Applying reduction rules to optimize conformance checking

Breukink, S.C.

Award date:
2019

Department of Mathematics and Computer Science
Information Systems Research Group

Applying Reduction Rules to Optimize Conformance Checking

Master Thesis

Sander Breukink

Supervisor:
Boudewijn van Dongen

Assessment committee:
Boudewijn van Dongen
Alexander Serebrenik
Renata Medeiros

Eindhoven, August, 2019

Abstract

The challenge of conformance checking is to determine the fitness of the combination of a petri net model and a corresponding event log. But for petri net models containing a high number of transitions and places and event logs containing a large amount of traces, this process can be rather computationally expensive and time consuming. This thesis proposes an approach to accelerate the task of conformance checking and reduce the computational power required to solve this problem.

In this thesis the existing ProM plug-in PNetReplayer is altered to apply reduction rules on data used. These rules are applied in 2 phases, in the first phase the rules are applied on the model and every trace in the log whereas in the second phase a subset of these rules is applied on the synchronous product used to compute the alignment and thus the corresponding fitness. In order to apply these rules, a novel reduction DAG data structure is used which is optimized to apply reductions to traces in a minimal amount of time. This altered implementation of the existing plug-in showed a significant decrease in required computational power for the test cases with the logs containing the highest number of traces, but a rather large increase of the required time for the smaller cases.

Table of Contents

List of Figures

List of Tables

Glossary

alignment A series that can contain synchronous moves, model moves and log moves, corresponding with several steps to terminate the corresponding model and so that all events in the corresponding trace occur. 2, 9–13, 18–20

event log A collection of traces, each trace again is a series of events corresponding to transitions in a petri net model, in this thesis often denoted as just "log", see section Event logs. 1, 6, 7, 23

fitness A value between 0 and 1 indicating the compatibility of a trace with a net, a 1 indicates that a trace matches perfectly, while a 0 indicates that the trace completely misses the model, which is the case for an empty trace. 2, 7, 9, 11–15, 17–20, 23

horizontal distribution Decomposing a petri net model into several submodels in order to speed up the process of conformance checking. 8

petri net model A model containing places, tokens, transitions and arcs connecting places and transitions, used to model various business processes, see section Petri net models. 1–3, 5, 6, 8, 23, 24

reduction DAG A directed acyclic graph that can be constructed for every petri net model, containing all possible reduction rules that can be applied to the net and reduced versions of the net in case a rule is applied, see chapter Reduction DAG. 3, 23

reduction rule A rule that reduces the problem of conformance checking while ensuring the same fitness result, see section Reduction rule. 2, 3, 7, 9–13, 17, 19, 20, 23, 24

sound A petri net model is sound if it is free of deadlocks, meaning that for any reachable state from the initial state, it is possible to reach the final state. 6

synchronous product A petri net model in which every transition is associated with a certain cost, it is computed to determine the alignment and fitness between a trace and a model, see section Synchronous product. 2, 6, 7, 14, 15, 17–21

vertical distribution To distribute the task at hand over multiple systems or cpu cores in order to speed up the process in comparison to performing the same task on a single system or cpu core. 8

workflow net Also called block structured net, a net with a clear initial and final state. The initial state is marked by a source place being the only place with a token, such a place has no incoming arcs. The final state is marked by a single sink place with no outgoing arcs which is the only place containing tokens. See section Workflow petri nets.. 6

Chapter 1

Introduction

In this chapter, an introduction to the subject and the related research domains is given. Firstly, the section Domain of process mining provides a brief introduction to the field of process mining. Secondly, in Subdomain of conformance checking, the subdomain of conformance checking is clarified. In section Problem description and motivation the challenge of this thesis is explained and motivated and in the section Approach the set-up used to implement the solution is described. Lastly, the Thesis summary contains a summary of the main contribution of this thesis.

1.1 Domain of process mining

The domain of process mining is based on creating a model from event log data. This event log data can originate from all kinds of business processes, a typical example is that of a loan application were such an application can be submitted, analysed, accepted or declined. A log consists of multiple traces, in which each trace represents a run through the system, which in the earlier mentioned example would be a single loan application. Each trace consists of labelled events, each event represents a step in the process.

When applying process mining to such an event log, a petri net model is created based on the data provided. This model allows to replay all the events in the log. A far more extensive explanation of process mining can be found in [19].

1.1.1 Example

See the following log consisting of 10 traces:
$\langle t1, t3, t5, t4, t6 \rangle$
$\langle t1, t3, t4, t5, t6 \rangle$
$\langle t1, t2, t5, t4, t6 \rangle$
$\langle t1, t2, t5, t4, t6 \rangle$
$\langle t1, t2, t5, t4, t6 \rangle$
$\langle t1, t2, t4, t5, t6 \rangle$
$\langle t1, t3, t5, t4, t6 \rangle$
$\langle t1, t3, t5, t4, t6 \rangle$
$\langle t1, t3, t5, t4, t6 \rangle$
$\langle t1, t3, t4, t5, t6 \rangle$

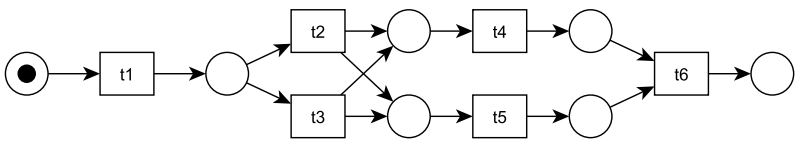

Figure 1.1: A petri net model resulting from applying process mining to a log.

Applying process mining to this log will result in the model shown in figure 1.1. Note that in practice, a log will probably consist of a significantly higher number of traces, traces are likely to

contain more events and that not all traces will perfectly align the model. However, these kinds of deviations when creating such a model are beyond the scope of this research.

1.2 Subdomain of conformance checking

When checking the conformance, the required input not only consists of a log, but also of a petri net model. Note that it is not essentially required that this model is created by using process mining, but in most occurrences this will be the case. The goal of conformance checking is to determine how well traces in a log fit the model, this yields a value between 0 and 1, which is defined as fitness. This fitness is calculated based on the number of model moves, log moves and synchronous moves, in which each move has its user-defined cost. In this report it is assumed that all costs of non-synchronous moves are 1, unless explicitly defined otherwise, also note that they are frequently omitted.

The maximum cost of a trace is defined as the cost of all model moves needed to successfully terminate the model + the cost of all model moves in the trace as if none is matching with the model. The actual cost is defined as the sum of the costs of the model moves and log moves actually needed. The fitness can be calculated as follows: fitness $= \frac{maximum\ cost - actual\ cost}{maximum\ cost}$. A far more extensive explanation of conformance checking can be found in [10].

1.2.1 Example

Take, again, the model shown in figure 1.1. However, this time, take the following log:
0: $\langle t1, t1, t3, t5, t4, t6 \rangle$
1: $\langle t1, t3, t5, t6 \rangle$
2: $\langle t1, t2, t2, t4, t6 \rangle$
3: $\langle \rangle$
4: $\langle t1, t2, t5, t4, t6 \rangle$

It can easily be observed that only the last trace matches the model perfectly. However, if the alignments are computed, the following results are obtained:
0: $\langle t1, t1, t3, t5, t4, t6 \rangle$
1: $\langle t1, t3, t4, t5, t6 \rangle$
2: $\langle t1, t2, t2, t5, t4, t6 \rangle$
3: $\langle t1, t2, t5, t4, t6 \rangle$
4: $\langle t1, t2, t5, t4, t6 \rangle$

In this visualization the model moves are coloured blue, the log moves are coloured green and the synchronous moves remain black. The next step is to calculate all of the fitnesses, the shown alignments yield in the following calculations:
0: $\frac{11-1}{11} = 0,91$ with maximum cost $= 6 + 5 = 11$
1: $\frac{9-1}{9} = 0,89$ with maximum cost $= 4 + 5 = 9$
2: $\frac{10-2}{10} = 0,80$ with maximum cost $= 5 + 5 = 9$
3: $\frac{5-5}{5} = 0,00$ with maximum cost $= 0 + 5 = 5$
4: $\frac{10-0}{10} = 1,00$ with maximum cost $= 5 + 5 = 10$

1.3 Problem description and motivation

As models and corresponding logs can grow rather large in practice, running the process of conformance checking can be a time consuming job. Hence it has become relevant to research whether this process could be accelerated. In the current implementation, a synchronous product is created for the model and every unique trace in the log. Although this results in very reliable and

comparable balanced fitness result for each unique trace, it is a computationally intensive task. Note that the reduction rules used in this report were already known when this method was implemented and have been successfully applied numerous times to reduce problems with regards to petri net models. Because of this it is a logical choice that this approach is also used for reducing the computational time required to run conformance checking.

1.4 Approach

This work is an extension of the research done in [5]. The solution is created as an adaptation of the PNetReplayer plug-in for ProM Tools, more information on this software can be found in [1] and [12]. This plug-in has been altered at 2 points. The first alteration is directly at the start, here the log and net are reduced using the reduction rules in described in chapter Phase 1 Reduction Rules, as a part of this the software also analyses the petri net model to create the reduction DAG. The second alteration is at the creation of the synchronous product which is reduced using the rules described in the chapter Phase 2 Reduction Rules.

1.5 Thesis summary

In the Introduction chapter, the fields of process mining and conformance checking are briefly introduced. Furthermore the problem description is given, explaining that conformance checking can be a time consuming task and that it is a process that can be accelerated. Then the general approach is explained and the thesis is summarized.

The next chapter Background Information introduces the terminology used in this thesis. The most relevant terms are petri net model, event log and synchronous product. Previous work on this problem and related work are also mentioned.

The 6 main reduction rules are explained in Phase 1 Reduction Rules, here they are applied to the net and log before any alignment is computed. The first 2 rules, namely fuse series places and fuse series transitions, merge places and transitions that occur consecutively. The second 2 rules, namely fuse parallel places and fuse parallel transitions, merge places and transitions that are similar in sources and targets. The last 2 rules, elimination of self-loop places and elimination of self-loop transitions, eliminate self-loops in the net and alter the corresponding log.

The chapter Phase 2 Reduction Rules explains reduction on the synchronous product. For this 3 of the aforementioned rules can be applied, namely fuse series places, fuse series transitions and fuse parallel transitions. These variants of the same rules have less strict preconditions and can thus be applied to a broader set of inputs.

To apply the rules in an efficient manner a DAG data structure is implemented. This data structure is presented in the Reduction DAG chapter. The chapter contains the main motivation behind this data structure, the process of creating it and the way it is applied. As this is a novelty, an example is provided to clarify the process.

In the chapters Performance Evaluation and Conclusions a testing set is shown, the implementation is evaluated and compared with the conventional approach. From this comparison it follows that applying reduction rules is beneficial when large logs are involved, but is a disadvantage when smaller logs are used. Lastly, suggestions for future work are given.

Chapter 2

Background Information

In this chapter several terms are introduced that are used throughout this thesis. Firstly, the very basic elements of process mining and conformance checking are briefly introduced in sections Petri net models and Event logs. Afterwards, the terms more relevant for reducing the problem of conformance checking are explained in the sections Synchronous product and Reduction rule. For every term an example is given. In the last 2 sections Previous work and Related work, comparable work is referenced.

2.1 Petri net models

Petri net models, or for short: petri nets, were firstly defined in [20] and provide a method to model all kinds of business processes. A petri net consists of places, tokens, transitions and arcs, places can contain 0, 1 or multiple tokens. Arcs always connect a place with a transition and always have a direction. Note that a consequence of this is that if the source of an arc is a place, the target will be a transition and vice versa.

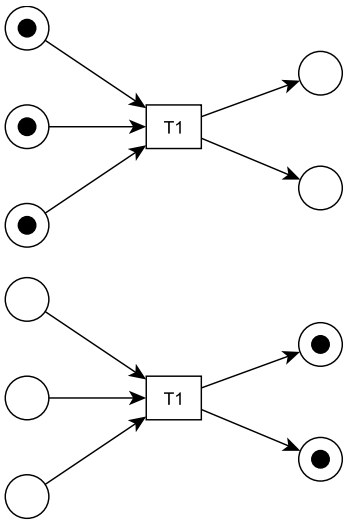

Figure 2.1: An example of a petri net transition that fires, in this example 3 tokens are consumed and 2 tokens are produced

All places connected to arcs that have a transition as its target, are called input places or source places for that respective transition. All places connected to arcs that have a transition as its source are called target places or output places. A transition is enabled if all of its input places contain a token. If a transition is enabled, it can fire, which consumes a token from every input place, but produces a token in every target place. An example of this is shown in figure 2.1. Following this definition, a place has input transition, also called source transitions and output transitions, also called target transitions.

An example of a petri net model is shown in figure 2.2, note that this is an adapted model from the model used in [10]. To prevent distraction from the main topic, more abstract names for transitions and places are used during the rest of this thesis. For the same reason, place labels are frequently omitted, as they often lack relevance.

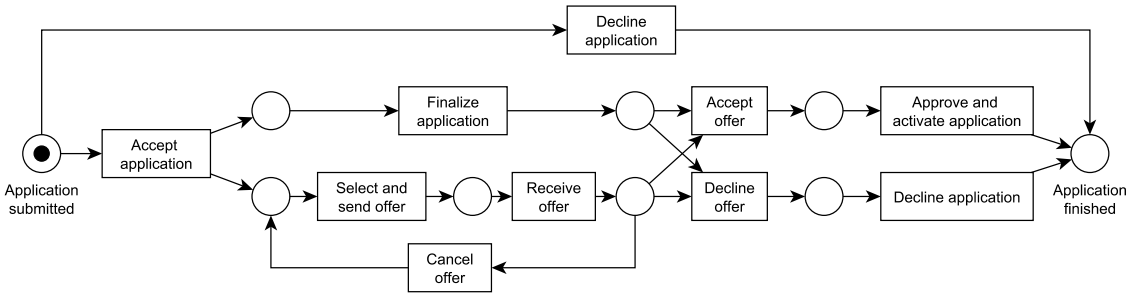

Figure 2.2: An example of a petri net model that models a loan application. Note that this is an adapted version of the model used in [10].

2.1.1 Workflow petri nets

This research mainly focusses on workflow nets that are sound, additional information on workflow nets can be found in [15]. These are nets that have a clear initial state and final state in which no deadlocks can occur. In both the initial and the final state, there can be at most one place that contains tokens. The place containing tokens in the initial state is called the source place and the place containing tokens in the end is called the sink place.

The source place can be recognized by the fact the fact that it contains a token in the initial state and by the lack of incoming arcs, see the place "application submitted" in figure 2.2. The sink place can also be recognized by the lack of any outgoing arcs, in other words: if all tokens in the model have arrived at the sink place, there are no more transitions that can fire and the model is terminated, in figure 2.2 this place is labelled "Application finished".

2.2 Event logs

An event log, or for short: log, contains several traces, each trace represents a single replay on the corresponding model. Each trace consists of several events, each events corresponds to a transition firing in the model. An example of a log is shown in the subsection of Subdomain of conformance checking.

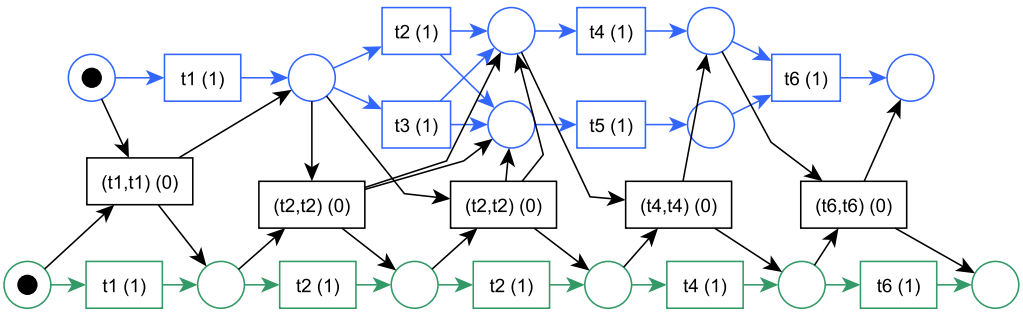

Figure 2.3: A synchronous product for the trace $\langle t1, t2, t2, t4, t6 \rangle$ and the model shown in figure 1.1. Every log move and model move are assigned a cost of 1 and every synchronous move is assigned a cost of 0.

2.3 Synchronous product

To determine the fitness of a trace and corresponding model, a synchronous product is created in the form of a petri net. Such a product consists of model moves, log moves and synchronous moves, each of these moves have a cost assigned to them. An algorithm solves this synchronous product with the lowest cost possible, this cost is then used to calculate the fitness, note that there can be multiple optimal solutions. Ideally the cost is equal to 0, which is the case if the trace matches the model perfectly because synchronous moves typically have a cost of 0. However, if a trace does not match the model perfectly, log moves and/or model moves have to be used, which typically have a cost greater than 0. The synchronous product of the model in figure 1.1 and trace 2 mentioned in the example of Subdomain of conformance checking, is shown in figure 2.3. All model moves are shown in blue, all log moves are shown in green and all synchronous moves are shown in black, the respective cost of each transition is shown after the label in between brackets. Note that a model can have duplicate labels, so multiple transitions with the same label, in such a case a synchronous transition is created for every combination of a transition and event with the same label. So if a model contains 2 transitions with the label A, then there will be 2 synchronous transitions for every event labelled A in the trace.

2.4 Reduction rule

In this thesis, the main goal of a reduction rule is to simplify the problem of conformance checking. This can be achieved by merging places and transitions, which implies merging events in a trace. The result of this is a faster computation of the fitness while still keeping the same score in the end result. In this thesis there are 2 kinds of reduction rules, the first kind, described in chapter Phase 1 Reduction Rules, takes a log and a model as input and reduces both of them, but keeps them separated. The second kind, described in chapter Phase 2 Reduction Rules, looks at the synchronous product and applies reductions on that level. Take note that phase 1 reductions are preferable as they are less computational intensive and can often be applied on multiple traces whereas the phase 2 reductions have to be calculated for every synchronous product specifically.

2.4.1 Example

Take again the model of figure 1.1, here the transitions $t2$ and $t3$ can be merged into a single transition $t2 \vee t3$ which has the same source place and target places as either of the transitions, the result is shown in figure 2.4. If this reduction is then applied to the event log, it means that every event with the label $t2$ or $t3$ is replaced by an event with the label $t2 \vee t3$. This is an example of a phase 1 reduction rule.

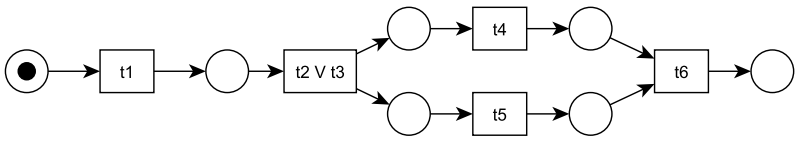

Figure 2.4: The model of figure 1.1 in which $t2$ and $t3$ have been merged.

In this reduced model, the transitions $t1$ and $t2 \vee t3$ can be merged into a single transition $t1 \wedge (t2 \vee t3)$. This implies that for each trace the 2 corresponding events have to be merged into a single event, this however, is not possible for the trace $\langle t1, t4, t6 \rangle$. Hence this reduction can only be applied on the synchronous product by creating a synchronous transition with a cost equal to the cost of $t2 \vee t3$, this can be observed in figure 2.5.

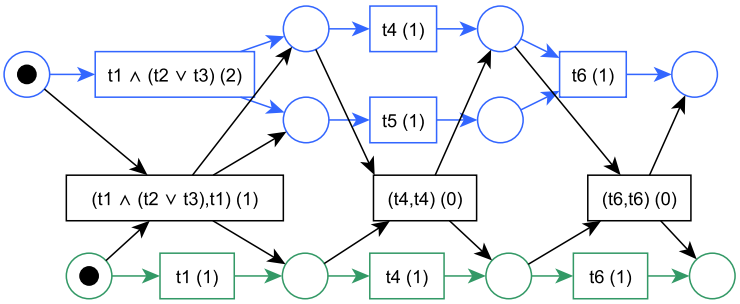

Figure 2.5: The synchronous product of the trace $\langle t1, t4, t6 \rangle$ and the model of figure 2.4 in which $t1$ and $t2 \vee t3$ have been merged.

2.5 Previous work

Previous work with respect to reduction has been carried out in [6], the approach used there however, is fundamentally different. Another approach which applies reduction rules on time petri nets is presented in [13], but this approach does not take the element of conformance checking into account. Besides these 2 approaches, there is of course the already mentioned report in which the reduction rules were initially defined, namely [8], but this research does not apply these rules to the field of conformance checking.

2.6 Related work

There have been several approaches to accelerate computations related to petri net models. An idea to speed up the process of conformance checking, is to split up the log and do the process in parallel, note that this can be distributed over multiple cpu cores, but also over multiple computer systems. This is approach is called vertical distribution and is applied in [16] and [4].

Another approach is horizontal distribution, also known as divide and conquer, which is applied in [11], [17], [9] and [18]. This approach divides petri net models and the corresponding traces into smaller parts, solving several smaller problems rather than one big problem. This is a common approach to reduce the required time.

A third approach to accelerate conformance checking is to not solve the problem exactly, but compute an approximation which requires far less computational power. An example of such an approach can be found in [7]. Note that this approach focusses on returning multiple optimal alignments, whereas this research only requires a single optimal solution.

Chapter 3

Phase 1 Reduction Rules

In this chapter, the different reduction rules that are applied in the first phase to reduce the problem of conformance checking are presented. These rules are inspired by the reduction rules that preserve properties that were firstly defined in [8]. Observe that these rules were originally written for petri nets the reductions mentioned in this thesis are applicable to the combination of petri nets corresponding logs. As there is a lot of overlap in the preconditions, all of them are motivated in the section 3.7. For all of these rules, the correctness proof is based on the assumption that fitness is calculated on the cost of an alignment and that the cost of these alignments will stay the same if the reduction is applied.

3.1 Fuse series places

The general idea of this reduction is that if a token in place A can only go to place B, the 2 places can be merged. The transition in between them can be merged with the input transitions of A or the output transitions of B. Take the example model of figure 3.1, then the result of the first version of this reduction is shown in figure 3.2, whereas the result of the second version is shown in figure 3.3.

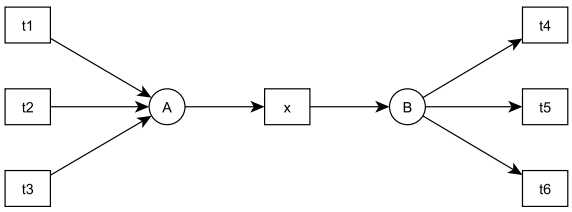

Figure 3.1: A model on which the fuse series places reduction can be applied.

Let place A and place B be the places to be merged, let T be the transition in between them and let F be the set of transitions that T will be fused with. Note that F consists of all input transitions of place A or all output transitions of place B. The fuse series places is applied if the following preconditions are met:

- The set F is not empty.
- For all occurrences of T and events corresponding to transitions in F in the trace it holds that they occur consecutive in the trace, in other words: no other events occur in between.
- For the transition T and all transitions F it holds that they are unique in the model, so no other transitions with the same label occur.

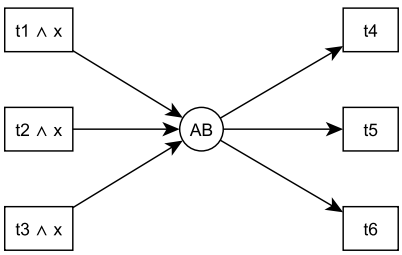

Figure 3.2: The model of figure 3.1 on which the fuse series places reduction has been applied, in this case the transition in between A and B has been merged with the source transitions of A.

If the rule is applied, the following postconditions will hold afterwards:

- In the model, the transition T is merged with all the transitions in set F, reducing the number of transitions by 1.
- In the trace, all occurrences of transition T are merged with their respective predecessor or successor in set F, reducing the trace length by the number of occurrences of transition T.
- The cost of the transition T is added to all of the transitions in the set F in the model.
- All of the merged events in the log have a cost equal to the sum of the costs of the 2 original events.

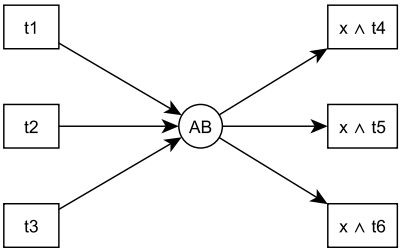

Figure 3.3: The model of figure 3.1 on which the fuse series places reduction has been applied, in this case the transition in between A and B has been merged with the target transitions of B.

Correctness proof
Due to the preconditions, it can be assumed that the events corresponding to the transitions that are merged always occur consecutively. Hence it holds that if 1 of the 2 events in the trace fits the model, then so does the other, but also: if 1 of the 2 events in the trace does not fit the model, then the other event will not either. Because of this, merging the 2 events in the trace and the 2 transitions in the model, will change the alignment cost, given that the costs of the events or transitions are added during the merge.

3.2 Fuse series transitions

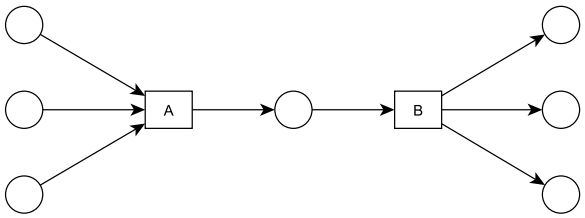

Figure 3.4: A model on which the fuse series transitions reduction can be applied.

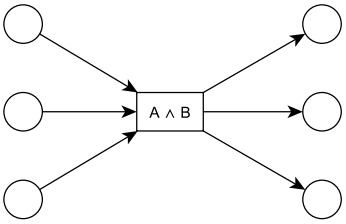

Figure 3.5: The reduced version of figure 3.4.

In the fuse series transitions reduction, 2 transitions are merged if they are connected by a single place with 1 input arc and 1 output arc. This implies that if the first transition fires, the second transition will always fire afterwards. A graphical representation of this merge can be observed in figures 3.4 and 3.5.

Let A and B be transitions in the model (or events in the trace) and let P be a place in between them. This reduction rule can be applied if the following preconditions are met:

- P has exactly one input arc with A as its source and exactly one output arc with B as its target.
- Transition A has exactly one output arc with P as its target and transition B has exactly one input arc with P as its source.
- All occurrences of A and B in the trace are consecutive, in other words: all A events are directly followed by a B and all B events occur directly after an A.
- The transitions A and B are unique, meaning no other transitions occur in the model with the same label.

If all of these transitions are met, the rule can be applied and transitions A and B can be merged into one. This implies the following postconditions:

- The place that was in between transitions A and B is removed.
- Transitions A and B are merged into a single transition $A \wedge B$, this merged transition has the same input arcs as the original transition A and the same output arcs as the original transition B.
- All occurrences of A and B in the trace are merged into a single event $A \wedge B$, which reduces the trace in length by the number of occurrences of each of this event.
- The costs of A and B are added up to each other in both the model and the trace and form the cost of the new transition and event.

Correctness proof
Due to the preconditions, it can be assumed that the merged events always occur consecutively. It also holds that if 1 of the 2 events in the trace fits the model, then so does the other, but also vice versa. therefore, merging the 2 events in the trace and the 2 transitions in the model does not affect the alignment cost, given that the costs of the events and transitions are added during the merge.

3.3 Fuse parallel places

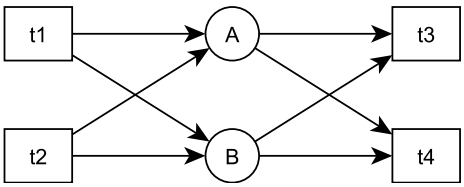

Figure 3.6: A model on which the fuse parallel places reduction can be applied.

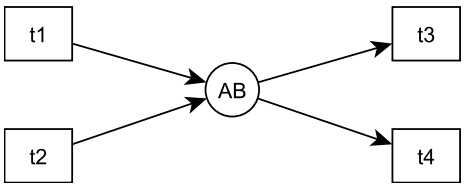

Figure 3.7: The reduced version of the model shown in figure 3.6.

The fuse series places reduction rule will not affect the trace, but it is possible that preconditions of other reduction rules are enabled because of this. The idea is that duplicate places are merged into a single place. This is graphically represented in figures 3.6 and 3.7. For this reduction rule to be applied the following preconditions must hold for 2 places A and B in the model:

- For every input arc of place A, there is an input arc to place B that has the same source.
- For every input arc of place B, there is an input arc to place A that has the same source.
- For every output arc of place A, there is an output arc to place B that has the same target.
- For every output arc of place B, there is an output arc to place A that has the same target.

If all of these conditions are met, the rule can be applied and the following postcondition will hold:

- Place A and B are merged into a single place with the same input and output arcs as A and B had.

Correctness proof
As this reduction rule does not affect transitions in the model or events in the trace, the alignment cost is not changed in any possible scenario. Because of this, it is certain the the fitness calculated will be the same as for the original model and trace.

3.4 Fuse parallel transitions

This reduction rule merges transitions that are practically the same. This is shown in figures 3.8 and 3.9. Let A and B be 2 transitions in the model (or events in the trace), these transitions will be merged if they meet the following preconditions:

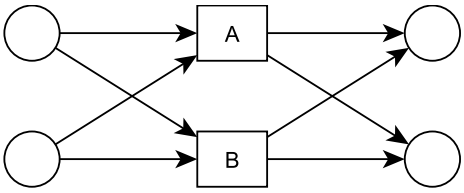

Figure 3.8: A model on which the fuse parallel transitions reduction can be applied.

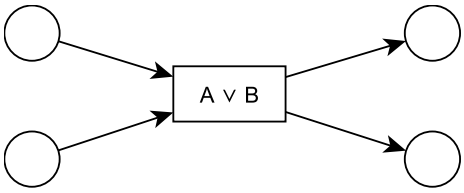

Figure 3.9: The model of figure 3.6 after the fuse parallel transitions reduction has been applied.

- For every input arc of transition A, there is an input arc to transition B that has the same source.
- For every input arc of transition B, there is an input arc to transition A that has the same source.
- For every output arc of transition A, there is an output arc to transition B that has the same target.
- For every output arc of transition B, there is an output arc to transition A that has the same target.
- The cost of log moves A and B are equal.
- Transitions A and B are unique, meaning no other transitions with the same label exist.

If these conditions are met, the 2 transitions can be merged into a single transition $A \vee B$ and the following postconditions will hold:

- The 2 transitions in the model are merged, the merged transition has the minimum value of the costs of A and B.
- All occurrences of A and B in the trace are replaced by a single $A \vee B$ event with cost value equal to either A or B, which does not matter as they are equal.

Correctness proof
As an optimal alignment will always favour the cheapest transition when 2 transitions have the same sources and targets, giving the merged transition the cheapest cost will not affect the cost of the alignment. Because the cost of the alignment will always be the same, the fitness is not affected.

3.5 Elimination of self-loop places

The elimination of self-loop places reduction rule removes places that do not limit any behaviour, such as the place shown in figure 3.10. Let A be a place in the model, then this place can be removed if it meets the following preconditions:

- Place A has exactly an equal amount of incoming arcs and outgoing arcs.
- For every incoming arc of A with source transition S it holds that there exists an outgoing arc that has S as its target.
- Place A contains a token in the initial state.

After this reduction is applied, the following postcondition will hold:

- The place and its connected arcs are removed from the model.

Observe that this does not affect the trace, hence the advantage of this rule on its own is rather limited, but it is possible that this enables other reduction rules.

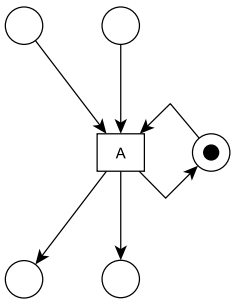

Figure 3.10: A self-loop place.

Correctness proof
As places that meet these preconditions do not limit the behaviour of the model, they can be omitted without having any influence on the reachability graph or the trace. Hence doing so will not affect any alignment cost calculation, meaning neither the fitness will be affected.

3.6 Elimination of self-loop transitions

The elimination of self-loop transitions reduction rule reduces the length of the trace if a self-loop occurs multiple times after each other, which is demonstrated in figure 3.11. Let T be a transition in the model corresponding to an event T in the trace, let M be the number of consecutive groups of event T and let N_x denote the length of the x'th group for $1 \leq x \leq M$. Then the trace can be reduced if the following preconditions are met:

- For every incoming arc with source place S it holds that there exists an outgoing arc that has S as its target and vice versa.
- Transition T is unique, so no other transition with the same label exists.
- There is at least one value N_x that is at least 2.

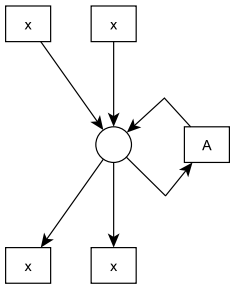

Figure 3.11: A self-loop transition.

After this reduction rule is applied, the following postconditions will hold:

- Every group of consecutive T events in the trace is replaced by a single T_N_x event.
- The cost of this single transition is equal to the cost of a single original event multiplied by N_x.

An example trace corresponding to model with a self-loop transition labelled A would be $\langle B, A, A, A, C, A, A, D\rangle$ which would be reduced to B, A_3, C, A_2, D in which A_3 has 3 times the cost of the original A and A_2 has 2 times this cost.

Correctness proof
Because all occurrences of the event T in a consecutive group occur straight after each other, it holds they all them are synchronous moves or all of them are log moves in the alignment. Observe that model moves will not occur with this kind of transition as it does not change the state of the model. Hence in this case the alignment cost is determined by the value of N_x multiplied by the cost of a log move or by the value of N_x multiplied by the cost of a synchronous move, which in practice is always equal to 0. Note that if the move has a cost of 0, the total cost will not be affected in this case. However, in the case of log moves, the original cost is the value of N_x multiplied by the initial cost of a log move. As the value of N_x will be decreased to 1 by this reduction, multiplying the cost of a log move with the original value of N_x will keep the total cost equal to that of the original model and trace. Hence applying this rule will not affect the fitness.

3.7 Preconditions motivation

3.7.1 Unique labels

Take the model shown in figure 3.12 and the trace $\langle A, B, C, D, C, E\rangle$, observe that the model contains two transitions labelled C. Note that this combination has a fitness of 1 if the first C corresponds to the transition in between transitions A and D in the model. However, if the fuse series transitions is applied on the transitions B and C, the trace no longer fits because the first C will now correspond to the merged transition of B and C and the D and C events after this merged transition in the trace, will no longer fit the model. This demonstrates the need for the precondition of unique labels.

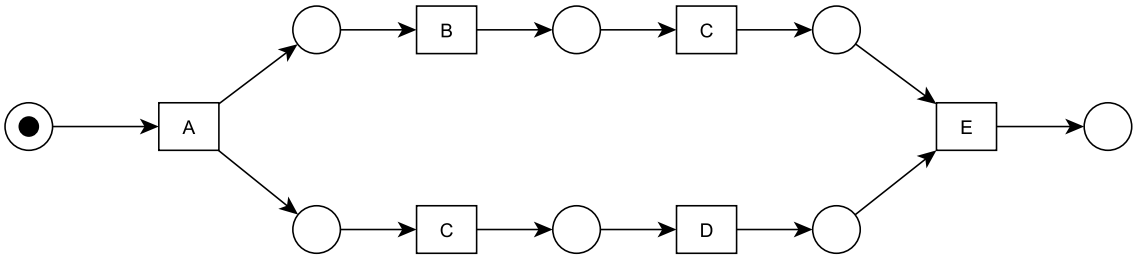

Figure 3.12: A model to show the need for unique labels.

3.7.2 For every

Observe the model shown in figure 3.13 and take the traces $\langle A, B, C, B, C, D \rangle$, $\langle A, B, C, C, B, D \rangle$ and $\langle A, B, C, B, C, B, D \rangle$, observe that the first trace has a fitness of 1, while the second trace has a B and a C swapped and the third trace contains an additional B. Because of this, only the first trace matches the preconditions to apply the fuse series transition rule to merge transitions B and C into a single transition. If one would apply this rule to the second and third trace as well, the trace would contain loose B and C events, which no longer corresponds to any transition in the model. For the third trace this would imply the extra B is non-fitting, while it can also indicate a missing C. This illustrates that when merging 2 transitions, all occurrences in the trace of these 2 transitions must fit perfectly for a reduction to be applied.

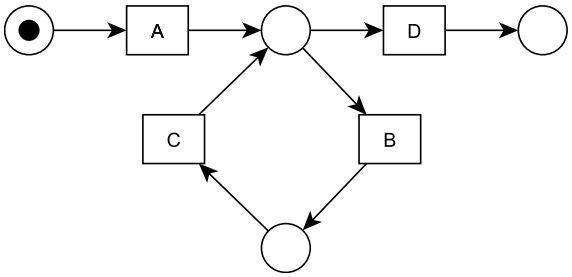

Figure 3.13: A model that shows the need of the "for all" precondition.

3.7.3 Consecutive events in trace

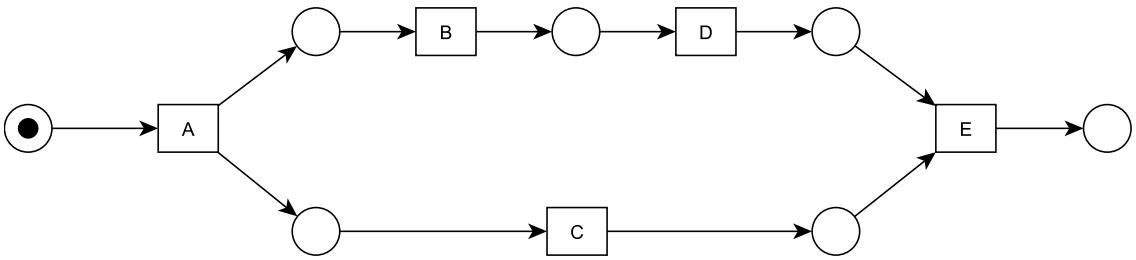

Figure 3.14: A model that shows the need of events in a trace being consecutive without other events in between them.

Observe the model in figure 3.14 and the possible merge of transitions B and D. However, for the trace $\langle A, B, C, D, E \rangle$, it holds that C event occurs in between the B and D event. Because of this, the 2 events in the trace cannot be merged as this would lead to a C event within the $B \wedge D$ event. Another issue with such a merger would be to identify the position of the arcs of the synchronous transition in the synchronous product, this combination has a fitness of 1, but a synchronous transition in between the model and the trace cannot be matched on the trace side as it would skip the C event.

3.7.4 Equal costs

See the model displayed in figure 3.15, let log move A have a cost of 2 and log moves B and C a cost of 1. Now take the traces $\langle A \rangle$ and $\langle B \rangle$ that both consist of a single event, since transition A has a higher cost than B, the fitness of the trace containing just A is higher than the fitness of just B as the cost of the missing C has less impact on the total cost.

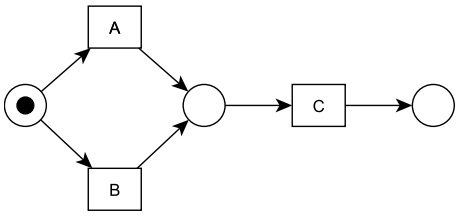

Figure 3.15: A model that shows the need of equal costs of transitions to be merged.

Merging transitions A and B into a single transition would imply that the 2 corresponding events in the trace have the same cost. If this was the case, the 2 traces would result in the same fitness. This is however is not the case, therefore, these 2 transitions cannot be merged.

3.8 Overview

An overview the 6 different rules presented in this chapter can be found in table 3.1. Observe that each rule on its own has a rather limited impact on the net and log, but that these rules can be combined and used in an additive manner. Because of this, a combination of multiple rules can be used to significantly decrease the size of the net and the corresponding trace.

Rule name	**Petri net reductions**	**Log reductions**
Fuse series places	Size reduced by 1 place and 1 transition	All occurrences of the reduced events are merged into other events
Fuse series transitions	Size reduced by 1 place and 1 transition	All occurrences of the corresponding 2 events are merged into 1 event
Fuse parallel places	Size reduced by 1 place	None
Fuse parallel transitions	Size reduced by 1 transition	All occurrences of the corresponding 2 events are replaced by the new event, the amount of events stays equal
Elimination of self-loop places	Size reduced by 1 place	None
Elimination of self-loop transitions	Self-loop is splitted up into several transitions	All occurrences of the self-loop are reduced to a single event

Table 3.1: An overview of the reduction rules in the first phase

Chapter 4

Phase 2 Reduction Rules

These phase 2 reduction rules are applied after all the possible reductions of the previous section have been applied. The phase 2 rules differ from the phase 1 rules due to the fact that they do not reduce the trace and model, instead they look at the synchronous product as a whole and make reductions on the model-side of this. As a consequence, synchronous transitions can be added. Observe that all rules mentioned in this section have already been discussed in the previous section, this section however uses a different approach which makes them more flexible. To prevent an exponential growth of the number of synchronous transitions, the rules in sections Fuse series places and Fuse series transitions can only be applied at most once to the same transition. It should be noted that the reduction rules in this chapter have less strict preconditions and can hence be applied more often, this is demonstrated by an example in section Preconditions motivation.

For all figures in this chapter: blue indicates the model moves, black indicates the synchronous moves and green indicates the log moves. Observe that these figures show a synchronous product to determine the fitness.

4.1 Fuse series places

Let place A and place B be the places to be merged, let T be the transition in between them and let F be the set of transitions that T will be fused with. Note that T consists of all input transitions of place A or all output transitions of place B. The fuse series places is applied if the following preconditions are met:

- The model-side of the synchronous product contains more than 1 transition in total, so set T is not empty.
- All the transitions in F and transition T have not been reduced by phase 2 fuse series places or phase 2 fuse series transitions.

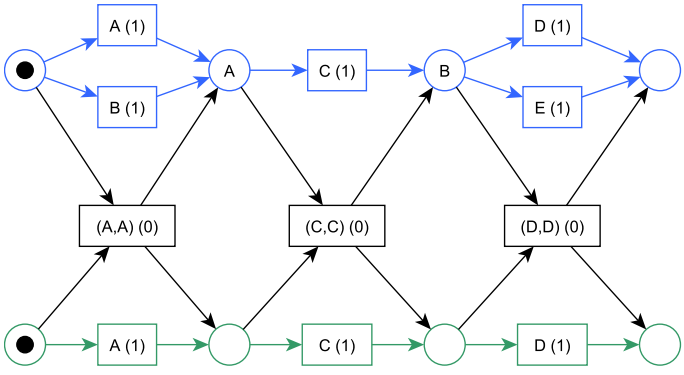

Figure 4.1: A synchronous product on which the fuse series places reduction can be applied.

If this rule is applied, the following postconditions will hold afterwards:

- The transition T is merged into the transitions in F and the model move costs are added up.
- A synchronous place is added to represent the state where 1 of the 2 (T and the transition in F) synchronous transitions has been taken, but the other one has not. This replaces the model side place that has been removed for all corresponding synchronous transitions
- Synchronous transitions are added to give a path if 1 of the 2 (T and the transition in F) events is missing. This means that for every synchronous transition corresponding to a transition in F, a synchronous transition is added for that move and a model move T, but for every synchronous transition corresponding to T a synchronous transition for every possible combination with F. The newly added synchronous transitions have the cost equal to that of the missing model move.
- A synchronous place is added to represent the state where one of the 2 (T and one transition in F) synchronous steps have been taken, but the other has not.

Observe that the preconditions of a perfect matching log and model combination, the unique labels and the consecutive events in the trace have been dropped. This implies that this version of the fuse series transitions rule can be applied more often. Also observe that this rule only makes changes to the synchronous part and the model part of the synchronous product. The trace remains in its original state, hence it is not required to make any preconditions on the trace. This is however likely to imply a smaller optimization in required computational power when compared to the phase 1 version of this rule.

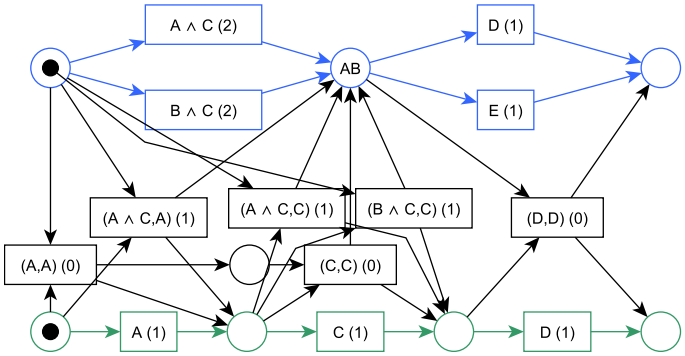

Figure 4.2: The synchronous product of figure 4.1 after applying the fuse series places reduction with the transition in between place A and B merged with the source places of A.

Correctness proof

In this reduction, 3 differences are made to the synchronous product: model moves are merged, a synchronous place is added and synchronous transitions are added. For the model moves that are merged, it holds that their costs are added, for the place that is added, it holds that it simply replaces the old place in the model side and do not affect the synchronous cost and for the synchronous transitions that are added, it holds that they have the cost of the log move that is missing, hence they do not affect the alignment cost either. Since it is now proven that for every change that is made, the cost will not be affected, it can be concluded that the fitness will also stay the same after this reduction.

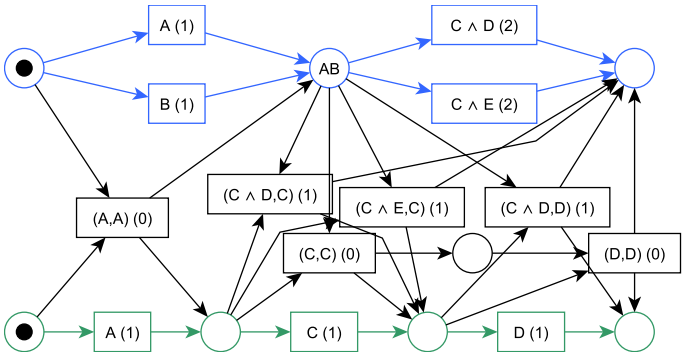

Figure 4.3: The synchronous product of figure 4.1 after the fuse series places reduction has been applied with the transition in between place A and B merged with the target places of B.

4.2 Fuse series transitions

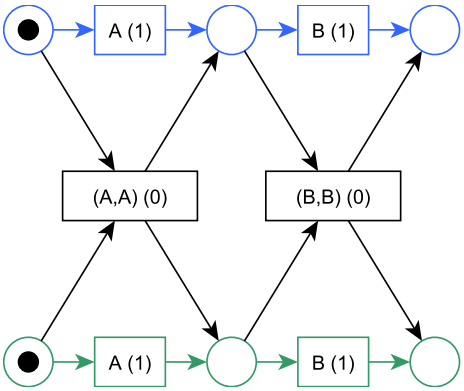

Figure 4.4: A synchronous product on which the fuse series transitions reduction can be applied.

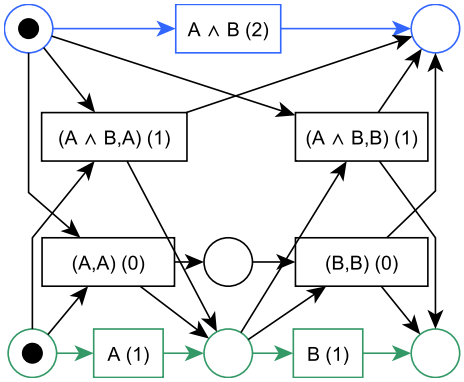

Figure 4.5: The synchronous product of figure 4.4 on which the fuse series transitions reduction can be applied.

A graphical representation of a fuse series transition reduction is shown in figures 4.4 and 4.5. Let A and B be transitions in the model side of the synchronous product and let P be a place in between them. This reduction rule can be applied if the following preconditions are met:

- P has exactly one model move only input arc with A as its source and exactly one model move only output arc with B as its target, arcs from or to synchronous transitions are neglected.
- Transition A has exactly one output arc with P as its target and transition B has exactly one input arc with P as its target.
- All involved transitions have not been reduced by phase 2 fuse series places or phase 2 fuse series transitions.

If all of these preconditions are met, the rule can be applied and the following postconditions will hold:

- Transitions A and B on the model side are merged into a single transition with the cost equal to the sum of the cost of A and B.
- A synchronous place is added to represent the state where a synchronous A transition has fired but a synchronous B has not.
- Synchronous transitions are added where the model takes both A and B steps, but the log only takes 1 of the 2 corresponding events, these synchronous transitions have a cost equal to that of the missing step.

Correctness proof
Since the 2 merged transitions on the model side of the synchronous product have their costs added up, a path on the model part will always result in the same cost. Since the synchronous transitions are practically the same, they will change any costs either. The trace part of the synchronous product remains the same, hence any alignment using this part will always yield the same costs. For the cases in which only 1 of the 2 moves is synchronous, a separate synchronous transition is added which has the cost of the missing move, hence this will always yield the same cost. As this covers all possible paths along the synchronous product, it has been proven that this reduction will not affect the fitness.

4.3 Fuse parallel transitions

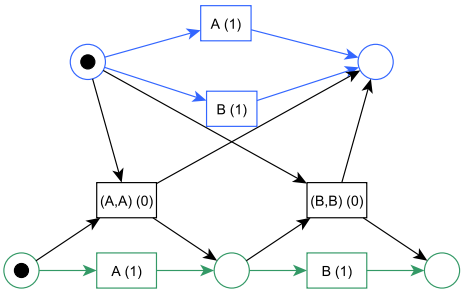

Figure 4.6: A synchronous product on which the fuse parallel transitions reduction can be applied.

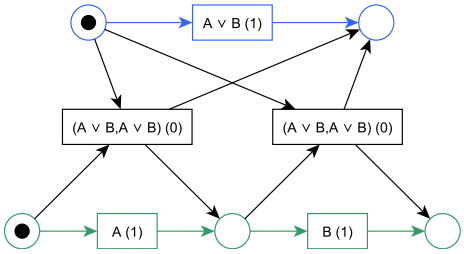

Figure 4.7: The synchronous product of figure 4.6 after the fuse parallel transitions reduction has been applied.

The fuse parallel transitions reduction rule merges 2 transitions that have the same inputs and outputs and changes the labels of corresponding synchronous transitions. This rule can be applied if 2 transitions A and B meet the following preconditions:

- For every input arc of transition A, there is an input arc to transition B that has the same source.
- For every input arc of transition B, there is an input arc to transition A that has the same source.
- For every output arc of transition A, there is an output arc to transition B that has the same target.
- For every output arc of transition B, there is an output arc to transition A that has the same target.

After this rule has been applied, the following postconditions will hold:

- In the model side of the synchronous product, transitions A and B are merged into a single transition with a cost equal to the minimum cost in between A and B.
- All synchronous transitions with labels A or labels B are replaced by a synchronous transitions with label $A \vee B$, indicating A or B, all arcs and costs stay the same.

Observe that because this rule only changes labels of synchronous transitions and does not add any, this rule can be applied as many times as possible without any problems. Hence the preconditions of this rule are a subset of the phase 1 version of this rule, without any additions.

Correctness proof

Observe that this reduction rule only merges transitions in the model part of the synchronous product. The costs of these 2 transitions can differ, but since the algorithm solving this problem will always look for the path with the lowest cost, it will always take the transition with the lowest cost if it has to choose between 2 transitions having the same input and output arcs. Hence giving a merged transition the lowest cost will not affect the cost of any alignment, which proves that applying this reduction rule will not affect fitness.

4.4 Preconditions motivation

Observe that only a single precondition has been added to the preconditions already motivated in chapter 3, hence only this precondition is motivated here. The precondition holds for the rules fuse series places and fuse series transitions and states that these rules cannot be applied if 1 of these 2 rules has already been applied to one of the involved transitions. Observe that this is an optimization and not necessarily a strict precondition, a theoretical application of fuse series places reduction on 3 transitions in shown in figures 4.8 and 4.9.

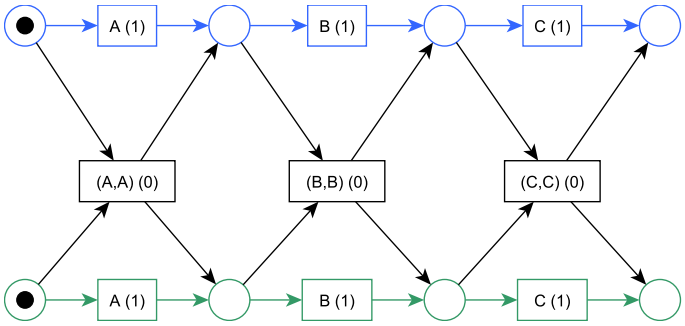

Figure 4.8: A synchronous product in which 3 transitions can be merged.

The reasoning behind not allowing these reductions is that these reductions add synchronous transitions and places. This addition however, is exponential in terms of the merged transitions because all the possible combinations of the log moves have to be modelled as synchronous transitions. As a consequence, applying reduction after reduction will create a massive synchronous product, which can be more challenging to solve than the original. A first example of this explosion can be seen in figure 4.9, it should be noted that this is only a merger of 3 transitions and that this problem grows exponentially if more transitions would be reduced.

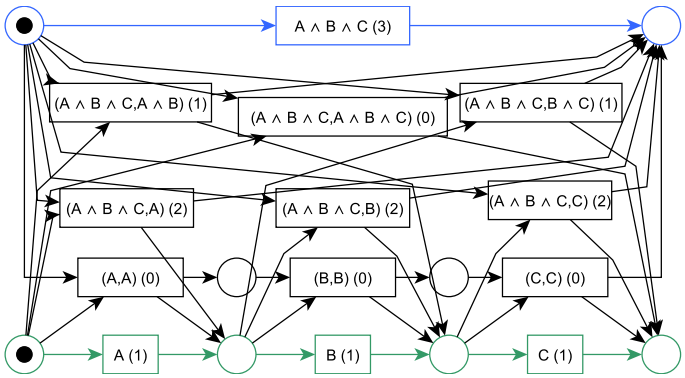

Figure 4.9: The synchronous product of figure 4.8 after 3 transitions have been reduced using the phase 2 fuse series transitions reduction.

4.5 Overview

The table 4.1 contains an overview of the removed preconditions, hence this indicated situations in which the phase 1 rules cannot be applied but the phase 2 variant can.

Rule name	**Removed preconditions**
Fuse series places	All occurrences of the merged events have to occur consecutively. All labels have to be unique.
Fuse seris transitions	All occurrences of the merged events had to occur consecutively. All labels have to be unique.
Fuse parallel transitions	The costs of the merged transitions and events have to be equal. All labels have to be unique.

Table 4.1: An overview of the reduction rules in the second phase.

Chapter 5

Reduction DAG

To apply the reduction rules, a directed acyclic graph, or DAG, is created. In this chapter the motivation behind it is explained in the Motivation behind the reduction DAG section. The creation and usage of the reduction DAG is explained in sections Creating the reduction DAG and Using the reduction DAG. To illustrate this process, an example is provided in the section Reduction DAG example.

5.1 Motivation behind the reduction DAG

It frequently occurs that if a reduction rule is applied, other rules are enabled or rules that were enabled before the reduction are still enabled. This means reductions follow up on each other. Hence it makes sense to store such rules in a DAG in which each arc indicates a reduction and each place indicates a model.

Another argument for the use of this data structure is efficiency. The whole goal of this research is to speed up the process of conformance checking, hence the reduction rules need to be applied in an efficient manner. As the reductions have preconditions on the model and, in most cases, the corresponding trace, and the model is the same in all cases, it is most efficient to use the model as a basis for finding the rules. This way, a set of rules is created of which it is known for which model they can be applied to without repetitiously verifying the preconditions for each of them. Summarizing: it is faster to create all the possible rules once rather then a subset of them for every trace.

Another efficiency argument is the terms of the size, for any input of this plug-in it holds that there is only one petri net model, but as the fitness needs to be computed for every trace. The amount of traces in an event log can be extremely high, hence the goal is to optimize for a minimal computational time per trace, even if this implies more required time for the model. Using this implementation, only a very limited number of rules have to verified for each trace, rather than repetitively looping through all the possible reductions.

5.2 Creating the reduction DAG

The DAG consists of places and directed arcs, each arc represents a reduction and each place represents a model, note that the place without input arcs represents the original net and that every place without output arcs represent models that can no longer be reduced further.

The first step of creating the DAG is finding the reduction rules. This is done using a loop that verifies for each rule and each combination of places or combination of transitions (depending on the actual rule) if the preconditions are met. If these conditions are met, a rule can be

applied. When an applicable rule is found in this loop, it is determined straight away if the rule is applicable for phase 1, phase 2 or both, note that both is also relevant, it can occur that a rule can be applied to one trace in phase 1, but only is applicable in phase 2 for another trace. Every time a rule is found, it is applied and if the resulting model is not yet in the DAG, it is added. The same goes for rules found: if the rule is already present in the DAG, the newly added arc will receive the same label as where the rule has been previously used.

As the Fuse parallel places, Fuse parallel transitions and Elimination of self-loop places reduction rules do not pose any preconditions with respect to the log, it is certain that they can always be applied. This is used as an optimization to reduce the size of the DAG. Because they can always be applied, there is no need for other outgoing arcs in a place in the DAG if one of such reductions is possible, hence the loop looking of reduction rules analyses the model for those rules first and stops searching further if one of these rules has been found. As a consequence of this, it is possible the the first reductions applied will be exactly the same for every trace.

5.3 Using the reduction DAG

During the creation of the DAG, an empty log is constructed for every model in it. Every trace traverses down the DAG starting at the place corresponding with the original model, during this phase the algorithm only looks at rules that are applicable for phase 1 and both phase, not rules that are only applicable for phase 2. At every place the trace arrives in, it goes through all the reduction rules represented by the outgoing arcs until it finds a rule for which it meets the phase 1 preconditions, if this is the case the trace is reduced accordingly and it arrives at the target place of the respective arc and starts looking for a further reduction. If the trace no longer meets the precondition of any of the outgoing arcs, the trace is added to the log corresponding to the model represented by the current place.

Once all traces have been reduced as much as possible, the plug-in continues to create the synchronous products. For every synchronous product, the plug-in looks if it can be further reduced using the same method used for phase 1, but this time it only looks at rules applicable for phase 2 or both the phase, so it no longer looks are rules exclusively for phase 1. This way the plug-in utilizes the efficient DAG data structure as much as possible.

5.4 Reduction DAG example

Take as input the petri net model displayed in figure 5.1 and the following log:
0: $\langle a, b, c\rangle$
1: $\langle a, d, b, e, c\rangle$
2: $\langle a, d, b, f, d, c\rangle$
3: $\langle a, b, e, c\rangle$
4: $\langle a, b, b, c\rangle$
5: $\langle d, a, b, f, c\rangle$
6: $\langle a, c\rangle$
7: $\langle a, b, a, b, c\rangle$
8: $\langle d, e, a, b, d, e, c\rangle$
9: $\langle d, e, d, e, d, f, d, e, c\rangle$
10: $\langle a, b, d, e, d, f, c\rangle$

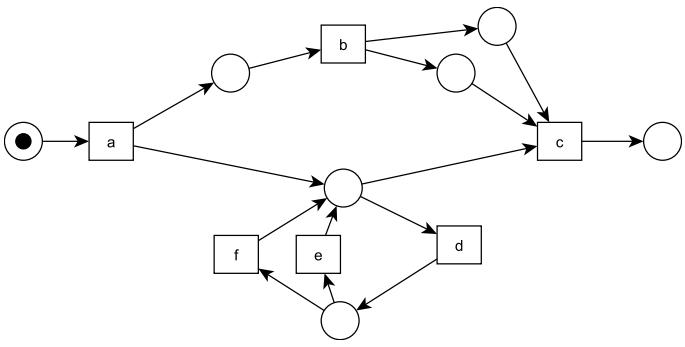

Figure 5.1: The original petri net used in this example before any reduction has been applied.

The first step is creating the reduction DAG, this is done purely based on the model, which implies that it would be the same for

Rule id	Reduction rule
R0	Fuse parallel places between b and c
R1	Fuse parallel transitions e and f into $e \vee f$
R2	Fuse series places: merge b into c and create $b \wedge c$
R3	Fuse series places: merge b into a and create $a \wedge b$
R4	Fuse series transitions d and $e \vee f$ into $d \wedge (e \vee f)$
R5	Elimination of self-loop transitions $d \wedge (e \vee f)$

Table 5.1: An overview of the reduction rules in the reduction DAG of figure 5.2.

every possible log, the reduction DAG is shown in figure 5.2. Observe that every place is labelled M followed by an integer, indicating it refers to a model and every arc is labelled R followed by an integer, indicating it refers to a reduction rule. All the reduction rules are explained in table 5.1

For the models it holds that $M0$ is the original model as displayed in 5.1 and $M9$ and $M10$ are the fully reduced versions. The model $M10$ is shown in figure 5.3, the model $M9$ will look very similar, except with transitions labelled a and $b \wedge c$ instead of $a \wedge b$ and c, observe that for these versions no further reduction rules can be applied. For the DAG it holds that every model displayed occurs only once, this is due to the fact that if a model is reduced and the reduced model is already in the DAG, a new arc to the existing model is added and the model is not added again.

All the rules in the DAG are further explained in the table 5.1. Note that the several arcs can have the same label, this is due to rules being applicable to multiple models. For the rules $R0$ and $R1$ it holds that they are applied right at the start of the DAG and are the only outgoing arcs of their respective input places, this is due to the fact that they have no preconditions for the log, indicating they can be applied to any trace in it. Furthermore it can be observed that $R5$ is enabled after $R4$ is applied, which can be explained by the fact that $R4$ creates a self-loop.

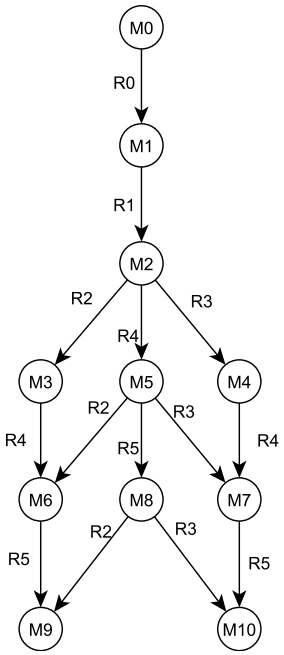

Figure 5.2: The reduction DAG constructed as a consequence of reducing the model displayed in figure 5.1.

The next step is to apply the phase 1 reduction to the traces in the logs and map each reduced trace with the corresponding reduced model. This mapping is shown in table 5.2, observe the models with no

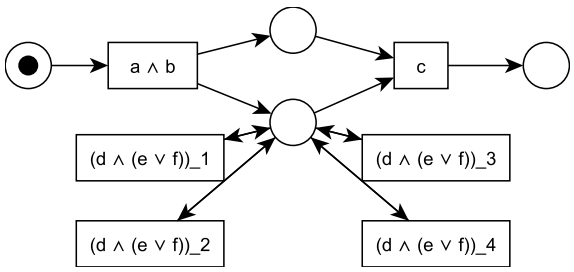

Figure 5.3: A fully reduced version of the model shown in figure 5.1.

Model id	Assigned traces	Applied reduction rules (to model and traces)
$M2$	1, 2	$R0, R1$
$M4$	3, 5	$R0, R1, R3$
$M5$	4, 6	$R0, R1, R4$
$M7$	0, 7, 8	$R0, R1, R3, R4$
$M10$	9, 10	$R0, R1, R3, R4, R5$

Table 5.2: An overview of the several reduced models and their corresponding reduced traces.

corresponding traces have been omitted in this overview. The rules $R0$ and $R1$ have no preconditions on the traces and are thus applied to all traces in the log.

With respect to the traces, it holds that the precondition of $R2$ is that all b and c events must occur consecutively, for $R3$ there is the same precondition for events a and b and the same goes for $R4$ and events d and $e \vee f$. The traces 1 and 2 do not meet these preconditions and are hence mapped to the reduced model $M2$, all other traces match at least one of these preconditions. But for the traces 3 and 5 it holds that they do meet the preconditions of $R3$, but not those of $R4$, hence these traces are matched with $M4$. For traces 4 and 6 the opposite is true: they meet the preconditions of $R4$, but not those of $R2$ and $R3$, hence these 2 traces are matched with $M5$.

After these steps we have the traces 0, 7, 8, 9 and 10 left on which all rules except $R4$ have been applied. In all the models where $R4$ has been applied, the transitions d and $e \vee f$ have been merged into a single self-loop transition $d \wedge (e \vee f)$. But rule $R5$ has as a precondition for the corresponding trace that at least 2 consecutive events in the trace have the label $d \wedge (e \vee f)$. However, as this event doesn't even occur in traces 0 and 7 this rule is not applied to these traces. This event does occur twice in trace 8, but there are other events in between, indicating applying the rule here wouldn't make a difference, hence it is not applied to trace 8 either, so traces 0, 7 and 8 are assigned to $M7$. For the traces 9 and 10 it holds that these events do occur multiple times in a consecutive manner, in fact, this event occurs 4 times in a row in trace 9 and 2 consecutive times in trace 10, making this traces eligible for the the final reduction rule $R5$, matching it with the model $M10$ which is also shown in figure 5.3. If the alignments would be computed after these reductions, the result would that as displayed in table 5.3, note that the reduced models and reduced traces, which are the same as the alignments when leaving out the model moves, are the starting point for the synchronous products used in the phase 2 reductions.

Id	Alignment	Cost	Fitness
0	$a \wedge b, c$	0	1
1	$a, d, b, e \vee f, d, c$	1	0,89
2	$a, d, b, e \vee f, c$	0	1
3	$a \wedge b, e \vee f, c$	1	0,86
4	a, b, b, c	1	0,86
5	$d, a \wedge b, e \vee f, c$	2	0,75
6	a, b, c	1	0,8
7	$a \wedge b, a \wedge b, c$	2	0,75
8	$d \wedge (e \vee f), a, \wedge b, d \wedge (e \vee f), c$	2	0,8
9	$a \wedge b, (d \wedge (e \vee f))\_4, c$	2	0,83
10	$a \wedge b, (d \wedge (e \vee f))\_2, c$	0	1

Table 5.3: The results of fitness calculation for every trace after phase 1 reduction, model moves are shown in blue, log moves in green, synchronous moves remain black.

After these reductions, it is time for the phase 2 reductions, which can also be applied. Note that only the rules $R2$, $R3$ and $R4$ are eligible for the phase 2 reduction, rule $R1$ would have been a phase 2 only reduction rule if the costs of event moves e and f would differentiate, but this is not the case. Because of this the only models worth looking at for phase 2 reductions are $M2$, $M4$ and $M5$, note that if $M3$ or $M8$ had traces assigned to them, they would be in this list as well.

For the synchronous product corresponding to $M2$, the reductions of $R2$, $R3$ and $R4$ can be applied, but note that applying $R2$ disables applying $R3$ because transition b no longer exists. Hence only $R2$ and $R4$ are applied. When rule $R2$ is applied, the model moves b and c of the synchronous transitions are merged into a single $b \wedge c$ transition at the place of c. To still allow for all possible paths, synchronous transitions $(a \wedge b, a)$ and $(a \wedge b, b)$ are added, both having the cost equal to the missing model move. Furthermore, the synchronous transitions (a, a) and (b, b) have arcs redirect to a new synchronous place which illustrates the situation where a synchronous a move has been done but a synchronous b move has not. When rule $R4$ is applied, the model move transitions d and $e \vee f$ are merged into a single $d \wedge (e \vee f)$. Synchronous transitions are added and redirect in the same manner as done when merging a and b. Note that in this case the merged transition is a self-loop, but as only phase 2 reductions can be applied, this will not be reduced any further.

For the synchronous product corresponding to $M4$ it holds that the reduction of $R4$ is applied. For the synchronous product corresponding to $M5$ the reductions of $R2$ and $R3$ can be applied, but again $R3$ can no longer be applied after $R2$ has been applied, so in practice only $R2$ is applied.

Id	Alignment	Cost	Fitness
0	$a \wedge b, c$	0	1
1	$a, d, b, e \vee f, d, c$	1	0,89
2	$a, d, b, e \vee f, c$	0	1
3	$a \wedge b, e \vee f, c$	1	0,86
4	$a, b, b \wedge c$	1	0,86
5	$d, a \wedge b, e \vee f, c$	2	0,75
6	$a, b \wedge c$	1	0,8
7	$a \wedge b, a \wedge b, c$	2	0,75
8	$d \wedge (e \vee f), a, \wedge b, d \wedge (e \vee f), c$	2	0,8
9	$a \wedge b, (d \wedge (e \vee f))\_4, c$	2	0,83
10	$a \wedge b, (d \wedge (e \vee f))\_2, c$	0	1

Table 5.4: The final results of fitness calculation for every trace, model moves are shown in blue, log moves in green, synchronous moves with no cost remain black, but synchronous moves with a cost are shown in red.

Although these reductions change the synchronous products for all 6 of the involved traces, it does not always affect the final alignment. This can be explained by the fact that adding and altering synchronous transitions, an optimal solution of the original synchronous product can still be the easiest one to find, perhaps even the only optimal solution. The final alignments of this conformance checking problem are shown in table table 5.4. The traces 4 and 6 are the only traces of which the corresponding alignment is changed. In trace 4 the original synchronous transitions (b, b) and (c, c) has been replaced by a single $(b \wedge c, b \wedge c)$ transition with a cost of 0. In alignment 6 the model move b and the synchronous move c are replaced by a synchronous $b \wedge c$ corresponding to the synchronous transition $(b \wedge c, c)$, which has the cost equal to that of the model move b, which is equal to 1.

Chapter 6

Performance Evaluation

In this chapter the actual implementation of this approach is tested and compared to the original version of the plug-in. In the section Evaluation setup the models and logs that have been used for the evaluation are explained and motivated. In the following section Results and Comparison, the results of the several test runs are displayed and compared to each other.

6.1 Evaluation setup

For the evaluation of the plug-in, 3 artificially created nets are used, these are displayed in figures 6.1, 6.2 and 6.3. It must be noted that the first net is the same net that was used in the section Reduction DAG example. The main motivation behind the first net is that 5 of the 6 reduction rules need to be applied to make a fully reduced model. The only rule not used is the elimination of self-loop places, but the situation reduced by this rule cannot occur in workflow nets. The net in figure 6.2 is used because it shows a lot of and/or splits and merges, which also occurs in real-life business processes. Another interesting characteristic is that it can be reduced to just a net with just a single transition. The net shown in figure 6.3 is used to try reductions on a more complex net.

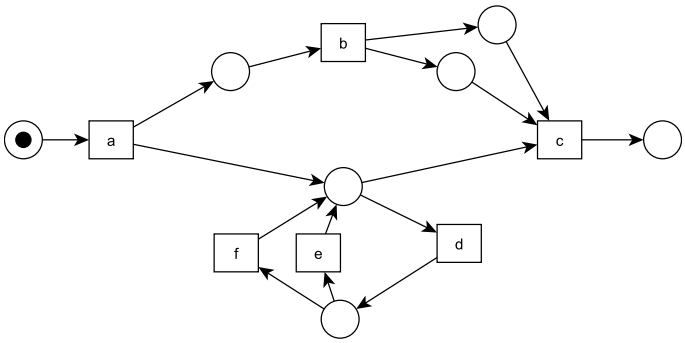

Figure 6.1: The first net used for the evaluation.

For each of these nets, 3 logs are generated using the plug-in named "Perform a simple simulation of a (stochastic) Petri net" by A. Rogge-Solti. The first log contains 100 traces, the second log 1.000 traces and the third log 5.000 traces, the motivation behind these different numbers is to be able to observe the difference in their performance as the size of the input problem grows.

For each log, 2 other variants are created by using the "Add Noise to Log Filter" plug-in by R. S. Mans. This plug-in contains 2 parameters: noise level for removing events and noise level for adding events, both expressed in percentages. The first variant is created by setting both parameters to 10 and the second variant is created by setting both parameters to 25. This is done to compare performances for log that match perfectly, that differ slightly and that differ significantly. Observe that there are now 9 different logs for each artificial net.

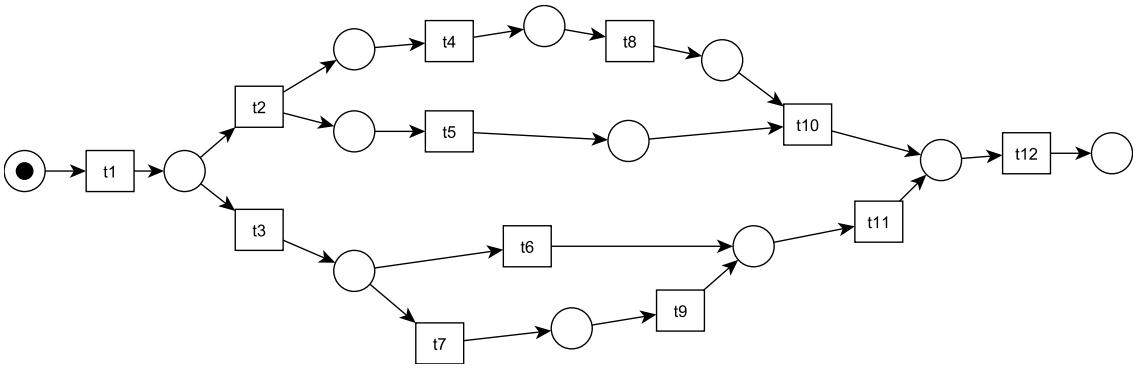

Figure 6.2: The second net used for the evaluation.

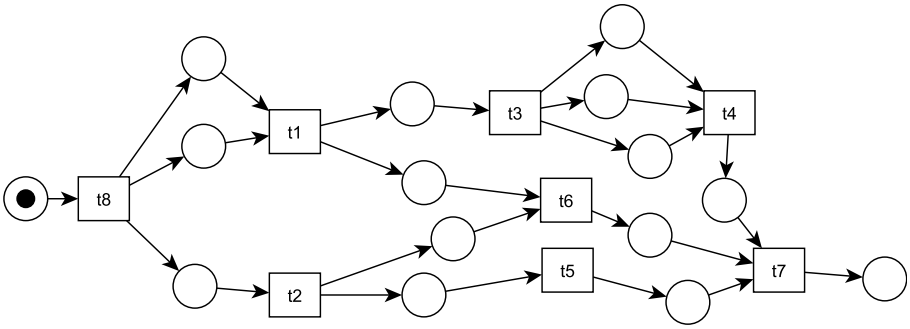

Figure 6.3: The third net used for the evaluation.

For each combination of log and net, 3 versions of the plug-in are executed and timed, namely the original plug-in, the plug-in with all reduction rules and the plug-in with phase 1 reductions only. First, the original plug-in is timed from start to end, secondly the plug-in with the reduction rules, named "reduced plug-in" is timed, thirdly a version of the reduced plug-in with only phase 1 reductions is timed. For the sake of insight, the versions with implemented reductions also have a measurement point after the reduction DAG has been created and after the phase 1 reductions have been applied. This allows for a comparison of the advantage of each phase separately and also see the influence of the new implementation of the synchronous product required for the phase 2 reductions.

6.2 Results and Comparison

The results of the evaluation are shown in table 6.2. Note that all the times measured from the point of starting the computation. So for example, the time shown for phase 1 reductions is the time needed to create the reduction DAG and apply the phase 1 reduction. As this table is rather large and overwhelming, a short summary of these results is shown in table 6.1.

Property	Original	Reduced	Phase 1 only
Model 1	6,49	4,93	5,75
Model 2	5,46	10,26	13,16
Model 3	8,55	7,30	6,94
100 traces	1,58	3,59	4,29
1.000 traces	4,73	6,74	7,48
5.000 traces	14,18	12,15	14,23
0% noise	4,87	2,67	3,12
10% noise	6,31	7,60	8,16
25% noise	13,64	12,22	14,71

Table 6.1: A summary of the results shown in table 6.2 with the most relevant averages.

The first general observation is that applying reductions will not always result in a faster computation of the alignment. Es-

pecially in the cases with only a 100 traces, the reduced versions need at least twice as much time. This can be explained by the fact that creating the reduction DAG can take up a significant amount which is not affected by the number of traces.

The second general observation is that the application of both phase 1 and phase 2 is faster in almost all the cases than just the reductions in phase 1. This proves that the phase 2 reductions are beneficial, something which could be argued not to be the case due to the extra time required per synchronous product.

When looking at the several levels of noise it can be concluded that the reductions allow for a faster computation of the fitness if no noise is added. If noise is added however, no consistent improvement can be observed. Because of this, it seems that adding noise worsens the performance of the plug-in with reduction rules more than the performance of the plug-in without reduction rules. A possible explanation for this is that less phase 1 reductions can be applied, hence more phase 2 reductions are applied which are less efficient.

Another observation is the differences in time required for creating the reduction DAG, for the models 1 and 3 this is usually a matter of hundredths of seconds while for the model 2 this is in the range of 5 to 10 seconds. A deeper inspection of the DAGs themselves showed that model 1 yields a DAG with 11 places and 14 edges, model 2 yields a DAG with 2.073 places and 3603 edges and model 3 yields a DAG with 46 places and 75 edges. Since the DAG of model 2 is significantly larger than those of the other models, this explains the difference in time needed. The main reason for the difference in this size is that model 2 has a lot of fuse series places reductions, of which there are 2 variants, that follow up on each other and thus create a lot of different combinations.

To obtain an impression of what the running times would be if the DAG is already constructed, it is possible to subtract the time needed for creating the DAG from the total running time. If this is done the average time for the reduced plug-in decreases from 7,49 to 5,36 seconds and for the phase 1 only plug-in this time is reduced from 8,67 to 6,03 seconds, note that the original plug-in has an average running time of 6,83 seconds. This shows that although the construction of the DAG can be time consuming, the application of it rather efficient.

The sixth and final conclusion is that the phase 1 reduction is always applied very efficiently, for all the test runs it holds that time to apply it is rather minimal. This again shows that the reduction DAG is indeed a very efficient data structure to apply reductions, which was the main goal of it.

Input	Original plug-in	Reduced plug-in DAG creation	Reduced plug-in apply phase 1	Reduced plug-in finished	Phase 1 only DAG creation	Phase 1 only apply phase 1	Phase 1 only finished
Model 1, 100 traces, 0% noise	0,22	0,01	0,01	0,18	0,02	0,03	0,23
Model 1, 100 traces, 10% noise	4,01	0,01	0,02	3,42	0,02	0,05	4,43
Model 1, 100 traces, 25% noise	7,15	0,01	0,02	6,38	0,02	0,04	8,17
Model 1, 1.000 traces, 0% noise	1,23	0,02	0,06	0,62	0,08	0,15	0,79
Model 1, 1.000 traces, 10% noise	8,38	0,01	0,03	7,43	0,02	0,05	8,17
Model 1, 1.000 traces, 25% noise	9,30	0,02	0,03	8,31	0,02	0,04	10,51
Model 1, 5.000 traces, 0% noise	2,29	0,01	0,12	0,66	0,02	0,16	0,81
Model 1, 5.000 traces, 10% noise	12,08	0,01	0,02	8,11	0,02	0,04	8,01
Model 1, 5.000 traces, 25% noise	13,73	0,02	0,03	9,24	0,01	0,02	11,28
Model 2, 100 traces, 0% noise	0,06	6,25	6,26	6,30	7,24	7,24	7,30
Model 2, 100 traces, 10% noise	0,38	7,22	7,22	7,67	7,35	7,36	7,87
Model 2, 100 traces, 25% noise	0,69	5,90	5,91	6,53	7,81	7,81	8,71
Model 2, 1.000 traces, 0% noise	0,07	7,79	7,87	8,06	7,41	7,45	7,53
Model 2, 1.000 traces, 10% noise	2,11	5,70	5,76	8,36	8,51	8,57	11,89
Model 2, 1.000 traces, 25% noise	6,27	6,87	6,90	12,63	7,75	7,79	16,90
Model 2, 5.000 traces, 0% noise	0,12	6,16	6,32	6,51	9,86	10,02	10,26
Model 2, 5.000 traces, 10% noise	8,02	5,59	5,73	11,57	7,09	7,31	14,54
Model 2, 5.000 traces, 25% noise	31,43	5,42	5,56	24,71	7,66	7,86	34,82
Model 3, 100 traces, 0% noise	0,22	0,05	0,06	0,26	0,04	0,04	0,30
Model 3, 100 traces, 10% noise	0,55	0,05	0,06	0,64	0,04	0,04	0,58
Model 3, 100 traces, 25% noise	0,94	0,05	0,05	0,92	0,04	0,04	0,99
Model 3, 1.000 traces, 0% noise	0,32	0,13	0,19	0,85	0,05	0,07	0,37
Model 3, 1.000 traces, 10% noise	4,83	0,09	0,17	5,28	0,05	0,08	4,18
Model 3, 1.000 traces, 25% noise	10,10	0,07	0,11	9,13	0,05	0,08	7,67
Model 3, 5.000 traces, 0% noise	0,34	0,05	0,19	0,61	0,03	0,16	0,51
Model 3, 5.000 traces, 10% noise	16,44	0,05	0,21	15,88	0,05	0,17	13,80
Model 3, 5.000 traces, 25% noise	43,21	0,10	0,21	32,10	0,03	0,13	34,04

Table 6.2: An overview of all the running times needed for the different conformance checking inputs, all times displayed are in seconds.

Chapter 7

Conclusions

In this chapter the main conclusions of this thesis are stated in the section Contributions, but besides the contributions of this work, ideas for future work also came to mind. These ideas are listed in the section Future work, the most elaborate suggestions are explained in their own subsection.

7.1 Contributions

When observing the conclusions and implementations of this research, it can be observed that multiple novel techniques have been proposed in this research. The application of reduction rules itself is not a novelty and it has already been used to optimize conformance checking as well as can be seen in [6]. But also applying the reduction rules to the synchronous product and the 2 phase approach is a novelty.

The next contribution is the reduction DAG. Do to this data structure the rules can be applied in a very efficient manner and this pose an example for future conformance checking optimizations, for example conformance checking of timed petri nets or data petri nets.

The last and main contribution is of course the reduced time required to compute the fitness for the case of large logs. Although the required time is increased when the log is rather small, the main focus of this thesis was to optimize for large problems as the benefit of optimizing small problems negligible. Hence it can be concluded that this main goal was successfully achieved.

7.2 Future work

7.2.1 Merge equivalent traces

It frequently occurs that different traces are equal in fitness, this can be explained by the fact that if 2 transitions are in parallel, their corresponding events can occur in any order in the trace without affecting the fitness. One could define a notion of *swappable events*, which are events that can be exchanged in order without affecting the fitness, this would allow to leverage significantly more of the efficient phase 1 reductions. Following from this definition, it is also possible to define a notion of *fitness equivalent* traces, traces are not necessarily the same, but do return the same fitness. If such a definition is used and applied to the traces, only a single trace of the set of fitness equivalent traces would have to be aligned with the model.

An alternative idea would be to define a *generic trace* that represents a whole class of fitness equivalent traces. This would be a data structure that can be seen as something in between a model and a trace. In a generic trace, one could define an exclusive or-move as $a \vee b$, 2 events

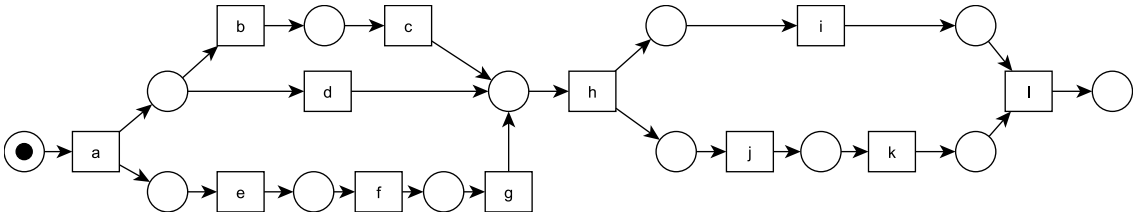

Figure 7.1: A demo model for which a gerenic trace can easily be defined.

that need to be consecutive as a, b and 2 events for which the order does not matter as $\frac{a}{b}$. Take for example the model shown in figure 7.1, the generic trace for all perfectly fitting trace would look like this:

$$a, {}^{d \vee b,c}_{e,f,g}, h, {}^{i}_{j,k}, l$$

It can be beneficial to compare every trace to this generic trace, return a fitness of 1 it it is a match, but determine the fitness the usual way if there was no match. It may also be beneficial to create other generic traces that deviate from this and do not have a fitness of 1 and use those in the computation as well.

7.2.2 Update visualisations

The plug-in returns a data structure called PNRepResult. This data structure was originally created for the input of a single net and a single log. Due to the phase 1 reductions however, this assumption on the input is no longer satisfied as multiple nets and logs are created. As a consequence of this, many of the visualisations to inspect the result yield assertion errors or null pointer exceptions. One solution would be to update these visualisation tools, another solution would be to create a new data structure.

Another interesting subject to add to the final visualisations would be the reduction graph, in the current implementation, there is no possibility to view this graph. An elegant solution to this lacking visualisation would be to show the graph and allow the user to click on every edge and every place to obtain more information on the reduction rule or every reduced model and its assigned traces.

The last suggestion for the visualisation is with regards to the colouring of the alignments. In the current implementation, model moves are shown in purple, log moves in yellow and synchronous moves in green, which is very suitable if all synchronous moves have a cost of 0. However, since this thesis introduced synchronous moves with a cost higher than 0, this representation can be misleading as an alignment consisting of merely synchronous moves no longer necessarily has a fitness of 1. Hence it could be a more informative representation to give synchronous moves with a cost higher than 0 a different colour.

7.2.3 Use a single synchronous product

The original implementation of the synchronous product is rather inconvenient to reduce. Hence, to allow for the phase 2 reduction, a different implementation is used as an intermediate version. This intermediate version, or *reducible synchronous*

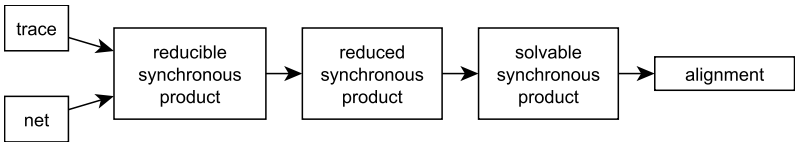

Figure 7.2: A schematic representation of the current process to create an alignment.

product, is then reduced using the phase 2 reductions and transformed into the original implementation to allow for the computation of the alignment. This process is graphically displayed in figure 7.2.

This transformation of a synchronous product into a different synchronous product occurs for every computed alignment, which in practice means frequently. This indicates that it is useful to research if this transformation can be omitted. This optimization could be achieved in two different ways, by applying the reductions on the solvable synchronous product or by directly solving the reduced synchronous product. The first solution would require a change in the implementation of the phase 2 reduction rules, the second solution would require a change in the algorithm used to compute the alignment from the synchronous product.

To support this hypothesis, a brief test was executed with the first model of the Performance Evaluation chapter. In this test, the normal plug-in without any reductions was used, but it did have the reducible synchronous product, which was then transformed into the solvable product rather then computing the solvable product right away. On average, this version needed nearly 20% more time than the original plug-in needed. This proves that this transformation of the product has a significant impact on the total computational time required.

7.2.4 Optimize the reduction DAG

As was shown in the Results and Comparison section, the construction of the reduction DAG can be a time consuming task. Since this DAG is constructed in almost the same way for every petri net model regardless of the log, the only exception being the amount of self-loop transitions, a possible optimisation would be to export this DAG and allow the user to reuse it for the same model with a different log.

During this thesis, it occurred multiple times that a model in the reduction DAG was not used because no traces were mapped to it. This indicates that it might be possible in some cases that the DAG contains unreachable places. It can be worth the effort to research if such places are truly unreachable or if there just did not happen to be a trace corresponding with it. If it turns out that unreachable places do occur, this knowledge can be used to optimize the process of creating the DAG.

A third suggestion is to optimize the process of creating the DAG. In the current implementation, all possible rules are tested against every net without using previous knowledge. The advantage of this approach is that it is guaranteed that all possible reductions are found, but the disadvantage is that this approach is inefficient. A more intelligent approach for this would be to take the last reduction into account and test for new rules in a more specific manner using this knowledge.

7.2.5 Other suggestions

Besides the suggestions mentioned, there are some smaller suggestions as well. The first one being to look at duplicate labels during the phase 1 reductions, for example the net shown in figure 7.3 can be reduced using phase 1 reductions, which is currently not done due to the fact that both transitions have the same label.

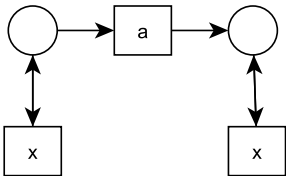

Figure 7.3: An example of a net with 2 self-loops that have the same label.

Another suggestion would be to do research on the order in which rules are applied. In the current implementation, a trace goes through the reduction rules ordered by the way they were discovered and the first applicable rule is used. This indicates that in some cases it might

be possible that the DAG is not optimal in every case, further research is needed to provide certainty on this.

The final suggestion is to look at further reductions. One way to do so is to apply phase 1 reductions again after phase 2 is finished. For example in the case of self-loop transitions, it can occur that this can be applied and reduce the problem further. Another suggestion would be to alter the phase 2 precondition that rules cannot be applied subsequently due to an explosion in the number of transitions. One could argue that it would be more optimal to allow for such an explosion or perhaps allow the stacking of rules up to a higher level than currently is the case as this could result in a shorter optimal solution of the synchronous product.

Bibliography

[1] Arya Adriansyah, Eindhoven Technical University. PNetReplayer. https://svn.win.tue.nl/repos/prom/Packages/PNetReplayer/, 2017. 3

[2] Boudewijn van Dongen. BPI Challenge 2012. https://data.4tu.nl/repository/uuid:3926db30-f712-4394-aebc-75976070e91f, 2012.

[3] Boudewijn van Dongen. BPI Challenge 2017. https://data.4tu.nl/repository/uuid:7e326e7e-8b93-4701-8860-71213edf0fbe, 2017.

[4] Daniele Loerti, Federico Chesani, Anna Ciampolini, Paola Mello. Distributed Compliance Monitoring of Business Processes over MapReduce Architectures, 2017. 8

[5] F. Mannhardt, M. de Leoni, H.A. Reijers and W.M.P. van der Aalst. Balanced multi-perspective checking of process conformance, 2016. 3

[6] Farbod Taymouri, Josep Carmona. Model and Event Log Reductions to Boost the Computation of Alignments, 2014. 8, 33

[7] Farbod Taymouri, Josep Carmona. An Evolutionary Technique to Approximate Multiple Optimal Alignments, 2018. 8

[8] G. Berthelot and Lri-Iie. Checking Properties Of Nets Using Transformations, 2005. 8, 9

[9] Jorge Munoz-Gama, Josep Carmona, Wil M.P.van der Aalst. Single-Entry Single-Exit decomposed conformance checking, 2014. 8

[10] Josep Carmona, Boudewijn van Dongen, Andreas Solti and Matthias Weidlich. *Conformance Checking*. 2018. vii, 2, 5, 6

[11] J. C. Massimiliano de Leoni, Jorge Munoz-Gama and W. M. van der Aalst. Decomposing Alignment-based Conformance Checking of Data-aware Process Models, 2014. 8

[12] Process Mining Group, Eindhoven Technical University. ProM tools. http://www.promtools.org/doku.php, 2018. 3

[13] Robert H. Sloan, Ugo Buy. Reduction Rules for Time Petri Nets, 2014. 8

[14] S. J. van Zelfst, B.F. van Dongen, L.M.A. Tonnaer. Alpha Miner. https://www.futurelearn.com/courses/process-mining/0/steps/15637, 2003.

[15] Wil M. P. van der Aalst. The Application of Petri Nets to Workflow Management, 1998. 6

[16] Wil M. P. van der Aalst. Distributed Process Discovery and Conformance Checking, 2012. 8

[17] Wil M. P. van der Aalst. Decomposing Petri nets for process mining: A generic approach, 2013. 8

[18] Wil M.P. van der Aalst. A General Divide and Conquer Approach for Process Mining, 2013. 8

[19] Wil van der Aalst. *Process Mining*. 2011. 1

[20] Wilfried Brauer, Wolfgang Reisig. Fundamentals on the description of discrete processes, 1966. 5