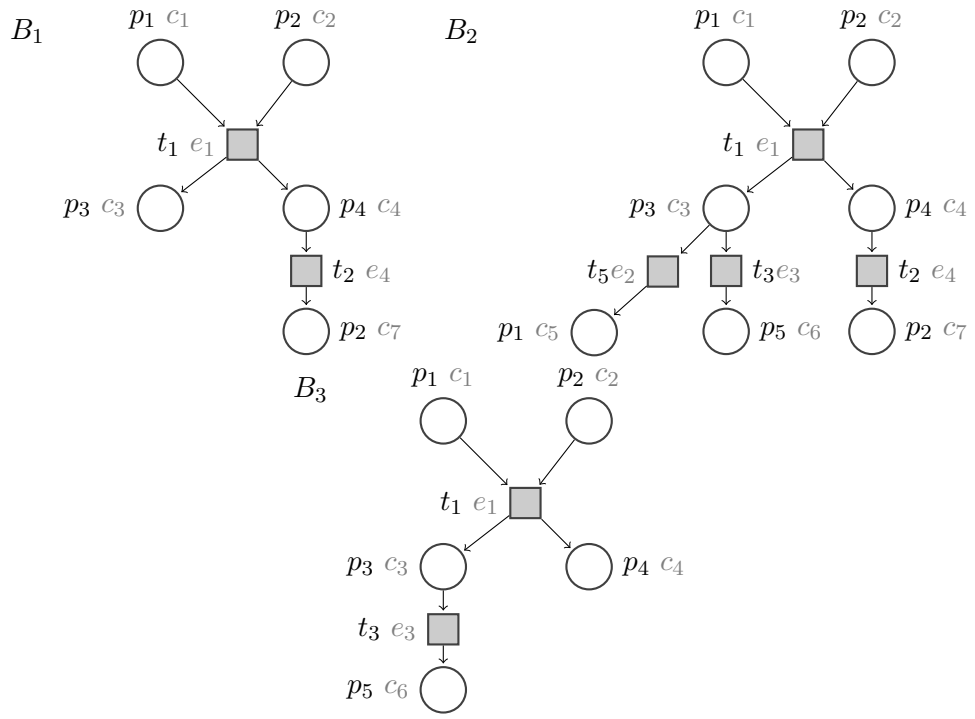
Therefore, a branching process is an occurrence net with a net homomorphism mapping its conditions and events to places and transitions of the original Petri net. Furthermore, we enforce that there are no duplicate events meaning two events with the same preset and the same label.

**Definition 11** (Branching process)**.** A *branching process* $B = (O, \pi)$ of a Petri net $N = (P, T, F, M_0)$ consists of an occurrence net $O = (C, E, F_O, M_{0_O})$ and a net homomorphism $\pi \colon C \cup E \mapsto P \cup T$ such that for all $e_1, e_2 \in E$ with ${}^\bullet e_1 = {}^\bullet e_2$ there is $\pi(e_1) \neq \pi(e_2)$.

We can compare branching processes with regard to their sizes and structures. We illustrate this with Example 2.2.

Intuitively, we could say that $B_1$ is "contained" in the branching process $B_2$. We observe the same for the branching process $B_3$ and $B_2$. However, $B_1$ and $B_3$ do not seem related in that way. This intuition leads us to the relation "$\sqsubseteq$". Two branching processes are in that relation iff there exists an injective homomorphism from the first one to the second one.

**Definition 12** (Partial order on branching processes)**.** Given two branching processes




Example 4: Some branching processes of the Petri net Example 1

$B_1, B_2$ of the Petri net $N = (P, T, F, M_0)$ there is $B_1 \sqsubseteq B_2$ iff there exists an injective net homomorphism from $B_1$ to $B_2$.

Regarding Example 2.2 we have $B_1 \sqsubseteq B_2$ and $B_3 \sqsubseteq B_2$.

When we look at Example 1 and the first branching process we looked at in Example 3 we see that we can extend this branching process such as done in Example 5.

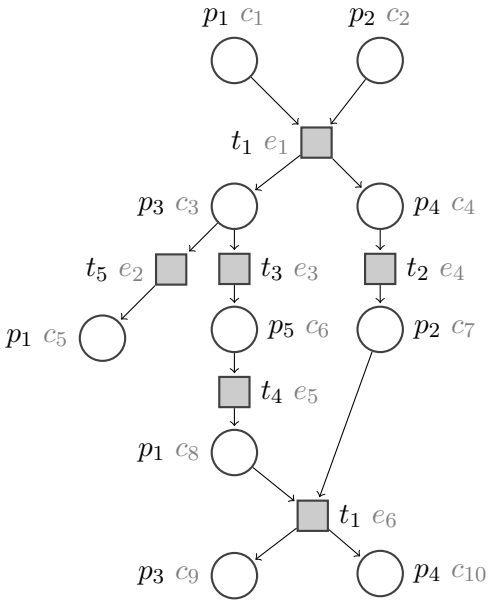
So we have $B_0 \sqsubseteq B_4$. We can extend the branching process from Example 5 again.

When we speak about the unfolding of the Petri net we mean the branching process that is maximal up to isomorphism with regard to $\sqsubseteq$. The unfolding may be infinite in general.

**Definition 13.** Given a Petri net $N = (P, T, F, M_0)$ its *unfolding* is its maximal branching process up to isomorphism with regard to $\sqsubseteq$.

Additionally, the following theorem states that the unfolding of a Petri net always exists and is unique.

**Theorem 1.** *Each Petri net $N = (P, T, F, M_0)$ has a unique unfolding up to isomorphism.*




Example 5: A branching process of the Petri net Example 1, $B_4$

The proof is out of scope of this report and can be found in [Eng91]. □

Although the unfolding contains useful information about the Petri net, its use is very limited due to the fact that it is infinite in general. Therefore, it would be helpful if we could obtain information relevant for us, only by looking at a part of the unfolding that preserves this information. This leads us to the idea of a finite prefix of the unfolding. To introduce that concept we first introduce the concept of configurations and cuts.

A *configuration* is a set of events that are causally closed and conflict free.

**Definition 14** (Configuration)**.** Given an occurrence net $O = (C, E, F, M_0)$ a *configuration* $Conf$ is a set of events, $Conf \subseteq E$, satisfying the following conditions:

- If $e \in Conf$ then for all $e' \leq e$ there is $e' \in Conf$, too.
- For no two events $e_1, e_2 \in Conf$ there is $e_1 \# e_2$.

Regarding Example 5, $Conf_{Ex} = \{e_1, e_2, e_4\}$ is a configuration among others.

A *cut* of a configuration is the marking we obtain after firing all events in the configuration.

**Definition 15** (Cut)**.** Given an occurrence net $O = (C, E, F, M_0)$ and a configuration $Conf \subseteq E$ the *cut* of $Conf$, $Cut(Conf)$ is defined as follows:

$$Cut(Conf) = (M_0 \cup C^\bullet) \backslash {}^\bullet C$$

The cut of the example from above is

$$Cut(Conf_{Ex}) = Cut(\{e_1, e_2, e_4\}) = (\{c_1, c_2\} \cup \{c_3, c_4, c_5, c_7\}) \backslash \{c_1, c_2, c_3, c_4\} = \{c_5, c_7\}.$$

Furthermore, the *marking* of a configuration is defined as the set of places corresponding to the conditions in the cut of the configuration.

**Definition 16** (Marking)**.** Given a Petri net $N = (P, T, F_N, M_{0_N})$ and one of its branching process $B = (O, \pi)$ with $O = (C, E, F_O, M_{0_O})$, and $\pi$ is a net homomorphism from $N$ to $O$. Then, the *marking* of a configuration $Conf \subseteq E$, $Mark(Conf)$ is defined as follows:

$$Mark(Conf) = \pi(Cut(Conf))$$

The marking of the configuration $Conf_{Ex}$ is

$$Mark(Conf_{Ex}) = \pi(Cut(Conf_{Ex})) = \pi(\{c_5, c_7\}) = \{p_1, p_2\}.$$

In the following, we use a special kind of configurations, *local configurations*. Given an event its *local configuration* contains all events causally preceding this event.

**Definition 17** (Local configuration)**.** Given an occurrence net. The *local configuration* of event $e \in E$ is given by

$$[e] = \{e' | e' \leq e\}$$

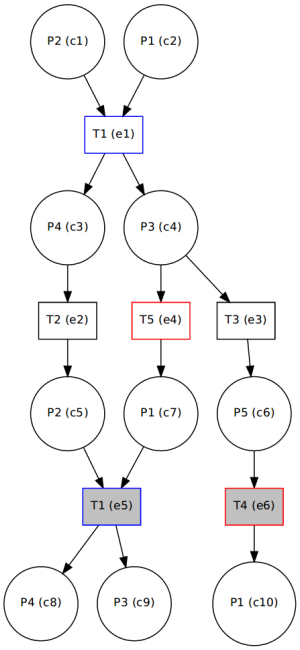
Regarding Example 5, we have for example

$$[e_6] = \{e_1, e_3, e_4, e_5, e_6\}$$

The idea on how to shorten the unfolding and obtain only a finite branching process still containing enough information is the following:

As we deal only with safe Petri nets, we have a finite number of different markings. What other transitions can be fired from a marking does not depend on how we reached it. So if we have two events whose local configurations have the same marking it suffices to explore only the future of one of these events. To decide which event is further extended we need an order on the local configurations. We call such a finite branching process *prefix*. A total adequate order "$\prec$" on configurations is suggested in [ERV96]. From now on we refer with the term "ERV-prefix" to the prefix generated with this total order according to the algorithm in the previously mentioned paper. We generate the ERV-prefix for a Petri net with the tool mole [Sch].

Now, we look at the Petri net from Example 1 and its ERV-prefix.




Example 6: A prefix of the Petri net from Example 1 and its ERV-prefix

Regarding the example we notice that $e_5$ and $e_6$ are not further extended due to the fact that the marking of their local configurations already appeared, as we have

$$\begin{aligned} Mark([e_6]) = Mark(\{e_1, e_3, e_6\}) &= \pi(\{c_3, c_{10}\}) \\ &= \{p_1, p_4\} \\ &= \pi(\{c_3, c_7\}) = Mark(\{e_1, e_4\}) = Mark([e_4]) \end{aligned}$$

We have a similar situation for $[e_5]$ and $[e_1]$.

We denote events such as $e_5$ and $e_6$ as cut-off events. An event is a *cut-off event* iff there exists another event whose local configuration is less with regard to the order $\prec$ and these two local configurations have the same marking.

**Definition 18** (Cut-off event)**.** A *cut off event* is an event $e_{cut}$ in the EVR-prefix such that there exists another event $e_{orig}$ which satisfy the following two constraints

- $Mark([e_{cut}]) = Mark([e_{orig}])$
- $[e_{orig}] \prec [e_{cut}]$

With $orig(e_{cut})$ we denote the most minimal $e_{orig}$ satisfying these two constraints. As $\prec$ used in the ERV-prefix is a total order for each cut-off event, such an event exists.

The cut-off events of Example 2.2 are the ones filled gray and the reason for the cut-off and their frame has the same color as the event that was the reason for the cut-off.

So, $e_6$ is a cut-off event because of event $e_4$ and $e_5$ is a cut-off event because of event $e_1$.

# 3 Cut Sets of a Petri net

In the following section we introduce the notion of cut sets of a Petri net. We first define what a (minimal) cut set is and illustrate this with an example.

Given a safe Petri net $\mathcal{P} = (P, T, F, M_0)$ and a bad place $p_b \in P$ a cut set is a set of places such that if there is never a token on these places there will never be a token on the bad place. Speaking from a biological perspective a cut set can propose a therapy: If we avoid a situation corresponding to having a token on a place of a bad event then we will never have the situation corresponding to having a token on the bad place. We specifically enforce that the cut set cannot contain any places that are part of the initial marking. Again, speaking from the biological perspective having a cut set that contains a place of the initial marking is not of a lot of use as we cannot avoid this behaviour.

Formally, a *cut set* is defined with regard to the unfolding $\mathcal{U} = (O, \pi)$ where $O$ is an occurrence net and $\pi$ a labelling function mapping the conditions of the unfolding to the places of the original net. A cut set is a set of places such that each condition of the unfolding labelled by the bad place is preceded by a condition labelled by a place of the cut set.

**Definition 19** (Cut set)**.** $K \subseteq P \backslash (M_0 \cup \{p_b\})$ is a *cut set* for $p_b \in P \backslash M_0$ if the following holds:

For all $c \in \mathcal{C}(\mathcal{U}(N))$ with $c \in \pi^{-1}(p_b)$ there exists $\pi(c') \in K$ with $c' \leq c$.

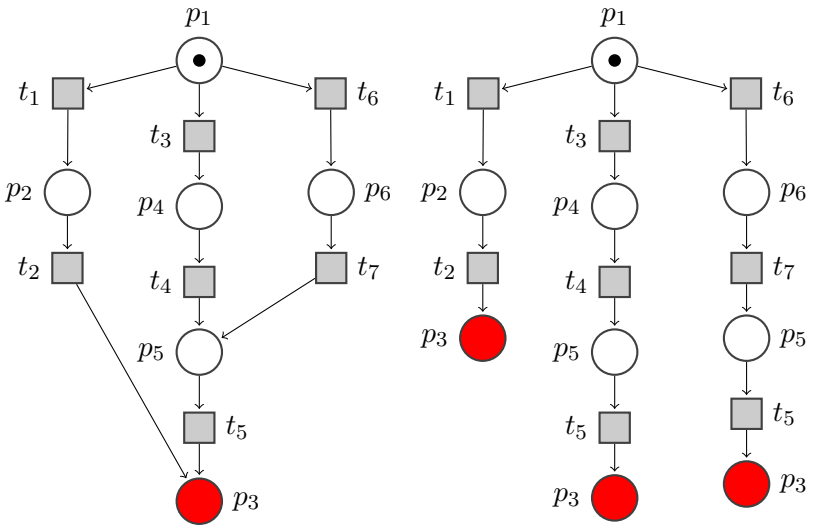
We illustrate this definition with an example. For Example 7 among others the following sets are cut sets for the bad place $p_3$: $\{p_2, p_4, p_6\}$, $\{p_2, p_5\}$ and $\{p_2, p_4, p_5\}$. The place $p_4$ in the last cut set $\{p_2, p_4, p_5\}$ is redundant as $\{p_2, p_5\}$ is already a cut set. This leads us to the definition of a *minimal* cut set. A cut set is minimal if none of its proper subsets is also a cut set.

**Definition 20** (Minimal cut sets)**.** Given a Petri net $N = (P, T, F, M_0)$ and a bad place $p_b \in P$ a cut set $K \subseteq (M_0 \cup \{p_b\})$ is *minimal* iff the following holds:

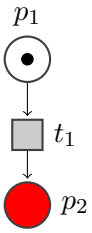
For all $K' \subsetneq K$, $K'$ is not a cut set for $N$ and $p_b$.

Beside minimality one can imagine other desirable properties for a cut set, e.g. it should be preferably small or it should not contain certain other places that we cannot prevent having a token.

As we excluded the initial marking and the bad place from any cut set we should keep in mind that there are Petri nets and bad places for which no cut set exists. See for this




Example 7: A Petri net (left) and its unfolding (right). Among others the following sets are cut sets for the bad place $p_3$: $\{p_2, p_4, p_6\}$, $\{p_2, p_5\}$ and $\{p_2, p_4, p_5\}$.




Example 8: Petri net that has no cut set

Example 8: The bad place is $p_2$. As a cut set must not contain any place that is initially marked there is no cut set here.

## 4 Computation of a Cut Set on the Petri Net

Cut sets are defined with regard to the unfolding of a Petri net. As the unfolding can be infinite using the unfolding to obtain cut sets is not useful. Therefore, we might consider using a finite prefix of the unfolding or the Petri net itself for computing cut sets. We first suggest an approach that is based on the Petri net and its reachable markings. The idea is to characterize properties a cut set has with regard to the Petri net and then formulate an ASP problem to find all minimal cut sets. In the second part of the section we study the computation of cut sets based on a unfolding prefix.

## 4.1 Properties of a Cut Set

For computing cut sets of a Petri net we can use the following relationship between a cut set and a Petri net: A set of places is a cut set iff there is no marking sequence from the initial marking to a marking that contains a bad place that does not also contain a marking with places from the cut set. We illustrate this with Example 7. In this example possible marking sequences from the initial marking to one that contains the bad place are the following:

$$\{p_1\} \xrightarrow{t_1} \{p_2\} \xrightarrow{t_2} \{p_3\}$$
$$\{p_1\} \xrightarrow{t_3} \{p_4\} \xrightarrow{t_4} \{p_5\} \xrightarrow{t_5} \{p_3\}$$
$$\{p_1\} \xrightarrow{t_6} \{p_6\} \xrightarrow{t_7} \{p_5\} \xrightarrow{t_5} \{p_3\}$$

In this example we have only finitely many marking sequences. That does not hold in the general case. The set $\{p_2, p_5\}$ is a cut set as there is no marking sequence from the initial marking $\{p_1\}$ to a marking that contains $\{p_3\}$ that does not see a marking that contains $p_2$ or $p_5$ beforehand. The set $\{p_4\}$ for example is not a cut set as the first and the third marking sequence do not contain the place $p_4$ in any marking but still reach the bad place $p_3$.

We can also modify the Petri net with regard to our potential cut set: We remove all transitions that are enabled by places in the potential cut set. If there is still a marking sequence which contains a marking putting a token on the bad place, we know that our set is not a cut set. In fact, the other direction is also true: If we have a set of places and we block all transitions enabled by these places and cannot reach a marking that contains the bad place then our set of places is a cut set.

**Theorem 2.** *Given a Petri net $N = (P, T, F, M_0)$, a bad place $p_b \in P$ and a set of places $K \subseteq P \backslash (M_0 \cup \{p_b\})$ the following holds:*

*$K$ is a cut set iff there is no marking $M_b$ with $p_b \in M_b$ reachable in $N' = (P, T', F', M_0)$ with*

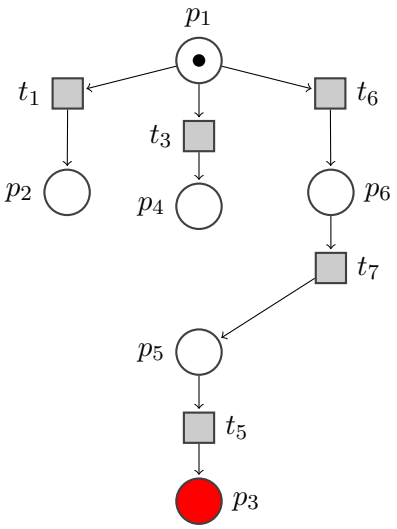
- $T' = \{t \in T | \ {}^\bullet t \cap K = \emptyset\}$
- $F' = \{(p, t) \in F | t \in T'\} \cup \{(t, p) \in F | t \in T'\}$

**Proof** We prove both directions by contraposition.

$\Rightarrow$ There is a marking, $M_b$, reachable in $N'$ with $p_b \in M_b$. Then there exists a marking sequence

$$M_0 \xrightarrow{T_b}{}^* M_b$$

in $N'$ with $T_b \subseteq T'$. We can find a configuration $C$ corresponding to that sequence in the unfolding of $N$ with Mark($C$)=$M_b$ and $\pi(C) = T_b$. None of the conditions




Example 9: The modified net for the set $K = \{p_2, p_4\}$ and the net of Example 7

preceding an event of the configuration are in $K$ as those places are not in the preset of any transition of the modified net. So, $K$ is not a cut set.

$\Leftarrow$ $K$ is not a cut set. Then there exists a condition $c$ in the unfolding with $\pi(c) = p_b$ and for all $c' < c$ there is $\pi(c') \notin K$. We look at the local configuration of the event $e$ with $c \in e^\bullet$ with $[e] = \{e_1, ..., e_k\}$ and $e_i \leq e_j$ for all $i < j$. This local configuration corresponds to a marking sequence

$$M_0 \overset{\pi(e_1)}{\rightarrow} \dots \overset{\pi(e_k)}{\rightarrow} M_b$$

As for all events there is $\pi({}^\bullet e) \cap K = \emptyset$ there is also ${}^\bullet\pi(e) \cap K = \emptyset$ so all $\pi(e_i) \in T'$ for all $i \in \{1, .., k\}$. So there is a marking reachable in $N'$ that contains $p_b$. $\square$

We illustrate this theorem with Example 7 and the set $\{p_2, p_4\}$. $\{p_2, p_4\}$ is not a cut set as we have in $N'$ with regard to $K = \{p_2, p_4\}$ still a marking sequence with a marking that contains the bad place (see Example 9):

$$\{p_1\} \overset{t_6}{\rightarrow} \{p_6\} \overset{t_7}{\rightarrow} \{p_5\} \overset{t_5}{\rightarrow} \{p_3\}$$

Therefore, the set $\{p_2, p_4\}$ is not a cut set.

To summarize what we found out: A set of places, not containing any initial places or the bad place, is a cut set iff there is no marking sequence containing a marking in which we have a token on the bad place in the Petri net that does not contain all transitions whose preset contains at least one of the places of the set.

## 4.2 Using Answer Set Programming to Compute Cut Sets

With the knowledge from the previous subsection in mind we can obtain all minimal cut sets using Answer Set Programming (ASP). The goal of this logic program is to find a cut set for a given Petri net and a bad place. Our program has a solution if there exists a cut set for the given Petri net and bad place. We characterize this solution by making use of Theorem 2. Our program has a solution if there is a set of places such that no initially marked place nor the bad place belong to that set and in the modified net (the one where we omit all transitions whose preset contains at least one place of the set) we cannot reach a marking that contains the bad place.

An ASP problem can be structured into three different parts [Lif08]. Regarding our application they have to reflect the following properties of cut sets:

- In the generate part the solution candidates are generated. All subsets of places are potential cut sets.
- In the test part we express constraints that have to hold for the solution. In our case, this is that no marking containing the bad place is reachable and that neither initially marked places nor the bad place are part of the cut set.
- In the define part we express rules to compute the reachable markings.

To obtain the answer sets (in our case cut sets) of the program we use the ASP solver clingo [GKK$^+$]. We illustrate the conception of the ASP instance by giving the instance for Example 7.

Our program contains different predicates and rules. We have the following predicates:

- $cutSet(p)$ where $p$ is a place and it is only true iff $p$ is in the cut set.
- $marking$ a predicate that has as many arguments as the net has places. Each argument can be set to "on" or "off" regarding whether the place represented by the argument is present in the marking or not.
- $isPlace(p)$ where $p$ is a place.

In the generate part we specify that all places could be on the cut set:

```
{ cutSet(P)} <= 1 :- isPlace(P).
```

In the define part we first specify the places and the initial marking as this is always reachable.

```
isPlace(p1). isPlace(p2). isPlace(p3).
isPlace(p4). isPlace(p5). isPlace(p6).
marking(on,off,off,off,off,off).
```

That logic program has a rule for each transition of the Petri net. That rule says that to fire the transition none of the places of the preset of transition are in the cut set

(otherwise they are not part of the modified net) and all of the places in its preset are marked. The rule tells furthermore that we can reach the marking which holds after we fired the transition.

The rule has the form:

```
marking_after_firing_transition:-
        marking_enabeling_transition , not cutSet(preset_of_transition).
```

When specifying the markings for a transition $t \in T$ we distinguish several cases for all $p \in P$:

- For all places that are neither in the preset nor in the postset of a transition, $p \notin (t^\bullet \cup^\bullet t)$, we use variables to specify whether there is a token on these places. The idea behind this is that none of these places effects whether $t$ is enabled and none of these places is effected by firing $t$. Looking at $t_1$ in Example 7 these are the places $p_3, p_4, p_5$ and $p_6$. We use the variables $P3, P4, P5$ and $P6$ to refer to whether these places hold a token or not.
- For all places that are in the preset of the transition, $p \in^\bullet t$, we specify that their status has to be `on` to enable the transition. Regarding $t_1$ is Example 7 this is the place $p_1$. Depending on whether these places are also in the postset of the transition, their status in the resulting marking is `on` or `off`.
- Places that are in the postset of the transition but not in its preset, $p \in (t^\bullet \backslash^\bullet t)$, prior to that, hold a token after firing the transition. So their status in the new marking is `on`. Before they do not hold a token as otherwise the net would not be one-bounded. For $t_1$ of Example 7 this is $p_2$.

Here the resulting rules for the transitions of Example 7:

```
% t_1
marking(off,on,P3,P4,P5,P6):-marking(on,off,P3,P4,P5,P6), not cutSet(p1).
% t_2
marking(P1,off,on,P4,P5,P6):-marking(P1,on,off,P4,P5,P6), not cutSet(p2).
% t_3
marking(off,P2,P3,on,P5,P6):-marking(on,P2,P3,off,P5,P6), not cutSet(p1).
% t_4
marking(P1,P2,P3,off,on,P6):-marking(P1,P2,P3,on,off,P6), not cutSet(p4).
% t_5
marking(P1,P2,on,P4,off,P6):-marking(P1,P2,off,P4,on,P6), not cutSet(p5).
% t_6
marking(off,P2,P3,P4,P5,on):-marking(on,P2,P3,P4,P5,off), not cutSet(p1).
% t_7
marking(P1,P2,P3,P4,on,off):-marking(P1,P2,P3,P4,off,on), not cutSet(p6).
```

In the test part, we specify that we should not be able to derive a marking containing the bad place. Furthermore, we enforce that the initially marked places and the bad place are not part of cut set.

```
:-marking(_,_,on,_,_,_).
```

```
:-cutSet(p1).
:-cutSet(p3).
```

By adding an optimization statement we could enforce that we want to obtain only cut sets that are minimal with regard to cardinality. We can generate these ASP instances automatically for a given net.

Taking everything into consideration, we showed that cut sets correspond to not reaching a marking that contains the bad place in a modified net. The modifications were to remove or block transitions enabled by places from the cut set. This led to the formulation of an ASP problem. The answer sets of this ASP problem are all minimal cut sets for a given Petri net and a bad place. The ASP problems can automatically be generated with regard to the specified Petri net and bad place. Furthermore, adding an optimization statement allows to obtain the smallest possible cut set(s) [GKK$^+$11].

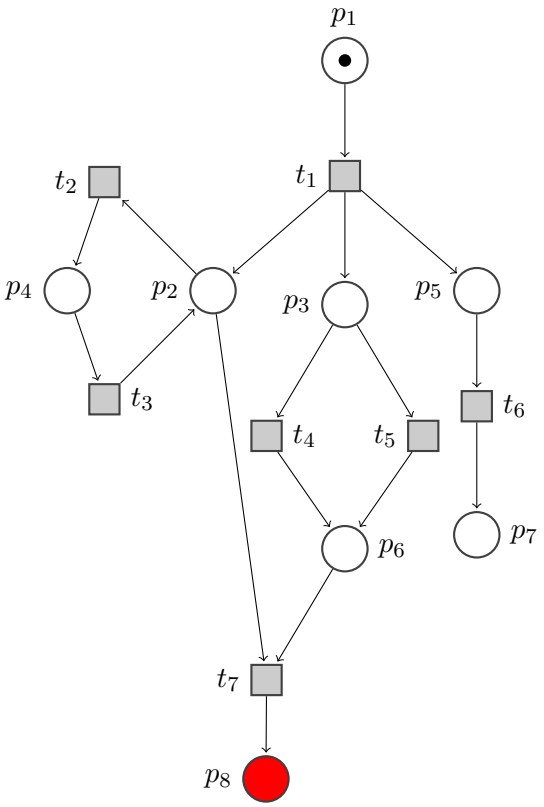
## 4.3 Computation of Cut Sets Based on the Goal-Driven Unfolding

In the previous subsection we have seen how we can compute cut sets by using the information of possible markings. This information can be obtained based on an ASP encoding of the Petri net. What we did was basically computing the marking graph on the fly. As the marking graph can be unnecessary large due to concurrent transitions, an unfolding prefix could be a more suitable structure for computing cut sets. This also corresponds better to the definition of cut sets: Cut sets are defined with regard to the unfolding. As the unfolding can be infinite in general we could use for our computation a finite structure related to the unfolding, hence a prefix of the unfolding. To compute cut sets it is essential to be able to restore the information by what configurations we can get a token on the bad place. However, not all configurations and transition sequences are relevant for this purpose. We can discard for example all transition sequences containing a cycle. In the following, we give a definition of cut sets based only on certain *minimal* configurations. The concept of minimal configurations and transition sequences was introduced in the context of a so-called goal-driven unfolding in [CP16]. Afterwards, we give an introduction into a goal-driven unfolding and illustrate how cut sets can be computed with one of its prefixes.

### 4.3.1 Prefixes Suitable for Computing Cut Sets

In this subsection we illustrate that only certain transition sequences are relevant for determining whether a set of places is a cut set. We motivate this concept with Example 10.

We analyse the Petri net of Example 10 with regard to the bad place $p_8$. Transition




Example 10: Petri net

sequences leading to the bad place are, among others:

$$\{p_1\} \xrightarrow{t_1} \{p_2, p_3, p_5\} \xrightarrow{t_4} \{p_2, p_5, p_6\} \xrightarrow{t_7} \{p_5, p_7\}$$
$$\{p_1\} \xrightarrow{t_1} \{p_2, p_3, p_5\} \xrightarrow{t_2} \{p_3, p_4, p_5\} \xrightarrow{t_3} \{p_2, p_3, p_5\} \xrightarrow{t_4} \{p_2, p_5, p_6\} \xrightarrow{t_7} \{p_5, p_7\}$$
$$\{p_1\} \xrightarrow{t_1} \{p_2, p_3, p_5\} \xrightarrow{t_2} \{p_3, p_4, p_5\} \xrightarrow{t_4} \{p_4, p_5, p_6\} \xrightarrow{t_3} \{p_2, p_5, p_6\} \xrightarrow{t_7} \{p_5, p_7\}$$
$$\{p_1\} \xrightarrow{t_1} \{p_2, p_3, p_5\} \xrightarrow{t_4} \{p_2, p_5, p_6\} \xrightarrow{t_7} \{p_5, p_8\} \xrightarrow{t_6} \{p_7, p_8\}$$
$$\{p_1\} \xrightarrow{t_1} \{p_2, p_3, p_5\} \xrightarrow{t_4} \{p_2, p_5, p_6\} \xrightarrow{t_6} \{p_2, p_6, p_7\} \xrightarrow{t_7} \{p_7, p_8\}$$

We can avoid the first transition sequence from happening by putting one of the places $\{p_1, p_2, p_3, p_6\}$ in the cut set and the second one by putting one of the places $\{p_1, p_2, p_3, p_4, p_6\}$ in the cut set. As the set resulting from the second transition sequence is a subset of the one from the first sequence we do not obtain any new information by looking at the second transition sequence. This also corresponds to the observation that the second transition sequence sees the marking $\{p_2, p_3, p_5\}$ twice. The third transition sequence uses the same transitions as the second one. So we do not gain any new information when looking at it. This corresponds to the observation that the third transition sequence is a permutation of the second one. When we look at the fourth transition

sequence we see that to prevent having a token on the bad place we have to block the transitions $t_1$, $t_4$ or $t_7$. $t_6$ is not relevant as it fires if there is already a token on the bad place. The fifth sequence is a permutation of the fourth one and gives us as such also no new information.

We refer to transition sequences that lead to the bad place like the first one with the term *minimal bad transition sequence*. The characteristics of a minimal bad transition sequence are the following:

- It reaches a marking in which the bad place is marked. In that sense all three transition sequences from above are bad.
- It does not repeat any marking. If a transition sequence sees the same marking twice we call it *cycling*. The second transition sequence is for example violating this criterion.
- There is no cycling transition sequence to the bad place that is a permutation of the transition sequence. The third transition sequence violates this criterion as it is a permutation of the second transition sequence that is cycling.
- There is no marking that puts a token on the bad place reached before the end of the sequence. The same holds for permutations of the transition sequence.

**Definition 21** (Bad, cycling and minimal transition sequence)**.** Let $N = (P, T, F, M_0)$ be a Petri net with a bad place $p_b \in P \backslash M_0$.

A *bad transition sequence* is a transition sequence $\sigma$ with $M_0 \overset{\sigma}{\rightarrow}^* M_b$ with $p_b \in M_b$.

The transition sequence $\sigma$ of $N$ is *cycling* if there is a marking $M'$ such that $\sigma$ it sees the same marking twice. Meaning $\sigma = \sigma_1\sigma_2\sigma_3$ with

$$M_0 \overset{\sigma_1}{\rightarrow}^* M' \overset{\sigma_2}{\rightarrow}^* M' \overset{\sigma_3}{\rightarrow}^* M_b$$

A bad transition sequence $\sigma$ is *minimal* if it is not cycling and all transition sequences that are permutations of $\sigma$ are not cycling. In addition, there is no marking that puts a token on the bad place reached before the end of the sequence. The same holds for permutations of the transition sequence.

We do not restrict permutations of transitions to transitions that are in an specific independent relation.

If we define minimal bad transition sequences we can also generalise this concept to configurations.

**Definition 22** (Minimal bad configuration)**.** Let $N = (P, T, F, M_{0_N})$ be a Petri net, $p_b \in P \backslash M_{0_N}$ a bad place and $U = (O, \pi)$ with $O = (C, E, F_O, M_{0_O})$ its unfolding.

A *minimal bad configuration* is a configuration $C$ that corresponds to a minimal bad transition sequence.

Now, we prove that a minimal bad configuration is always a local configuration. The intuition behind this is the following: Only one event of the local configuration is responsible for having a token on a condition corresponding to the bad place. In case a minimal configuration contained an event concurrent to this event we could obtain a shorter transition sequence by changing their order.

**Lemma 1.** *Let $N = (P, T, F, M_{0_N})$ be a Petri net, $p_b \in P \backslash M_{0_N}$ a bad place and $U = (O, \pi)$ with $O = (C, E, F_O, M_{0_O})$ its unfolding.*

*A minimal bad configuration $C_{min}$ is always a local configuration. Hence, there exists an event $e \in C_{min}$ such that for all $e' \in C_{min}$ there is $e' \leq e$.*

**Proof** Our proof consists of several steps each one proving a specific property of a minimal bad configuration $Conf_{min}$.

- $Conf_{min}$ contains exactly one event $e*$ with $p_b \in \pi(e*^\bullet)$.

  Assume that there are two (or even) more events with this property. Hence, there are $e_1, e_2$ with $p_b \in \pi(e_1^\bullet)$ and $p_b \in \pi(e_2^\bullet)$. Then $e_1$ and $e_2$ are in causal relation as otherwise the net was not safe. However, this would mean that $Conf_{min}$ was not minimal as we could construct a non-minimal bad transition sequence corresponding to it that reaches two markings in which the bad place is marked. So, there is at most one such event $e*$ with $p_b \in \pi(e*^\bullet)$.
  There has to be at least one event like this as otherwise the configuration was not bad.

- There are no events $e$ with $e* < e$.

  Assume there is an event $e$ such that $e* < e$. It follows immediately that we can construct a non-minimal transition sequence corresponding to the configuration. This would mean that the transition is not minimal.

- There are no events $e$ that are concurrent to $e*$. Assume there is an event $e \in Conf_{min}$ that is concurrent to $e*$. Then we could construct a transition sequence that looks like that:

  $$M_0 \overset{\pi(Conf_{min} \backslash \{e\})}{\rightarrow^*} M_b \overset{e}{\rightarrow} M_b'$$

  with $p_b \in M_b$ and $p_b \in M_b'$. However, this transition sequence is not minimal.

So all events $e$ of the configuration are in causal relation with $e$, $e* \leq e$. This means $Conf_{min} = [e*]$ and it is a local configuration. □

In the following, we prove a lemma stating that if we have a bad transition sequence that is not minimal, then there is a minimal bad transition sequence that uses only transitions that are also part of the original transition sequence. For Example 10 and the five transition sequences the first transition sequence is a minimal bad transition sequence and all the other transition sequences are not. Furthermore, the first transition

sequence uses only transitions that occur also in all the other non minimal bad transition sequences.

**Lemma 2.** *Let $N = (P, T, F, M_0)$ be a Petri net and $p_b \in P \backslash M_0$ a bad place.*

*For all finite bad transitions sequences $\sigma$ that are not minimal there exists a minimal bad transition sequence $\sigma_{min}$ such that for all $t \in \sigma_{min}$ there is $t \in \sigma$.*

**Proof** We can obtain a minimal bad transition sequence $\sigma_{min}$ for all non-minimal bad transition sequences $\sigma$ by iteratively applying one of the following steps distinguishing the different criteria that are violated:

- $\sigma$ is not minimal because we reach a marking with a token on the bad place before the end of the sequence.

  Hence, our transition sequence $\sigma$ has the following structure: $\sigma = \sigma_1\sigma_2$ with

  $$M_0 \overset{\sigma_1}{\rightarrow}{}^* M_{b_1} \overset{\sigma_2}{\rightarrow}{}^* M_{b_2}$$

  with $p_b \in M_{b_1}$ and $p_b \in M_{b_2}$. If $\sigma_1$ is already minimal we are done. Otherwise, we construct a minimal bad transition sequence for the bad transition sequence $\sigma_1$. As $\sigma_1$ contains only transitions from $\sigma$ this minimal bad transition sequence also contains only transitions from $\sigma$.

- $\sigma$ is not minimal because there is a permutation of it that reaches in an intermediate step a marking that puts a token on the bad place.

  We name this permuted transition sequence $\sigma'$. $\sigma'$ has the form $\sigma' = \sigma'_1\sigma'_2$ with

  $$M_0 \overset{\sigma'_1}{\rightarrow}{}^* M_b \overset{\sigma'_2}{\rightarrow}{}^* M$$

  with $p_b \in M_b$. If $\sigma'_1$ is already minimal we are done. Otherwise, we construct a minimal bad transition sequence for the bad transition sequence $\sigma'_1$. As $\sigma'_1$ contains only transitions from $\sigma'$ and hence from $\sigma$ this minimal bad transition sequence also contains only transitions from $\sigma$.

- $\sigma$ is not minimal because it is cycling.

  We remove the cycle and obtain by that a new bad transition sequence $\sigma'$. If $\sigma'$ is not minimal we construct a minimal bad transition sequence for it. As this minimal bad transition sequence contains only transitions from $\sigma'$ and $\sigma'$ contains only transitions from $\sigma$ we have then achieved our goal.

- $\sigma$ is not minimal because there exists a cycling permutation $\sigma_{cyc}$.

  If $\sigma_{cyc}$ is already bad we construct a minimal bad transition sequence for it which contains only transitions that are also in $\sigma$. If $\sigma_{cyc}$ is not bad we still know that it reaches some bad marking before the end as it contains a transition that has the

bad place in its postset (otherwise $\sigma$ was not bad). We cut $\sigma_{cyc}$ after reaching a marking that contains the bad place and continue our algorithm with that transition sequence. The so-obtained minimal bad transition sequence contains only transitions from $\sigma_{cyc}$ and thus, only transitions from $\sigma$.

As a non-minimal bad sequence always violates one of the four criteria above, we can always modify the sequence by one of the four procedures from above. In addition, if we have to apply several steps in a row we ensure that after each step we continue the algorithm with a bad transition sequence. As the transition sequences are always finite the algorithm terminates as we shorten the transition sequences with each step. □

In the following we show that for the computation of cut sets only minimal bad configurations are relevant. We do this by proving a theorem that states that a set of places is a cut set for a bad place if and only if the preset of each minimal bad configuration contains at least one place from the cut set.

**Theorem 3.** *Let $N = (P, T, F_N, M_{0_N})$ be a Petri net, $p_b \in P \backslash M_0$ a bad place and $U = (O, \pi)$ its unfolding with $O = (C, E, F_O, M_{0_O})$.*

*$K \subseteq P \backslash (M_{0_N} \cup \{p_b\})$ is a cut set iff for all events $e$ such that $[e]$ is a minimal bad configuration there exists $c \in {}^\bullet [e]$ with $\pi(c) \in K$.*

### Proof

⇒ If $K$ is a cut set then for all $c \in C$ with $\pi(c) = p_b$ there exists $\pi(c') \in K$ with $c' < c$. As $c$ has exactly one preceding event $e = {}^\bullet c$ this means that for all $e \in E$ with $p_b \in \pi(e^\bullet)$ there exists a $c' \in {}^\bullet [e]$ with $\pi(c) \in K$. This also includes all the minimal bad configurations as those are all local configurations by Lemma 1.

⇐ We prove this direction by contraposition.

If $K$ is not a cut set there exists $c \in C$ with $\pi(c) = p_b$ and for all $c' < c$ there is $\pi(c') \notin K$. Hence, there exists a finite local configuration $[e]$ with $e = {}^\bullet c$ with $\pi({}^\bullet [e]) \cap K = \emptyset$. If $[e]$ is a minimal configuration we are done. If it is not minimal there is a non-minimal bad transition sequence corresponding to it. With Lemma 2 we can find a minimal bad transition sequence using only transitions that are also in the non-minimal bad transition sequence. In addition, there exists a minimal configuration corresponding to that minimal bad transition sequence. This minimal configuration is the local configuration of an event $e_{min}$ as by Lemma 1 all minimal bad configurations are local. There is $\pi({}^\bullet [e_{min}]) \cap K = \emptyset$ because otherwise there was a $c' < c$ with $\pi(c') \in K$ as there only transitions used that are also in $[e]$. □

This theorem gives rise to a new method for computing cut sets as we can see in the next subsection.

### 4.3.2 Computing Cut Sets on a Prefix of the Goal-Driven Unfolding

In this subsection we introduce the concept of a goal-driven unfolding. In the last subsection we have shown that only certain configurations are necessary to compute cut sets. The characteristic of a goal-driven unfolding is that it contains all minimal bad configurations. Configurations that are no minimal bad configurations or not subsets of any of those are not necessarily preserved in the goal-driven unfolding. The concept as such and an algorithm for computing such unfoldings and prefixes was introduced in [CP16].

**Definition 23** (Goal-driven unfolding)**.** Let $N = (C, E, F_N, M_{0_N})$ be Petri net, $p_b \in P \backslash M_{0_N}$ a bad place and $U = (O, \pi)$ be the unfolding of $N$. A *goal-driven unfolding* $U_{gd}$ of $N$ is a branching process with $U_{gd} \sqsubseteq U$ that contains all minimal bad configurations of $N$ and $p_b$.

With the concept of goal-driven unfoldings we also allow smaller branching process than the unfolding to serve as a structure to obtain cut sets. However, as we do not enforce that only the minimal bad configurations and their subsets are part of the goal-driven unfolding it can still be infinite. Therefore, we suggest as an order for computing a prefix the subset-relation between two configurations. Before, we prove that such a prefix preserves all minimal configurations we illustrate the differences between the prefixes we have seen so far with Example 11. All the prefixes displayed in Example 11 are prefixes of the unfolding of the Petri net in Example 10.

In general, a prefix of the "normal" unfolding generated with the order "$\subseteq$" is larger than the ERV-prefix. The reason for that is that the total adequate order of [ERV96] is much more discriminaiting than "$\subseteq$". Unfortunately, it is too distinctive for our case as the ERV-prefix does not contain all minimal configuration as shown in Example 11.

The following theorem states that such a prefix is finite and preserves all minimal bad configurations.

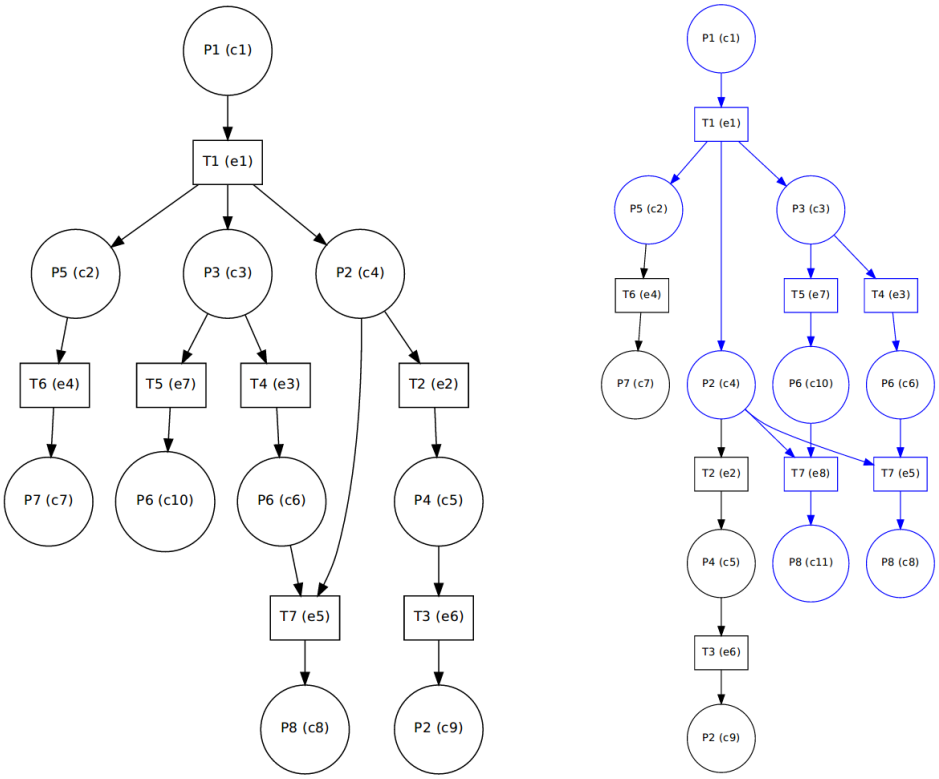
**Theorem 4.** *Let $N = (P, T, F_N, M_{0_N})$ be a Petri net, $p_b \in P \backslash M_{0_N}$ be a bad place and $U_{gd} = (O, \pi)$ one of its goal-driven unfoldings with $O = (C, E, F_O, M_{0_O})$.*

*If we construct the prefix of the unfolding $U_{gd}$ with the algorithm in [ERV96] taking as order "$\subseteq$" the prefix is finite and preserves all minimal configuration to $p_b$.*

**Proof** We prove the two properties separately.

- The prefix is finite.

  This proof can be found in [ERV96]. Although "$\subseteq$" is no adequate order as defined in that paper (because it does not preserve extensions) the proof holds in our case, too. The reason for that is that the proof of finiteness of the prefix only requires that the order preserves the subset relation.




Example 11: All the prefixes are with regard to the Petri net of Example 10 and the bad place $p_6$. On the left-hand side there is the ERV-prefix displayed. On the right-hand side a prefix based on the order "$\subseteq$" is shown. In blue are the parts of the prefix are those which belong to all prefixes of goal-driven unfoldings (or their isomorphisms) as they are part of minimal configurations.

- The prefix preserves all minimal configurations to the bad place $p_b$. Proof by contradiction.

  Assume there is a minimal configuration $C$ leading to the bad place that is not preserved. Then it contains a cut-off event $e_{cut}$. $e_{cut}$ is a cut-off event because there exists a mirror event $e_{orig} \neq e_{cut}$ with $[e_{orig}] \subseteq [e_{cut}]$ and $Mark([e_{orig}]) = Mark([e_{cut}])$. This also implies that $e_{orig} < e_{cut}$. That means our configuration corresponds to a transition sequence as follows:

  $$M_0 \rightarrow^* M' \overset{\pi(e_{orig})}{\rightarrow} M_{eq} \rightarrow^* M'' \overset{\pi(e_{orig})}{\rightarrow} M_{eq} \rightarrow^* M_b$$

  However, this sequence is cycling and as such not minimal. Contradiction. □

From the last two theorems there follows immediately that we can compute cut sets on the goal-driven prefix.

**Corollary 1.** *$K$ is a cut set iff there is no condition $c$ in the goal-driven prefix with $\pi(c) = p_b$ and for all $c' < c$ there is $\pi(c') \notin K$.*

The corollary gives rise to an ASP encoding of the problem.

### 4.3.3 ASP Encoding for Computing Cut Sets with a Prefix of a goal-driven Unfolding

In the following, we specify an ASP problem based on the prefix of the goal-driven unfolding. The answer sets of this problem should be all minimal cut sets. Opposed to the ASP problem directly based on reachable markings this time the ASP problem consists of two parts. The encoding is independent of the specified prefix and describes general properties of a cut set. The instance relates this encoding to the prefix computed beforehand.

We instantiate the prefix and encode the properties of a cut set with the help of the following predicates:

- $isPlace(p)$ where $p$ is a place of the underlying net.
- $cutSet(p)$ where $p$ is a place and it is true iff $p$ is in the cut set.
- $isBad(p)$ where $p$ is a place and the predicate is true iff $p$ is the bad place.
- $isCond(c)$ where $c$ is a condition of the prefix
- $isInitial(c)$ where $c$ is a condition and the predicate is true iff $c$ is initially marked
- $origin(c, p)$ where $c$ is a condition and $p$ is a place. The predicate is true iff $c$ is labelled by $p$.
- $isEvent(e)$ where $e$ is an event.
- $preset(n_1, n_2)$ where $n_1$ and $n_2$ are nodes of the prefix and $n_2 \in^\bullet n_1$.
- $precededbycutset(n)$ where $n$ is a node. This predicate is true iff there exists a node $n'$ with $cutSet(n')$ and $n' \leq n$.

In the ASP instance we have the predicates $isEvent(e)$, $isCond(c)$, $isPlace(p)$, $origin(c, p)$, $preset(n_1, n_2)$ and $isBad(p)$ describing the prefix on which the computation of cut sets is based on. For the goal-driven prefix of Example 11 (the blue prefix) the ASP instance looks as follows:

```
isPlace(p1). isPlace(p2). isPlace(p3). isPlace(p4). isPlace(p5). isPlace(p6).
isPlace(p7). isPlace(p8).

isBad(p8).

isCond(c1). isCond(c2). isCond(c3). isCond(c4). isCond(c6). isCond(c8).
isCond(c10). isCond(c11).
```

```
isInitial(c1).

origin(c1,p1).origin(c2,p5).origin(c3,p3).origin(c4,p2).origin(c6,p6).
origin(c8,p8).origin(c10,p6).origin(c11,p8).

isEvent(e1).isEvent(e3).isEvent(e5).isEvent(e7).isEvent(e8).

preset(e1,c1).preset(e3,c3).preset(e5,c6).preset(e5,c4).preset(e7,c3).
preset(e8,c4).preset(e8,c10).preset(c2,e1).preset(c3,e1).preset(c4,e1).
preset(c6,e3).preset(c8,e5).preset(c10,e7).preset(c11,e8).
```

The ASP-encoding is structured into a generate part, a define part and a test part.

The generate part contains a rule to specify that all places could be in the cut set:

```
{ cutSet(P)} <= 1 :- isPlace(P).
```

The rules of the define part formalize the predicate *precededbycutset*. A node is *precededbycutset* iff it is in the cut set or if a node of its preset is *precededbycutset*.

```
precededbycutset(C):-isCond(C), origin(C,P), cutSet(P).
precededbycutset(N):-preset(N,Npre), precededbycutset(Npre).
```

In the test part we enforce certain properties of the cut set:

- No initially marked place is in the cut set.
- The bad place is not in the cut set.
- There is no condition not preceded by the cut set that corresponds to the bad place.

This is ensured by the following encoding:

```
:-cutSet(P), origin(C,P), isInitial(C).
:-cutSet(P), isBad(P).
:-isBad(P), origin(C,P), not precededbycutset(C).
```

Taking everything into consideration, we saw how to compute cut sets based on a prefix. We introduced the concept of a prefix of a goal-driven unfolding that allows us to generate an ASP problem. The answer sets of this ASP problem are the minimal cut sets. We can also just generate the optimal cut set with regard to cardinality if we add an optimization statement to the ASP encoding.

# 5 Implementation, Complexity and Experimental Results of the Two Methods

In the last sections, we saw two ways of computing cut sets for a Petri net. Both methods generated an ASP problem. For the first method this ASP problem contained as atoms

the places of the Petri net and its initial marking. The rules referred to markings that we can reach by firing certain transitions. That way, for solving this ASP problem the marking graph (or at least parts of it) must be generated on the fly. The second approach was based on a prefix of a goal-driven unfolding. This prefix contained all minimal bad configurations. The ASP instance had atoms referring to the places of the underlying Petri net, conditions and events of the prefix, the labelling function of the prefix and the presets of conditions and events of the prefix. The ASP encoding contained two rules describing when a condition is preceded by a place of the cut set. The test part ensured then that all conditions corresponding to the bad place are preceded by a condition whose corresponding place is in the cut set. In the following, we describe the implementation, complexity and experimental results of the two methods and compare them.

## 5.1 Implementation

Both ASP problems were solved with the ASP solver clingo [GKK+]. Clingo takes as input an ASP problem and converts it into a propositional program by replacing variables with constants. Afterwards, answer sets are computed by a solver. In the following, we briefly describe how the ASP problems were generated in the two cases. The application for the algorithm intended by us is for biological networks.

- The ASP problem is directly generated from that Petri net.
- The encoding needed for the second method is not dependent on the specific Petri net or prefix respectively. The prefix is obtained with an algorithm that works on an automata network corresponding to the biological network. From this prefix the ASP encoding is generated.

## 5.2 Complexity

To compare the two methods we look at the time complexity of the two algorithms.

Both methods result in an ASP problem. Solving ASP problems is is NP-hard [Lif08]. However, the used ASP solver clingo (its built-in solver clasp respectively) implemented many heuristics. It is therefore considered an efficient tool in context of ASP solving [ASP]. The time to solve the ASP problem is dependent on the size of the specified instance and encoding [GKK+11].

For the first algorithm the time consuming step is solving the ASP problem. In this case we have $\mathcal{O}(|P|)$ constants, $\mathcal{O}(|T|)$ rules and $\mathcal{O}(|M_0|)$ tests. The second algorithm consists of mainly two steps. The first one is constructing the prefix and the second one solving the ASP problem that is generated based on this prefix. The prefix that is required for the algorithm is in general a lot larger than the ERV-prefix as the order we use is not total. The McMillan prefix can be exponentially larger than the original system [ERV96]. As the relation we use is strictly less distinguishing than the one used by McMillan our

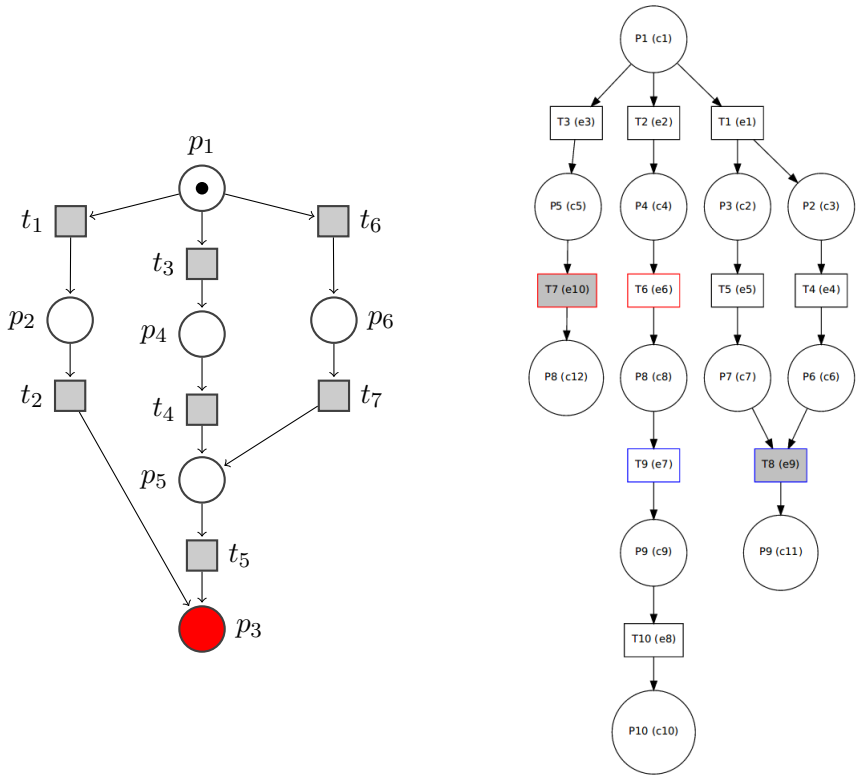
| Model | Cut Sets | Prefix | $\text{ASP}_{\text{pref}}$ | $\text{ASP}_{\text{mark}}$ |
|---|---|---|---|---|
| RB/E2F<br>$p_b =$ "$a837 = 1$", $\lvert P\rvert = 80$, $\lvert T\rvert = 54$ | 85 | 3 s | 0.2 s | 4 m |
| T-LGL<br>$p_b =$ "$PI3K = 1$", $\lvert P\rvert = 98$, $\lvert T\rvert = 159$ | 1 | 0.5 s | 0.03 s | OM |
| Mammalian Cell Cycle-10<br>$p_b =$ "$CycA = 1$", $\lvert P\rvert = 20$, $\lvert T\rvert = 35$ | 1 | OM | - | 0.1 s |

Table 1: Experimental Results. In the table above we show some experimental results for the two methods. We provide the model, the bad place specified by us and the size of the model. Furthermore, we show the number of cut sets that exist for the specific model and bad place. The column "prefix" refers to the computation time of the prefix, $\text{ASP}_{\text{pref}}$ refers to the time that was needed to solve the prefix based-ASP problem and $\text{ASP}_{\text{mark}}$ the time needed to solve the marking-based ASP problem. The ASP problems were solved with clingo. OM means that there was an out-of-memory error thrown during the computation. The results were obtained on an Intel©Core$^{\text{TM}}$i5 2.3 GHz, 8 GB RAM.

prefixes can be even larger. However, the reduction procedure discarding transitions as useless for reaching the goal prevents reduces the size of the prefix. Depending on the Petri net the goal-driven unfolding prefix can even be smaller than the McMillan prefix (Example 11. The ASP problem contains $\mathcal{O}(|P| + |C| + |E|)$ constants, 2 rules and 3 tests.

## 5.3 Experimental Results

In this section, we provide experimental results for the two methods. For the tests biological networks were used. In the table the models that were used for the tests are presented and their size described. Furthermore, Table 1 contains the computation times for computing the prefix, solving the prefix-based ASP problem and solving the marking-based ASP problem. The results show that we have cases where the prefix-based method is better and cases in which the marking-based method achieves better results. Furthermore, these result do not strictly correspond to the size of the network. This shows that the performance of the two methods depends more on the structure of the model. Especially, the model reduction procedure used for computing the goal-driven unfolding influences the overall performance of the second method heavily. As the model reduction procedure is not complete, it relies on structural properties of the underlying automata network. If transitions are not discarded although they are not part of minimal bad configurations, the prefix becomes unnecessary large.




Example 12: On the left-hand side a Petri net, on the right-hand side its ERV-prefix. Gray events are cut-off events and their frame colour is the same as the frame colour of the events that served as a reason for the cut-off.

# 6 Underapproximating Cut Sets on the EVR-Prefix

In the last subsection we have seen how we can compute a cut set by using ASP. The ASP problem was based on computing reachable markings and a goal-driven unfolding prefix, respectively. The goal-driven unfolding can in general be larger than the ERV-prefix. Therefore, we introduce an algorithm that works on the ERV-prefix. This algorithm the under-approximates the cut sets of a Petri net. Unfortunately, there is no method yet that is complete and correct for all Petri nets that is based on the ERV-prefix.

We illustrate the method with the prefix of Example 12.

The idea of the algorithm is based on the fact that we can easily construct a first cut set by looking at the prefix. We can simply study all conditions of the prefix labelled by the bad place and their preceding events. All sets that contain at least one place for each preset of the transition corresponding to these events are cut sets. We illustrate this in the context of Example 12 and the bad place $p_9$.

The conditions $c_9$ and $c_{11}$ correspond to the bad place $p_9$. The events preceding these

conditions are $e_7$ and $e_{11}$. They correspond to the transitions $t_9$ and $t_8$. The presets of $t_9$ and $t_8$ are $\{p_8\}$ and $\{p_7, p_6\}$. So the sets $\{p_8, p_7\}$, $\{p_8, p_6\}$ and $\{p_8, p_6, p_7\}$ are cut sets.

Crucial for the correctness of this method is the completeness property of the ERV-prefix. Completeness of a prefix means that every reachable marking is represented in the prefix and for all transitions that are enabled by a reachable marking there is some event in the prefix labelled by this transition. By our construction we ensure that for each transition that can occur and puts by firing a token on the bad place one place of the preset is in the cut set. We hereby prevent that any of these transitions can fire without coming across a place that is in the cut set.

To prove this proposition we first introduce several helpful definitions.

One of those is the notion of *relevant transitions* of a place $p$. This set is constructed with respect to a specific place. It contains all transitions that have in the postset this place $p$ and for which events in the prefix exist that are labelled by these transition. In Example 12 there is $\texttt{RelevantTransitions}(p_9) = \{t_9, t_8\}$ and $\texttt{RelevantTransitions}(p_{10}) = \{t_{10}\}$.

**Definition 24.** For a Petri net $N = (P, T, F_N, M_{0_N})$, its EVR-prefix $Pref = (O, \pi)$ with $O = (C, E, F_O, M_{0_O})$ with labelling $\pi$ and one of its places $p \in P$ the *set of relevant transitions* is defined as following:

$$\texttt{RelevantTransitions}(p) = \{t | \exists e \in E : \pi(e) = t \wedge \pi^{-1}(p) \cap e^\bullet \neq \emptyset\}$$

The second definition is the one of a combination set with regard to certain other sets. The idea is that if we have several sets a combination set is a set containing elements of each of these sets. It is minimal with regard to set inclusion.

**Definition 25.** For the sets $S_1, ..., S_n$ each $S \subseteq S_1 \cup ... \cup S_n$ is a *combination set* iff the following conditions are satisfied:

- For all $S_i \in \{S_1, ..., S_n\}$ there is $S \cap S_i \neq \emptyset$.
- There is no $S' \subsetneq S$ that is also a combination set.

Regarding our example the combination sets of the sets $\{p_8\}$ and $\{p_6, p_7\}$ are $\{p_6, p_8\}$ and $\{p_7, p_8\}$. $\{p_6, p_7, p_8\}$ is not a combination set as it fulfils the first condition regarding a non-empty cut with all the given sets but it is not minimal.

We proposed beforehand that all combination sets based on the presets of relevant transitions of the bad place are minimal cut sets. Additionally, we have to enforce that it does not contain any initially marked places nor the bad place.

**Theorem 5.** *For a Petri-net $N = (P, T, F, M_0)$, its EVR-prefix and a bad place $p_b \in P$ such that $\texttt{RelevantTransitions}(p_b) = \{t_1, ..., t_n\}$ each combination set $K$ based on ${}^\bullet t_1, ..., {}^\bullet t_n$ with $K \cap (M_0 \cup \{p_b\}) = \emptyset$ is a cut set.*

**Proof by contradiction** Assume there is such a combination set $K$ as described in the theorem that is not a cut set. Then there exists a condition $c_u$ in the unfolding of $N$ such that for all $c'_u < c_u$ there is $\pi(c'_u) \notin K$. We now look at the event preceding $c_u$, $e_u =^\bullet c_u$. We also have $\pi(c'_u) \notin K$ for all $c'_u \in^\bullet e_u$. That means $\pi(e_u)$ is not a relevant transition of $p_b$. However, $\pi(^\bullet c_u)$ can occur and has $p_b$ in its postset so it is a relevant transition. Contradiction. □

The idea for an algorithm that finds several minimal cut sets is to first explore the combination sets we get by looking at the presets of the relevant transitions of the bad place, checking the minimality of those and afterwards constructing other cut sets based on those cut sets.

Checking minimality of a cut set can be done by checking for each subset whether it is a cut set. This check can be done via constructing the prefix without the places in the potential cut set and checking the reachability of the bad place in this prefix. Constructing new cut sets from existing ones is done by replacing one place of the cut set by one of the combination sets based on the presets of the relevant transitions of this place.

We illustrate this procedure in the context of Example 12, this time with $p_{10}$ as the bad place:

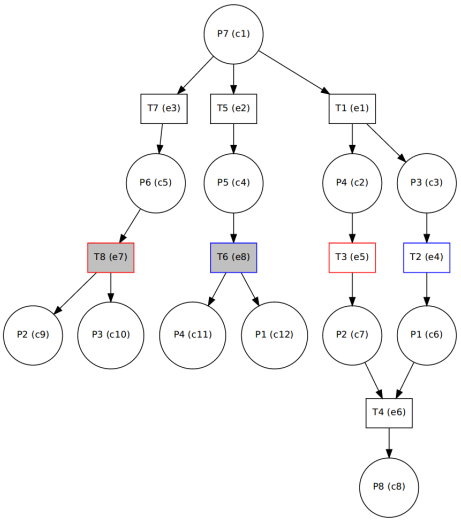
- `RelevantTransitions`$(P_{10}) = \{t_{10}\}$. This way we get initially the cut set $\{p_9\}$. This is minimal.
- Now, we replace $p_9$ by combination sets of the presets of its relevant transitions. We have `RelevantTransitions`$(p_9) = \{t_9, t_8\}$ and we obtain the new cut set $\{p_8, p_7\}$ and $\{p_8, p_6\}$. Both of these cut sets are minimal.
- We look at the cut set $\{p_8, p_7\}$. We have `RelevantTransitions`$(p_8) = \{t_7, t_6\}$ and the combination sets for $^\bullet t_7$, $^\bullet t_6$ are $\{p_5, p_4\}$. Replacing $p_8$ in the previously found cut sets by these new set gives us new cut sets $\{p_4, p_5, p_7\}$ and $\{p_4, p_5, p_6\}$. All of these new cut sets are minimal.

We could continue further but at this point we stop to formalize what we just did. We used a rule by which we construct new cut sets based on previous ones. We now formalize this rule and prove its correctness.

**Theorem 6.** *For a Petri-net $N = (P, T, F_N, M_{0_N})$, a bad place $p_b \in P$ and a minimal cut set $K$, $K_{new}$ defined as follows is a cut set:*

*There exists a place $p \in K$ with `RelevantTransitions`$(p) = \{t_1, ...t_n\}$ such that there exists a combination set $K_{com}$ of $^\bullet t_1, ...,^\bullet t_n$ with $K_{com} \cap (M_0 \cup \{p_b\}) \neq \emptyset$ and $K_{new} = (K \backslash \{p\}) \cup K_{com}$.*

**Proof by contradiction** Assume that a set $K_{new}$ we find that way is not a cut set. Then there exists a condition $c_u$ with $\pi(c_u) = p_b$ in the unfolding of $N$ such that for all




Example 13: ERV-prefix

$c'_u < c_u$ there is $\pi(c'_u) \notin K_{new}$. As $K$ was a cut set there must be a $c_p$ with $c_p < c_u$ and $\pi(c_p) = p$. However, the transition corresponding to the preceding event of $c_p$, $e_p =^\bullet c_p$, is a relevant transition of $p$. The reason for this is that $p$ is in the postset of $\pi(e_p)$ and there is an event labelled by $\pi(e_p)$ in the EVR-prefix as it is complete. As $K_{new}$ contains a combination set $K_{com}$ of the presets of the relevant transitions of $p$ there is a condition $c' \in^\bullet e_p$ with $\pi(c') \in K_{com} \in K_{new}$. Contradiction. □

The algorithm works as follows:

- We first construct all cut sets that can be obtained by Theorem 5.
- We check minimality of these cut sets. We continue only with the minimal cut sets we found.
- We apply iteratively Theorem 6 to all minimal cut sets we found so far. After each application we check minimality for the new cut sets and continue with the minimal cut sets we have not used yet for obtaining new cut sets with Theorem 6.

Unfortunately, this method is not complete as we show with Example 13. This prefix exposes the weaknesses of the method as this method does not obtain the cut set $\{p_3, p_4\}$ for the bad place $p_8$:

- With Theorem 5 we obtain the cut sets $\{p_2\}$ and $\{p_1\}$. Both of these cut sets are minimal.
- From the minimal cut set $\{p_1\}$ we obtain the cut set $\{p_3, p_5\}$ with Theorem 6. This cut set is minimal. There are no other cut sets that can be obtained by Theorem 6.
- From the minimal cut set $\{p_2\}$ and Theorem 6 we obtain the cut set $\{p_4, p_6\}$. This

cut set is minimal. There are no other cut sets that can be obtained by Theorem 6 and $\{p_4, p_6\}$.

So the minimal cut sets this methods finds are $\{p_1\}, \{p_2\}, \{p_3, p_5\}$ and $\{p_4, p_6\}$. It misses $\{p_3, p_4\}$.

Taking everything we have seen about this method into consideration, we remark the following aspects:

- This method benefits from the prefix by taking only transitions into account that can actually occur.
- The complexity of checking minimality of the cut sets we obtain by Theorem 5 and Theorem 6 is not yet estimated and could be a very time-consuming step.
- The method is incomplete (probably due to the fact that we do not reason about configurations but about single conditions and places).
- The method is probably efficient when there is no cut set. We notice this in the first step as there are no cut sets that can be obtained by Theorem 5.

The method over-approximates the set of processes. Hence, it returns an under-approximation of the minimal cut sets.

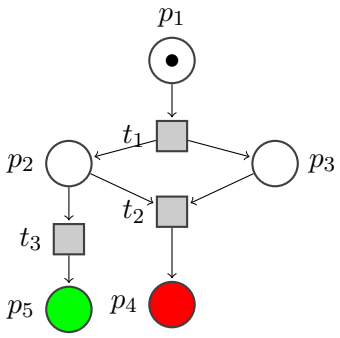
# 7 Control Sets

In the last sections we saw different approaches on how to avoid bad behaviour. However, we do not want to avoid bad behaviour at any costs. Maybe there is some good behaviour that we want or need to preserve. Speaking from a biological perspective this could be certain enzymes that are vital for a cell. This section will be dealing with the question how we can compute a control set of a Petri net with respect to a certain bad and a certain good place. We will show how the method based on the marking of the Petri net can be generalized to compute only sets that also preserve the good place.

Primarily, we define control sets. Control sets are defined analogously to cut sets. However, beside a bad place that need to be avoided there is also a good place that should be preserved. A control set is also defined with regard to the unfolding. A set of places is a control set with respect to a bad place and a good place iff the set is a cut set regarding the bad place and there exists a condition in the unfolding that corresponds to the good place and is not causally preceded by any place in the cut set.

**Definition 26** (Control Set)**.** Given a bad state $p_b \in P \backslash M_0$ and a good state $p_g \in P$, $K \subseteq P \backslash (M_0 \cup \{p_b, p_g\})$ is a *control set* iff

- $K$ is a cut set
- There exists $g \in \pi^{-1}(p_g)$ such that for all $k \in \pi^{-1}(K)$ there is not $k \leq g$.




Example 14: Petri net with bad place ($p_4$) and good place ($p_5$)

We illustrate the difference between a cut set and a control set with Example 14. The unfolding of this Petri net is finite and isomorphic to the Petri net itself. Cut sets of the Petri net with respect to the bad place $p_4$ are $\{p_2\}$ and $\{p_3\}$. However, $\{p_2\}$ prevents not only having a token on $p_4$ but also on $p_5$. As $p_5$ is the good place $\{p_2\}$ is not a control set. However, $\{p_3\}$ still allows putting a token on $p_5$. Hence, $\{p_3\}$ is a control set.

For cut sets we have proven a connection between the notion of reachable markings and the constraints on the unfolding in subsection 4.1. For the bad place and a potential cut set we have looked at a modified Petri net to get information on reachable markings. This modified net contained all places and transitions of the original net with one exclusion: All transitions that had at least one place of the cut set in its preset were removed from the net. We then derived a theorem that stated that a set is a cut set if there is no marking with a token on the bad place is reachable in this modified net (Theorem 2). We will extend that theorem and include that for a control set a marking containing the good place must be reachable in the modified net.

**Theorem 7.** *Let $N = (P, T, F, M_0)$ be a Petri net, $p_b \in P$ a bad place, $p_g \in P$ a good place and $K \subseteq P \backslash (M_0 \cup \{p_b, p_g\})$ a set of places.*

*$K$ is a control set iff $K$ is a cut set with respect to $p_b$ and there is marking $M_g$ with $p_g \in M_g$ reachable in $N' = (P, T', F', M_0)$ with*

- $T' = \{t \in T | {}^\bullet t \cap K = \emptyset\}$
- $F' = \{(p, t) \in F | t \in T'\} \cup \{(t, p) \in F | t \in T'\}$

**Proof**

$\Rightarrow$ If $K$ is a control set $K$ is also a cut set by definition. In addition, there is a condition $c_g$ in the unfolding such that for all $c \in C$ with $c \leq c_g$ there is $\pi(c) \notin K$. Hence, there is a local configuration $[e]$ with $e =^\bullet c_g$ such that ${}^\bullet\pi([e]) \cap K = \emptyset$. This implies that for all $e' \in [e]$ $\pi(e') \in T'$. So this configuration $[e]$ corresponds

to a transition sequence in $N'$ leading to a good marking $M_g$.

$\Leftarrow$ If there exists a reachable marking $M_g$ in $N'$ there exists a transition sequence

$$M_0 \overset{T_g}{\rightarrow}{}^* M_g$$

in $N'$ with $T_g \in T'$. This transition sequence corresponds to a finite configuration $C$ in the unfolding with Mark($C$)=$M_g$ and $\pi(C) = T_g$. None of the conditions preceding an event of the configuration are in $K$ as those places are not in the preset of any transition in the modified net. Fruthermore, $p_g \notin K$ by definition of $K$. Furthermore, $K$ is a cut set by definition. Hence, $K$ is a control set. □

This theorem gives rise to an ASP problem for obtaining all control sets for a Petri net, a bad place and a good place. For this purpose we extend the ASP problem of subsection 4.2 by another predicate *preserved*. This predicate is true iff there is some reachable marking in $N'$ with a token on the good place. In the test part of the ASP problem we enforce that *preserved* has to be true in order to obtain a control set. Furthermore, we enforce that the good place itself is not in the control set. Regarding Example 14 the ASP problem looks as follows:

```
isPlace(p1). isPlace(p2). isPlace(p3). isPlace(p4). isPlace(p5).
marking(on,off,off,off,off).
%t_1
marking(off,on,on,P4,P5):-marking(on,off,off,P4,P5), not cutSet(p1).
%t_2
marking(P1,off,off,on,P5):-marking(P1,on,on,off,P5), not cutSet(p2),
                                    not cutSet(p3).
%t_3
marking(P1,off,P3,P4,on):-marking(P1,on,P3,P4,off), not cutSet(p2).
:-marking(_,_,_,_,on,_).
:-cutSet(p1).
:-cutSet(p4).
:-cutSet(p5).
preserved:-marking(_,_,_,_,on).
:-not preserved.
```

The cut sets that are answer sets to this ASP problem are also control sets.

Regarding the tests we have run for computing control sets we found that in many cases no control set existed as all the cut sets that prohibited reaching the bad place were also directly blocking the good place.

# 8 Conclusion

In this report, we introduced the notion of cut sets of a Petri net with regard to one of its places. Furthermore, we suggested two methods for computing all cut sets and one for under-approximating it. The two complete methods differed in the way that

one was based on reachable markings whereas the other one relied on a goal-driven unfolding prefix. The experimental results have shown that there are for both methods cases where one outperforms the other. So it depends on the properties of the Petri net which method is more suitable. The under-approximation of cut sets done by a method that is based on the ERV-prefix of a Petri net has drawn attention to the difficulty of reconstructing configuration of the unfolding by performing backward steps in the ERV-prefix. Furthermore, in the last section we introduced the notion of control sets and suggested a method for computing them. From the practical perspective we saw that in many cases no such control sets exist if the networks are too dense.

Future work could involve extending the definition of cut sets and the existing methods to more than one bad place. Furthermore, it could also involve the computation of control sets based on the unfolding.

## References


[ASP] ASP competition 2015. `http://aspcomp2015.dibris.unige.it/results`. [Online; accessed 22-July-2016].

[CHJ+14] T. Chatain, S. Haar, L. Jezequel, L. Paulevé, and S. Schwoon. Characterization of reachable attractors using Petri net unfoldings. In Pedro Mendes, editor, *Proceedings of the 12th Conference on Computational Methods in System Biology (CMSB'14)*, volume 8859 of *Lecture Notes in Bioinformatics*, pages 129–142, Manchester, UK, November 2014. Springer-Verlag.

[CP16] T. Chatain and L. Paulevé. Goal-driven unfolding of petri nets, 2016.

[EH08] J. Esparza and K. Heljanko. *Unfoldings: A Partial-Order Approach to Model Checking (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[Eng91] J. Engelfriet. Branching processes of Petri nets. *Acta Iformatica*, 28(1):575–591, 1991.

[ERV96] J. Esparza, S. Römer, and W. Vogler. An improvement of mcmillan's unfolding algorithm. In *Formal Methods in System Design*, pages 87–106. Springer-Verlag, 1996.

[GKK+] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Clingo. `http://potassco.sourceforge.net/`.

[GKK+11] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam answer set solving collection. 24(2):107–124, 2011.

[HK11] O. Hädicke and S. Klamt. Computing complex metabolic intervention strategies using constrained minimal cut sets. *Metabolic Engineering*, 13(2):204–2015, 2011.

[Lif08] V. Lifschitz. What is answer set programming? In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 1594–1597, 2008.

[PAK13] L. Paulevé, G. Andrieux, and H. Koeppl. Under-approximating cut sets for reachability in large scale automata networks. In *Proceedings of the 25th International Conference on Computer Aided Verification*, CAV'13, pages 69–84, Berlin, Heidelberg, 2013. Springer-Verlag.

[Pau16] L. Paulevé. Goal-Oriented Reduction of Automata Networks. In *CMSB 2016 - 14th conference on Computational Methods for Systems Biology*, 2016. accepted.

[Rei13] W. Reisig. *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Springer Berlin Heidelberg, 2013.

[SBW06] L. J. Steggles, R. Banks, and A. Wipat. *Modelling and Analysing Genetic Networks: From Boolean Networks to Petri Nets*, pages 127–141. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[Sch] S. Schwoon. Mole. `http://www.lsv.ens-cachan.fr/~schwoon/tools/mole/`.

[SvKK10] R. Samaga, A. von Kamp, and S. Klamt. Computing combinatorial intervention strategies and failure modes in signaling networks. *Journal of Computational Biology*, 17(1):39–53, 2010.